DAYELE MONIQUE CRUZ

BANCOS DE DADOS NOSQL VERSUS RELACIONAL:

UM ESTUDO DE CASO ENTRE O MYSQL E O HBASE

LAVRAS – MG

2014

DAYELE MONIQUE CRUZ

BANCOS DE DADOS NOSQL VERSUS RELACIONAL:
UM ESTUDO DE CASO ENTRE O MYSQL E O HBASE

Artigo apresentado ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador
Denilson Alves Pereira

LAVRAS – MG

2014

DAYELE MONIQUE CRUZ

BANCOS DE DADOS NOSQL VERSUS RELACIONAL: UM ESTUDO DE CASO ENTRE O HBASE E O MYSQL

Trabalho de Conclusão de Curso de Graduação apresentado ao Colegiado do Curso de Bacharelado em Sistemas de Informação, para obtenção do título de Bacharel.

APROVADA em 25 de novembro de 2014.

Dr. Ahmed Ali Abdalla Esmin

Drª Marluce Rodrigues Pereira

Dr. Denilson Alves Pereira (Orientador)

LAVRAS-MG
Novembro/2014

Bancos de dados NoSQL versus relacional: Um estudo de caso entre o HBase e o MySQL

Dayele Monique Cruz
Denilson Alves Pereira (orientador)

Departamento de ciência da computação – Universidade Federal de Lavras (UFLA)
Cep 37200000 – Lavras – MG – Brazil

dayelemonique@gmail.com, denilsonpereira@dcc.ufla.br

Abstract. *Relational database is a type of database used by most developers, but is not suitable for all types of applications. NoSQL databases have emerged as technologies for applications requiring distributed processing and storage of large volumes of data. The aim of this work is to study relational databases and NoSQL, through a case study between MySQL and HBase. A simple social network was created to demonstrate how to do modeling, operations of query and data manipulation, and analysis of performance on reading and writing in these databases. The results of performance analysis show that HBase not distributed has a better write performance than MySQL and HBase distributed, MySQL has a better read performance than HBase not distributed in all cases studied. For few data, achieved a performance of reading data better than the HBase distributed, but does not endured all read operations performed. HBase distributed, in some cases, when data increased achieved a performance similar to or better than MySQL in readings.*

Resumo. *Banco de dados relacional é um dos tipos de banco de dados mais utilizados pelos desenvolvedores, mas não é adequado para todos os tipos de aplicações. Os bancos de dados NoSQL surgiram como tecnologias para aplicações que necessitam de processamento distribuído e armazenamento de grandes volumes de dados. O objetivo deste trabalho é realizar um estudo de bancos de dados relacionais e NoSQL, por meio de um estudo de caso entre o MySQL e o HBase. Foi criada uma rede social simples para demonstrar como são feitas as modelagens, as operações de consulta e manipulação de dados e uma análise de desempenho sobre a leitura e escrita nestes bancos de dados. Os resultados da análise de desempenho mostram que o HBase não distribuído possui um melhor desempenho de escrita de dados que o MySQL e o HBase distribuído, já o MySQL possui um melhor desempenho de leitura que o HBase não distribuído em todos os casos estudados. Para poucos dados, o HBase não distribuído obteve um desempenho de leitura de dados melhor que o HBase distribuído, mas não suportou todas as operações de leitura realizadas. O HBase distribuído, em alguns casos, a medida que os dados cresceram, obteve um desempenho para leituras semelhante ou melhor que o MySQL.*

1. Introdução

1.1. Contextualização

O modelo relacional para banco de dados foi introduzido por E. F. Codd em 1970 [Elmasri and Navathe 2010] e a partir da década de 80, cresceu muito até dominar o

mercado de banco de dados. Entretanto, os avanços da tecnologia, e principalmente a popularização da Web, fizeram com que os bancos de dados lidassem com grandes volumes de dados e de usuários, fazendo com que as necessidades relacionadas ao armazenamento e ao gerenciamento de banco de dados mudassem. Isso significa que algumas aplicações necessitam que seu banco de dados armazene um volume de dados cada vez maior e manipule esses dados com eficiência. Para isso é necessário maior poder de processamento, que pode ser conseguido adicionando-se mais memória e processadores a uma mesma máquina ou utilizando-se várias máquinas trabalhando em conjunto. A primeira alternativa é geralmente usada pelos bancos de dados relacionais, porém é uma alternativa cara e insuficiente para grandes volumes de dados. A segunda alternativa é mais barata e se consegue lidar com grandes volumes de dados, porém os banco de dados relacionais são inadequados para serem utilizados nesse tipo de estrutura. Devido a isso, abriu-se espaço para novas soluções relacionadas a banco de dados que atendessem a essas novas necessidades. Para essas soluções de banco de dados deu-se o nome NoSQL (*Not Only SQL*) [Vaish 2013].

Bancos de dados NoSQL são banco de dados que não seguem o modelo relacional, não utilizam a linguagem SQL (*Structured Query Language*) [Elmasri and Navathe 2010] como linguagem de consulta e cada solução possui um modelo de dados diferente, e este pode ser escolhido de acordo com o tipo da aplicação. Eles foram criados para permitir o particionamento de seus dados em várias máquinas, de tal forma que possam manipular grandes volumes de dados com baixo tempo de resposta e alta disponibilidade. Para obter isso, esses bancos de dados não garantem a consistência dos dados em todo momento. Apesar de atender aos requisitos ligados a armazenamento e gerenciamento de grandes volumes de dados, bancos de dados NoSQL não surgiram para substituir os bancos de dados relacionais, eles foram desenvolvidos para servir de alternativa para armazenamento e gerenciamento de dados de aplicações que não são adequados para os bancos de dados relacionais. Para escolher qual o melhor tipo banco de dados para determinada aplicação é necessário conhecer as características de cada tipo de banco de dados.

1.2. Objetivos gerais e específicos

O objetivo geral deste trabalho é realizar um estudo sobre bancos de dados relacionais e bancos de dados NoSQL a partir de um estudo de caso. Serão analisadas especificamente como são desenvolvidas e manipuladas aplicações em bancos de dados utilizando o MySQL [Ferrari 2007] e o HBase [George 2011].

Os objetivos específicos são:

- Demonstrar como são modelados os dados em um banco de dados relacional e em um banco de dados NoSQL orientado a colunas.
- Demonstrar como os bancos de dados são implementados utilizando os sistemas MySQL e HBase.
- Demonstrar como são feitas as operações de consulta e manipulação dos dados.
- Comparar o desempenho das duas tecnologias em uma aplicação exemplo.

