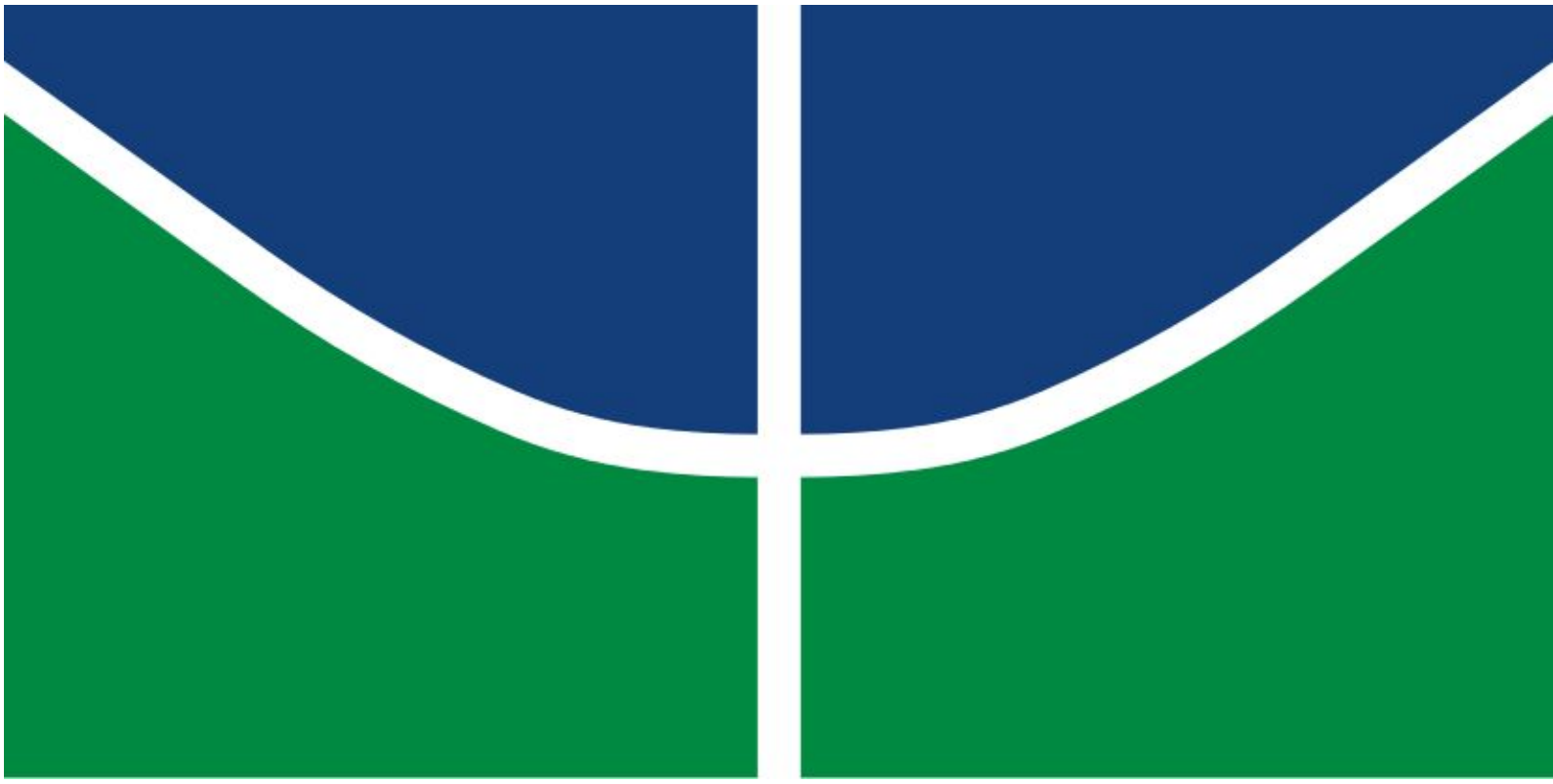

PROPOSIÇÃO DE UM MODELO E SISTEMA DE GERENCIAMENTO DE DADOS DISTRIBUÍDOS PARA INTERNET DAS COISAS - GDDIoT

RUBEN CRUZ HUACARPUMA

TESE DE DOUTORADO EM ENGENHARIA ELÉTRICA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROPOSIÇÃO DE UM MODELO E SISTEMA DE GERENCIAMENTO DE DADOS DISTRIBUÍDOS PARA INTERNET DAS COISAS - GDDIoT

RUBEN CRUZ HUACARPUMA

ORIENTADOR: RAFAEL TIMÓTEO DE SOUSA JUNIOR
COORIENTADORA: MARISTELA TERTO DE HOLANDA

TESE DE DOUTORADO EM ENGENHARIA ELÉTRICA

PUBLICAÇÃO: PPGENE.TD - Nº 120/2017
BRASÍLIA/DF: JUNHO – 2017

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROPOSIÇÃO DE UM MODELO E SISTEMA DE GERENCIAMENTO DE DADOS DISTRIBUÍDOS PARA INTERNET DAS COISAS - GDDIoT

RUBEN CRUZ HUACARPUMA

TESE DE DOUTORADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR.

APROVADA POR:

RAFAEL TIMÓTEO DE SOUSA JÚNIOR, Dr., ENE/UNB
(ORIENTADOR)

WILLIAM FERREIRA GIOZZA, Dr., ENE/UNB
(EXAMINADOR INTERNO)

CLARIMAR JOSÉ COELHO, Dr., UCG
(EXAMINADOR EXTERNO)

MÁRIO ANTÔNIO RIBEIRO DANTAS, Dr., UFSC
(EXAMINADOR EXTERNO)

Brasília, 27 de julho de 2017.

FICHA CATALOGRÁFICA

Huacarpuma, Ruben Cruz

Proposição de um Modelo e Sistema de Gerenciamento de Dados Distribuídos para Internet das Coisas - GDDIoT [Distrito Federal] 2017.

Xiii, 93p., 210x297 mm (ENE/FT/UnB, Doutor, Tese de Doutorado – Universidade de Brasília. Faculdade de Tecnologia.

Departamento de Engenharia Elétrica

1. Sistemas Distribuídos 2. Internet das coisa - IoT

3. Gerenciamento de dados 4. Big data

5. NoSQL

I. ENE/FT/UnB II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

Huacarpuma., R. C. (2017). Proposição de um Modelo e Sistema de Gerenciamento de Dados Distribuídos para Internet das Coisas - GDDIoT. Tese de Doutorado em Engenharia Elétrica, Publicação PPGENE.TD-120/17, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 93p.

CESSÃO DE DIREITOS

AUTOR: Ruben Cruz Huacarpuma

TÍTULO: Proposição de um Modelo e Sistema de Gerenciamento de Dados Distribuídos para Internet das Coisas - GDDIoT.

GRAU / ANO: Doutor / 2017

_____

Ruben Cruz Huacarpuma
SQN 408 BL G Apartamento do Térreo - Asa norte
Tel. 55-61-981272276 / rubencruzh@unb.br

AGRADECIMENTOS

Ao longo dos anos de trabalho que resultaram nesta tese, pessoas e instituições me ajudaram, ensinando e apoiando. Agora que alcancei um dos meus objetivos não poderia deixar de reconhecê-las.

Gostaria de agradecer, em primeiro lugar, a minha família que mesmo distante de mim teve o completo apoio e compreensão. O seu carinho e amor são fundamentais para a realização deste trabalho.

A toda equipe do LabRedes, meus colegas de estudo e de trabalho que tenho o prazer de trabalhar e conviver com pessoas tão maravilhosas e especiais.

Meu reconhecimento a minha professora e orientadora, Maristela Holanda. Para mim é uma imensa honra e orgulho tê-la como orientadora por mais de sete anos (desde o mestrado). Pouco são tão privilegiados como eu por ter tido a sorte de conviver com uma pessoa tão generosa, dedicada, eficiente, objetiva, diligente e ante tudo apaixonada pelo que faz.

Em especial, agradeço ao Professor Rafael Timóteo, meu orientador, pela confiança e compartilhamento do grande conhecimento que possui, agradeço por sua cumplicidade e responsabilidade direta na construção desta Tese.

Finalmente, faço questão de agradecer de coração a todas as pessoas que torceram ou intercederam por mim, mesmo que de forma anônima ou discreta. Fazendo eco das palavras de Vinícius de Moraes, "Você não faz amigos, você os reconhece". A todos esses amigos e amigas meu muito obrigado de coração.

Durante o desenvolvimento desta tese, tive o apoio do Ministério do Orçamento, Desenvolvimento e Gestão (TED 26/2012 Projeto SEGEP CGAUD SIGA - Sistema de Inteligência e Gestão de Auditoria) e do Ministério da Justiça (TED 52/2014 SRJ Projeto Atlas do Acesso à Justiça – Ontologia do Sistema de Justiça, e TED 001/2015 SENACON Prospecção de tecnologias da informação aplicadas às políticas públicas de proteção à defesa do consumidor), órgãos aos quais agradeço sobremaneira.

RESUMO

PROPOSIÇÃO DE UM MODELO E SISTEMA DE GERENCIAMENTO DE DADOS DISTRIBUÍDOS PARA INTERNET DAS COISAS - GDDIoT

Autor: Ruben Cruz Huacarpuma
Orientador: Professor Dr. Rafael Timóteo de Sousa Junior
Programa de Pós-graduação em Engenharia Elétrica
Brasília, 27 de julho de 2017.

O desenvolvimento da Internet das Coisas (IoT) levou a um aumento do número e da variedade de dispositivos conectados à Internet. Dispositivos tais como 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 objetos conectados, uma gama de aplicações tem sido desenvolvida e implantada, incluindo aquelas que lidam com grandes volumes de dados. Nesta tese, apresenta-se uma proposta e implementação de um modelo para o gerenciamento de dados em um ambiente de IoT. Este modelo contribui com a especificação das funcionalidades e a concepção de técnicas para coletar, filtrar, armazenar e visualizar os dados de forma eficiente. Uma característica importante deste trabalho é capacidade de integrar diferentes *middlewares* IoT. A implementação deste trabalho foi avaliada através de diferentes estudos de casos sobre cenário de sistemas inteligentes: Sistema de Casas Inteligentes, Sistema de Transporte Inteligente e a comparação do GDDIoT com *middleware* IoT.

ABSTRACT

Author: Ruben Cruz Huacarpuma
Supervisor: Professor Dr. Rafael Timóteo de Sousa Junior
Programa de Pós-graduação em Engenharia Elétrica
Brasília, 27 July 2017

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. Smart objects such as 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 smart objects, a broad range of applications has been developed and deployed, including those dealing with massive volumes of data. In this thesis, it is proposed a data management approach and implementation for an IoT environment, thus contributing with the specification of functionalities and the conception of techniques for collecting, filtering, storing and visualization data conveniently and efficiently. An important characteristics of this work is to enable multiple and distinct middleware IoT to work together in a non-intrusive manner. The corresponding implementation of this work was evaluated through different case studies regarding a smart system scenarios: Smart Home System, Smart Transportation System and comparison between GDDIoT and an IoT middleware.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ACRÔNIMOS

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
CSV	Comma Separated Values
GDDIoT	Gerenciamento de Dados Distribuído para IoT
GPS	Global Positioning System
HDFS	Hadoop Distributed File System
HTTP	Hypertext Transfer Protocol
IoT	Internet of Thing
JSON	JavaScript Object Notation
LPS	Local Positioning System
M2M	Machine-to-Machine
MQTT	Protocol Message Queue Telemetry Transport
NoSQL	Not Only SQL
RFID	Radio Frequency Identification
SCADA	Supervisory Control and Data Acquisition
SI	Sistema Internacional de Unidades
SQL	Structured Query Language
TCP	Transmission Control Protocol
UIoT	Universal IoT
UUID	Universal Unique Identifier
VM	Virtual Machine
WSN	Wireless Sensor Networks
XML	eXtensible Markup Language

1 – INTRODUÇÃO

A crescente prevalência de objetos inteligentes em todos os lugares (Atzori *et al*., 2010), tanto nos mundos real, digital e virtual convergem para criar ambientes que tornem inteligentes as cidades, os transportes, as indústrias, as roupas (por exemplo, *Google Glass* e *iWatch*) e tantos outros serviços. Tais dispositivos conectam-se automaticamente sem a interação humana, possuindo sensores transmissores de dados, possibilitando a construção de grandes redes de comunicação de dados. A essa rede, composta por bilhões de dispositivos conectados, denominou-se Internet das Coisas (Misra *et al*., 2016). A ideia de conectar objetos é discutida desde 1991, quando a conexão TCP/IP e a Internet se popularizaram. Em 1999, Kevin Ashton do MIT (*Massachusetts Institute of Technology*) propôs o termo "Internet das Coisas" e desde então esta expressão começou a se popularizar por todo o mundo. Prevêem-se mais de 50 bilhões de dispositivos interconectados até 2020 (Mashal *et al*., 2015; Misra *et al*., 2015), considerando a existência de mais de 4 bilhões de usuários de telefones inteligentes com a capacidade de acessar à Internet. Os ambientes IoT estão composto por diferentes tipos de dispositivos acarreando diferentes formatos de dados.

O processamento dos dados dos ambientes IoT possui algumas propriedades, incluindo o volume, a variedade e a velocidade dos dados. Tais ambientes requerem o processamento de grandes quantidades de dados coletados de vários objetos inteligentes (Gubbi et al., 2013; Xu, He et al., 2014). Assim, a IoT torna-se um paradigma com muitos desafios, uma vez que permite a disponibilidade de diversos objetos – dispositivos – a qualquer hora e em qualquer lugar enviar seus dados. Tais dispositivos deveriam ser capazes de modificar seu comportamento segundo o estado do ambiente e os pedidos dos usuários.

Enquanto a IoT oferece inúmeras potencialidades e oportunidades, a administração do grande volume de dados produzido em tal ambiente continua sendo um desafio. Como exemplo disso, considere-se um sistema de monitoramento de transporte, com diferentes sensores para luzes, sinalizações e postes, onde a cada cinco segundos cada sensor envia dados com 160 *bytes* de tamanho. Em apenas um único dia, o volume de dados produzidos por cada sensor seria 2,64 *megabytes* (17.280 registros); em um mês, 81,74 *megabytes* (535.680 registros); e, em um ano, 962.40 *megabytes* (6.307.200 registros). Assim, se uma rede de dispositivos detiver 10 mil sensores, poderá gerar mais de 9,6 *terabytes* (63 bilhões

de registros) de dados por dia. Nesta classe de aplicação, o número de sensores tende a crescer diariamente. Portanto, o desenvolvimento de ferramentas e abordagens para a administração dos dados gerados é fundamental.

1.1 – MOTIVAÇÃO

Recentes avanços tecnológicos têm possibilitado o surgimento da IoT, tais como: redes de sensores sem fio, comunicação móvel e computação ubíqua. No entanto, grandes são os desafios a serem superados em relação ao gerenciamento dos dispositivos e ao gerenciamento dos dados gerados pelos ambientes IoT. Tal ambiente é decorrente da alta heterogeneidade inerente da diversidade das tecnologias de *hardware* e *software* e a geração de grandes volumes de dados a altas velocidades. Neste contexto, plataformas de *middleware* têm surgido como soluções promissoras para prover tal interoperabilidade e gerenciar a crescente variedade de dispositivos associados a aplicações, bem como o consumo de dados por parte dos usuários finais (Teixeira et al., 2011).

Neste contexto, um *middleware* IoT é um sistema que gerencia eficazmente os vários componentes heterogêneos que compõem um ambiente IoT (Fersi, 2015; Razzaque et al., 2016). O *middleware* é uma camada de *software* entre a camada física de dispositivos e suas interfaces e a camada de aplicação que integra a lógica e o controle de uma instância IoT. Para facilitar a integração e comunicação de componentes heterogêneos (Ghosh et al., 2008), é necessário que o *middleware* IoT proveja um conjunto de abstrações de programação, tais como: serviços de busca, acesso e gerenciamento de dispositivos, entre outros.

Para esta classe de sistemas, a transmissão de dados em grande escala através da Internet é um requisito padrão, dada a necessidade de compartilhamento de dados de modo pervasivo na rede. Gerenciar esse grande volume de dados ainda é um desafio. Além disso, cada *middleware* pode fazer uso de um formato de dados específico, como, por exemplo, o formato XML em algum *middleware* e o formato JSON em outros. Consequentemente, é desafiador integrar os dados provenientes de diferentes ambientes IoT com outros formatos que surgem em grande volume e alta variedade. Tal desafio requer melhores abordagens sobre como coletar, processar e armazenar os dados. Do contrário, cada aplicação teria que lidar com as questões relacionadas.

Para o gerenciamento de grande volume de dados dos ambientes IoT, os

middlewares IoT implementam coletores de dados específicos, que fazem uso de diferentes arquiteturas de armazenamento, desde sistemas de gerenciamento de banco de dados relacional centralizado até sistemas distribuídos NoSQL, mas todos fortemente acoplados à arquitetura do *middleware* IoT. Por exemplo, os *middlewares* EcoDif (Delicato et al., 2013), Xively (Bahga et al., 2014), OpenIoT (Soldatos et al., 2012), RestThing (Qin et al., 2011; Soldatos et al., 2012), WSO2 (Fremantle, 2014) e Kaa (Technologies, 2017) implementam seus próprios coletores de dados. Em uma estrutura tradicional de um sistema IoT, um ou mais *middlewares* que gerenciará(ão) os dispositivos IoT devem enviar dados para um servidor de aplicações da Internet, sendo o servidor de aplicação e os dispositivos gerenciados pertencentes à mesma organização. Porém, prevê-se que futuros sistemas IoT envolverão aplicações para acessar dados coletados de dispositivos pertencentes a diferentes indivíduos e organizações. Em tais sistemas, os serviços de gerenciamento de dados amplamente distribuídos e confiáveis serão necessários para lidar com a heterogeneidade e apoiar a análise de dados e os processos de aprendizagem para o IoT.

Diante do exposto, o problema de pesquisa que orienta esta tese é a falta de mecanismos para o gerenciamento especializado de dados para Internet das Coisas, devido ao forte acoplamento existente dos diferentes *middlewares* IoT e a falta da integração dos dados desses sistemas. Neste contexto, a presente pesquisa tem por objetivo propor o GDDIoT (Gerenciamento de Dados Distribuídos para Internet das Coisas), a fim de coletar, processar e armazenar os dados provenientes de diferentes *middlewares* IoT. O serviço de dados proposto pode se adaptar a diferentes níveis de paralelismo, conforme o ambiente IoT. Tal serviço é composto por quatro componentes principais, a saber: 1) interface de comunicação; 2) coleta de dados; 3) agregação de dados; e, 4) visualização de dados. O componente da interface de comunicação deve suportar múltiplas conexões de *middlewares* IoT. O componente de coleta deve ter a capacidade de capturar eficientemente dados produzidos a partir de uma grande rede de dispositivos. O componente de agregação de dados é importante para apresentar a capacidade de sumarizar eficientemente enormes volumes de dados em tempo real. O componente de visualização de dados aponta, de modo gráfico, os dados IoT. Este serviço de dados deve enfrentar vários desafios, tais como: armazenamento dos dados, evitar gargalos de processamento, lidar com a heterogeneidade de dispositivos/sensores e apresentar alto rendimento. Para

validar o serviço proposto, prosseguiram-se com a simulação de três projetos, quais sejam: Sistema de Casas Inteligentes, Sistema de Transporte Inteligente e Coletor de dados do *middleware* Kaa.

A heterogeneidade dos dados IoT é um dos problemas que devem ser superados pelo GDDIoT. Outra característica importante dos dados IoT é a comunicação em tempo real das informações dos objetos que fazem parte dos sistemas IoT (Yasumoto et al., 2016a). Devido ao grande número de dispositivos gerando dados em tempo real ou perto do tempo real, tem-se o desafio de lidar com o tratamento dos dados produzidos em alta velocidade. Frequentemente, as aplicações IoT demandam respostas imediatas, onde o tratamento dos dados em tempo real é um requisito importante.

Com o GDDIoT proposto nesta tese, o desenvolvimento de novos *middlewares* IoT podem fazer uso deste serviço especializado para o gerenciamento dos seus dados, ao invés de desenvolver um novo componente para o gerenciamento dos dados específicos do *middleware* IoT implementado.

A justificativa do trabalho apresentado nesta Tese pretende dar suporte ao gerenciamento em tempo real de grande volume de dados heterogêneos, de fluxo contínuo e geograficamente dispersos criado por milhões de diversos dispositivos que enviam periodicamente observações sobre certos fenômenos monitorados ou relatam a ocorrência de eventos anormais ou de certos eventos de interesse. Em resumo, pretende-se resolver vários desafios, quais sejam: integrar os dados de diferentes *middlewares* IoT (desafio da Variedade); armazenar dados de todos os eventos (desafio da Velocidade e Volume); executar consultas sobre os eventos armazenados (desafio da Velocidade e Volume); realizar análises sobre os dados para a obtenção de conhecimento; e, apresentação dos resultados de modo gráfico.

A presente pesquisa pode ser considerada como uma pesquisa aplicada (Burstein et al., 1999; Neuman, 2002), uma vez que visa ajudar os profissionais a melhorar sua compreensão de um problema de domínio específico: o gerenciamento de dados IoT. E ainda, pode ser tratada como uma pesquisa de formulação (Nunamaker et al., 1990), devido à sua aplicabilidade em uma pesquisa onde um problema é identificado (gerenciamento de dados IoT), e logo adquirir conhecimentos para aumentar a familiaridade com a área problemática ou domínio de estudo. Além disso, também poderia ser considerada como uma pesquisa de desenvolvimento, com viés de crescimento

tecnológico (Gregg et al., 2001).

1.2 – OBJETIVOS

Desenvolver um serviço distribuído de gerenciamento de dados para IoT que seja capaz de administrar os dados provenientes de várias redes de dispositivos/sensores em um ambiente IoT. Em particular, os cenários a serem avaliados incluem a coleta de dados, o processamento e a visualização dos dados advindos de diferentes *middlewares* IoT.

1.2.1 – Objetivos Específicos

Para alcançar o objetivo geral desta proposta, os seguintes objetivos específicos foram definidos:

- Definir a arquitetura do GDDIoT;
- Desenvolver os módulos constituintes da arquitetura do GDDIoT proposta e especificar suas interações:
  1. Componente de comunicação com *middlewares* IoT;
  2. Componente de coleta, processamento e consulta de dados adequados para IoT;
  3. Componente de agregação de dados;
  4. Componente de visualização de dados;
- Definir os requerimentos de *software* e *hardware* da arquitetura do GDDIoT e configurar o ambiente de desenvolvimento; e
- Avaliar o desempenho do GDDIoT segundo os resultados alcançados;

1.3 – CONTRIBUIÇÕES DO TRABALHO

Nas linhas que se seguem tem-se por norte as seguintes contribuições:

- Apresentação de um modelo genérico de serviço distribuído de gerenciamento de dados IoT capaz de comportar um grande volume de dados. Atualmente, os dados gerados pelos ambientes IoT frequentemente são administrados por componentes fortemente acoplados à arquitetura dos *middlewares* IoT que gerenciam os ambientes IoT. Esta é a arquitetura predominante das plataformas IoT, que detêm componentes para a coleta, o processamento e a visualização dos dados. Tais componentes dão suporte a suas tarefas, mas não de forma especializada. Neste contexto, um serviço para o gerenciamento de dados especializado para a coleta, o processamento e a

visualização dos dados dos ambientes IoT é de grande importância. Além disso, o GDDIoT pretende comportar a massiva quantidade de dados – o que é possível devido ao modelo distribuído para o armazenamento dos dados. Assim, os *middlewares* IoT existentes ou novos *middlewares* IoT podem fazer uso do serviço especializado para o gerenciamento dos dados IoT; logo, as plataformas IoT podem se preocupar com outras funcionalidades no que tange o gerenciamento de objetos (dispositivos/sensores, por exemplo).

- Modelo orientado a serviço. O GDDIoT pretende ser um serviço de baixo acoplamento e reutilizável, devido à definição do GDDIoT que funcione de modo não intrusivo dentro dos sistemas que fazem parte de um ambiente IoT. Além disso, o GDDIoT pode ser utilizado por novos sistemas IoT, tais como: novos *middlewares* IoT ou *middlewares* IoT já existentes que requerem um serviço especializado para o gerenciamento de dados IoT.
- O GDDIoT pretende dar suporte a múltiplos ambientes IoT simultaneamente. Diversos ambiente IoT podem-se se conectar com o GDDIoT de modo simultâneo, logrando lidar com a heterogeneidade dos dados gerados pelos diferentes ambiente IoT.
- Apresentação do modelo conceitual do GDDIoT. Define-se um modelo conceitual com base no modelo Lambda. O modelo GDDIoT pretende lidar com *middlewares* IoT heterogêneos, definindo diferentes componentes para a integração de dados. Dentro de cada um dos componentes existem módulos especializados para lidar com a complexidade dos dados através do uso de metadados. O modelo aponta detalhadamente como os componentes interagem entre eles.
- O modelo do GDDIoT pretende atender os requisitos de processamento em lote e em tempo real de forma simultânea. Na arquitetura GDDIoT, pretende-se integrar as funcionalidades do processamento em lote e em tempo real em um único mecanismo de fluxo projetado para o processamento em lotes e em tempo real – o que é possível graças às características temporais dos dados IoT. Tal integração supera a complexidade da programação em duas plataformas diferentes distribuídas, como a camada de velocidade e a camada de lote, a fim de produzir resultados em lote e em tempo real compatíveis.
- Apresentação da implementação de uma proposta que faz uso das arquiteturas

distribuídas e paralela para o processamento, o armazenamento e a visualização dos dados em tempo real gerados pelos *middlewares* IoT define-se a implementação do GDDIoT fazendo uso de tecnologias que dão suporte ao modelo para o processamento distribuído e paralelo. Tal ação possibilita a configuração do nível de paralelismo segundo a demanda para o processamento e armazenamento dos dados. Além disso, a arquitetura distribuída logra lidar com a massiva quantidade dados, sendo possível o processamento dos dados em tempo real, a fim de satisfazer as demandas das novas aplicações que precisam de respostas imediatas aos requerimentos dos usuários finais, tais como: aplicações de saúde, de transporte, de segurança, entre outros. Depois que o serviço foi implementado, a viabilidade e correção do serviço são demonstradas pelo desenvolvimento da lógica do serviço – o que auxilia no refinamento das ideias, adiciona rigor e generaliza o serviço. Além disso, evita ou reduz a possibilidade de erro e mal funcionamento dos componentes do serviço.

- Apresentação de uma opção de implantação para o serviço proposto. A implementação do GDDIoT serve como uma 'prova de conceito' de que o serviço aqui proposto pode ser, de fato, implementado, evidenciando que a estratégia/método para o gerenciamento de dados advindos de diferentes ambientes IoT atende aos requisitos da administração de dados IoT.

1.4 – METODOLOGIA

Para alcançar os objetivos desta tese, é necessário construir um projeto de pesquisa para determinar as diferentes fases da pesquisa. Este tese encaixa-se no método de pesquisa *Design Science* (Nunamaker et al., 1990). Este tipo de pesquisa é fundamental principalmente dentro da linha de pesquisa Gestão de Informação e do conhecimento, em que muitas pesquisas prescrevem artefatos como modelos e sistemas de informação e onde as metodologias mais clássicas têm alcance limitado. Segundo a metodologia de pesquisa *Design Science*, as atividades de pesquisa consistem em seis fases principais: construir estrutura conceitual, estabelecer a formalização da estrutura, desenvolver arquitetura do sistema, analisar e projetar o sistema, construir o protótipo do sistema e observar e avalie o sistema.

Neste contexto, o presente trabalho segue estas seis fases que são detalhadas a

seguir:

- Fase 1: Definição de arquitetura para o gerenciamento de dados IoT no modo de GDDIoT.
- Fase 2: Detalhamento da arquitetura conceitual do GDDIoT na forma dos diferentes componentes constituintes do GDDIoT.
- Fase 3: Desenvolvimento da arquitetura conceitual do GDDIoT.
- Fase 4: Analisar e projetar a arquitetura do GDDIoT.
- Fase 5: Prototipação da arquitetura do GDDIoT.
- Fase 6: Após a prototipação da arquitetura observa-se a funcionalidade do protótipo e a avaliação através de diferentes estudos de casos.

1.5 – ESTRUTURA DO TRABALHO

Para facilitar o entendimento da presente Tese, os demais capítulos estão organizados conforme se segue:

Capítulo 2 apresenta a revisão bibliográfica sobre a temática "Internet das Coisas", bem como outros temas relacionados à administração de dados gerados pelos ambientes IoT. Além disso, tem-se a discussão da administração de *big data* para IoT e os requerimentos das bases de dados para IoT;

Capítulo 3 apresenta os trabalhos relacionados ao gerenciamento de dados nos ambiente IoT.

Capítulo 4 apresenta o serviço distribuído de gerenciamento de dados para IoT, bem como a definição abstrata do serviço.

Capítulo 5 apresenta a implementação de todos os componentes do serviço de dados.

Capítulo 6 apresenta três estudos de casos para a validação da abordagem, a saber: sistema de casa inteligentes, o sistema de transporte inteligente e avaliação do coletor de dados de um *middleware*.

Capítulo 7 apresenta os resultados alcançados pelos estudos de casos.

Capítulo 8 apresenta a discussão dos resultados obtidos.

Capítulo 9 apresenta as conclusões deste trabalho, sinalizando algumas perspectivas possíveis, o fechamento dos resultados obtidos e os caminhos futuros para a sequência da pesquisa em questão.

2 – REFERENCIAL TEÓRICO

Neste capítulo são apresentados os conceitos básicos relacionados a IoT (*Internet of Things*), administração de grandes volumes de dados, bem como às tecnologias de banco de dados aplicadas na administração de dados de ambientes IoT.

Na Seção 2.1 tem-se a definição de IoT. Na Seção 2.2 são analisados os diferentes tipos de dados IoT. A Seção 2.3 apresenta uma revisão dos conceitos relacionados ao gerenciamento de grandes volumes de dados. Na Seção 2.4 são apresentados os requerimentos referentes ao banco de dados para *middlewares* IoT. Na Seção 2.5 tem-se a arquitetura Lambda, usada como base para a implementação do GDDIoT foco desta tese. Finalmente, a Seção 2.6 apresenta as tendências para o processamento de dados IoT.

2.1 – Internet das Coisas

O termo "Internet das Coisas" foi mencionado pela primeira vez por Kevin Ashton em uma apresentação de 1999 (Ashton, 2011). Semanticamente, o termo em questão significa "uma rede mundial de objetos interconectados que são endereçados de forma única, baseada em protocolos de comunicação padrão" (Bassi et al., 2008). Neste contexto, uma "coisa" na IoT pode ser um objeto que faz uso de dispositivos inteligentes, um automóvel equipado com sensores que avisa ao motorista que alguns componentes não estão funcionando, um animal com um *chip* de sensoriamento ou qualquer objeto que possua um método de identificação tais como um endereço IP e que possa transmitir dados fazendo uso da rede de comunicação de dados (Zhu, 2015). Observa-se então que a questão de um elevado número de objetos (heterogêneos) envolvidos nos processos de identificação, representação e armazenamento das trocas de informação tornar-se um tema desafiador.

A definição de IoT deriva da perspectiva de "coisa orientada a objeto". Nesta, a "coisa" é uma variedade de objeto com presença pervasiva, como, por exemplo, um item que pode ser identificado por radiofrequência (RFID – *Radio Frequency Identification*), sensores, celulares, entre outros (Hamraz, 2013). Tal conceito pode se estender para coisas identificadas ou conectadas por *WiFi*, *bluetooth*, ou outros protocolos de comunicação de dados.

Tem-se ainda a IoT orientada a semântica (Atzori et al., 2010). Uma abordagem voltada à semântica é uma perspectiva que tenta levar a IoT mais perto da realidade. Os

temas relacionados à representação, ao armazenamento, à interconexão, à busca e à organização de informação gerada pela IoT são aprofundados nesse conceito. Neste contexto, a semântica (*Linked Data*, ontologias, entre outros) possui papel importante para modelar, descrever e analisar os dados gerados pela IoT (Toma et al., 2009).

Desde o ponto de vista técnico, a Internet das Coisas não é resultado de uma única tecnologia inovadora. Vários desenvolvimentos técnicos complementares proporcionam capacidades que, juntas, ajudam a preencher as lacunas entre o mundo virtual e o físico (Huang, 2013; Mattern et al., 2010). Assim, tais recursos incluem:

- Comunicação e cooperação: os objetos têm a capacidade de interconectar com recursos da que a Internet provê fazendo uso de dados, serviços e atualizar seu estado;
- Endereçamento: os objetos podem localizar-se e endereçar-se através de serviços de descoberta e consulta, sendo consultados e configurados remotamente;
- Identificação: os objetos são identificados de modo único;
- Sensoriamento: os objetos coletam informações sobre seu ambiente com sensores, registrando, encaminhando a informação ou reagindo diretamente a ele;
- Atuação: os objetos contêm atuadores para manipular seus ambientes (transformando sinais elétricos em movimentos mecânicos, por exemplo);
- Processamento de informação embebido: os objetos inteligentes possuem um processador ou microcontrolador, além da capacidade de armazenamento; tais recursos podem ser utilizados para processar e interpretar as informações dos sensores, ou para dar aos objetos uma memória de como eles foram utilizados;
- Localização: os objetos inteligentes estão cientes da sua localização física ou podem ser localizados; e
- Interfaces para usuário: os objetos inteligentes podem se comunicar com os indivíduos de modo apropriado, de forma direta ou indireta (via telefones inteligentes, por exemplo).

