NoSQL no desenvolvimento de aplicações Web colaborativas

Bernadette Farias Lóscio (bfl@cin.ufpe.br), Hélio Rodrigues de Oliveira (hro@cin.ufpe.br), Jonas César de Sousa Pontes (jcsp@cin.ufpe.br)

Resumo

A Web tem revolucionado a forma como criamos conteúdo e trocamos informações, além de habilitar o desenvolvimento de uma série de novas aplicações. O grande volume de dados gerado por aplicações Web, juntamente com os requisitos diferenciados destas aplicações, como a escalabilidade sob demanda e o elevado grau de disponibilidade, têm contribuído para o surgimento de novos paradigmas e tecnologias. Neste contexto uma nova categoria de Banco de Dados, chamada NoSQL (Not Only SQL), foi proposta com o objetivo de atender aos requisitos de gerenciamento de grandes volumes de dados, semi-estruturados ou não estruturados. Este minicurso apresenta os fundamentos da tecnologia de banco de dados NoSQL, enfatizando as suas principais características e diferenciais, bem como suas áreas de aplicação.

1 Introdução

O grande volume de dados gerado por aplicações Web, juntamente com os requisitos diferenciados destas aplicações, como a escalabilidade sob demanda e o elevado grau de disponibilidade, têm contribuído para o surgimento de novos paradigmas e tecnologias. As redes sociais, por exemplo, requerem o gerenciamento de grandes quantidades de dados não estruturados, os quais são gerados diariamente por milhões de usuários em busca do compartilhamento de informações, conhecimentos e interesses.

Neste contexto, surgiu uma nova categoria de Banco de Dados, chamada NoSQL (*Not Only SQL*), que foi proposta com o objetivo de atender aos requisitos de gerenciamento de grandes volumes de dados, semi-estruturados ou não estruturados, que necessitam de alta disponibilidade e escalabilidade. A necessidade de uma nova tecnologia de BD surgiu como consequência da ineficiência dos bancos de dados relacionais em lidar com esta tarefa. Vale ressaltar que a tecnologia relacional foi proposta na década de 70, quando as aplicações de banco de dados caracterizavam-se por lidar com dados estruturados, ou seja, dados que possuem uma estrutura fixa e bem definida. Além disso, por maior que fossem os volumes de dados gerados por tais aplicações, ainda assim não se comparam ao volume de dados gerado pelas redes sociais, por exemplo, que também possuem um universo de usuários que cresce exponencialmente, o que não acontece nas aplicações convencionais de banco de dados.

Os bancos de dados NoSQL tem sido amplamente adotados em empresas como Facebook, Amazon e Google com o intuito de atender às suas demandas de escalabilidade, alta disponibilidade e dados não estruturados. Além disso, atualmente,

diversos bancos de dados NoSQL de código livre estão disponíveis, como: Cassandra[1], Hypertable[2], MongoDB[3], Redis[4], CouchDB[5] e Dynamo[6].

Tendo em vista o crescente interesse na adoção desta tecnologia, bem como os novos desafios gerados pelo uso do NoSQL, torna-se fundamental o conhecimento de seus principais conceitos e sua utilização. Em particular, estes conhecimentos são de grande relevância para aqueles que têm interesse em aplicações Web colaborativas, as quais, na maioria das vezes, necessitam de uma tecnologia que ofereça suporte ao gerenciamento e escalabilidade de grandes volumes de dados, de maneira simples e eficiente.

Neste minicurso são apresentados os fundamentos da tecnologia de banco de dados NoSQL, enfatizando as suas principais características e diferenciais, bem como suas áreas de aplicação.

1.1 Histórico

Os primeiros Sistemas Gerenciadores de Banco de Dados (SGBD) surgiram por volta de 1960 e foram desenvolvidos com base nos primitivos sistemas de arquivos. Dentre as principais características de um SGBD, destacamos: controle de concorrência, segurança, recuperação de falhas, gerenciamento do mecanismos de armazenamento de dados e controle das restrições de integridade do BD. Outra importante função de um SGBD é o gerenciamento de transações. Uma transação pode ser definida como uma coleção de operações que desempenha uma função lógica dentro de uma aplicação do sistema de banco de dados, em outras palavras um transação representa um conjunto de operações de leitura ou escrita que são realizadas no banco de dados. A execução de transações em um SGBD deve obedecer a algumas propriedades a fim de garantir o correto funcionamento do sistema e a respectiva consistência dos dados. Estas propriedades são chamadas de propriedade ACID e são definidas a seguir:

- **A**tomicidade: todas as operações da transação são executadas, ou seja a transação é executada por completo, ou nada é executado;
- **C**onsistência: após uma transação ter sido concluída, o banco de dados deve permanecer em um estado consistente, ou seja, deve satisfazer as condições de consistência e restrições de integridade previamente assumidas;
- **I**solamento: se duas transações estão sendo executadas concorrentemente seus efeitos devem ser isolados uma da outra. Esta propriedade está relacionada ao controle de concorrência do SGBD;
- **D**urabilidade: uma vez que uma transação ocorreu com sucesso, seu efeito não poderá mais ser desfeito, mesmo em caso de falha. Esta propriedade está relacionada a capacidade de recuperação de falhas do SGBD

Diferentes modelos de dados foram propostos desde o surgimento dos SGBDs, os quais diferenciam-se pelos conceitos adotados para representação dos dados do

1 http://incubator.apache.org/cassandra/
2 http://hypertable.org/
3 http://www.mongodb.org/
4 http://redis.io
5 http://couchdb.apache.org/
6 http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html

mundo real. Inicialmente, foram propostos os modelos hierárquico e de rede. No modelo hierárquico os dados são representados como registros, organizados em árvores e relacionados através de associações do tipo pai-filho. O modelo em rede foi proposto como uma extensão ao modelo hierárquico, onde não existe o conceito de hierarquia e um mesmo registro pode estar envolvido em várias associações.

