Análise de Desempenho de Catálogo de Produtores de Dados para Internet das Coisas baseado em SensorML e NoSQL

Marcelo Iury S. Oliveira[1,2], Bernadette Farias Lóscio[1], Kiev Santos da Gama[1]

[1] Centro de Informática – Universidade Federal de Pernambuco (UFPE)
A. Jornalista Aníbal Fernandes, S/N – Cidade Universitária – Recife – PE – Brasil

[2] Unidade Acadêmica de Serra Talhada - Universidade Federal Rural de Pernambuco
Fazenda Saco, S/N –Serra Talhada – PE – Brasil

iury@uast.ufrpe.br, {miso,bfl,kiev}@cin.ufpe.br

Abstract. *The advent of Internet of Things enabled the development of smarter systems in which things and objects from real-world can exchange information with each other. The amount of participants in such systems is potentially massive. Thus, it is essential to use a catalog to enable publication and discovery of "things". This paper presents a performance analysis of NoSQL database and SensorML as a solution foundations for the development of catalogs for Internet of Things context. The analysis was performed through simulations and the results showed the feasibility of the analyzed solutions.*

Resumo. *O advento da Internet das Coisas permitiu o desenvolvimento de sistemas mais inteligentes nos quais "coisas" e objetos do mundo real podem trocar informação entre si. A quantidade de participantes de um sistema deste tipo é potencialmente massiva. Desta forma, torna-se imprescindível a existência de um catálogo para viabilizar serviços de publicação e descoberta de "coisas". Este artigo apresenta uma análise de desempenho do SensorML e do banco de dados NoSQL como soluções para o desenvolvimento de catálogos voltados para o contexto de Internet das Coisas. A análise foi realizada através de simulações e, como resultado, pode-se verificar a viabilidade das soluções analisadas.*

1. Introdução

O advento da Internet das Coisas (do inglês, *Internet of Things*, IoT) permitiu o desenvolvimento de sistemas mais inteligentes nos quais os objetos do mundo real trocam informações entre si a respeito de seus *status*, localização, problemas e fatores ambientais que, eventualmente, monitoram [1]. Esse novo paradigma vem sendo cada vez mais aplicado a diversos domínios, tais como monitoramento de meio ambiente, gestão de trânsito, controle médico, entre outras aplicações.

Usualmente, as aplicações desenvolvidas, no contexto de IoT, consistem em uma rede de dispositivos inteligentes na qual a Internet é usada como principal veículo de disseminação das informações [1, 2]. Estes dispositivos podem ser equipados com sensores e identificação por radiofrequência (RFID) e são usados no monitoramento e controle de uma determinada região, seja esta uma área externa ou interna. Estes dispositivos podem atuar tanto como produtores de dados (PDs) quanto como consumidores de dados (CD). Os produtores de dados podem ser qualquer tipo de recurso, físico ou virtual, que seja capaz de prover dados

para os consumidores. Os dados, neste caso, podem advir de sensores, *smartphones*, até equipamentos domésticos.

Segundo a IDC (*International Data Corporation*), até 2020, a quantidade de dispositivos conectados a Internet alcançará cerca 30,1 bilhões [3]. Todos esses novos dispositivos aumentam a demanda para o processamento e armazenamento de informações, assim como trazem mais tráfego de dados para as redes, aumentando, consequentemente, a pressão sobre a infraestrutura de servidores. Como a quantidade de potenciais recursos que podem atuar como produtor de dados é massiva e crescente, selecionar qual o produtor mais apropriado para o provimento de informação se torna um novo desafio a ser resolvido. De maneira semelhante, a publicação de novos produtores também precisa ser realizada de acordo com certos padrões a fim de garantir a interoperabilidade e facilitar a descoberta de produtores. Em geral, o problema da publicação e descoberta de produtores de dados requer a resolução de questões, tais como recursos autônomos e heterogêneos e interoperabilidade [5].

Desta forma, torna-se imprescindível a existência de mecanismos eficientes para a publicação e descoberta de produtores de dados. Um serviço de publicação, por exemplo,deve permitir que produtores sejam capazes de publicar detalhes sobre seus serviços para que possam ser encontrados por consumidores de dados. E, alternadamente, o serviço de descoberta deve permitir que consumidores sejam capazes de descobrir os produtores mais adequados as suas necessidades. Uma vez descobertos os produtores, o serviço de descoberta deve retornar aos consumidores dados suficientes para que possam avaliar os produtores de forma independente, antes de contatá-los ou integrá-los às suas aplicações.

Uma das formas de desenvolver um serviço de descoberta é a partir do registro de metadados de descrição dos produtores em um catálogo [4]. Usualmente, estes catálogos são implementados em sistemas de gerenciamento de banco de dados (SGBDs). Devido ao volume massivo de produtores e consumidores de dados, é essencial que o catálogo ofereça suporte a replicação automática e distribuição horizontal, permitindo assim atender mais facilmente requisitos de escalabilidade. O catálogo também deve permitir tanto o registro de produtores heterogêneos de dados, quanto o atendimento à requisições de busca oriundas de consumidores de dados também heterogêneos. Em função disso, o catálogo deve utilizar um esquema de dados que permita a extensão e atualização dos esquemas de representação de produtores de dados, assim como o suporte a registros esparsos nos quais alguns campos não são preenchidos.

Dentre as soluções existentes, os bancos de dados NoSQL [19] e o SensorML [6] têm se destacado no desenvolvimento de catálagos e na modelagem de dispostivos de IoT, respectivamente.. Sistemas de Banco de Dados NoSQL são uma classe de banco de dados não-relacionais que possuem como características alta disponibilidade, menor tempo de resposta, suporte a paralelismo e flexibilidade de esquema, sendo propícias ao contexto de IoT [19]. Por sua vez, o SensorML é uma especificação aprovada pelo *Open Geospatial Consortium* que fornece modelos padrões em codificação XML para descrever sensores e processos de medição [8]. O SensorML pode ser usado para descrever uma ampla gama de produtores de dados, incluindo dispositivos embarcados e até sensores remotos.

