Concepção e Desenvolvimento de um Serviço Distribuído de Coleta e Tratamento de Dados para Ambientes de Internet das Coisas

Ruben C. Huacarpuma[1], Rafael T. de Sousa Júnior[1], Maristela Holanda[2], Sérgio Lifschitz[3]

[1]Depart. de Engenharia Elétrica – Universidade de Brasília (UnB)

[2]Depart. de Ciência da Computação – Universidade de Brasília (UnB)

[3]Depart. de Informática – Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)

Rubencruzh@unb.br, {desousa, mholanda}@unb.br, sergio@inf.puc-rio.br

Abstract. *The development of the Internet of Things (IoT) has led to a considerable increase in the number and variety of devices connected to the Internet. Sensors have become a regular part of our environment, installed in cars and buildings, as well as smart phones and other devices that continuously collect data about our lives even without our intervention. With such connected devices, a broad range of applications has been developed and deployed, including those dealing with massive volumes of data. In this paper, we propose a data management approach within an IoT environment, thus contributing with the specification of functionalities and the conception of techniques for collecting, filtering and storing data conveniently and efficiently. Our main goal is to enable multiple and distinct middleware to work together in a non-intrusive manner. The corresponding developed prototype is used for the validation of our proposal in a case study regarding a Smart Home System scenario.*

Resumo. *O desenvolvimento da Internet das Coisas (IoT) levou a um aumento do número e da variedade de dispositivos conectados à Internet. Sensores tornaram-se uma parte regular do nosso ambiente, instalados em carros e edifícios, bem como telefones inteligentes e outros dispositivos que coletam continuamente dados sobre nossas vidas, mesmo sem a nossa intervenção. Com tais dispositivos conectados, uma gama de aplicações tem sido desenvolvida e implantada, incluindo aquelas que lidam com grandes volumes de dados. Neste artigo, apresentamos uma proposta para o gerenciamento de dados em um ambiente de IoT, contribuindo com a especificação de funcionalidades e a concepção de técnicas para coletar, filtrar e armazenar dados de forma eficiente. O diferencial da proposta apresentada neste trabalho é que o sistema proposto pode ser integrado a diferentes middlewares. Um protótipo foi desenvolvido para validação da proposta em um estudo de caso sobre um cenário de sistema de casa inteligente.*

1. Introdução

Uma das tendências de novas utilizações da Internet decorre da conexão de grandes quantidades de componentes inteligentes encontrados em dispositivos, tais como

telefones celulares, meios de transporte (carros e bicicletas), fábricas, máquinas e até mesmo em roupas (*Google Glass* e *iWatch*). À rede composta por esse grande número de dispositivos conectados e aplicações vinculadas, dá-se o nome de Internet das Coisas (*Internet of Things - IoT*) (Ashton, 2009, Gubbi et al., 2013).

Mashal et al. (2015) estima que o número desses dispositivos possa atingir 50 bilhões em 2020, devido à sua popularização e baixo custo. O grande volume de dados gerado pelos dispositivos/sensores é fonte de dados para diferentes aplicações de IoT: aplicações de monitoramento de casas, aplicações para transporte público, aplicações médicas, entre outras. Os usuários dessas aplicações demandam respostas imediatas; consequentemente a coleta, processamento e armazenamento em tempo real é um desafio.

Para facilitar o desenvolvimento de aplicações nesses ambientes de IoT, um componente de software da categoria dos *middlewares* é utilizado. O *middleware* é uma camada de software entre a camada física e a camada de aplicação e proporciona um conjunto de abstrações, de modo a facilitar a integração e comunicação de componentes heterogêneos (Fersi, 2015). Um objetivo do *middleware* é esconder os detalhes tecnológicos dos objetos físicos (coisas) e oferecer múltiplos serviços para os desenvolvedores de aplicações (Chaqfeh et al., 2012) (Perera et al., 2014). Os *middlewares* IoT têm-se afirmado como importantes elementos computacionais, que devem proporcionar o acesso aos recursos de sensores heterogêneos de forma transparente aos desenvolvedores das aplicações (A. Ghosh, 2008).

Para gerenciar o grande volume de dados desse ambiente, os *middlewares* implementam seus próprios coletores de dados, que utilizam diferentes arquiteturas de armazenamento, além de definir o modelo lógico de dados, variando de modelo relacional centralizado aos sistemas NoSQL distribuídos. Todos são fortemente acoplados à arquitetura do *middleware*. Como por exemplo, pode-se citar os *middlewares* EcoDiF (Pires et al., 2014), Xively (LogMeIn, Inc, 2016), OpenIoT (Soldados, 2015), RestThing (Qin et al., 2011) e WSO2 (Fremantle, 2014) que implementam seus próprios coletores de dados.

