**Avaliação de Performance do Neo4j e MongoDB no Contexto de Internet das Coisas**

**Hélder S. Faria1**

1Pontifícia Universidade Católica de Minas Gerais (PUCMINAS) Caixa Postal 30535-901 – Belo Horizonte – MG – Brazil

[heldersfaria@gmail.com](mailto:heldersfaria@gmail.com)

***Abstract.*** *New technologies have emerged as smart objects which have data processing, Internet access and other features. With these devices is inevitable the large volume of data and structured or unstructured data. As Relational Database have shown poor performance in this scenario, the alternative to this problem arose with NoSQL. This study evaluates the database performance Neo4J and MongoDB. In order to perform this study is carried out operations using a database similar to an Internet of Things scenario. The key contribution of this paper is guidance on how to evaluate and choose the DBMS for projects containing the Internet of Things scenario.*

***Resumo.*** *Novas tecnologias têm surgido como os objetos inteligentes que possuem processamento de dados, acesso a internet e outras funcionalidades. Com estes aparelhos é inevitável o grande volume de dados e dados estruturados ou não. Como Bancos Relacionais têm apresentado baixa performance a este cenário, a alternativa para este problema surgiu com os NoSql. Este trabalho avalia o desempenho do banco de dados Neo4J e MongoDB. Como forma de realizar este estudo é realizado operações utilizando uma base de dados semelhante a de um cenário de Internet das Coisas. A contribuição chave deste artigo é a orientação sobre como avaliar e escolher o SGBD para projetos contenham o cenário de Internet das Coisas.*

# Introdução

Sistemas de Gerenciamento de Base de Dados Relacionais (SGBDR) têm sido largamente utilizado nos últimos anos. Porém novas aplicações tem exigido cada vez mais alta disponibilidade, escalabilidade e alto desempenho (RITH et all, 2014).

Observando isto, Cuzzocrea, Song e Davis (2011) citam o termo *BigData c*om foco em alto desempenho e quantidade de dados, podendo tratar coleções de dados complexos, sendo estruturados ou não, e com uma vasta família de aplicações.

Cattel (2011) diz que banco de dados que tratam o termo *BigData* são chamados de *Not Only Sql* (*NoSQL*) e que a existência dos mesmos não seria para substituir os bancos relacionais, mas sim apenas para dar uma alternativa de solução aos problemas encontrados atualmente. Através de um teorema, o autor também cita que qualquer SGBD pode possuir apenas duas das três características consistência, disponibilidade e tolerância a partição.

Para possuir tais características, alguns sistemas de armazenamento visam relaxar as propriedades de Atomicidade, Consistência, Isolamento e Durabilidade (ACID) por não haver garantias de atingir os objetivos de desempenho e escalabilidade. Assim surge o conceito Basicamente Disponível, Estado flexível e Eventualmente Consistente (BASE) (CATTEL, 2011).

Tavares e outros (2016) associam o conceito *BigData* à *Internet of Things(IoT)*. Este último é também citado por Jenkins (2015) como uma tendência para que vários tipos de artefatos físicos que possuem processamento e conectividade a redes de comunicação possam ser capazes de trocar mensagens entre dispositivos.

*BigData* e IoT tem gerado grandes desafios de processamento de dados com grande volume, [heterogeneidade](https://www.google.com.br/search?q=heterogeneidade&spell=1&sa=X&ved=0ahUKEwjFvZCKoOPPAhWCj5AKHePGBq0QvwUIGygA), necessidade de escalabilidade e autonomia para objetos produtores e consumidores de informações (TAVARES et all, 2016).

Assim, este trabalho deseja avaliar as seguintes questões sobre os bancos de dados estudados:

- Qual teria o desempenho mais adequado para atender uma aplicação que esteja inserida em um cenário de internet das coisas visando apenas armazenar e recuperar todas as mensagens trocadas entre dispositivos?

- Qual é o desempenho de requisições de escritas e leituras neste contexto?