No início dos anos 70, surgiram os bancos de dados relacionais, os quais se firmaram como solução comercial para armazenamento e gerenciamento de dados convencionais, ou seja, dados que possuem uma estrutura fixa, bem definida e com tipos de dados simples, como os dados gerados e manipulados por aplicações convencionais de bancos de dados (ex: sistemas de controle de estoque e folha de pagamento). Os conceitos básicos do modelo relacional são: relação (tabela), atributo (coluna) e tupla (linha). Especificamente, uma tabela representa um conceito (ex: Pessoa, Veículo) ou uma associação entre conceitos (ex: Pessoa possui Veículo), as colunas definem as propriedades destes conceitos (ex: nome, endereço e fone, no caso de uma pessoa) e as linhas representam as instâncias propriamente ditas(ex: João, Rua A, 2321.2243). Dentre os motivos do grande sucesso do modelo relacional destacam-se a padronização de conceitos, sua base formal e a facilidade de uso da linguagem SQL (*Structured Query Language*), linguagem padrão para consultas e manipulação de dados relacionais.

De uma maneira geral, a simplicidade do modelo relacional contribuiu para sua grande disseminação e adoção. Porém, a medida em que as aplicações de bancos de dados foram evoluindo surgiu a necessidade de manipulação de outros formatos de dados, como imagem, som e vídeo, bem como de tipos de dados complexos. A fim de atender aos requisitos destas aplicações de bancos de dados não convencionais, novas soluções foram propostas, como os bancos de dados orientados a objetos (BDOO) e os bancos de dados objeto-relacionais (BDOR).

Posteriormente, com o surgimento da Web, outras aplicações de banco de dados começaram a ser desenvolvidas, originando, dessa forma, novos requisitos de bancos de dados. Dentre estes requisitos destaca-se a necessidade de manipulação de grandes volumes de dados não estruturados ou semi-estruturados, bem como novas necessidades de disponibilidade e escalabilidade. Assim, a fim de atender aos requisitos destas aplicações, novas soluções para gerenciamento de dados começaram a ser propostas, como, por exemplo, os bancos de dados NoSQL.

Inicialmente, as propostas de bancos de dados não relacionais foram desenvolvidas por pequenas empresas e por comunidades de software livre. Tais soluções foram então agrupadas em um termo, NoSQL (*Not Only SQL*), que significa "não apenas SQL". Este termo faz referência a SGBDs que não adotam o modelo relacional e são mais flexíveis quanto às propriedades ACID. Esta flexibilidade torna-se necessária devido aos requisitos de alta escalabilidade necessários para gerenciar grandes quantidades de dados, bem como para garantir a alta disponibilidade dos mesmos, característica fundamental para as aplicações da Web 2.0.

1.2 Principais características dos bancos de dados NoSQL

Os bancos de dados NoSQL apresentam algumas características fundamentais que os diferenciam dos tradicionais sistemas de bancos de dados relacionais, tornando-os

adequados para armazenamento de grandes volumes de dados não estruturados ou semi-estruturados. A seguir, descrevemos alguma destas características.

- **Escalabilidade horizontal**: a medida em que o volume de dados cresce, aumenta a necessidade de escalabilidade e melhoria de desempenho. Dentre as soluções para este problema, temos a escalabilidade vertical, que consiste em aumentar o poder de processamento e armazenamento das máquinas, e a escalabilidade horizontal, onde ocorre um aumento no número de máquinas disponíveis para o armazenamento e processamento de dados. A escalabilidade horizontal tende a ser uma solução mais viável, porém requer que diversas *threads*/processos de uma tarefa sejam criadas e distribuídas. Neste caso, o uso de um banco de dados relacional poderia ser inviável, uma vez que diversos processos conectando simultaneamente um mesmo conjunto de dados causaria uma alta concorrência, aumentando, consequentemente, o tempo de acesso às tabelas envolvidas. A ausência de bloqueios é uma característica fundamental dos bancos de dados NoSQL, permitindo a escalabilidade horizontal e tornando esta tecnologia adequada para solucionar os problemas de gerenciamento de volumes de dados que crescem exponencialmente, como os dados da Web 2.0. Uma alternativa muito conhecida para alcançar escalabilidade horizontal é o Sharding, que consiste em dividir os dados em múltiplas tabelas a serem armazenadas ao longo de diversos nós de uma rede. A aplicação desta técnica traz o grande problema de "quebrar" a lógica de relacionamentos, o que é o grande forte dos bancos relacionais. Neste caso, as aplicações têm que resolver a complexidade gerada pela partição de informações como, por exemplo, a execução de joins e outros comandos. Fazer sharding, em um contexto de bacos de dados relacionais, de forma manual não é uma tarefa simples e exige considerável esforço da equipe de desenvolvimento.
- **Ausência de esquema ou esquema flexível**: uma característica evidente dos bancos de dados NoSQL é a ausência completa ou quase total do esquema que define a estrutura dos dados modelados. Esta ausência de esquema facilita tanto a escalabilidade quanto contribui para um maior aumento da disponibilidade. Em contrapartida, não há garantias da integridade dos dados, o que ocorre nos bancos relacionais, devido à sua estrutura rígida. Como veremos na Seção 2, estas estruturas são, em sua maioria, baseadas em um conceito de chave-valor, permitindo uma alta flexibilidade na forma como os dados são organizados.
- **Suporte nativo a replicação**: outra forma de prover escalabilidade é através da replicação. Permitir a replicação de forma nativa, diminui o tempo gasto para recuperar informações. Existem duas abordagens principais para replicação:
    - *Master-Slave* (Mestre-Escravo): cada escrita no banco resulta em N escritas no total, onde N é o número de nós escravos. Nesta arquitetura a escrita é feita no nó mestre, sendo a escrita refeita em cada nó escravo pelo nó mestre. A leitura torna-se mais rápida, porém a capacidade de escrita torna-se um gargalo nesta abordagem. Geralmente não é recomendada quando tem-se um grande volume de dados.
    - *Multi-Master*: admitimos que temos, não apenas um, mas vários nós mestres, de forma que é possível diminuir o gargalo gerado pela escrita que ocorre na abordagem mestre-escravo. Porém, a existência de diversos nós mestres pode causar um problema de conflito de dados.

