NoSqlayer: a Framework for Migrating Relational Datasets to NoSQL Models*

Fernando Vale[1], **Leonardo Rocha**[1]

[1] DCOMP/UFSJ - São João del-Rei, MG , Brasil

{fvale,lcrocha}@ufsj.edu.br

Abstract. *In software development, migration from a Data Base Management System (DBMS) to another with distinct characteristics, is a challenge for programmers and database administrators. Changes in the application code in order to comply with new DBMS are vast and may causing migrations infeasible. In this work we present NoSQLayer, a framework that supports migrating from relational to NoSQL DBMS. This framework is divided in two parts: (1) migration module, that perform the migration between DBMSs and (2) mapping module, that is a persistence layer to process database requests. Experiments show NoSQLayer as a handful solution suitable to handle large volume of data.*

Resumo. *A migração de um Sistema de Gerenciamento de Banco de Dados (SGBD) para outro com características distintas é um desafio para programadores e administradores de banco de dados. Adaptações no código da aplicação para utilizar o novo SGBD são muitas, podendo tornar a migração impraticável. Apresentamos neste trabalho o NoSQLayer, um framework capaz de migrar de SGBD relacional para outro NoSQL. Ele está dividido em duas partes: o (1) módulo de migração, que realiza automaticamente a migração dos dados e o (2) módulo de mapeamento, uma camada de persistência para processar as requisições de banco de dados. Experimentos demonstram o NoSQLayer é uma boa alternativa para cenários de grande volume de dados.*

1. Introdução

A nova geração de aplicações, construídas para atender desde demandas simples de pequenos grupos de usuários até grandes organizações, culminou com um enorme crescimento no volume de dados a serem processados e armazenados. Durante muitas décadas Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDR), baseados no Modelo Relacional, atenderam satisfatoriamente os requisitos das mais diversas aplicações propiciando simplicidade, robustez, desempenho e alta compatibilidade. Entretanto esses SGBDRs têm se mostrado ineficientes para manipular essa nova demanda de aplicações, focadas, em sua maioria, em grandes volumes de dados não estruturados. A partir desse cenário, surgiram os sistemas "Not only SQL" (NoSQL) [Padhy et al. 2011], utilizados por grandes empresas da área de TI, tais como Google, Facebook, Twitter e Amazon. Dentre as características do NoSQL podemos destacar a facilidade de particionamento e replicação de dados.

*Esse trabalho foi parcialmente financiado por CNPq, CAPES, FINEP, Fapemig, e INWEB.

Apesar das diversas qualidades mencionadas dos banco de dados NoSQL, a maior parte das aplicações de grandes organizações ainda são baseadas em SGBDRs. Existem diversos desafios relacionados à essa tarefa. O primeiro deles é o volume de dados a ser migrado, uma vez que a decisão de migração parte da constatação de que o SGBDR não vem atendendo as expectativas de desempenho para processar esses dados. Outro desafio é manter o novo banco semanticamente idêntico ao banco original, representando adequadamente todos os relacionamentos existentes sem que nenhuma informação seja perdida ou distorcida. Por fim, o terceiro desafio é o custo associado a adaptação das aplicações a esses novos modelos de dados para que as mesmas sejam capazes de comunicar adequadamente com o novo modelo adotado. Apesar de existirem na literatura algumas soluções que visam adaptar os modelos de dados relacional para essa nova realidade, bem como algumas propostas de migração automática, a todas elas existe um custo de adaptação das aplicações que não pode ser desprezado.

Dessa forma, apresentamos nesse trabalho o *NoSQLayer*, um framework capaz de realizar automaticamente a migração de dados entre banco de dados relacionais e NoSQL, mantendo a semântica do banco original. Além disso, nosso framework possui uma camada de abstração de dados que permite que as aplicações acessem os dados no modelo NoSQL de forma transparente, sem a necessidade de alteração das consultas já existentes. Atualmente, o *NoSQLayer* está adaptado para o SGBDR MySql[1] e para a abordagem NoSQL orientada a documentos, mais especificamente MongoDB[2], entretanto nosso objetivo é estender nosso framework para diversos outros SGBDRs e outras abordagens NoSQL.