Visto que ambas as tecnologias podem impactar diretamente no desempenho geral do catálogo, este artigo tem como objetivo apresentar uma análise de desempenho e de viabilidade do SensorML e do banco de dados MongoDB como soluções para o desenvolvimento de um catálogo de produtores de dados no contexto de IoT. A análise foi

realizada através de simulação e como resultado verificou-se a viabilidade de ambas as soluções.

A organização deste documento segue da seguinte forma: após esta seção introdutória, na qual é feita uma explanação geral sobre a problemática que motiva a realização deste trabalho, é apresentado o *background* teórico do trabalho. Na Seção 3, é apresentado o método de pesquisa. A avaliação proposta é apresentada na Seção 4. Na Seção 5, são apresentados alguns trabalhos relacionados. A Seção 6 conclui o trabalho com um breve sumário das contribuições e apontando possíveis trabalhos futuros.

2. *Background* Teórico

O embasamento teórico para este trabalho fundamenta-se em banco de dados NoSQL e no modelo de dados SensorML.

2.1. Banco de Dados NoSQL

Sistemas de bancos de dados NoSQL (*Not only SQL*) tem emergido como uma abordagem promissora para armazenamento e processamento de consultas envolvendo grandes volumes de dados [12]. Estas soluções baseiam-se no teorema CAP (*Consistent, Available, Partition*) ao sacrificar a consistência em busca de permitir uma maior disponibilidade e particionamento dos dados [12]. Desta forma, diferentemente dos bancos de dados tradicionais, que buscam garantir as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade), os bancos NoSQL fundamentam-se no paradigma BASE (*Basically Available, Soft State, Eventually consistent*) suportando um modelo de consistência mais fraco (*e.g.* consistência eventual), e um conjunto mais limitado de operações [12].

Diversas soluções NoSQL têm sido disseminadas e aplicadas nos mais diversos domínios, a exemplo de BigData e aplicações científicas [12]. Embora possuam várias características comuns entre si, é possível classificá-las de acordo com o modelo de representação de dados utilizado. Atualmente, os principais modelos das bases de dados NoSQL são: chave-valor, orientado a coluna, baseado em grafos e orientado a documentos [12].

2.2. Sensor Web Enablement e SensorML

A especificação *Sensor Model Language* (SensorML) é uma linguagem legível por máquinas, para descrever sensores e processos de medição. O SensorML é pertencente à arquitetura *Sensor Web Enablement* (SWE) do grupo OGC (*Open Geographic Consortium*) [9] que por sua vez define um conjunto de padrões para formatos de dados, metadados e interfaces de serviços. O SWE é uma das estratégias mais robustas e aceitas pela comunidade científica [8]. Os outros padrões especificados pelo SWE são:

- *Observations & Measurements* (O&M) - Define um modelo de dados para observações e medidas de um sensor.
- *Sensor Observations Service* (SOS) - Interface para requisitar observações dos sensores, sejam elas em tempo real ou armazenadas.
- *Sensor Planning Service* (SPS) - Define interfaces para consultas que informam os recursos dos sensores, bem como permitem configurar e manipular os sensores.
- *Sensor Alert Service* (SAS) - Padrão para notificação de alertas dos sensores.
- *Web Notification Services* (WNS) - Interface para troca de mensagem entre componentes do SWE.

O SensorML é baseado em XML e define diversos elementos/entidades que podem ser usados para modelar sensores e dispositivos de Intenert das Coisas. Dentre os elementos existentes, destacam-se o PhysicalComponent, PhysicalSystem, ProcessChain, Detector, Sensor e outros. Em especial, um dos elementos mais importantes é PhysicalComponent que permite a descrição de atributos de identificação, caracterização, funcionalidades, outputs, parâmetros e serviços.

Algumas implementações do SWE foram propostas, dentre as mais robustas destacam-se os projetos OSIRIS [1] e GENESIS[2]. Inclusive, o projeto OSIRIS investiu esforços no desenvolvimento de mecanismos de busca de sensores baseados em SensorML e de ontologias para descrever fenômenos observados pelos sensores. No entanto, estes projetos são restritos ao domínio de dados geoespaciais e a persistência é implementada com SGBD relacional. Dessa forma, além de não ser facilmente generalizável para outros tipos de aplicação, ainda há o problema de escalabilidade e rigidez no esquema de dados.

3. Método de Pesquisa

O presente trabalho de pesquisa, sob o ponto de vista de seus objetivos, tem caráter exploratório, na medida em que busca proporcionar maior entendimento sobre um problema proposto e também permitir a construção de hipóteses [13]. Mais especificamente, este estudo busca a avaliação de tecnologias usualmente aplicadas no desenvolvimento de catálogos de PDs para IoT.

A escolha do método de pesquisa está relacionada com aspectos relativos ao tipo de investigação que se pretende executar. Segundo Yin [13], para determinar a melhor estratégia de pesquisa, devem-se considerar fatores, tais como: qual o tipo de questão de pesquisa, o nível de controle sobre a unidade de pesquisa e suas variáveis. Diante do exposto, o presente trabalho busca responder primeiramente a seguinte pergunta de pesquisa.

> *As tecnologias banco de dados NoSQL e SensorML são soluções viáveis para o desenvolvimento de catálogos de produtores de dados para o contexto de IoT?*

Esta pergunta de pesquisa possui característica causal, tendo como objetivo avaliar a eficiência da solução desenvolvida identificando as relações causa e efeito. Este tipo de pergunta de pesquisa, usualmente, é respondido através de experimentos. Sendo assim, neste trabalho, foi realizada uma prova de conceito.