- Qual requisições que utilizam filtro tem desempenho parecido com requisições que não utilizam filtros?

- Qual o comportamento diante de um aumento da base de dados da base de dados?

Este artigo tem como objetivo principal avaliar cenários em que Banco de Dados *NoSql* possam ser usados simulando um ambiente de *IoT*. Durante os estudos os sistemas Neo4J e MongoDB foram avaliados. O primeiro tem o modelo de dados orientado a grafos e o segundo orientado documentos.

# Referencial Teórico

# 2.1. Internet das Coisas

Segundo Karakostas e Bessis (2016), objetos inteligentes têm sido propostos no contexto de Internet das Coisas para serem dispositivos equipados com capacidade de processamento e comunicação através da internet.

Tais aparelhos chamados de “coisas” podem ser objetos simples ou complexos. Não precisam estar conectados diretamente a internet, mas precisam ser conectáveis por uma rede qualquer (JAMMES, 2016).

Ha e outros (2015) classificam serviços de *IoT* em 3 domínios, baseando-se em três características: temporal, espacial e relacional.

Sobre a característica temporal o que é levado em conta é se dados são servidos em um repositório instantaneamente ou permanentemente. Se houver existência de limite por período na criação e processamento de dados é do tipo instantâneo, caso contrário é do tipo permanente.

Para a característica espacial há duas classificações: local e global. Um serviço é local quando há uma localização particular com uma existência de limite localizacional. Já um serviço é global quando pode ser criado e processado através de uma área inteira com nenhum ou grande limite de localização.

O critério relacional define os termos privado e público, caracterizando o serviço de *IoT* de acordo com a quantidade e relação dos usuários de um serviço. Uma relação privada teria como exemplo um serviço que uma família utiliza e uma relação pública haveria um compartilhamento de serviços com diversas pessoas.

Com estas características apresentadas é possível perceber a grande aplicabilidade de IoT em diversas áreas. Hefnawy e outros (2016) e CrabTree e Tolmie (2016) exploram várias áreas que podem possuir objetos inteligentes, como por exemplo: Energia, Vestuário, Comunicações, Transportes, Ambiental, Cobertura de Desastre, Agricultura, Saúde, Educação e outras.

# MongoDB

MongoDB é um banco de dados *NoSQL* implementado em C++ pela empresa 10gen em 2007 e com lançamento público em 2009, atualmente está disponível em (<http://www.mongodb.org/downloads>) (ABRAMOVA et all, 2013).

[Chodorow](http://shop.oreilly.com/product/0636920028031.do#tab_04_2) (2013) diz que esse banco possui um modelo de dados flexível orientado a documentos. É possível trabalhar com dados de propósito geral, possuir escalabilidade horizontal, replicação de dados e *sharding* de seus dados.

Marchioni (2015) descreve o modelo de dados em três principais estruturas: base de dados, coleções e documentos. A base de dados contém um conjunto de coleções. Já as coleções contêm um conjunto de documentos, sendo que seu nome é o que diferencia de outra coleção na base de dados.

Como unidade básica, existe os documentos que tem um esquema dinâmico ao logo de sua vida, ou seja, pode ter mais ou menos campos, além de poder suportar diferentes tipos de dados. Figura 01 ilustra este modelo de dados.

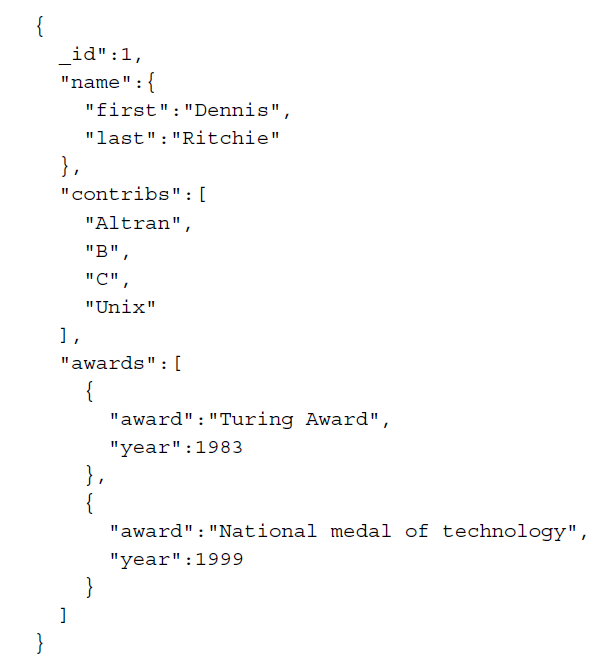
**Figura 01- Ilustração de um modelo de dados orientado a documentos**



