Comparação de Performance entre PostgreSQL e MongoDB

Cristiano Politowski[1], **Vinícius Maran**[1]

[1] DCEEng – Departamento de Ciências Exatas e Engenharias - Universidade Regional do Noroeste do Estado do Rio Grande do Sul (UNIJUÍ) – RS 344 s/n, Santa Rosa - RS

crispolitowski@gmail.com, vinicius.maran@unijui.edu.br

Abstract. *From traditional DBMS solutions involving the storage of documents, graphs and even key-value structures, models of databases are created with the passage of time, and made the term NoSQL arise. However, doubts permeate this scenario, for example, the performance of these new technologies with respect to relational databases. The main aim of this work is perform a comparison between the database MongoDB document-oriented and relational database PostgreSQL, based on the runtime of read and write operations. As a result we realized a big difference in performance between the tested data bases, especially in search operations.*

Resumo. *Dos SGBDs tradicionais à soluções envolvendo o armazenamento de documentos, grafos e até estruturas chave-valor, modelos de bancos de dados são criados com o passar do tempo, e fizeram surgir o termo NoSQL. No entanto, algumas dúvidas permeiam este cenário, como, por exemplo, a performance de novas tecnologias em relação aos bancos de dados relacionais. O principal objetivo deste trabalho é realizar uma comparação entre o banco de dados orientado a documentos MongoDB e o banco de dados relacional PostgreSQL, baseado no tempo de execução de operações de leitura e escrita. Como resultado percebeu-se uma grande diferença de performance entre os bancos testados, principalmente nas operações de busca.*

1. Introdução

O mundo nunca trabalhou com volumes de dados tão grandes [Diana and Gerosa 2010], e as grandes aplicações Web são as grandes responsáveis. O alto tráfego de dados exige que empresas procurem soluções para poder suprir a demanda cada vez maior de performance e disponibilidade que fazem com que outros requisitos até então indiscutíveis, como consistência de dados, sejam revistos [Vogels 2009, Pritchett 2008].

Bancos de dados Relacionais são os mais utilizados para armazenamento de dados a décadas nos permitindo gravar grandes porções de dados no disco e provendo uma maneira simples de coletar estes dados através de *queries* SQL. Esta, uma linguagem padrão (na maioria dos Sistemas de Gerenciamento de Banco de Dados (SGBDs)) de interação com o banco de dados que mantém todos familiarizados com sua sintaxe, evitando problemas relacionados ao *language cacophony problem*[1].

[1]Cacofonia de Linguagem: consiste na preocupação de que se aprender uma linguagem é difícil, ao usar várias se tornaria ainda mais [Fowler 2010]

Outro fator importante do sucesso dos Banco de Dados Relacionais (BDRs) é a integração de aplicações usando a mesma base de dados. Isso assegura a consistência dos dados e os mantém sempre atualizados. Esta característica só é possível devido ao controle de concorrência provido pelos SGBDs, como, por exemplo, o uso de transações. No entanto, tecnologias emergentes, recentemente chamadas de movimento NoSQL (*Not Only SQL*), surgiram como um nova classe de SGBDs, que utilizam outros modelos, diferentes do relacional e que vem evoluindo trazendo um paradigma diferente de armazenamento de dados, como, por exemplo, armazenamento de documentos sem esquema definido.

Por se tratar de um conjunto de tecnologias e modelos relativamente novos, os bancos de dados NoSQL são vistos com desconfiança por parte da comunidade e descrença por parte de outros. Apesar de grandes corporações como Google e Twitter terem adotado a solução, usá-la em produção é sempre um risco, visto a sua prematuridade em relação aos BDRs.

Ainda existem dúvidas em relação ao potencial da aplicação da tecnologia NoSQL em algumas áreas. Portanto o principal objetivo deste artigo é realizar uma comparação de desempenho de consultas entre dois bancos de dados gratuitos e de grande adoção por parte dos usuários. Para realizar este comparativo, foram escolhidas as ferramentas PostgreSQL [PostgreSQL 2014a] e MongoDB [MongoDB 2014a]. Através da implementação de um ambiente de testes, foi possível realizar a comparação, analisar os resultados e tirar conclusões sobre o desempenho das duas ferramentas no cenário utilizado.