4. Avaliação

Os principais critérios de avaliação da solução desenvolvida podem ser divididos em duas categorias: robustez e flexibilidade. Estes critérios foram derivados da pergunta de pesquisa apresentada na Seção 3, sendo eles:

1. Robustez:
    a) Sistemas de Banco de Dados NoSQL são capazes de manipular requisições frequentes de cadastro, atualização e consulta de PDs?
    b) A incorporação e/ou remoção de atributos afeta o desempenho do sistema?
2. Flexibilidade

[1] http://www.osiris-fp6.eu

[2] http://www.genesys-project.eu/

a) O uso conjunto de banco de dados NoSQL e SensorML oferece suporte para o registro de PDs heterogêneos?
b) O SensorML oferece suporte para a incorporação ou remoção de atributos de um PD?

Um problema que surge ao se projetar experimentos para soluções de IoT é a complexidade em controlar e viabilizar as variáveis e unidades experimentais. A realização de experimentos usando sistemas e componentes reais pode ser impossível ou impraticável por causa do alto custo de testes com componentes reais, ou por causa da duração do experimento em tempo real ser impraticável.

A alternativa usada neste trabalho foi o emprego de simulação como um meio para a realização de experimentos. A simulação permite que se modele um sistema próximo do real, aumentando a confiabilidade nos resultados e nas decisões [14]. Desta forma, visando validar a solução proposta, foi desenvolvido um simulador que faz uso de *mocks* para simular a interação de PDs e CDs. Os *mocks* são objetos pré-programados que simulam partes do comportamento de um sistema real e são capazes de verificar se o sistema está funcionando conforme especificado.

Por fim, o catálogo foi avaliado através de um protótipo com as funções fundamentais da solução almejada. Isso permitiu avaliar a viabilidade das tecnologias, aferindo a complexidade, tempo necessário para o desenvolvimento completo e os custos envolvidos.

4.1. Protótipo do Catálogo

O protótipo do catálogo de PDs é composto essencialmente de dois componentes: um catálago propriamente dito e um módulo de acesso ao catálogo. O catálogo é responsável pelo armazenamento centralizado dos metadados de descrição dos PDs, tendo sido implementado em um banco de dados NoSQL. Dentre as diversas soluções existentes, foi escolhido o MongoDB por estar entre os SGBDs mais usados [20] e por permitir o uso de esquemas de dados flexíveis[5]. O MongoDB é um banco de dados NoSQL *open source* orientado a documentos [20, 5]. Os dados são armazenados em pares de chave/valor dentro de documentos em uma versão binária de JSON [5]. As consultas do MongoDB podem ser facilmente convertidas a partir de instruções SQL. Isso facilita a migração de aplicações e a integração com bibliotecas de acesso a dados, tais como o *framework* Spring [7].

O MongoDB fornece suporte nativo a indexação, distribuição horizontal, replicação automática e compatibilidade com o paradigma MapReduce. O MongoDB fornece duas abstrações para arquiteturas em *clusters*: *replica sets* e *sharding*. Os *replicas sets* são *clusters* de replicação assíncrona e com tecnologia automática de tolerância a falhas e o *sharding* é um sistema de particionamento de dados automático. Aumentar a quantidade de instâncias em um *replica set* e um *sharding* contribui para o aumento da tolerância a falhas e da escalabilidade horizontal, respectivamente. A versão usada no desenvolvimento do protótipo foi a versão 2.6.6 para Windows 64bits.

Conjuntamente com o MongoDB, foi utilizado o *framework* Apache JCS[3] para implementação do sistema de *cache* de consultas. JCS é um sistema de *cache* distribuído escrito em Java. Este foi especialmente construído para acelerar aplicações, fornecendo um meio para gerenciar dados em *cache* de diversas naturezas dinâmicas.

[3] http://commons.apache.org/proper/commons-jcs/

Por sua vez, o modelo de acesso é responsável pelo provimento das operações de criação, atualização, remoção e consultas de instâncias de PDs no catálogo. Mais especificamente, as operações de consulta permitiam a busca por PD de acordo com parâmetros, tais como fenômeno de interesse, limitação espacial, temporal, identificador e outros. O modelo de acesso foi desenvolvido na linguagem Java[4], versão 1.7.0_71, tendo sido escolhida por apresentar facilidades para a implementação do sistema. Dentre as facilidades oferecidas, podem-se destacar: alta portabilidade do código-fonte, existência de suporte para comunicação através da rede, bibliotecas nativas para acesso a banco de dados e suporte a programação concorrente. No mais, o módulo de acesso, fez uso do driver JDBC oficial do MongoDB, versão 2.13.0[5]

Foi utilizada a biblioteca SWE *Common Library*[6] como base para a implementação das funções de codificação, decodificação e validação dos registros dos PDs descritos em SensorML. Em virtude de o SensorML ser especificado em XML e o MongoDB utilizar um formato derivado do JSON, foi necessária a implementação de *parsers* que permitissem a conversão entre os dois formatos. A propósito, empregou-se a convenção *Badgerfish* [15] na tradução de XML e JSON. Essa convenção estabelece um conjunto de regras que, sobretudo, lida com os aspectos de conversão de atributos, elementos vazios e a ordem de disposição das informações.

4.2. Modelo de Simulação

O ambiente de simulação foi composto por um conjunto de *mocks* PDs e CDs que, de forma programada, submetiam requisições de cadastro, atualização e consulta para o protótipo de catálogo apresentado na subseção anterior. Em relação ao lado servidor da solução, o simulador dispunha de um conjunto de *threads* responsáveis por tratar as requisições. Cada *thread* executava uma instância do protótipo.