Diante do exposto, a pesquisa descrita neste artigo tem por objetivo propor um Serviço Distribuído de Coleta de Dados (SDCD) para ambientes IoT que possa ser utilizado com diferentes plataformas de *middlewares*, através de uma interface de comunicação que estabelece a comunicação entre o *middleware* e o serviço de coleta de dados com o objetivo de gerenciar os dados desses ambientes. O serviço proposto tem a característica de adaptar o grau de distribuição e paralelismo do processamento, de acordo com a necessidade do ambiente de IoT. Desta forma, o serviço foi projetado para armazenar e processar o grande volume dados produzidos, além de ser baseado em sistemas computacionais de tempo real. O serviço proposto neste trabalho foi efetivamente implementado e avaliado para um sistema simulado de casa inteligente utilizando o *middleware* UIoT (Ferreira et al., 2014).

A estrutura do artigo é a seguinte: a Seção 2 apresenta os dados IoT; a Seção 3 trata de alguns trabalhos relacionados; a Seção 4 contém a especificação do serviço distribuído para coleta de dados proposto neste artigo; a Seção 5 detalha a implementação particular do SDCD; a Seção 6 apresenta a análise do SDCD em um estudo de caso prático com um sistema de casa inteligente; e, finalmente, a Seção 7 contém as conclusões.

2. Dados IoT

As medições feitas pelos dispositivos/sensores geralmente são organizadas como séries temporais, sendo uma série temporal uma sequência de dados numéricos de uma mesma variável onde cada número representa um valor em um determinado ponto do tempo (Rafiei et al., 1997). Desta forma, em um ambiente IoT os dados são coletados em intervalos de tempo regulares ao longo de um período de tempo. Existem dois tipos de séries temporais: estacionárias e não estacionárias. As séries temporais estacionárias são aquelas que flutuam em torno de uma mesma média ao longo do tempo. Já as séries temporais não estacionárias são aquelas cuja média varia ao longo do tempo.

Os dados gerados pelos dispositivos/sensores podem ser discretos ou contínuos. De acordo com Cooper et al. (2009) os dados de IoT podem ser classificados em diferentes categorias: dados de RFID (Radio Frequency Identification) - *tags* (Ashton, K. 2009); endereços únicos de identificação, por exemplo UUID (*Universal Unique Identification*); dados descritivos sobre os objetos, processos e sistemas (ex. Id: 34.672.673.98, tipo: TV, color: preto, marca: *sangsung*); dados posicionais e dados de ambientes pervasivos, por exemplo dados de GPS; dados de sensores, por exemplo dados do clima, temperatura e ruído; dados históricos; grandezas físicas que representam a realidade, por exemplo gravidade, força, luz, som e magnetismo; dados dos estados dos atuadores e dados dos comandos para o controle, estes mostrando o estado atual dos atuadores dos dispositivos.

3. Trabalhos Relacionados

São apresentadas e discutidas a seguir algumas propostas de gerenciamento de dados encontradas na literatura para *middleware* IoT. De maneira geral, essas propostas têm o objetivo de capturar os dados gerados por dispositivos heterogêneos, processá-los e armazená-los em sistemas de armazenamento persistentes.

O Ecossistema Web de Dispositivos Físicos – EcoDiF (Pires et al., 2014), possui um componente *publish-subscribe* que gerencia o registro e recebimento de informações provindas dos dispositivos/sensores. Com base em especificações de serviços *web* RESTful (JAX-RS), foi utilizada a implementação RESTEasy para a criação dos serviços de recebimento das informações. Os dados dos sensores são convertidos no formato XML e enviados via HTTP para armazenamento em banco de dados MySQL.

Xively (LogMeIn, Inc, 2016) possui o componente 'séries temporais' que provê serviços de coleta, gerenciamento e consulta de dados dos dispositivos. Este componente armazena os dados em uma arquitetura em nuvem baseada nos sistemas gerenciadores de bancos de dados MySQL e Cassandra. Assim como o EcoDif, a Xively foi projetada tendo como base princípios REST e padrões *web*. Na Xively, os dados são organizados em *feeds*, *datapoints* e *datastreams*. Um *feed* representa os dados de um ambiente monitorado. Os *datastreams* representam dados enviados por um determinado sensor dentro de uma *feed*. E o *datapoint* representa um único valor de um *datastream* em um determinado instante de tempo.

OpenIoT (Soldados, 2015) dá suporte na descoberta e coleta de dados vindos de sensores móveis. Estes serviços são implementados em um *middleware Publish-Subscribe* chamado *CloUd-based Publish-Subscribe* para IoT (CUPUS). Este *middleware* usa um banco de dados em nuvem permitindo subscrições em paralelo. A

coleta dos dados ocorre através de uma porta de comunicação serial, conexões UDP e requerimentos HTTP.