2.1.1 – *Middleware* IoT

O *middleware* é uma camada de *software* entre a camada física e a camada de aplicação (Figura 2.1) que proporciona um conjunto de abstrações, de modo a facilitar a integração e comunicação de componentes heterogêneos (Fersi, 2015). Um *middleware* abstrai as

complexidades dos sistemas ou hardware, permitindo o desenvolvedor da aplicação focar todos seus esforços nas tarefas para serem resolvidas, sem as distrações ao nível do sistema ou hardware, tais como as complexidades relacionada às questões de comunicação. O 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; Merk et al., 2001). Outro objetivo é proporcionar funções críticas, tais como: serviço de descoberta, gerenciamento de dados, controle de acesso, entre outros (Perera, Zaslavsky, Christen, & Georgakopoulos, 2014).

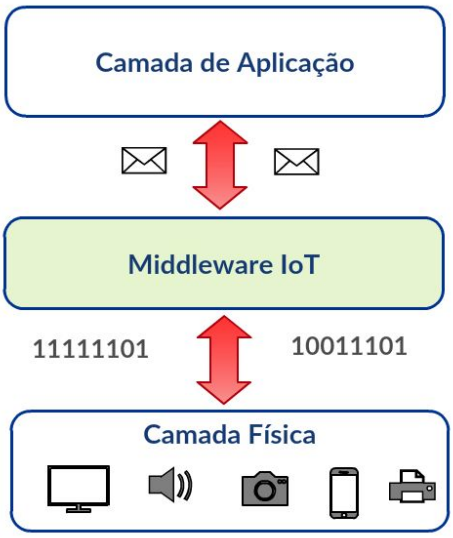

Figura 2.1. Sistema IoT composto por camadas.

O fluxo dos dados nos sistemas IoT pode ser composto por três camadas: camadas de aplicação, *middleware* IoT e a camada física. O fluxo de dados neste ambiente acontecem por meio do middleware IoT. O *middleware* IoT recebe as mensagens da camada física, executa processamento e encaminha para a aplicação. Assim como também, a camada de aplicação se comunica com o *middleware* para enviar mensagens à camada física.

A comunicação entre as diferentes camadas que compõem um sistema IoT normalmente é feita por meio de mensagens. Essas podem ser do tipo requisição ou resposta. Uma mensagem de requisição dá a ordem para a execução de uma ação, e uma mensagem de resposta envia a resposta de uma requisição. O formato das mensagens depende de cada *middleware*. Frequentemente, o padrão JSON tem sido usado no ambiente

de IoT (Bray, 2014; Nurseitov et al., 2009). Por exemplo, a camada de aplicação recebe uma mensagem JSON de requisição, para logo encaminhá-la ao *middleware*, até chegar na camada física da arquitetura IoT. A camada física executa a requisição. Após a execução, uma mensagem JSON é gerada como resposta e encaminhada para a camada de aplicação. O padrão JSON tem seu uso contínuo pelos *middlewares* IoT para lidar com a heterogeneidade dos dados geradas pela IoT.

A característica de escalabilidade nos *middlewares* é algo essencial. Basicamente, a escalabilidade se refere à capacidade de um sistema aumentar a sua capacidade total sob o aumento de carga quando os recursos de *hardware* são adicionados, suportado por protocolos de comunicação padrão e interoperáveis. Mesmo fazendo uso de padrões de comunicações, não se tem conhecimento de uma padronização para um *middleware* para computação pervasiva (Chaqfeh et al., 2012).

Os *middlewares* existentes podem ser agrupados com base na arquitetura das abordagens (Razzaque et al., 2016): em eventos, orientado a serviços, VM (*Virtual Machine*), em agentes, em tupla-espaço, orientado a base de dados e em aplicações específicas. Cada um destes middlewares é descrito a seguir.

Os *middlewares* baseados em eventos fazem uso das mensagens que são enviados por aplicações (produtores) para serem recebidos por outras aplicações (consumidores). Os *middlewares* orientados a mensagens são um tipo de *middleware* baseado em eventos. Neste modelo, a comunicação tem por norte as mensagens. Tipicamente, os *middlewares* com base em eventos fazem uso do padrão *publish/subscribe*.

O *middleware* orientado a serviço (SOA – *Service Oriented Architecture*) tem por base a arquitetura orientada a serviços. Este tipo de abordagem é caracterizado pela neutralidade tecnológica, baixo acoplamento, reutilização do serviço, composição de serviço e descoberta de serviço. Tal abordagem pode dar suporte na implantação, publicação, descoberta e acesso ao serviço em tempo de execução.

Os *middlewares* orientados a VM proveem suporte para a programação criando um ambiente seguro de execução para usuários de aplicações virtualizando a infraestrutura. Desta forma, as aplicações são divididas em pequenos módulos separados, os quais serão injetados e distribuídos através da rede.

A abordagem fundamentada em agentes, divide as aplicações em módulos para facilitar a distribuição através da rede usando agentes móveis. Esses mantêm seu estado de

execução ao migrar de um nó para outro. Isto facilita a criação de sistemas descentralizados tolerante a falhas (Mamei et al., 2006).

Nos *middlewares* baseado em tupla-espaço, cada membro da infraestrutura mantém uma estrutura de tupla-espaço local. Uma tupla-espaço é um repositório de dados que pode ser acessado de forma concorrente (Mottola et al., 2006). Todas as tupla-espaços formam uma federação de tupla-espaço em um *gateway*. Essa abordagem é adequada para dispositivos móveis em uma infraestrutura de IoT, pois podem compartilhar dados de forma transitória dentro das restrições de conectividade do *gateway* (Razzaque et al., 2016).

No *middleware* orientado a banco de dados, uma rede de sensores é vista como um sistema de banco de dados relacional virtual. Onde a camada de *middleware* funciona como um processador de consultas centralizado e cada objeto que faz parte da rede possui seu processador de consulta que encaminha no *middleware*. Uma aplicação pode consultar essas informações usando uma linguagem de consulta parecida o SQL, o que permite elaborar consultas complexas (Azzarà et al., 2013).

Uma abordagem de aplicação-específica (isto é, orientada por aplicação) para *middleware* foca-se no suporte do gerenciamento de recursos para um destino específico ou o domínio da aplicação. Isto é possível mediante a implementação de uma arquitetura ajustada a rede ou a uma infra-estrutura com base nos requisitos da aplicação ou domínio da aplicação.

2.2 – DADOS IoT

Os dados gerados pelos ambientes IoT representam fenômenos do mundo físico. Estes dados, frequentemente, originados a partir de dispositivos/sensores são organizados como séries temporais devido a suas características temporais (Abu-Elkheir et al., 2013; Cecchinel et al., 2014; Vongsingthong, 2015). A característica temporal é peça imprescindível no mundo IoT, uma vez que o atributo temporal (*timestamp*) está presente nos dados gerados pelos ambientes IoT. Uma série temporal é definida como 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). Assim, em um ambiente IoT os dados são coletados em intervalos de tempo variável, sendo na maioria dos casos

regulares ao longo de um determinado período. Por exemplo, o registro da leitura de temperaturas máximas e mínimas diárias em uma cidade.

Existem dois tipos de séries temporais: estacionárias e não estacionárias (da Silveira Bueno, 2008). A primeira são aquelas que flutuam em torno de uma mesma média ao longo do tempo e a segunda aquela em que a média varia ao longo do tempo. Neste contexto, os dados IoT têm características de séries estacionárias e não estacionárias, uma vez que as medições feitas pelos objetos podem ou não variar em torno a uma média, isto dependerá do tipo de objeto que faz as medições.

De acordo com Cooper et al., (2009) os dados de IoT podem ser classificados como segue:

- Dados de Identificação de Radio Frequência (RFID – *Radio Frequency IDentification*): são utilizados para a identificação e o rastreamento de objetos através ondas de rádio. A *tags* RFID (Ashton, 2011) podem ser inseridas nos objetos e utilizadas para transmitir e receber informação. Esta tecnologia pode ser utilizada em diferentes áreas, quais sejam: controle de estoque, pedágio, gestão da cadeia de abastecimento, entre outras.
- Dados de endereçamento e identificação única: em geral, os objetos IoT precisam ser identificados de forma única com endereços IP no universo da Internet. Neste contexto, à medida que a quantidade dos objetos IoT cresce, o número de identificadores necessários crescerá na mesma proporção. Entre algumas formas únicas de identificação utilizadas pela IoT, tem-se, por exemplo, a Identificação Única Universal (UUID – *Universal Unique Identifier*).
- Metadados sobre os objetos, processos e sistemas: grande parte do poder do IoT vem dos dados ou metadados que são registrados pelos objetos, processos e sistemas participantes em um ambiente IoT. Os metadados são dados sobre dados, sendo essenciais para possibilitar aos usuários que encontrem e acessem os dados apropriados. Os metadados não somente são utilizados para descrever objetos, mas para descrever processos e sistemas.
- Dados de posicionamento: provê a localização de um objeto dentro de um ambiente IoT. A localização de um objeto pode ser feito através do Sistema de Posicionamento Global (GPS – *Global Positioning System*) ou Sistema de Posicionamento Local (LPS – *Local Positioning System*). O GPS é implementado

com múltiplos satélites enviando sinais para uma unidade controle, podendo acertar a posição de um objeto através de triangulação. Os dados de posicionamento são importantes para a IoT, uma vez que pode descrever se um objeto seja estático ou móvel.

- Dados de sensores: uma das fontes de dados para IoT são as redes de sensores. Estas redes podem monitorar fenômenos ambientais, tais como, o clima, temperatura, e ruído. As tecnologias de sensores fazem possível a captura de grandes quantidades de dados de forma rápida e em tempo real.
- Dados históricos: *petabytes* de dados são capturados pelos dispositivos/sensores no IoT. Tais dados precisam ser armazenados. Com o passar do tempo, os dados se tornam históricos. O volume de dados históricos tornam-se um desafio para sistemas orientados a decisões, já que deve ser definido como e qual dados tem que ser armazenados.
- Modelos físicos: são modelos que representam realidade, por exemplo, a gravidade, força, luz, som, e magnetismo. A incorporação desses modelos no IoT poderia melhorar sua funcionalidade.
- Dados de estado dos atuadores e dados de comando para o controle: os dispositivos/sensores podem ser controlado remotamente. Isto implica algum tipo de retorno sobre o estado dos atuadores dos dispositivos/sensores. Alguns dos dados que entram nos ambientes IoT são dados de comando, para controlar dispositivos. Por exemplo, o ar-condicionado pode ser configurado para ligar trinta minutos antes do horário de chegada em casa.

De fato, os diferentes tipos de dados sendo produzidos continuamente em um ambiente IoT apresenta um grande desafio na gestão de grande volume de dados, que é o assunto da próxima seção.

2.3 – ADMINISTRAÇÃO DE GRANDES VOLUMES DE DADOS

O gerenciamento de grande volume de dados está relacionado com a organização, administração e governança de grande volume de dados estruturados e não estruturados. A adequada administração de grandes conjuntos de dados auxilia na criação de informação valiosa de uma ou várias fontes de dados.

2.3.1 – Dados IoT como *Big Data*

O grande número de objeto que compõem uma rede IoT capturando eventos em tempo real produz grandes volumes de dados. Estes podem ter diversos formatos, tais como: alfanumérico, áudio, imagem, vídeo, entre outros. A complexidade estrutural dos dados e as características do fluxo de dados podem variar segundo o conteúdo dos dados. Tais dados heterogêneos em enormes quantidades, gerados pelos ambientes IoT apresentam desafios no que tange à armazenamento e processamento. Posteriormente, os dados coletados precisam ser analisados para gerar conhecimento. Embora o processamento possa ser realizado em um ambiente de computação tradicional (servidores centralizados), este modo de operação é propenso a limitações de capacidade quanto processamento e ao volume de armazenamento (Abu-Elkheir et al., 2013; Zaslavsky et al., 2013). Em contraposição, emergem ambientes distribuídos com características mais adequadas, incluindo escalabilidade e processamento paralelo, para superar as limitações citadas de arquiteturas centralizadas. Neste contexto, os dados gerados pela IoT podem se encaixar como uma fonte de *Big Data* (Assunção et al., 2015).

Grande parte das características do *Big Data* tem-se concentrado nas chamadas características 3V (Russom et al., 2011): volume, variedade e velocidade. Mas, o conceito de *Big Data* evoluiu para incluir características denominadas 7V (Khan et al., 2014), com a inclusão dos itens veracidade, validade, valor, volatilidade e visualização. A veracidade requer que os dados sejam verificáveis e verdadeiros. O valor está relacionado à importância, ao valor ou à utilidade das informações derivadas da exploração do *Big Data*. A volatilidade refere-se ao significado que os dados estão em constante mudança, e a visualização refere-se a todos os meios que tornam compreensível a grande quantidade de dados de um modo fácil de entender e ler. Além disso, existem vários outros V que podem ser considerados, quais sejam: viabilidade, vincularidade, vitalidade e outros.

As características dos dados IoT são compatíveis com as definições das características dos múltiplos Vs de *Big Data*, como argumentado por alguns autores (Chen et al., 2014; Hashem et al., 2015; Li et al., 2015; Vongsingthong, 2015), conforme se segue:

- Variedade (dados heterogêneos): as aplicações de IoT frequentemente envolvem diversos tipos de dados de diferentes aplicações. Os dados podem ser oriundos, por

exemplo, de sensores de temperatura, umidade e iluminação em aplicações nas áreas de agricultura e logística. Em aplicações na área da saúde, têm-se os dados dos sensores de monitoramento cardíaco, pressão sanguínea, batimento cardíaco e outros índices corporais para aplicações médicas (Li et al., 2015). É evidente que as diferentes fontes de dados mencionadas não estão limitadas a números e/ou textos. A estrutura dos dados podem aparecer na combinação de informações estruturadas (registros padrões), semiestruturados e não estruturado (vídeo, áudio e outros dados multimídia).

- Velocidade (dados em tempo real): os dados dos sensores são gerados em tempo real. O fluxo constante de sequência de dados que são enviados ou recebidos é conhecido como *stream* de dados. Como exemplo, tem-se um sensor que monitora o nível de glicose no organismo de uma pessoa. Em caso de elevados ou baixos valores de glicose, uma medida de glicemia capilar é recomendada para que se confirme o valor e a tomada de decisão, a fim de preservar a saúde de outrem. No cenário onde é possível a existência de milhões de dispositivos gerando dados em tempo real no intervalo de segundos, é muito rápida a geração de dados.
- Volume (dados são massivos): uma grande quantidade de dispositivos está conectada à *Internet*, gerando não apenas bilhões de dados, mas trilhões de dados, de modo constante e automático – o que leva a uma rápida expansão da escala de dados (Li et al., 2012). Esses necessitam de grande espaço de armazenamento e robustos sistemas de processamento. Neste sentido, é possível imaginar um cenário em uma loja de varejo onde milhões de mercadorias são disponibilizadas diariamente. Se tais objetos necessitam de rastreamento diário, no qual cada rastreamento gera 100 *bytes* de dados, o total de dados produzidos constantemente pode chegar de 100 GB até 36,5 *Terabytes* em um ano em um sistema de acompanhamento e descoberta (Vongsingthong, 2015).
- Veracidade: no IoT refere-se aos desvios, ruídos e anormalidades nos dados. A veracidade na análise de dados é tão desafiador quanto ao volume e à velocidade (Kaur, 2015). No escopo da estratégia do *Big Data* tem-se a necessidade de uma equipe e parceiros para o trabalho em conjunto, a fim de "limpar" os dados no sistema e evitar que os dados "sujos" se acumulem nos sistemas.

- Validade: lida com a precisão dos dados que são originários de diferentes fontes - não todas plenamente conhecidas ou verificáveis, entretanto a baixa qualidade dos dados tem se tornado um sério problema, uma vez que os dados de baixa qualidade podem acarretar em resultados com pouca precisão ou distorcidas (Hashem et al., 2015). Por outro lado, os dados de alta qualidade podem ser devidamente aproveitados para a tomada de decisões e conclusões futuras.
- Volatilidade: a volatilidade do *Big Data* remete à questão de quanto tempo os dados são válidos e se podem ser armazenados. No contexto dos ambientes IoT, os dados originados em tempo real necessitam de estudos prévios para saber até que ponto eles já não são mais relevantes para as análises atuais.
- Valor: é um aspecto importante do *Big Data* porque refere-se ao processo de descoberta dos valores escondidos de grandes conjuntos de dados com vários tipos e de geração rápida (Chen et al., 2014). Muitas vezes, o valor da análise dos dados IoT vem da combinação desses dados com dados de outras fontes, tais como dados relacionados com dos clientes.

Com o desenvolvimento dos ambientes IoT, os dados gerados por vários dispositivos, sensores e aplicações têm se tornado cada vez mais essenciais. Segundo (O'Leary, 2013), existe uma relação importante entre a IoT e o *Big Data,* pois na IoT os dados exigem mais flexibilidade, agilidade e escalabilidade.

Os bancos de dados relacionais podem ser uma opção para lidar com os dados da IoT. Porém, eles são mais adequados para dados estruturados, pois necessitam de um esquema (estrutura) para serem implementados. Além disso, são limitados enquanto escalabilidade horizontal para vários servidores. Neste sentido, os bancos de dados necessitam adotar os novos requerimentos da IoT com maior agilidade de processamento de dados, com base nas várias ferramentas de análise em tempo real e visões consistentes dos dados (Zhu, 2015).

2.3.2 – Bancos de Dados NoSQL

Devido ao desenvolvimento da Web 2.0 e as redes sociais, um grande volume de dados não estruturados tem sido criado. Tal fenômeno acarreta em problemas de desempenho para os bancos de dados relacionais quando lidam com grandes quantidades de dados

(Hadjigeorgiou et al., 2013; Stonebraker et al., 2005). Os sistemas gerenciadores de banco de dados NoSQL são projetados para processamento de grandes quantidades de dados não estruturados. Os bancos de dados NoSQL podem ser classificados em diferentes categorias, dependendo das características dos dados, quais sejam (Nayak et al., 2013; Strauch et al., 2011): bancos de dados chave-valor; bancos de dados orientados a documentos; banco de dados orientados a família de colunas; bancos de dados orientados a grafos e bancos de dados de séries temporais.

2.3.2.1 – Banco de Dados Orientado a Chave-Valor

Os bancos de dados NoSQL são de simples implementação (Nayak et al., 2013), similares a um mapa ou dicionário, onde o valor é recuperado pela sua chave correspondente. A estrutura de um banco de dados chave-valor é simples, uma vez que estes sistemas não têm restrições de esquema. Qualquer novo dado, não importando sua estrutura, pode ser armazenado a qualquer momento sem afetar os itens de dados existentes e da disponibilidade do sistema de banco de dados. Desde que os bancos de dados chave-valor têm por base uma chave de acesso, eles são eficientes para operações relativamente simples, apresentando boa capacidade de escalabilidade.

Entre os bancos de dados chave-valor mais populares tem-se: Riak (Muhammad, 2011), Amazon DynamoDB (DeCandia et al., 2007), SimpleDB (Sciore, 2007) e Redis (Zawodny, 2009).

2.3.2.2 – Banco de Dados Orientado a Documentos.

A estrutura básica de um sistema gerenciador de banco de dados baseado em documentos é um conjunto de pares chave-valor onde os documentos no banco de dados são endereçados usando uma chave exclusiva que representa esse documento. Todas as chaves devem ser únicas para cada documento. Embora os valores sejam transparentes ao sistema de banco de dados, eles também podem ser pesquisados. Portanto, os bancos de dados orientados a documentos podem ser convenientes para lidar com dados de estrutura complexa, como, por exemplo, objetos aninhados. Tais chaves podem ser uma sequência de caracteres que se refere a URI (Uniform Resource Identifier) ou caminho.

Os bancos de dados orientados a documentos são usados em aplicações nos quais os dados não precisam ser armazenados em tabelas com campos de tamanho uniforme, mas

os dados devem ser armazenados como um documento com características especiais. Os bancos de dados orientados a documentos mostram-se muito convenientes para a integração de dados e migração de esquema. Os bancos de dados orientados a documentos mais populares são o MongoDB (Chodorow, 2013) e o CouchDB (Chris et al., 2010).

2.3.2.3 – Banco de Dados Orientado a Coluna.

Os bancos de dados orientados a colunas armazenam os registros em famílias de colunas. Uma família de coluna é tratada como um registro que contém várias colunas que pode ser identificada por uma chave única. Neste tipo de sistema, cada registro pode armazenar muitos pares de valor-chave. O registro (uma família de coluna) no banco de dados orientado a coluna pode ser comparado com uma linha de colunas de um banco de dados relacional, onde se tem a identificação deste por uma chave primária e contém todas as colunas em cada tabela. A diferença é que todas as filas em um banco de dados orientado a coluna não necessitam do mesmo número de colunas, e as colunas em cada linha podem ser diferentes.

Uma característica importante deste tipo de banco de dados é a organização das colunas em grupos de colunas chamados de família de colunas. Porém, tal característica faz com que o sistema seja menos flexível que um banco de dados valor-chave. Entre os bancos de dados orientados a coluna tem-se: Cassandra (Lakshman et al., 2010), HBase (Naheman et al., 2013) (Carstoiu et al., 2014) e Hypertable (Khetrapal et al., 2006).

2.3.2.4 – Banco de Dados Orientado a Grafos.

Os bancos de dados orientados a grafos são bancos de dados que armazenam dados sob a forma de um grafo. O grafo consiste em nós e vértices, onde os nós atuam como objetos e os vértices agem como a relação entre os objetos. O grafo também consiste em propriedades relacionadas aos nós. Ele usa uma técnica chamada índice livre de adjacência significando que cada nó consiste em um ponteiro direito que aponta para o nó adjacente. Milhões de registros podem ser percorridos usando esta técnica.

Nos bancos de dados orientado a grafos, a ênfase principal está na conexão entre dados. Este tipo de bancos provê armazenamento eficiente de dados semiestruturados e carece de esquema de dados. Comparado com bancos de dados relacionais e os outros três tipos de bancos NoSQL mencionados anteriormente, os bancos de dados orientados a

grafos são propícios para o gerenciamento de dados interligados. Neste contexto, as aplicações com dados que tenham muitos relacionamentos são as mais adequadas para os bancos de dados orientados a grafos, uma vez que as operações mais complexas para outros sistemas tornam-se mais simples em bancos de dados orientados a grafos.

Neo4J (Webber, 2012), *Infinite Graph* (Buerli et al., 2012) e OrientDB (Tesoriero, 2013) são alguns dos mais populares bancos de dados orientados a grafos.

2.3.2.5 – Banco de Dados de Series temporais

Os dados gerados pelos dispositivos/sensores são normalmente fornecidos como séries temporais, com uma frequência fixa em intervalos discretos. Bases de dados especializadas oferecem suporte para o tratamento de grandes quantidades de dados indexados pelo tempo. Os dados armazenados neste tipo de banco de dados têm uma ordem temporal. Este tipo de banco de dados suporta operações como criação, atualização, remoção e enumeração das séries temporais. Entre os bancos de dados de séries temporais tem-se (Wlodarczyk, 2012): TempoDB, OpenTSDB, InfluxDB.

2.4 – REQUERIMENTO DE BANCO DE DADOS PARA IoT

A Internet das Coisas apresenta um novo conjunto de desafios para sistemas de gerenciamento de banco de dados, tais como consumir grandes volumes de dados em tempo real ou quase em tempo real. Isto implica processar os dados conforme eles são gerados, lidando com volumes de dados significativamente maiores de aqueles frequentemente encontrados em aplicações empresariais (Abu-Elkheir et al., 2013). Para tratar adequadamente os dados IoT e seus requisitos, uma escolha crítica diz respeito ao banco de dados ou conjunto de bancos de dados necessários.

Existem muitos fatores que devem ser levados em consideração ao escolher um sistema gerenciador de banco de dados para dados IoT. Tais fatores nem sempre se alinham com as características dos sistemas de banco de dados mais tradicionais (Abu-Elkheir et al., 2013). Neste contexto, entre os muitos requisitos que devem ser considerados, alguns dos mais importantes são: escalabilidade, capacidade de consumir dados a taxas elevadas, flexibilidade do esquema, integração com ferramentas analíticas e custos.

Em um contexto ideal, os bancos de dados IoT seriam linearmente escaláveis. Assim, por exemplo, adicionando mais um servidor a um *cluster* de 10 nós, aumentaria a taxa de transferência em 10%, mas geralmente as operações de coordenação e comunicação dificultam este crescimento linear. Além disso, considerando que o IoT é um sistema distribuído, os bancos de dados IoT geralmente serão distribuídos, a menos que o aplicativo colete apenas uma pequena quantidade de dados que não crescerão significativamente. Os bancos de dados distribuídos podem ser executados em *hardware* de prateleira e escalar, adicionando novos servidores ao invés de trocar um servidor por outro maior.

Uma base de dados IoT também deve ser tolerante a falhas e com alta disponibilidade. Isto implica que se um nó do *cluster* de banco de dados estiver inativo, o serviço de banco de dados ainda poderá aceitar solicitações de leitura e escrita. Os bancos de dados distribuídos fazem cópias ou réplicas de dados e os gravam em vários servidores. Em caso de falha de um dos servidores que armazena um determinado conjunto de dados, então um outro servidor que tenha uma réplica do conjunto de dados poderá responder as consultas em relação a esse conjunto de dados. Em relação às solicitações de escrita, se o servidor que recebe frequentemente uma solicitação estiver desativado, outro nó do *cluster* poderá aceitar a solicitação e encaminhá-la ao servidor de destino quando ele voltar a seu estado operacional.

Uma vez que as bases de dados IoT devem ser flexíveis como exigido pelas aplicações IoT, os bancos de dados NoSQL, tais como: valor-chave, colunar e orientada a documento, acomodam facilmente tipos e estruturas de dados diferentes sem a necessidade de esquemas fixos predefinidos. Os bancos de dados NoSQL são uma opção valiosa quando é necessário acomodar vários tipos de dados e esses tipos de dados podem mudar ao longo do tempo.

Nos sistemas de banco de dados relacional – principalmente centrados no armazenamento –, o volume de dados, em geral, é coletado de fontes pré-definidas e finitas e armazenado de acordo com regras de normalização. Neste tipo de banco de dados, os mecanismos de gerenciamento de transação garantem as propriedades ACID, a fim de reforçar a integridade geral dos dados (Ramakrishnan et al., 2000; Tamer et al., 2011).

As bases de dados NoSQL tentam resolver algumas das limitações tradicionais do banco de dados por relaxar algumas das propriedades de ACID, uma vez que relaxar certas

restrições pode ser uma boa medida para várias aplicações. Por exemplo: bases de dados NoSQL podem executar muito bem em circunstâncias onde as restrições de coerência não são necessárias (Luo et al., 2009). Aqui, tais bancos de dados NoSQL podem se tornar tolerantes às partições de rede, o que torna possível adicionar servidores à instalação quando o número de dispositivos/sensores aumenta, em vez de adicionar mais capacidade a um único servidor.

2.5 – ARQUITETURA LAMBDA

O termo de "arquitetura Lambda" é uma arquitetura de processamento de dados genérica, escalável e tolerante a falhas para ecossistemas de Big data (Marz et al., 2015). A arquitetura Lambda visa satisfazer os requerimentos do big data baseado em sistemas de processamento distribuídos. Esta tem por objetivo satisfazer as necessidades para um sistema robusto que seja linearmente escalável, tolerante a falhas, sendo capaz de atender a uma ampla gama de carga de trabalho e casos de uso onde são necessárias leituras e atualizações de baixa latência. A arquitetura Lambda é definida em três camadas: camada *batch*, de velocidade e de serviço, conforme evidenciado na Figura 2.2.

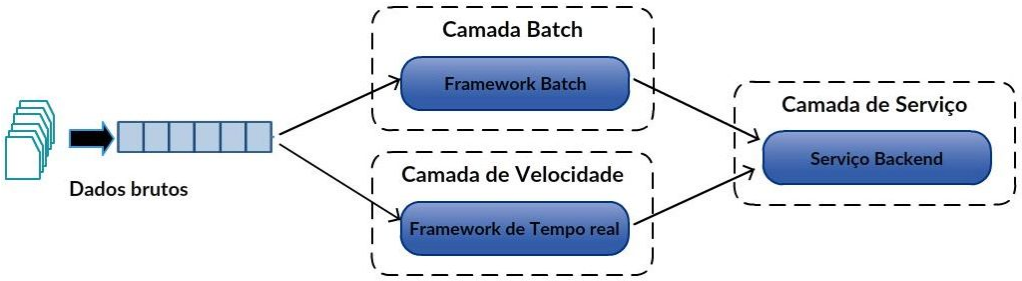

Figura 2.2. Arquitetura Lambda.

A camada *batch* pré-processa os dados fazendo uso de um sistema de processamento distribuído que pode lidar com grande quantidade de dados. Esta camada é capaz de armazenar os dados imutáveis que crescem de modo constante, além de processar funções arbitrárias sobre os dados normalmente com alta latência de resposta.

A camada de serviço indexa as visões da camada *batch* e da camada de velocidade, logrando responder às consultas, retornando as visões pré-processadas ou construindo visões a partir dos dados processados. Basicamente, a camada serviço é uma base de dados escalável que se atualiza com as visões *batch* e as visões da camada de velocidade quando

estas se tornam disponíveis. Devido à latência da camada *batch*, os resultados da camada serviço sempre possuem um atraso considerável.

A camada de velocidade processa *stream* de dados em tempo real sem se preocupar com os requisitos de correção ou completude. Esta sacrifica a taxa de transferência para minimizar latência. disponibilizando visões em tempo real dos dados mais recentes. Basicamente, a camada de velocidade é responsável pelo preenchimento do *gap* causado pelo atraso da camada *batch* ao fornecer visões em tempo real – que estão sempre atualizadas e as armazena em bases de dados tanto para operações de leitura quanto escrita.

Cada uma dessas camadas pode ser implementada fazendo uso de várias tecnologias de *Big Data*. Por exemplo: os dados de camada *batch* podem ser armazenados em um sistema de arquivos distribuído, enquanto MapReduce pode ser utilizados para criar visões *batch* que podem ser alimentadas para a camada de serviço. A camada de serviço pode ser implementada fazendo uso de tecnologias NoSQL, enquanto a consulta pode ser implementada através de sistemas. Finalmente, a camada de velocidade pode ser implementada com sistemas de processamento de *stream* de dados.

2.6 – PROCESSAMENTO DE DADOS IoT

Para lidar com complexos ambientes do IoT e o seu grande volume de dados associados é essencial definir técnicas adequadas para suportar a complexidade estrutural e requisitos de desempenho para o processamento de dados dentro dos ambientes IoT. Uma plataforma de dados que gerencia dados de dispositivos IoT de forma confiável e em larga escala deve atender a alguns requisitos.

As aplicações IoT, em geral, requerem que a plataforma possa suportar nativamente o processamento de *streams* e que possa lidar com consultas de baixa latência ou restrições temporais. Para fins desta tese o termo tempo real será referido ao processamento de baixa latência ou restrições temporais. O tratamento dos dados IoT requer uma arquitetura escalável para poder suportar o grande volumes de dados. Tais requerimentos envolvem questões na articulação de técnicas como sistemas de mensagens e computação em tempo real. No contexto IoT, os sistemas de mensageria podem ser utilizados para coletar dados provenientes de vários dispositivos/sensores (Karagiannis et al., 2015) e serem processados por diferentes clientes subscritos, enquanto os sistemas em tempo real são muito importantes para processar dados em tempo real provenientes de instâncias IoT.

Neste contexto, a mensagem é definida como a troca de mensagens entre sistemas que transportam dados especialmente formatados descrevendo eventos, requerimentos e respostas – ação realizada através de um servidor de mensageria que fornece uma interface de comunicação para diferentes clientes. Tal servidor também fornece funcionalidades administrativas e de controle, incluindo a persistência de dados e a confiabilidade (Mirco et al., 2004). Existem dois modelos principais de sistema de mensageria (Mirco et al., 2004): os modelos ponto a ponto e *publish-subscribe*. O modelo ponto a ponto permite aos clientes enviar e receber mensagens de forma síncrona e assíncrona via canais virtuais denominados filas. A característica principal deste modelo é que as mensagens enviadas a uma fila são recebidas por um e somente um receptor, mesmo existindo muitos receptores escutando essa fila pela mesma mensagem. O modelo *publish-subscribe* tem por base a utilização de um canal virtual chamados de tópico. Os produtores de mensagens são denominados *publishers*, onde os consumidores de mensagens são chamados de *subscribers*. Neste modelo, as mensagens publicadas em um tópico podem ser recebidas por múltiplos *subscribers*; as mensagens são automaticamente transmitidas aos *subscribers*, sem que eles tenham que solicitar ou pesquisar o tópico por novas mensagens. No contexto dos ambientes IoT, a abstração do modelo *publish-subscribe* visa atender os requerimentos de comunicação de um ambiente IoT (Hakiri *et al*., 2015). Além disso, o modelo *publish-subscribe* dá suporte ao controle do processo de aquisição e filtragem de dados nos ambientes IoT (Žarko et al., 2014).

Para dar suporte ao requerimento do processamento em tempo real (baixa latência) tem-se o uso de sistemas de processamento em tempo real. Os sistemas em tempo real são definidos como aqueles sistemas onde o funcionamento correto do sistema depende não apenas do resultado lógico da computação, mas do tempo em que os resultados são produzidos (Stankovic et al., 1992). Existem três componentes principais, bem como sua interação, que caracterizam os sistemas em tempo real (Shin et al., 1994) : tempo, confiabilidade e o ambiente sob o qual um computador funciona. Tais componentes são cruciais para evitar danos graves, econômicos ou em termos de perda de vidas.