A simulação foi executada em ciclos, na qual todos os CDs ou PDs participantes submetiam requisições individualmente em um tempo aleatório máximo de 1 segundo. A quantidade de ciclos executados por cada conjunto era determinada pela quantidade de requisições a serem submetidas por cada entidade. Por exemplo, 10.000 consumidores enviando 100 consultas equivaliam a 100 ciclos. Além disso, não havia sincronização entre os ciclos, podendo, assim, existir consultas oriundas de diferentes conjuntos sendo atendidas simultaneamente.

Como o objetivo da simulação era verificar como o sistema se comportava frente ao recebimento intermitente de requisições, independentemente do tempo de transmissão das mesmas, foram abstraídos aspectos relativos à comunição, tais como topologia de rede, congestionamento e variação de tempo de transmissão. Desta forma, todas as entidades participantes da simulação foram executadas em uma única máquina, estando plenamente disponíveis e podendo se comunicar entre si sem a ocorrência de falhas.

O resultado obtido na simulação é composto por dois valores: a latência (*i.e.* duração) total de execução e a vazão de atendimento de consultas. Ressalta-se que, devido à aleatoriedade entre a ordem de atendimento das requisições e o intervalo de tempo entre as mesmas, calcular o valor exato do tempo de execução não é simples. Assim, o nosso

[4] https://www.java.com

[5] https://github.com/mongodb/mongo-java-driver/releases

[6] https://github.com/sensiasoft/lib-sensorml

simulador calcula um valor aproximado considerando o tempo de relógio, ao invés do tempo individual de cada requisição [14].

4.3. Experimentos e Cenários Simulados

Foram realizados três tipos de experimentos que tinham como principal objetivo avaliar a viabilidade da aplicação em termos de desempenho e robustez. Em um dos experimentos, chamado de **Experimento A**, foram realizadas simulações nas quais todos os PDs pertencentes ao testbed tiveram seus registros cadastrados, atualizados e removidos do sistema, sendo executada apenas uma única vez cada tipo de operação. Este experimento tinha como principal objetivo avaliar a latência e a vazão no atendimento de requisições. Por sua vez, os outros experimentos se diferenciavam por fazer uso de uma carga de trabalho maior que permitia avaliar, além do desempenho, a robustez do catálogo. Estes experimentos se diferenciavam quanto ao tipo de carga de trabalho a ser executada, havendo uma configuração que possuia maior quantidade de requisições de atualização (**Experimento B**) e outra que apresentava uma maior quantidade de requisições de consulta (**Experimento C**).

De maneira a avaliar a flexibilidade da solução, nos experimentos B e C, utilizaram-se esquemas distintos para representação dos PDs que compunham o *testbed*, como pode ser visto na coluna Atributos do Quadro 1. Todos os PDs possuiam em comum a descrição de sua identificação, características e output, podendo apresentar outras descrições, tais como localização e capacidades. Além disso, no Experimento B, as operações de atualização realizavam de forma alternada a inclusão ou remoção de alguns dos atributos de descrição do PD. Por exemplo, enquanto em um ciclo eram especificadas as coordenadas de localização, no ciclo seguinte, estas coordenadas eram removidas da descrição do PD.

Nas simulações, foram usados tipos distintos de CDs e PDs, conforme apresentado no Quadro 1. Estas entidades são comumente usadas na produção de sistemas para cidades inteligentes, que podem ser consideradas um sistema de larga escala de IoT [16]. A quantidade de PDs foi determinada com base em estimativas relacionadas a cidade do Rio de Janeiro, tendo esta sido escolhida como referência tanto pela dimensão da cidade, que é segunda maior cidade do Brasil, quanto pela significativa disponibilidade de bases de dados abertas sobre o Rio de Janeiro[7]. Todos os PDs foram modelados com o elemento *PhysicalComponent* do SensorML, por ser um dos componentes mais abrangentes em termos de atributos descritivos.

Há um conjunto de CDs para cada tipo de PD usado no testbed. Isto é, existe um conjunto de CD que busca apenas táxis, outro que busca apenas ônibus e assim por diante. Todos CDs submetem requisições de consulta de PDs que possuam um determinado valor para o atributo TESTE. Este atributo pertence ao grupo descritivo Características presente em todos os PDs. No total, foram geradas 80 consultas distintas.

A quantidade de tipos distintos de CDs (*e.g.* táxis, ônibus e câmeras) foi proporcional a quantidade total de PDs e CDs disponíveis. Por exemplo, considerando um total de 10mil CDs e 100mil PDs, dos quais 34.230 (34,23%) representam PDs de Táxis, houve 3.423 CDs efetuando buscas relativas a taxis. Isto é, dos 10 mil consumidores, 34,23% deles efetuaram buscas sobre táxis, seguindo a mesma lógica, 900 (9%) e 160 (1,6%) efetuaram de ônibus e câmeras de tráfego, respectivamente. A quantidade total de consumidores diferiu de acordo com tipo de experimento. No Experimento B, foi usado um total de apenas 1.000 CDs.

[7] http://data.rio.rj.gov.br/

Enquanto, nos cenários do Experimento C, foram utilizados 10.000 e 25.000 CDs. Estes valores foram definidos com base na quantidade média de usuários ativos do Waze em grandes cidades da Europa [17].