RestThing (Qin et al., 2011) é composto por um coletor de dados de dispositivo embarcados, sendo o banco de dados responsável pelo armazenamento dos dados e pela diferenciação de suas características. O armazenamento se dá por meio de arquivos XML e JSON, como forma de apresentação para os dados históricos.

WSO2 (Fremantle, 2014), com o seu módulo de monitoramento de atividade de negócio, pode coletar *streams* de dados por lotes e em tempo real. Além disso, através de agentes Java pode receber eventos IoT por uma API para o processamento em tempo real, *batch* ou a combinação deles. Similar a WSO2, a nossa proposta faz uso do processamento paralelo e distribuído em tempo real e armazenamento de dados distribuído Cassandra.

Quanto à coleta de dados, os *middlewares* IoT mencionados até aqui têm características em comum, principalmente em termos de integração e interoperabilidade entre os dispositivos físicos heterogêneos. Entretanto, todos foram projetados para um *middleware* específico. O diferencial da proposta apresentada neste trabalho é que o SDCD pode trabalhar com diferentes *middlewares*, de maneira não intrusiva. Além disso, apesar de utilizar um processo padrão de coleta, filtragem e armazenamento dos dados, a nossa abordagem propõe uma arquitetura totalmente distribuída e paralela que garante a escalabilidade do serviço de coleta de dados para ambientes IoT que geram grandes quantidades de dados, além de tratar e armazenar esses dados.

4. Arquitetura do SDCD

O SDCD proposto neste artigo tem como objetivo o gerenciamento de dados provenientes de *middlewares* IoT, o que significa coletar, processar e armazenar as leituras feitas pelos dispositivos/sensores de IoT, de maneira distribuída e adaptável ao ambiente computacional. A Figura 1, apresenta uma visão geral do SDCD, composto por 04 (quatro) componentes:

- Captura de dados: realiza a coleta das mensagens encaminhadas pelo *middleware*.
- Filtragem de dados: verifica o domínio das mensagens coletadas pelo componente de captura de dados. Analisa as leituras dos dispositivos/sensores para verificar se estão dentro dos valores especificados para os dispositivos/sensores. As mensagens que estão fora do domínio são descartadas.
- Obtenção de metadados: onde são identificados e coletados alguns metadados importantes, como, por exemplo, a localização na qual os dispositivos estão atuando.

- Organização de séries temporais: componente que agrupa os dados coletados, organizando-os com base em uma janela de tempo.

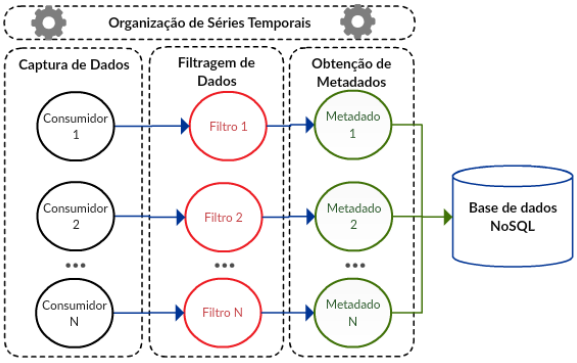

Figura 1. Componentes do SDCD.

Os processos consumidores, filtros e metadados lidam com grandes volumes de processamento e podem adaptar-se a diferentes níveis de paralelismo segundo a necessidade do ambiente. Estes processos são executados seguindo um fluxo de dados, começando pelos consumidores, passando pelos filtros até chegar nos metadados.

O componente *Organização de Séries Temporais* tem como objetivo agrupar e reordenar os dados com base em uma janela de tempo. O tamanho desta janela de tempo é um ponto importante nesta arquitetura, podendo ser configurável de acordo com as características do ambiente computacional. Tal configuração é utilizada para melhorar o desempenho desse componente. A Figura 2 apresenta, através de um exemplo, o comportamento da janela de tempo para um fluxo de dados constante. No primeiro momento (Fase 1), tem-se a janela de tempo t_0 com 9 (nove) grupos de dados armazenados no banco de dados. Ao atingir o valor da janela de tempo t_0, esses dados são agrupados, ordenados e armazenados no banco de dados. Na fase 2, é possível observar outros 9 grupos de dados que atingiram a janela de tempo t_1 prontos para serem agrupados, ordenados e armazenados no banco de dados. Já na fase 3, pode-se ver 2 grupos de dados novos que pertencem à próxima janela de tempo e serão agrupados e ordenados ao fim desta janela t_2. Tal processo segue de forma contínua no tempo.