1.3. Justificativa

Banco de dados são essenciais para várias aplicações que precisam armazenar e gerenciar seus dados de maneira eficiente, e em alguns casos um banco de dados relacional não é o

mais adequado para a aplicação. Por isso é importante conhecer quais as tecnologias de bancos de dados disponíveis, como funcionam e quais os benefícios que elas trazem, para que se faça a melhor escolha. Para isso, uma análise comparativa é muito útil

Há diversos trabalhos que trazem comparações entre bancos dados relacionais e bancos de dados NoSQL [Bonnet et al. 2011] [van der Veen et al. 2012] [Stonebraker 2010] [Cattell 2011] [Han et al. 2011] [Tudorica and Bucur 2011], mas geralmente são feitas comparações gerais sobre as características de cada tecnologia e/ou trazem um estudo de caso demonstrando como é feita uma aplicação em um determinado banco de dados NoSQL. A diferença entre este trabalho e os anteriores está no nível de detalhamento, pois será demonstrado detalhadamente como é implementada uma aplicação tanto em uma ferramenta de bancos de dados relacional quanto em uma ferramenta de banco de dados NoSQL. A escolha do MySQL para este estudo foi devido ao fato de ser um software livre muito popular, que já foi utilizado por redes sociais como Facebook [Borthakur et al. 2011]. A escolha do banco de dados orientado a colunas foi feita porque é o modelo de banco de dados NoSQL mais parecido com um banco de dados relacional, e a tecnologia HBase foi escolhida porque é um banco de dados NoSQL orientado a colunas de código aberto que assim como o MySQL, utilizado pelo Facebook [Borthakur et al. 2011]. As características deste trabalho ajudam os desenvolvedores na escolha do tipo de banco de dados que seja mais adequado para uma determinada aplicação.

2. Modelos de dados

2.1. Modelo relacional

No modelo relacional são criadas relações que são representações de entidades e relacionamentos do mini-mundo a ser modelado. Essas relações podem ser vistas como tabelas e armazenam dados das entidades e seus relacionamentos por meio de atributos, também conhecidos como colunas. Segundo [Elmasri and Navathe 2010] " Uma linha representa um fato que geralmente corresponde a uma entidade do mundo real ou relacionamento. O nome da tabela e os nomes das colunas são usados para ajudar a interpretar o significado dos valores em cada linha". Cada atributo possui um conjunto de valores que ele pode armazenar. Para identificar uma linha entre as demais em uma tabela, é utilizado um atributo chamado chave primária, que é único em cada linha.

Em um projeto de banco de dados relacional são utilizadas restrições de integridade, que segundo [Elmasri and Navathe 2010] "garantem que as mudanças feitas no banco de dados por usuários autorizados não resultem em uma perda da consistência dos dados. Portanto, as restrições de integridade protegem contra danos acidentais". E também são utilizadas regras de normalização para reduzir redundâncias e inconsistências.

Outra característica de um banco de dados relacional é o processamento de transações de acordo com as propriedades ACID (Atomicidade, Consistência, Isolamento, Durabilidade). De acordo com [Pokorny 2011]

> As propriedades ACID para transações em banco de dados determinam que: a informação deve ser atômica, o que significa que uma falha de uma única parte de uma operação resultará na falha da totalidade da operação,

> uma vez que apenas os dados válidos podem ser gravados no banco de dados, a informação tem de ser consistente (ler e escrever erros são evitados); múltiplas transações ao mesmo tempo não afetam umas as outras, esta é a exigência de isolamento, e, finalmente, a informação tem que ser durável, o que significa que é efetivamente armazenada no banco de dados (e não na memória).

Exemplos: MySQL, Oracle e SQL Server.
MySQL O MySQL [Baron Schwartz and Tkachenko 2012] é um popular sistema gerenciador de banco de dados relacional de código aberto, que utiliza SQL como linguagem de consulta. O MySQL possui um servidor que é responsável por armazenar dados, atender requisições de aplicações clientes e controlar a consistência.

2.2. Modelos NoSQL

O termo NoSQL se refere a bancos de dados que surgiram nos primeiros anos do século 21 para suprir a necessidade de bancos de dados para escala Web. Esses bancos de dados não seguem o modelo relacional nem usam SQL como linguagem de consulta, eles tentam resolver problemas relacionados a escalabilidade e à disponibilidade. Quase todas as implementações NoSQL oferecem a representação de dados sem esquema. Isso significa que não é preciso pensar muito à frente para definir a estrutura e pode-se continuar a evoluir ao longo do tempo [Vaish 2013]. Os principais modelos de dados para banco de dados NoSQL são:

Bancos de dados chave e valor

Bancos de dados chave-valor de acordo com [Han et al. 2011] "significa que um valor corresponde a uma chave". Além disso segundo [Hecht and Jablonski 2011]

> Os valores são isolados e independentes uns dos outros porque os relacionamentos devem ser tratados na lógica da aplicação. Devido a esta estrutura de dados muito simples, os armazenamentos de chave-valor são completamente livres de esquema.

Exemplos: Redis, Memcached, Voldemort.

Bancos de dados orientados a documentos

Bancos de dados orientados a documentos segundo [Bonnet et al. 2011],

> são organizados como documentos em coleções. Uma coleção contém objetos bastante semelhantes. Este tipo de base de dados é sem-esquema, uma vez que o número de campos não é limitado e pode ser adicionado de forma dinâmica para um documento.

Além disso, bancos de dados orientados a documentos utilizam JSON (*JavaScript Object Notation*) e cada documento contém uma chave chamada "ID", que é única dentro de uma coleção de documentos e, portanto, identifica um documento explicitamente [Hecht and Jablonski 2011].

Exemplos: MongoDB e CouchDB.

Banco de dados orientados a colunas

Bancos de orientados a colunas, de acordo com [Indrawan-Santiago 2012] "podem ser considerados como um tipo específico de modelo par de chave-valor", pois uma chave corresponde a diversos valores armazenados em sua estrutura. Bancos de dados orientados a colunas são compostos de colunas, famílias de colunas e supercolunas.

- Colunas
  De acordo com [Indrawan-Santiago 2012] “Uma coluna pode ser definida como uma unidade atômica de informação suportada pelo banco de dados. É expressa como um par de nome e valor”.
- Supercoluna
  Segundo [Indrawan-Santiago 2012]“ supercolunas agrupam colunas associadas, que seriam recuperadas juntas do disco ou tem associação semântica”. Podem ser visualizada como tabelas aninhadas.
- Família de colunas
  De acordo [Indrawan-Santiago 2012] “Uma família de colunas agrupa colunas e super colunas juntas em um dado altamente estruturado”.

Segundo [Indrawan-Santiago 2012]

> A estrutura das supercolunas e família de colunas determinam o esquema do banco de dados. Entretanto, não é estritamente fixo. Uma nova coluna ou supercoluna pode ser adicionada com facilidade ao design quando o banco de dados já está implementado. Uma importante diferença entre bancos de dados orientados a colunas e bancos de dados relacionais é que cada linha do banco de dados de família de colunas não precisa ser do mesmo grau, ou seja, pode ter um número variável de colunas/supercolunas. Assim, família de colunas é muito eficaz no apoio a coleta de dados altamente esparso.

Exemplos: Cassandra, HBase e Google’s BigTable.