Fonte: Marchioni (2015)

Todo documento trabalha com um formato de arquivo chamado de JSON (JavaScript Object Notation), e seus campos são do formato chave/valor e com os tipos de dados suportados. Porém um JSON é persistido em formato de BSON (Binary JSON) que é um documento codificado em binário, conforme ilustrado na Figura 02.

**Figura 02- Exemplo de um documento *JSON***



Fonte: Marchioni (2015)

No documento representado na Figura 02 há um pouco dos diversos tipos suportados. Um deles é o *ObjectId* que identifica o documento e por isso não pode ser repetido. Há também documento embutido e um vetor de *Strings* e documentos.

Um *cluster* do MongoDB é formado de um nó mestre e um ou mais nós escravos. Caso ocorra uma falha no nó mestre, um escravo será eleito para substitui-lo. O nó mestre pode ler e escrever arquivos enquanto que o nó escravo pode apenas realizar operações de leitura. [Chodorow](http://shop.oreilly.com/product/0636920028031.do#tab_04_2) (2013) explica os conceitos de replicação e *sharding* dos dados.

Replicação é o conceito de copiar os dados e distribuir entre os servidores e *Sharding* dos dados é a ação de dividir os dados e deixar cada subconjunto dos dados em um servidor. Ambos conceitos dentro do MongoDB têm formas de configurar, otimizando assim o uso do *cluster.*

* 1. **Neo4J**

Kemper (2015) diz que o Neo4J surgiu em 2003 na empresa Neo Technology e tornou-se *open source* em 2007. Hoje esse código fonte Java encontra-se no GitHub para a comunidade de colaboradores. Apesar de ser *open source* há uma versão *Enterprise* disponível somente através de pagamento de licença. Nessa se contempla características adicionais e suporte ao produto.

Goel (2015) descreve a estrutura de grafos como sendo complemente transacional, possuindo as características ACID (Atomicidade, Consistência, Isolação e Durabilidade). Bruggen (2014) mostra este modelo como uma estrutura de grafos com vértices e arestas que representa como cada registro se relaciona com outra informação, tornando assim a sua representação mais natural, conforme ilustrado na Figura 03.

**Figura 03- Informações armazendas no Neo4J**



Fonte: Bruggen (2014)

Para entender como funciona os vértices e suas propriedades, fazer uma comparação com o modelo relacional é de grande ajuda. Um vértice seria um registro em uma tabela, já as propriedades seria uma tabela de um único registro na qual a coluna seria os campos e a linha seria os valores. Há possibilidade de criar referências para cada vértice, esta técnica é chama da *label*. Ao usar este recurso é possível trabalhar com categorização de vértices, criando assim um conjunto de sub grafos.

As arestas sempre possuem origem, destino e direção, além disso é possível criar propriedades. Assim como os vértices é possível definir um *type* para a aresta, o objetivo é bastante similar com as *labels*.

O Neo4J possui uma linguagem declarativa chamada *Cypher* de fácil compreensão para realizar requisições a base de dados. Linguagens declarativas permitem expressar de forma clara o que se procura, declarando o padrão que deseja e despreocupando como a base de dados irá recuperar a informação.