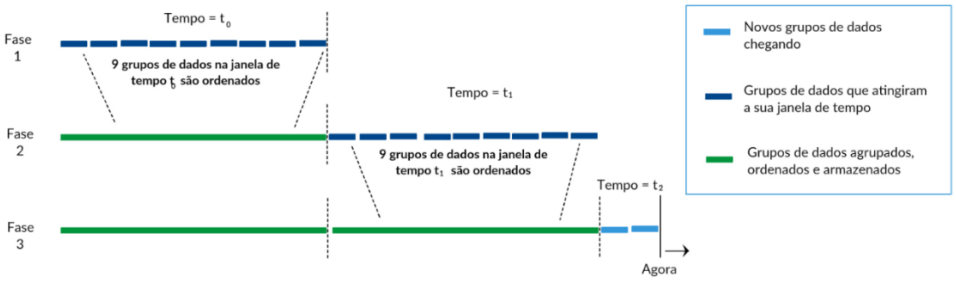

Figura 2. Exemplo de organização de séries temporais.

O fluxo de dados durante a execução do SDCD tem início na captura dos dados pelo componente *Captura de Dados*. Em seguida o componente de *Filtragem* verifica o domínio dos dados coletados. A partir dos dados filtrados, tem-se a criação dos metadados. O componente *Organização de Séries Temporais* não segue o fluxo de execução dos componentes anteriores, já que ele é executado simultaneamente aos processos dos outros componentes, atuando sobre os dados filtrados.

O SDCD tem a característica de não intrusividade no funcionamento do *middleware* já que é fornecida uma interface de comunicação que estabelece a comunicação entre o *middleware* e o SDCD. Esta interface de comunicação faz possível a tradução das mensagens usadas pelos *middlewares* para a estrutura das mensagens gerenciadas pelo SDCD. Uma cópia de toda mensagem gerada pelo *middleware* é traduzida e enviada através do canal de comunicação estabelecido para tal fim. Desta forma, o processo de coleta de dados não deve interferir no fluxo dos dados do *middleware* IoT. A Figura 2 mostra a sequência de troca de mensagens para o serviço. Primeiro, o *middleware* IoT estabelece a comunicação com o serviço (1), o qual transforma as mensagens JSON do *middleware* para o formato de mensagens que o serviço utilizará. Então, a mensagem transformada é encaminhada para o SDCD (2). O SDCD encaminha uma resposta do recebimento da mensagem (3). Finalmente, o *middleware* IoT recebe a resposta do recebimento da mensagem (4). Estas mensagens contêm atributos tais como: o estado do dispositivo, o identificador do dispositivo, o identificador do serviço e as variáveis de estado (o valor da leitura do dispositivo, por exemplo).

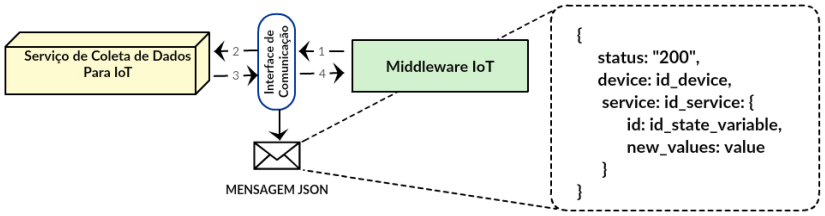

Figura 3. Sequência e formato das mensagens JSON.

5. Implementação do SDCD

Devido ao grande volume de dados gerados pelos ambientes IoT, a arquitetura da nossa proposta deve ser capaz de coletar, processar e armazenar um grande volume de dados. Para tal fim, foi definida uma arquitetura distribuída capaz de processar paralelamente diferentes tarefas do SDCD.

O Apache Kafka é um sistema de mensagens *publish-subscribe* distribuído, com código aberto, frequentemente utilizado como um *kernel* para arquiteturas que lidam com *stream* de dados (Garg, 2013). Foi projetado com as seguintes características: mensagens persistentes, alto rendimento, distribuição, suporte a múltiplos clientes e processamento em tempo real. O Apache Kafka tem três principais elementos (Kreps, 2011): tópico, produtores e consumidores. Um tópico é onde são publicadas as mensagens. Os produtores são processos que publicam ou criam mensagens para um tópico. Os consumidores processam as mensagens publicadas.

O Apache Storm é um *framework* para computação distribuída em tempo real tolerante a falhas (Storm, 2016). Um *cluster* Storm consta de um servidor Nimbus, Supervisores e o Zookeeper. O Nimbus é responsável pela distribuição do código pelo *cluster* e monitora o processamento. Um *cluster* Storm pode ter um ou mais nós supervisores, onde são executados os processos. Os nós supervisores se comunicam com o Nimbus através do Zookeeper (Zookeeper, 2016). O Zookeeper é um serviço de alto desempenho de coordenação distribuída do *cluster* Storm.