Banco de dados orientado a Grafos

Bancos de dados orientados a grafos segundo[Vaish 2013],

> representam uma categoria especial de bancos de dados NoSQL onde os relacionamentos são representados como grafos. Pode haver várias ligações entre dois nós em um grafo, representando as múltiplas relações que os dois nós compartilham. As relações representadas podem incluir as relações sociais entre as pessoas, ligações de transporte entre lugares, ou topologias de rede entre sistemas conectados.

Exemplos: Neo4J e FlockDB.

HBase O HBase é um sistema de banco de dados NoSQL orientado a colunas, que segundo [Dimiduk and Khuarana 2013], “fundamentalmente é uma plataforma para armazenar e recuperar dados com acesso aleatório”. Com esse banco de dados é possível a construção de um modelo de dados dinâmico e flexível, pois ele não restringe os tipos de dados inseridos nele e novas colunas podem ser adicionadas quando o banco de dados está implementado[Dimiduk and Khuarana 2013]. Os dados do HBase são geralmente inseridos dentro de uma célula que é encontrada através da *Rowkey*, que é uma espécie de chave no qual se identifica unicamente uma linha em uma tabela, família de colunas e colunas. Cada dado inserido no HBase é versionado, permitindo-se amarzenar valores diferentes referentes a uma célula apenas mudando sua versão. Na arquitetura do HBase encontra-se os seguintes componentes (ver Figura 1):

HFiles: são arquivos de armazenamento persistente. Cada HFile pertence a uma família de colunas de uma determinada tabela no HBase[Dimiduk and Khuarana 2013].

HDFS (*Hadoop Distribuited File System*): de acordo com [George 2011] “fornece um escalável, persistente, replicada camada de armazenamento para o HBase”.

Wal (*writed-ahead log*): segundo [George 2011], é o primeiro local onde o dado é armazenado no HBase.

Memstore: Segundo[George 2011], é o segundo local onde os dados são armazenados no hbase.

Region: é uma unidade de escalabilidade e balanceamento de carga, no qual são armazenados intervalos de linhas [George 2011].

Region server: de acordo com [George 2011], “ é responsável por todas as requisições de leitura e escrita de todas as regiões a que servem”.

Master server: segundo [George 2011], “ é responsável por atribuir regiões para as *Region server* ”.

Zookeeper: é de acordo com [George 2011] “um confiável, altamente disponível, persistente e distribuído serviço de coordenação”.

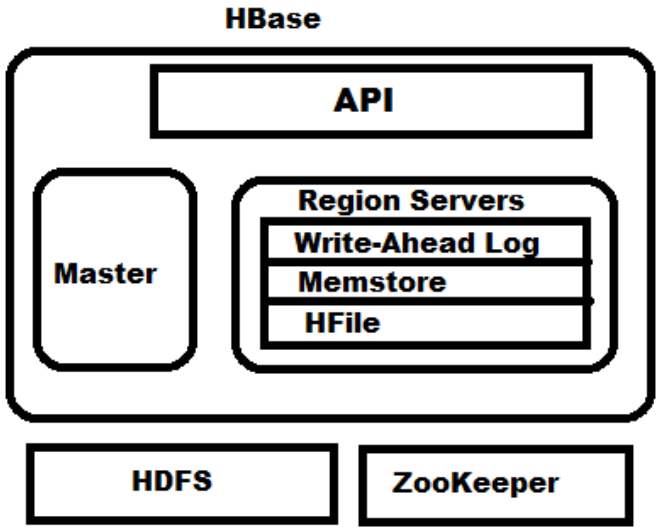

Figura 1. Componentes HBase adaptado de [George 2011]

3. Estudo de caso: MySQL x HBase

3.1. Descrição do estudo de caso

Neste trabalho é feito um estudo de caso no qual se implementa uma aplicação de rede social em um banco de dados relacional e em um banco de dados NoSQL orientado a colunas. Este estudo de caso mostra como são projetados esses banco de dados e como são executadas as operações de consulta, inserção, atualizações e exclusão de dados. São utilizados os softwares MySQL (sistema gerenciador de banco de dados relacional) e HBase (sistema de banco de dados NoSQL orientado a colunas) para a implementar esses bancos de dados. Após isso, é realizada uma análise de desempenho utilizando as operações de consulta e manipulação dos dados.

3.2. Modelagem conceitual

O banco de dados deve armazenar dados de uma rede social com as seguintes características:

- A rede social é composta por usuários, e cada um possui uma identificação (ID) única. O sistema deve armazenar além do ID, os dados pessoais do usuário contendo: nome, sobrenome, email, senha, data de nascimento, todos os telefones que o usuário tenha e informações pessoais descritas por ele.
- Cada usuário pode ter relação de amizade com outros usuários da rede social, esta relação deve ser armazenada pelo banco de dados.
- Cada usuário poderá criar álbuns de fotos e estes álbuns devem ter um nome fornecido pelo usuário e podem conter diversas fotos.
- Cada usuário pode trocar mensagens entre outros usuários da rede social. Cada mensagem deverá conter a pessoa que envia, a pessoa que recebe, o texto da mensagem, a data e hora de envio.

3.3. Modelagem Relacional

Para a modelagem do banco de dados relacional de acordo com a modelagem conceitual, foi criado o diagrama relacional da Figura 2, contendo as seguintes tabelas:

Usuario Esta tabela armazena as informações pessoais do usuário e tem como chave primária o idUsuario que contém o id fornecido pelo usuário, e os demais atributos são: nome, sobrenome, email, senha, dataNascimento e infoPessoais.

Telefone O atributo telefone das informações pessoais do usuário foi transformado em uma nova tabela por poder conter diversos valores, sendo caracterizado assim como um atributo multi-valorado.

Amizade Para armazenar as informações de amizade foi criada uma tabela chamada *Amizade*, esta tabela contém como chaves estrangeiras idUsuario e idUsuarioAmigo que são chaves estrangeiras da tabela usuário e representam participantes da relação de amizade.

Album Para armazenar os álbuns criados pelo usuário foi criada uma tabela chamada *Album*, o qual possui como chave primária idAlbum que é auto-incrimentável, o idUsuario que é uma chave estrangeira e o atributo nomeAlbum.

Foto Cada album pode conter diversas fotos, para isso, foi criada uma tabela chamada *Foto* contendo o idFoto como chave primária auto-incrementável, idAlbum como chave estrangeira e o atributo foto que contém a foto fornecida pelo usuário.

Mensagem Para armazenar as mensagens trocadas entre os usuários, foi criada uma tabela chamada *Mensagem*, que tem como chave primária o atributo idMensagem. Os atributos idUsuarioEnvia e idUsuarioRecebe são chaves estrangeiras e os demais atributos: mensagem e timestamp.