3 – TRABALHOS RELACIONADOS

Este capítulo tem como objetivo apresentar os trabalhos relacionados à proposta de gerenciamento de dados em ambiente IoT, GDDIoT, desta tese. O capítulo foi dividido nas seguintes seções: 3.1 apresenta uma visão geral sobre o gerenciamento de dados em middlewares IoT; seção 3.2 tecnologias de big data aplicadas a ambiente de IoT; por fim, na seção 3.3 as contribuições do GDDIoT em relação aos trabalhos relacionados são apresentadas.

3.1 - GERENCIAMENTO DE DADOS EM MIDDLEWARES IoT

Na área de armazenamento e processamento de dados IoT, numerosos esforços têm sido feitos para a gestão de grande volume de dados em tal ambiente. No aspecto do gerenciamento de dados IoT, os sistemas IoT tentam resolver o gerenciamento dos dados na camada *middleware* e via gerenciamento de dados com técnicas *big data*.

Além de facilitar a configuração e implantação de dispositivos para usuários não especialistas e usuários finais (Henricksen et al., 2005), os *middlewares* IoT capturam os dados gerados pelos dispositivos heterogêneos, processando-os e armazenando-os em sistemas de armazenamento persistentes.

Para gerenciar os dados, os *middlewares* usualmente implementam mecanismos para o gerenciamento dos dados. Muitas soluções têm sido propostas e implementadas proporcional à quantidade de *middlewares* IoT. Tipicamente, tais soluções são altamente acopladas aos *middlewares* diversos na sua abordagem (eventos, por exemplo), no nível de abstração (nível local ou de rede, por exemplo) e no domínio da implementação (WSN, RFID, M2M e SCADA, por exemplo).

No contexto de IoT, as atuais plataformas de *middleware* representam um artefato de *software* residindo entre a camada de aplicação e a infraestrutura de suporte representado pelas camadas de comunicação e sensoriamento proposto por Fremantle et al., (2014) e Bassi et al., (2013). Neste contexto, a discussão dos trabalhos relacionados destaca apenas os pontos-chave relacionados ao gerenciamento de dados, sem capturar exaustivamente seu desempenho em relação a outros requisitos. Na IoT, os dados referem-se principalmente a dados de sensoriamento ou informação de infraestrutura de interesse para aplicações. Um *middleware* IoT necessita fornecer serviços de

gerenciamento de dados para as diferentes aplicações IoT, incluindo a aquisição de dados, o processamento de dados (incluindo pré-processamento) e o armazenamento de dados. O pré-processamento pode incluir a filtragem de dados, a compressão de dados e a agregação de dados.

Diante do exposto, a seguir, tem-se algumas propostas de plataformas de *middleware* para IoT divididas em diferentes categorias, conforme (Razzaque et al., 2016), a saber: eventos, orientado a serviços, VMs, agentes, tupla-espaço, orientado a banco de dados e orientado a aplicações específicas. Aqui tem-se uma análise de como tais plataformas procuram lidar com o grande volume de dados gerados pelo IoT.

A Tabela 2.1 apresenta um resumo dos trabalhos encontrados na literatura para o gerenciamento de dados em *middlewares* IoT específicos baseados em eventos. Neste grupo de 9 *middlewares*, foi analisado o gerenciamento de dados realizado baseado nas seguintes propriedades: Armazenamento de Dados (AD), Agregação (A), Compressão (C), Filtragem (F), Não Informado (NI) e Não Suportado (NS). Como pode ser observado, quase todos os *middlewares* deste grupo tem a parte de armazenamento de dados porém poucos fazem a agregação dos dados. Nenhum dos *middlewares* estudados tratam da característica de compressão.

Tabela 2.1. Gerenciamento de dados em *middlewares* IoT baseado em eventos.

		Pré processamento de dados		
Middleware	**AD**	**A**	**C**	**F**
Hermes (Pietzuch, 2004)	NS	NS	NS	✔
EMMA (Mirco Musolesi et al., 2004)	✔	NS	NS	NS
GREEN (Sivaharan et al., 2005)	✔	NS	NS	✔
RUNES (Costa et al., 2007)	NI	✔	NS	NS
PRISMA (Silva et al., 2014)	✔	✔	NS	NS
SensorBus (Lima et al., 2008)	NS	NS	NS	NS
Mires (Souto et al., 2006)	NI	✔	NS	NS
Kaa (Technologies, 2017)	✔	NS	NS	NS
SmartCampus (Cecchinel et al., 2014)	✔	✔	NS	NS

A Tabela 2.2 evidencia o resumo do gerenciamento de dados feito pelos *middlewares* orientado a serviços. Foram encontrados 14 *middlewares* orientados a serviço. Grande parte dos *middlewares* deste grupo atende a propriedade de armazenamento de dados, porém, não foram encontradas informações sobre a filtragem dos dados. Em relação a característica de compressão apenas o Servilla (Fok et al., 2012) deixa claro o uso de uma variante do algoritmo FFT (*Fast Fourier Transform*).

Tabela 2.2. Gerenciamento de dados em *middlewares* IoT orientado a serviços.

		Pré processamento de dados		
Middleware	**AD**	**A**	**C**	**F**
Hydra (Eisenhauer et al., 2010)	✔	NS	NS	NS
Sensewrap (Evensen et al., 2009)	NI	NI	NI	NI
MUSIC (Rouvoy et al., 2009)	NS	NS	NS	NS
TinySOA (Avilés et al., 2009)	✔	NS	NS	NS
SOCRADES (Kenda et al., 2013)	NI	NI	NI	NI
SENSEI (Tsiatsis et al., 2010)	✔	✔	NS	NS
ubiSOAP (Caporuscio et al., 2012)	NI	NI	NI	NI
Servilla (Fok et al., 2012)	NS	NS	✔	NS
KASOM (Corredor et al., 2012)	NS	NS	NS	NS
CHOReOS (Hamida et al., 2013)	NI	✔	NS	NS
MOSDEN (Perera et al., 2014)	✔	✔	NS	NS
Xively (Bahga et al., 2014)	✔	✔	NS	NS
CarrIoT (Razzaque et al., 2016)	✔	✔	NS	NS
Echelon (Xu et al., 2014)	✔	✔	NS	NS

A Tabela 2.3 apresenta um resumo do gerenciamento de dados feito pelos *middlewares* baseados em VMs. Foram encontrados 12 *middlewares* baseado em VMs. Grande parte dos *middlewares* deste grupo atende a propriedade de armazenamento de

dados e agregação de dados, porém, não foram encontradas informações sobre a compressão. Em relação a característica de filtragem dos dados apenas o SwissQM (Mueller et al., 2007) suporta a filtragem dos dados, devido a que aumenta a o nível de abstração da programação, aceitando programas e consultas escritas em linguagem de alto nível.

Tabela 2.3. Gerenciamento de dados em *middlewares* IoT baseado em VMs.

		Pré processamento de dados		
Middleware	**AD**	**A**	**C**	**F**
Mate (Levis et al., 2002)	✔	✔	NS	NS
VM* (Koshy et al., 2005)	✔	✔	NS	NS
Melete (Yu et al., 2006)	✔	✔	NS	NS
MagnetOS (Kirsch et al., 2005)	✔	✔	NS	NS
Squawk (Simon et al., 2006)	✔	✔	NS	NS
Sensorware (Boulis et al., 2007)	✔	✔	NS	NS
Extended Maté (Philip et al., 2005)	✔	✔	NS	NS
DVM (Balani et al., 2006)	✔	✔	NS	NS
DAViM (Musolesi et al., 2006)	✔	✔	NS	NS
SwissQM (Mueller et al., 2007)	✔	✔	NS	✔
TinyVM (Hong et al., 2012)	NI	✔	NS	NS
TinyReef (Marques et al., 2009)	✔	✔	NS	NS

A Tabela 2.4 evidencia o resumo do gerenciamento de dados feito pelos *middlewares* baseado em agentes. Foram encontrados 10 *middlewares* baseado em agentes. Grande parte dos *middlewares* deste grupo atende a propriedade de agregação de dados, porém, não foram encontradas informações sobre a compressão dos dados. Em relação a característica da filtragem de dados apenas o TinyMAPS (Aiello et al., 2011). Além disso, grande parte dos *middlewares* não atende a propriedade de armazenamento de dados devido a estarem focados no gerenciamento dos recursos, gerenciamento de código,

disponibilidade, adaptabilidade e heterogeneidade.

Tabela 2.4. Gerenciamento de dados em *middlewares* IoT baseados em agentes.

		Pré processamento de dados		
Middleware	**AD**	**A**	**C**	**F**
Impala (Liu et al., 2003)	NI	✔	NS	NS
Smart message (Kang et al., 2004)	NI	✔	NS	NS
ActorNet (Kwon et al., 2006)	NI	✔	NS	NS
Agilla (Fok, Roman, & Lu, 2009)	NI	✔	NS	NS
Ubiware (Nagy et al., 2009)	NI	✔	NS	NS
UbiRoad (Terziyan et al., 2010)	✔	NS	NS	NS
AFME (Muldoon et al., 2006)	NI	✔	NS	NS
MAPS (Souto et al., 2006)	NI	✔	NS	NS
MASPOT (Lopes et al., 2011)	NI	✔	NS	NS
TinyMAPS (Aiello et al., 2011)	NI	✔	NS	✔

A Tabela 2.5 evidencia o resumo do gerenciamento de dados feito pelos *middlewares* baseado em tupla-espaço. Foram encontrados 5 *middlewares* baseado em tupla-espaço. Grande parte dos *middlewares* deste grupo atende a propriedade de armazenamento e agregação de dados, porém, não foram encontradas informações sobre a filtragem e compressão dos dados.

Tabela 2.5. Gerenciamento de dados em *middlewares* IoT baseado em tupla-espaço.

		Pré processamento de dados		
Middleware	**AD**	**A**	**C**	**F**
LIME (Murphy et al., 2001)	✔	NS	NS	NS
TeenyLIME (Costa et al., 2006)	✔	✔	NS	NS
TinyLINE (Curino et al., 2005)	NI	✔	NS	NS

TS-Mid (Lima et al., 2008)	✔	✔	NS	NS
A3-TAG (Baresi et al., 2013)	✔	✔	NS	NS

A Tabela 2.6 apresenta um resumo do gerenciamento de dados feito pelos *middlewares* baseados em banco de dados. Foram encontrados 8 *middlewares* baseados em banco de dados. Grande parte dos *middlewares* deste grupo atende as propriedades de armazenamento de dados e agregação de dados, porém, não foram encontradas informações sobre a compressão. Em relação a característica de filtragem dos dados apenas IrisNet (Gibbons et al., 2003), TinyDB (Madden et al., 2005) e GSN (Aberer et al., 2006) suportam a filtragem dos dados.

Tabela 2.6. Gerenciamento de dados em *middlewares* IoT baseados em banco de dados.

		Pré processamento de dados		
Middleware	**AD**	**A**	**C**	**F**
SINA (Shen et al., 2001)	✔	✔	NS	NS
COUGAR (Bonnet et al., 2001)	✔	✔	NS	NS
IrisNet (Gibbons et al., 2003)	✔	✔	NS	✔
Sensation (Hasiotis et al., 2005)	✔	✔	NS	NS
TinyDB (Madden et al., 2005)	✔	✔	NS	✔
GSN (Aberer et al., 2006)	✔	NS	NS	✔
KSpot (Nagy et al., 2009)	✔	✔	NS	NS
HyCache (Zhao et al., 2013)	✔	✔	NS	NS

A Tabela 2.7 apresenta um resumo do gerenciamento de dados feito pelos *middlewares* baseados em aplicação específica. Foram encontrados 5 *middlewares* baseados em aplicação específica. Todos os *middlewares* deste grupo atende as propriedades de armazenamento de dados e agregação de dados, porém, grande parte destes *middlewares* não suportam a compressão dos dados. Em relação a característica da filtragem dos dados apenas AutoSec (Han et al., 2001) e MidFusion (Alex et al., 2008) as

suportam.

Tabela 2.7. Gerenciamento de dados em *middlewares* IoT baseados em aplicação específica.

		Pré processamento de dados		
Middleware	**AD**	**A**	**C**	**F**
AutoSec (Han et al., 2001)	✔	✔	NS	✔
Adaptive middleware (Huebscher et al., 2004)	✔	✔	NS	NS
MiLAN (Heinzelman et al., 2004)	✔	✔	NS	NS
TinyCubus (Marrón et al., 2005)	✔	✔	NS	NS
MidFusion (Alex et al., 2008)	✔	✔	✔	✔

As Tabelas 2.1 a 2.7 apresentadas anteriormente sumarizam cada categoria dos *middlewares* no que tange ao requerimento de gerenciamento de dados originados dos *middlewares*. Grande parte dos *middlewares* pesquisados oferece suporte para a agregação de dados, mas não consideram a filtragem de dados. É provável que a filtragem de dados seja encontrada em abordagens específicas da aplicação, uma vez que o *middleware* é adaptado para uma aplicação específica ou grupo de aplicações. Além disso, é possível observar que muitas abordagens não ofertam compressão de dados, sendo uma questão importante no gerenciamento de dados IoT.

3.2 - GERENCIAMENTO DE DADOS EM BIG DATA

Nesta seção, diferentemente da anterior que tinha o foco no *middleware* IoT, são apresentados alguns trabalhos encontrados na literatura que usam tecnologia de big data para gerenciamento de dados IoT independentemente do *middleware* IoT.

Vargas et al., (2017), por exemplo, propõe uma abordagem com base em uma rede de comunicação distribuída e técnicas de processamentos de *stream* de dados. Tal abordagem faz uso de um *cluster* Kafka. O objetivo desta é focar no desenvolvimento de um sistema para coletar e armazenar stream de dados em alta velocidade de uma rede WSN (rede de sensores sem fio) de modo escalável. Além disso, faz uso da segmentação

de dados para o processamento, independente de um segmento específico de dados economizando tempo.

AllJoyn Lambda (Villari et al., 2014) é um *framework* para o desenvolvimento de sistemas IoT. Tal abordagem tem por base a arquitetura Lambda para o armazenamento de grandes volumes de dados, processamento e análises em tempo real. Esta faz uso do MongoDB como banco de dados distribuído e da Apache Storm para o processamento de *stream* de dados. Através de diferentes aplicações estabelece conexão com os dispositivos físicos, a fim de encaminhar os dados das leituras ao banco de dados. Os dados são armazenados em diferentes coleções (padrões regulares, padrões eventuais e padrões automáticos), podendo priorizar cada um de modo específico.

Kang et al., (2016) propõe um serviço de dados com base na arquitetura Lambda. Este é composto por um conjunto de serviços que pretende integrar sistemas para o processamento de *stream* de dados e processamento *batch* para ambiente IoT. Em tal abordagem faz-se o uso do *framework* Hadoop para a implementação da camada de processamento *batch*, e Storm para o processamento em tempo real. Além disso, nesta abordagem tem-se o uso do protocolo MQTT (Hunkeler et al., 2008) para o envio dos dados dos sensores, com armazenamento no HDFS como banco de dados.

No contexto do *big data*, grande parte das abordagens são uma implementação ou uma variante da implementação da arquitetura Lambda. Tal fato se dá devido à arquitetura Lambda ser genérica, podendo ser aplicada em diferentes ambientes com big data.

3.3 - CONTRIBUIÇÕES DO GDDIoT EM RELAÇÃO AOS TRABALHOS RELACIONADOS

Os sistemas mencionados neste capítulo têm características em comum, especialmente em termos de integração e interoperabilidade entre os dispositivos físicos heterogêneos, sendo suas implementações projetadas para *middlewares* em específico. No contexto do gerenciamento de dados, os *middlewares* IoT possuem algumas características comuns, como a implementação de seu próprio sistema de coleta de dados, embora cada um use sistemas de armazenamento diferentes, desde banco de dados relacionais centralizados até os sistemas NoSQL distribuídos. É importante ressaltar que tais coletores de dados estão bem acoplados às suas respectivas arquiteturas dos *middlewares* IoT.

O trabalho apresentado nesta tese na forma de GDDIoT difere das abordagens descritas, uma vez que foi projetado como um serviço especializado no gerenciamento de dados provenientes de diferentes ambientes IoT simultaneamente de forma não intrusiva. Este serviço pode ajudar aos novos *middlewares* e aos já existentes gerenciar a enorme quantidade de dados gerados pelos ambiente IoT. Embora a maioria dos *middlewares* IoT aqui estudados prove a agregação de dados, mas não consideram a compressão de dados e a filtragem de dados. Neste contexto, além de implementar a agregação, compressão e filtragem de dados, o GDDIoT coleta, armazena, organiza e apresenta uma maneira de visualização dos dados.

Apesar de usar um processo padrão para coleta, filtragem, processamento, armazenamento e visualização de dados, o GDDIoT oferece uma arquitetura totalmente distribuída e paralela, que garante a escalabilidade da coleta de dados, processamento de dados e serviço de visualização de dados para ambientes IoT que geram grandes quantidades de dados. Além disso, para suportar a heterogeneidade dos dados provenientes de diferentes *middlewares* IoT, o GDDIoT usa metadados para superar a heterogeneidade de dados. Isso é importante quando lidar com variedade de formatos das mensagens.

4 – GERENCIAMENTO DISTRIBUÍDO DE DADOS PARA Internet das Coisas

Este capítulo tem como objetivo apresentar o GDDIoT como serviço para o gerenciamento de dados para Internet das Coisas. O capítulo foi dividido nas seguintes seções: 4.1 apresenta a definição do GDDIoT assim como a sua arquitetura; 4.2 apresenta o componente de interface de comunicação; 3.3 apresenta o componente de coleção de dados; 3.4 apresenta o componente de agregação de dados; por fim, na 4.5 apresenta o componente de visualização de dados.

4.1 – GDDIoT

GDDIoT é um serviço para o gerenciamento de dados IoT que tem por objetivo coletar, processar, organizar e armazenar grandes quantidades de dados produzidos por diferentes ambientes IoT. Este pretende ser um serviço com baixo acoplamento e serviço reutilizável. A Figura 4.1, apresenta, de modo genérico, como se dá a comunicação do GDDIoT com os *middlewares* IoT. O GDDIoT é distribuído e adaptável ao ambiente computacional, e tem a habilidade de suportar múltiplas conexões simultaneamente com diferentes *middlewares* IoT.

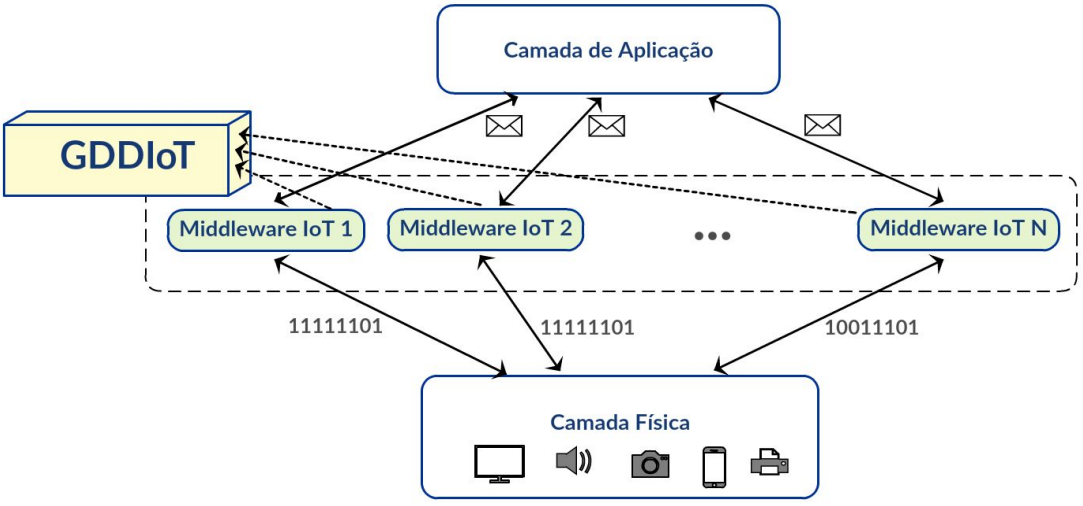

Figura 4.1. Modelo GDDIoT no contexto de uma arquitetura IoT.

O GDDIoT é definido com quatro componentes, a saber: interface de comunicação de dados, coleção de dados, agregação de dados, e visualização de dados, conforme expresso na Figura 4.2, a seguir. A interface de comunicação é responsável por estabelecer

a comunicação entre os *middlewares* IoT e o GDDIoT. O componente de coleção de dados recebe as mensagens dos *middlewares* IoT através da interface de comunicação. O componente de coleção de dados é responsável pela captura de dados vindos de *middlewares* IoT gerados pelos objetos (Huacarpuma *et al*., 2016). O componente de agregação de dados é responsável pela sumarização dos dados coletados pelo componente (Huacarpuma *et al*., 2017). Depois da coleta e agregação dos dados, estes podem ser visualizados em tempo real via componente de visualização.

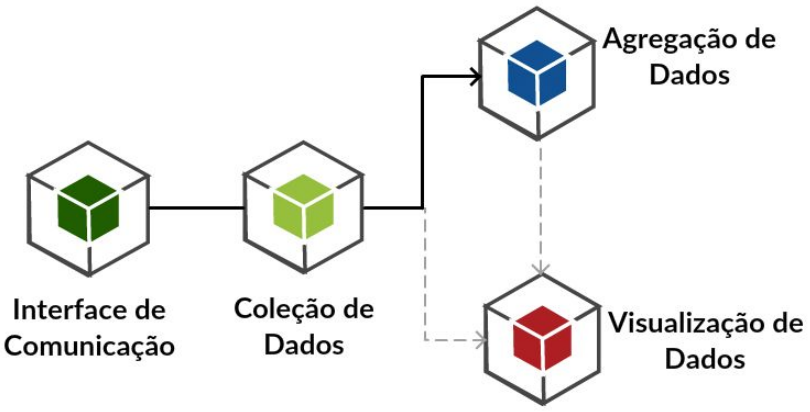

Figura 4.2. Arquitetura do GDDIoT.

A utilização do GDDIoT não deve interferir com o fluxo dos dados que os *middlewares* IoT realizam normalmente. Uma das características do GDDIoT é ser o menos intrusivo possível no funcionamento dos *middlewares* IoT. Assim, a interface de comunicação recebe as mensagens de diferentes *middlewares* IoT, processando-as para garantir um formato padrão de mensagens encaminhadas ao componente coleção de dados. Enquanto os dados são recebidos pelo componente coleção de dados, paralelamente, o componente agregação de dados realiza a sumarização dos dados coletados em determinados períodos de tempo. Depois da coleta e agregação dos dados, os mesmos podem ser visualizados através do componente visualização de dados – que torna possível a visualização e consulta dos dados em tempo real.

4.1 – COMPONENTE DE INTERFACE DE COMUNICAÇÃO DE DADOS

O GDDIoT possui características não intrusivas nas funções dos *middlewares*, uma vez que detém a interface de comunicação que estabelece a comunicação entre o *middleware* e

o GDDIoT. Tal componente torna possível traduzir as mensagens utilizadas de diferentes *middlewares* para uma estrutura padrão de mensagem gerenciada pelos componentes do GDDIoT. Através da referida interface, o *middleware* pode escolher os campos específicos que devem ser gerenciados pelo GDDIoT. Uma cópia de todas as mensagens geradas pelo *middleware* IoT deve ser enviada através do canal de comunicação estabelecido para esta finalidade. Assim, para lidar com a heterogeneidade dos dados IoT, é definida internamente no GDDIoT uma estrutura de mensagem JSON padrão capaz de comportar diferentes formatos de mensagens de várias fontes de dados.

No GDDIoT, a estrutura JSON utilizada inclui diferentes atributos, tais como: nome do *middleware*, identificador do dispositivo e identificação do serviço que inclui a variável de estado e o novo valor da leitura. A mensagem JSON também inclui metadados (geolocalização, tipo de dados, por exemplo). Para metadados mais detalhados, um atributo adicional é incluído na forma de uma lista de metadados – que descrevem dados específicos para determinadas aplicações IoT.

A interface de comunicação é projetada para ser transparente para o fluxo de dados *middleware* IoT. O primeiro passo é o estabelecimento da comunicação do *middleware* IoT com o GDDIoT para, posteriormente, realizar a transmissão de dados.

A Figura 4.3, evidencia a sequência da troca de mensagens para o estabelecimento dessa conexão. Primeiro, o *middleware* IoT envia a solicitação para estabelecer a conexão com o GDDIoT (1). Em seguida, o módulo de interface de comunicação de dados envia uma resposta da aceitação para o estabelecimento da comunicação (2). Após o estabelecimento da comunicação, o *middleware* inicia a transmissão das mensagens (3). Aqui se tem a transformação das mensagens utilizadas pelos *middlewares* IoT, tais como: CSV, JSON e XML, para mensagens padrão utilizadas pelo GDDIoT. Tais mensagens contêm determinados atributos, a saber: identificador de dispositivo, identificador de serviço e variáveis de estado (por exemplo, a medição feita pelos dispositivos tais como temperatura do ambiente). O componente da interface de comunicação estabelece a conexão com o GDDIoT através de uma porta onde todas as mensagens são encaminhadas.

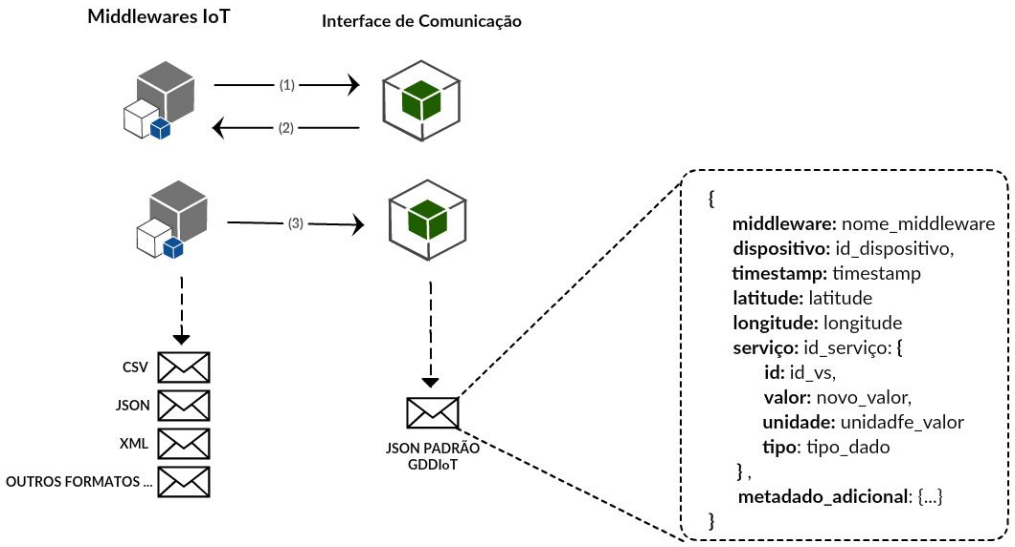

Figura 4.3. Sequência de comunicação e formato das mensagens JSON.

4.2 – COMPONENTE DE COLEÇÃO DE DADOS

O componente de coleção de dados recebe os dados de diferentes *middlewares* IoT através da interface de comunicação de dados. Tal processo não somente recebe dados, mas processa e armazena as leituras das medições dos objetos IoT. A Figura 4.4, a seguir, apresenta uma visão geral do componente de coleção de dados composto por quatro módulos, quais sejam:

- Captura de dados: recebe as mensagens encaminhadas pela interface de comunicação de dados;
- Filtragem de dados: verifica os valores das medições realizadas pelos objetos IoT (verificação de domínio das mensagens, por exemplo); analisa as medições dos objetos IoT para garantir que estejam dentro do intervalo especificado para os objetos IoT; as mensagens que estão fora do domínio são descartadas;
- Criação de metadados: identificação e listagem de alguns metadados importantes (local onde os dispositivos estão operando, por exemplo); e
- Organização de séries temporais: agrupa os dados coletados, organizando-os com base em uma janela de tempo.

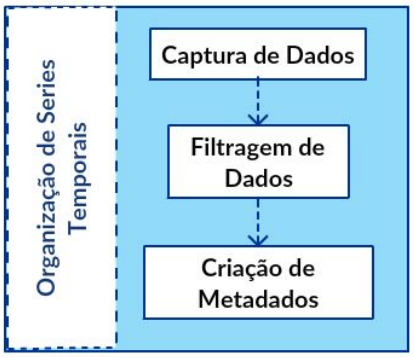

Figura 4.4. Módulos do componente coleção de dados.

A Figura 4.5, destaca o fluxo de dados durante a coleção de dados. Aqui tem início a captura de dados pelo módulo captura de dados. Em seguida, o módulo de filtragem de dados verifica o domínio dos dados coletados. A criação de metadados é feita a partir de dados filtrados. O módulo de organização de séries temporais não segue o fluxo de execução dos módulos anteriores, pois, ele é executado simultaneamente com os processos de outros componentes que atuam sobre os dados processados. Todos os módulos definidos pelo componente de coleção de dados estão compostos por processos que são executados paralelamente. Assim, é possível aumentar o desempenho quando se trabalha com processamento de grandes cargas de dados.

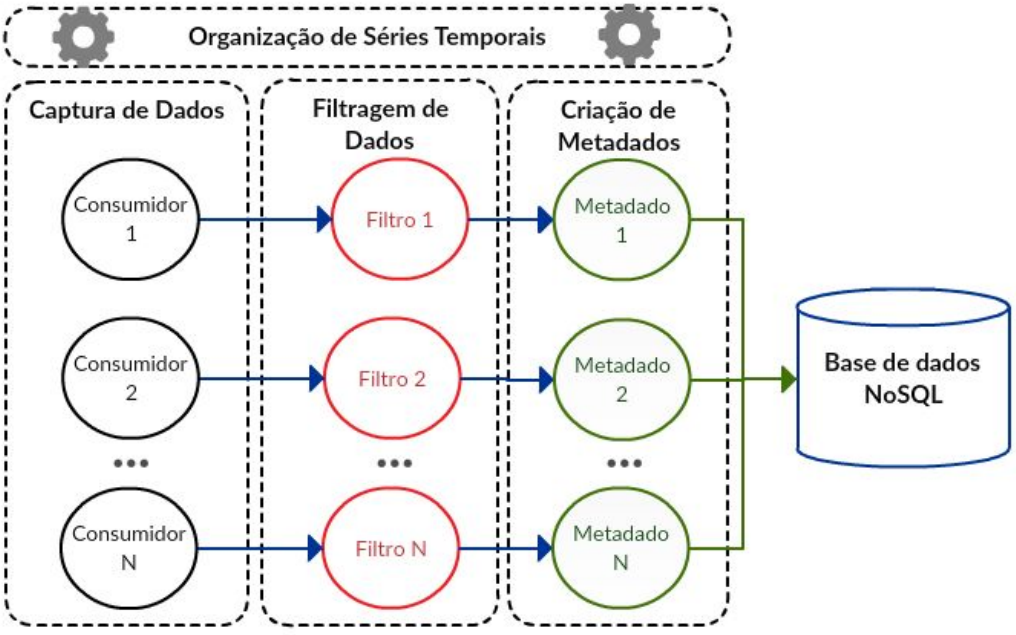

Figura 4.5. Fluxo de dados do componente de coleção de dados.

O módulo de captura de dados contém processos denominados consumidores, que recebem as mensagens provenientes da interface de comunicação de dados, armazenando mensagens sem qualquer processamento em uma tabela eventos, tornando-as, em seguida, disponíveis para serem processadas. Depois de capturados, os dados são armazenados no GDDIoT sem qualquer modificação, a fim de manter um registro de dados históricos, cujo tempo de retenção pode ser configurado. Este módulo executa vários processos simultâneos fazendo uso do modelo de comunicação de *publish*/*subscribe*.

A filtragem de dados verifica o domínio das mensagens coletadas pelo módulo de captura de dados. Uma característica importante deste módulo é que os processos são distribuídos e executados paralelamente. A filtragem de dados requer a consulta no banco de dados das informações para tomar decisões de acordo com as regras de filtragem. Por exemplo: a verificação analisa se as medições feitas pelos dispositivos/sensores estão na gama de valores especificados para eles. As mensagens que não correspondem à regra são descartadas. A filtragem de dados se dá conforme o tipo de objeto, uma vez que a variedade de objetos gerenciados pelos *middlewares* é alta. Os métodos de filtragem incluem os dados de entrada, bem como uma regra de implantação – que permite a filtragem de acordo com o tipo de objeto. Tais processos de filtragem podem ser executados em paralelo, de acordo com as demandas do ambiente IoT.

O módulo de criação de metadados obtém metadados relevantes das mensagens filtradas, tais como: a localização onde os dispositivos/sensores estão operando, o *timestamp* onde uma medição foi realizada e as especificações dos dados – frequentemente utilizados para descrever as medições de dispositivos/sensores.

Semelhante ao módulo de filtragem de dados, este módulo pode ser distribuído e executado paralelamente. No GDDIoT foram definidos dois grupos de metadados, a saber: necessários; e, adicionais. Os metadados necessários são compostos por atributos mínimos necessários ao funcionamento do GDDIoT, tais como: *timestamp*, localização geográfica, unidade da medição e tipo de dado – considerados essenciais para a descrição de dados provenientes de ambientes IoT. Os campos dos metadados necessários são organizados em um esquema com relações predefinidas para suportar a recuperação eficiente dos dados. Em geral, as medições de cada dispositivo/sensor deve acrescentar metadados que descrevem o formato dos dados. Isto é importante para a otimização das consultas, bem como o melhor entendimento dos dados, o que implica que o processamento das consultas

necessitam de adaptação a uma ampla gama de formatos de dados que podem não ser totalmente conhecidos antecipadamente.