Avaliamos nossa proposta sob duas perspectivas diferentes, uma qualitativa e uma quantitativa. Na avaliação qualitativa, utilizamos uma típica aplicação desenvolvida utilizando um modelo de dados relacional. Utilizando o NoSQLayer realizamos toda a migração dos dados do SGBDR original (MySql) para o MongoDB. Em seguida, comparamos os resultados de diversas operações SQL aplicadas sob ambos os bancos de dados, onde em 100% dos casos, os resultados foram idênticos. Na avaliação quantitativa, verificamos se o *overhead* gerado pela camada de abstração de dados era compensado pelo bom desempenho dos sistemas NoSQL. Para isso, comparamos os tempos de resposta para diversas operações, tanto utilizando o SGBDR original, quanto utilizando nosso framework, variando significativamente o volume de dados envolvido com a operação. Verificamos que esse *overhead* apenas impactava quando o volume de dados envolvido era pequeno. À medida que o volume de dados crescia, o *overhead* era compensado pela bom desempenho do sistema gerenciador NoSQL, mostrando que nossa solução, além de economicamente viável, também se mostrou computacionalmente eficiente.

Todo o trabalho apresentado nesse trabalho foi desenvolvido pelo aluno Fernando Vale, sob a coordenação do professor Leonardo Rocha. Toda a concepção da ideia, bem como todas as implementações associadas, foram desenvolvidas pelo aluno utilizando a infraestrutura disponível no Laboratório de Mineração de Dados de Alto Desempenho da UFSJ.

[1]https://www.mysql.com
[2]https://www.mongodb.org/

2. Trabalhos Relacionados

Comparações entre modelos de dados relacionais e modelos NoSQL já vem sendo apresentadas na literatura. Em [Diana and Gerosa 2010] os autores discutem algumas características que distinguem esses dois modelos, tais como o modo de armazenamento e a recuperação de informações por meio de consultas. Nos modelos relacionais os dados normalmente são armazenados em tabelas bem definidas por meio de um esquema, ao contrário do modelo NoSQL que não possui esquema pré-definido, fazendo com que os atributos de um registro não sejam necessariamente os mesmos. Os autores também realizam uma comparação entre os principais modelos de bancos de dados NoSQL, tais como Chave-Valor [Gilbert and Lynch 2002], utilizado nos sistemas gerenciadores Riak, Redis e Project Voldemort, orientado a documentos [Cattell 2011], utilizado pelo sistemas MongoDB, SimpleDB e CouchDB; e orientado a colunas [Vieira et al. 2012], onde se destaca o sistemas Google Big Table e Cassandra. Uma das principais constatações desses trabalhos é a dificuldade dos SGBRs em lidar com bancos de dados nos quais o volume de dados é grande.

Recentemente já observamos trabalhos que visam lidar com essa dificuldade dos SGBDRs. Em [Roijackers 2012], os autores argumentam que, dentro de uma mesma aplicação, existem certos conjunto de dados que podem ser melhor tratados utilizando SGBDRs e outros que são mais adequados para sistemas NoSQL. Seguindo essa argumentação, os autores sugerem uma solução na qual são mantidos dois modelos e todas as consultas passam por um módulo de avaliação o qual direciona as mesmas para o modelo que é considerado ideal para tal consulta. Nesse caso, diversas consultas NoSQL são criadas manualmente e armazenadas no banco de dados relacional para evitar que alterações sejam feitas na aplicação. Diferente desse trabalho, nossa abordagem mantem apenas a estrutura do banco de dados original e todos os dados são armazenados no banco NoSQL. As consultas oriundas das aplicações são traduzidas em tempo de execução para o banco NoSQL, sem a necessidade de nenhum trabalho manual por parte dos administradores e programadores.

Para contornar os problemas relacionados ao armazenamento de dados não estruturados, em [GAIOSO et al. 2007] os autores apresentam um framework capaz de integrar em uma mesma aplicação dados oriundos de diferentes fontes. Nessa proposta, os dados estruturados são mantidos em um banco relacional enquanto que os dados não estruturados são armazenados em diversos arquivos de diferentes formatos, tais como CSV, XML, etc. O framework possui um módulo intermediador que fica conectado na aplicação. Toda consulta realizada pela aplicação passa por esse intermediador que interpreta a consulta, busca os dados nas diferentes fontes e então retorna o resultado para a aplicação no formato desejado. Nosso framework foi baseado na ideia apresentada por esse artigo, porém todos os dados são armazenados no banco NoSQL. O *NoSQLayer* também possui um módulo intermediador que intercepta as operações SQL enviadas pela aplicação. Esse módulo realiza a tradução da operação, recupera os dados e os coloca no formato esperado e retorna para a aplicação de forma totalmente transparente. O principal desafio de nosso framework está relacionado à abstração completa do modelo relacional para o modelo NoSQL, visto que são completamente distintos em relação à sua estrutura. Na próxima seção detalhamos nossa proposta.