3.4. Modelagem Orientada a colunas

Para modelar o banco de dados orientado a colunas, foi criado o esquema relacional da Figura 3 com as seguintes tabelas:

Usuario

A *Rowkey* desta tabela é formada pelo id do usuário. Esta tabela é formada por duas família de colunas: DadosPessoais e Album. A família de colunas DadosPessoais tem as colunas: nome, sobrenome, email, senha, dataNascimento, infoPessoais e uma supercoluna que contém os telefones do usuário. O nome das colunas

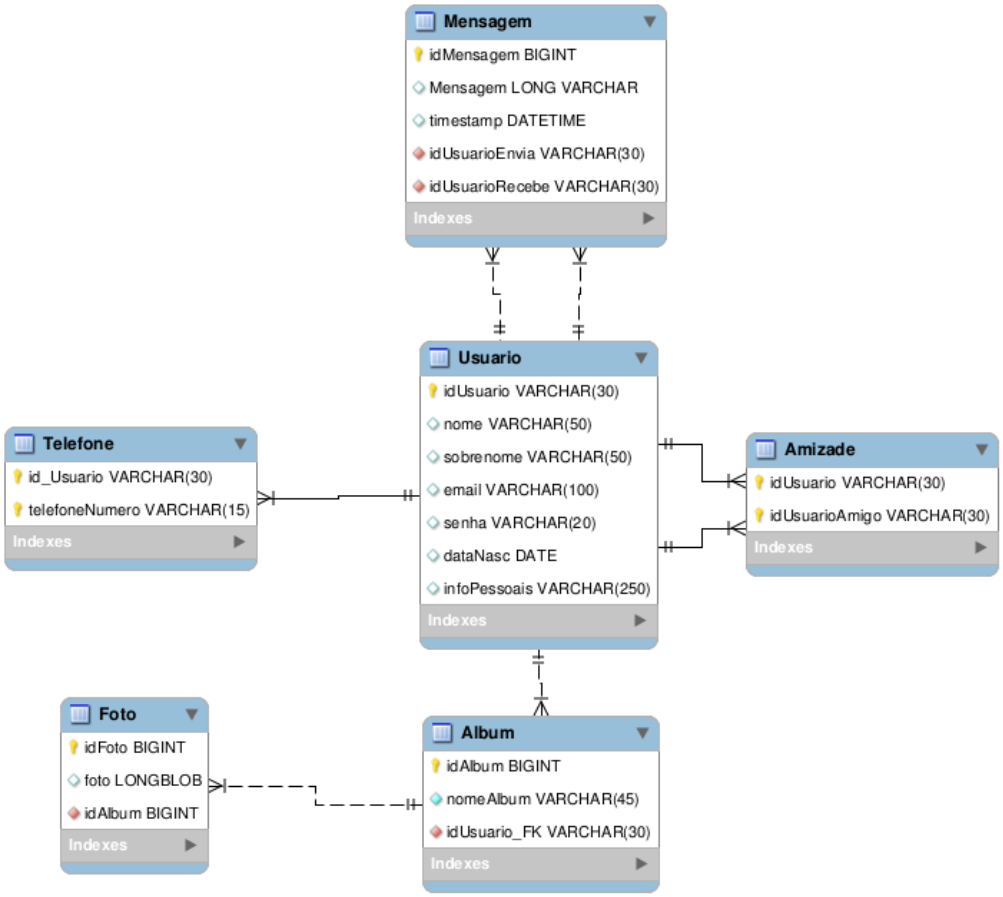

Figura 2. Diagrama relacional para a rede social

desta supercoluna armazenam os valores dos telefones e uma constante é armazenada na célula, porque esta não pode ficar vazia. A família de colunas Album armazena uma supercoluna contendo colunas com os nomes dos álbuns do Usuário, e outra supercoluna que armazena as fotos de cada álbum. Observe o exemplo de preenchimento da tabela *Usuario* na Figura 4. Os prefixos tel, nomeAlbum e foto são utilizados para identificar o membro da supercoluna, os timestamps utilizados no nome de cada coluna foto servem para ajudar a nomear dinamicamente estas colunas, fazendo com que cada coluna tenha um nome diferente.

Amizade

A *RowKey* desta tabela é formada pelo id do usuário e id do usuário amigo. A escolha desta *Rowkey* foi feita para que se saiba rapidamente se um usuário é amigo de outro, apenas com seus ids. Esta tabela tem uma família de colunas denominada Amizade, que contém no nome da coluna o valor que deveria ser inserido, que é o nome do amigo. Este mecanismo armazena dados redundantes, já que o nome é armazenado na tabela *Usuario*, mas permite uma rápida obtenção do nome do amigo. O valor que é armazenado realmente é uma constante, pois a célula que é referenciada não pode ser vazia. A Figura 5 ilustra como seriam inseridos os dados na tabela Amizade, a *RowKey* formada pela junção dos ids de Maria e de João, o nome de João armazenado dentro da célula e a constante representada pelo numero um.

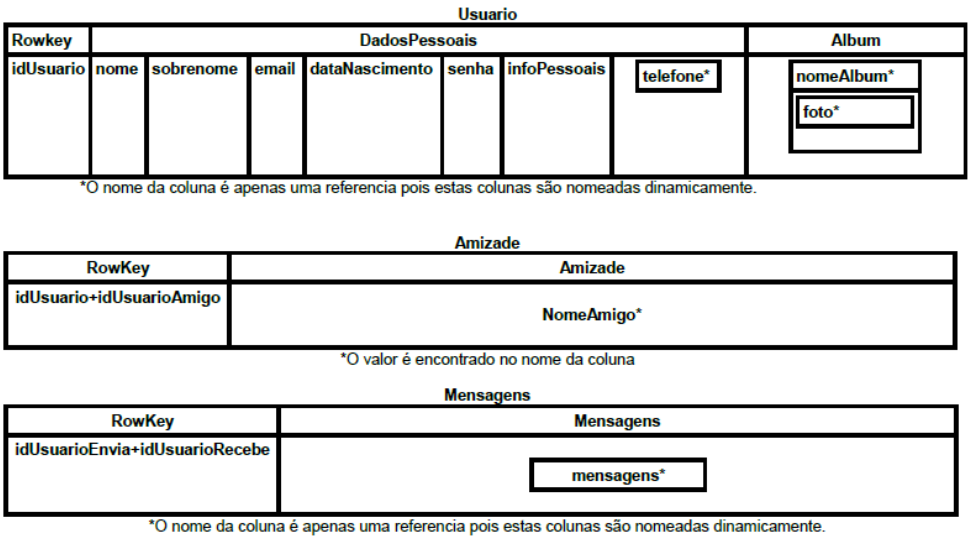

Figura 3. Banco de dados Orientado a colunas

Rowkey	DadosPessoais	Album
"idMaria"	**nome:**Maria	**nomeAlbum_Família**:1
	sobrenome: Silva	**foto_Família_1415060941000** : imagem.jpg
	email:mariasilva@ufla.br	**foto_Família_1415060941040** : imagem2.jpg
	dataNascimento: 19-04-1991	**nomeAlbum_Amigos**:1
	Senha: 123456	**foto_Amigos_1415060991040** : imagem3.jpg
	Tel_9999-0219:1	**foto_Amigos_1415070991040** : imagem4.jpg
	Tel_9292-9292:1	

Figura 4. Exemplo de preenchimento da tabela Usuario

Mensagem

A *RowKey* desta tabela é formada pelo id do usuário que envia a mensagem e id do usuário que recebe a mensagem. Esta tabela contém uma família de colunas denominada mensagem, que contém uma supercoluna que armazena todas as mensagens enviadas pelo usuario do primeiro id e recebidas pelo usuário do segundo id.