Os metadados, além de descreverem os dados IoT, podem ser utilizados para descreverem as aplicações IoT. Por exemplo, alguns metadados podem ser utilizados para descrever uma aplicação de domínio específico (aplicação IoT para agricultura, por exemplo). Para receber metadados adicionais, um atributo denominado metadados adicionais é definido como uma lista de metadados específicos para descrever detalhadamente os dados (o domínio de uma aplicação específica, por exemplo).

Os processos consumidores, filtros e metadados podem lidar com grandes volumes de processamento, podendo se adaptar a diferentes níveis de paralelismo, conforme as necessidades do ambiente. Tal fato significa que quando é necessária maior capacidade de processamento ou armazenamento, é possível a adição de mais nós ao *cluster*, além de aumentar o nível de paralelismo dos processos. O nível de paralelismo pode ser aumentado, criando mais processos quando é exigido ou diminuído o número de processos por ocasião da baixa demanda de processamento. Os processos em questão são executados de acordo com um fluxo de dados, tendo início no consumidor, passando pelos filtros para alcançar os metadados, e finalizando com o armazenamento dos dados em um banco de dados NoSQL.

Os bancos de dados necessitam oferecer suporte aos requisitos dos ambientes IoT para o processamento de grandes volumes, análise em tempo real e diferentes níveis consistentes dos dados. Além disso, os dados IoT requerem melhor flexibilidade pela variedade dos dados, agilidade para o processamento dos dados e escalabilidade. Neste contexto, os bancos de dados NoSQL são mais adequados para o gerenciamento do *big data* gerados pelos ambientes IoT devido à flexibilidade, disponibilidade e grande escalabilidade (Li et al., 2012; Zhu, 2015).

O módulo de organização de séries temporais tem por norte agrupar e ordenar os dados com base em uma janela de tempo. O tamanho desta janela de tempo é uma questão importante, tendo em vista que pode ser configurado de acordo com as características do ambiente computacional, tais como: espaço de memória livre, espaço de armazenamento disponível, velocidade onde as mensagens são encaminhadas, entre outras. Tal configuração é utilizada para melhorar o desempenho do módulo em relação ao espaço de

armazenamento e o desempenho da consulta para os componentes GDDIoT, entre outros aspectos.

A Figura 4.6, a seguir, apresenta um exemplo do funcionamento da janela de tempo para um fluxo de dados constante. No primeiro passo (Fase 1) se tem uma janela de tempo t_0 com nove conjuntos de dados armazenados no banco de dados. Quando t_0 atinge o valor da janela de tempo, os dados são agrupados, classificados e armazenados no banco de dados. Na fase 2, é possível observar outros nove grupos de dados que alcançaram a janela de tempo t_1 e estão prontos para serem agrupados, classificados e armazenados no banco de dados. Na fase 3 têm-se dois grupos de dados (blocos em verde) já classificados e dois novos grupos de dados chegando que pertencem à próxima janela do tempo e serão agrupados e ordenados quando a janela de tempo t_2 for alcançada. Tal processo decorre de forma contínua no tempo, desde que ele possua dados coletados.

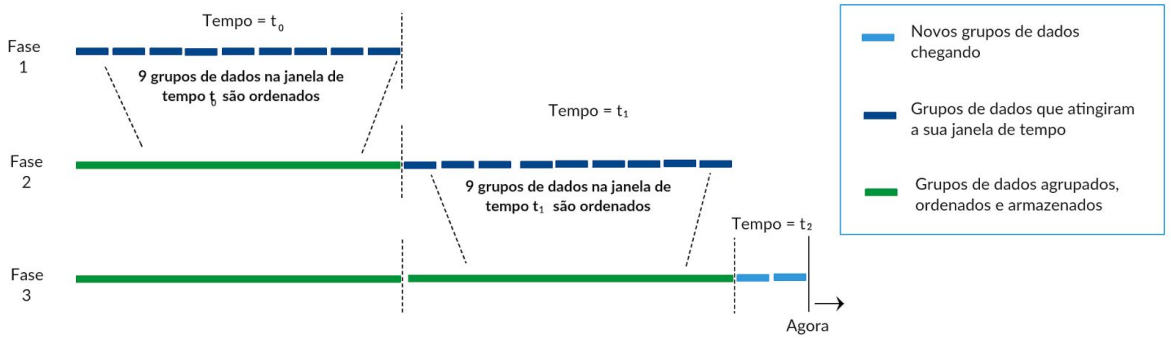

Figura 4.6. Exemplo de compactação de séries temporais.

4.3 – COMPONENTE DE AGREGAÇÃO DE DADOS

O componente agregação de dado é responsável por fornecer processos de solução eficientes (por exemplo, sumarização de dados para melhorar o tempo de resposta das consultas) para responder às solicitações, considerando as limitações de recursos, tais como limitação do armazenamento e processamento. Um método eficiente de agregação de dados é necessário para ampliar o tempo de vida dos dados coletados dos objetos (Sang et al., 2006). O GDDIoT implementa a agregação de dados através do componente de agregação de dados, que implementa dois módulos (vide Figura 4.7), a saber: sumarização de dados e extração de contexto. O processo de sumarização de dados desencadeia três processos, quais sejam: resumo de dados por minuto, resumo de dados por hora e resumo de dados por dia. A sumarização de dados é realizada fazendo uso do atributo temporal,

tais como minutos e horas, uma vez que os dados do IoT possuem características de séries temporais. Os resultados de tais processos são armazenados em tabelas sumarizadas. O módulo de extração de contexto fornece captura e gera variáveis de contexto sobre informações relevantes (geolocalização dos objetos, por exemplo).

Figura 4.7. Módulos do componente de agregação de dados.

A sumarização de dados é o processo da redução da quantidade de dados, porém, mantendo seus significados-chave (Hahn et al., 2000; Zechner, 1997). Neste sentido, o módulo de sumarização de dados representa o agrupamento de um conjunto de dados brutos de um objeto durante um período de tempo, reduzindo o uso de memória e custo computacional para o processamento de grandes quantidades de dados produzidos pelos objetos. Por esta razão, a representação semântica dos dados sumarizados é tão importante quanto a anotação de *stream* de dados. Este módulo reduz e cria uma nova forma de representação, uma vez que o componente coleção de dados armazena os dados coletados em uma tabela eventos que contém os atributos das mensagens coletadas, que é monitorada pelo componente de agregação de dados, produzindo resultados que são armazenados em diferentes tabelas sumarizadas. Assim, se tem um suporte na realização de consultas mais eficientes.

A Figura 4.8, apresenta as tabelas sumarizadas por_minuto, por_hora e por_dia. A tabela por_minuto destaca os seguintes atributos: id_dispositivo, data, por_min (minuto), total (valor total em um minuto), contagem (número de medições em um minuto) e contexto (o último lugar onde foi feita a medição). A tabela por_hora é semelhante à tabela por_minuto. Finalmente, a tabela por_dia apresenta os seguintes atributos: id_dispositivo, data, total (valor total em um dia) e contagem (número de medições em um dia).

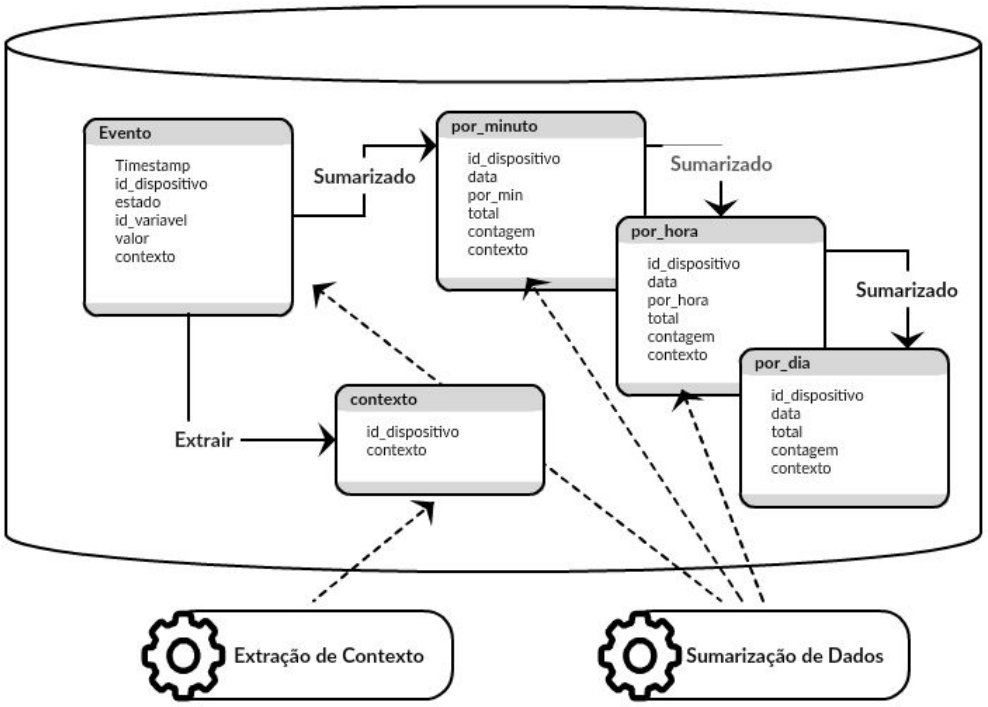

Figura 4.8. Esquema do componente de agregação de dados.

O módulo de extração de contexto fornece a captura de variáveis de contexto sobre informações relevantes tais como o padrão do comportamento dos objetos que fazem parte de um ambiente IoT. A informação de contexto é qualquer informação relevante sobre ambientes, usuários, sistemas, entidades ou eventos. No contexto dos ambientes IoT, as informações de contexto são utilizadas para fornecer mais informações e contexto sobre os objetos que fazem parte do ambiente, a fim de melhorar a análise dos dados IoT. As medições associadas a um objeto (tempo, data, localização e outras propriedades) criam dados de contexto (O'Leary, 2013). Assim, o módulo de extração de contexto é implementado por um processo semelhante ao módulo de sumarização destinado a extrair o atributo de localização da tabela eventos para serem armazenadas na tabela de contexto, fornecendo a posição exata de um objeto quando ocorre algum evento.

4.4 – COMPONENTE DE VISUALIZAÇÃO DE DADOS

O componente de visualização de dados é responsável pela exibição gráfica dos dados coletados e dos dados processados a partir de diferentes *middlewares* de IoT. Formam o componente de visualização de dados (vide Figura 4.9) os seguintes módulos: painéis e consulta de dados. Este componente adota uma arquitetura orientada a serviço.

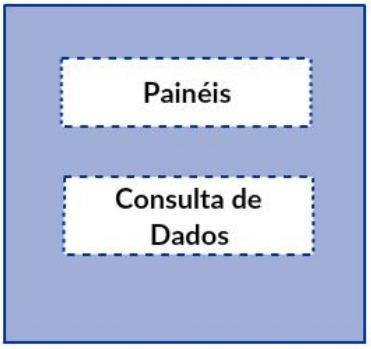

Figura 4.9. Módulos do componente de visualização de dados.

O módulo de painéis apresenta graficamente os dados coletados pelos componentes de coleção de dados e agregação de dados. Este deve suportar o constante envio de dados, bem como deter a capacidade de mostrar o comportamento dos dados em tempo real. Os painéis evidenciam os dados IoT para entender o comportamento dos dispositivos/sensores.

A consulta de dados é um tema importante para os bancos de dados. Uma consulta representa um estado dos dados de entidades definidas no esquema do banco de dados, e detalha a forma como as instruções podem ser executadas para a obtenção de um conjunto específico de dados armazenado no banco de dados. Neste contexto, o módulo de consulta é responsável pelo acesso aos dados coletados e/ou agregados, disponibilizando-os em painéis. Além disso, é possível fazer consultas *ad hoc* sobre os dados armazenados no banco de dados. O módulo de consulta de dados provê a consulta de forma gráfica e através de painéis.

A consulta de dados de forma gráfica se dá via interface GUI (*Graphical User Interface*), onde o usuário pode escolher a fonte de dados ou criar uma nova conexão para outra fonte de dados, possibilitando trabalhar com diferentes fontes de dados. As consultas podem ser montadas fazendo uso de diferentes campos que são especificados no esquema de banco de dados. Finalmente, após a montagem da consulta, o resultado pode ser exibido através de um painel e, de acordo com os resultados, pode ser modificado para mostrar o resultado esperado.

4.5 – CONSIDERAÇÕES

Neste capítulo foi apresentado a definição da arquitetura do GDDIoT. Foi detalhado cada um dos componentes do GDDIoT. Cada componente desta arquitetura foi projetado para atender objetivos específicos para atender os requerimentos enquanto ao gerenciamento de dados vindos de um ambiente IoT.

Como resultado é apresentado uma arquitetura que abrange vários requisitos. Tais como, integrar diferentes *middlewares* IoT de forma simultaneamente, lidar com a heterogeneidade dos dados, captura de grandes quantidades de dados, processamento distribuído e paralelo, tratamento dos dados, visualização de dados, entre outros requisitos.

Após a definição da arquitetura, cada componente é detalhado para atingir os requisitos desejados. Foram encontrados diferentes desafios durante a definição de cada componente integrante do GDDIoT. No componente de Interface de Comunicação teve o desafio da integração de diferentes *middlewares* IoT, a definição dos protocolos de comunicação, padronização dos dados, entre outros. Entre os desafios encontrados na definição do componente de Coleção de Dados teve que atender o requisito da captura de dados a altas taxas em tempo real, tratamentos dos dados inválidos ou incorretos e lidar com a heterogeneidade dos dados. Além disso, foi definido que todos os processos devem ser distribuídos e executados em paralelo para dar suporte aos requerimentos de processamento de grandes quantidades de dados em tempo real. Os desafios encontrados quando definido o componente de Agregação de Dados tais como a definição das variáveis de contexto, e a definição das variáveis para a sumarização dos dados tais como o tempo e localização geográfica. Finalmente, no componente de Visualização de Dados teve que se definir mecanismos adequados para apresentar de forma gráfica o tipo específico de dados usados pelo GDDIoT.

Finalmente, é realizada a comparação dos trabalhos relacionados com a arquitetura proposta nesta tese. A Tabela 4.1 apresenta a comparação do gerenciamento de dados IoT feitos pela maioria de *middlewares* IoT chamada de "Gerenciamento de dados em *Middlewares* IoT", Gerenciamento de dados em Big Data e o GDDIoT. Nesta comparação foi analisado o gerenciamento de dados realizado baseado nas seguintes propriedades: Armazenamento de Dados (AD), Agregação (A), Compressão (C), Filtragem (F) e Não Suportado Completamente (Não*).

A Tabela 4.1 apresenta um resumo do gerenciamento de dados feitos pelos trabalhos relacionados e o GDDIoT. É evidenciado que nem todos os *middlewares* armazenam os dados coletados, isto devido aos *middlewares* serem mais interessado no gerenciamento dos dispositivos. Enquanto a compressão dos dados, o GDDIoT não é realizada de forma especializada, mas através da agregação de dados a quantidade e volume de dados é reduzida mesmo não usando um algoritmo de compressão específica.

Tabela 4.1. Comparação do Gerenciamento de dados IoT.

		Pré - Processamento de dados		
Gerenciamento de Dados IoT	**AD**	**A**	**C**	**F**
Gerenciamento de dados em *Middlewares* IoT	Não*	Sim	Não	Não
Gerenciamento de dados em Big Data	Sim	Sim	Não	Não
GDDIoT	Sim	Sim	Não*	Sim

4.6 – LIMITAÇÕES DA PESQUISA

Esta tese está focada no gerenciamento dos dados IoT provenientes de diferentes ambiente IoT. Durante a definição da arquitetura do GDDIoT, este trabalho focou-se na integração de diferentes fontes de dados simultaneamente, coleção de dados, tratamento de dados no que diz respeito a filtragem e agregação de dados, organização dos dados e o armazenamento de grandes volumes de dados, criando um sistema para o gerenciamento de dados para Internet das Coisas.

No que tange às limitações, este trabalho não trata algumas propriedades relacionadas ao gerenciamento de dados tais como compressão de dados e gestão de conhecimento. Por exemplo, a compressão dos dados é delegado ao banco de dados o qual pode ou não ser implementado por algum mecanismos de compressão de dados do banco de dados. Além disso, esta tese não trata a configuração automática dos componentes e/o módulos integrantes da arquitetura do GDDIoT.

5 – IMPLEMENTAÇÃO DO GDDIoT

Devido ao grande volume de dados gerados pelos ambientes IoT, a arquitetura do GDDIoT deve ser capaz de processar diferentes tarefas de forma paralela, para que o processamento seja feito rapidamente, com alta disponibilidade. Neste contexto, os diferentes componentes do GDDIoT fazem uso de diferentes tecnologias para dar suporte aos requisitos de sua arquitetura, tais como: o gerenciamento de dados em tempo real, o armazenamento de grandes volume de dados, a disponibilidade e o processamento distribuído e paralelo.

A arquitetura do GDDIoT deve ser capaz de gerenciar os dados de diferentes ambientes IoT. Assim, o gerenciamento em questão está preparado para conectar com diferentes sistemas (*middlewares* IoT, por exemplo) de modo simultâneo – o que implica em coletar, processar, organizar e armazenar grandes quantidades de dados. Além disso, a arquitetura supramencionada logra apresentar os dados de forma gráfica.

5.1 – TECNOLOGIAS UTILIZADAS PARA IMPLEMENTAÇÃO DO GDDIoT

Os módulos do componente de coleção de dados fazem uso de diferentes tecnologias (vide Figura 5.1, a seguir). Para implementar a interface de comunicação GDDIoT com os *middlewares* IoT é utilizado o *framework* Tornado (Dory et al., 2012), capaz de realizar a comunicação assíncrona dentro de um sistema. O módulo de captura de dados faz uso do Apache Kafka (Garg, 2013) como plataforma para implementação deste módulo devido às suas características, que lida com enorme quantidade e atendendo as restrições temporais de dados em tempo real. Os módulos de filtragem de dados e a criação de metadados utilizaram o Apache Storm (Jain et al., 2014) como plataforma para processamento de dados em tempo real devido ao bom desempenho. Além disso, tanto os dados processados e não processados são armazenados no banco de dados NoSQL Cassandra (Lakshman & Malik, 2010).

Os módulos do componente de agregação de dados fazem uso do Apache Storm e do Apache Cassandra como plataforma para o processamento e armazenamento da sumarização de dados. Finalmente, o componente de visualização de dados é

implementado através do Grafana (Ödegaard, 2016) que dá suporte a visualização dos dados provenientes dos componentes coleção de dados e agregação de dados.

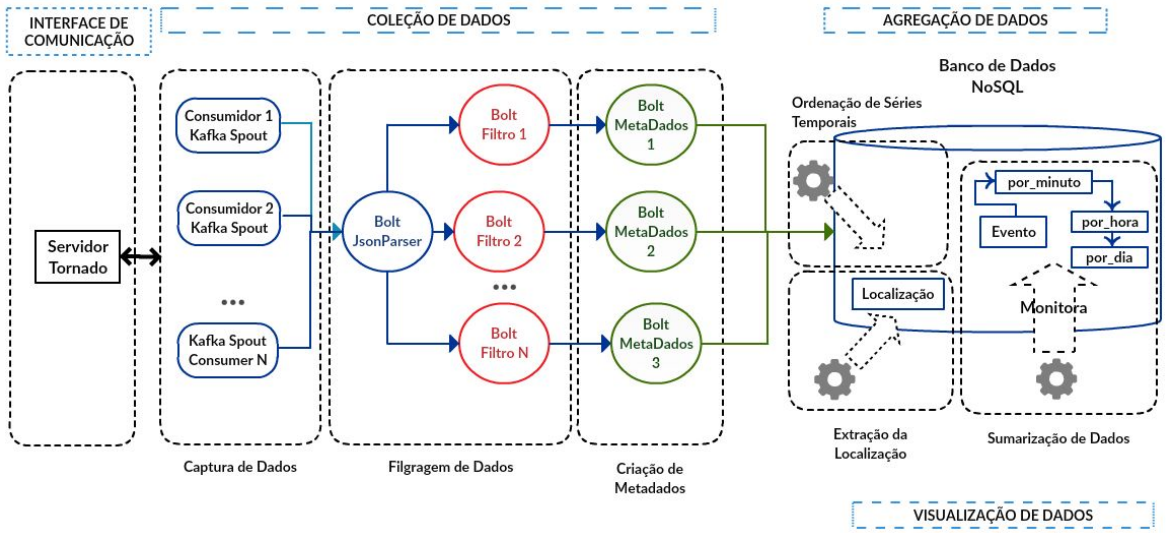

Figura 5.1. Componentes do GDDIoT e tecnologias usadas.

5.1.1 – Tornado

O Tornado é um servidor *web* escalável e assíncrono escrito em Python. Tal *framework* foi projetado para lidar com altas taxas de tráfego de dados. Pode ser utilizado em uma variedade de aplicações. Entre as características deste *framework* tem-se: alto desempenho e a interação assíncrona com serviços externos (Dory et al., 2012). Também pode escalar dezenas de milhares de conexões, sendo ideal para aplicações que necessitam de uma conexão duradoura para diferentes clientes, bem como suportar a comunicação via *webSocket* – um protocolo que permite a comunicação bidirecional por canais *full-duplex* sobre um único soquete, que faz uso do protocolo TCP: projetado para ser executado em navegadores e servidores *web*. O webSocket é muito utilizado em conexões em tempo real, sendo de baixa latência entre os clientes e o servidor (Pimentel et al., 2012).

O *framework* Tornado pode ser dividido em quatro componentes principais, a saber: *framework* web; implementação HTTP do lado do cliente e servidor; uma biblioteca de comunicação assíncrona e I/O não bloqueável, e a biblioteca coroutine, que permite a implementação de processos multitarefa.

No contexto dos ambientes IoT, o protocolo de *webSocket* pode dar suporte a um dos requerimento das aplicações IoT – comunicação em tempo real – devido a sua baixa latência entre clientes e servidores. Uma vantagem do *webSocket* é o dado de transmissão bidirecional em comparação com o protocolo HTTP. Além disso, a quantidade de

requerimento e o tamanho da mensagem do *webSocket* são maiores do que HTTP (Fette, 2011; Pimentel et al., 2012; Wang et al., 2013).

5.1.2 – Apache Kafka

O Apache Kafka é um sistema de mensageria *publish*/*subscribe* distribuído, de código aberto, frequentemente utilizado como um *kernel* para arquiteturas que lidam com *stream* de dados (Garg, 2013). Ele foi projetado com as seguintes características:

- Mensagem persistente: para obter o valor real de grandes quantidades de dados, qualquer perda de informação não é permitido; foi projetado com estruturas que proporcionam desempenho de em tempo constante (O(1)) mesmo com volume excessivo de mensagens armazenadas, o que está na ordem de Terabyte (TB).
- Alto desempenho: projetado para trabalhar com *hardware* básico e suportar milhões de mensagens por segundo;
- Distribuído: suporta a distribuição de mensagens sobre servidores Kafka; o consumo das mensagens é distribuído entre todas as máquinas consumidoras;
- Suporte a múltiplos clientes: pode ser integrado com vários clientes de diferentes plataformas (Java, .Net, PHP, Ruby e Python);
- Tempo real: as mensagens produzidas são imediatamente visíveis para os consumidores – característica essencial para sistemas com base em eventos.

Conforme a Figura 5.2, a arquitetura Kafka é semelhante a muitos sistemas de mensagens *publish*/*subscribe*. Sua composição se dá por produtores, tópicos e consumidores. Os produtores são processos que armazenam as mensagens nos tópicos. O tópico é uma estrutura onde são publicadas as mensagens. Os tópicos Kafka podem ter zero, um ou muitos processos consumidores que se inscrevem aos dados armazenados. Para cada tópico, os dados podem ser particionados em entidades denominadas partições, que são replicadas e distribuídas em diferentes nós de seu cluster. Dentro de uma partição, as mensagens são indexadas e armazenadas em conjunto com um *timestamp*. Finalmente, os consumidores são processos que podem consultar mensagens das partições dos tópicos (Kreps et al., 2011).

No contexto dos ambientes IoT, o Kafka pode dar suporte ao requerimento do processamento do dados em tempo real devido à baixa latência, à confiabilidade e ao alto

rendimento. Pode ser utilizado para o consumo de grande volume de dados proveniente do monitoramento de métricas operacionais de aplicações distribuídas em ambientes IoT. Neste contexto, os eventos podem ser registrados e ordenados segundo a mudança de estado de ditos sistemas distribuídos. Além disso, o Kafka tem a propriedade de suportar vários clientes simultaneamente – característica importante para gerenciar os dados vindos de diferentes ambientes IoT.

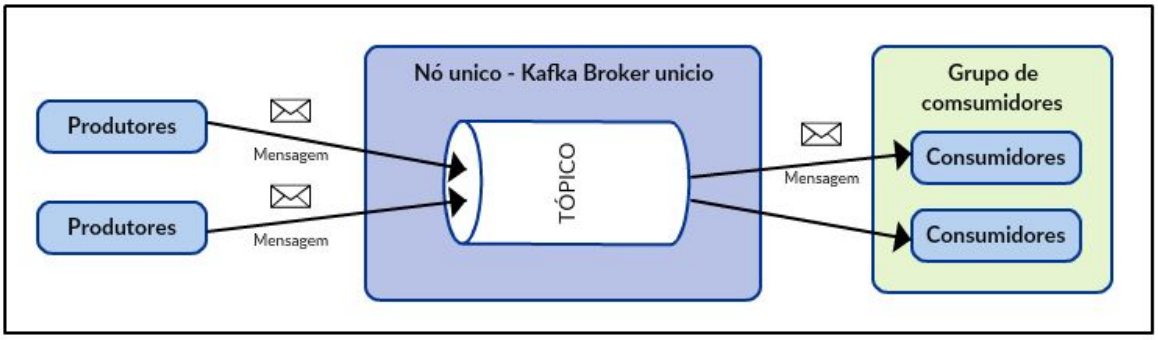

Figura 5.2. Arquitetura fundamental do Kafka (Garg, 2013).

5.1.3 – Apache Storm

O Apache Storm é um *framework* distribuído de processamento em tempo real para o processamento de grande volume de dados em alta velocidade (Jain et al., 2014). É uma arquitetura baseada no paradigma mestre-escravo (Namiot, 2015). Um *cluster* Storm consiste de um nó mestre (Nimbus) e nós escravos (Supervisores), com a coordenação feita pelo Zookeeper (Junqueira et al., 2013). A Figura 5.3, evidencia os componentes de um *cluster* Storm. O servidor mestre (Nimbus) é responsável pela distribuição do código pelos diferentes nós do *cluster* e o monitoramento dos processos. Um *cluster* Storm pode ter um ou mais nós supervisores, onde os processos são executados. Os nós supervisores se comunicam com o Nimbus através do Zookeeper. O Nimbus envia sinais ao supervisor para iniciar ou parar os processos. O Zookeeper é um serviço de coordenação distribuída de alto desempenho para manter a informação de configuração, bem como delegar e prover a sincronização distribuída e os grupos de serviços, sendo necessário para a coordenação do *cluster* Storm. Além disso, o Apache Storm implementa a topologia Storm: um diagrama de fluxo composto por processos denominados *Spout* e *Bolt*. Um *Spout* é uma fonte de *stream* de dados em uma topologia, e os *Bolts* podem realizar algum tipo de processamento sobre os *stream* de dados (Jain et al., 2014).

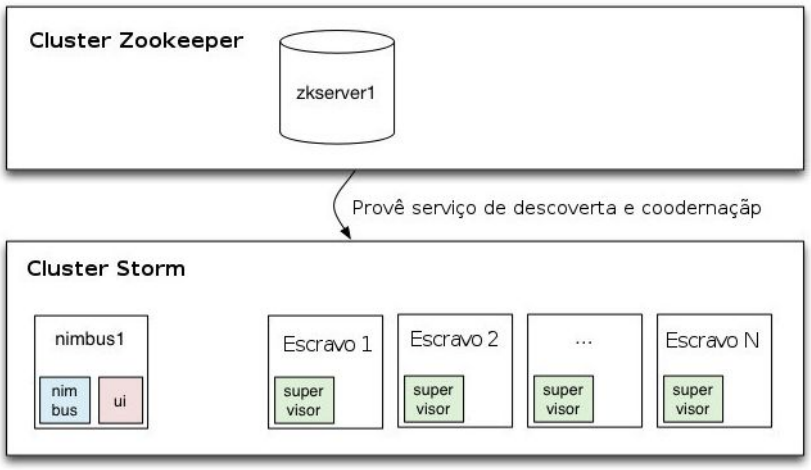

Figura 5.3. Componentes de um *cluster* Storm, adaptado da figura (Noll, 2015).

A seguir, tem-se um detalhamento dos componentes do *framework* Storm:

- Topologia: é um grafo composto de *Spouts* e *Bolts* (vide Figura 5.4). Cada nó no gráfico contém alguma lógica de processamento, e as arestas no grafo apontam como os dados serão passados e como o processamento se dará entre os nós. Quando uma topologia é submetida a um *cluster* Storm, o serviço Nimbus no nó mestre consulta os serviços dos nós supervisores, podendo submeter a topologia no *cluster* (Bahga et al., 2014). Cada supervisor cria um ou mais processos de trabalho, cada um com seu próprio ambiente de trabalho separado. Cada processo executa dentro de si *threads* – denominados executores. O *thread*/executor processa as tarefas computacionais: *Spout* ou *Bolt;*
- *Stream*: é uma abstração importante no Storm. Um *stream* é uma sequência ilimitada de tuplas (coleção de pares de chave-valor). Uma tupla é a estrutura de dados mais básica no Storm. Cada campo nos valores pode ser objeto de qualquer tipo serialização;
- *Spout*: é o ponto de entrada em uma topologia do Storm. É a fonte de *streams* na topologia Storm. Um *Spout* pode se conectar as fontes de dados (mensagens advindas do Kafka, por exemplo). Tal processo obtém dados contínuos, convertendo-os em *stream* dados e emitindo os *streams* para os *Bolts*, para seu processamento. Os *Spouts* são executados como tarefas através de *threads* executores.

- *Bolt*: contém a lógica do processamento. Os *Bolts* podem transformar os *stream* de dados, desde simples transformações até transformações mais complexas. Um *Spout* recebe streams de um ou mais *Spouts* ou *Bolts*. Os *Bolts* podem realizar diferentes tipos de processamento, tais como: filtrar as tuplas, fazer agregações de *streams*, juntar *streams*, interagir com bancos de dados etc.

A Figura 5.4, evidencia uma topologia Storm com um *Spout* e três *Bolts*. Os *Bolts* 1 e 2 estão subscritos ao *Spout* e consomem as *streams* emitidas pelo Spout. A saída dos *Bolts* 1 e 2 são consumidas pelo *Bolts* 3.

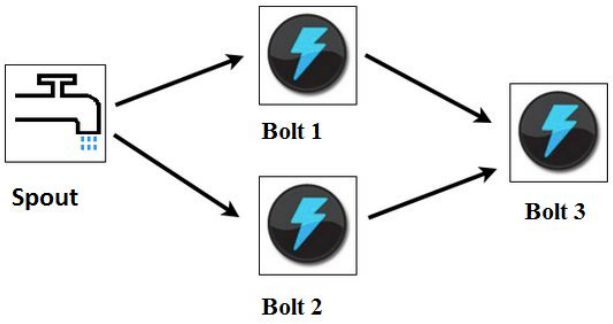

Figura 5.4. Exemplo de uma topologia Storm.

No contexto dos ambientes IoT, o Storm pode dar suporte ao requerimento do processamento em tempo real de grande quantidade de dados com características temporais (Yasumoto et al., 2016b). Além disso, o Storm dá suporte ao requerimento da arquitetura GDDIoT para o processamento dos dados de forma distribuída e paralela.

5.1.4 – Apache Cassandra

O Apache Cassandra é um sistema gerenciador de banco de dados desenvolvido inicialmente pelo Facebook (Lakshman et al., 2010). Posteriormente, o projeto foi disponibilizado como *open source* para a comunidade sobre a guarda da fundação Apache. Apache. Para a referida plataforma, os requisitos são: alta disponibilidade, desempenho, confiabilidade e arquitetura suficientemente escalável para suportar o crescimento da enorme quantidade de dados produzidas pela plataforma Facebook (Lakshman et al., 2010). Neste contexto, o Cassandra teve como norte na sua arquitetura dois sistemas gerenciadores de bancos de dados NoSQL, quais sejam: BigTable, do Google (Chang et al., 2008); e DynamoDB, da Amazon (Sivasubramanian, 2012). Do BigTable fez-se uso do

modelo de dados colunar, e do DynamoDB, aproveitou-se da sua arquitetura de replicação e divisão de dados.

O Cassandra é orientado a colunas, mas internamente trabalha com o conceito valor-chave. Assim, as consultas se dão por meio das chaves definidas. Porém, suporta a indexação de colunas secundárias, bem como a indexação da chave por faixa de valores de colunas adicionais, além da chave principal. A indexação secundária permite consultas nas colunas adicionais indexadas através da linguagem de consulta do Cassandra (CQL – Cassandra Query Language) (Cockcroft et al., 2011).