3. NoSQLayer

Nessa seção apresentamos em detalhes o desenvolvimento do *NoSQLayer*. Conforme mencionamos em seções anteriores, o framework é dividido em dois módulos principais: o módulo de migração de dados e o módulo de mapeamento de operações SQL. O funcionamento desses módulos serão apresentados nas duas subseções a seguir.

3.1. Módulo de migração dos dados

Esse módulo é responsável por identificar automaticamente todos os elementos que compõem o banco de dados original, tais como tabelas, atributos, relacionamentos, índices, etc., criar uma estrutura equivalente utilizando o modelo de dados NoSQL, e por fim, realizar a migração dos dados. Na Figura 1 apresentamos um diagrama que ilustra o funcionamento do módulo o qual é detalhado a seguir.

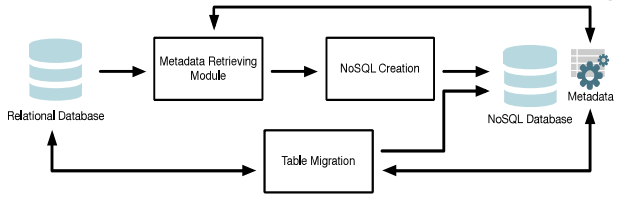


Figura 1. Funcionamento do Módulo de Migração dos Dados

Conforme mencionamos anteriormente, nessa primeira versão do *NoSQLayer* estamos focados no SGBDR MySQL e no sistema NoSQL MongoDB. O MySQL, assim como a maioria dos SGBDRs, possui um dicionário de dados contendo as informações necessárias para realizar a migração. A forma de obter essas informações varia significativamente entre cada SGBDR. Afim de tornar o *NoSQLayer* extensível a diferentes SGBDRs, para essa etapa optamos pela utilização da API *Java DatabaseMetaData* [DatabaseMetaData 2011], que consiste em diversas classes e métodos que facilitam a recuperação dessas metainformações. Essa API pode ser estanciada em diferentes SGBDRs, utilizando seus respectivos drivers. No caso do MySQL, utilizamos o MySQL Connector.

Identificados os elementos que compõe o banco de dados relacional, a próxima etapa consiste na criação automática de um novo esquema de dados apropriado para o NoSQL. O MongoDB representa os dados por meio de documentos, o que permite a representação de estruturas de dados mais complexas. No processo de construção desse novo esquema, seguimos o modelo de mapeamento apresentado em [Mapping 2013] no qual, para cada tabela do banco relacional é criada uma coleção no banco MongoDB. Os registros de cada tabela são recuperados e mapeados em documentos, sendo que cada atributo das tabelas é representado por um campo nestes documentos. Os relacionamentos existentes são representados por campos de referência entre documentos, de modo a facilitar o módulo de mapeamento nas operações que envolvem junções de tabelas. Os índices identificados também são setados no MongoDB. O *NoSQLayer* também cria uma coleção específica no MongoDB para armazenar todas as informações coletadas (nomes de tabelas, atributos, tipos dos atributos e restrições de integridade), denominada *Metadata*. É a partir dessa coleção que o módulo de mapeamento, descrito na seção seguinte, recupera as informações dos atributos envolvidos nas operações SQL para recompor a resposta que será retornado para a aplicação.

Por fim, a última etapa é a migração dos dados propriamente dita. Essa etapa consiste no mapeamento de consultas completas realizadas sobre cada uma das tabelas do banco de dados relacional armazenadas no MySql (*select * from table*) em inserções de dados nas coleções correspondentes no banco de dados NoSQL MongoDB. Todo o módulo foi construído utilizando a linguagem Java, utilizando os conceitos de programação orientada a objetos para que a inclusão de novos sistemas gerenciadores seja feita pela criação de novos métodos, tornando o framework extensível.