O Apache Cassandra é um sistema de banco de dados que busca atender os seguintes requisitos: alta disponibilidade, alto desempenho, confiabilidade e

escalabilidade para suportar o crescimento da quantidade de dados (Cassandra, 2016, Lakshman e Malik, 2009, Cockcroft et al., 2011). O Cassandra segue um modelo de dados orientado a colunas, mas, internamente, trabalha com o conceito de chave-valor. A escolha do Cassandra para o armazenamento dos dados se deu em função de sua estratégia para tratamento de séries temporais chamada '*DateTieredCompactionStrategy'* (Lu, B., *et al*., 2016), o que pode melhorar o desempenho do componente *Organização de Séries Temporais* do SDCD.

Por fim, consideramos também a plataforma Streamparse, que permite a execução de processamento paralelo e distribuído sobre *streams* de dados em tempo real via Apache Storm (Streamparse, 2016). O Streamparse oferta o estilo de processamento *map/reduce* em tempo real para *streams* de dados. Além disso, pode ser uma forma de garantir a escalabilidade dos processos Python com alto paralelismo. Tal ferramenta permite integrar o Apache Kafka, o Apache Storm e as bibliotecas Python - ferramentas que compreendem o núcleo da atual infraestrutura de processamento de dados.

Tomando como referência a Figura 1,no SDCD o componente *Captura de Dados* é composto por *n* processos consumidores implementados com o sistema de mensageria Kafka, chamados de *Kafkaspout*. O componente de *Filtragem de Dados* contempla dois estágios, a saber: JsonParser e filtros. Esses processos foram implementados utilizando o sistema de processamento em tempo real Apache Storm combinado com o Kafka. Os *bolts* JsonParser são processos que recebem as mensagens do consumidor KafkaSpout e dividem essas mensagens JSON em unidades menores. A filtragem das mensagens é implementada através dos processos *bolts* (Filtros), tendo como objetivo garantir a integridade de domínio dos valores das leituras dos sensores. Após a filtragem das mensagens, as mesmas são armazenadas no banco de dados Cassandra.

O componente *Obtenção de Metadados* foi implementado com a arquitetura Apache Storm/Kafka, onde os *bolts* metadados permitem criar metadados a partir das mensagens já filtradas. O processo metadados é o terceiro estágio dentro da topologia do Storm/Kafka. Desta forma, os 3 estágios da topologia - processos JsonParser, Filtros e Metadados - podem ser executados de forma paralela.

Por fim, para gravar os dados na base de dados foi montada uma chave composta pelo *timestamp* da leitura do dispositivo/sensor e o identificador do dispositivo/sensor. Esta chave garante a unicidade dos dados, uma vez que as leituras dos dispositivos junto com o seu identificador são únicas. Além disso, para aumentar a taxa de inserção dos dados no serviço proposto, foi usada uma abordagem de inserção *microbatching* (Shahrivari, 2014) que consiste em acumular a recepção de mensagens em um pequeno intervalo de tempo (2 segundos, por exemplo), para depois serem inseridas de forma conjunta, o que melhora a taxa de inserção no banco de dados. Os dados são gravados no banco de dados Cassandra em uma tabela de eventos com os seguintes atributos: Timestamp, id_dispositivo, estado, id_variavel, valor e localização.

6. Análise do SDCD

Nesta seção são apresentados os dados de análise do comportamento do SDCD em um ambiente IoT. Para essa análise o SDCD foi efetivamente implementado em um *cluster* computacional. Para a avaliação, realizamos uma simulação de um sistema de casas inteligentes utilizando o *middleware* UIoT (Cerqueira Ferreira et al., 2014).

6.1. Ambiente Computacional

Para a realização dos testes deste estudo de caso, a arquitetura proposta foi implementada em um *cluster* com 4 máquinas físicas, cada qual com as seguintes especificações: Intel Xeon 2.5 GHz de processador com 8 núcleos, 8GB de RAM, 500 GB de HD e 1Gb Ethernet.

O *cluster* Kafka foi configurado com 4 máquinas, o que significa que o fator de replicação máximo é 4. O fator de replicação é proporcional ao número de nós no *cluster* Kafka. Para garantir a escalabilidade horizontal do *cluster* foi necessário adicionar mais partições e particionar os dados em máquinas adicionais. O *cluster* Storm foi implementado com 4 máquinas: 1 máquina foi reservada como nó mestre (Nimbus), e os 3 nós restantes são nós escravos (supervisores ou trabalhadores). Finalmente, o *cluster* Cassandra foi configurado para 4 máquinas. É importante ressaltar que estes 3 *clusters* foram implementados nas 4 máquinas utilizadas nos experimentos.

6.2. Simulação de um Sistema de Casas Inteligentes

A simulação do sistema de casas inteligentes produz dados gerados por milhares de casas que possuem eletrodomésticos e dispositivos gerenciados pelo *middleware* UIoT. O processo de simulação considera que cada casa tem uma unidade de cada um dos sete dispositivos definidos na Tabela 1.