A arquitetura do Cassandra considera a possibilidade de ocorrência de falhas de sistema e de *hardware*. Assim, o referido banco de dados trata das falhas fazendo uso de um sistema distribuído *peer-to-peer*, onde todos os nós são iguais e os dados são distribuídos entre todos os nós do *cluster*. Cada nó troca informações constantemente com o *cluster*. Na Figura 5.5, é possível observar o fluxo das operações no Cassandra. Uma escrita sequencial grava dados no *commit log* de cada nó, que captura a atividade de escrita, a fim de garantir a durabilidade dos dados. Os dados também são descritos em uma estrutura na memória denominada *memtable*, que se assemelha a um cache *write-back* (Jouppi, 1993). Uma vez cheia a estrutura de memória, os dados são gravados no disco em grupos de dados por meio de arquivos de dados denominados *sstables*. Todas as escritas são automaticamente repartidas e replicadas em todo o *cluster*. Fazendo uso de um processo denominado compactação, o Cassandra consolida periodicamente *sstables*, descarta os *tombstones* - indicadores de que uma coluna foi excluída, e regenera o índice no *sstable*.

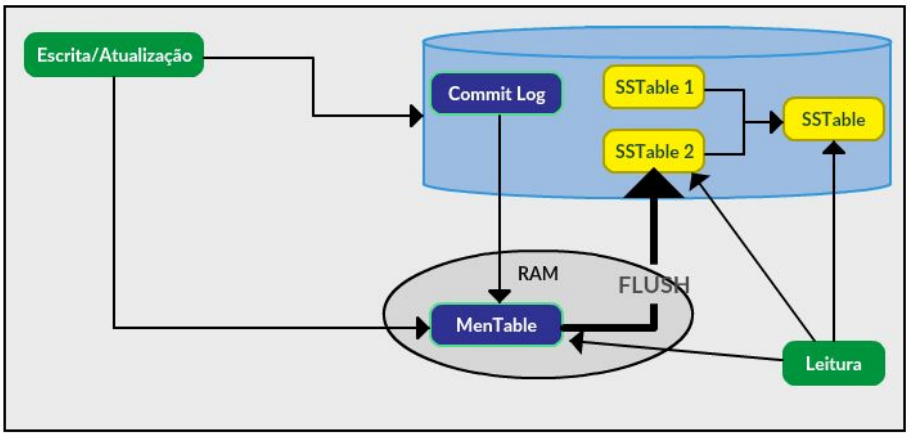

Figura 5.5. A arquitetura do Cassandra (Edureka, 2017)..

A distribuição e replicação de dados são conceitos que estão interligados, pois, o Cassandra é um sistema *peer-to-peer* que faz cópias de dados, distribuindo-as entre um grupo de nós. Os dados são organizados por tabela e identificados por uma chave primária. A chave primária determina em qual nó os dados serão armazenados. As cópias de linha de dados são denominadas réplicas. A Figura 5.6, evidencia um anel sem nós virtuais. Vale salientar que a cada nó é atribuído um único *token*, que representa um local no anel. A escolha do local onde ficará uma linha é determinada pelo mapeamento de chave de partição para um valor de *token* dentro de uma faixa, a partir do nó anterior até o valor designado. Em cada nó também existem cópias de cada linha de outros nós do *cluster*. Conforme demonstrado na Figura a seguir, o dado A que ingressa pelo nó 8 e é replicado nos nós 1, 2 e 3.

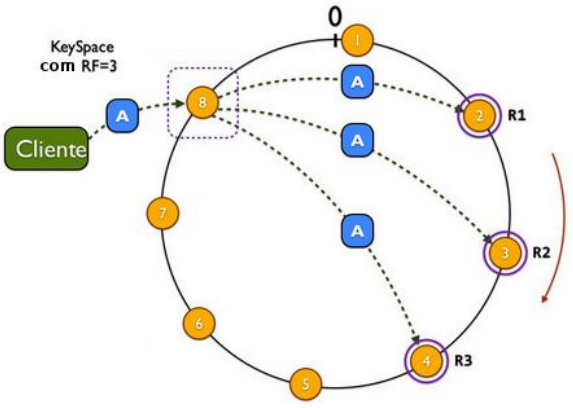

Figura 5.6. Partição e replicação de dados do Cassandra (Edureka, 2017).

No contexto do gerenciamento dos dados do ambiente IoT, o Cassandra, que tem como características: alta disponibilidade, alto desempenho, confiabilidade e escalabilidade, preenche os requisitos necessários (Cockcroft et al., 2011; Lakshman et al., 2010), que podem atender aos requisitos dos dados IoT (o consumo contínuo de grande volume de dados, por exemplo). Além disso, o esquema flexível deste banco de dados pode suportar a variedade dos dados IoT e dar suporte às séries temporais – uma características dos dados IoT – através de DateTieredCompactionStrategy (Lu et al., 2016), o que pode melhorar o desempenho do modelo de organização de séries temporais do componente coleção de dados do GDDIoT.

5.1.5 – Streamparse

A plataforma Streamparse (Streamparse, 2017) permite o processamento paralelo e a execução distribuída de fluxos de dados em tempo real via Apache Storm. Oferece o modelo de programação *map*/*reduce* para o processamento em tempo real de *stream* de dados. Isto pode ser uma forma de escalar os processos com alto paralelismo. Tal ferramenta permite a integração do Apache Kafka, Storm e Cassandra. Além disso, fornece ferramentas para gerenciar o *cluster* Storm e projetos, ajuda na integração das diferentes tecnologias utilizadas na implementação dos componentes do GDDIoT, e facilita a implantação de topologias Storm.

5.1.6 – Grafana

O Grafana é uma plataforma de análise e visualização de métricas. Suporta diferentes gráficos para séries temporais. É composta por um servidor e um cliente *web*. Basicamente, permite aos usuários criar e editar painéis de forma mais fácil. Os usuários podem criar gráficos abrangentes com formatos de eixos (como linhas e pontos). O resultado da renderização no lado do cliente Grafana é rápida, mesmo com longos períodos de tempo.

Como características importantes, a plataforma em questão apresenta:

- Especialização na apresentação de séries temporais com base em uma métrica específica;
- Flexibilidade para a criação de painéis; e
- Suporte a diferentes fontes de banco de dados – possui um editor de consultas específico, que é customizado para as características e capacidades do banco de dados.

No contexto dos ambientes IoT, devido à especialização na visualização de séries temporais, o Grafana pode dar suporte aos dados gerados pelos ambientes IoT, uma vez que os dados IoT tem características temporais.

Após a definição das tecnologias utilizadas para implementação do GDDIoT, nas próximas seções são apresentados como essas ferramentas foram aplicadas na implementação de cada componente e seus módulos do GDDIoT.

5.2 – COMPONENTE DE INTERFACE DE COMUNICAÇÃO

A implementação do componente de interface de comunicação é realizada fazendo uso do modelo *Streaming* API (*Application Programming Interface*) (Morstatter et al., 2014). Uma *Streaming* API mantém aberta a conexão cliente-servidor após a primeira requisição. Diferentemente do protocolo HTTP, quando da utilização do modelo em questão, não é necessário se preocupar em fazer novos pedidos para enviar novos dados da mesma chamada. Para lograr tal processo, é preciso que o cliente esteja preparado para manter a conexão aberta o tempo necessário. Neste contexto, as tecnologias utilizadas atualmente (HTTP, por exemplo) não atendem a implementação do *Streaming* API. Assim, novas tecnologias (*webSockets*, por exemplo) atendem aos requisitos básicos deste modelo, tais como: comunicação *full duplex* e conexão de tempo indeterminado.

O módulo do lado do cliente inicia a solicitação de conexão entre o *middleware* IoT com o GDDIoT. Após a requisição, o GDDIoT encaminha a resposta à solicitação do *middleware* IoT. A conexão é realizada via *webSocket*. Além de estabelecer a conexão, é criada uma *thread*, que é executada de forma contínua para o envio das mensagens ao servidor de *web* Tornado. As mensagens, antes de enviadas ao servidor *web*, são transformadas ao formato padrão utilizado pelo GDDIoT. A conversão das mensagens advindas dos *middlewares* IoT (CSV, JSON e XML, por exemplo) se dá através de uma função implementada neste módulo. Basicamente, esta função pega a mensagem original do sistema e o transforma em uma mensagem JSON. A mensagem JSON é encaminhada ao servidor *web* Tornado

O módulo do lado do servidor é implementado através do *framework* Tornado. Tal módulo implementa um processo que monitora constantemente a chegada de alguma mensagem. O Tornado pode definir uma porta dedicada para a recepção de dados. Uma vez que a mensagem tem chegado ao Tornado, este a encaminhará imediatamente ao componente coleção de dados. As mensagens são enviadas ao módulo de captura de dados através de productores Kafka, que publicam as mensagens através de um tópico Kafka utilizado pelo componente coleção de dados.

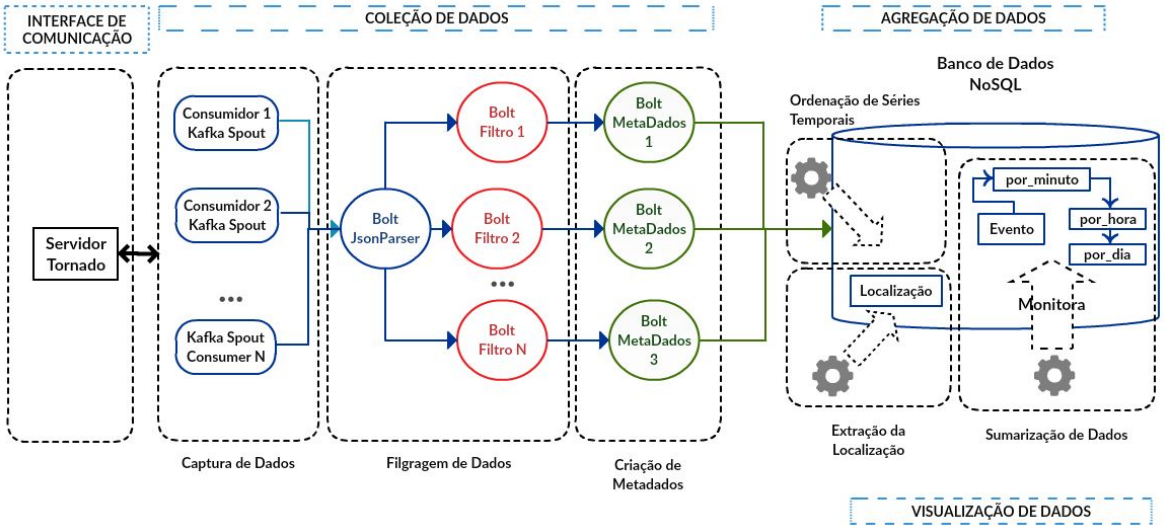

Figura 5.7. Topologia do serviço distribuído de tratamento de dados para IoT.

5.3 – COMPONENTE COLEÇÃO DE DADOS

Para a implementação do componente coleção de dados foram utilizadas as seguintes ferramentas: Streamparse, Kafka, Storm e Cassandra. O *framework* Streamparse foi utilizado para dar suporte à implantação da topologia Storm. A topologia Storm compreende o módulo de captura de dados, o módulo de filtragem de dados e o módulo de criação de metadados.

O módulo de captura de dados possui um número arbitrário de processos consumidores, que são implementados fazendo uso do sistema de mensagens Kafka. Estes consumidores são denominados KafkaSpout (Kreps et al., 2011). A Figura 5.7, apresenta vários consumidores KafkaSpout, o que indica que podem ser definidos múltiplos processos consumidores, sendo executados simultaneamente. Cada KafkaSpout pega as mensagens armazenadas no tópico Kafka e as disponibiliza à topologia como fonte de dados para o módulo de filtragem de dados. Além disso, o módulo em questão armazena as mensagens no banco de dados Cassandra na forma da tabela evento. Basicamente, esta tabela contém as informações encaminhadas pelos *middlewares* IoT. A Figura a seguir destaca detalhadamente os elementos que compõem a tabela evento.

Figura 5.7. Esquema da tabela evento.

O módulo de filtragem de dados verifica o domínio das mensagens coletadas pelo módulo de captura de dados. Este último é composto por dois processos : JsonParser e os filtros. O processo JsonParser é implementado através de um *Bolt* que recebe as mensagens dos consumidores *KafkaSpout*. Este realiza a divisão das mensagens JSON em unidades menores (atributos). Logo, é possível a recuperação de todos os atributos e valores que compõem as mensagens. Entre os atributos recuperados tem-se: nome do *middleware*, identificador do dispositivo, serviço, variáveis de estado e valores da medição do dispositivo. Os filtros são processos implementados mediante diferentes *Bolts*. Cada *Bolt* pode implementar um filtro em específico. Por exemplo, a verificação se as medições feitas pelos dispositivos/sensores estão na gama de valores especificados para eles. As mensagens que não satisfazem à regra são descartadas. A filtragem de dados é realizada de acordo com um tipo específico de dispositivos/sensores. Os métodos de filtragem incluem os dados de entrada e uma regra de implantação que permite implementar a regra por meio de pelo menos um processo a ser executado. Tais processos de filtragem podem ser executados paralelamente, conforme as demandas do ambiente IoT.

O módulo de criação de metadados foi implementado através de *Bolts* denominados metadados, que permitem a criação de metadados das mensagens já filtradas. Os processos de criação de metadados é a terceira etapa da topologia do componente de coleção de

dados. Percebe-se que os três estágios da topologia (processos JsonParser, filtros e metadados) podem ser executados em paralelo. Para lidar com a variedade de dados coletados dos ambientes IoT, é definida a tabela metadado_unidade (vide Figura 5.8) para armazenar as unidades das medições, bem como as regras de conversão das unidades. Esta tabela possui a unidade de origem (unidade_fonte), a unidade equivalente no Sistema Internacional de Unidades (SI) (unidade_SI), a fórmula de conversão para transformar da unidade original para a unidade do SI (regra_fonte_SI) e a fórmula de conversão para transformar da unidade do SI para a unidade original (regra_SI_fonte). Neste contexto, o processo de criação dos metadados faz uso da tabela metadado_unidade para converter o valor da medição da fonte para a unidade do SI e a conversão inversa. Por exemplo, é possível converter graus Celsius para graus Kelvin fazendo uso das fórmulas de conversão da unidade de origem para o SI.

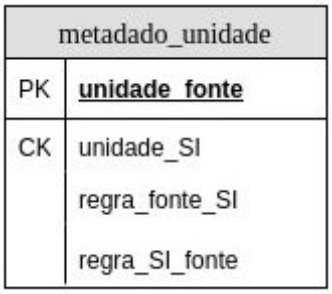

Figura 5.8. Esquema da tabela metadado da unidades.

Finalmente, para armazenar os dados no banco de dados Cassandra, na tabela evento foi definida uma chave composta capaz de dar suporte às consultas. Esta chave está composta pela *partition key* e *clustering key*. O identificador do objeto (dispositivo) é utilizada como *partition key*. O nome do *middleware* e timestamp são utilizadas como *clustering keys*. Esta chave garante a unicidade dos dados, uma vez que os identificadores dos objetos que fazem parte de um ambiente IoT são únicos dentro de um ambiente IoT. Para aumentar a taxa de inserção dos dados dentro do banco de dados Cassandra, é utilizada uma abordagem inserção denominada *microbatching* (Shahrivari, 2014), que consiste em acumular a recepção de mensagens em um pequeno intervalo de tempo (dois segundos, por exemplo), sendo inseridos, posteriormente, de modo conjunto, visando melhoria na taxa de inserção no banco de dados.

5.3.1 – Configuração do Banco de Dados Cassandra

O banco de dados Cassandra foi configurado para suportar séries temporais devido às características temporais dos dados vindos dos ambientes IoT. Tais dados são acomodados em diferentes famílias de colunas configuradas para suportar dados temporais, devido a uma estratégia especializada de compactação denominada *DateTieredCompactionStrategy* (Lu et al., 2016). Esta característica implementada no banco de dados em questão é utilizada pelo módulo de ordenação de séries temporais – estratégia de compactação que pode ser utilizada em cada uma das famílias de colunas que armazena séries temporais dos componentes coleção de dados e agregação de dados.

5.3.2 – Configuração dos Consumidores Kafka

Para processar o grande volume de dados coletados em um tópico, os consumidores Kafka precisam ser configurados corretamente. Em geral, um consumidor Kafka lê os dados de várias partições (P0, P1, P2, ...PN) quando da chegada destes, conforme evidenciado na Figura 5.9. No entanto, mesmo logrando a leitura das partições de forma paralela, o grau de paralelismo do Kafka não é suficiente para gerenciar um grande volume de dados previsto em uma grande rede de objetos de um ambiente IoT.

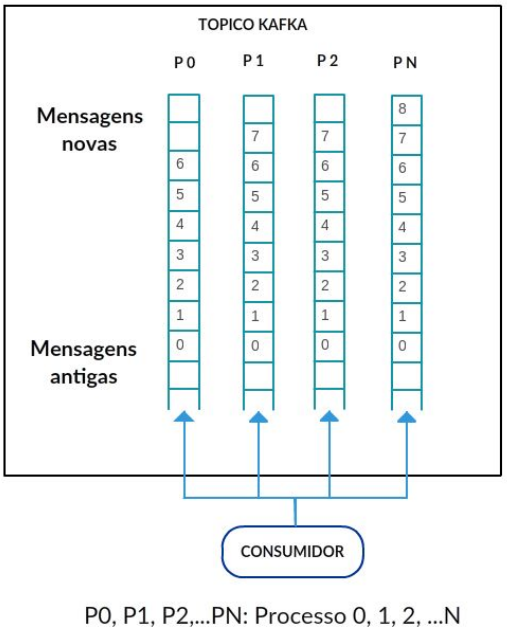

Figura 5.9. Consumidor Kafka.

A fim de melhorar o paralelismo, foi introduzida uma modificação no modo como as partições são lidas pelos consumidores Kafka, definindo um consumidor para cada partição existente, conforme evidenciado na Figura 5.10. Tal modificação aumenta o grau de paralelismo na leitura de partições, uma vez que, ao invés de ter um consumidor lendo várias partições, tem-se um consumidor dedicado à leitura de uma partição unicamente.

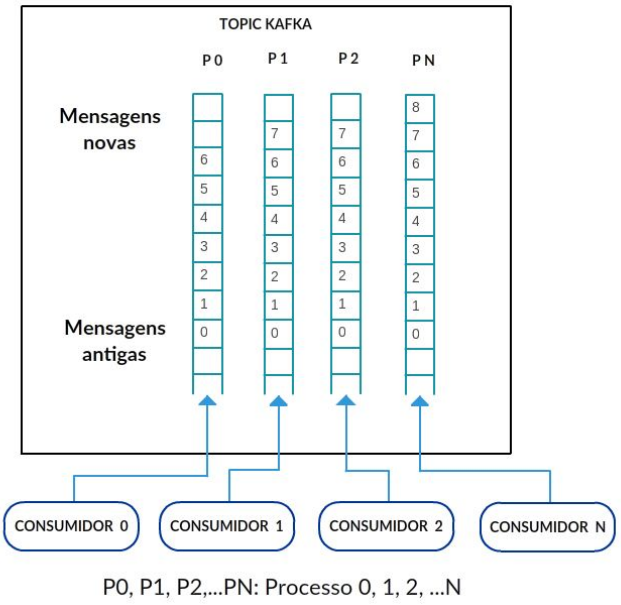

Figura 5.10. Consumidor Kafka modificado.

5.4 – COMPONENTE AGREGAÇÃO DE DADOS

O componente de agregação de dados foi implementado com dois módulos (vide Figura 5.7): sumarização de dados e extração de contexto.

O módulo de sumarização de dados agrupa os dados armazenados na tabela evento no intuito de reduzir a quantidade de dados, porém, mantendo seus significados chaves (Hahn et al., 2000; Zechner, 1997). O processo de sumarização estabelece conexão com o banco de dados Cassandra e executa consultas periódicas sobre a tabela eventos. Tal programa implementa três processos para sumarizar os dados a intervalos de minutos, de horas e de dias, respectivamente. Os resultados destes são armazenados em tabelas sumarizadas que contêm o número de mensagens processadas e o valor total das medições durante um período de tempo (vide Figura 5.11, a seguir), permitindo diferentes cálculos dos dados sumarizados, como a média em um período de tempo determinado..

Figura 5.11. Esquema das tabelas sumarizadas.

Não apenas a agregação de dados pode ser implementada fazendo uso do atributo temporal, mas também com base em outros atributos (geolocalização etc.). A combinação dos atributos determina o nível de agregação dos dados. Por exemplo: é possível combinar as variáveis para agregar os dados desde o nível de região até o nível de município.

O módulo de extração de contexto implementa a extração de localização, o que é especialmente útil quando um dispositivo/sensor está conectado a um objeto em movimento, como um carro ou uma bicicleta. A localização de um objeto pode mudar frequentemente, ou sua localização pode ser relativamente estática, conforme a mobilidade do objeto. Além disso, os dados provenientes de objetos remotos (satélites, imagens e vídeos de câmeras, por exemplo) exigem a localização de origem dos dados (Sheth et al., 2008). E ainda, o módulo de extração de contexto é implementado de modo semelhante ao módulo de sumarização, onde a tabela eventos é utilizada como fonte de dados. Aqui, o processo pretende extrair o atributo de localização da tabela eventos para armazenar a última localização do objeto onde se deu a medição, armazenando, posteriormente, as tabelas sumarizadas no atributo de localização. Os dados são armazenados na tabela de localização dos objetos (localização_objeto). Por meio desta, é possível conhecer a posição atual de cada objeto, conforme evidenciado na Figura 5.12.

localização_objeto	
PK	**id_objeto**
CK	timestamp
	latitude
	longitude

Figura 5.12. Esquema da tabela de localização dos objetos.

5.5 – COMPONENTE VISUALIZAÇÃO DE DADOS

O componente visualização de dados foi implementado fazendo uso da ferramenta de visualização de métricas de séries temporais Grafana. Tal ferramenta pode dar suporte à visualização dos dados IoT devido às suas características temporais. Este componente estabelece conexão com o banco de dados Cassandra, onde estão armazenados os dados dos diferentes ambientes IoT. O processo de conexão do Grafana com o banco de dados Cassandra se dá por meio de um *plugin* denominado KairosDB (KairosDB Team, 2015). Após a conexão com o Cassandra, é possível apresentar os dados coletados e agregados por meio de uma interface gráfica.

O módulo de painéis disponibiliza painéis de controle geral, que monitoram os objetos dentro de um ambiente IoT específico. Por exemplo: painéis que apontam o número de objetos funcionando em diferentes intervalos de tempos, tais como: nos últimos 10 minutos, nos últimos cinco minutos, nos últimos dois minutos, nos últimos 30 segundos, nos últimos cinco segundos, respectivamente. Por exemplo: na Figura 5.13, têm-se os dados gerados por dois semáforos utilizados para controlar o tráfego de veículos a cada cinco minutos, sendo que os intervalos de tempo podem mudar a qualquer momento.

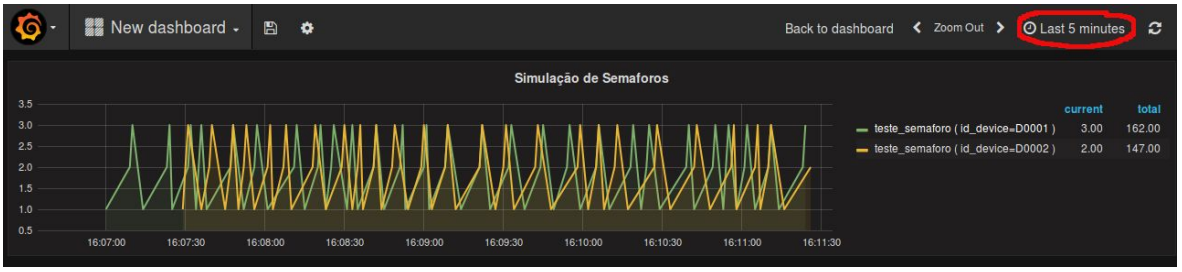

Figura 5.13. Exemplo do painel para apresentar os dados de dois semáforos.

O módulo de consulta de dados, semelhante ao módulo de painéis, disponibiliza painéis de monitoramento customizados. Isto é possível graças à montagem de consulta de dados de forma gráfica. Assim, os usuários não precisam ter conhecimento avançado de linguagens de consulta de dados (SQL, por exemplo) para lograr montar uma consulta personalizada que atenda as necessidades específicas de controle dos objetos e ambientes IoT. Neste contexto, a Figura 5.14, apresenta o editor gráfico onde as consultas podem ser personalizadas. A interface deste editor apresenta alguns elementos tais como campos (*tags*), agrupamentos (*group by*) e operações de agregações (*aggregators*). Além disso, pode ser definido o intervalo de tempo na qual a consulta é executada. O editor de consulta pode variar segundo o banco de dados utilizado. Por exemplo: no caso do Cassandra, as tabelas são denominadas métricas.

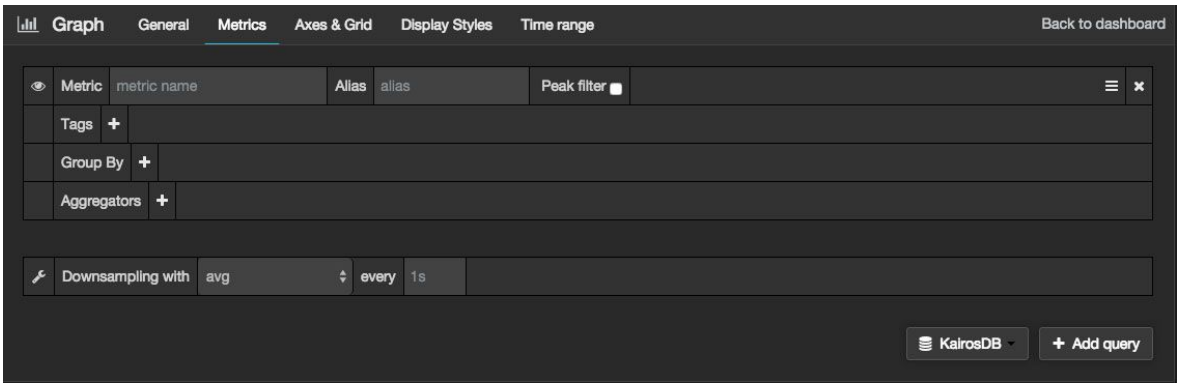

Figura 5.14. Editor de consulta gráfico.

5.6 – ESQUEMA DE DADOS DO GDDIoT

O GDDIoT é composto por três esquemas de dados principais (vide Figura 5.10): esquema de coleta, esquema de agregação e esquema de metadados. O esquema da coleção de dados é composto por uma tabela de eventos responsável para armazenar dados brutos coletados pelo módulo de captura de dados. O esquema de agregação é composto por tabelas sumarizadas. O esquema de metadados é composto por uma tabela para tratar a variedade de dados e uma tabela para armazenar variáveis de contexto (tabela de localização).

O esquema de coleção de dados está composto pela tabela de eventos. Esta tabela tem como chave os campos, id_objeto, *middleware* e *timestamp*. O campo id_objeto é a *partition key*, e os campos *middleware* e *timestamp* são as *clustering keys*. Devido à grande quantidade de objetos que são gerenciados pelos *middlewares* IoT, o identificador dos

objetos faz parte da chave como *partition key*. Este tipo de chave ajuda na distribuição dos dados com base nos diferentes nós que fazem parte do *cluster* Cassandra.

O esquema de sumarização de dados está composto por três tabelas sumarizadas e semelhantes estruturalmente, contendo uma chave composta. Esta chave tem com *partition key* o identificador do objeto, e como *clustering key*, os campos data e a variável temporal minuto, hora e dia. Aquelas tabelas ainda armazenam a contagem de medições e os subtotais referentes a um período de tempo (minuto, hora e dia). Além disso, é contemplado o campo localização, que representa a localização neste período de tempo. Este é processado pelo módulo de extração de contexto.

Finalmente, o esquema de metadados dá suporte ao problema de variedade de dados e o contexto dos objetos. Este esquema apresenta duas tabelas: metadado_unidade e localização_objeto. A primeira tabela armazena informações para lidar com a variedade dos dados. Por exemplo, se a medição de um fenômeno se deu em graus Fahrenheit, esta tabela ajuda na conversão para o SI segundo a regra conversão ali armazenada, sendo possível a inversão do processo. E finalmente, a tabela localização_objeto armazena a informação da localização de cada objeto no tempo onde as medições foram realizadas.

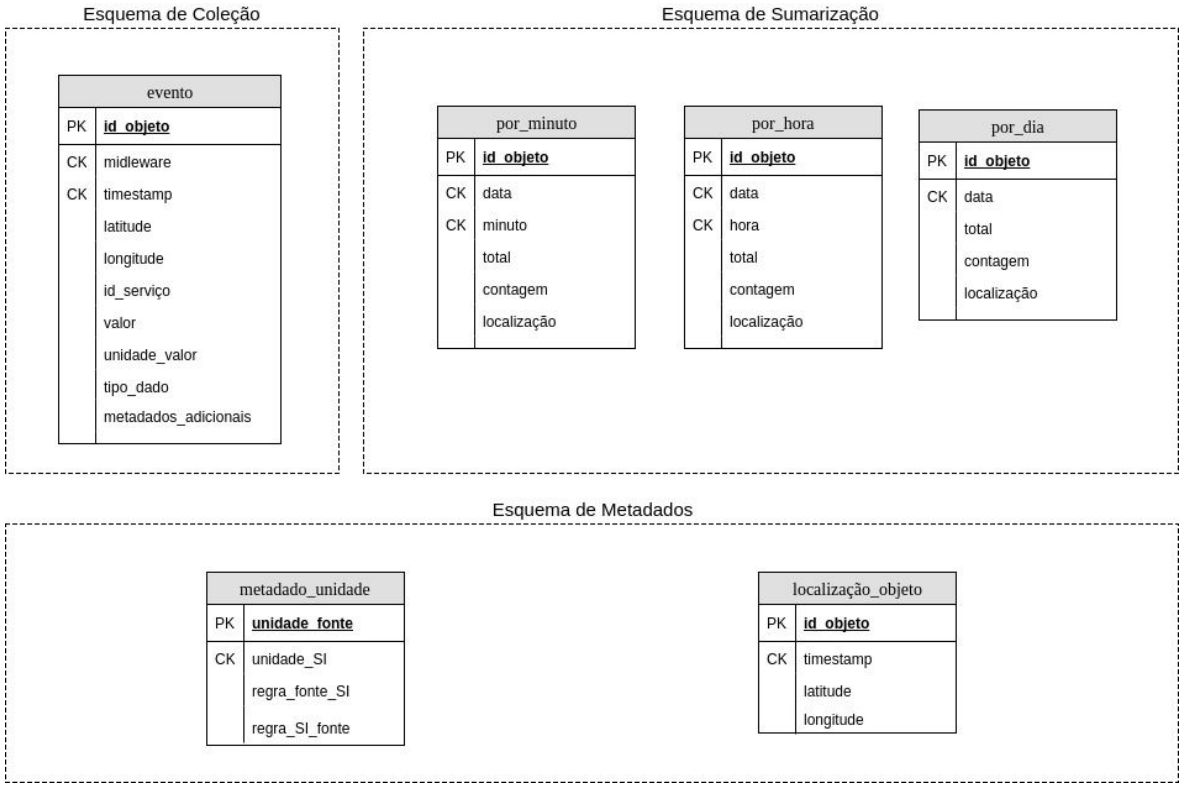

Figura 5.13. Esquema de dados do GDDIoT.

5.7 – CONSIDERAÇÕES

Nesta seção é apresentado a relação dos trabalhos publicados decorrente do desenvolvimento dos componente do GDDIoT. Além disso, é apresentado detalhadamente a implementação do GDDIoT. Nesse contexto, é apresentada de forma mais detalhada dificuldades e motivos pelos quais foram escolhidas as diferentes tecnologias para a implementação do GDDIoT.

Decorrente da implementação dos diferentes componentes do GDDIoT foram escritos trabalhos acadêmicos. Após a conclusão da implementação do Componente de Coleção de dados, o detalhamento deste componente foi publicado no Simpósio Brasileiro de Banco de Dados (SBBD) (Huacarpuma *et al*., 2016). Como segundo trabalho acadêmico foi realizado um extensão do primeiro trabalho acrescentando a implementação do Componente de Agregação de Dados e melhorando os resultados com um novo estudo de caso. Este estudo de caso compara o desempenho do coletor de dados do *middleware* Kaa e o componente de coleção de dados do GDDIoT, este trabalho foi publicado na revista Sensors (Huacarpuma *et al*., 2017).

No componente de interface de comunicação foi definido a utilização do protocolo de comunicação *websocket*. Este protocolo de comunicação que permite a comunicação bidirecional por canais *full-duplex* sobre um único *socket*. Esta tecnologia permite a utilização via *browsers*, servidores webs e qualquer aplicação cliente-servidor. Neste contexto, para a implementação do componente de interface de comunicação foi definido o *framework* Tornado. Este *framework* tem características que ajudam a lidar com alguns requisitos para este componente tais como a conexão contínua e transmissão contínua de dados. O *framework* Tornado suporta o protocolo *websocket*, possui bibliotecas para a comunicação assíncrona e capacidade de escalar dezenas de milhares de conexões abertas. Estas características são ideais para manter a conexão de diferentes *middlewares* simultaneamente de forma contínua, transmissão ininterrupta de dados de forma assíncrona.

Para a implementação do componente de Coleção de Dados, foi escolhido três plataformas: Kafka, Storm, Cassandra e Streamparse. O Apache Kafka foi escolhido pela sua capacidade para consumir grandes quantidades de mensagens, arquitetura distribuída e

processamento paralelo que é requisito indispensável deste componente. Esta ferramenta usa o modelo *publish-subscriber* que visa ser um modelo adequado para o processamento de dados IoT. Além disso, o Kafka permite realizar alguma modificações internas para suportar aos requisitos do Componente de Coleção de Dados tais como a capacidade de consumir grandes quantidades de dados em tempo real. Neste contexto, foi realizado a modificação nos processos consumidores para alcançar taxas maiores de captura de dados aumentando a paralelização dos processos.