3.2. Módulo de Mapeamento

Esse módulo consiste em uma camada de abstração entre a aplicação e o banco de dados migrado para o MongoDB, tornando transparente para aplicação a migração, evitando que a mesma necessite de qualquer alteração de seu código. Na Figura 2 ilustramos o funcionamento desse módulo.

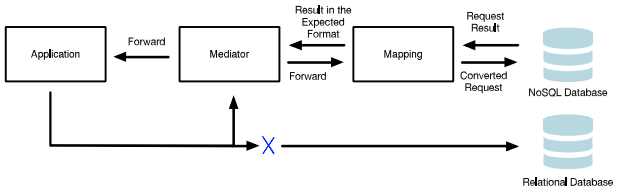


Figura 2. Funcionamento do Módulo de Mapeamento

A primeira tarefa desse módulo é interceptar as operações que são solicitadas pela aplicação ao SGBDR. Para essa tarefa criamos um submódulo, denominado Mediador, construído utilizando o MySQL Proxy, uma ferramenta Open Source que faz a comunicação entre o servidor MySQL e a aplicação cliente. Alteramos o código do MySQL Proxy para que as operações fossem interceptadas e encaminhadas para o segundo submódulo de conversão (Conveter) das consultas. O Converter foi desenvolvido sob a forma de um *WebService* que fica à espera de novas requisições. A comunicação entre o Mediador e o Converter é feita por meio de arquivos XML os quais contêm todas as informações relacionadas às operações. Abaixo apresentamos um exemplo desse XML de comunicação.

```
<?xml version='1.0' encoding='UTF-8'?>
<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'
        xmlns:ns1='http://queryInterceptor/'>
        <soap:Body>
                <ns1:Intercepta>
                        <query>SELECT id, name FROM users                    WHERE status='A' AND
(id>10 OR name='guest')
                        </query>
                        <tipoQuery>SELECT</tipoQuery>
                </ns1:Intercepta>
        </soap:Body>
</soap:Envelope>
```

Exemplo do XML de Comunicação entre Mediador e Converter

A primeira tarefa do Conveter é avaliar o parâmetro *tipoQuery* para identificar qual deverá ser o próximo procedimento a ser executado de acordo com cada tipo de operação (Select, Insert, Delete ou Update). Para cada operação existe um método correspondente. Todos esses métodos executam os mesmos passos descritos a seguir, com algumas peculiaridades relacionadas a cada uma das operações:

- **Extração de informações sobre a consulta**: nesse passo avalia-se o parâmetro *query* no intuito de identificar as informações relacionadas a operação SQL, tais como tabelas, atributos envolvidos na operação, além dos critérios utilizados em cláusulas *where*. Para isso, utilizamos a

biblioteca Java JSQLParser. A diferença entre os método nesse passo são as informações que cada um irá trabalhar. Por exemplo, enquanto o método correspondente a operação *Select* precisa tratar as tabelas envolvidas, os *joins* existentes, comandos de ordenação e agrupamento, uma operação *Insert* precisa tratar apenas os atributos e tabelas envolvidas nas operações.

- **Geração e execução da operação equivalente no NoSQL**: Extraídas as informações necessárias para cada tipo de operação, cada um dos métodos correspondentes constroem a operação NoSQL equivalente. Esse processo de construção é baseado no modelo oficial do MongoDB para mapeamentos de operações SQL [Mapping 2013]. Nesse passo, a coleção *Metadata* é muito utilizada. Por fim, executa-se essa nova operação sobre o MongoDB e os resultados são tratados no próximo passo.
- **Mapeamento dos resultados de retorno**: Os resultados retornados pelo MongoDB devem ser enviados ao Mediador, que repassa as informações à aplicação. Nesse passo, primeiramente remonta-se o cabeçalho do resultado, o qual deve conter as identificações corretas de tabelas, atributos e demais elementos relacionados ao resultado. Mais uma vez, a coleção *Metadata* é utilizada nesse passo. Por fim, cada registro retornado pelo MongoDB é mapeado seguindo o cabeçalho construído. A partir dessas informações é construído um XML que é enviado ao Mediador. No caso de operações que envolvam alterações nos dados do banco (*Delete, Update e Insert*), esse XML retorna o total de registros afetados pela operação. Abaixo apresentamos um exemplo desse XML que é enviado ao Mediador.