- **API simples para acesso aos dados**: o objetivo da solução NoSQL é prover uma forma eficiente de acesso aos dados, oferecendo alta disponibilidade e escalabilidade, ou seja, o foco não está em como os dados são armazenados e sim como poderemos recuperá-los de forma eficiente. Para isto, é necessário que APIs sejam desenvolvidas para facilitar o acesso a estas informações, permitindo que qualquer aplicação possa utilizar os dados do banco de forma rápida e eficiente.
- **Consistência eventual**: é uma característica de bancos NoSQL relacionada ao fato da consistência nem sempre ser mantida entre os diversos pontos de distribuição de dados. Esta característica tem como princípio o teorema CAP (*Consistency*, *Availability* e *Partition tolerence*), que diz que, em um dado momento, só é possível garantir duas de três propriedades entre consistência, disponibilidade e tolerância à partição. No contexto da Web, por exemplo, geralmente são privilegiadas a disponibilidade e a tolerância à partição. Como consequência, as propriedades ACID não podem ser obedecidas simultaneamente. Em vez disso, tem-se outro conjunto de projetos denominado BASE (Basicamente disponível, Estado leve e Consistente em momento indeterminado). Neste caso, o sistema deve ser planejado para tolerar inconsistências temporárias a fim de poder priorizar disponibilidade.

Algumas técnicas são importantes para a implementação das funcionalidades do NoSQL, incluindo:

- O *map/reduce* dá suporte ao manuseio de grandes volumes de dados distribuídos ao longo dos nós de uma rede. Na fase de *map*, os problemas são quebrados em subproblemas que são distribuídos em outros nós na rede. E na fase *reduce*, os subproblemas são resolvidos em cada nó filho e o resultado é repassado ao pai, que, sendo ele também filho, repassaria ao seu pai, e assim por diante até chegar ao nó raiz do problema.
- O *consistent hashing* suporta mecanismos de armazenamento e recuperação em banco de dados distribuídos, onde a quantidade de sites está em constante modificação. O uso desta técnica é interessante porque evita um grande número de migração de dados entre esses sites, os quais podem ser alocados e desalocados para a distribuição dos dados.
- O *multiversion concurrency control* (MVCC) é um mecanismo que dá suporte a transações paralelas em um banco de dados. Ao contrário do esquema clássico de gerenciamento de transações, como não faz uso de *locks*, permite que operações de leitura e escrita sejam feitas simultaneamente.
- *Vector clocks* são usados para gerar uma ordenação dos eventos acontecidos em um sistema. Pela possibilidade de várias operações estarem acontecendo ao mesmo tempo sobre o mesmo item de dado distribuído, o uso de um log de operações com as suas respectivas datas é importante para determinar qual a versão de um determinado dado é a mais atual.

2 Modelos de Dados NoSQL

Nesta seção apresentamos os principais tipos de modelos de dados NoSQL, enfatizando suas diferenças, bem como vantagens e desvantagens. Os modelos apresentados são: chave-valor, orientado a documentos, orientado a colunas e orientado a grafos.

2.1 Chave-valor (*key-value*)

Este modelo é considerado bastante simples e permite a visualização do banco de dados como uma grande tabela *hash*. De maneira bem simples, o banco de dados é composto por um conjunto de chaves, as quais estão associadas um único valor, que pode ser uma string ou um binário. A Figura 2.1. apresenta um exemplo de um banco de dados que armazena informações pessoais no formato de chave-valor. A chave representa um campo como nome e idade, enquanto que o valor representa a instância para o campo correspondente.

Este modelo, por ser de fácil implementação, permite que os dados sejam rapidamente acessados pela chave, principalmente em sistemas que possuem alta escalabilidade, contribuindo também para aumentar a disponibilidade de acesso aos dados. As operações disponíveis para manipulação de dados são bem simples, como o *get( )* e o *set( )*, que permitem retornar e capturar valores, respectivamente. A desvantagem deste modelo é que não permite a recuperação de objetos por meio de consultas mais complexas.

Nome	Hélio Rodrigues
Idade	45
Sexo	Masculino
Fone	99 99999999

Figura 2.1. Exemplo modelo chave-valor

Como exemplo de bancos de dados NoSQL que adota este modelo destacamos o Dynamo, o qual foi desenvolvido pela Amazon e, posteriormente, foi utilizado como base para o desenvolvimento do banco de dados Cassandra. Dentre as principais características do Dynamo temos a possibilidade de realizar particionamento, replicação e versionamento dos dados. Além do Dynamo, outras soluções NoSQL que seguem o conceito de chave-valor são: Redis[7], Riak[8] e o GenieDB[9].