O Apache Storm dá suporte para a implementação dos módulos de filtragem e extração de metadados. Esta ferramenta tem propriedades que suportam ao processamento distribuído e paralelo dos dados com baixa latência. Além disso o Apache Storm possui uma arquitetura escalável tolerante a falhas, o qual é compatível aos requisitos para o processamento dos dados do GDDIoT. Como sistema de armazenamento de dados foi escolhido o banco de dados NoSQL Cassandra. O modelo orientado a coluna consegue atender aos requisitos dos dados IoT devido a flexibilidade decorrente deste modelo. A arquitetura distribuída, tolerante a falhas e de alta disponibilidade atende aos requisitos para o gerenciamento de dados IoT. Além disso, o Cassandra detém um módulo específico para o armazenamento de séries temporais fazendo uso do modelo de dados baseado em família de colunas. Mas este módulo pode ser configurado para lidar com a grande quantidade dos dados provenientes dos ambiente IoT.

Finalmente, a plataforma Streamparse integra diferentes tecnologías como o Apache Kafka, Storm e Cassandra. Além disso, através do Streamparse pode administrar a topologia Storm definida no GDDIoT, onde pode ser configurado o nível de paralelismo do processamento para todos os módulos constituintes do GDDIoT.

Para a implementação do componente de Agregação de Dados foi desenvolvido três processos que são executados periodicamente. Estes processos fazem uso de *Threads* como técnica de processamento. Foi definido uma *Thread* por cada processo de agregação feito no módulo de sumarização de dados: por minuto, por hora, por dia. A implementação destes processos tem como fonte de dados a tabela de eventos e após o processamento os resultados são armazenados nas tabelas sumarizadas. Na implementação de cada processo verificou-se que os mesmos devem ser executados simultaneamente pelo qual a técnica de utilização de *Threads* visou ser adequada. De forma similar o módulo de extração de

contexto faz uso de *Threads* para a extração de contexto nos mesmos períodos de tempos, por minuto, hora e dia.

Finalmente, para a implementação do Componente de Visualização de Dados foi usada uma ferramenta especializada para apresentação gráfica de séries temporais devido às propriedades temporais dos dados IoT. Neste contexto, o Grafana é uma ferramenta de visualização de métricas e séries temporais capaz de comportar os dados IoT. A utilização do Grafana gerou alguns desafios, tais como a conexão com diferentes fontes de dados e a combinação dos dados dessas fontes. O Grafana geralmente contempla unicamente bancos de dados especializados em séries temporais tais como InfluxDB, Graphite, OpenTSDB, entre outros. Neste contexto o Grafana não suporta o banco de dados Cassandra o qual trouxe um problema na implementação. Para superar este problema foi usado o *plugin* KairosDB que usa como plataforma base o banco de dados Cassandra.

6 – ESTUDOS DE CASO

Este capítulo apresenta o comportamento do GDDIoT para o gerenciamento de dados em diferentes ambientes IoT. Foram simulados três estudos de caso: Sistema de Casas Inteligentes, Sistema de Transporte Inteligente e comportamento do coletor de dados do *middleware* Kaa, fazendo uso dos *middlewares* UIoT (Ferreira, 2014) para o primeiro estudo de caso; os *middlewares* UIoT e SmartCampus (Cecchinel et al., 2014) para o segundo estudo de caso, e o *middleware* Kaa para o terceiro estudo de caso.

Cada uma das simulações foi projetada para atender objetivos específicos. A simulação de casas inteligentes foi desenvolvida para testar a funcionalidade dos componentes de coleção de dados e agregação, quando de grande quantidade de dados provenientes de ambientes IoT. Em tal processo se tem a simulação da criação de mensagem advinda de milhares de objetos que fazem parte de uma rede de casas inteligentes gerenciado pelo *middleware* UIoT. O segundo processo de simulação foi desenvolvido com o objetivo de testar o desempenho dos componentes, a interface de comunicação de dados e a visualização de dados. Neste é simulada a criação de mensagens advindas de milhares de objetos que fazem parte de uma rede de transporte gerenciada pelo *middleware* SmartCampus. O terceiro processo de simulação pretende avaliar o coletor de dados do *middleware* Kaa visando comparar o desempenho deste com o componente coleção de dados do GDDIoT.

Cada uma das simulações foi realizada em diferentes ambientes computacionais. A primeira foi testada em um ambiente computacional de quatro máquinas físicas, enquanto o segundo e terceiro processos foram testados sobre ambientes computacionais composto por 10 e quatro máquinas virtuais, respectivamente. Os dois ambientes foram configurados para ser o mais parecido possível tanto em *software* como em *hardware*.

6.1 – ESTUDO DE CASO: SISTEMA DE CASAS INTELIGENTES

O conceito de casa inteligente integra várias tecnologias e serviços através de redes domésticas para uma melhor qualidade de vida. Uma casa inteligente faz uso de diferentes tecnologias para equipar peças domésticas para o monitoramento e controle remoto mais inteligente, permitindo, assim, a interação entre os objetos domésticos, de modo que a casa

e suas instalações sejam automatizadas sem intervenção do usuário ou com o fácil, conveniente, eficiente, seguro e menos oneroso controle remoto do usuário (Kadam, Mahmauni et al., 2015).

6.1.1 – Descrição

O processo de simulação contempla a criação dos dados de um sistema de casas inteligentes de grande porte que seja composto por milhares de casas. O sistema em questão foi escolhido devido ao maior número de variáveis. Por exemplo: a grande variedade e quantidade de objetos que podem compor um sistema de casas inteligentes, tais como: SmartTV, ar-condicionado, relógios inteligentes, portão da garagem, entre outros aparelhos. O processo de simulação contempla sete dispositivos em cada casa, devidamente listados na Tabela 6.1. O processo de simulação representa milhares de casas que contém sete dispositivos em cada uma delas. Grande parte dos dispositivos é de uso cotidiano dentro de uma casa. A escolha dos dispositivos foi arbitrária.

A Tabela 6.1 apresenta os sete dispositivos que foram simulados. Cada dispositivo está associado a uma ou mais variáveis de estados. Além disso, é possível observar a gama de valores existente em cada variável de estado. Por exemplo, a variável de estado canal somente pode ter valores contemplados entre 1 até 1000.

Tabela 6.1 – Dispositivos do sistema de casas inteligentes (Cerqueira Ferreira, 2014).

Nome do 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
	Distancia	2 até 700
Despertador	Power	Ligado, desligado
Estéreo	Power	Ligado, desligado
	Volume	1 até 100
	Estação	1 até 6

Para a simulação de trocas de mensagens entre o *middleware* UIoT e o GDDIoT, foram implementados vários produtores Kafka. Cada produtor Kafka cria oito milhões de

mensagens simultaneamente. As mensagens criadas pelo processo de simulação têm 160 *bytes* de tamanho. A estrutura das mensagens UIoT é composta pelos seguintes atributos: id_dispositivo, *timestamp*, latitude, longitude, id_sv e valor, conforme evidenciado na Figura 6.1.

Figura 6.1. Estrutura das mensagens do *middleware* UIoT.

A Tabela 6.2, apresenta o número de casas, a quantidade de dados que são criados e o volume de dados que a simulação representa para diferentes intervalos de tempo (dia, mês e ano). O cálculo do volume e a quantidade de dados têm por base as medições feitas pelos objetos a cada 10 segundos.

O número de produtores (Nro. Pr.), na Tabela 6.2, representar a fonte de dados do *middleware* UIoT. O número de produtores pode ser incrementado para aumentar o número de mensagens criadas, aumentando o número de casas, buscando, assim, avaliar a *performance* do componente coleção de dados do GDDIoT. No caso de um produtor, ele representa 33.600 casas, cada casa com 7 dispositivos nelas, gerando em apenas um dia 2.032 milhões de registros, de um total de 290 *gigabytes* (0,29 TB); em um mês são 60.963 milhões registros, com 8,87 *terabytes*; e, em um ano são 741.680 milhões de registros, com 107.92 *terabytes*. No caso de dois, três e quatro produtores tem-se a mesma conta, evidenciando que toda vez que é incrementado o número de produtores, incrementa-se a quantidade de casas monitoradas. Assim, a quantidade de registros e o tamanho dos dados aumentam proporcionalmente.

Tabela 6.2 – Número de casas, quantidade de mensagens e volume de dados da simulação do *middleware* UIoT. (Mi - Milhões de registros, TB - *Terabytes*).

Nro. Pr.		Dia		Mês		Ano	
	Nro. Cas	Quant.	Tam.	Quant.	Tam.	Quant.	Tam.
1	33.600	2.032 Mi	0,29 TB	60.963 Mi	8,87 TB	741.680 Mi	107.92 TB
2	63.492	3.840 Mi	0,56 TB	115.000 Mi	16,76 TB	1401.600 Mi	203,96 TB
3	95.238	5.760 Mi	0,84 TB	172.800 Mi	25,15 TB	2102.400 Mi	305,92 TB
4	142.857	8.640 Mi	1,26 TB	259.200 Mi	37,72 TB	3153.600 Mi	458,91 TB

6.1.2 – Ambiente Computacional

Para o ambiente computacional do presente estudo de caso foi definido um *cluster* com quatro máquinas físicas, cada uma com as seguintes especificações: processador Intel Xeon 2.8 GHz com oito núcleos, 8GB de RAM, 500GB HD e 1Gb Ethernet. A Figura 6.2, apresenta o ambiente computacional onde são definidos três *clusters* virtuais: Kafka, Storm, e Cassandra, implementados no *cluster* físico.

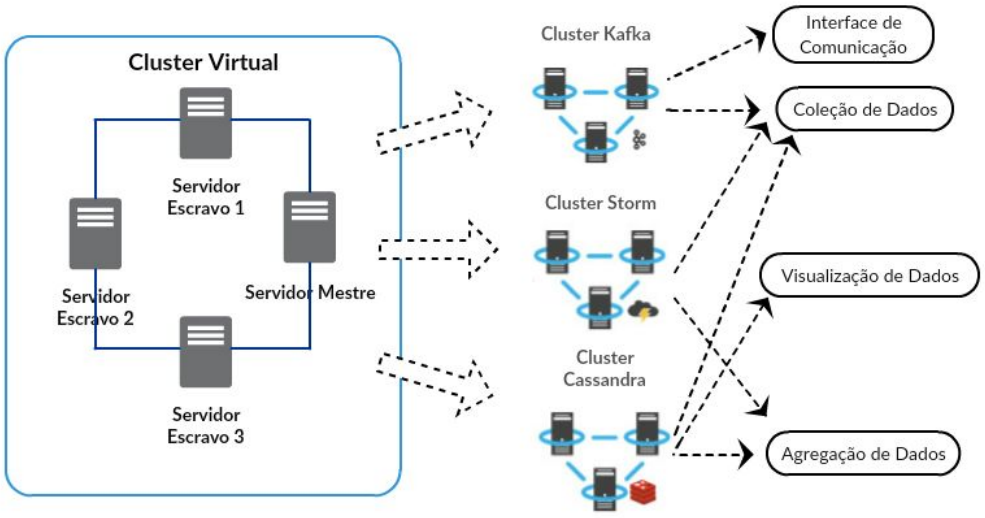

Figure 6.2. Ambiente computacional para o Sistema de Casas Inteligentes

No presente estudo de caso, o *cluster* Kafka foi configurado com quatro máquinas, fazendo com que o fator de replicação máximo seja 4. O fator de replicação é proporcional ao número de nós do *cluster* Kafka. Para garantir a escalabilidade horizontal do *cluster*, foi

necessário adicionar mais partições e, posteriormente, distribuí-las em máquinas adicionais. O *cluster* Storm foi implementado com quatro máquinas, sendo que uma máquina foi configurada como um nó mestre (Nimbus); e, os três nós restantes são nós escravos (supervisores). Finalmente, o *cluster* Cassandra foi configurado com quatro máquinas. É importante notar que os *clusters* supramencionados compartilhavam a mesma infraestrutura física aqui utilizada.

6.2 – ESTUDO DE CASO: SISTEMA DE TRANSPORTE INTELIGENTES

O estudo de caso Sistema de Transporte Inteligente compreende do processo de simulação de criação de dados oriundos de um sistema de transporte inteligente de grande porte que esteja composto por milhares de objetos. Este foi escolhido devido a sua importância em uma civilização moderna. Neste contexto, o objetivo deste estudo de caso é a simulação de um sistema de transporte para avaliar o desempenho dos componentes de interface de comunicação de dados e de visualização de dados.

Através desta simulação pretende-se mostrar a capacidade de integrar diferentes *middlewares* IoT com o GDDIoT, uma vez que a interface de comunicação de dados suporta várias conexões simultaneamente e recebe diferentes tipos de mensagens provenientes dos *middlewares* IoT. Além disso, mostrou-se o desempenho do componente de visualização de dados através de diferentes painéis implementados pelo componente em questão.

6.2.1 – Descrição

A simulação do sistema de transporte cria dados de milhares de semáforos gerenciados pelos *middlewares* UIoT e SmartCampus. Diferentemente do estudo de caso Casas Inteligentes, o estudo de caso do transporte implementa vários processos que estabelecem a conexão de entre os *middlewares* UIoT e SmartCampus, simultaneamente. Estes criam milhões de mensagens simultâneas de diversos objetos que podem fazer parte de um sistema de transporte inteligente. Tal processo de simulação contempla três objetos, devidamente listados na Tabela 6.3, a seguir. Todos os objetos são de uso cotidiano dentro de um sistema de transporte. Cada objeto está associado a uma ou mais variáveis de estados. Além disso, é possível observar a gama de valores que cada variável de estado

pode possuir. Por exemplo, a variável de estado luz somente pode ter valores de verde, amarelo ou vermelho.

Tabela 6.3 – Dispositivos do sistema de transporte inteligente.

Nome do Dispositivo	**Variáveis de estado (VS)**	**Valores permitidos das VS**
Carro	Power	Ligado, desligado
	Latitude	+90.00 até -90.00
	Longitude	+90.00 até -90.00
Semáforo	Luz	Verde, amarelo, vermelho
	Power	Ligado, desligado
Camera	Power	Ligado, desligado
	Velocidade	0 até 250.00
	Latitude	+90.00 até -90.00
	Longitude	+90.00 até -90.00

No que tange à simulação dos semáforos, tais objetos possuem como variável de estado a luz que indica os possíveis valores – que representam o estado desta variável de estado, podendo ser verde, amarelo ou vermelho. A luz verde e vermelha podem ficar ligadas até cinquenta segundos, e a luz amarela pode ficar ligada por dois segundos. Os valores de tempo de operação de cada estado foram escolhidos arbitrariamente, podendo variar conforme os ambientes IoT.

As mensagens dos *middlewares* UIoT e SmartCampus possuem características especiais com atributos específicos, de acordo com a mensagem do próprio *middleware* IoT. A estrutura das mensagens UIoT é composta pelos seguintes atributos: id_dispositivo, *timestamp*, latitude, longitude, id_sv e valor (vide Figura 6.3A). A estrutura de mensagens do *middleware* SmartCampus é composta de 3 atributos: n (identificador do sensor - *a44e317a83d0*), v (valor de medição - *20.3* ºC) e t (*timestamp* associado - *2017-07-27 14:00:00*) (veja a Figura 6.3B).

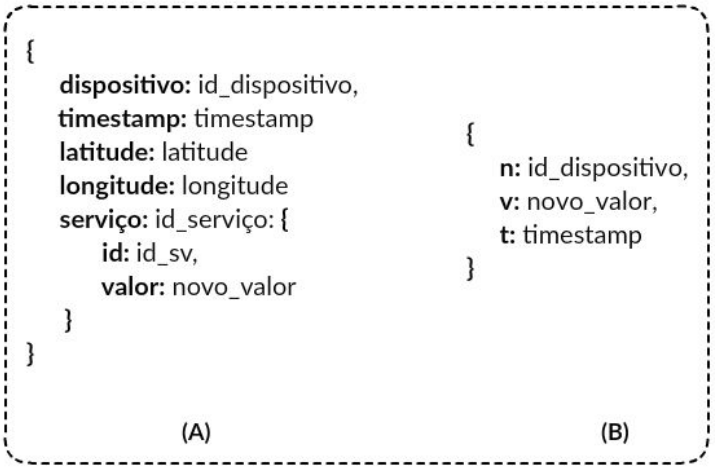

Figura 6.3. (A) Estrutura das mensagens UIoT e (B) estrutura das mensagens SmartCampus.

A Figura 6.3 aponta que as mensagens dos *middlewares* UIoT e SmartCampus possuem estruturas diferentes em relação aos números de atributos e os nomes de atributos, mas têm conteúdos semelhantes. Neste contexto, as mensagens UIoT e SmartCampus têm identificadores dos atributos dos objetos, valor de medição e *timestamp* com diferentes nomes de atributos e em ordem diferente, mas detentores da mesma informação. Assim, foi necessário desenvolver um esquema de mensagem genérico definido no componente da interface de comunicação, logrando suportar as diferentes estruturas das mensagens dos *middlewares* IoT.

Para a simulação das mensagens de intercâmbio entre os *middlewares* UIoT e SmartCampus com o GDDIoT, dois processos – um para cada *middleware* – foram definidos. Estes geram um milhão de mensagens cada um simultaneamente. Neste contexto, no caso das mensagens do *middleware* UIoT, estas são de 160 *bytes* de tamanho, enquanto que as mensagens do *middleware* SmartCampus possuem 40 *bytes* de tamanho. Tanto o *middleware* UIoT como SmarCampus fazem uso do formato JSON em suas mensagens. Neste sentido, os processos geram mensagens JSON com a estrutura definida para cada *middleware* específico. Quando da criação das mensagens JSON, estas são convertidas para uma estrutura de mensagem padrão pelo módulo do lado do cliente do componente de interface de comunicação do GDDIoT. A mensagem padrão é uma mensagem JSON que, posteriormente, é encaminhada para o GDDIoT fazendo uso do protocolo de comunicação *webSocket*, que faz uso da interface de comunicação do GDDIoT.

6.2.2 – Ambiente Computacional

O ambiente computacional para o presente estudo de caso é composto por um *cluster* com 10 máquinas virtuais, cada uma com as seguintes especificações: processador Intel Xeon 2.8 GHz com dois núcleos, 8GB de RAM, 50GB HD e 1Gb Ethernet. A Figura 6.4, apresenta o ambiente computacional, onde são definidos os três *clusters* virtuais (Kafka, Storm e Cassandra) implementados sobre máquinas virtuais.

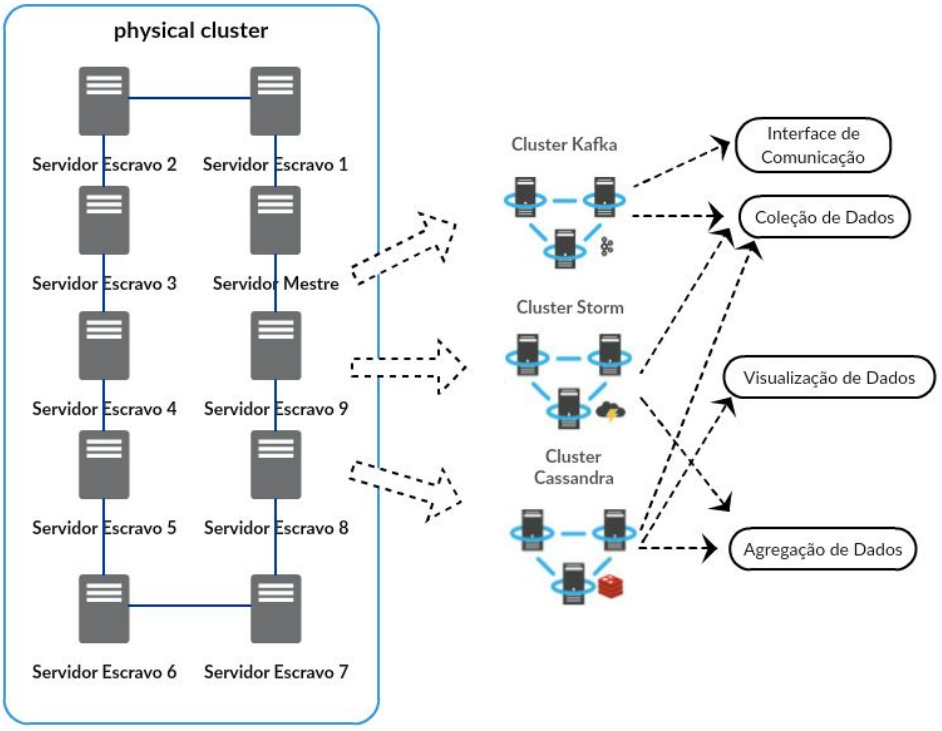

Figure 6.4. Ambiente computacional para o estudo de caso.

No estudo de caso em questão, o *cluster* Kafka foi configurado com dez máquinas virtuais, fazendo com que o fator de replicação máximo seja 10. O fator de replicação é proporcional ao número de nós do *cluster* Kafka. Para garantir a escalabilidade horizontal do *cluster*, foi necessário adicionar mais partições e, posteriormente, distribuí-las em máquinas adicionais. O *cluster* Storm foi implementado com 10 máquinas, sendo uma máquina configurada como um nó mestre (Nimbus) e os nove nós restantes como nós escravos (supervisores). Finalmente, o *cluster* Cassandra foi configurado com 10 máquinas virtuais. É importante notar que os *clusters* supramencionados compartilhavam a mesma infraestrutura aqui utilizada.

6.3 – ESTUDO DE CASO: COLETOR DE DADOS DO *MIDDLEWARE* Kaa

O processo de simulação contempla a geração de dados criado por um sistema de casa inteligente de grande porte que esteja composto por milhares de objetos. Esta simulação é similar ao primeiro estudo de caso (Seção 6.1). Porém, o objetivo do presente estudo de caso foi avaliar especificamente o coletor de dados do *middleware* Kaa, comparando o seu desempenho com o componente coleção de dados do GDDIoT. A simulação aqui evidenciada pretendeu mostrar a capacidade do coletor de dados do *middleware* Kaa em um cenário o mais parecido possível àquele utilizado pelo GDDIoT.

6.3.1 – Descrição

Semelhantemente ao processo de simulação da Seção 6.1 – o processo de criação de dados de um Sistema de Casas Inteligentes de grande porte que esteja composto por milhares de casas –, o processo contempla os sete dispositivos listados na Tabela 6.1. Neste cenário, o *middleware* Kaa contempla milhares *endpoints* enviando milhões de mensagens (*logs*) simultaneamente a uma taxa constante. Devido à natureza síncrona do *middleware* Kaa, a geração de mensagens se dá apenas de modo síncrono. A estrutura dos *logs* dos *endpoints* inclui dados relativos ao identificador da fonte, à descrição do serviço, às *timestamps* de transmissão e ao valor da medição. No presente estudo de caso, a estrutura das mensagens do *middleware* Kaa foi definida pelos seguintes atributos: id_dispositivo, valor e *timestamp*, conforme evidenciado na Figura 6.5. O formato das mensagens pode ser personalizado e pré-configurado em um esquema, e o tamanho das mensagens é de 50 *bytes*. Quando o servidor do *middleware* Kaa recebe uma mensagem, ele armazena as mensagens no banco de dados MongoDB e, posteriormente, encaminha uma resposta (*ack*) no *endpoint*.

Figura 6.5. Estrutura das mensagens do *middleware* Kaa.

Similar ao primeiro estudo de caso, o processo de criação das mensagens se dá via processos, que geram mensagens. O *middleware* Kaa provê protótipos para a simulação dos dados, segundo a estrutura pré-configurada no *middleware* Kaa. Com base no protótipo, foram realizadas modificações para que o processo de criação possa atingir oito milhões de mensagem por processo (equivalente ao produtor Kafka). A Tabela 6.4, a seguir, apresenta o número de casas e o volume de dados para diferentes períodos de tempo. O cálculo se dá com base nas medições dos dispositivos a cada 10 segundos, com 50 *bytes* de tamanho de cada mensagem.

Tabela 6.4 – Número de casas e volume de dados da simulação gerenciadas pelo *middleware* Kaa.

Nro. Produtores	**Nro. Casas**	**Dia**	**Mês**	**Ano**
1	33.600	0,09 TB	2,86 TB	33,73,TB
2	63.492	0,17 TB	5,41 TB	63,74 TB
3	95.238	0,26 TB	8,12 TB	95,61 TB
4	142.857	1,39 TB	12,18 TB	143,41 TB

6.3.2 – Ambiente Computacional

Para realizar os testes do presente estudo de caso, foi implementado um ambiente computacional composto por quatro máquinas virtuais, cada uma com as seguintes especificações: processador Intel Xeon de 2,5 GHz com quatro núcleos, 8 GB de RAM, 50 GB de HD e 1 Gb Ethernet. O sistema operacional Ubuntu Linux 14.04 foi configurado em cada nó com o *middleware* Kaa v0.10.0. Além disso, o *middleware* Kaa está suportado por um *cluster* composto pelos nós Kaa. Internamente, o *cluster* Kaa é suportado pelo banco de dados distribuído MongoDB como banco de dados NoSQL.

A Figura 6.4, evidencia o ambiente computacional, onde são definidos dois *clusters* virtuais: Kaa e MongoDB.

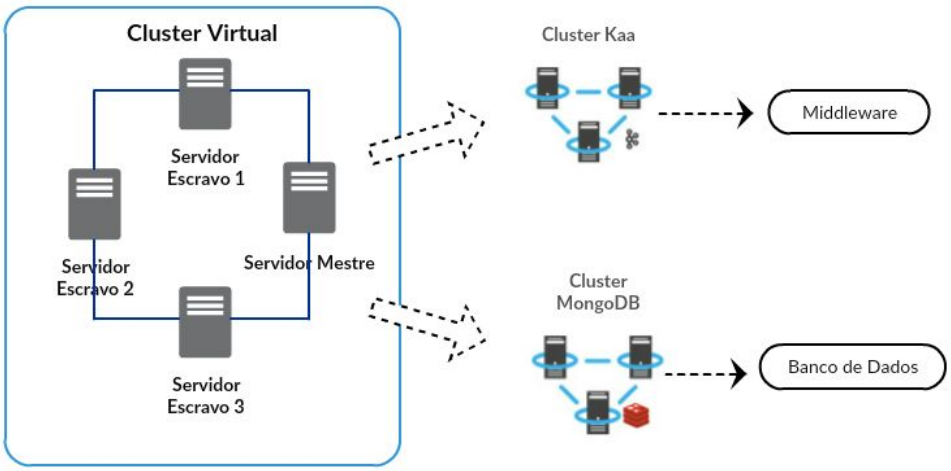

Figure 6.4. Ambiente computational Kaa.

No presente estudo de caso, o *cluster* Kaa foi configurado com quatro máquinas virtuais. O *cluster* do banco de dados MongoDB foi implementado com três máquinas virtuais, sendo uma máquina configurada como nó mestre (*config server*) e os dois nós restantes como nós escravos (*App server*). É importante notar que os *clusters* supramencionados compartilhavam a mesma infraestrutura aqui utilizada.

6.4 – CONSIDERAÇÕES

Os estudos de casos definidos nesta tese tiveram objetivos específicos já definidos e implementados em ambientes computacionais diferentes. No primeiro estudo de caso, o ambiente computacional faz uso de 4 máquinas físicas. Devido às limitações dos recursos, no segundo e terceiro estudo de caso foi usada máquinas virtuais. Este ambiente virtual foi configurado o mais parecido possível com o primeiro ambiente computacional para dar continuidade aos testes decorrentes do segundo e terceiro estudo de caso.

O primeiro estudo de caso - Sistema de Casa Inteligentes foi escolhido devido a variedade de objetos e em consequência a variedade de formato de dados e protocolos de comunicação usados por eles. Neste contexto, através do gerenciamento dos dados desse estudo de caso demonstra a funcionalidade do Componente de Coleção de Dados. Nos três estudos de caso foram definidos a criação de 8 milhões de mensagens tanto de forma síncrona como assíncrona. Este número de mensagens foi definida de forma arbitrária para ser o mais alto possível, tais mensagens são criadas por centenas de *Threads* sendo executadas simultaneamente.

No segundo estudo de caso, o interesse não foi focado no volume de dados, mas na interoperabilidade do GDDIoT com diferentes *middlewares* IoT. Desta forma, no segundo estudo de caso o foco principal foi como lidar com os diferentes formatos e protocolos usados pelos *middlewares* para estabelecer a comunicação e transmissão dos dados. Neste contexto, foram encontrados diferentes problemas tais como o padrão da mensagem usada pelos *middlewares*, as nomenclaturas usadas nessas mensagens e o número de atributos. Todos esses problemas foram resolvidos através da padronização das mensagens através do uso de metadados.

No terceiro estudo de caso, o interesse foi demonstrar o melhor desempenho do GDDIoT comparado como o *middleware* Kaa referente a coleção de dados. No que tange à implementação deste estudo de caso e os outros estudos de casos foram implementadas usando a linguagem programação Python, enquanto que a simulação do coletor Kaa foi implementada usando a linguagem de programação Java.

7 – RESULTADOS E DISCUSSÃO

No presente capítulo têm-se os resultados alcançados pelos três estudos de caso apresentados no capítulo anterior – Capítulo 6, a saber:

- Casa Inteligente: análise do desempenho do componente de coleção de dados do GDDIoT;
- Sistema de Transporte: análise dos resultados obtidos pela interface de comunicação e o componente de visualização do GDDIoT; e
- Coletor de dados do *middleware* Kaa: análise dos resultados obtidos pelo coletor de dados do *middleware* Kaa, para logo ser comparado com os resultados obtidos pelo GDDIoT.

7.1 – RESULTADOS

O primeiro estudo de caso – simulação do sistema de casas inteligentes – tem como objetivo avaliar a capacidade do GDDIoT no processamento de grandes quantidades de dados provenientes de *middlewares* IoT. Aqui, os componentes de coleção de dados e agregação de dados foram testados com grande quantidade de mensagens provenientes do *middleware* UIoT. Logo, é avaliada a capacidade da entrada de dados do componente de coleção de dados através do módulo de captura de dados. No que tange a avaliação do componente de agregação de dados, compara-se o desempenho da consulta de dados com base em dados sumarizados e dado sem qualquer agregação.

No segundo estudo de caso – simulação de transporte inteligente – tem-se a avaliação da propriedade de interoperabilidade do GDDIoT com diferentes *middlewares* IoT – que se dá através da interface de comunicação proveniente de diferentes *middlewares* IoT com diferentes formatos de mensagens. Aqui, o objetivo é verificar o comportamento do componente de visualização de dados para apresentar dados de modo gráfico.

O terceiro estudo de caso – coletor de dados do *middleware* Kaa – analisou o desempenho do coletor de dados Kaa quando do recebimento de milhões de mensagens advindas de uma rede composta de milhares de objetos. Para garantir a equanimidade, o ambiente de implementação do GDDIoT e Kaa foi definido de forma semelhante, em

termos de quantidade de nós dos *clusters*, quantidade de memória dos nós, capacidade de processamento e sistema operacional. No entanto, em relação ao ambiente computacional de teste, o ambiente do GDDIoT foi implementado por máquinas físicas, enquanto o ambiente Kaa foi implementado por máquinas virtuais.

7.1.1 – Resultados: Sistema de Casas Inteligentes

O cenário da simulação representa milhares de casas que possuem sete dispositivos cada uma das casas. Tal processo foi implementado através de produtores Kafka. No processo de simulação, aqueles produtores criaram mensagens síncrona e assíncrona. O processo foi testado de forma incremental, a partir de um único produtor, até o alcance de quatro produtores. A Figura 7.1, evidencia os resultados obtidos dos dados gerados de forma assíncrona pelos produtores. O desempenho do GDDIoT foi proporcional ao número de produtores que estão sendo executados. Além disso, aprecia-se a quantidade de casas que podem ser suportadas no cenário da geração de mensagens síncronas. Por exemplo, no caso de dois produtores, representam 2.934 casas que criam 20.539 mensagens por segundo. Quando o número de produtores aumentou para quatro, enviando 30.798 mensagens, o número de casas atendidas foi de 142.857. Assim, é possível perceber que quando do aumento do número de produtores, o número de mensagens/casas que o componente de coleção de dados pode suportar aumentou proporcionalmente. Logo, o serviço de coleta de dados GDDIoT manteve-se com tendência escalável mesmo com o aumento de mensagens enviadas.

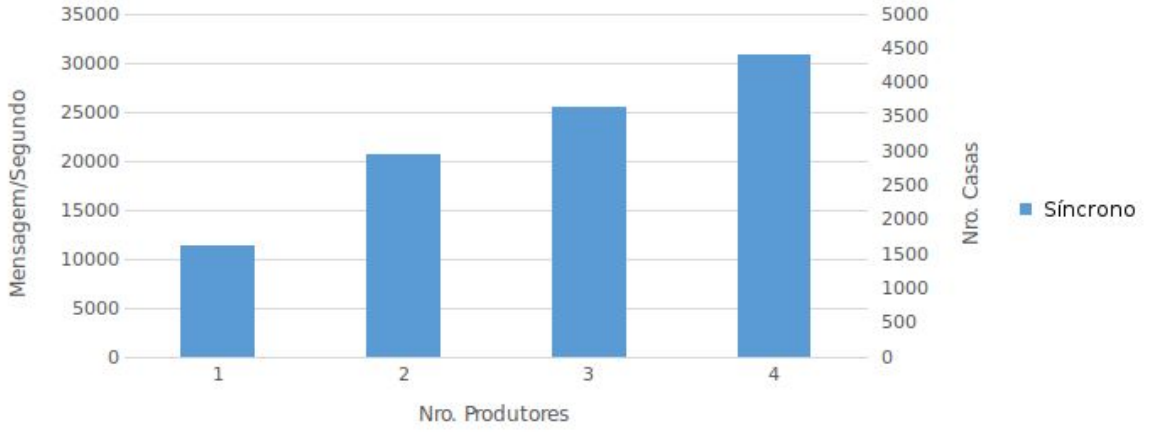

Figura 7.1. Produtor de dados síncrono do *middleware* UIoT.