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
        <S:Body>
                <ns2:interceptaResponse xmlns:ns2="http://queryInterceptor/">
                        <return>header = {'id','name'}</return>
                        <return>
                                result_set = {{'12', 'John'},
                                 {'15', 'Steve'},
                                 {'2','guest'}}
                        </return>
                </ns2:interceptaResponse>
        </S:Body>
</S:Envelope>
```

Exemplo do XML de Comunicação entre Converter e Mediador

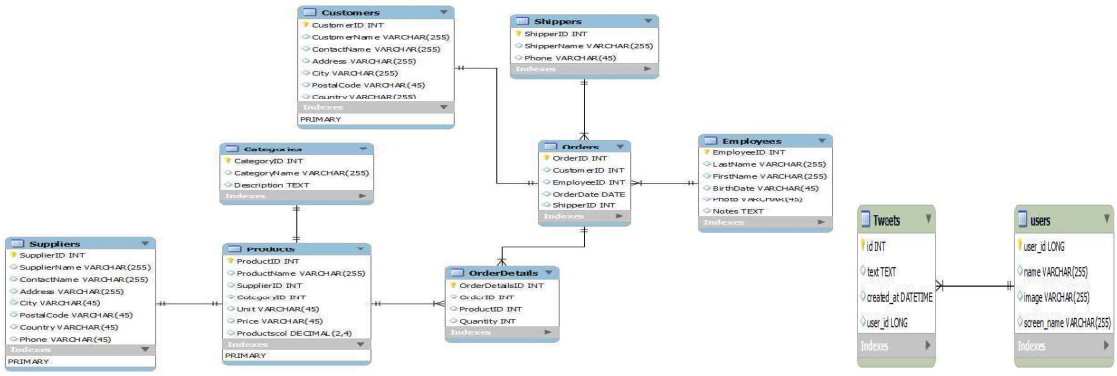
Um detalhe importante de se mencionar é com relação as operações aninhadas, por exemplo um *Delete* no qual os itens a serem removidos dependem de um operação de *Select*. Primeiramente o método correspondente à operação mais externa, é acionado. Durante a extração das informações da operação, ao se identificar uma operação aninhada, é acionado o método correspondente a essa operação para que a mesma seja resolvida, e então retorne as informações necessárias para executar a operação mais externa. Retomando o exemplo do *Delete*, o método correspondente a tratar tal operação identifica a operação ıSelect e o método responsável pelo *Select* é acionado. O retorno do *Select* é então utilizado pelo método do *Delete*. Por meio dessa estratégia, o *NoSQLayer* permite que existam diversas chamadas recursivas (por exemplo, várias operações de *Select* aninhadas), no qual a limitação dessas chamadas é definida apenas pelo hardware utilizado. Por fim, o XML enviado pelo Converter é tratado pelo Mediador, que é quem trata esse XML e transforma as informações contidas no mesmo no formato esperado pelo MySQL, o *MySQL Resource*, novamente utilizando os recursos que estão disponíveis no MySQLProxy. Na próxima seção apresentamos o conjunto de experimentos realizados no intuito de avaliar nossa proposta.

4. Avaliação Experimental

Nessa seção apresentamos os resultados relacionadas a avaliação experimental do NoSQLayer. Nossa avaliação está focada em dois tipos de análises, uma quantitativa e outra qualitativa, que serão descritas nas sessões seguintes. Primeiramente apresentamos o ambiente experimental.

4.1. Ambiente experimental

Em todas as nossas análises utilizamos basicamente dois banco de dados relacionais, conforme apresentando na Figure 3. O primeiro deles é um banco de dados fornecido pela W3Schools[3]. Trata-se de um banco de dados que consegue capturar bem diversas das características típicas utilizadas em grande parte das aplicações, tais como restrições de integridade, diversos relacionamentos (binário, ternário, etc.), vários tipos de dados, etc. O segundo banco de dados possui uma modelagem bem simples, relacionada a uma aplicação que coleta e armazena posts de usuários na rede social Twitter. Uma característica importante desse banco é o elevado volume de dados armazenados.



(a) W3schools (b) Twitter

Figura 3. Modelos do Banco de Dados.