Quadro 1. Sumário de Produtores de Dados Usados na Avaliação

Tipo de Produtores	Atributos do Produtor	Quantidade de PDs	
		Experimentos A e B	Experimentos C
Táxi	Identificação, Localização, Características e Output	34.000	34.230
Ônibus	Identificação, Localização, Características e Output	9.000	9.000
Estação Meteorológica	Identificação, Localização, Características, Funcionalidades e Output	320	320
Câmera de Tráfego	Identificação, Localização, Características, Funcionalidades, Input e Output	1.600	1.600
Base de Dados Aberta	Identificação, Características, Funcionalidades, Informações de Contato e Output	28.500	28.500
Semáforos de Transito	Identificação, Localização, Características, Funcionalidades e Output	-	5.250
Viaturas de (Polícia e Transito)	Identificação, Localização, Características e Output	-	2.000
Metrô e Trem	Identificação, Localização, Características e Output	-	100
Sistemas de Alerta	Identificação, Localização, Características, Funcionalidades e Output	-	320
Outras Fontes (páginas web e aplicativos sociais)	Identificação, Características, Funcionalidades, Informações de Contato e Output	-	18.680
Total		**73.420**	**100.000**

Visto que a solução de banco de dados contribui significativamente para o desempenho do catálogo, foram avaliadas diferentes configurações do MongoDB com o objetivo de definir os limites do sistema de acordo com a máquina disponível e de como o sistema responde a carga de trabalho. As abordagens utilizadas foram: criação de índices, uso de subcoleções e *cache* de resultados.

No experimento C, as simulações ocorreram com um percentual dos PDs já cadastrados e, ao longo da simulação, os PDs não registrados solicitavam o cadastro de seus metadados. Em todos os experimentos, houve uma limitação de 100 registros retornados por cada consulta. Essa limitação é uma prática comum em diversos sistemas da Web [18].

Além dos parâmetros já citados, que definem uma configuração base para daavaliação, foram definidos diferentes cenários a partir da variação de dois outros parâmetros do modelo de simulação: quantidade de CDs e quantidade de ciclos de requisições.No Experimento A, procurou-se avaliar o efeito da quantidade de *threads* utilizadas para o atendimento das requisições. Em contraposição, nos experimentos B e C, foi empregado um total fixo de 50 *threads*. O Quadro 2 apresenta os cenários executados para os Experimentos B e C.

Todos os experimentos foram executados em um notebook Asus S46C, - Processador Intel Core i5-3317U 1.7GHz, Memória RAM 6GB e Sistema Operacional Windows 8.1 64 bits. Cada experimento executou um mínimo de 4 rodadas para cada cenário executado. Para grande maioria dos cenários, as 4 rodadas foram suficientes para obtermos resultados com 95% de nível de confiança e erro máximo de 5%.

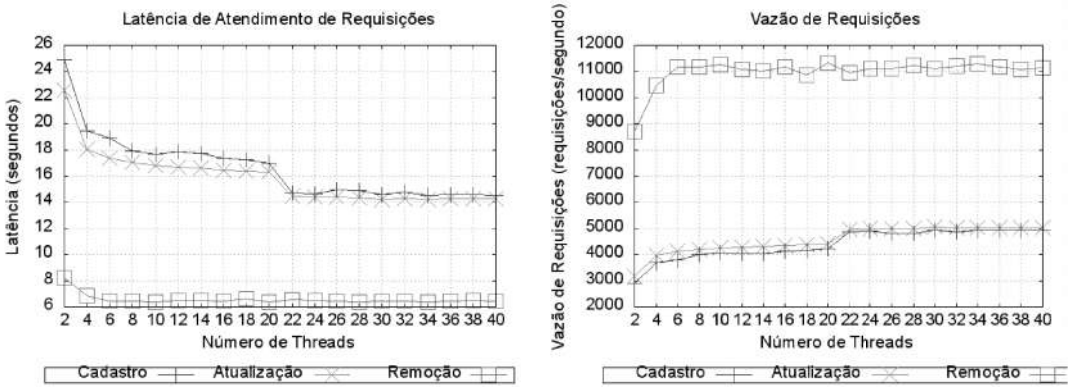
Quadro 2. Cenários Simulados

Experimento	Id. Cenário	Quant. de CDs	Quant. de Ciclos	Quant Máx. de Tuplas	Percentual de PDs Previamente Cadastrados	Política de BD
B	Cenário B1	1 mil	1000	100	0%	Com índices
C	Cenário C1	10 mil	100	100	50%	Sem índices
	Cenário C2	10 mil	100	100	50%	Com índices
	Cenário C3	10 mil	500	100	50%	Com índices
	Cenário C4	10 mil	100	100	50%	Com índices, subcoleções
	Cenário C5	10 mil	500	100	50%	Com índices, subcoleções
	Cenário C6	10 mil	100	100	50%	Com índices, subcoleções e *cache*
	Cenário C7	10 mil	100	150	50%	Com índices, subcoleções e *cache*
	Cenário C8	10 mil	100	100	75%	Com índices, subcoleções e *cache*
	Cenário C9	10 mil	100	150	75%	Com índices, subcoleções e *cache*
	Cenário C10	10 mil	100	100	100%	Com índices, subcoleções e *cache*
	Cenário C11	10 mil	100	150	100%	Com índices, subcoleções e *cache*
	Cenário C12	10 mil	500	100	50%	Com índices, subcoleções e *cache*
	Cenário C13	10 mil	500	150	50%	Com índices, subcoleções e *cache*
	Cenário C14	10 mil	500	100	75%	Com índices, subcoleções e *cache*
	Cenário C15	10 mil	500	150	75%	Com índices, subcoleções e *cache*
	Cenário C16	10 mil	500	100	100%	Com índices, subcoleções e *cache*
	Cenário C17	10 mil	500	150	100%	Com índices, subcoleções e *cache*
	Cenário C18	25 mil	100	100	50%	Com índices, subcoleções e *cache*
	Cenário C19	25 mil	100	150	50%	Com índices, subcoleções e *cache*
	Cenário C20	25 mil	100	100	75%	Com índices, subcoleções e *cache*
	Cenário C21	25 mil	100	150	75%	Com índices, subcoleções e *cache*
	Cenário C22	25 mil	100	100	100%	Com índices, subcoleções e *cache*
	Cenário C23	25 mil	100	150	100%	Com índices, subcoleções e *cache*
	Cenário C24	25 mil	500	100	50%	Com índices, subcoleções e *cache*
	Cenário C25	25 mil	500	150	50%	Com índices, subcoleções e *cache*
	Cenário C26	25 mil	500	100	75%	Com índices, subcoleções e *cache*
	Cenário C27	25 mil	500	150	75%	Com índices, subcoleções e *cache*
	Cenário C28	25 mil	500	100	100%	Com índices, subcoleções e *cache*
	Cenário C29	25 mil	500	150	100%	Com índices, subcoleções e *cache*