3.5. Comparações entre as modelagens

As modelagens feitas para o banco de dados relacional e o banco de dados orientado a colunas são muito diferentes, as principais características são:

Estrutura das tabelas

Na modelagem para o banco de dados relacional, as informações contidas na modelagem conceitual são transformadas em relações, de acordo com as entidades e relacionamentos encontrados na modelagem conceitual. Cada relação possui atributos que armazenam dados da entidade ou do relacionamento que ela representa. As relações são criadas de forma a atender aos requisitos da normalização dos dados.

RowKey	Amizade
idMaria_idJoao	João da silva:1
idCarlos_idMarcos	Marcos José:1

Figura 5. Exemplo de preenchimento da tabela Amizade

Na modelagem para o banco de dados orientado a colunas, as tabelas são desenhadas com foco nas consultas. Dados que costumam ser obtidos juntos devem ser armazenados na mesma família de colunas, porque as colunas em uma determinada família de colunas são armazenadas juntas, na mesma região do disco, assim, as informações contidas na modelagem conceitual foram transformadas em tabelas de acordo com a recuperação de seus dados. Por exemplo, a tabela *Usuario* possui duas famílias de colunas: DadosPessoais e Album. As colunas contidas nelas costumam ser obtidas juntas, assim como as outras colunas das famílias de colunas das demais tabelas.

Estrutura das colunas

Em um banco de dados relacional, os atributos de uma relação precisam ser definidos no projeto assim como seus domínios. Quando um atributo de uma relação é mutivalorado, este atributo é armazenado em uma nova tabela. O atributo telefone da entidade Usuário da modelagem conceitual e o atributo foto da entidade Album foram transformados em novas tabelas na modelagem para o banco de dados relacional. Já para as colunas de um banco de dados orientado a colunas, o tipo de valores contidos nelas não precisa ser definidos na hora do projeto, colunas podem ser criadas e nomeadas dinamicamente quando o banco de dados já está implantado e o banco de dados trata o nome das colunas e os seus valores como vetor de bytes. As linhas em um banco de dados orientado a colunas podem ter colunas e quantidade de colunas diferentes. Na tabela *Usuario*, a supercoluna telefone da família de colunas DadosPessoais pode ter vários valores, formando uma tabela aninhada dentro da família de colunas. Isso é possível porque cada coluna que contém o valor telefone é nomeado dinamicamente. O mesmo ocorre nas supercolunas foto e nomeAlbum da família de colunas Album e na tabela *Mensagem* com a supercoluna mensagens da família de colunas Mensagens. O nome das colunas pode conter valores, esta propriedade pode ser observada na tabela *Amizade*, o nome do amigo pode ser recuperado através do nome da coluna.

Estrutura das chaves

Em um banco de dados relacional, cada relação possui uma chave primária, que é um atributo que identifica unicamente cada linha em uma tabela. Na relação *Usuario* o atributo idUsuario é usado como chave primária, na relação *Amizade* idUsuario e idUsuarioAmigo formam uma chave primária composta, que também são chaves estrangeiras. As tabelas *Mensagem*, *Foto* e *Album* possuem chaves primárias auto incrementáveis.

Cada linha de um banco de dados orientado a colunas possui uma *RowKey* que identifica unicamente esta linha. Estas *RowKeys* são uma importante questão de

projeto, pois podem ajudar a recuperar informações de forma eficiente. Na modelagem realizada, o idUsuario é usado como *RowKey* da tabela *Usuario*, na tabela *Mensagem* a *RowKey* é composta pela id do usuário que envia e o id do usuário que recebe, pois pode-se identificar rapidamente as mensagens envidas de um usuario para outros conhecendo seus ids.

Relacionamentos

Em um banco de dados relacional, relacionamentos representam associação entre tabelas. As relações *Album*, *Mensagem*, *Amizade* e *Telefone* possuem relacionamento com a tabela *Usuario*, as quais são representas por chaves estrangeiras usando o atributo idUsuario.

Bancos de dados orientados a coluna não dão suporte a relacionamentos usando o conceito de chave estrangeira, e não é recomendado fazer códigos que criem este tipo de restrição de integridade, por questões de eficiência.

3.6. Consulta e manipulação dos dados

Para realizar operações de consulta e manipulação de dados no MySQL pode se utilizar uma *interface de linha de comandos* e também utilizar APIs que possibilitam a conexão de uma aplicação cliente com o banco de dados através de um driver. As operações de manipulação de dados no HBase podem ser feitas através do *shell* que é uma interface que permite executar comando sobre o banco de dados, e através de APIs.

3.6.1. Inserção de dados

MySQL

No MySQL as inserções são feitas com o comando SQL INSERT, no qual é necessário indicar o nome da tabela, as colunas e valores a serem inseridos. O exemplo abaixo insere na tabela *Mensagem* os valores “idMaria”, “idJoao”, “Tudo bem?” e a data corrente nas colunas idUsuarioEnvia, idUsuarioRecebe, Mensagem, timestamp, respectivamente.

```
1  INSERT into Mensagem (idUsuarioEnvia, idUsuarioRecebe,
       Mensagem, timestamp) values ('idMaria','idJoao','Tudo
       Bem?','02-04-14 00:00:00');
```

HBase

Através da API Put insere-se valores dentro se uma tabela no HBase. Esta API exige que seja indicado o nome da tabela, a *RowKey*, o nome da família de colunas, a coluna e o valor a ser inserido. O HBase automaticamente cria um timestamp para o valor inserido que indica o dia e a hora que o valor foi inserido. O exemplo abaixo insere na tabela *Mensagem* a *RowKey* que é a junção do idMaria e idJoao e é criada uma coluna com o prefixo “msg” junto com a data corrente para que as colunas sejam nomeadas dinamicamente. O valor inserido na coluna criada é “Tudo bem?”. No exemplo as APIs Configuration, HConnection, HTableInterface são responsáveis por criar uma configuração com recursos do HBase, criar uma conexão e dar acesso à tabela respectivamente.

```
1        Configuration config = HBaseConfiguration.create();
2        HConnection connection = HConnectionManager.
             createConnection(config);
3        HTableInterface table = connection.getTable(Bytes.toBytes(
             Mensagem));
4        String chave = ``idMaria" + ``_" + ``idJoao";
5        Put put = new Put(Bytes.toBytes(chave));
6        Date data = new Date();
7        String colMsg = ``msg"+``_" + data.getTime();
8        put.add(Bytes.toBytes(Mensagem), Bytes.toBytes(colMsg),
             Bytes.toBytes(`` tudo bem?"));
9        table.put(put);
```