Para a simulação da troca de mensagens entre o *middleware* UIoT e o SDCD, os produtores Kafka criam 8 milhões de mensagens simultaneamente. Neste contexto, as mensagens criadas pelo processo de simulação possuem 160 bytes de tamanho. A Tabela 2 apresenta o número de casas e o volume de dados, estes nos diferentes intervalos de tempo (dia, mês e ano) que a simulação representa. O cálculo tem por base a leitura dos dispositivos a cada 10 segundos.

Tabela 1 – Dispositivos da Casa (Cerqueira Ferreira, 2014).

Dispositivo	Variáveis de estado (VS)	Valores permitidos das VS
Televisão	Canal	1 até 1000
	Volume	1 até 100
	Power	Ligado, desligado
Cortina	Movimento	Aberto, fechado
Forno micro-ondas	Power	Ligado, desligado
Cafeteira	Power	Ligado, desligado
Portão	Movimento	Aberto, fechado, parado
	Distância	2 até 700
Despertador	Power	Ligado, desligado
Som Estéreo	Power	Ligado, desligado
	Volume	1 até 100
	Estação	1 até 6

Tabela 2. Quantidade de dispositivos usada nas simulações.

Nro. Prod.	Nro. Casas	Dia	Mês	Ano
1	33600	0,29 TB	8,87 TB	107,92 TB
2	63492	0,56 TB	16,76 TB	203,96 TB
3	95238	0,84 TB	25,15 TB	305,92 TB
4	142857	1,26 TB	37,72 TB	458,91 TB

6.3. Resultados Obtidos

O cenário de simulação da criação de dados de leitura dos dispositivos de rede fez-se com o uso de produtores Kafka. No processo de simulação, foram utilizadas tanto a maneira síncrona como a assíncrona de criação de mensagens. Este processo foi testado de forma incremental, começando com um único produtor até chegar aos 4 produtores como mostrado na Tabela 2 e nos gráficos de desempenho da Figura 4. Se for criada uma linha de tendência em relação ao número de produtores e a quantidade de mensagem que a arquitetura consegue processar, seria possível observar que toda vez que é incrementado o número de produtores, o número de mensagens/casas que o serviço de coleta de dados consegue suportar é proporcional a esse incremento de mensagens/casas. Isto significa que o serviço de coleta de dados tende a ser escalável.

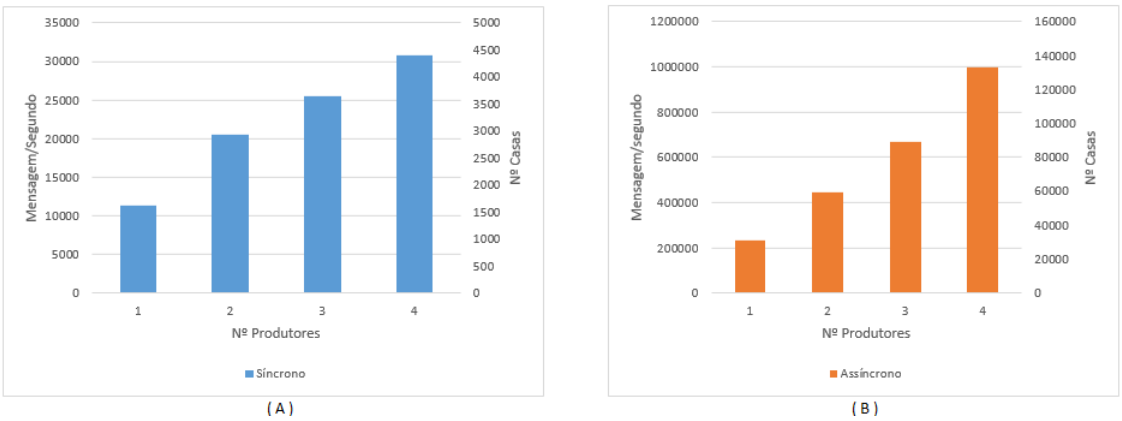

Figura 4. Desempenho dos Produtores de dados síncrono (a) e assíncrono (b).

A Figura 4 ilustra o comportamento dos produtores síncrono e assíncrono, ambos tendo a mesma tendência de desempenho, ou seja, ao aumentar o número de produtores, aumenta-se o número de mensagens produzidas. Entretanto, o número de mensagens processadas em um produtor assíncrono (Figura 4b) é muito maior do que a de um produtor síncrono (Figura 4a). Por exemplo, para dois produtores, o número de mensagens por segundo no caso síncrono é de 20.000 (2.934 casas) enquanto que, para o assíncrono, é de 400.000 (63.492 casas). A ingestão dos dados criados pelos produtores assíncronos é mais rápida devido ao fato de que toda mensagem enviada não precisa receber a confirmação de que a mensagem foi recebida, o que possibilita o envio de mensagens sem ter que esperar pela resposta para poder enviar a próxima mensagem.