O artigo está estruturado como descrito a seguir. Na Seção 2 são apresentados os trabalhos relacionados e seus resultados. Na Seção 3 é apresentado um estudo sobre a área de banco de dados NoSQL. Na Seção 4, são apresentadas as características dos bancos de dados testados. A Seção 5 apresenta a etapa de preparação do ambiente para a execução dos testes. Na Seção 6 são apresentados os resultados dos testes. Por fim, a Seção 7 apresenta as conclusões finais deste trabalho.

2. Trabalhos Relacionados

Um dos aspectos que motivaram a realização deste trabalho é a pequena quantidade de trabalhos relacionados. Boicea [Boicea et al. 2012] faz uma comparação de performance entre MongoDB e Oracle [Oracle 2014], detalhando suas diferenças e fazendo testes de inserção, remoção e atualização de dados. Como resultado houve uma grande diferença de desempenho, em favor do banco de dados MongoDB, principalmente quando o número de registros no banco de dados é grande.

Li [Li and Manoharan 2013], por sua vez, faz testes de leitura, escrita, remoção e instanciação de vários bancos NoSQL e também do banco relacional MS SQL Server [Microsoft 2014]. Como resultado, foi possível afirmar que os bancos NoSQL obtiveram melhores resultados, porém, variando muito de acordo com a ferramenta utilizada.

Parker [Parker et al. 2013] também compara MongoDB com MS SQL Server. Os resultados são semelhantes para uma estrutura de base de dados modesta. MongoDB obteve resultados um pouco melhores em todas as comparações exceto quando foram utilizadas funções agregadas.

Há trabalhos relacionados ao uso dos bancos de dados MongoDB e Postgresql [Carniel et al. 2012b, Carniel et al. 2012a, Litt et al.], porém, estes trabalhos não fazem uma comparação direta entre os bancos de dados, que é o objetivo deste trabalho.

3. NoSQL - Not Only SQL

Bancos de Dados Relacionais foram projetados para serem executados em uma máquina apenas, onde para escalar, é necessário comprar uma máquina melhor (ou fazer um *upgrade*) com mais recursos, necessários para lidar com o alto tráfego de dados na web (*big data*). Esta técnica é chamada de escalabilidade vertical. Porém percebe-se que essa técnica, além de economicamente inviável, pois o custo de se melhorar um hardware é muito alto, é também limitada, pois chegará um momento em que o hardware não poderá ser melhorado. Em contrapartida, quando a escalabilidade vertical não é mais possível, técnicas de escalabilidade horizontal, através do uso de clusters de máquinas, podem ser úteis.

O ascensão dos *Web Services* proveram uma alternativa efetiva aos bancos de dados compartilhados para a integração de aplicativos, tornando fácil para diferentes aplicações escolherem seu próprio meio armazenamento de dados [Fowler 2012]. Pode-se dizer então que os precursores do movimento NoSQL foram empresas que, dada esta necessidade, buscaram novas soluções como o Bigtable [Bigtable 2014] e o Dynamo [Dynamo 2014].

NoSQL não possui um significado padrão, segundo Fowler [Fowler 2012] NoSQL pode ser traduzido em um conjunto de características, apresentadas a seguir:

- Não usa modelo de dados relacional e portanto não usa a linguagem SQL;
- Costuma ser projetado para ser executado em um *cluster*;
- Costuma ser *Open-Source*;
- Não possui esquema fixo (*SCHEMA-LESS*), permitindo gravar qualquer dado em qualquer estrutura.

4. Bancos de Dados Utilizados no Comparativo

4.1. MongoDB

MongoDB (de "*hu**mongo**us*") é um banco de dados *schema-less*, orientado a documentos, *open-source* escrito em C++ [MongoDB 2014a].