Este banco de dados suporta requisitos de escalabilidade horizontal e vertical, alta disponibilidade e tolerância a falhas. Em seu cluster há um nó mestre e nó(s) escravo(s). Como os dados são copiados em vários nós todos estes são capazes de servir as requisições.

Caso haja alguma inconsistência ou conflito de dados o nó mestre decidirá qual dado e qual servidor irá servir tal requisição. Há um algoritmo de eleição de um nó mestre em caso de interrupção do funcionamento do mesmo.

# Metodologia

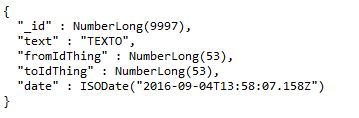
A metodologia usada neste artigo consiste em realizar um levantamento bibliográfico e análise das obras, definição dos cenários de teste, implementação e execução dos testes e por último a análise dos resultados obtidos.

O foco do trabalho é realizar uma avaliação comparativa sobre desempenho entre MongoDB e do Neo4J em um cenário de Internet das Coisas e de grande crescimento da base de dados.

O cenário modelado simula trocas de mensagens entre dispositivos em que todas as informações foram persistidas e recuperadas. A identificação do dispositivo que enviou a mensagem, a identificação de quem recebeu foram consideradas, assim como também o conteúdo da mensagem.

A Figura 4 mostra o modelo de dados implementado no MongoDB e a Figura 5 o modelo no Neo4J.

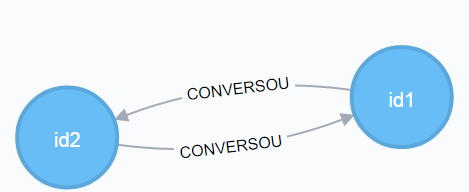
**Figura 04- Modelo de dados implementado**



Fonte: próprio autor

Esse modelo utiliza o \_id como um número do tipo *Long*, assim também com os identificadores dos aparelhos, um campo texto referente a mensagem e um campo data para saber quando esta mensagem foi enviada.

**Figura 05- Modelo de dados implementado no Neo4J**



Fonte: próprio autor

Para o modelo orientado a grafos foi modelado o vértice com a propriedade que corresponde ao id e as arestas representando que a troca de mensagem, cada aresta possui uma propriedade com o conteúdo da mensagem.

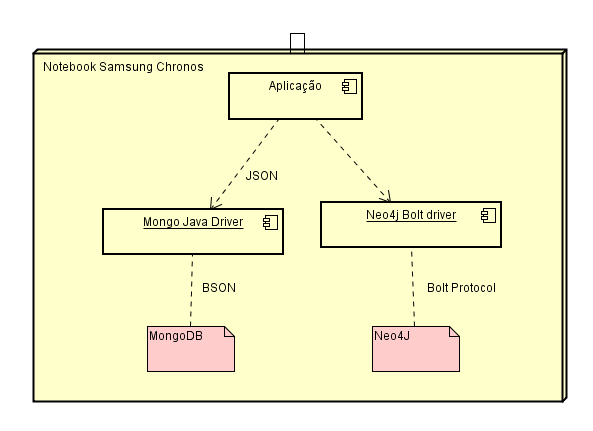
As avaliações foram executadas em um notebook com 6 Gb de memória RAM, um HD SSD de 240GB, processador Intel i5 de 2.50 GHz e Windows 7.

A aplicação que realiza a avaliação de desempenho foi construída em Java SE, utilizando a IDE Eclipse para desenvolvimento.

Para acesso as bases de dados foram utilizados *drivers* da linguagem Java. Para uso do SGBD MongoDB foi utilizado o *driver* Mongo Java Driver que está disponível em MongoDB (2016). Já ao Neo4J foi acessado usando o *driver* Neo4J Bolt Driver que está disponível em Neo4J (2016).

A Figura 6 que mostra como foi feita a arquitetura do software desenvolvido.

**Figura 05 – Diagrama Componentes e Implantação**



Fonte: próprio autor

Para a execução das avaliações os SGBDs são iniciados de forma manual antes dos testes começarem, o MongoDB através de linhas de comandos e o Neo4J através de um executável.