3.6.2. Alteração de dados

MySQL

Para atualizar um valor em uma tabela no MySQL o comando UPDATE é utilizado. Neste comando são utilizados o nome da tabela que se deseja alterar, as colunas e os novos valores. Em uma alteração, os valores antigos são substituídos pelos valores novos. Para especificar um conjunto de linhas a serem modificados, operações lógicas podem ser utilizadas. O exemplo abaixo demonstra como alterar na tabela *Usuario* o valor da senha de um usuário com o idUsuario "idMaria".

```
1    UPDATE Usuario set senha ='1234' where idUsuario ='idMaria';
```

HBase

Atualizações no HBase são feitas da mesma forma que a inserção, utilizando-se a API Put. O novo valor é armazenado como uma nova versão e o antigo valor permanece no banco de dados como uma versão mais antiga. O número de versões armazenas pelo banco de dados pode ser determinada por configuração. O exemplo abaixo demonstra como alterar na tabela *Usuario* o valor da coluna senha utilizando a *RowKey* "idMaria"e indicando a família de colunas "DadosPessoais".

```
1        Configuration config = HBaseConfiguration.create();
2        HConnection connection = HConnectionManager.
             createConnection(config);
3        HTableInterface table = connection.getTable(Bytes.toBytes(
             Mensagem));
4    Put put = new Put(Bytes.toBytes(``idMaria"));
5     put.add(Bytes.toBytes(DadosPessoais), Bytes.toBytes(``senha")
         , Bytes.toBytes(``123"));
6     table.put(put);
```

3.6.3. Exclusão de dados

MySQL

A exclusão de linhas no MySQL é realizada através do comando SQL DELETE, para executar este comando, é necessário o nome da tabela e para restringir as linhas a serem deletadas operaração lógica devem ser utilizadas. O exemplo abaixo demonstra como excluir uma foto da Tabela *Foto* utilizando o id da foto igual a 1.

```
1
2   DELETE from Foto where idFoto =1
```

HBase

No HBase, através da API Delete, é possível excluir uma linha, família de colunas e colunas dentro de uma tabela no HBase. As versões a serem excluídas podem ser especificadas. Para realizar essa operação, é necessário que se indique o nome da tabela, a *RowKey* da linha será excluída e para restringir os valores a serem excluídos, pode se indicar a família de colunas e coluna, e, caso queira excluir apenas uma versão específica, o timestamp. O exemplo abaixo demonstra como excluir a última versão de uma foto da tabela *Usuario* com a *RowKey* "idMaria" indicando a família de colunas Album e o nome da coluna no qual a foto será excluída.

```
1  Configuration config = HBaseConfiguration.create();
2  HConnection connection = HConnectionManager.createConnection(
       config);
3  HTableInterface table = connection.getTable(Bytes.toBytes(
       Mensagem));
4  Delete delete = new Delete(Bytes.toBytes(``idMaria"));
5  delete.deleteColumn(Bytes.toBytes(Album), Bytes.toBytes(foto));
6  table.delete(delete);
```

3.6.4. Consulta a dados

MySQL

No MySQL, as consultas são realizadas através do comando SQL SELECT. O exemplo abaixo demonstra como obter o valor da coluna email do usuário que tem o id "idMaria"da tabela *Usuario* .

```
1  SELECT   email from Usuario where idUsuario = 'idMaria'
```

HBase

Existem dois tipos de consulta no HBase usando as APIs Get e Scan. A primeira delas, é utilizada para obter uma linha específica dentro da tabela, que é realizada através da API GET, na qual é necessário que se indique o nome da tabela e a *RowKey* que se deseja o resultado, esta API também permite que restrinja o resultado da consulta à algumas família de colunas e colunas, apenas indicando o nome

delas. O exemplo abaixo mostra como através da *RowKey* “idMaria”, obter o valor da coluna email da tabela *Usuario* dentro da família de colunas DadosPessoais. O resultado das consultas com Get são obtidos através da API Result.

```
        Configuration config = HBaseConfiguration.create();
        HConnection connection = HConnectionManager.
            createConnection(config);
        HTableInterface table = connection.getTable(Bytes.toBytes(
            Mensagem));

    Get get = new Get(Bytes.toBytes(``idMaria''));
    Result result = table.get(get);
    Cell kv = result.getColumnLatestCell(Bytes.toBytes(``
        DadosPessoais''), Bytes.toBytes(``email''));
    String email = new String(CellUtil.cloneValue(kv), ``UTF-8'');
```

Com o segundo tipo de consulta, utilizando a API Scan, pode-se obter um conjunto de linhas e/ou realizar consultas mais restritas. Um Scan pode ser realizado de diversas formas, por exemplo, indicando as *RowKeys* de começo e fim ou indicando uma instancia da API Get. Para restringir este tipo de consulta, podem ser utilizados o nome da família de colunas e/ou nome da coluna do qual se quer obter o valor. Outro mecanismo para restrição de consultas é através de filtros, com os quais pode-se fazer diversos tipos de restrições. Por exemplo, obter um conjunto de colunas com comecem com determinado prefixo. Os filtros no HBase são imprescidíveis para se obter as supercolunas. No exemplo abaixo, é demonstrado como obter todas as mensagens enviadas de um usuário para outro utilizando a *RowKey* que é obtida atraés da junção do “idMaria”e “idJoao”e indicando a família de colunas DadosPessoais. Foi necessário utilizar um filtro para obter as colunas que tem o prefixo “msg”. O resultado das consultas com Scan são obtidos através da API ResultScanner.

```
    Configuration config = HBaseConfiguration.create();
    HConnection connection = HConnectionManager.createConnection(
        config);
    HTableInterface table = connection.getTable(Bytes.toBytes(
        Mensagem));
    String chave = ``idMaria'' + ``_'' + ``idJoao'';
    Scan scan = new Scan(Bytes.toBytes(chave), Bytes.toBytes(
        chave));
    Filter f = new ColumnRangeFilter(Bytes.toBytes(``msg''), true,
         null, false);
    scan.setFilter(f);
    ResultScanner scanner = table.getScanner(scan);
    for (Result row : scanner) {
      for (Cell cel : row.rawCells()) {
        Cell kv = row.getColumnLatestCell(cfMensagem,
            CellUtil.cloneQualifier(cel));
        String msg = new String(CellUtil.cloneValue(kv), ``UTF-8''
            );
        Long timestamp = kv.getTimestamp();
        mensagens.add(new Mensagem(msg, timestamp));
      }
```