MongoDB persiste documentos no formato BSON (*Binary JSON*), que são objetos JSON binários. BSON, por sua vez, suporta estruturas como *arrays* e *embedded objects* assim como JSON. MongoDB permite usuários realizem modificações de apenas um atributo em um documento sem a necessidade de interação com o restante da estrutura.

Documentos podem ser armazenados em coleções (*collections*), onde serão efetuadas operações de busca (*queries*) e indexação (*indexing*). *Queries* são expressadas na sintaxe JSON e enviadas ao MongoDB como objetos BSON pelo *driver* de conexão ao banco.

Para persistência, MongoDB usa arquivos mapeados em memória, deixando o gerenciador de memória virtual do sistema operacional decidir quais partes vão para o

disco e quais ficam na memória. Isto faz com que o MongoDB não tenha controle sobre o momento onde os dados são escritos no disco [Matthes and Orend 2010].

MongoDB não possui controle de concorrência e gerenciamento de transações. Então, se um usuário lê um documento, escreve uma modificação e devolve ao banco de dados, pode acontecer de outro usuário escrever uma nova versão do documento entre o processo de leitura e escrita do primeiro.

4.2. PostgreSQL

O PostgreSQL é um sistema de gerenciamento de banco de dados objeto-relacional (SGBDOR)[2] baseado no POSTGRES Versão 4.2, desenvolvido pelo Departamento de Ciência da Computação da Universidade da Califórnia em Berkeley. É totalmente compatível com ACID[3], tem suporte completo a chaves estrangeiras, junções, visões e gatilhos. Inclui a maior parte dos tipos de dados do ISO SQL:1999, incluindo INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, e TIMESTAMP. Suporta também o armazenamento de objetos binários, incluindo figuras, sons ou vídeos.

Possui controle de concorrência multiversionado (MVCC - *Multi-Version Concurrency Control*), recuperação em um ponto no tempo (PITR - *Point in Time Recovery*), *tablespaces*, replicação assíncrona, transações agrupadas (*savepoints*), cópias de segurança (*online/hot backup*), um planejador de consultas e registrador de transações sequencial para tolerância a falhas.

5. Ambiente de testes

Os testes foram feitos em um computador com processador Intel Core i7-3630QM CPU de 2.4 GHz com 8 GB de RAM e SSD de 128 GB com o sistema operacional Linux **Ubuntu** [Ubuntu 2014] 13.04 de 64 bits. A versão do PostgreSQL usada foi a 9.1 a do MongoDB foi a 2.4.9.

Para implementar os testes, a linguagem **Python** [Python 2014] foi utilizada juntamente com os respectivos *drivers* necessários para a conexão com o banco. Para o PostgreSQL, foi utilizado o *driver* **psycopg** [Psycopg 2014] e para o MongoDB foi utilizado o *driver* **pymongo** [Pymongo 2014]. Para medir o tempo de execução, foi usado o módulo da linguagem Python **timeit** [Timeit 2014].

A estrutura do banco foi estabelecida através de um diagrama de classe, apresentado na Figura 1. Podemos entender esse diagrama como a modelagem de um *blog*. É difícil fazer uma comparação entre dois bancos de dados que armazenam estruturas de dados diferentes, por isto a mesma estrutura foi utilizada nas duas ferramentas durante os testes.

Os testes foram divididos em três categorias: **inserção**, **busca simples** e **busca complexa**. Cada teste foi executado em um *loop* de 1, 10, 100, 1.000, 10.000 e 100.000 vezes, repetidos três vezes sendo considerada a média aritmética dos mesmos.

[2]Um banco de dados objeto-relacional (ORD), ou sistema de gerenciamento de banco de dados objeto-relacional (ORDBMS ou SGBDOR), é um sistema de gerenciamento de banco de dados relacional que permite aos desenvolvedores integrar ao banco de dados seus próprios tipos de dado e métodos.