É importante ter conhecimento que há variáveis no ambiente de avaliação usado que não são possíveis controlar, como por exemplo um processo do sistema operacional impactando o desempenho de um processo da aplicação, podendo assim influenciar no resultado de um teste.

A aplicação foi desenvolvida possibilitando o usuário escolher o tipo de simulação, a quantidade de registros iniciais para cada *thread* e quantas vezes os testes serão repetidos. Ao final de cada teste é gerado um (1) gráfico com desempenho dos bancos de dados para cada tipo de simulação. Também foi desenvolvido a possibilidade de salvar os gráficos em um local informado pelo usuário.

Os testes em cada SGBD são executados de forma independente, ou seja, primeiro o MongoDB e depois o Neo4J.

* Para cada modelo de dados são feitos três (3) tipos de simulações:
  + Escritas de registros;
  + Leitura de todos os registros;
  + Leitura de registros de acordo com identificador do dispositivo que enviou a mensagem.
* É simulado a existência de 4 dispositivos usando *threads* para que possa haver requisições concorrentes ao banco a cada repetição.
* Os testes são repetidos N vezes (valor inserido pelo usuário) com quatro (4) *threads* em funcionamento cada uma simulando um dispositivo.
* Cada *thread* é responsável por recuperar ou escrever todos os registros.
* Sendo assim, os momentos de início e fim das operações de todos os registros são recuperados em milissegundos usando o comando em Java *System.currentTimeMilis*.
* Este comando em Java recupera todos os milissegundos desde 1º de Janeiro de 1970.
* Conhecendo o início e fim do teste é feito a diferença para saber o tempo real gasto pela *thread* para executar todas as operações.
* Logo após é feita a média aritmética das informações sobre o tempo médio gasto por cada uma das quatro (4) *threads* para realizar todas as operações com uma determinada quantidade de registros.
* A cada vez que é repetido o teste, é adicionado uma quantidade de quatro (4) vezes o número de registros iniciais a base de dados, sendo que toda esta quantidade de registros é dividida igualmente entre as *threads*.
* A cada vez que N (valor inserido pelo usuário) testes são repetidos, a sua média de resultados é inserida é uma matriz de resultados.
* Com uma matriz de resultados preenchida é possível realizar a geração de gráficos com seus resultados.

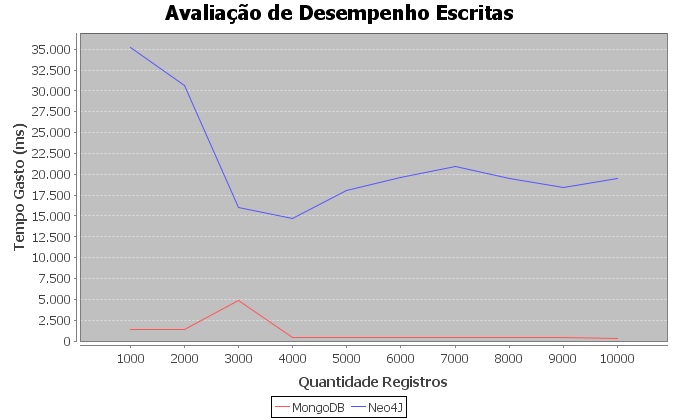
A visualização gráfica mostra o desempenho do banco de dados de acordo com o tipo de simulação, a medida é a função do tempo gasto para executar todas as operações pela quantidade de registros na base.

# Resultados

# Os testes foram feitos usando os três tipos de simulações, com uma carga inicial de mil (1000) registros para cada *thread*, ou seja, a quantidade de registros inicial será de quatro mil (1000 vezes 4) e com dez (10) repetições dos testes.