2.2 Orientado a Colunas

Este modelo é um pouco mais complexo que o modelo chave-valor e, neste caso, muda-se o paradigma de orientação a registros ou tuplas (como no modelo relacional) para orientação a atributos ou colunas (modelo NoSQL). Neste modelo os dados são indexados por uma tripla (linha, coluna e timestamp), onde linhas e colunas são identificadas por chaves e o timestamp permite diferenciar múltiplas versões de um mesmo dado. Vale ressaltar que operações de leitura e escrita são atômicas, ou seja, todos os valores associados a uma linha são considerados na execução destas operações, independentemente das colunas que estão sendo lidas ou escritas. Outro conceito associado ao modelo é o de família de colunas (*column family*), que é usado com o intuito de agrupar colunas que armazenam o mesmo tipo de dados. Observe o exemplo da Figura 2.2. que modela o conceito de amigos, onde primeiroNome e sobrenome são colunas pertencentes à família de colunas denominada nome. De maneira semelhante, as colunas endereço, cidade e estado pertencem à família local. Observe que para a linha 001, o amigo Hélio tem diferentes endereços, onde para cada um deles tem um timestamp correspondente. Como já foi dito, o acesso à linha é atômico, ou seja, mesmo

[7] http://try.redis-db.com/
[8] http://riak.basho.com
[9] http://www.geniedb.com

que tenhamos interesse apenas na coluna sobrenome da linha 001, todas as colunas são retornadas quando esta linha for consultada.

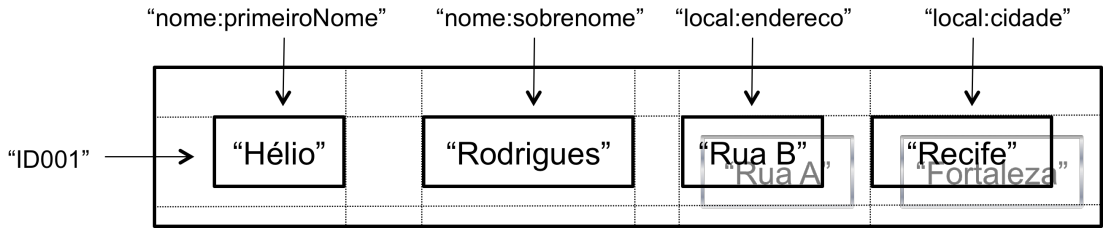

Figura 1.2. Exemplo de modelo baseado em colunas

Este modelo de dados surgiu com o BigTable[10] da Google, por isso é comum falar sobre o modelo de dados BigTable. Dentre as características deste modelo temos que permite particionamento dos dados, além de oferecer forte consistência, mas não garante alta disponibilidade. Outras soluções surgiram após o Bigtable, dentre elas o Cassandra[11], desenvolvido pelo Facebook e baseado no Dynamo. Hbase[12] é um banco de dados open-source semelhante ao BigTable, que utiliza o Hadoop[13], que é uma infraestrutura para processamento distribuído de dados em grande escala.

2.3 Orientado a Documentos

Como o próprio nome diz, este modelo armazena coleções de documentos. Um documento, em geral, é um objeto com um identificador único e um conjunto de campos, que podem ser strings, listas ou documentos aninhados. Estes campos se assemelham a estrutura chave-valor já mencionado anteriormente. No modelo chave-valor, apenas uma única tabela *hash* é criada para todo o banco. No modelo orientado a documentos temos um conjunto de documentos e em cada documento temos um conjunto de campos (chaves) e o valor deste campo. Outra característica importante é que este modelo não depende de um esquema rígido, ou seja, não exige uma estrutura fixa como ocorre nos bancos relacionais. Assim, é possível que ocorra uma atualização na estrutura do documento, com a adição de novos campos, por exemplo, sem causar problemas ao banco de dados. Na Figura 2.3. temos um exemplo de documento representando um post de um blog, onde os campos foram definidos por: Assunto, Autor, Data, Tags e Mensagem. Para cada um destes campos, temos os seus valores associados. Não há restrições em atualizar o documento de forma que agora, exista um novo campo chamado, comentários. Esta flexibilidade é uma das grandes vantagens deste modelo.

[10] http://labs.google.com/papers/bigtable.html
[11] http://incubator.apache.org/cassandra/
[12] http://hbase.apache.org/
[13] http://hadoop.apache.org/

ID: P001
Assunto: “Eu gosto de laranjas”
Autor: “Hélio”
Data: “27/01/2011”
Tags: [“laranjas”, “suco”, “plantas”]
Mensagem: “Hoje estou com vontade de tomar suco de laranja!”

Figura 2.3. Exemplo de Documento

Como principais soluções que adotam o modelo orientado a documentos destacamos o CouchDB e o MongoDB. CouchDB utiliza o formato JSON e é implementado em Java, além disso permite replicação e consistência. O MongoDB foi implementado em C++ e permite tanto concorrência como replicação.

2.4 Orientado a Grafos

O modelo orientado a grafos possui três componentes básicos: os nós (são os vértices do grafo), os relacionamentos (são as arestas) e as propriedades (ou atributos) dos nós e relacionamentos. Neste caso, o banco de dados pode ser visto como um multigrafo rotulado e direcionado, onde cada par de nós pode ser conectado por mais de uma aresta. A Figura 4 apresenta um exemplo de grafo que representa os dados de uma aplicação que deseja manter informações relativas a locais de viagens e locais onde pessoas residem. Uma consulta relevante para esta aplicação seria: “Quais cidades foram visitadas anteriormente (seja residindo ou viajando) por pessoas que viajaram para o Rio de Janeiro?”. No modelo relacional esta consulta poderia ser muito complexa devido a necessidade de múltiplas junções, o que poderia acarretar uma diminuição no desempenho da aplicação. Porém, por meio dos relacionamentos inerentes aos grafos, estas consultas tornam-se mais simples e diretas.