Ambos os bancos de dados foram implementados utilizando o SGBDR MySql e para cada um deles foi criada uma aplicação escrita em Java responsável por realizar diversas operações SQL. Denominamos essa composição de Cenário MySql. Para cada configuração aplicação/banco de dados, criamos também uma versão a qual utilizou nosso framework para realizar toda a migração da estrutura do banco de dados, bem como os dados propriamente ditos, para o MongoDB. Além disso, para esse cenário, o qual denominamos Cenário NoSQLayer, o *NoSQLayer* também foi utilizado para realizar a captura e mapeamentos necessários entre as operações realizadas pelas aplicações (direcionadas ao MySql) e o MongoDB. Diversas operações SQL foram avaliadas e testadas, porém por uma questão de restrição de espaço, focaremos nesse artigo apenas na avaliação das operações apresentadas Tabela 1. Nessas operações procuramos utilizar operações que utilizavam variados operadores.

4.2. Análise Qualitativa

O objetivo da análise qualitativa é avaliar se a utilização do framework não afeta o funcionamento das aplicações no que refere às diversas operações SQL executadas, ou seja, avaliar a efetividade do *NoSQLayer*. Em nossos experimentos, para cada

[3]http://www.w3schools.com/

Identificador	Coleção	Operação
Select1	Twitter App.	*SELECT created_at,user_id FROM collections WHERE user_id IN (831178813,14152035,341925705)*
Select2	Twitter App.	*SELECT COUNT(*) AS NumberOfTweets FROM collections*
Select3	W3Schools App.	*SELECT Customers.CustomerName, Orders.OrderID FROM Customers LEFT JOIN Orders ON Customers.CustomerID=Orders.CustomerID*
Select4	Twitter App.	*SELECT MAX(user_id) AS HighestID, MIN(user_id) AS SmallestUserID FROM collections*
Select5	W3Schools App.	*SELECT * FROM Customers WHERE Country='Germany' AND (City='Berlin' OR City='München')*
Select6	W3Schools App.	SELECT * FROM Customers ORDER BY Country DESC
Select7	W3Schools App.	*SELECT * FROM Employees WHERE FirstName LIKE '%a'*
Select8	W3Schools App.	*SELECT Orders.OrderID, Employees.FirstName FROM Orders RIGHT JOIN Employees ON Orders.EmployeeID=Employees.EmployeeID*
Delete1	W3Schools App.	*DELETE FROM Categories WHERE CategoryID >= 1 AND CategoryID <= end_value*
Delete2	W3Schools App.	*DELETE FROM Customers WHERE CustomerID >=1 AND CustomerID <=end_value*
Insert1	W3Schools App.	*INSERT INTO OrderDetails*
Insert2	W3Schools App.	*INSERT INTO Customers*
Update1	W3Schools App.	*UPDATE Categories SET CategoryName='NewCategoryName', Description='NewDescription' WHERE CategoryID >= 1 AND CategoryID <= end_value*
Update2	W3Schools App.	*UPDATE Customers SET CustomerName='NewCustomerName', ContactName='NewContactName', Address='NewAddress', City='NewCity', PostalCode='NewPostalCode', Country='NewCountry' WHERE CustomerID >=1 AND CustomerID <= end_value*

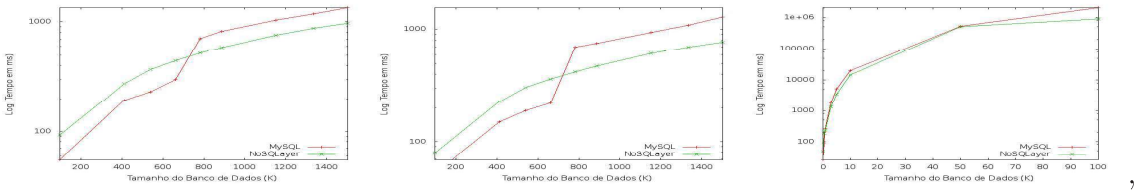
Tabela 1. Operações Avaliadas na Experimentação

cenário criado, as aplicações eram executadas e o resultado final de cada operação SQL era avaliado. No caso das operações de consulta, a cada execução de uma operação em ambos os cenários, os resultados das consultas eram comparados. Para as operações que envolviam alteração no estado do banco, ao final da execução de cada operação, uma operação de consulta completa nas tabelas envolvidas na operação e os resultados dessas consultas para cada cenário eram comparadas. No intuito de realizar uma avaliação bem completa, em ambos os casos, variamos significativamente o número de registros selecionados/afetados pelas operações executadas. Avaliando os resultados de retorno de cada operação, constatamos que não haviam diferenças para todos os casos avaliados, significando que nenhuma anomalia havia ocorrido e que, portanto, o NoSQLayer havia funcionado corretamente.