# Seguem as tabelas, gráficos e avaliações sobre cada tipo de teste realizado.

**Figura 06 - Avaliação de Desempenho de Escritas**



Fonte: resultados do trabalho

**Tabela 01 - Avaliação de desempenho de Escritas**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Quantidades de registros | 4000 | 8000 | 12000 | 16000 | 20000 | 24000 | 28000 | 32000 | 36000 | 40000 |
| MongoDB | 1449 | 1386 | 4853 | 427 | 382 | 487 | 436 | 463 | 376 | 358 |
| Neo4J | 35161 | 30654 | 16040 | 14684 | 19576 | 20956 | 18050 | 19448 | 18409 | 19514 |

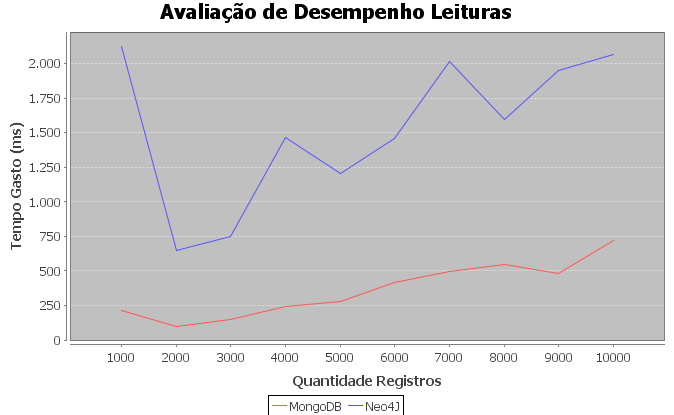
Fonte: resultados do trabalho

Percebe-se que o MongoDB realizou requisições de escrita com pouca variação de seus tempos. Porém o mesmo não acontece com o Neo4J que teve grande variação.

Outro resultado nítido é a grande vantagem do MongoDB quando se trata a proporção da diferença de seus tempos médios com o Neo4J.

Com o gráfico é possível visualizar que o aumento da base de dados para operações de escrita não influência o desempenho do MongoDB, porém o mesmo não acontece com o Neo4J que teve uma piora próxima de 5 mil milissegundos.

**Figura 07 - Avaliação de Desempenho de Todas as Leituras**

****

Fonte: resultados do trabalho

**Tabela 02 - Avaliação de desempenho de Todas as Leituras**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Quantidades de registros | 4000 | 8000 | 12000 | 16000 | 20000 | 24000 | 28000 | 32000 | 36000 | 40000 |
| MongoDB | 215 | 101 | 150 | 242 | 418 | 495 | 546 | 485 | 280 | 723 |
| Neo4J | 2120 | 646 | 749 | 1463 | 1202 | 1454 | 2014 | 1596 | 1947 | 2064 |

Fonte: resultados do trabalho

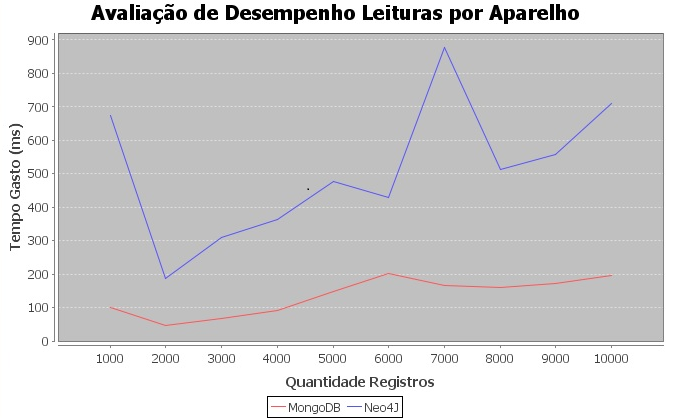
Para todas as operações de leitura o MongoDB se mostrou mais estável, variando muito pouco o seu tempo médio. Já o Neo4J teve grande variação em suas médias.

Assim como aconteceu com as requisições de escrita houve diferença entre os tempos médios do MongoDB e o Neo4J, porém a diferença foi com uma proporção menor.