Depois de criados pelos produtores, os dados são consumidos por uma instância KafkaSpout (consumidor) na topologia Storm. Esta entidade capta as mensagens, enviando-as ao processo JsonParser. Depois de analisados, os dados são armazenados no banco de dados Cassandra. Este processo é denominado inserção de dados do SDCD para IoT e foi medido com 1, 2 e 3 supervisores (trabalhadores) na topologia do Storm, apresentando os resultados mostrados na Figura 5.

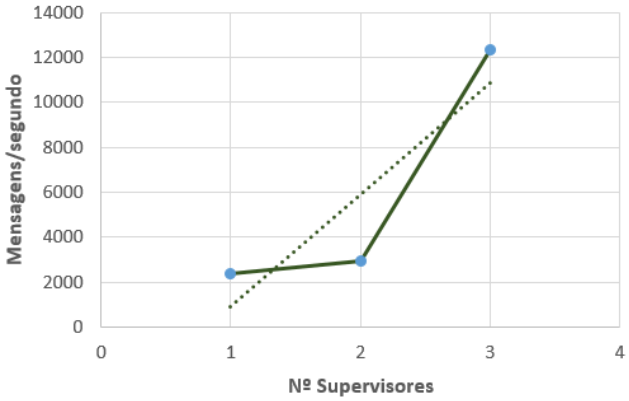

Figura 5. Inserção de dados no banco de dados Cassandra.

Para a simulação proposta, o SDCD conseguiu coletar e realizar o tratamento dos dados provenientes de dispositivos em 3.499 casas com a simulação síncrona e 142.857 casas para a simulação assíncrona. Vale ressaltar que o ambiente computacional da validação inclui apenas 4 máquinas com 8 GB RAM. Pelos resultados apresentados foi possível perceber uma tendência que mostra a boa escalabilidade da ingestão de dados da topologia, pois ao aumentar os nós produtores no *cluster* o serviço conseguiu processar uma maior quantidade de dados.

7. Conclusão

A IoT tem obtido cada vez mais a atenção da indústria e da academia. O uso de tecnologias distribuídas para o armazenamento e processamento dos grandes volumes de dados gerados pelos *middlewares* IoT parece ser a chave para resolver o problema da captura, organização e análise de dados em tempo real. Neste contexto, o SDCD para IoT aqui proposto constitui um ambiente distribuído com programação distribuída e paralela, com objetivo de suportar o processamento de grandes volumes de dados.

O SDCD aqui descrito consegue um bom desempenho na coleta de dados de mensagens do *middleware* UIoT. O Apache Kafka conseguiu suportar o volume de dados gerados pelo sistema de coleta apresentado, gerenciando as mensagens do *middleware* UIoT. Vários testes evidenciaram o desempenho do Kafka quando utilizado em parceria com o Storm para processamento da arquitetura proposta em tempo real.

Neste artigo, apresentam-se testes apenas com o *middleware* UIoT, mas em trabalhos futuros objetiva-se integrar a arquitetura proposta com outros *middlewares* IoT. Além disso, planeja-se avaliar o SDCD em um estudo de caso em ambiente real de dispositivos/sensores, a ser realizado em laboratório específico da Faculdade de Tecnologia da Universidade de Brasília (FT/UnB), onde já se têm instalados vários tipos de sensores para implantar um ambiente inteligente real.

Agradecimentos

Os autores agradecem à Secretaria Nacional do Consumidor do Ministério da Justiça (TED 001/2015) e à Defensoria Pública da União (TED 066/2016 DPU-FUB), pelo suporte ao presente trabalho.

Referências

Ashton, K. (2009). That 'internet of things' thing. RFiD Journal, 22(7), 97-114.

Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer networks*, 54(15), 2787-2805.

Bahga, A., and Madisetti, V. (2014). Internet of Things: A Hands-on Approach. 1ª. ed. VPT.

Botta, A., de Donato, W., Persico, V., & Pescapé, A. (2016). Integration of cloud computing and internet of things: a survey. *Future Generation Computer Systems*. 56(1), 684-700.

Cassandra (2016) "Cassandra". Disponível em: <http://cassandra.apache.org>. Acessado em Janeiro, 2016.

Ferreira, H. G. C., Canedo, E. D., De Sousa Júnior, R. T. (2014). A ubiquitous communication architecture integrating transparent UPnP and REST APIs. *International Journal of Embedded Systems*, 6(2-3), 188-197.

Chaqfeh, M., and Mohamed, N. (2012). "Challenges in middleware solutions for the internet of things." In *Collaboration Technologies and Systems (CTS)*, 2012 International Conference on, pages 21-26. IEEE.