Ainda no exemplo da Figura 2.4. temos três pessoas: Berna, Hélio e Jonas, que são nós do grafo e estão conectados à cidades que ou já residiram ou visitaram. Por exemplo, Hélio morou em Fortaleza e Recife, e visitou Paraty e Rio de Janeiro. Para responder a pergunta acima, buscamos por todos os nós das pessoas que visitaram Rio de Janeiro, no caso Berna e Hélio. Agora observamos quais cidades foram visitadas ou resididas por Berna e Hélio, que no caso são: Fortaleza, Rio de Janeiro, Paraty, São Paulo, Recife e João Pessoa.

A vantagem de utilização do modelo baseado em grafos fica bastante clara quando consultas complexas são exigidas pelo usuário. Comparado ao modelo relacional, que para estas situações pode ser muito custoso, o modelo orientado a grafos tem um ganho de performance, permitindo um melhor desempenho das aplicações.

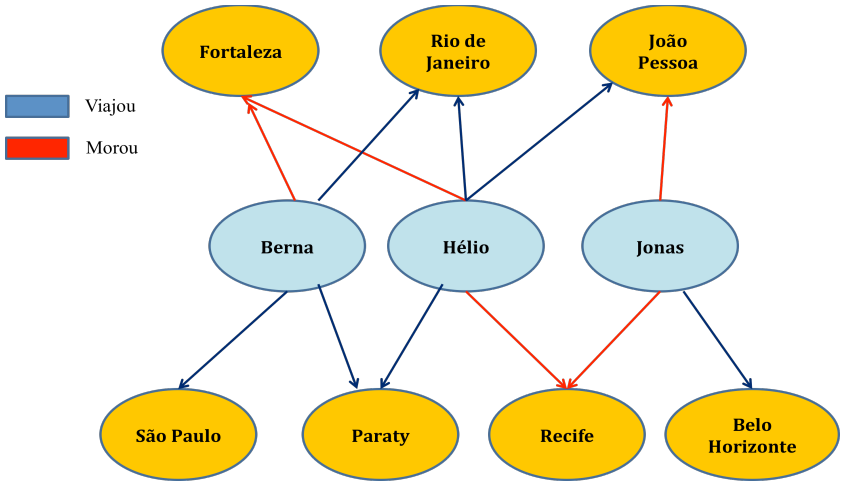

Figura 2.2. Exemplo estrutura baseada em grafos

Exemplos de bancos de dados baseados em grafos são: Neo4j[14], AllegroGraph[15] e Virtuoso[16]. O Neo4j é *open-source*, implementado em Java e utiliza um modelo de grafos nativo. O AllegroGraph e o Virtuoso são baseados no modelo RDF.

É importante deixar claro que um modelo não deve ser considerado melhor que o outro, uma vez que cada tipo de modelo pode ser mais adequado para determinadas aplicações. Por exemplo, para manipulação de dados estatísticos, frequentemente escritos mas raramente lidos (como um contador de *hits* na Web), pode ser usado um banco de dados orientado a chave/valor, como o Redis, ou um banco de dados orientado a documentos, como o MongoDB. Aplicativos de alta disponibilidade, onde a minimização da inatividade é fundamental, podem utilizar um banco de dados orientado a colunas, como o Cassandra. Aplicações que exigem alto desempenho em consultas com muitas junções, podem adotar um banco orientado a grafos. A importância de conhecer cada solução e adotar aquela que for mais adequada poderá contribuir para a diminuição do custo de criação do banco de dados, bem como para o aumento da eficiência no processamento dos dados.

3 Casos de Sucesso

A tecnologia NoSQL tem se tornado cada vez mais evidente nos últimos anos. Tal evidência é comprovada pela forte adoção de soluções NoSQL nas grandes empresas, para o gerenciamento dos seus dados, como veremos a seguir.

3.1 Twitter

O Twitter é uma rede social e servidor para microblogging, que permite aos usuários enviar e receber atualizações pessoais de outros contatos, por meio do site do serviço, por SMS, bem como por softwares específicos de gerenciamento.

Com o crescimento exponencial do uso do Twitter, resolver o problema de acesso a grandes volumes de dados em tempo-real tornou-se um grande desafio. Em Fevereiro de 2010, o número de tweets era 1.2 bilhões por mês. A preocupação com o

[14] http://www.neo4j.org/
[15] http://www.franz.com/agraph/
[16] http://www.openlinksw.com/

problema de disponibilidade fez com que a empresa substituisse o banco de dados MySQL pelo Cassandra, uma solução NoSQL. A empresa utiliza o Cassandra para armazenar resultados de *data mining* realizados sobre a base dos usuários, resultados de *trend topics*, *@toptweets* e análises em tempo real em larga escala. A utilização do Cassandra trouxe vantagens tanto na implementação da modelagem dos dados relacionados aos *tweets*, *timeline,* entre outros, como no desempenho com relação às buscas por usuários ou palavras-chaves. Além disso, aumentou a disponibilidade dos seus serviços. Em 2010, a empresa Pingdom informou que o Twitter esteve no ar por 99,72% do tempo (downtime de 23 horas e 45 minutos). Em 2008, o site esteve fora do ar por 84 horas.

3.2 Facebook

Após seis anos de sua criação, o facebook possui, atualmente, cerca de 3,5 bilhões de conteúdos (*links*, *posts*, etc) compartilhados por semana. Para evitar problemas com a escalabilidade e disponibilidade dos dados, a empresa desenvolveu o Cassandra, uma solução NoSQL. Inicialmente criado para otimização do sistema de busca do facebook, atualmente o Cassandra é utilizado para dar suporte à replicação, detecção de falhas, armazenamento em cache dentre outras funcionalidades., Posteriormente, o Cassandra tornou-se um projeto da Apache Incubator, vindo a ser utilizado por outras empresas como Cisco, Digg e Twitter.