4.3. Análise Quantitativa

O foco dessa análise é avaliar a eficiência de nossa proposta, mais especificamente, verificar se mesmo com o *overhead* gerado ao se utilizar o *NoSQLayer* como uma camada entre a aplicação e o banco de dados NoSQL, o uso desses sistemas gerenciadores ainda é uma opção é vantajosa em termos de desempenho se comparado com os SGBDRs. Dessa forma, realizamos um conjunto de experimentos nos quais medimos o tempo de execução de diversas operações SQL, considerando cada um dos cenários previamente mencionados (Cenário MySql e Cenário NoSQL). Novamente as operações consideradas foram aquelas apresentadas na Tabela 1. O volume de dados envolvidos com cada operação foi variado. No caso das operações de consulta, antes de cada consulta, realizamos diversas inserções de dados para que os mesmos fossem recuperados pela consulta (esse tempo de inserção não era considerado). Essa mesma estratégia foi utilizada para as operações de Update e Delete, nesse caso, alterando o volume de dados afetados por essas operações. Por fim, para a operação de Insert, variamos o volume de dados que era enviado para inserção. O tempo era medido entre o momento em que a aplicação realizava a requisição e o momento em que os resultados da requisição eram retornados para a aplicação, considerando ambos os cenários. Cada operação era executada 10 vezes e o tempo final de cada uma delas foi dado em função da média dessas 10 execuções. Todos esses experimentos foram executados utilizando um máquina equipada com um processador Intel Core i3 de 2.53GHz, 4GB de memória RAM, 500GB de disco utilizando o sistema operacional Ubuntu 13.04 64 bits. Abaixo detalhamos os resultados alcançados agrupando os mesmos de acordo com cada tipo de operação. Utilizamos a escala log no eixo Y, refe-

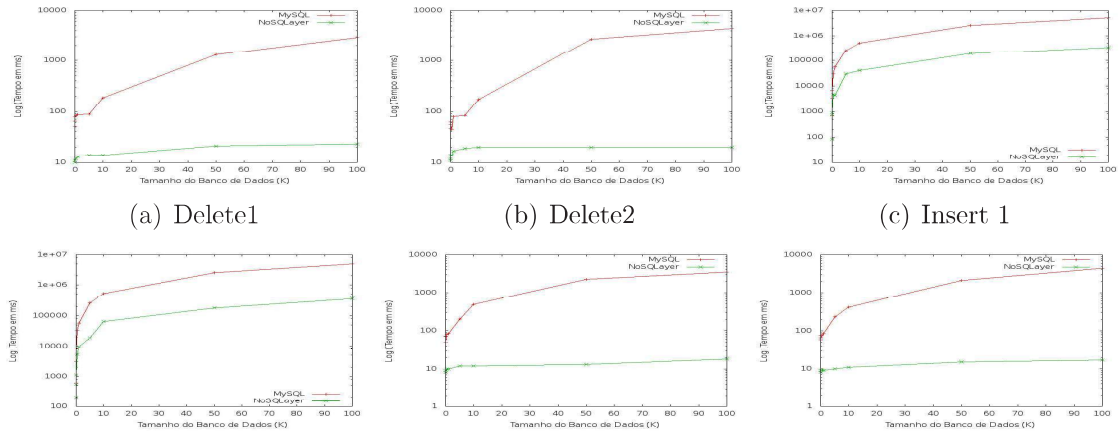
rentes ao tempo de execução, para uma para uma melhor visualização dos resultados em função da grande diferença entre os tempos relacionados ao MySql e ao NoSQL.



(a) Select1 (b) Select2 (c) Select3

Figura 4. Tempo de Execução de Operações Select.

Nos gráficos da figura 4 apresentamos os resultados relacionados ao tempo de execução das operações Select. Por uma limitação de espaço, apresentamos apenas os gráficos relacionadas às 3 primeiras operações listadas na tabela 1. Como podemos perceber, o MySql apresenta um tempo de resposta menor se comparado ao *NoSQLayer*, apesar de muito próximos. A medida que o volume de dados aumenta, o tempo de resposta também cresce em ambos cenários. Porém, a partir de um certo ponto, o tempo de resposta do MySql apresenta abruptamente um aumento no tempo de execução, e o mesmo não ocorre com *NoSQLayer*. Esse resultado mostra que, mesmo com o overhead gerado pela camada de abstração de dados, nosso framework é mais eficiente que um banco de dados relacional quando o volume de dados envolvidos cresce muito, conforme sua proposta original.



(a) Delete1 (b) Delete2 (c) Insert 1

(d) Insert 2 (e) Update 1 (f) Update 2

Figura 5. Tempo de Execução de Operações Delete. Insert e Update.