4.4. Resultados

As Figuras 1a e 1b mostram os resultados obtidos no Experimento A. Em cada um dos três cenários apresentados (*i.e.* Cadastro, Atualização e Remoção) foram submetidas aproximadamente 74 mil requisições. O Eixo x apresenta a variação da quantidade de *threads* utilizadas para atender as requisições e o Eixo y apresenta as latências e as vazões obtidas. Analisando os resultados, pode-se observar que as operações possuem desempenho distinto, sendo as operações de remoção e inserção as mais rápidas e lentas, respectivamente. Apesar da operação de atualização ser mais rápida que a operação de cadastro, ambas estão muito próximas em termos de desempenho. Este comportamento acontece porque ambas as operações realizam a escrita completa de um regitro de PD no banco de dados.

Contudo, como a operação de atualização não modifica o valor da chave primária, não é necessário atualizar o índice, consequentemente tornando-as mais rápidas que as operações de inserção. Por fim, as operações de remoção são mais rápidas que as demais, pois apenas marcam os registros a serem removidos, tendo a sua efetiva exclusão sendo executada posteriormente pelo gerenciador do MongoDB.



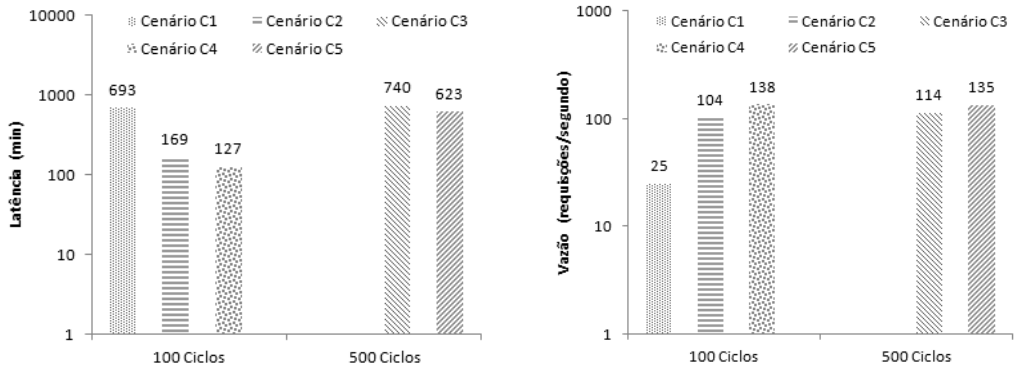
(a) (b)

Figura 1. Experimento A. (a) Latência e (b) Vazão de Atendimento de Requisições

Pode-se ainda perceber que aumentar a quantidade de *threads* resulta em melhoria de desempenho. Todavia, há um limiar no qual a adição de novas *threads* não oferece uma melhora significativa de desempenho. Ressalte-se que esse limiar está diretamente ligado à quantidade de núcleos de processamento existentes na máquina de execução dos experimentos. Certamente, dispor de mais núcleos de processamento poderá aumentar a vazão da aplicação.

Sobre os resultados do Experimento B, o sistema executou a carga de trabalho em um tempo médio de 7,6 horas com uma vazão média de aproximadamente 3 mil requisições atendidas por segundo. Durante o experimento, apesar de consumir toda capacidade de processamento da máquina de teste, a aplicação não apresentou problemas de concorrência e conteve o consumo de memória em torno de 2 GB. É importante salientar que a memória RAM é um dos principais pontos de análise em um mecanismo de registro e descoberta, pois está diretamente ligada ao número de requisições que o servidor é capaz de atender em determinado tempo.

As Figuras 2a e 2b mostram os resultados obtidos através da simulação dos cenários C1 a C5 do experimento C, onde foi executada uma carga de trabalho entre 1 milhão e, aproximadamente, 5 milhões de consultas. Um resultado esperado que pode ser observado na Figura 2a é que, para a maioria dos cenários, a duração para o atendimento de todas as requisições aumenta à medida que a quantidade de ciclos também cresce. Entretanto, esse aumento da duração do experimento não está relacionado a uma melhoria na vazão do sistema. Pode-se perceber isso a partir dos cenários C2 e C3, nos quais há apenas uma diferença de aproximadamente 9% (10 requisições), mesmo existindo aumento de 100 para 500 ciclos de simulação. Esse mesmo padrão pode ser percebido nos cenários C4 e C5.



(a) (b)

Figura 2. Experimento C. (a) Latência e (b) Vazão de Atendimento de Requisições

Outro resultado que se destacou foi o ganho de eficiência com uso de *cache* para armazenamento das consultas, que apresentou resultados significativamente superiores ao restante das abordagens, como pode ser observado na Figura 3. Considerando todos os cenários que fazem uso de *cache*, a vazão mínima alcançada foi de 887 requisições processadas por segundo (Cenário C11). Esse resultado é aproximadamente 542% superior que a vazão obtida no Cenário 4, que apresentou a melhor vazão dentre aqueles em que não há uso de *cache*.