3.3 Google

A Google também desenvolveu sua própria solução NoSQL, chamada de BigTable, que é um sistema de armazenamento distribuído para gerenciar dados estruturados em larga escala. Mais de 60 produtos (como Gmail, Google Docs, Google Analytics, Orkut, Personalized Search, Google Earth etc) utilizam o BigTable. Esta solução é utilizada em conjunto com outros pacotes Google como o GFS (Google File System) para gerenciamento de informações e o map/reduce para distribuição dos dados. Esta solução permite a escalabilidade de recursos, bem como alta performance no processamento das consultas, dos processos e dos serviços.

3.4 Amazon

Um dos grandes desafios enfrentados pela Amazon.com diz respeito a confiabilidade do grande volume de dados gerenciado por suas aplicações, não apenas por questões financeiras e devido aos gastos com soluções convencionais, mas também por causa do impacto da confiança de seus cliente em seus produtos. Em 2007, com o intuito de garantir alta disponibilidade dos dados de seus serviços "*always-on*", a Amazon desenvolveu uma solução NoSQL, o Dynamo. Como resultado da adoção desta nova tecnologia, diversos serviços da Amazon tem se mantido disponíveis em 99,9995% das requisições realizadas.

3.5 LinkedIn

O LinkedIn é uma rede de negócios, fundada em 2002, que tem como foco principal o estabelecimento de relações entre profissionais. Em março de 2011, atingiu a marca de 100 milhões de usuários. Como em outras redes de relacionamentos, o desempenho no processamento das consultas foi afetado com o crescimento na quantidade de dados. Para suprir a demanda das aplicações do Linkedin, diversas soluções relacionais foram utilizadas, mas com pouco sucesso. Como consequência, a empresa desenvolveu sua

própria solução NoSQL, chamada de Voldemort[17] que tem trazido ótimos resultados de desempenho. Voldemort suporta escalabilidade horizontal, replicação, particionamento, transparência a falhas dentre outras funcionalidades.

4 Estudo de caso

Para ilustrar os conceitos discutidos, nesta seção apresentamos um exemplo de criação de uma aplicação Web utilizando a linguagem de programação PHP e o banco de dados MongoDB. Para um bom entendimento do exemplo são necessários apenas alguns conhecimentos básicos de banco de dados e programação.

Como visto anteriormente, o MongoDB é um banco de dados orientado a documentos, sendo possível utilizá-lo em diferentes sistemas operacionais, como Windows, Linux, OS X e Solaris. O MongoDB possui drivers para diversas linguagens de programação, entre elas: C, C#, C++, Java, Perl, PHP, Python e Ruby.

O modelo de dados do MongoDB é bastante simples de compreender e pode ser descrito como se segue:

- Um **banco de dados** armazena um conjunto de coleções;
- Uma **coleção** armazena um conjunto de documentos;
- Um **documento** é um conjunto de campos;
- Um **campo** é um par chave-valor;
- Uma **chave** é um nome (string);
- Um **valor** é um(a):
    - caracter, inteiro, ponto flutuante, timestamp ou binário;
    - um documento;
    - um "array" de valores;

Para implementação do exemplo proposto, utilizamos a linguagem PHP, que é uma linguagem interpretada e de código aberto. Também fazemos uso do WampServer[18], um projeto *open source*, gratuito e distribuído através da licença GPL, que fornece um ambiente integrado de desenvolvimento Web para Windows, onde estão incluídos o servidor Web Apache e o interpretador PHP.

4.1 Descrição da aplicação e configuração do ambiente

Como estudo de caso propomos o desenvolvimento de uma aplicação onde usuários podem postar mensagens de texto simples, que podem ser visualizadas pelo próprio usuário ou por seus amigos. Esta aplicação permitirá que:

- Um usuário crie uma conta de usuário, informando seu email, nome e definindo uma senha;
- Um usuário registrado possa gravar mensagens de texto em seu mural;
- Um usuário registrado possa adicionar amigos ao seu perfil e também possa ser adicionado por outros usuários;

[17] http://project-voldemort.com
[18] http://www.wampserver.com/

- Todas as mensagens de um usuário sejam mostradas nos murais de seus amigos e vice-versa;

Para a implementação da aplicação é necessário configurar o MongoDB e o PHP, como descrito a seguir:

1. Fazer o download da versão adequada em: http://www.mongodb.org/downloads;
2. Seguir os procedimentos de configuração, de acordo com o sistema operacional, descritos em: http://www.mongodb.org/display/DOCS/Quickstart;
3. Fazer o download do instalador do WampServer em http://www.wampserver.com/;
4. Instalar o WampServer (instalação padrão de aplicativos Windows);
5. Habilitar o driver do MongoDB no arquivo de inicialização do PHP (php.ini) o mesmo encontra-se em: http://www.php.net/manual/en/mongo.installation.php#mongo.installation.windows

4.2 Modelando a aplicação

Nesta seção apresentamos a modelagem conceitual do banco de dados para a aplicação proposta. Na Figura 4.1., é apresentado o diagrama ER que descreve os requisitos de dados da aplicação, onde cada usuário possui um id que o identifica unicamente, um nome, email e senha; cada mensagem possui um id, time (data em que a mensagem foi criada) e texto (a mensagem propriamente dita); e cada usuário possui um ou mais amigos e pode postar várias Mensagens.