[3]Atomicidade, Consistência, Isolamento e Durabilidade

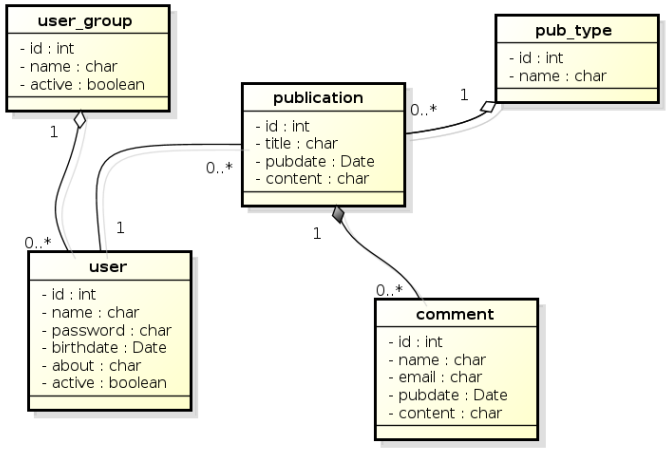

Figura 1. Diagrama de classe representando a estrutura de dados utilizada nos testes.

A **inserção** foi feita englobando todas as tabelas no PostgreSQL e um documento completo no MongoDB com todos os dados preenchidos. A **busca simples** consiste na busca de todas as publicações com o título `Teste publicacao`. Já na **busca complexa** o resultado deve trazer todas as publicações entre as datas `2013-1-1` e `2015-1-1`, que sejam do tipo `Novidade`, de usuários nascidos a partir de `1980-10-27`, que estejam `ativos`, pertencentes ao grupo de nome `Administracao` e que possua, pelo menos, `1` comentário. Tudo isso ordenado pela data de publicação. Todas retornaram 333.333 (trezentos e trinta e três mil, trezentos e trinta e três) registros, que foi a quantidade de inserções em ambos os bancos.

5.1. Operações com PostgreSQL

Para as operações no banco de dados PostgreSQL, o código foi estruturado da seguinte maneira:

```
conn = psycopg2.connect("dbname=testedb user=teste")
cur = conn.cursor()
# <QUERIES AQUI>
conn.commit()
cur.close()
conn.close()
```

Na primeira linha a conexão é feita através do *driver* `psycopg2`. Na linha dois um cursor é criado para a manipulação do banco. Em `<QUERIES AQUI>` são colocadas as consultas na linguagem SQL pura, sem *frameworks*. Todas as consultas são executadas e persistidas (`commit`) apenas uma vez dentro do *loop* de repetições. No final da execução, tanto o cursor quanto a conexão são fechados.

A **inserção** é feita com cinco inserções nas respectivas tabelas, usando a interpolação de strings do Python (`%s`) para passar os valores. Essa técnica é recomendada pela documentação oficial da linguagem por questões de performance. O trecho `datetime.datetime.utcnow()` é uma função Python que retorna a data atual do sistema. O *driver* faz a conversão automática de tipos. O trecho de código a seguir apresenta um exemplo de realização de um conjunto de inserções realizadas nos testes:

```
cur.execute("INSERT INTO testesc.usuario_grupo (nome, ativo) VALUES (%s, %s)",
```

```
  ("Administracao", True))
cur.execute("INSERT INTO testesc.usuario (nome, senha, datanascimento, sobre, ativo,
  id_grupo) VALUES (%s, %s, %s, %s, %s, %s)", ("Claudio Francisco da Silva", "admin123",
  "1987-10-27", "Pesquisador e programador nas horas vagas.", "true",0))
cur.execute("INSERT INTO testesc.pub_tipo (nome) VALUES (%s)", ("Novidade",))
cur.execute("INSERT INTO testesc.publicacao (titulo, datapub, conteudo, id_usuario,
  id_pub_tipo) VALUES (%s, %s, %s, %s, %s)", ("Publicacao teste",
  datetime.datetime.utcnow(), "Bla bla bla...", 0, 0))
cur.execute("INSERT INTO testesc.comentario (nome, email, datapub, conteudo,
  id_publicacao) VALUES (%s, %s, %s, %s, %s)", ("Fulano de Tal", "fulano@gmail.com",
  datetime.datetime.utcnow(), "First!", 0))
```