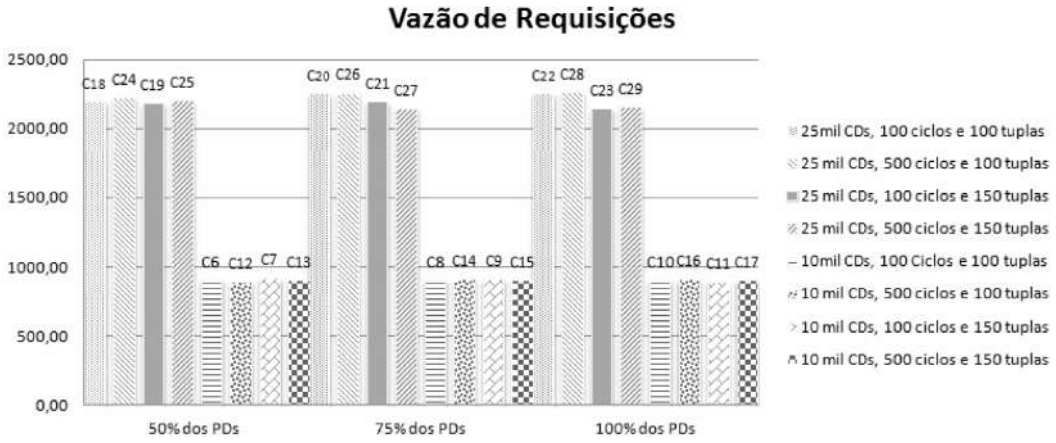


Figura 3. Experimento C: Vazão de Atendimento de Requisições

No que tange a quantidade máxima de tuplas retornadas por consulta, não houve uma diferença significativa quandoo *cache* foi usado. Isso pode ser percebido, por exemplo, comparando os pares de cenários "C18 e C19" e "C6 e C7" que diferem apenas na quantidade de tuplas. Apesar dos cenários C19 e C7 retornarem 50% mais tuplas que os cenários C18 e C6, os mesmos apresentam uma diferença máxima de apenas 1%. Entretanto, é importante destacar que o aumento da quantidade de tuplas eleva, consequentemente, a quantidade de memória ocupada pelo *cache*.

Outro resultado que se destacou foi o aumento da quantidade de consumidores, apesar de aumentar a carga de trabalho a ser executada, não resultou em aumento de latência da execução do experimento. Na verdade, houve uma melhoria média de 143% na vazão de atendimento das requisições. Considerando os cenários C8 e C23 que foram, respectivamente,

os cenários que apresentaram a melhor e a pior vazão para 10 mil e 25 mil consumidores, há uma diferença aproximadamente 134% entre eles. Esse resultado é explicado por dois fatores: pouca variação nas consultas submetidas e maior quantidade de consultas submetidas. Como o conjunto de tipos distintos de consultas é pequeno (80 consultas), a aumentar do número de consumidores implica em mais consultas repetidas sendo submetidas simultaneamente. Isso aumenta a probabilidade de *cache hit,* reduzindo a necessidade de busca em disco rígido e, consequentemente, diminuindo o tempo médio para atendimento das consultas.

Aliado ao *cache*, a criação de índices se mostrou muito importante para alcançar um bom desempenho. Isso pode ser mais facilmente percebido no Cenário C1, que não fez uso de índices. Este cenário apresentou tempo de duração similar aos dos cenários C3, C5 e C7 que utilizaram índices e possuíam carga de trabalho 5 vezes superior ao Cenário 1. Por sinal, o uso de índices se torna mais determinante na medida em que a quantidade de registros armazenados cresce. Segundo os comandos de profile do MongoDB, as consultas executadas sem o suporte dos índices efetuam uma leitura completa dos registros existentes no sistema.

Além disso, durante todos os experimentos pode-se perceber que o consumo de memória também se manteve estável em torno de 2 GB. Em particular, o *testebed* aplicado no Experimento C, que consistia da maior quantidade de dados usada, ocupou apenas 300MB de armazenamento, possuindo uma média de 4 KB por registro. Em virtude de tratar-se também de um sistema de registro de dados, o baixo consumo de armazenamento dos registros torna viável manipular uma quantidade maior de produtores de dados pertencentes a uma cidade inteligente. No mais, apesar de consumir toda capacidade de processamento da máquina de teste, a aplicação não apresentou problemas de concorrência durante os experimentos.

Em relação à avaliação da flexibilidade, o protótipo foi capaz de lidar com o registro e a busca de PDs heterogêneos. O SensorML usado para descrever os metadados de um PD, por ser baseado em XML, é plenamente extensível. Além disso, o SensorML não delimita quais atributos devem ser especificados, podendo ser incorporados ou removidos elementos descritivos. Em relação ao desempenho, não houve significativa diferença entre os tipos distintos de PDs registrados no sistema, embora possuíssem esquemas de dados também distintos. É importante ressaltar que a escolha de um banco de dados orientado a documentos e do SensorML ofereceuflexibilidade para registro e consulta de entidades pertencentes ao contexto de uma cidade inteligente.

5. Trabalhos Relacionados

A avaliação de desempenho de sistemas e soluções de banco de dados têm sido alvo de vários estudos reportados na literatura. No que diz respeito à avaliação do MongoDB, há trabalhos similares ao nosso, tais como [10, 11, 12]. Entretanto, nenhum dos estudos tem como foco a avaliação do MongoDB usando o SensorML como esquema de dados.

Nyati e colegas [10] realizou uma análise comparativa de desempenho para a execução de operações de inserção e consultas no MongoDB e MySQL. O estudo avaliou o impacto do uso de *threads*, quantidade de máquinas e uso do sharding. Contudo, o estudo utilizou um esquema de dados *flat*, no qual todos os dados estão em um mesmo nível, composto por 18 atributos, não sendo considerado o uso de uma estrutura mais complexa e variável de dados. Além disso, o artigo não específica quais foram os tipos de consultas executadas.