3.7. Análise de desempenho

As análises de desempenho realizadas neste trabalho avaliaram o tempo que o MySQL e o HBase gastam para realizar tarefas de leitura e escrita de dados. Essas análises foram feitas utilizando-se as mesmas quantias de dados nos bancos de dados observados. Os dados armazenados são iguais para o MySQL e HBase e as operações de consulta e manipulação de dados são realizadas focando-se em operações que se insira e recupere valores semelhantes. As análises são feitas em duas etapas. Na primeira etapa, as tecnologias estudadas foram instaladas em uma máquina virtual e o desempenho destas tecnologias comparados. A máquina virtual utilizada tem as seguintes configurações: sistema operacional Ubuntu 14.04.1 32 bits, 2GB de memória RAM e CPU Intel(R) Core(TM) i3-3110M CPU @ 2.40GHz. Foi utilizada a versão 5.5.38 do MySQL e a versão 0.98.5 do HBase. Na segunda etapa, as análises foram feitas com o HBase distribuído em um cluster contendo cinco máquinas. As mesmas operações de consulta e manipulação de dados utilizadas na primeira etapa foram executadas. O cluster tem a seguinte configuração: sistema operacional Ubuntu 12.04.4 32 bits, 2GB de memória RAM e CPU Core 2 Duo E4400 2.0GHz para a máquina master, 4GB de memória RAM e CPU Core 2 Duo E4500 2.2GHz para a primeira e segunda máquinas escravas, 4GB de memória RAM e CPU Athlon x2 Dual Core 2.6GHz para o terceira e quarta máquinas escravas. O cluster utilizou a versão 0.96.1 do HBase e a versão 2.0 do HADOOP, a configuração utilizada foi a padrão, sem qualquer mecanismo de otimização. O tempo médio das operações de leitura e de escrita é obtido através da média do tempo gasto para realizar cinco repetições de cada operação. É importante ressaltar que existem fatores podem influenciar nos resultados obtidos, como a diferença entre os processadores utilizados nas máquinas do ambiente distribuído e do processador usado na máquina do ambiente não distribuído e os atrasos decorrentes da comunicação entre as máquinas do cluster.

3.7.1. Operações de escrita

As análises sobre as operações de escrita de dados foram feitas observando-se o tempo que o MySQL e o HBase gastam para armazenar determinada quantia de dados em suas tabelas. A tabela da Figura 6 mostra o tempo médio que as tecnologias estudadas gastam para inserir os valores e quantidades descritas.

De acordo com os resultados obtidos, o tempo gasto pelo HBase não distribuído para realizar as operações de escrita em todas as tabelas foi menor que o MySQL, e o tempo que o MySQL realizou esta operação foi menor que HBase distribuído.

3.7.2. Operações de leitura

Para as análises de leitura dos dados foram realizados três tipos de consulta: o primeiro tipo analisa o tempo gasto para recuperar todos os valores armazenados em uma tabela, por exemplo, obter todas as mensagens armazenadas na tabela *Mensagem*. O segundo

Operação	Tecnologia utilizada	Tempo médio em segundos
Inserir 90.000 usuários	MySQL	63,2
	HBase não-distribuído	55,4
	HBase distribuído	139,5
Inserir 1.500 fotos	MySQL	74,8
	HBase não-distribuído	28,8
	HBase distribuído	152,8
Inserir 90.000 mensagens	MySQL	56,4
	HBase não-distribuído	43,8
	HBase distribuído	110,2

Figura 6. Tempo gasto nas operações de escrita de dados

tipo analisa o tempo gasto para recuperar uma linha utilizando-se a chave primária no MySQL e a *RowKey* no HBase, por exemplo, obter todos os álbuns e fotos de um usuário através de seu id. O último tipo de consulta analisou o tempo gasto para recuperar uma linha de uma tabela através de um atributo que não seja chave (MySQL) e *RowKey* (HBase), por exemplo, obter através do atributo nome do usuário, todos os seus amigos.

O gráfico da Figura 7 mostra o tempo em que as tecnologias estudadas demoraram para obter todos os valores contidos na tabela Amizade de acordo com as quantidades descritas. Nele pode-se observar que o HBase não distribuído não suportou consultas com mais dois milhões e setecentas mil linhas. Este gráfico caracteriza o primeiro tipo de consulta, e como pode ser observado, o MySQL demora menos que as demais tecnologias para realizar esse tipo de consulta. Outras consultas semelhantes foram realizadas e o resultado foi similar.

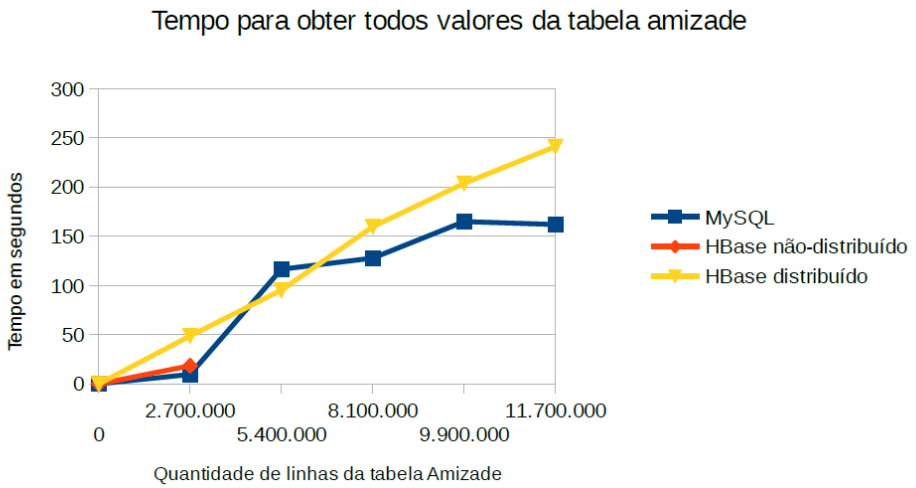

Figura 7. Operações de leitura

No segundo tipo de consulta, tanto o MySQL quanto o HBase distribuído, em média, demoraram aproximadamente 0,01 segundos, independente da quantidade de dados na tabela, tornando-se assim equivalentes. Já o HBase não distribuído em média realiza este tipo de consulta entre 0,01 e 0,5 segundos.

O gráfico da Figura 8 mostra o tempo em que as tecnologias estudadas demoraram para obter todos os amigos de um usuário pelo nome, nesta consulta, o MySQL utilizou as tabelas *Usuario* e *Amizade* e o HBase utilizou apenas a tabela *Amizade*. Note que a partir de três milhões e seiscentas mil linhas na tabela, o HBase não-distribuído não suporta este tipo de consulta. Neste tipo de consulta, com poucos dados, o MySQL demora menos tempo para concluir a consulta que as demais tecnologias, porém a medida que os dados crescem, o HBase distribuído torna-se mais rápido que o MySQL.