O trecho do código para a operação de **busca simples** usa apenas uma cláusula `WHERE` para achar a o título `'Publicacao teste'` através do `LIKE`. O trecho de código a seguir apresenta a realização desta busca nos testes:

```
cur.execute( "SELECT * FROM testesc.publicacao p WHERE p.titulo LIKE 'Publicacao teste'")
```

O trecho do código para a operação de **busca complexa** (apresentado abaixo) usa várias junções entre tabelas e inúmeras funções `AND` além de uma sub-consulta para retornar o número de comentários. Novamente foi usado interpolação de strings.

```
cur.execute( "SELECT * FROM testesc.publicacao p, testesc.usuario u,
    testesc.usuario_grupo ug, testesc.pub_tipo pt, testesc.comentario c
  WHERE p.id_usuario = u.id_usuario AND u.id_grupo = ug.id_usuario_grupo
  AND p.id_pub_tipo = pt.id_pub_tipo AND c.id_publicacao = p.id_publicacao
  AND p.datapub > %s AND p.datapub < %s  AND pt.nome LIKE %s
  AND u.datanascimento > %s AND u.ativo = %s
  AND ug.nome = %s AND (SELECT count(*) FROM testesc.publicacao p,
    testesc.comentario c WHERE c.id_publicacao = p.id_publicacao) >= 1
  ORDER BY p.datapub", (datetime.datetime(2013, 11, 12, 12),
    datetime.datetime(2015, 11, 12, 12), "Novidade", "1980-1-1", "True",
    "Administracao"))
```

5.2. Operações com MongoDB

Para as operações no banco de dados MongoDB, o código de conexão com o banco de dados foi estruturado da seguinte maneira:

```
client = MongoClient('localhost', 27017)
db = client.testedb
# <QUERIES AQUI>
client.close()
```

Na primeira linha, uma instância do MongoDB é criado. Após, é criado/recuperado o banco de dados de nome `testedb`. Assim como na estrutura do banco PostgreSQL, em `<QUERIES AQUI>` vão as respectivas consultas das operações. Ao final, a instância do MongoDB é fechada.

Para a **inserção**, um documento contendo todos os dados estruturados em formato de JSON válido foi criado e inserido através da função `insert()` dentro de uma coleção de documentos (`collection`) de nome `'publicacoes'`. O trecho de código referente a operação de inserção nos testes é apresentado abaixo:

```
publicacao = { "publicacao": { "titulo": "Publicacao teste",
      "datapub": datetime.datetime.utcnow(), "conteudo": "Bla bla bla...",
      "tipo": "Novidade", "usuario": { "nome": "Claudio Francisco da Silva",
  "senha": "admin123", "datanascimento": "1987-10-27",
  "sobre": "Pesquisador e programador nas horas vagas.", "ativo": "True",
  "grupo": { "nome": "Administracao", "ativo": "True" } },
      "comentario": [ { "nome": "Fulano de Tal", "email": "fulano@gmail.com",
      "datapub": datetime.datetime.utcnow(), "conteudo": "First!" } ]} }
publicacoes = db.publicacoes
publicacoes.insert(publicacao)
```

A **busca simples** (código apresentado abaixo) é feita através da função `find()` da coleção, passando um dicionário com as respectivas chaves e valores. Este dicionário de dados também deve ser um JSON válido.

```
publicacoes = db.publicacoes
publicacoes.find({"publicacao.titulo":"Publicacao teste"})
```

Para a **busca complexa** (código apresentado abaixo) utilizamos vários operadores de comparação da linguagem de consulta do MongoDB, como `$gt`, `$lt` e `$size`. Este último usado para o cálculo do número de comentários.

```
publicacoes = db.publicacoes
publicacoes.find({ "publicacao.datapub": {"$gt": datetime.datetime(2013, 11, 12, 12)},
  "publicacao.datapub": {"$lt": datetime.datetime(2015, 11, 12, 12)},
  "publicacao.tipo": "Novidade", "publicacao.usuario.datanascimento" : {"$gte": "1980-1-1"},
  "publicacao.usuario.ativo":"True", "publicacao.usuario.grupo.nome":"Administracao",
  "publicacao.comentario":{ "$size": 1 }}).sort("publicacao.datapub")
```