A Figura 7.2, apresenta o resultado do comportamento do componente coleção de dados quando os dados são gerados por produtores assíncronos em comparação com os dados gerados pelos produtores síncronos. Aqui, o número de mensagens geradas pelos produtores assíncronos e coletadas pelo GDDIoT é maior em relação ao número de mensagens gerados pelos produtores síncronos. Por exemplo: para dois produtores, a quantidade de mensagens coletadas por segundo – no caso dos produtores síncronos – é de 20.539 (2.934 casas), enquanto que para os produtores assíncronos é de 444.444 (63.492 casas). Uma vez que a captura de dados gerada pelos produtores assíncronos é mais rápida, cada mensagem enviada não necessita de confirmação de que foi recebida, permitindo ao sistema enviar mensagens sem ter que esperar por uma resposta para enviar outra mensagem.

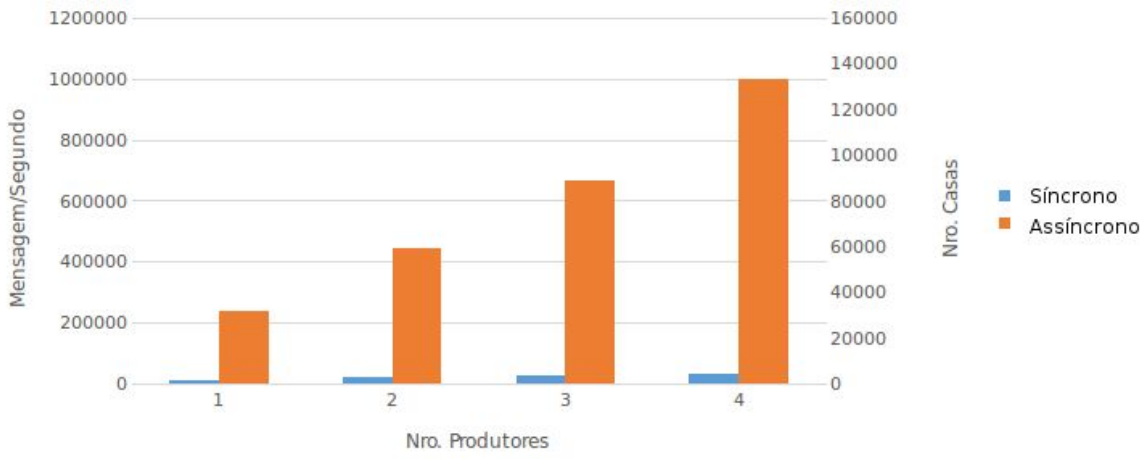

Figura 7.2. Produtor de dados síncrono e assíncrono do *middleware* UIoT.

Além dos testes de volume de dados, o presente estudo analisou o componente agregação de dados – relacionado ao tempo gasto na consulta dos dados. Neste módulo, a principal tabela é a de evento – que armazena os dados brutos enviados pelo *middleware* IoT. Após a execução do módulo de sumarização de dados, os dados provenientes da tabela evento são sumarizados e armazenados nas tabelas sumarizadas. Para evidenciar o melhor desempenho em relação ao tempo de resposta das consultas, com base na abordagem de agregação de dados, foram especificadas três consultas, a saber: busca por minuto, busca por segundo e busca por hora. Na Tabela 7.1, a seguir, tem-se os resultados referentes ao desempenho quando da consulta dos dados das tabelas sumarizadas e a tabela

evento sem nenhum pré-processamento. Faz-se importante salientar que a tabela evento armazena as mensagens coletadas pelo componente de coleção de dados. É uma tabela que tende a crescer muito em quantidade de dados e volume. Ao contrário da tabela evento, as tabelas sumarizadas tendem a ter menos quantidade de dados e volume. Neste contexto, tem-se a comparação do tempo de resposta das consultas feitas sobre a tabela evento e as tabelas sumarizadas. E ainda, no que tange às consultas realizadas sobre a tabela evento e sumarizadas, tem-se o retorno do mesmo resultado. Neste contexto, as consultas realizadas sobre as tabelas sumarizadas tiveram melhores resultados quando comparados com o tempo de resposta da tabela evento – fato evidente para períodos de tempo maiores. Por exemplo: a consulta realizada por dia sobre a tabela evento é bem maior em comparação com o tempo de resposta da consulta na tabela sumarizada por dia.

Tabela 7.1 – Tempo de consulta sobre a tabela evento e sumarizadas.

	Tempo de Consulta	
Período	**Tabela de Evento**	**Tabelas Sumarizadas**
Por minuto	1,51 seg	4 msec
Por hora	2,12 seg	3 msec
Por dia	43,94 seg	3 msec

7.1.2 – Resultados: Sistema de Transporte Inteligentes

O cenário de simulação envolve o envio de dados de diferentes *middlewares* com diferentes tipos de estrutura de mensagens. A interface de comunicação revela-se flexível para o suporte de diferentes tipos de mensagens através do módulo da interface de comunicação. A interface de comunicação é composta por duas partes, a saber: a do cliente (*middleware* IoT) e a do servidor (GDDIoT). Do lado do cliente, através de uma *Application Programming Interface* (API) – que auxilia no estabelecimento da conexão entre os *middlewares* IoT e o GDDIoT. Além disso, esta também ajuda a transformar diferentes tipos de mensagens em uma estrutura padrão para ser consumida pelo GDDIoT. Neste contexto, para mostrar a capacidade de lidar com a heterogeneidade das mensagens dos *middlewares* IoT, o GDDIoT consegue lidar com os dados provenientes dos *middlewares* UIoT e SmartCampus simultaneamente. Mesmo quando os atributos estiverem faltando nas mensagens de origem, é possível encaminhar os dados parciais para

o GDDIoT. Por exemplo: o *middleware* SmartCampus não possui os atributos de geo-localização (latitude e longitude), mas, o envio de dados ainda é possível. Além disso, o nome dos atributos, em geral, é diferente em grande parte dos *middlewares* IoT. Por exemplo: no *middleware* SmartCampus, o nome dos atributos é completamente diferente dos nomes dos atributos do GDDIoT. Tal problema é superado quando a mensagem do *middleware* IoT é transformada na mensagem padrão do GDDIoT.

No que tange ao componente visualização de dados do GDDIoT, os dados provenientes dos diferentes *middlewares* IoT que foram coletados pelo componente de coleção de dados e que passaram por algum tipo de processamento são visualizados pelo componente em questão. Os dados são exibidos através de diferentes painéis que podem ser customizados de acordo com as necessidades. A Figura 7.3, apresenta um painel com a contagem dos dados recebidos pelos *middlewares* UIoT e SmartCampus. Esta evidência, no eixo Y, o número de mensagens e, no eixo X, o *timestamp*. Aqui, o eixo Y aponta a contagem de mensagens a cada dez minutos dos *middlewares* UIoT e SmartCampus. O tempo de agrupamento pode ser configurado com valores diferentes de acordo com as necessidades dos usuários. Por exemplo: 10 minutos, cinco minutos, dois minutos, 30 segundos etc.

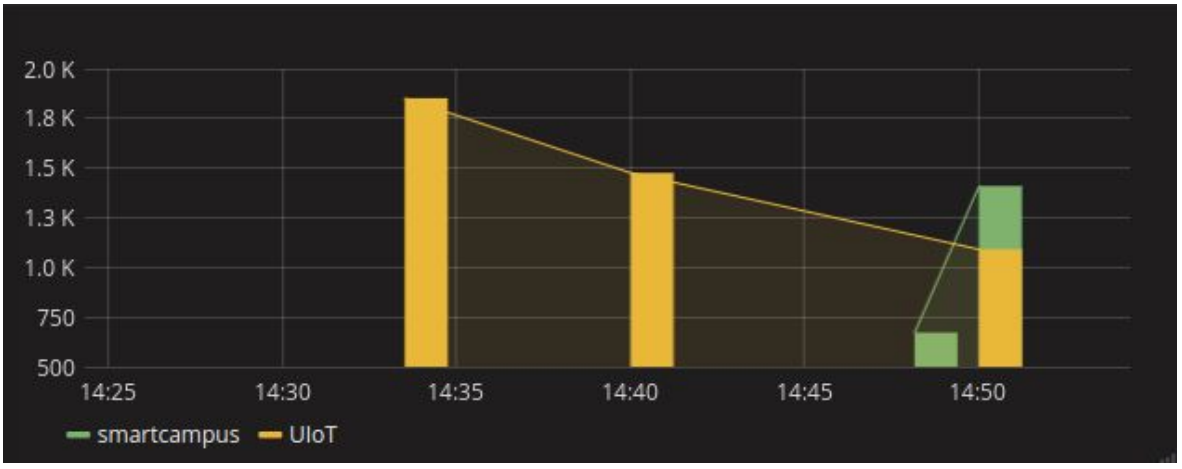

Figura 7.3. Exemplo de contagem de mensagens por *middleware* IoT.

A Figura 7.4, apresenta o exemplo de um painel de controle customizado. Este exibe o comportamento de um dispositivo específico gerenciado pelo *middleware* UIoT. É possível, então, observar o comportamento de um semáforo específico. Semelhante ao

exemplo anterior, o eixo Y aponta o tempo de funcionamento do semáforo para cada luz acendida e, o eixo X, o *timestamp*. Ainda no exemplo em questão, o eixo Y evidencia o comportamento de um semáforo específico a cada dois minutos para UIoT. Análogo ao painel anterior, o tempo de verificação do dispositivo pode ser configurado conforme as necessidades dos usuários. Por exemplo: no presente painel é de 2 minutos, mas podem ser modificados.

Figura 7.4. Funcionamento de um semáforo gerenciado pelo *middleware* UIoT

7.1.3 – Resultados: coletor de dados do *Middleware* Kaa

O *middleware* Kaa utilizado no terceiro estudo de caso da pesquisa em questão somente trabalha com mensagens de forma síncrona. Assim, a comparação com o coletor de dados do GDDIoT se deu por mensagens síncronas. A Figura 7.5, apresenta a relação entre a velocidade de recepção das mensagens por segundo e o número de casas suportadas por diferentes números de nós Kaa. Aqui, o eixo Y evidencia a taxa de mensagens coletadas por segundo e o equivalente em número de casas suportadas, e o eixo X representa o número de produtores de mensagens.

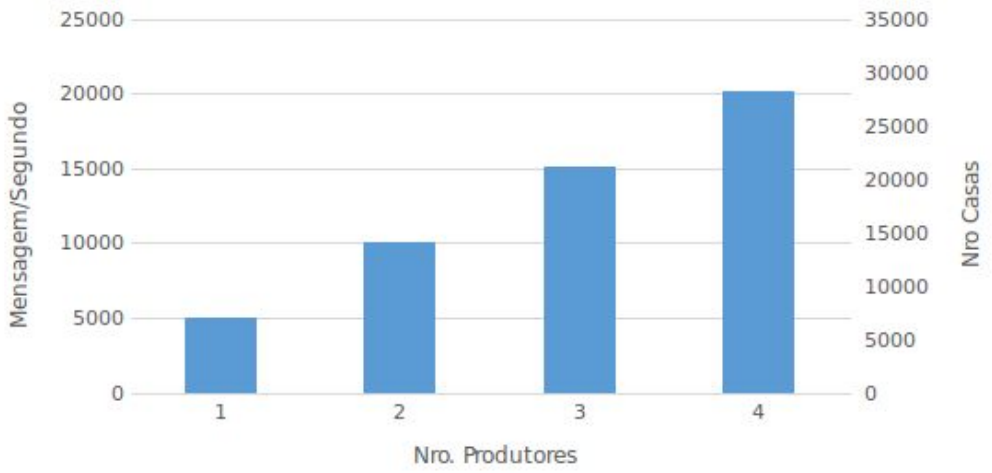

Figura 7.5. Produtor de dados síncrono do *middleware* Kaa.

Nos resultados da simulação em questão para o *middleware* Kaa tem-se a velocidade da coleta de dados (mensagens/segundo) para diferentes números de produtores de mensagens. Tal processo foi testado de forma incremental, de um produtor até quatro produtores. Enquanto o produtor é incrementado, o número de nós do *cluster* Kaa também é incrementado proporcionalmente para suportar o aumento do volume de dados. Neste sentido, de modo escalável, tem-se, na Figura 7.5, os resultados alcançados pelo coletor do *middleware* Kaa. Aqui, uma vez incrementada a quantidade de nós Kaa no *cluster*, é incrementado proporcionalmente seu desempenho. Além disso, é possível observar o número de casas que podem ser suportadas. Por exemplo: para dois produtores, o número de mensagens consumidas por segundo pelo coletor Kaa é de 10.094, ou seja, 1.442 casas.

Comparando os resultados da coleta de dados entre o GDDIoT e o coletor do *middleware* Kaa, é possível apreciar o melhor desempenho do GDDIoT em relação ao coletor de dados Kaa quando lida com grande quantidade de dados. Por exemplo: na Figura 7.6, a seguir, o GDDIoT consegue coletar 11.283 mensagens por segundo, e o coletor do *middleware* Kaa consegue coletar 5.046 mensagens por segundo. Outro exemplo é quando se compara com três nós: o resultado é mais de 25 e 15 mil mensagens por segundo, respectivamente. É importante ressaltar que a comparação em questão se dá com produtores síncronos tanto para o GDDIoT como para o coletor do *middleware* Kaa.

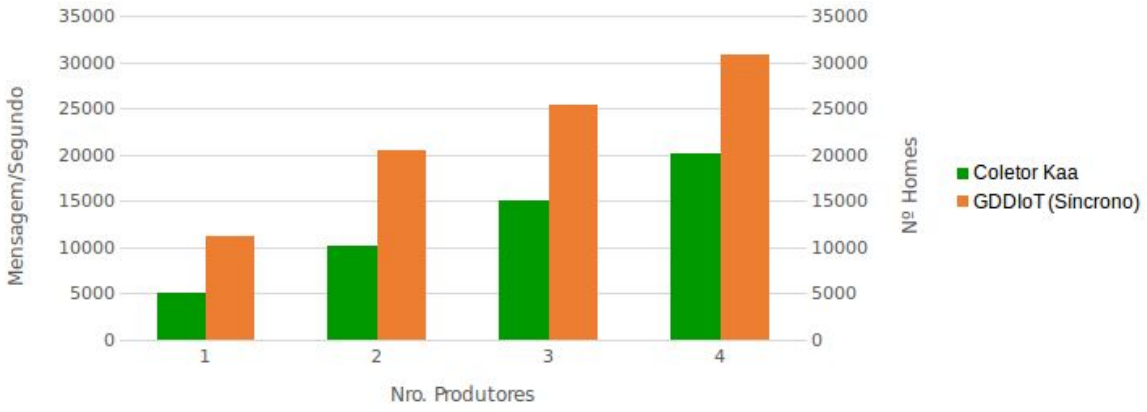

Figura 7.6. Comparativo da quantidade de mensagens coletados de forma síncrona pelo GDDIoT e o coletor *middleware* Kaa.

Outra comparação em relação aos resultados do GDDIoT quando da coleta de dados criados pelos produtores assíncronos e pelo coletor de dados do *middleware* Kaa se dá na Figura 7.7. É preciso salientar que esta não se mostra totalmente justa, uma vez que o melhor desempenho do GDDIoT é devido ao fato de que as operações são feitas de forma assíncrona. Por exemplo: para quatro nós, o GDDIoT pode coletar um milhão de mensagens por segundo, enquanto o coletor do *middleware* Kaa consegue coletar mais de 20 mil mensagens por segundo.

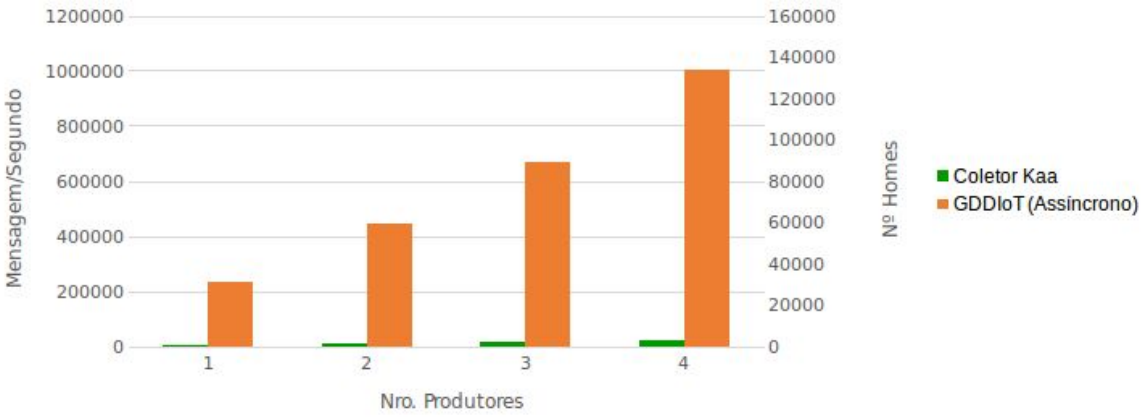

Figura 7.7. Comparativo da quantidade de mensagens coletados pelo GDDIoT e o coletor *middleware* Kaa.

7.2 – DISCUSÃO

A presente seção busca analisar: os resultados de coleção de dados e agregação de dados do estudo UIoT – GDDIoT; o desempenho dos componentes da interface de comunicação e a visualização de dados do GDDIoT; e, o desempenho do coletor de dados do

middleware Kaa em comparação ao desempenho do componente de coleção de dados do GDDIoT em termos de consumo de dados de grande quantidade de dados.

No primeiro estudo de caso – Casas Inteligentes – é possível notar que o desempenho do GDDIoT para a coleta de dados provenientes dos produtores assíncronos apresentou melhores resultados do que quando recebe dados dos produtores síncronos, conforme evidenciado na Figura 7.2. A recepção rápida dos dados pelo GDDIoT se deve pelo fato de que os dados gerados pelos produtores assíncronos não necessitam da confirmação da recepção sucedia das mensagens enviadas. Neste sentido, a recepção dos dados para a simulação assíncrona é mais rápida do que seu equivalente síncrono. Os resultados apontam que o GDDIoT pode suportar a coleta de dados à taxa de um milhão de mensagens por segundo. Por exemplo: quando se tem a coleta de dados gerados por dois produtores, o número de mensagens por segundo é de 660.000, ou seja, 95.238 casas. Assim, a Figura 7.2 evidenciou que o GDDIoT tende a ser escalável em relação à coleção de dados – o que significa que a recepção de dados pelo GDDIoT pode suportar uma grande quantidade de dispositivos que enviam dados ao mesmo tempo, uma vez que é possível adicionar mais nós ao *cluster*.

Os resultados, em termos de coleção de dados, apontam que o GDDIoT possui bom desempenho em parte devido à configuração do Kafka. O Kafka funciona muito bem com vários *brokers* (nós) que fazem parte do *cluster* Kafka. No primeiro estudo de caso aqui apresentado, a Kafka é responsável pela recepção de milhões de mensagens IoT criadas pelo produtores Kafka, atendendo, assim, ao requisito recepção de grande quantidade de dados em tempo real. Enquanto isso, os consumidores KafkaSpout da topologia Storm suportam facilmente o consumo dos dados coletados dos produtores – resultado das múltiplas partições criadas em um tópico Kafka. As partições permitem operações de leitura e escrita em paralelo. Em geral, enquanto algumas partições são lidas pelo consumidor KafkaSpout, outras partições são escritas por produtores Kafka. Além disso, os resultados evidenciam que a topologia definida pelo GDDIoT é executada adequadamente pelo Storm de forma distribuída e paralela, fazendo uso dos recursos de processamento de cada nó do *cluster*, a fim de processar todos os dados em tempo real.

No que tange à filtragem dos dados, através dos estudos de casos aqui evidenciados – Casas Inteligentes e Sistema de Transporte Inteligente –, a filtragem dos dados realizada pelo componente de filtragem de dados do GDDIoT é importante para o desenvolvimento

de diferentes tipos de análises. O módulo em questão pode implementar diversos tipo de filtragem sobre os dados IoT. Neste contexto, o módulo de filtragem implementa a verificação de domínio dos dados coletados, culminando em dados mais confiáveis em termos de medições que estejam dentro dos valores previstos pelos objetos. Além disso, tal processo de filtragem ajuda a não considerar medições erradas, uma vez que estas são descartadas. No contexto do gerenciamento de dados IoT realizado pelos *middlewares* IoT (seção 3), grande parte destes sistemas não contempla nenhum tipo de filtragem sobre os dados gerenciados por eles. Assim, o GDDIoT, através do módulo de filtragem de dados, pretende auxiliar na carência do tratamento dos dados IoT. O GDDIoT não somente contempla a verificação de domínio, mas podem ser implementados outros tipos de filtragem, conforme o tipo de objeto que faz parte de um ambiente IoT.

O GDDIoT contempla o requisito integração de diferentes sistemas, como, por exemplo, *middlewares* IoT, de forma simultânea. Integrar diversos *middlewares* IoT implica lidar com a diversidade de dados (o formato e o tipo de dado, por exemplo). A fim de mostrar a capacidade de integração de diferentes sistemas, bem como a integração dos dados desses sistemas, conforme demonstrado pelo segundo estudo de caso aqui apresentado, foram integrados os *middlewares* UIoT e SmartCampus. O GDDIoT consegue lidar com a variedade dos formatos dos dados advindos dos *middlewares* IoT através do módulo de interface de comunicação, que ajuda a integrar diversos *middlewares* IoT de forma simultânea, padronizando o formato das mensagens coletadas. Após a integração dos sistemas, é importante a integração dos dados, que se dá através do módulo de criação de metadados do GDDIoT – módulo que lidar com a variedade de dados IoT através do uso de metadados que ajudam a integrar os dados advindos de diferentes ambientes IoT.

Sobre o processo de agregação de dados, a Tabela 7.1 apresenta uma coluna que representa o tempo de consulta durante um período de tempo. Os resultados das consultas sobre as tabelas sumarizadas obtiveram melhores resultados em comparação com o tempo de resposta das consultas sobre a tabela evento. Tal fato se dá devido à grande quantidade de dados armazenada na tabela evento. Por exemplo: para um milhão de objetos que fazem parte de uma ambiente IoT, é possível gerar cerca de 8.640 milhões de registros em um dia – uma consulta sobre esse volume de dados pode demorar muito para ser processada. Neste sentido, a consulta de dados por dia é mais lenta em comparação com a consulta de dados

por hora ou por minuto. É claro que o tempo de consulta sobre os dados sumarizados é muito rápido e, aparentemente, tem-se um tempo de processamento constante – consequência da reduzida quantidade de dados existentes nas tabelas supramencionadas, além do fato de comportar dados já pré-processados, ou seja, tem-se menos registros a contemplar, sendo necessário um menor processamento para a obtenção da mesma resposta fazendo uso da tabela evento.

Após a coleta de dados e o processamento dos dados IoT tem-se uma apresentação gráfica do componente de visualização de dados do GDDIoT – fato evidente através do segundo estudo de caso aqui empreendido. Tais painéis apresentam os dados, que podem ser genéricos ou personalizados. Neste contexto, muitos *middlewares* IoT contemplam algum tipo de mecanismo para a visualização dos dados, como, por exemplo, o *middleware* Kaa. Mas, regularmente, a visualização dos dados dos sistemas em questão é restritiva. Logo, o GDDIoT, através do módulo de consulta de dados, apresenta os dados de forma personalizada e segundo os requerimentos dos sistemas.

7.3 – CONSIDERAÇÕES

No processo de avaliação dos resultados decorrentes dos estudos de casos foi considerado a avaliação da funcionalidade e desempenho dos componentes do GDDIoT. A funcionalidade de cada componente do GDDIoT foi demonstrada através da capacidade de integração diferentes sistemas, captura, organização, processamento, armazenamento e visualização dos dados. Para a avaliação do desempenho foram usada diferentes métricas tais como a taxa de mensagens coletada por segundo e tempo de resposta das consultas realizada sobre os dados agregados.

Os resultados alcançados são consequência de sucessivas avaliações até lograr resultados que possam ser aceitáveis segundo os requisitos que foram definidos para o gerenciamento dos dados IoT. Através das diferentes avaliações ou testes foram avaliadas as funcionalidades de cada componente de forma individual e conjunta para demonstrar a funcionalidade do GDDIoT como todo. Enquanto aos resultados quantitativos, foram realizados sucessivos testes ajustando diferentes parâmetros ou realizando modificações na implementação dos componentes para tentar melhorar os resultados conseguidos.

Os resultados conseguidos pelos diferentes estudos de casos satisfaz a maioria dos requisitos que foram definidos para o gerenciamento de dados IoT, mas outros testes

podem ser realizados para abranger mais requisitos no gerenciamento dos dados IoT. Na literatura não foram encontradas avaliações semelhantes ao desenvolvido nesta tese com foco em características do gerenciamento de dados pouco estudado pelo trabalhos relacionados como apresentado no capítulo respectivo desta tese.

8 – CONCLUSÕES E TRABALHOS FUTUROS

A principal motivação para o gerenciamento de dados IoT reside no fato que a *Internet das Coisas* tem cada dia mais presença em diferentes áreas (indústria, Academia etc.) onde o volume de dados aumenta rapidamente. Muitos sistemas inteligentes – que envolvem casas, transporte, cidades, entre outros – necessitam do gerenciamento dos seus dados. Consequentemente, tem-se uma demanda de ferramentas eficazes para fins de gerenciamento de dados IoT. Neste contexto, a revisão da literatura aqui delineada evidenciou a existência de poucos estudos em relação ao gerenciamento de dados para Internet das coisa.

Nesta tese, se faz uso de tecnologias distribuídas para superar os requisitos do gerenciamento de dados no cenário da Internet das Coisas. Este trabalho foca-se no armazenamento e processamento de grandes quantidades de dados, sendo o modelo de armazenamento e processamento distribuído e paralelo como chave para resolver o problema da captura, organização e análise de dados em tempo real. Neste contexto, o gerenciamento de dados distribuídos para IoT baseou-se em um ambiente distribuído com programação paralela, com objetivo de suportar o processamento de grande volume de dados.

Neste contexto, esta tese apresentou um modelo para o gerenciamento de dados IoT. A análise decorrente dos resultados obtidos por três estudos de casos e a implementação do GDDIoT mostraram que em termos de gerenciamento de dados IoT, esta proposta trouxe benefícios, incluindo a integração de várias fontes de dados, com dados heterogêneos, capacidade de suportar grandes quantidades de dados, alto desempenho para consulta dos dados e a habilidade para a visualização dos dados gerenciados pelo GDDIoT.

No que tange à integração de diferentes fontes de dados, foi demonstrado a capacidade do GDDIoT de receber mensagens de diferentes *middlewares* IoT, tais como UIoT e SmartCampus. A integração dos diferentes middlewares é realizada de forma simultânea. Além disso, o GDDIoT conseguiu lidar com os diferentes tipos de mensagens provenientes desses *middlewares*. Isto devido ao mecanismo de padronização das mensagens vindas dos *middlewares* IoT, tal mecanismo é implementado pelo componente de interface de comunicação do GDDIoT. Após a integração das fontes de dados, o GDDIoT mostrou a capacidade da integração dos dados através do uso de metadados.

Desta forma, é possível lidar com a heterogeneidade dos dados, como por exemplo o domínio específico de uma aplicação.

Esta tese também propõe uma arquitetura distribuída para o processamento e armazenamento dos dados. Isto é devido ao suporte das tecnologias que contribuem ao requisito de processamento e armazenamento distribuído. Tecnologias tais como o Kafka, Storm e Cassandra são ferramentas que em parceria ajudam a implementação dos diferentes componentes do GDDIoT. Por exemplo, o GDDIoT conseguiu bons resultados no que diz respeito a coleção de grandes quantidades de dados em tempo real, isto grande medida ao modelo de processamento distribuído e paralelo realizado pelos diferentes módulos do componente de coleção de dados do GDDIoT.

O GDDIoT evidencia a capacidade para o processamento de dados históricos através da agregação de dados. No processo de agregação de dados tem-se a implementação do componente de agregação de dados do GDDIoT, que evidenciou um bom desempenho em termos de tempo de resposta quando é realizado a consulta dos dados agregados. Além disso, após a coleta, o processamento e o armazenamento dos dados, eles podem ser exibidos de forma gráfica em diferentes painéis e podendo ser personalizados segundo os requerimentos dos usuários.

Em resumo, a presente pesquisa é importante para a resolução de alguns dos problemas advindos da grande quantidades de dados IoT, sobretudo, a coleta, o armazenamento, a análise e o gerenciamento de tais dados. Sem dúvida, as reflexões críticas sobre o gerenciamento de dados IoT forneceram informações sobre os problemas atuais nesta área, permitindo compreender plenamente a temática apresentada.

5.1 – TRABALHOS FUTUROS

Como proposta de trabalhos futuros são indicados alguns pontos que podem ser evoluídos sobre o GDDIoT apresentado nesta tese. Existe a necessidade de desenvolver um mecanismo para detectar as mudanças do ambiente computacional para a configuração automática do módulo de organização de séries temporais – que pode ser realizado através de um procedimento que monitore as variáveis de ambiente de modo constante, além de definir a nova configuração do módulo.

Outro aspecto que merece atenção é a análise das mensagens descartadas pelo módulo de filtragem de dados. As mensagens descartadas podem conter informações

importantes referentes às causas de elas terem sido descartadas pelo módulo de filtragem de dados. Neste contexto, as regras e domínios usados pelo módulo de filtragem de dados poderiam ser armazenados no banco de dados, oferecendo mais flexibilidade na análise de domínio das medições dos objetos. Além disso, é possível a implementação de outros tipos de filtragem, visando uma ação especializada, conforme o tipo de objeto que faz as medições.

Em relação ao armazenamento dos dados no sistema gerenciador de banco de dados, a configuração do processo de inserção dos dados através do método de *microbatching* pode ser ajustada automaticamente, considerando variáveis tais como a intensidade de mensagens recebidas. Neste sentido, o tempo de inserção pode ser ajustada para menor tempo quando a intensidade seja alta e aumentando o tempo de inserção quando a intensidade seja baixa.

Finalmente, o GDDIoT não trata a geração do conhecimento para gestão do ambiente IoT, essa área é muito importante e pode ser considerada em diferentes trabalhos futuros, pois de fato, os dados são armazenados com o objetivo de gerar conhecimento.

5.2 – PUBLICAÇÕES RELACIONADAS A ESTE TRABALHO

As publicações dos trabalhos acadêmicos relacionados à presente Tese estão divididas em dois grupos, a saber: 1) trabalhos relacionados ao tema de pesquisa; e, 2) trabalhos específico ao tema de pesquisa. Os trabalhos relacionados ao tema de pesquisa são trabalhos associados ao gerenciamento de big data e uso de tecnologias NoSQL 2 . Já os trabalhos específicos ao tema de pesquisa são aqueles relacionados a tema de gerenciamento de dados distribuídos para IoT.

A seguir tem-se uma listagem dos trabalhos já publicados:

1) Trabalhos relacionados ao tema de pesquisa:

Huacarpuma, R. C., Rodrigues, D. D. C., Serrano, A. M. R., da Costa, J. P. C. L., de Sousa Jr, R. T., Holanda, M., & Araujo, A. P. F. (2013). “Big data: A case study on data from the Brazilian ministry of planning, budgeting and management”. *IADIS Applied Computing*. (pp. 201-205).

- Serrano, A. M. R., Rodrigues, P. H., **Huacarpuma, R. C**., da Costa, J. P. C. L., de Freitas, E. P., de Assis, V. L., & Pilon, B. H. (2014). "Improved Business Intelligence Solution with Reimbursement Tracking System for the Brazilian Ministry of Planning, Budget and Management". *In Knowledge Engineering and Knowledge Management* (pp. 434-440).
- **Huacarpuma, R. C**., Rodrigues, D. D. C., Serrano, A. M. R., da Costa, J. P. C. L., de Sousa Júnior, R. T., Leite, L., & Araujo, A. P. (2015). "Evaluating NoSQL Databases for Big Data Processing within the Brazilian Ministry of Planning, Budget, and Management". *In Artificial Intelligence Technologies and the Evolution of Web 3.0* (pp. 230-247). IGI Global.

2) Trabalhos específicos ao tema de pesquisa:

- **Huacarpuma, R. C**., de Sousa Jr, R. T., Holanda, M., & Lifschitz, S. (2016). "Concepção e Desenvolvimento de um Serviço Distribuído de Coleta e Tratamento de Dados para Ambientes de Internet das Coisas". *In Simpósio Brasileiro de Banco de Dados* (pp. 28-39).
- **Huacarpuma, R. C**., de Sousa Junior, R. T., de Holanda, M. T., de Oliveira Albuquerque, R., García Villalba, L. J., & Kim, T. H. (2017). "Distributed Data Service for Data Management in Internet of Things Middleware". *Sensors*, 17(5), 977.