Por fim, analisando os gráficos relacionados às operações de alteração de estado do banco de dados (Delete, Insert e Update) apresentados na figura 5 temos que o tempo de execução do *NoSQLayer* foi muito inferior aos tempos relacionados ao MySQL, para todas as operações avaliadas. A principal explicação para esses resultados está na ausência de restrições de integridade dos bancos de dados NoSQL e, mesmo com o *overhead* gerado pela camada de abstração, o *NoSQLayer* é beneficiado por essa característica dos bancos NoSQL. Assim, um ponto que ainda precisa ser melhorado seria tratar e verificar internamente essas restrições. Atualmente, por meio da coleção *Metadata*, verificamos apenas a restrição de integridade.

5. Conclusões e Trabalhos Futuros

Nesse trabalho propomos o *NoSQLayer*, um framework capaz de realizar automaticamente a migração de dados entre banco de dados relacionais e NoSQL, mantendo

a semântica do banco original. Esse framework também possui uma camada de abstração de dados que permite que as aplicações acessem os dados no modelo NoSQL de forma transparente, sem a necessidade de alteração das consultas já existentes. Nessa primeira versão, nosso framework está habilitado para tratar da migração entre dois sistemas gerenciadores, o MySQL (relacional) e o MongoDB (NoSQL). Todo o framework foi implementado em Java, utilizando os conceitos de programação orientada a objetos, com o objetivo de torná-lo facilmente extensível a outros sistemas gerenciadores. Avaliamos nossa proposta sob duas perspectivas, uma qualitativa e outra quantitativa. Na avaliação qualitativa comparamos os resultados de diversas operações SQL aplicadas diretamente ao MySQL e aplicadas ao MongoDB passando por nosso framework. Em todos as consultas avaliadas os resultados foram idênticos, demonstrando assim o efetividade de nossa proposta. Na avaliação quantitativa comparamos o tempo de execução de diferentes consultas, variando significativamente o total de registros envolvidos em cada operação. Em nossos resultados constatamos que, apesar do *overhead* gerado pela camada de abstração (módulo de mapeamento) tornar o framework um pouco menos eficiente para pequenos volumes de dados, à medida que o volume de dados crescia, nosso framework se mostrou bem mais eficiente do que o uso apenas do MySQL. Como trabalhos futuros, nosso objetivo é estender o *NoSQLayer* para outros sistemas gerenciadores.

Referências

Cattell, R. (2011). Scalable sql and nosql data stores. *SIGMOD Rec.*, 39(4):12–27.

DatabaseMetaData, A. (2011). Javadoc databasemetadata. `http://docs.oracle.com/javase/6/docs/api/java/sql/DatabaseMetaData.html`. Data de acesso: 20/08/2013.

Diana, M. D. and Gerosa, M. A. (2010). Nosql na web 2.0: Um estudo comparativo de bancos não-relacionais para armazenamento de dados na web 2.0.

GAIOSO, R., LUCENA, F., and SILVA, J. (2007). Integrate: Infra-estrutura para integração de fontes de dados heterogêneas.

Gilbert, S. and Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59.

Mapping (2013). Sql to mongodb mapping chart. `http://docs.mongodb.org/manual/reference/sql-comparison`. Data de acesso: 10/09/2013.

Padhy, R. P., Patra, M. R., and Satapathy, S. C. (2011). Rdbms to nosql: Reviewing some next-generation non-relational database's. *International Journal of Advanced Engineering Science and Technologies*, 11(1).

Roijackers, J. (2012). Bridging sql and nosql. *Master Thesis, Eindhoven University of Technology. Department of Mathematics and Computer Science.*

Vieira, M. R., de Figueiredo, J. M., L., G., and Viebrantz, A. F. M. (2012). Bancos de dados nosql: Conceitos, ferramentas, linguagens e estudos de casos no contexto de big data.