6. Resultados

A Tabela 1 apresenta os resultados dos testes feitos no banco PostgreSQL onde os valores estão representados em `segundos`. Nas operações de inserção o banco relacional obteve bons resultados, onde em `um centésimo de segundo` foi realizada a inserção de apenas um registro em todas as tabelas e em cerca de `18 minutos` foi realizada a mesma operação, mas com trezentos mil registros. Para uma busca simples os problemas começam a aparecer, chegando a um total de `11 horas e meia` para buscar todos os registros cem mil vezes. No entanto, a busca complexa, que possui *queries* mais elaboradas, teve um crescimento de 600% no tempo para recuperar apenas um registro e mais de `77 horas`, em média, para fazer essa operação cem mil vezes.

Tabela 1. Testes com o banco PostgreSQL.

Operações	Número de repetições					
	1	**10**	**100**	**1000**	**10000**	**100000**
Inserção	0,0096	0,1108	1,0792	11,0729	105,1221	1106,3959
B. simples	0,3902	3,9198	39,5233	451,3150	4286,2453	40892,9120
B. complexa	2,3452	24,4739	250,1055	2568,9904	26929,7949	278438,6800

A Tabela 2 apresenta os resultados dos testes feitos no banco MongoDB onde os valores também estão representados em `segundos`. Ao contrário do banco anterior, a tarefa que mais despende esforço, nesse caso, é a operação de inserção. Isto acontece por ser uma ação bloqueadora, diferente das buscas. Essas, por sua vez, mostram nos resultados obtidos o motivo do banco de dados MongoDB ser bem aceito na comunidade[4]. A primeira constatação é de que a diferença entre as duas buscas é inexpressiva. O outro detalhe, é a velocidade para a busca de grandes quantidades de dados, onde foi possível recuperar todos os dados, cem mil vezes, em pouco mais de `50 segundos`.

Para mostrar que a comparação entre os valores médios é significativa, foi utilizado o Teste T de Student que usa conceitos estatísticos para rejeitar ou não uma hipótese nula. Usando uma significância de 5%, achamos um **valor-p** sempre abaixo desse parâmetro, com exceção da operação de inserção de 1 registro / documento, mostrando a relevância na diferença dos tempos coletados. O resultado deste teste é apresentado na Tabela 3.

[4]Quinto lugar no ranking do site *DB-Engines* [DB-Engines 2014].

Tabela 2. Testes com o banco MongoDB.

Operações	Número de repetições					
	1	10	100	1000	10000	100000
Inserção	0,0046	0,0072	0,0781	0,7294	7,0441	77,7849
B. simples	0,0007	0,0051	0,0528	0,5159	5,0282	49,7979
B. complexa	0,0008	0,0085	0,0529	0,5068	5,1852	51,7124

Tabela 3. Teste T Student

Operações	Número de repetições					
	1	10	100	1000	10000	100000
Inserção	0,30366106	0,00043316	0,00000004	0,00016811	0,00038688	0,00000283
B. simples	0,00001081	0,00010017	0,00019375	0,00543608	0,00000190	0,00000259
B. complexa	0,00000193	0,00030572	0,00006304	0,00031423	0,00021793	0,00036125

Para melhor comparar os resultados, gráficos foram criados e os resultados dos testes foram separados por operação. Na Figura 2, os resultados da operação de inserção são comparados. Considerando que os resultados para o banco PostgreSQL na operação de inserção foram melhores em relação as outras operações, e para o banco MongoDB ocorre justamente o oposto, ainda assim, o banco sem esquema fixo é largamente superior.