Cockcroft, A., and Sheahan, D. (2011). "Benchmarking cassandra scalability on aws-over a million writes per second." Disponível em <http://techblog.netflix.com/2011/11/benchmarking-cassandra-scalability-on. html>. Acessado em janeiro de 2016.

Cooper, J., and James, A. (2009). Challenges for database management in the internet of things. *IETE Technical Review*, 26(5), 320-329.

Fersi, G. (2015). "Middleware for Internet of Things: A Study." In *International Conference on Distributed Computing in Sensor Systems*, pages 230-235. IEEE.

Fremantle, P. (2014). A reference architecture for the Internet of Things. *WSO2 White paper*.

Garg, N. (2013). *Apache Kafka*. Packt Publishing Ltd. Birmingham.

Ghosh, A., and Das, S. K. (2008). Coverage and connectivity issues in wireless sensor networks: A survey. *Pervasive and Mobile Computing*, 4(3), 303-334.

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7), 1645-1660.

Kreps, J., Narkhede, N., and Rao, J. (2011). Kafka: A distributed messaging system for log processing. In *Network Meets Database*, pages 1-7.

Lakshman, A., and Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*. 44(2), 35-40.

Le-Phuoc, D., Nguyen-Mau, H. Q., Parreira, J. X., and Hauswirth, M. (2012). A middleware framework for scalable management of linked streams. *Web Semantics: Science, Services and Agents on the World Wide Web*, 16:42-51.

Xively platform. Disponível em <http://xively.com>. Acessado em Janeiro, 2016.

Lu, B., & Xiaohui, Y. (2016). Research on Cassandra Data Compaction Strategies for Time-Series Data. In *Journal of computers*, 11(6), 504 – 512.

Mashal, I., Alsaryrah, O., Chung, T. Y., Yang, C. Z., Kuo, W. H., and Agrawal, D. P. (2015). Choices for interaction with things on Internet and underlying issues. *Ad Hoc Networks*, 28(1), 68-90.

Misra, G., Kumar, V., Agarwal, A., and Agarwal, K. (2016). Internet of Things (IoT)–A Technological Analysis and Survey on Vision, Concepts, Challenges, Innovation Directions, Technologies, and Applications. *American Journal of Electrical and Electronic Engineering,* 4(1), 23-32.

Misra, P., Simmhan, Y., and Warrior, J. (2015). Towards a Practical Architecture for the Next Generation Internet of Things. *arXiv preprint arXiv:1502.00797*.

Nan, C., Lee, Y., Tila, F., and Kim, D. H. (2016). Design and Implementation of Middleware Based on ID and IP Address for Actuator Networks. *International Journal of Smart home*. 10(1), 41-48.

Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2014). Context aware computing for the internet of things: A survey. *IEEE Communications Surveys & Tutorials*, 16(1):414-454.

Pires, P. F., Cavalcante, E., Barros, T., Delicato, F. C., Batista, T., and Costa, B. (2014). A platform for integrating physical devices in the Internet of Things. In *Embedded and Ubiquitous Computing, 12th IEEE International Conference*, pages 234-241.

Qin, W., Li, Q., Sun, L., Zhu, H., Liu, Y. (2011) RestThing: A Restful Web service infrastructure for mash-up physical and Web resources. In *Embedded and Ubiquitous Computing, 9th International Conference*, pages 197-204.

Rafiei, D., and Mendelzon, A. (1997). Similarity-based queries for time series data. *ACM SIGMOD,* 26(2):13-25.

Shahrivari, S. (2014). Beyond batch processing: towards real-time and streaming big data. *Computers*, 3(4):117-129.

Soldatos, J., Serrano, M., Hauswirth, M. (2012b) Convergence of Utility Computing with the Internet-of-Things. In *Innovative Mobile and Internet Services in Ubiquitous Computing, 2012 6th International Conference*, pages 874-879.

Soldatos, J., Kefalakis, N., Hauswirth, M., Serrano, M., Calbimonte, J. P., Riahi, M., and Skorin-Kapov, L. (2015b). OpenIoT: Open Source Internet-of-Things in the Cloud. In *Springer International Publishing*, pages 13-25.

Storm. Disponível em <http://storm.incubator.apache.org>. Acessado em Janeiro, 2016.

Streamparse. Disponível em <https://pypi.python.org/pypi/streamparse/2.1.3>. Acessado em Janeiro, 2016.

Teixeira, T., Hachem, S., Issarny, V., and Georgantas, N. (2011). *Service oriented middleware for the internet of things: A perspective*. Pages 220-229. Springer Berlin Heidelberg, Berlin, Heidelberg.

Zookeeper. Disponível em <http://zookeeper.apache.org>. Acessado em Janeiro, 2016.