O trabalho apresentado por Dede e colegas [11] avaliou o uso do MongoDB com Hadoop em relação a critérios de desempenho, escalabilidade e tolerância a falhas. O objetivo deles foi identificar o ambiente de *software* adequado para a análise de dados científicos que, assim como dados gerados pelos produtores de dados em IoT, são semi-estruturados. Porém, o *workload* usado nos experimentos era composto por arquivos CSV que possuem uma estrutura *flat*, diferindo apenas na presença de valores vazios para alguns campos.

Carniel e colegas [12] investigou o uso do MongoDB e de outros bancos de dados relacionais e NoSQL no desenvolvimento de *Data Warehouses*. Foram avaliados os tempos de respostas no processamento de consultas, o uso de memória e o uso porcentual de CPU, considerando as consultas do *Star Schema Benchmark*. *Data Warehouses* apesar de armazenarem grandes volumes de dados e utilizarem um esquema multimensional, possuem uma estrutura estática, o que não permite o armazenamento de registros estruturados de forma heterogênea.

6. Conclusão

Este artigo apresentou uma análise de desempenho de um protótipo de catálogo de produtores de dados voltado para o contexto de Internet das Coisas. O protótipo foi desenvolvido usando o banco de dados NoSQL MongoDB para o armazenamento dos dados, e o SensorML como esquema de dados para representação dos dispositivos e fontes de dados. A avaliação foi realizada por meio de simulação e execução de três experimentos distintos. Para viabilizar a avaliação, foi desenvolvido um protótipo em Java, auxiliado pelo *framework* Apache JCS e pela sistema de banco de dados MongoDB.

As simulações confirmaram que o uso das tecnologias supracitadas é viável em dois casos: (1) quando não há uso de cache e a carga de trabalho envolve, de forma majoritária, requisições de inserção, atualização e remoção, e (2) quando há uso de cache e a carga de trabalho envolve, de forma majoritária, requisições de consultas. Para estes casos, os resultados coletados corroboram as suposições quanto à robustez e à flexibilidade das soluções no suporte a produtores e consumidores de dados heterogêneos.

É importante ressaltar que o tempo de duração também depende da máquina utilizada. Os resultados apresentados neste artigo foram coletados através da execução do sistema em uma máquina doméstica. Certamente, em um servidor ou em um cluster, a capacidade de atendimento a consultas pode ser melhorada.

Dentre os trabalhos futuros, pretende-se avaliar outras soluções de banco de dados NoSQL que não sejam orientadas a documentos. Outro desafio é avaliar uma extensão do SensorML que incorpore semântica e metadados relativos a contexto.

6. Agradecimentos

Este trabalho foi parcialmente suportado pelo Instituto Nacional de Ciência e Tecnologia para Engenharia de Software (INES). Marcelo Iury S. Oliveira é bolsista do CNPq–Brasil na modalidade doutorado.

7. Referencias

[1] GUBBI, Jayavardhana et al. Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, v. 29, n. 7, p. 1645-1660, 2013.

[2] ZHOU, Honbo. The internet of things in the cloud: A middleware perspective. CRC Press, 2012.
[3] MACGILLIVRAY, Carrie; TURNER, Vernon; LUND, Denise. Worldwide Internet of Things (IoT) 2013--2020 Forecast: Billions of Things, Trillions of Dollars. IDC. Doc, v. 243661, n. 3, 2013.
[4] MESHKOVA, Elena et al. A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. Computer networks, v. 52, n. 11, p. 2097-2128, 2008.
[5] MongoDB. disponível em http://www.mongodb.org, acessado em 04-03-2015
[6] Botts, Mike, and Alexandre Robin. "OpenGIS sensor model language (SensorML) implementation specification." *OpenGIS Implementation Specification OGC* (2007): 07-000.
[7] CHODOROW, Kristina. MongoDB: the definitive guide. " O'Reilly Media, Inc.", 2013.
[8] BRÖRING, Arne et al. New generation sensor web enablement. Sensors, v. 11, n. 3, p. 2652-2699, 2011.
[9] BOTTS, Mike et al. OGC® sensor web enablement: Overview and high level architecture. In: GeoSensor networks. Springer Berlin Heidelberg, 2008. p. 175-190.
[10]NYATI, Suyog S.; PAWAR, Shivanand; INGLE, Rajesh. Performance evaluation of unstructured NoSQL data over distributed framework. In: Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference on. IEEE, 2013. p. 1623-1627.
[11]DEDE, Elif et al. Performance evaluation of a mongodb and hadoop platform for scientific data analysis. In: Proceedings of the 4th ACM workshop on Scientific cloud computing. ACM, 2013. p. 13-20.
[12]CARNIEL, A. C. et al. Dept. de Comput., Univ. Fed. de Sao Carlos, Sao Carlos, Brazil. In: Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana En. IEEE, 2012. p. 1-9.
[13]YIN, Robert K. Estudo de Caso-: Planejamento e Métodos. Bookman editora, 2015.
[14]Chwif, Leonardo, and Afonso Celso Medina. Modelagem e simulação de eventos discretos. Afonso C. Medina, 2006.
[15]BADGERFISH. disponível em http://badgerfish.ning.com/ acessado em 04-03-2015
[16]SCHAFFERS, Hans et al. Smart cities and the future internet: Towards cooperation frameworks for open innovation. Springer Berlin Heidelberg, 2011.
[17]WAZE STATS. disponível em http://wazestats.com/ acessado em 04-03-2015
[18]VESDAPUNT, Norases; GARCIA-MOLINA, Hector. Identifying Users in Social Networks with Limited Information. 2014.
[19]VIEIRA, Marcos Rodrigues et al. Bancos de Dados NoSQL: conceitos, ferramentas, linguagens e estudos de casos no contexto de Big Data. SBBD. São Paulo, 2012.
[20]DBENGINES http://db-engines.com/en/ranking em 04-03-2015