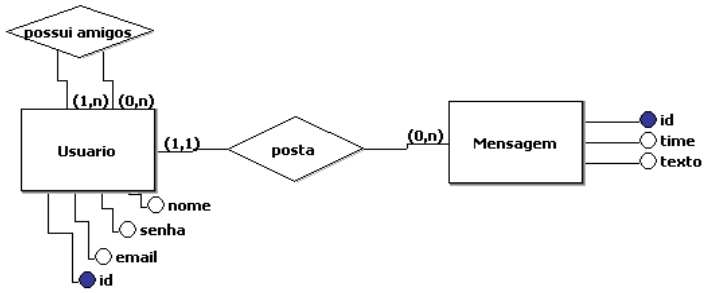

Figura 4.1. Modelo conceitual do banco de dados

A partir da modelagem conceitual, construímos o modelo lógico do banco de dados relacional conforme apresentado na Figura 4.2.

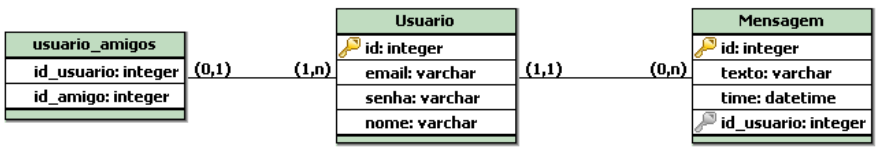

Figura 4.2. Modelo lógico do banco de dados

A partir deste modelo lógico podemos criar uma base de dados relacional. Porém, no paradigma NoSQL não utilizamos nenhum destes artefatos produzidos, pois, conforme vimos anteriormente, o modelo NoSQL é livre de esquema, não exigindo uma estrutura prévia para que possa ser utilizada pela aplicação. Neste caso, apenas é necessário conectar-se ao banco de dados, inserir, recuperar ou alterar os dados.

4.3 Realizando o mapeamento de Relacional para NoSQL

Como citado anteriormente, não definiremos um esquema prévio para a base de dados NoSQL. Porém, para uma melhor compreensão da estrutura, utilizamos a representação de arrays do PHP para mostrar como o banco de dados MongoDB irá armazenar as informações, conforme a Figura 4.3.

```php
<?php
$usuario = array(
    'id' => '01',
    'nome' => 'Jonas',
    'email' => 'jonascesar@gmail.com',
    'senha' => '123456',
    'mesagens' => array(
        array('id' => '01', 'texto' => 'mensagem 01'),
        array('id' => '02', 'texto' => 'mensagem 02')
     )
);
?>
```

Figura 4.3. Estrutura de dados em forma de arrays PHP

Observe que a variável PHP $usuario, definida acima, possui os atributos de usuário do sistema: identificador, nome, email, senha e mensagens. É importante notar que também que o atributo mensagens é construído como um array de arrays que contém as mensagens do usuário.

Os conceitos de tabelas e de relacionamentos através de chaves estrangeiras não são utilizados. A partir de agora, apenas inserimos uma coleção de mensagens no objeto usuário sem nos preocuparmos com a definição de tabelas e de seus relacionamentos.

Devido à falta de estrutura definida, não temos os conceitos de cardinalidade e participação. Podemos a qualquer instante modificar um documento sem precisar respeitar uma estrutura previamente definida, transferindo qualquer carga de validação de tipo de dado ou estrutura para a aplicação.

Podemos observar que a estrutura exibida fica bem mais próxima da realidade, ou seja, um usuário possui uma coleção de mensagens em sua estrutura, não sendo necessário realizar uma junção para buscar as mensagens de um usuário. Isto nos permite um ganho de desempenho quando tratamos de aplicações mais complexas e com grandes volumes de dados e usuários, como é o caso das aplicações colaborativas.

4.4 Implementando a aplicação

Inicialmente criamos as classes de domínio no PHP, que representam as informações de usuário e mensagens, neste caso, temos a classe Usuario.php e Mensagem.php. A seguir, criamos uma classe BancoMongo.php para realização da conexão ao MongoDB, permitindo que as operações de inclusão, atualização e recuperação dos dados possam ser executadas. Outras classes que serão utilizadas nesta aplicação são:

- CadastroUsuario.php: responsável pela realização de cadastros dos usuários no sistema;
- AdicionarAmigo.php: lista todos usuários do sistema, para o usuário logado, e permite adicioná-los como amigos;
- PostarMensagem.php: permite que um usuário logado publique mensagens de texto que serão vistas por todos os seus amigos;
- Login.php: permite que um usuário realize autenticação no sistema. Após a validação, o usuário é redirecionado para o seu mural de mensagens;
- Mural.php: responsável pelo gerenciamento do mural dos usuários;
- Menu.php: exibirá os links de navegação para que o usuário possa navegar pelas telas do sistema.

Todo o código-fonte utilizado neste estudo de caso encontra-se disponível em: www.cin.ufpe.br/~hro, inclusive o código para criação de todas as classes mencionadas acima.

A seguir apresentamos a operação de criação do banco de dados e algumas operações básicas de inserção e atualização do banco de dados.

Criação do banco de dados

Para criação do banco de dados, utilizaremos o seguinte comando:

```
$this->db = $con->curso;
```

Este comando é executado na classe BancoMongo.php, e neste caso, faz a seleção da base de dados "curso". Observe que a base não existe inicialmente e só será criada efetivamente após o primeiro comando de inserção de dados;

Realizando as operações básicas no BD

Adicionando um usuário:

Após efetuar a conexão com o banco, criamos o usuário com id 1, com seus dados. Em seguida inserimos no banco através da função *insert*, como mostrado a seguir.

```
<?php
$m = new BancoMongo();
$db = $m->getDb();
$usuario = array(
    'id' => 1, 'nome' => 'usuario01', 'email' => 'email@email.com', 'senha' => '123456'
);
$db->usuario->insert($usuario);
?>
```

Consultando os dados de um usuário:

Aqui realizamos uma consulta, onde é retornado um array com os dados do usuário de id = 1 através do método *find*.

```
<?php
$m = new BancoMongo();
$db = $m->getDb();
$usuario = $db->usuario->find(array('id' => 1));
?>
```

Alterando usuário:

A alteração dos dados de uma usuário pode ser feita através do comando *update*, como mostrado a seguir. Neste exemplo alteramos o nome do usuário em que o id é igual a 1.

```
<?php
$m = new BancoMongo();
$db = $m->getDb();
$update = array('nome'=>'usuarioAlterado');
$db->usuario->update(array('id' => 1),$update);
?>
```

Adicionando uma mensagem:

Neste caso, primeiramente construímos uma mensagem adicionando o id e o texto. Em seguida, através do comando *update*, inserimos a mensagem no array correspondente ao usuário com id 1.

```
<?php
$m = new BancoMongo();
$db = $m->getDb();
$mensagem = array(array('id' => '01', 'texto' => 'mensagem 01'));
$db->usuario->update(array('id' => 1),$mensagem);
?>
```

Adicionando um amigo:

Para inserir um amigo, o processo é semelhante à inserção de uma mensagem. Primeiramente cria-se a variável que armazena os dados do amigo, em seguida adiciona através do método update ao usuário com id 1.

```
<?php
$m = new BancoMongo();
$db = $m->getDb();
$amigo = array(array('id' => '02', 'nome' => 'usuario2'));
$db->usuario->update(array('id' => 1),$amigo);
?>
```

Removendo usuário:

Por fim temos o exemplo de como excluir um usuário através do comando *remove*. Observe que não há validação de chave como ocorre no modelo relacional, sendo a aplicação responsável pelo controle de eventuais anomalias.

```
<?php
$m = new BancoMongo();
$db = $m->getDb();
$db->usuario->remove(array('id' => 1));
?>
```

Após a criação dos arquivos do sistema, poderemos executar a aplicação através de uma navegador web e utilizar todas funcionalidades do sistema.

5 Conclusão

Um ponto comum a todas as empresas que têm adotado a tecnologia NoSQL são os problemas enfrentados quando tem-se uma grande quantidade de dados, e estes precisam ser compartilhados em tempo real. Para isto, é necessário que as aplicações sejam escaláveis e seus dados tenham alta disponibilidade. Em sistemas colaborativos que necessitam destas características, como portais e comunidades online, fórum, correio eletrônico, agendas, abordagens NoSQL podem ser utilizadas como solução para armazenamento de dados. Devido ao gargalo causado pelos problemas encontrados nos modelos tradicionais de bancos de dados, várias empresas como a Google, Facebook e Amazon já aderiram a soluções NoSQL, cada uma de acordo com as necessidades dos serviços prestados. Isto nos mostra que esta nova visão dos bancos de dados terá uma boa perspectiva nos próximos anos, onde outras aplicações Web, dentre elas as colaborativas, tenderão à utilização do paradigma NoSQL.

É importante deixar claro que a solução NoSQL não veio com o intuito de substituir o modelo relacional, mas permitir que aplicações da Web 2.0 possam gerenciar os seus dados de forma mais eficiente, o que nem sempre é possível utilizando bancos de dados relacionais. Isto permite que as aplicações tenham vantagens como: alta disponibilidade, escalabilidade, esquema flexível, alta performance e gerenciamento de dados semi-estruturados. Em troca destes fatores, é importante ressaltar que nem sempre será possível garantir a consistência dos dados, controle de concorrência, dentre outras características fundamentais dos bancos de dados convencionais.

Referências

[Cattell 2010] Cattell, R., Scalable SQL and NoSQL data stores, ACM SIGMOD Record, v.39 n.4, 2010.

[Chang 2008]Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows M., Chandra,T., Fikes, A., Gruber, R. E., Bigtable: A Distributed Storage System for Structured Data, ACM Transactions on Computer Systems (TOCS), v.26 n.2, p.1-26, June 2008.

[DeCandia 2007] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S.,Vosshall, P., Vogels, W., Dynamo: Amazon'shighly available key-value store. In Proceedings of the21st ACM Symposium on Operating Systems Principles, October 2007.

[Hewitt 2011]Hewitt, E., Cassandra: the definitive guide, Journal of the Electrochemical Society, v. 129, p. 330, 2011.

[Lakshman 2010] Lakshman, A., Malik, P., Cassandra: a decentralized structured storage system, ACM SIGOPS Operating Systems Review, v.44 n.2, April 2010.

[Leavit 2010] Leavitt,N., Will NoSQL Databases Live Up to Their Promise?, IEEE Computer (COMPUTER), v.43 n.2, pp:12-14, 2010

[Pritchett 2008]Pritchett, D., BASE: AN ACID ALTERNATIVE, ACM Queue, v.6 n.3, May/June 2008.

[Stonebraker 2010]Stonebraker, M., SQL databases v. NoSQL databases, Communications of the ACM, v.53 n.4, April 2010.

[Stonebraker 2007] Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N., and Helland, P. 2007. The end of an architectural era: (it's time for a complete rewrite). In Proceedings of the 33rd international Conference on Very Large Data Bases, VLDB Endowment, pp: 1150-1160, 2007.

[Xiang 2010] Xiang P., Hou, R., Zhou, Z.,Cache and consistency in NOSQL, Computer Science and Information Technology ICCSIT 2010 3rd IEEE International Conference on, v. 6, pp: 117-120, 2010.

Cassandra Data Model(http://maxgrinev.com/2010/07/09/a-quick-introduction-to-the-cassandra-data-model/acesso em 30/05/2011).

NoSQLWhite Paper, CouchBase(http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf acesso em 30/05/2011).

NoSQL – Your Ultimate Guide to Non – Relational Universe, http://nosqldatabases. org/