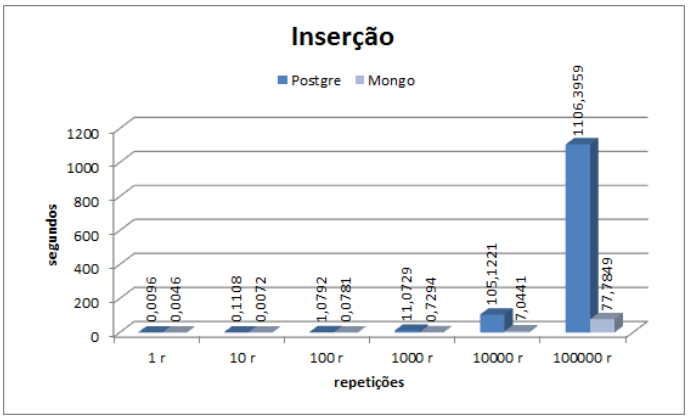

Figura 2. Comparação da Inserção: PostgreSQL e MongoDB

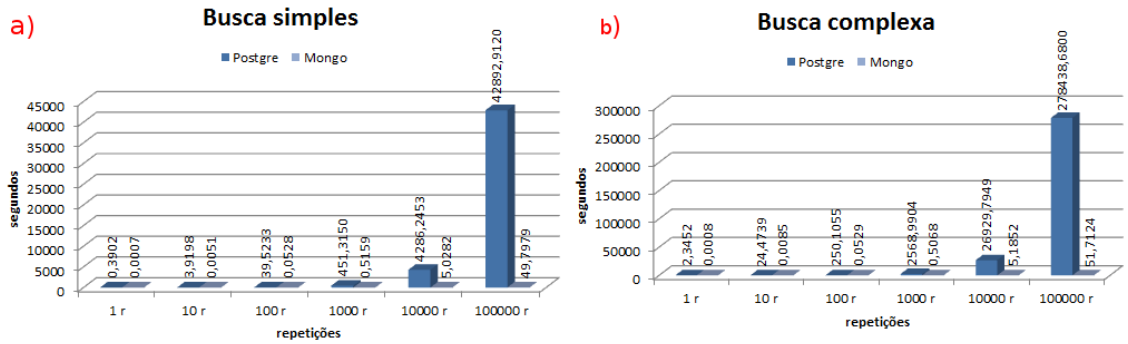

Figura 3. Comparação da Busca Simples (a) e Busca Complexa (b)

Na Figura 3 (buscas simples e complexas), a diferença é ainda maior. O banco MongoDB mostrou melhores resultados na velocidade das consultas de recuperação de

documentos. Já o PostgreSQL, apesar de ser possível otimizar as consultas utilizando um código SQL diferente, demonstrou resultados piores.

7. Conclusão

Utilizando os dois bancos de dados na configuração padrão, sem ajustes de otimização[5] e baseado no ambiente de testes proposto, podemos concluir que o MongoDB obteve melhores resultados. Isso se dá pelo fato de que os bancos NoSQL terem nascido para suprir a demanda por performance, deixando outros detalhes, como atomicidade, por exemplo, em segundo plano.

Bancos de dados NoSQL e Relacionais utilizam paradigmas diferentes e, por sua vez, possuem finalidades diferentes,mas com o mesmo propósito: persistir dados. Segundo os testes de performance, para uma aplicação com alta carga de consultas à base de dados, como serviços web, por exemplo, o banco MongoDB é uma ótima alternativa. Entretanto, se a aplicação necessita de uma camada de segurança mais robusta, com controle de acessos simultâneos à base de dados, o banco PostgreSQL é a melhor alternativa.

Para trabalhos futuros abre-se um grande leque de possibilidades que vem de encontro com este tema, como: abordar outros bancos Relacionais e NoSQL como MySQL, FireBird e CouchDB, incrementar as buscas utilizando mais critérios e tabelas, fazer testes de backup, replicação e concorrência.

Referências

[Bigtable 2014] Bigtable (2014). Bigtable web site. `http://research.google.com/archive/bigtable.html`.

[Boicea et al. 2012] Boicea, A., Radulescu, F., and Agapin, L. I. (2012). MongoDB vs Oracle – Database Comparison. *MongoDB vs Oracle - database comparison*, pages 330–335.