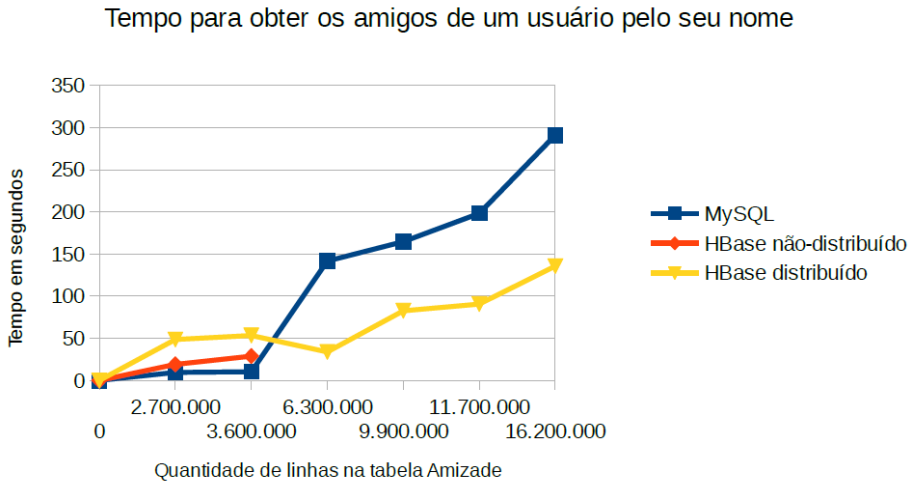

Figura 8. Operações de leitura

3.8. Análise qualitativa

Na escolha entre o MySQL e o HBase as seguintes características podem ajudar:

Consistência O MySQL é totalmente consistente pois atende a todos os princípios ACID, enquanto o HBase por questões de desempenho, tem somente consistência a nível de linha [Tudorica and Bucur 2011].

Flexibilidade O MySQL oferece um esquema fixo, pois ele exige que sejam atendidas as regras de normalização de dados, enquanto o HBase oferece um esquema bastante flexível, pois além de permitir a inserção de dados denormalizados, todas as colunas podem ser definidas quando o banco de dados já está implementado e trata todos os dados inseridos nele como um vetor de bytes.

Tolerância à falhas Esta é a propriedade que indica se o sistema é capaz de continuar funcionando mesmo com alguma falha. Tanto o MySQL quanto o HBase são tolerantes à falhas quando estão operando em um cluster [Tudorica and Bucur 2011].

4. Conclusão

Neste trabalho foi realizado um estudo sobre bancos de dados relacionais e NoSQL a partir de um estudo de caso. Foram desenvolvidos bancos de dados para uma rede social simples, utilizando-se as tecnologias HBase e MySQL. As modelagens realizadas

demonstram que esses bancos de dados são muito diferentes entre si, pois foram desenvolvidos com propósitos diferentes. Enquanto a modelagem para banco de dados relacional define todo um conjunto de regras rígidas para se armazenar dados de forma consistente, a modelagem para bancos de dados NoSQL orientado a colunas fornece diversos mecanismos para facilitar a distribuição dos dados em diversas máquinas e é extremamente flexível e adaptável à aplicação.

Através dos mecanismos de manipulação de dados utilizados pelas tecnologias MySQL e HBase, pode-se observar que a linguagem SQL é extremamente rica e de fácil compreensão, já as APIs disponibilizadas pelo HBase para manipulação de dados são muito flexíveis para que se encaixe ao modelo de banco de dados orientado a colunas, porém exige do desenvolvedor uma habilidade maior de programação.

A análise de desempenho realizada neste trabalho, estudou o tempo gasto pelo MySQL e pelo HBase para realizar operações de escrita e de leitura em um ambiente não distribuído e realizar as mesmas operações com HBase em um ambiente distribuído. Os resultados mostram que o HBase foi desenvolvido para ambientes distribuídos, não sendo viável para manipular grandes volumes de dados em ambientes não distribuídos. O HBase distribuído obteve um desempenho melhor para leitura de dados que não são nem índices, nem *RowKey*, para volumes maiores de dados. Para consultas feitas por *RowKey* e chaves primárias, o HBase distribuído obteve um desempenho semelhante ao MySQL. Nas consultas para obter todos os dados de uma tabela, o MySQL obteve um melhor desempenho. É importante destacar que, mesmo usando um ambiente de testes limitado neste trabalho, foram avaliados o desempenho das tecnologias com alguns milhões de linhas de dados nas tabelas. Porém, redes sociais reais, como o Facebook, armazenam volumes de dados muito maiores, o que pode tornar o uso do MySQL inviável [Borthakur et al. 2011].

Referências

Baron Schwartz, P. Z. and Tkachenko, V. (2012). *High Performance MySQL*. O'Reilly Media, USA, 3 edition.

Bonnet, L., Laurent, A., Sala, M., Laurent, B., and Sicard, N. (2011). Reduce, you say: What nosql can do for data aggregation and bi in large repositories. In *22nd International Workshop on Database and Expert Systems Applications (DEXA)*, pages 483–488.

Borthakur, D., Gray, J., Sarma, J. S., Muthukkaruppan, K., Spiegelberg, N., Kuang, H., Ranganathan, K., Molkov, D., Menon, A., Rash, S., Schmidt, R., and Aiyer, A. (2011). Apache hadoop goes realtime at facebook. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 1071–1080, New York, NY, USA. ACM.

Cattell, R. (2011). Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39:12–27.

Dimiduk, N. and Khuarana, A. (2013). *HBase in Action*. Manning Publications Co., Shelter Island, NY.

Elmasri, R. and Navathe, S. B. (2010). *Fundamentals of database systems*. Addison-Wesley, USA, 6 edition.

Ferrari, F. A. (2007). *Crie banco de dados em MySQL*. Digerati Books, São Paulo, SP.

George, L. (2011). *Hbase The definitive guide*. O'Reilly Media, USA.

Han, J., Haihong, E., Le, G., and Du, J. (2011). Survey on nosql database. In *6th International Conference on Pervasive Computing and Applications (ICPCA)*, pages 363–366.

Hecht, R. and Jablonski, S. (2011). Nosql evaluation: A use case oriented survey. In *International Conference on Cloud and Service Computing (CSC)*, pages 336–341.

Indrawan-Santiago, M. (2012). Database research: Are we at a crossroad? In *Network-Based Information Systems (NBiS)*, NBIS '12, pages 45–51, Washington, DC, USA. IEEE Computer Society.

Pokorny, J. (2011). Nosql databases: A step to database scalability in web environment. In *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*, iiWAS '11, pages 278–283, New York, NY, USA. ACM.

Stonebraker, M. (2010). Sql databases v. nosql databases. *Communications of the ACM*, 53:10–11.

Tudorica, B. and Bucur, C. (2011). A comparison between several nosql databases with comments and notes. In *Roedunet International Conference (RoEduNet), 2011 10th*, pages 1–5.

Vaish, G. (2013). *Getting Started with NoSQL*. Packt Publishing, Birmingham B3 2PB, UK.

van der Veen, J. S., Waaij, B. V. D., and Meijer, R. J. (2012). Sensor data storage performance: Sql or nosql, physical or virtual. In *IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 431–438.