Para as requisições de leitura em ambos os bancos a quantidade de dados influência no tempo das requisições, havendo assim um aumento do tempo médio de resposta.

Porém quanto mais se aumenta a base de dados o tempo de leitura no banco MongoDB aumenta mais devagar do que no banco Neo4J.

**Figura 08 - Avaliação de Desempenho de Todas as Leituras por Dispositivo**



Fonte: resultados do trabalho

**Tabela 03 - Avaliação de desempenho de Todas as Leituras por Dispositivo**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Quantidades de registros | 4000 | 8000 | 12000 | 16000 | 20000 | 24000 | 28000 | 32000 | 36000 | 40000 |
| MongoDB | 101 | 47 | 69 | 91 | 147 | 201 | 165 | 161 | 172 | 197 |
| Neo4J | 674 | 186 | 310 | 364 | 476 | 430 | 876 | 514 | 558 | 709 |

Fonte: resultados do trabalho

O resultado de testes executando leituras com filtros foi bastante parecido com o resultado anterior sobre leitura de todos os registros.

O único item que fez diferenciar seu resultado foi que com um filtro a quantidade de informações a serem recuperadas foi menor, diminuindo assim seu tempo médio.

Durante o desenvolvimento da aplicação houve o desejo de inserir uma quantidade de dados maior, passando de 1000 registros para 5000 registros por thread. Porém o Neo4J não conseguiu responder as requisições e o MongoDB obteve um tempo de resposta maior do que apresentado na tabela mas conseguiu responder a todas a requisições.

# 

# Conclusões e Trabalhos Futuros

# Com estes resultados conclui-se que o banco de dados MongoDB há uma melhor condição de lidar com grandes quantidades de dados tanto para requisições de escrita quanto para leituras. E que a quantidade de dados influência menos em seu desempenho.

# Em ambos os SGBDs requisições de leitura que possuem filtro por identificador de dispositivo tem comportamento parecido com requisições de leitura que não utilizam filtros entendendo assim que possuir filtros não impacta no desempenho à medida que a base de dados cresce.

# Percebe-se que o MongoDB pode ser usado com um banco de propósito geral, já o uso do Neo4J teria o uso indicado para uma base de dados não muito grande e com informações que se relacionam de forma muito complexa com outras informações, preferencialmente se puderem ser modelas com grafos.

# Durante o desenvolvimento do trabalho percebeu-se que é notável a grande a comunidade e documentação do MongoDB quando se comparado com o Neo4J. Este tipo de suporte é de grande importância quando se está desenvolvendo de *softwares* e provendo manutenção em servidores.

# A área sobre internet das coisas ainda tem muito a crescer não apenas os dispositivos, mas assim também softwares e infraestrutura de TI.

# Para trabalhos futuros na área de Bancos de Dados no contexto de Internet das Coisas há a possibilidade de realização de avaliações simulando dispositivos reais conectados à internet e enviando informações diretamente para um banco de dados ou para um servidor de aplicações.

# Assim como no estudo de Tavares e outros (2016) que realizaram um estudo sobre catálogo de dispositivos, podem haver outras necessidades para a área de internet das coisas que necessite de um estudo para saber qual o melhor SGBD e modelo de dados para certa aplicação e assim poder estudar qual a melhor forma de modelar a base de dados.

# 

**Referências Bibliográficas**

Abramova, V and Bernardino, J. NoSQL databases: MongoDB vs cassandra. In *Proceedings of the International C\* Conference on Computer Science and Software Engineering* (C3S2E '13), Ana Maria Almeida, Jorge Bernardino, and Sudhir Mudur (Eds.). ACM, New York, NY, USA, 14-22. (2013). DOI= <http://dx.doi.org/10.1145/2494444.2494447>

Bruggen, R. V. (2014), Learning Neo4J, Packt Publishing, 1th edition.