[Carniel et al. 2012a] Carniel, A., de Aguiar Sa, A., Brisighello, V., Ribeiro, M., Bueno, R., Ciferri, R., and de Aguiar Ciferri, C. (2012a). Análise Experimental de Bases de Dados Relacionais e NoSQL no Processamento de Consultas sobre Data Warehouse. d:113–120.

[Carniel et al. 2012b] Carniel, A., de Aguiar Sa, A., Brisighello, V., Ribeiro, M., Bueno, R., Ciferri, R., and de Aguiar Ciferri, C. (2012b). Query processing over data warehouse using relational databases and nosql. In *Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana En*, pages 1–9.

[DB-Engines 2014] DB-Engines (2014). Db-engines web site. `http://db-engines.com/en/ranking`.

[5]Há várias maneiras de otimizar consultas aos bancos. O MongoDB tem uma área exclusiva [MongoDB 2014b] onde são indicados métodos como o uso de *index* e *capped collections* entre outros recursos para a realização da otimização. Já o PostgreSQL possui um recurso chamado *Generic Query Optimizer* [PostgreSQL 2014b], que possibilita melhorar a performance de *queries* que possuem muitos *joins*. Entretanto, o objetivo do teste é realizar a comparação dos bancos de dados em seu estado inicial, sem modificações de configuração.

[Diana and Gerosa 2010] 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.

[Dynamo 2014] Dynamo (2014). Dynamo web site. http://aws.amazon.com/pt/dynamodb/.

[Fowler 2012] Fowler (2012). Martin Fowler's talk from the GOTO Aarhus Conference 2012. http://www.youtube.com/watch?v=qI_g07C_Q5I.

[Fowler 2010] Fowler, M. (2010). *Domain-Specific Languages*.

[Li and Manoharan 2013] Li, Y. and Manoharan, S. (2013). A performance comparison of SQL and NoSQL databases. pages 15–19.

[Litt et al.] Litt, G., Thompson, S., and Whittaker, J. Improving performance of schemaless document storage in PostgreSQL using BSON.

[Matthes and Orend 2010] Matthes, F. and Orend, K. (2010). Analysis and Classification of NoSQL Databases and Evaluation of their Ability to Replace and Object-relational Persistence Layer.

[Microsoft 2014] Microsoft (2014). Microsoft sql server web site. https://www.microsoft.com/en-us/sqlserver/default.aspx.

[MongoDB 2014a] MongoDB (2014a). Mongodb web site. http://mongodb.org.

[MongoDB 2014b] MongoDB, D. (2014b). Mongodb documentation web site. http://docs.mongodb.org/manual/administration/optimization/.

[Oracle 2014] Oracle (2014). Oracle web site. http://www.oracle.com/technetwork/indexes/downloads/index.html.

[Parker et al. 2013] Parker, Z., Poe, S., and Vrbsky, S. V. (2013). Comparing NoSQL MongoDB to an SQL DB. *Proceedings of the 51st ACM Southeast Conference on - ACMSE '13*, page 1.

[PostgreSQL 2014a] PostgreSQL (2014a). Postgresql web site. http://postgresql.org.

[PostgreSQL 2014b] PostgreSQL, D. (2014b). Postgresql documentation web site. http://www.postgresql.org/docs/9.1/static/geqo.html.

[Pritchett 2008] Pritchett, D. (2008). Base: An acid alternative. *Queue*, 6(3):48–55.

[Psycopg 2014] Psycopg (2014). Psycopg web site. http://initd.org/psycopg/docs/index.html.

[Pymongo 2014] Pymongo (2014). Pymongo web site. http://api.mongodb.org/python/current/index.html.

[Python 2014] Python (2014). Python web site. http://python.org.

[Timeit 2014] Timeit (2014). Timeit web site. http://docs.python.org/2/library/timeit.html.

[Ubuntu 2014] Ubuntu (2014). Ubuntu web site. http://www.ubuntu.com/.

[Vogels 2009] Vogels, W. (2009). Eventually consistent. *Commun. ACM*, 52(1):40–44.