REFERÊNCIAS BIBLIOGRÁFICAS

Aberer, K., Hauswirth, M., & Salehi, A. (2006). A Middleware for Fast and Flexible Sensor Network Deployment. In *Proceedings of the 32Nd International Conference on Very Large Data Bases* (p. 1199–1202). Seoul, Korea: VLDB Endowment.

Abu-Elkheir, M., Hayajneh, M., & Ali, N. A. (2013). Data management for the Internet of things: design primitives and solution. *Sensors* , *13*(11), 15582–15612.

Aiello, F., Fortino, G., Galzarano, S., & Vittorioso, A. (2011). TinyMAPS: A Lightweight Java-Based Mobile Agent System for Wireless Sensor Networks. In *Intelligent Distributed Computing V* (p. 161–170). Springer, Berlin, Heidelberg.

Alex, H., Kumar, M., & Shirazi, B. (2008). MidFusion: An adaptive middleware for information fusion in sensor network applications. *An international journal on information fusion*, *9*(3), 332–343.

Ashton, K. (2011). That "Internet of things" thing. *RFiD Journal*, *22*(7). Recuperado de http://www.itrco.jp/libraries/RFIDjournal-That%20Internet%20of%20Things%20Thing.pdf

Assunção, M. D., Calheiros, R. N., Bianchi, S., Netto, M. A., Buyya, R. (2015). Big Data computing and clouds: Trends and future directions. *Journal of parallel and distributed computing*, *79*(80), 3–15.

Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, *54*(15), 2787–2805.

Avilés, L. E., & Antonio, G. M. (2009). TinySOA: a service-oriented architecture for wireless sensor networks. *Service Oriented Computing and Applications*, *3*(2), 99–108.

Azzarà, A., Bocchino, S., Pagano, P., Pellerano, G., & Petracca, M. (2013). Middleware solutions in WSN: The IoT oriented approach in the ICSI project. In *21st International Conference on Software, Telecommunications and Computer Networks* (p. 1–6). Croacia.

Bahga, A., & Madisetti, V. (2014). *Internet of Things: A Hands-On Approach*. VPT.

Balani, R., Han, C., Rengaswamy, R. K., Tsigkogiannis, I., & Srivastava, M. (2006). Multi-level software reconfiguration for sensor networks. In *6th International*

conference on Embedded software Conference (p. 112–121). ACM.

Baresi, L., Guinea, S., & Saeedi, P. (2013). Achieving Self-adaptation through Dynamic Group Management. In J. Cámara, R. de Lemos, C. Ghezzi, & A. Lopes (Orgs.), *Assurances for Self-Adaptive Systems* (p. 214–239). Springer Berlin Heidelberg.

Bassi, A., Bauer, M., Fiedler, M., Kramp, T., Van, K. R., Lange, S., & Meissner, S. (2013). *Enabling Things to Talk: Designing IoT solutions with the IoT Architectural Reference Model*. Springer Berlin Heidelberg.

Bassi, A., & Horn, G. (2008). Internet of Things in 2020: A Roadmap for the Future. *European Commission: Information Society and Media*, *22*, 97–114.

Bonnet, P., Gehrke, J., & Seshadri, P. (2001). Towards Sensor Database Systems. In K. L. Tan, M. J. Franklin, & J. C. S. Lui (Orgs.), *Mobile Data Management* (Vol. 1987, p. 3–14). Berlin, Heidelberg: Springer Berlin Heidelberg.

Boulis, A., Han, C. C., Shea, R., & Srivastava, M. B. (2007). SensorWare: Programming sensor networks beyond code update and querying. *Pervasive and mobile computing*, *3*(4), 386–412.

Bray, T. (2014). *The JavaScript Object Notation (JSON) Data Interchange Format*. (T. Bray, Org.). RFC Editor. https://doi.org/10.17487/rfc7158.

Buerli, M., & Obispo, C. (2012). The current state of graph databases. *Department of Computer Science, Cal Poly San Luis Obispo, Mbuerli@ Calpoly. Edu*, *32*(3), 67–83.

Burstein, F., & Gregor, S. (1999). The systems development or engineering approach to research in information systems: An action research perspective. In *Proceedings of the 10th Australasian Conference on Information Systems* (p. 122–134). Victoria University of Wellington, New Zealand.

Caporuscio, M., Raverdy, P. G., & Issarny, V. (2012). ubiSOAP: A Service-Oriented Middleware for Ubiquitous Networking. *IEEE Transactions on Services Computing*, *5*(1), 86–98.

Carstoiu, D., Lepadatu, E., & Gaspar, M. (2014). Hbase-non sql database, performances evaluation. In H. András (Org.), *11th European Workshop, European Performance Engineering Workshop* (p. 16– 29). Florence, Italy: Springer International Publishing.

Cecchinel, C., Jimenez, M., Mosser, S., & Riveill, M. (2014). An Architecture to Support the Collection of Big Data in the Internet of Things. In *IEEE World Congress on Services* (p. 442–449).

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., … Gruber, R. E. (2008). Bigtable: A Distributed Storage System for Structured Data. *ACM Transactions on Computer Systems*, *26*(2), 1–26.

Chaqfeh, M. A., & Mohamed, N. (2012). Challenges in middleware solutions for the Internet of things. In *International Conference on Collaboration Technologies and Systems* (p. 21–26).

Chen, M., Mao, S., & Liu, Y. (2014). Big Data: A Survey. *Mobile Networks and Applications*, *19*(2), 171–209.

Chodorow, K. (2013). *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage*. "O'Reilly Media, Inc."

Chris, A. J., Lehnardt, J., & Slater, N. (2010). *CouchDB: The Definitive Guide: Time to Relax*. "O'Reilly Media, Inc."

Cockcroft, A., & Sheahan, D. (2011). Benchmarking cassandra scalability on aws-over a million writes per second. *URL: http://techblog. netflix. com/2011/11/benchmarking-cassandra-scalability-on. html (visited on 05/20/2013)*.

Cooper, J., & James, A. (2009). Challenges for Database Management in the Internet of Things. *IETE Technical Review*, *26*(5), 320–329.

Corredor, I., Martínez, J. F., Familiar, M. S., & López, L. (2012). Knowledge-Aware and Service-Oriented Middleware for deploying pervasive services. *Journal of Network and Computer Applications*, *35*(2), 562–576.

Costa, P., Coulson, G., Gold, R., Lad, M., Mascolo, C., Mottola, L., … Zachariadis, S. (2007). The RUNES Middleware for Networked Embedded Systems and its Application in a Disaster Management Scenario. In *5th International Conference on Pervasive Computing and Communications* (p. 69–78).

Costa, P., Mottola, L., Murphy, A. L., & Picco, G. P. (2006). TeenyLIME: Transiently Shared Tuple Space Middleware for Wireless Sensor Networks. In *Proceedings of the International Workshop on Middleware for Sensor Networks* (p. 43–48). New York, NY, USA: ACM.

Curino, C., Giani, M., Giorgetta, M., Giusti, A., Murphy, A. L., & Picco, G. P. (2005). Mobile data collection in sensor networks: The TinyLime middleware. *Pervasive and mobile computing*, *1*(4), 446–469.

da Silveira Bueno, R. (2008). *Econometria de séries temporais*. Cengage Learning.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., … Vogels, W. (2007). Dynamo: Amazon's Highly Available Key-value Store. *ACM SIGOPS Operating Systems Review*, *41*(6), 205–220.

Delicato, F. C., Pires, P. F., & Batista, T. (2013). *Middleware Solutions for the Internet of Things:* Springer London.

Dory, M., Parrish, A., & Berg, B. (2012). *Introduction to Tornado: Modern Web Applications with Python*. O'Reilly Media, Inc.

Edureka. (2017). Introduction to Cassandra Architecture - Edureka. Retrieved June 27, 2017, from https://www.edureka.co/blog/introduction-to-cassandra-architecture/.

Eisenhauer, M., Rosengren, P., & Antolin, P. (2010). HYDRA: A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems. In *The Internet of Things* (p. 367–373). Springer, New York, NY.

Evensen, P., & Meling, H. (2009). SenseWrap: A service oriented middleware with sensor virtualization and self-configuration. In *International Conference on Intelligent Sensors, Sensor Networks and Information Processing* (p. 261–266).

Ferreira, H. G. C. (2014). *Arquitetura de middleware para Internet das Coisas*. Recuperado de http://repositorio.unb.br/handle/10482/17251

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

Fette, I. (2011). *The websocket protocol* (No. 6455) (p. 1–70). tools.ietf.org. Recuperado de https://tools.ietf.org/html/rfc6455%3E

Fok, C. L., Roman, G. C., & Lu, C. (2009). Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. *ACM Transactions on Autonomous and Adaptive Systems*, *4*(4), 1–25.

Fok, C. L., Roman, G. C., & Lu, C. (2012). Servilla: A flexible service provisioning middleware for heterogeneous sensor networks. *Science of Computer Programming*, *77*(6), 663–684.

Fremantle, P. (2014). *A reference architecture for the Internet of Things* (Versão 0.8.2). WSO2. Recuperado de https://www.researchgate.net/profile/Paul_Fremantle/publication/308647314_A_Refe

rence_Architecture_for_the_Internet_of_Things/links/57ea00b708aef8bfcc963153.pdf

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

Ghosh, A., & Das, S. K. (2008). Coverage and connectivity issues in wireless sensor networks: A survey. *Pervasive and mobile computing*, *4*(3), 303–334.

Gibbons, P. B., Karp, B., Ke, Y., Nath, S., & Seshan, S. (2003). IrisNet: an architecture for a worldwide sensor Web. *IEEE pervasive computing / IEEE Computer Society [and] IEEE Communications Society*, *2*(4), 22–33.

Gregg, D. G., Kulkarni, U. R., & Vinzé, A. S. (2001). Understanding the Philosophical Underpinnings of Software Engineering Research in Information Systems. *Information Systems Frontiers*, *3*(2), 169–183.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generations computer systems*, *29*(7), 1645–1660.

Hadjigeorgiou, C., & Others. (2013). *Rdbms vs nosql: Performance and scaling comparison* (Master). The University of Edinburgh. Recuperado de https://static.epcc.ed.ac.uk/dissertations/hpc-msc/2012-2013/RDBMS%20vs%20NoSQL%20-%20Performance%20and%20Scaling%20Comparison.pdf

Hahn, U., & Mani, I. (2000). The challenges of automatic summarization. *Computer*, *33*(11), 29–36.

Hamida, A. B., Kon, F., Lago, N., Zarras, A., Athanasopoulos, D., Pilios, D., … Others. (2013). *Integrated CHOReOS middleware-Enabling large-scale, QoS-aware adaptive choreographies* (No. FP7-257178). IME-USP - Department of Computer Science. Recuperado de https://hal.inria.fr/hal-00912882/

Hamraz, S. H. (2013). *Internet of Things*. University of Kentucky. Recuperado de http://cs.engr.uky.edu/~hhamraz/reports/Internet_of_things.pdf

Han, Q., & Venkatasubramanian, N. (2001). Autosec: An integrated middleware framework for dynamic service brokering. *IEEE Distributed Systems Online*, *2*(7), 22–31.

Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., & Ullah Khan, S. (2015). The rise of “big data” on cloud computing: Review and open research issues. *Information systems*, *47*, 98–115.

Hasiotis, T., Alyfantis, G., Tsetsos, V., Sekkas, O., & Hadjiefthymiades, S. (2005). Sensation: a middleware integration platform for pervasive applications in wireless sensor networks. In *Proceeedings of the Second European Workshop on Wireless Sensor Networks* (p. 366–377).

Heinzelman, W. B., Murphy, A. L., Carvalho, H. S., & Perillo, M. A. (2004). Middleware to support sensor network applications. *IEEE network*, *18*(1), 6–14.

Henricksen, K., Indulska, J., McFadden, T., & Balasubramaniam, S. (2005). Middleware for Distributed Context-Aware Systems. In R. Meersman & Z. Tari (Orgs.), *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE* (Vol. 3760, p. 846–863). Berlin, Heidelberg: Springer Berlin Heidelberg.

Hong, K., Park, J., Kim, S., Kim, T., Kim, H., Burgstaller, B., & Scholz, B. (2012). TinyVM: an energy-efficient execution infrastructure for sensor networks. *Software: practice & experience*, *42*(10), 1193–1209.

Huacarpuma, R. C., de Sousa Junior, R. T., de Holanda, M. T., de Oliveira Albuquerque, R., García Villalba, L. J., & Kim, T.-H. (2017). Distributed Data Service for Data Management in Internet of Things Middleware. *Sensors* , *17*(5). https://doi.org/10.3390/s17050977

Huacarpuma, R. C., de Sousa, R. T., Jr, Holanda, M., & Lifschitz, S. (2016). Concepção e Desenvolvimento de um Serviço Distribuído de Coleta e Tratamento de Dados para Ambientes de Internet das Coisas. In *SBBD* (pp. 28–39). sbbd2016.fpc.ufba.br.

Huang, F. (2013, julho 7). *Web Technologies for the Internet of Things* (Master). (J. K. Nurminen, Org.). Aalto University.

Huebscher, M. C., & McCann, J. A. (2004). Adaptive middleware for context-aware applications in smart-homes. In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing* (p. 111–116). New York, NY, USA: ACM.

Hunkeler, U., Truong, H. L., & Stanford-Clark, A. (2008). MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks. In *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on* (pp. 791–798). IEEE.

Jain, A., & Nalya, A. (2014). *Learning Storm*. Packt Publishing.

Jouppi, N. P. (1993). Cache Write Policies And Performance. In *Proceedings of the 20th Annual International Symposium on Computer Architecture* (p. 191–201). New York,

NY, USA: ACM.

Junqueira, F., & Reed, B. (2013). *ZooKeeper: Distributed Process Coordination*. O'Reilly Media, Inc.

Kadam, R., Mahmauni, P., & Parikh, Y. (2015). Smart home system. *International Journal of Innovative research in Advanced Engineering*, *2*(1), 81–86.

KairosDB Team. (2015). KairosDB. Retrieved 27 de Junho 2017, from https://kairosdb.github.io/.

Kang, P., Borcea, C., Xu, G., Saxena, A., Kremer, U., & Iftode, L. (2004). Smart Messages: A Distributed Computing Platform for Networks of Embedded Systems. *Computer Journal*, *47*(4), 475–494.

Kang, Y., & Kang, K. (2016). Software Architecture for Building DDS Application in IoT Environment. *Advanced Science and Technology Letters*, *139*(2016), 123–126.

Karagiannis, V., Chatzimisios, P., Vazquez-Gallego, F., & Alonso-Zarate, J. (2015). A survey on application layer protocols for the Internet of things. *Transaction on IoT and Cloud Computing*, *3*(1), 11–17.

Kaur, S. B. S. (2015). Big Data and Hadoop. *International Journal of Advanced Engineering Research and Applications*, *1*(6), 233–238.

Kenda, K., Fortuna, C., Moraru, A., Mladenić, D., Fortuna, B., & Grobelnik, M. (2013). Mashups for the Web of Things. In B. Endres-Niggemeyer (Org.), *Semantic Mashups* (p. 145–169). Springer Berlin Heidelberg.

Khan, M. A., Uddin, M. F., & Gupta, N. (2014). Seven V's of Big Data understanding Big Data to extract value. In *Conference of the American Society for Engineering Education* (p. 1–5). Connecticut, USA: University of Bridgeport.

Khetrapal, A., & Ganesh, V. (2006). HBase and Hypertable for large scale distributed storage systems. *Dept. of Computer Science, Purdue University*, 22–28.

Kirsch, C. M., Sanvido, M. A. A., & Henzinger, T. A. (2005). A programmable microkernel for real-time systems. In *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments* (p. 35–45). New York, NY, USA: ACM.

Koshy, J., & Pandey, R. (2005). vm 击: Synthesizing scalable runtime environments for sensor networks. In *Proceedings of the International Conference on Embedded Networked Sensor Systems* (p. 243–254). New York, NY, USA: University of

Massachusetts.

Kreps, J., Narkhede, N., Rao, J., & Others. (2011). Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB* (p. 1–7).

Kwon, Y., Sundresh, S., Mechitov, K., & Agha, G. (2006). ActorNet: An Actor Platform for Wireless Sensor Networks. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems* (p. 1297–1300). New York, NY, USA: ACM.

Lakshman, A., & Malik, P. (2010). Cassandra: A Decentralized Structured Storage System. *ACM Special Interest Group on Operating Systems*, *44*(2), 35–40.

Levis, P., & Culler, D. (2002). Maté: A tiny virtual machine for sensor networks. *ACM Sigplan Notices*, 85–95.

Levis, P., Gay, D., & Culler, D. (2005). Active sensor networks. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2* (p. 343–356). Berkeley, CA, USA: USENIX Association.

Lima, A., Coelho, F. L., Lisboa F., C. R., & Weyl A. C., J. C. (2008). SensorBus - A Policy-Based Middleware for Wireless Sensor Networks. *IEEE Latin America Transactions*, *6*(7), 647–654.

Lima, R. D. C. A., Rosa, N. S., Marques, I. R. L. (2008). TS-Mid: Middleware for Wireless Sensor Networks Based on Tuple Space. In *Proceedings of the 22Nd International Conference on Advanced Information Networking and Applications* (p. 886–891). Washington, DC, USA: IEEE Computer Society.

Li, M., Liu, Y., & Cai, Y. (2015). A Dynamic Processing System for Sensor Data in IoT. *International Journal of Distributed Sensor Networks*, *11*(8), 750452.

Li, T., Liu, Y., Tian, Y., Shen, S., & Mao, W. (2012). A Storage Solution for Massive IoT Data Based on NoSQL. In *2012 IEEE International Conference on Green Computing and Communications* (p. 50–57). Washington, DC, USA: IEEE Computer Society.

Liu, T., & Martonosi, M. (2003). Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems. *SIGPLAN Not.*, *38*(10), 107–118.

Lopes, R., Assis, F., & Montez, C. (2011). MASPOT: A Mobile Agent System for Sun SPOT. In *10th International Symposium on Autonomous Decentralized Systems* (p. 25–31).

Lu, B., & Xiaohui, Y. (2016). Research on Cassandra Data Compaction Strategies for

Time-Series Data. *Journal of Computational Physics*, *11*(6), 504–512.

Luo, C., Wu, F., Sun, J., & Chen, C. W. (2009). Compressive Data Gathering for Large-scale Wireless Sensor Networks. In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking* (p. 145–156). New York, NY, USA: ACM.

Madden, S. R., Franklin, M. J., Hellerstein, J. M., & Hong, W. (2005). TinyDB: An Acquisitional Query Processing System for Sensor Networks. *ACM Transactions on. Database Systems*, *30*(1), 122–173.

Naheman, W., & Wei, J. (2013). Review of NoSQL databases and performance testing on HBase. In *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)* (pp. 2304–2309). ieeexplore.ieee.org.

Mamei, M., & Zambonelli, F. (2006). *Field-Based Coordination for Pervasive Multiagent Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Marques, I. L., Ronan, J., & Rosa, N. S. (2009). TinyReef: a register-based virtual machine for Wireless Sensor Networks. In *IEEE Sensors* (p. 1423–1426).

Marrón, P. J., Minder, D., Lachenmann, A., & Rothermel, K. (2005). TinyCubus: An Adaptive Cross-Layer Framework for Sensor Networks (TinyCubus: Ein Adaptives Cross-Layer Framework für Sensornetze). *Information Technology*, *47*(2), 87–97.

Marz, N., & Warren, J. (2015). *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications Co.

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

Mattern, F., & Floerkemeier, C. (2010). From the Internet of Computers to the Internet of Things. *From active data management to event-based systems and more*, 242–259.

Merk, L., Nicklous, M. S., Stober, T., & Hansmann, U. (2001). Pervasive computing handbook. Springer Verlag.

Misra, G., Kumar, V., Agarwal, A., & Agarwal, K. (2016). Internet of things (iot)--a technological analysis and survey on vision, concepts, challenges, innovation directions, technologies, and applications (an upcoming or future generation computer communication system technology). *American Journal of Electrical and Electronic Engineering*, *4*(1), 23–32.

Misra, P., Simmhan, Y., & Warrior, J. (2015, fevereiro 3). *Towards a Practical Architecture for the Next Generation Internet of Things*. *arXiv [cs.CY]*. Recuperado de http://arxiv.org/abs/1502.00797

Morstatter, F., Pfeffer, J., & Liu, H. (2014). When is It Biased?: Assessing the Representativeness of Twitter's Streaming API. In *Proceedings of the 23rd International Conference on World Wide Web* (p. 555–556). New York, NY, USA: ACM.

Mottola, L., Murphy, A. L., & Picco, G. P. (2006). Pervasive games in a mote-enabled virtual world using tuple space middleware. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games* (p. 29). New York, NY, USA: ACM.

Mueller, R., Alonso, G., & Kossmann, D. (2007). SwissQM: Next Generation Data Processing in Sensor Networks. In *Conference on Innovative Data Systems Research* (Vol. 7, p. 1–9).

Muhammad, Y. (2011). Evaluation and Implementation of Distributed NoSQL Database for MMO Gaming Environment. diva-portal.org. Retrieved from http://www.diva-portal.org/smash/get/diva2:447210/FULLTEXT01.pdf

Muldoon, C., O'Hare, G. M. P., Collier, R., & O'Grady, M. J. (2006). Agent Factory Micro Edition: A Framework for Ambient Applications. In V. N. Alexandrov, G. D. van Albada, P. M. A. Sloot, & J. Dongarra (Orgs.), *Computational Science – ICCS 2006* (Vol. 3993, p. 727–734). Berlin, Heidelberg: Springer Berlin Heidelberg.

Murphy, A. L., Picco, G. P., & Roman, G. C. (2001). LIME: a middleware for physical and logical mobility. In *Proceedings 21st International Conference on Distributed Computing Systems* (p. 524–533). Washington, DC, USA: IEEE Computer Society.

Musolesi, M., Mascolo, C., & Hailes, S. (2004). Adapting asynchronous messaging middleware to ad hoc networking. In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing* (p. 121–126). ACM.

Musolesi, M., Mascolo, C., & Hailes, S. (2006). EMMA: Epidemic Messaging Middleware for Ad hoc networks. *Personal and Ubiquitous Computing*, *10*(1), 28–36.

Nagy, M., Katasonov, A., Khriyenko, O., Nikitin, S., Szydlowski, M., & Terziyan, V. (2009). Challenges of middleware for the Internet of things. In *Automation Control-Theory and Practice*. InTech.

Namiot, D. (2015). On big data stream processing. *International Journal of Open Information Technologies*, *3*(8), 48–51.

Nayak, A., Poriya, A., & Poojary, D. (2013). Type of NOSQL databases and its comparison with relational databases. *International Journal of Applied Information Systems*, *5*(4), 16–19.

Neuman, L. (2002). *Social Research Methods: Qualitative and Quantitative Approaches*. Allyn & Bacon.

Noll, M. G. (2015). Running a Multi-Node Storm Cluster.

Nunamaker, J. F., Chen, M., & Purdin, T. D. M. (1990). Systems Development in Information Systems Research. *Journal of Management Information Systems*, *7*(3), 89–106.

Nurseitov, N., Paulson, M., Reynolds, R., & Izurieta, C. (2009). Comparison of JSON and XML data interchange formats: a case study. In *New Developments in Circuits, Systems, Signal Processing, Communications and Computers* (Vol. 2009, p. 157–162).

O'Leary, D. E. (2013). "BIG DATA", THE "Internet OF THINGS" AND THE "Internet OF SIGNS". *Intelligent Systems in Accounting, Finance and Management*, *20*(1), 53–65.

Ödegaard, T. (2016). Grafana, The Leading Graph And Dashboard Builder For Visualizing Time Series Metrics.

Perera, C., Jayaraman, P. P., Zaslavsky, A., Christen, P., & Georgakopoulos, D. (2014). MOSDEN: An Internet of Things Middleware for Resource Constrained Mobile Devices. In *47th Hawaii International Conference on System Sciences* (p. 1053–1062). Waikoloa, HI, USA.

Perera, C., Zaslavsky, A., Christen, P., & Georgakopoulos, D. (2014). Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys Tutorials*, *16*(1), 414–454.

Pietzuch, P. R. (2004). *Hermes: A scalable event-based middleware*. University of Cambridge, Computer Laboratory. Recuperado de http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-590.html

Pimentel, V., & Nickerson, B. (2012). WebSocket for communication and display of real-time data [j]. *Internet Computing*, 364–368.

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 *2011 IFIP 9th International Conference on Embedded and Ubiquitous Computing*. https://doi.org/10.1109/euc.2011.59

Rafiei, D., & Mendelzon, A. (1997). Similarity-based Queries for Time Series Data. In *Proceedings ACM Special Interest Group on Management of Data* (p. 13–25). New York, NY, USA: ACM.

Ramakrishnan, R., & Gehrke, J. (2000). *Database management systems*. McGraw Hill.

Razzaque, M. A., Milojevic-Jevric, M., Palade, A., & Clarke, S. (2016). Middleware for Internet of Things: A Survey. *IEEE Internet of Things Journal*, *3*(1), 70–95.

Rouvoy, R., Barone, P., Ding, Y., Eliassen, F., Hallsteinsen, S., Lorenzo, J., … Scholz, U. (2009). MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments. In B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, & J. Magee (Orgs.), *Software Engineering for Self-Adaptive Systems* (p. 164–182). Springer Berlin Heidelberg.

Russom, P., & Others. (2011). Big data analytics. *TDWI best practices report, fourth quarter*, *19*, 40.

Sang, Y., Shen, H., Inoguchi, Y., Tan, Y., & Xiong, N. (2006). Secure Data Aggregation in Wireless Sensor Networks: A Survey. In *2006 Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'06)* (p. 315–320). ieeexplore.ieee.org.

Sciore, E. (2007). SimpleDB: A Simple Java-based Multiuser Syst for Teaching Database Internals. *SIGCSE Bull.*, *39*(1), 561–565.

Shahrivari, S. (2014). Beyond Batch Processing: Towards Real-Time and Streaming Big Data. *Computers*, *3*(4), 117–129.

Shen, C.-C., Srisathapornphat, C., & Jaikaeo, C. (2001). Sensor information networking architecture and applications. *IEEE Personal Communications*, *8*(4), 52–59.

Sheth, A., Henson, C., & Sahoo, S. S. (2008). Semantic Sensor Web. *IEEE Internet Computing*, *12*(4), 78–83.

Shin, K. G., & Ramanathan, P. (1994). Real-time computing: a new discipline of computer science and engineering. *Proceedings of the IEEE*, *82*(1), 6–24.

Silva, J. R., Delicato, F. C., Pirmez, L., Pires, P. F., Portocarrero, J. M. T., Rodrigues, T.

C., & Batista, T. V. (2014). PRISMA: A publish-subscribe and resource-oriented middleware for wireless sensor networks. In *Proceedings of the Tenth Advanced International Conference on Telecommunications* (Vol. 2024, p. 87–97). Paris, France: IARA.

Simon, D., Cifuentes, C., Cleal, D., Daniels, J., & White, D. (2006). Java™ on the bare metal of wireless sensor devices: the squawk Java virtual machine. In *Proceedings of the 2nd international conference on Virtual execution environments* (p. 78–88). New York, NY, USA: ACM.

Sivaharan, T., Blair, G., & Coulson, G. (2005). GREEN: A Configurable and Re-configurable Publish-Subscribe Middleware for Pervasive Computing. In R. Meersman & Z. Tari (Orgs.), *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE* (Vol. 3760, p. 732–749). Berlin, Heidelberg: Springer Berlin Heidelberg.

Sivasubramanian, S. (2012). Amazon dynamoDB: A Seamlessly Scalable Non-relational Database Service. In *Proceedings in Special Interest Group on Management of Data* (p. 729–730). New York, NY, USA: ACM.

Soldatos, J., Serrano, M., & Hauswirth, M. (2012). Convergence of Utility Computing with the Internet-of-Things. In *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. https://doi.org/10.1109/imis.2012.135

Souto, E., Guimarães, G., Vasconcelos, G., Vieira, M., Rosa, N., Ferraz, C., & Kelner, J. (2006). Mires: a publish/subscribe middleware for sensor networks. *Personal and Ubiquitous Computing*, *10*(1), 37–44.

Stankovic, J. A., & Others. (1992). Real-time computing. *Springer*, *127*, 65–82.

Stonebraker, M., & Cetintemel, U. (2005). " One size fits all": an idea whose time has come and gone. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on* (p. 2–11). Washington, DC, USA: IEEE.

Strauch, C., Sites, U.-L. S., & Kriha, W. (2011). NoSQL databases. In *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services* (Vol. 20, p. 278–283}). New York, NY, USA: ACM.

Streamparse. (2017). Streamparse. Acessado em 20 de Julho, 2017, de http://streamparse.readthedocs.io/en/stable/index.html

Tamer, M., & Valduriez, P. (2011). *Principles of Distributed Database Systems*. Springer

Science & Business Media.

Technologies, K. (2017, July 20). Kaa Open-Source IoT Platform 2017. Retrieved July 20, 2017, from https://www.kaaproject.org/

Teixeira, T., Hachem, S., Issarny, V., & Georgantas, N. (2011). Service Oriented Middleware for the Internet of Things: A Perspective. In W. Abramowicz, I. M. Llorente, M. Surridge, A. Zisman, & J. Vayssière (Orgs.), *Towards a Service-Based Internet* (Vol. 6994, p. 220–229). Berlin, Heidelberg: Springer Berlin Heidelberg.

Terziyan, V., Kaykova, O., & Zhovtobryukh, D. (2010). UbiRoad: Semantic Middleware for Context-Aware Smart Road Environments. In *2010 Fifth International Conference on Internet and Web Applications and Services* (p. 295–302). Barcelona, Spain.

Tesoriero, C. (2013). *Getting Started with OrientDB*. Packt Publishing Ltd.

Toma, I., Simperl, E., & Hench, G. (2009). A joint roadmap for semantic technologies and the Internet of things. In *Proceedings of the Third STI Roadmapping Workshop* (Vol. 1). Crete, Greece. Recuperado de https://www.researchgate.net/profile/Ioan_Toma/publication/228667796_A_joint_roadmap_for_semantic_technologies_and_the_Internet_of_Things/links/00b7d5379e95a3c696000000.pdf

Tsiatsis, V., Gluhak, A., Bauge, T., Montagut, F., Bernat, J., Bauer, M., … Krco, S. (2010). The SENSEI Real World Internet Architecture. In G. Tselentis, A. Galis, A. Gavras, S. Krco, V. Lotz, E. Simperl, … T. Zahariadis (Orgs.), *Towards the Future Internet* (p. 247–256). Ios Pr Inc.

Vargas, W., Munoz, A., & Rodríguez, J. (2017). A distributed system model for managing data ingestion in a wireless sensor network. In *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)* (p. 1–5).

Villari, M., Celesti, A., Fazio, M., & Puliafito, A. (2014). AllJoyn Lambda: An architecture for the management of smart environments in IoT. In *2014 International Conference on Smart Computing Workshops* (p. 9–14). Hong Kong, China .

Vongsingthong, S. (2015). A review of data management in Internet of things. *Asia-Pacific Journal of Science and Technology*, *20*(2), 215–240.

Wang, V., Salim, F., & Moskovits, P. (2013). Introduction to HTML5 WebSocket. In *The Definitive Guide to HTML5 WebSocket* (p. 1–12). Apress.

Webber, J. (2012). A Programmatic Introduction to Neo4J. In *Proceedings of the 3rd*

Annual Conference on Systems, Programming, and Applications: Software for Humanity (pp. 217–218). New York, NY, USA: ACM.

Wlodarczyk, T. W. (2012). Overview of Time Series Storage and Processing in a Cloud Environment. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings* (pp. 625–628). ieeexplore.ieee.org.

Xu, L. D., He, W., & Li, S. (2014). Internet of Things in Industries: A Survey. *IEEE Transactions on Industrial Informatics*, *10*(4), 2233–2243.

Yasumoto, K., Yamaguchi, H., & Shigeno, H. (2016a). Survey of Real-time Processing Technologies of IoT Data Streams. *Journal of Information Processing*, *24*(2), 195–202.

Yasumoto, K., Yamaguchi, H., & Shigeno, H. (2016b). Survey of Real-time Processing Technologies of IoT Data Streams. *Journal of Information Processing*, *24*(2), 195–202.

Yu, Y., Rittle, L. J., Bhandari, V., & LeBrun, J. B. (2006). Supporting concurrent applications in wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems* (p. 139–152). Boulder, Colorado, USA : ACM.

Žarko, I. P., Pripužić, K., Serrano, M., & Hauswirth, M. (2014). IoT data management methods and optimisation algorithms for mobile publish/subscribe services in cloud environments. In *European Conference on Networks and Communications* (p. 1–5). Bologna, Italy.

Zaslavsky, A., Perera, C., & Georgakopoulos, D. (2013, janeiro 2). *Sensing as a Service and Big Data*. *arXiv [cs.CY]*. Recuperado de http://arxiv.org/abs/1301.0159

Zawodny, J. (2009). Redis: Lightweight key/value store that goes the extra mile. *Linux Magazine*, *79*.

Zechner, K. (1997). A literature survey on information extraction and text summarization. *Computational Linguistics Program*, *22*, 1–27.

Zhao, D., & Raicu, I. (2013). HyCache: A User-Level Caching Middleware for Distributed File Systems. In *IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum* (p. 1997–2006). Washington, DC, USA: IEEE Computer Society.

Zhu, S. (2015). Creating a NoSQL database for the Internet of Things : Creating a

key-value store on the SensibleThings platform. Mid Sweden University. Recuperado de http://www.diva-portal.org/smash/record.jsf?pid=diva2:841604.

ZooKeeper, A. (2013). Apache ZooKeeper.