Cattell, R. Scalable SQL and NoSQL data stores. *SIGMOD Rec.* 39, 4 (May 2011), 12-27. (2011). DOI: <http://dx.doi.org/10.1145/1978915.1978919>

[Chodorow](http://shop.oreilly.com/product/0636920028031.do#tab_04_2), C. MongoDB: The Definitive Guide, O'Reilly, 2nd edition. (2013).

Crabtree, A. and Tolmie, P. A Day in the Life of Things in the Home. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (CSCW '16). ACM, New York, NY, USA, 1738-1750. (2016). DOI: <http://dx.doi.org/10.1145/2818048.2819954>

Cuzzocrea, A., Song, I., and Davis, K. C. Analytics over large-scale multidimensional data: the big data revolution!. In *Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP* (DOLAP '11). ACM, New York, NY, USA, 101-104. (2011). DOI: <http://dx.doi.org/10.1145/2064676.2064695>

Elliman, D. “The Big in BigData Misses The Point”, <https://www.thoughtworks.com/de/insights/blog/big-big-data-misses-point> , April. (2014)

Goel, A. Neo4J Cookbook, Packt Publishing, 1th edition. (2015)

Ha, J., Yoon, J., Heo, J., Han, Y., Jung, J., Yun, Y. and Eun, S. A perspective on the IoT services through a multi-dimensional analysis. In *Proceedings of the 2015 Conference on research in adaptive and convergent systems* (RACS). ACM, New York, NY, USA, 479-481. (2015). DOI= <http://dx.doi.org/10.1145/2811411.2814188>

Hefnawy, A., Bouras, A and Cherifi, C. IoT for Smart City Services: Lifecycle Approach. In *Proceedings of the International Conference on Internet of things and Cloud Computing* (ICC '16), Djallel Eddine Boubiche, Faouzi Hidoussi, Lyamine Guezouli, Ahcène Bounceur, and Homero Toral Cruz (Eds.). ACM, New York, NY, USA, Article 55 , 9 pages. (2016). DOI= <http://dx.doi.org/10.1145/2896387.2896440>

Jammes, F. Internet of Things in Energy Efficiency: The Internet of Things (Ubiquity symposium). *Ubiquity* 2016, February, Article 2 (February 2016), 8 pages. (2016). DOI= <http://dx.doi.org/10.1145/2822887>

Jenkins, T. Designing the "Things" of the IoT. In Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction (TEI '15). ACM, New York, NY, USA, 449-452.(2015). DOI:

<http://dx.doi.org/10.1145/2677199.2691608>

Karakostas, B and Bessis, N. Intelligent Brokers in an Internet of Things for Logistics. In *Proceedings of the International Conference on Internet of things and Cloud Computing* (ICC '16), Djallel Eddine Boubiche, Faouzi Hidoussi, Lyamine Guezouli, Ahcène Bounceur, and Homero Toral Cruz (Eds.). ACM, New York, NY, USA, Article 5 , 7 pages. (2016). DOI= <http://dx.doi.org/10.1145/2896387.2896390>

Kemper, C. Beginning Neo4J, Apress, 1th edition. (2015).

Marchioni, F. MongoBD for Java Developers, Packt Publishing, 1th edition. 2015.

MongoDB, MongoDB, Inc. 2016. Accessed in October 20, 2016. (2015), Available at: <<https://docs.mongodb.com/ecosystem/drivers/java/>>

Neo4J, Neo Technology, Inc. 2016. Accessed in October 20, 2016. Available at: <<https://neo4j.com/docs/developer-manual/current/drivers/>>

Rith, J., Lehmayr, P. S. and Meyer-Wegener, KSpeaking in tongues: SQL access to NoSQL systems. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*(SAC '14). ACM, New York, NY, USA, 855-857. . (2014) DOI: <http://dx.doi.org/10.1145/2554850.2555099>

Tavares, A. T., Oliveira, M. I., and Lóscio, B. F. Data producer catalogs for the web of things: a study on NoSQL solutions. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing* (SAC '16). ACM, New York, NY, USA, 980-985. (2016). DOI: <http://dx.doi.org/10.1145/2851613.2851660>