Programa Interdisciplinar de Pós-Graduação em

Computação Aplicada

Mestrado Acadêmico

Márcio Miguel Gomes

Eliot - Uma Arquitetura Para Internet das Coisas: Explorando a Elasticidade da Computação em Nuvem Com Alto Desempenho

São Leopoldo, 2015

Márcio Miguel Gomes

ELIOT - UMA ARQUITETURA PARA INTERNET DAS COISAS: **Explorando a Elasticidade da Computação em Nuvem com Alto Desempenho**

Dissertação apresentada como requisito parcial para a obtenção do título de Mestre pelo Programa de Pós-Graduação em Computação Aplicada da Universidade do Vale do Rio dos Sinos — UNISINOS

Orientador:
Prof. Dr. Cristiano André da Costa

Coorientador:
Prof. Dr. Rodrigo da Rosa Righi

São Leopoldo
2015

G633e Gomes, Márcio Miguel.
Eliot : uma arquitetura para internet das coisas : explorando a elasticidade da computação em nuvem com alto desempenho / Márcio Miguel Gomes. – 2015.
102 f. : il. ; 30 cm.

Dissertação (mestrado) – Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, 2015.
"Orientador: Prof. Dr. Cristiano A. Costa ; coorientador: Prof. Dr. Rodrigo da Rosa Righi."

1. Internet das coisas. 2. Sistemas de identificação por radiofrequência. 3. Middleware. 4. Redes de computadores – Escalabilidade. 5. Computação em nuvem. I. Título.
CDU 004

Dados Internacionais de Catalogação na Publicação (CIP)
(Bibliotecário: Flávio Nunes – CRB 10/1298)

UNISINOS

UNIVERSIDADE DO VALE DO RIO DOS SINOS
Ciências Exatas e Tecnológicas
Programa Interdisciplinar de Pós-Graduação em Computação Aplicada – PIPCA

Márcio Miguel Gomes

Eliot - Uma Arquitetura Para Internet das Coisas: Explorando a Elasticidade da Computação em Nuvem Com Alto Desempenho

Dissertação apresentada à Universidade do Vale do Rio dos Sinos – Unisinos, como requisito parcial para obtenção do título de Mestre em Computação Aplicada.

Aprovado em ____/____/_____

BANCA EXAMINADORA

Prof. Dr. Philippe Olivier Navaux – UFRGS

Prof. Dr. Jorge Luiz Victoria Barbosa – UNISINOS

Prof. Dr. Rodrigo da Rosa Righi

Prof. Dr. Cristiano André da Costa (Orientador)

Visto e permitida a impressão

São Leopoldo, ____ de _____ de 2015

Prof. Dr. Cristiano André da Costa
Coordenador PPG em Computação Aplicada

RESUMO

O universo digital vem crescendo a taxas expressivas nos últimos anos. Um dos principais responsáveis por esse aumento no volume de dados é a Internet das Coisas, que em uma definição simplista, consiste em identificar unicamente objetos de forma eletrônica, rastreá-los e armazenar suas informações para posterior utilização. Para lidar com tamanha carga de dados, são necessárias soluções a nível de *software*, *hardware* e arquitetura. Estudos realizados neste trabalho apontam que a arquitetura adotada atualmente apresenta limitações, principalmente no quesito escalabilidade.

Como escalabilidade é uma característica fundamental para atender a demanda crescente de coleta, processamento e armazenamento de dados, este trabalho apresenta a arquitetura intitulada Eliot, com propostas para resolver justamente a escalabilidade além de oferecer elasticidade ao sistema. Para isso, está sendo proposto o uso de técnicas de bancos de dados distribuídos, processamento paralelo e nuvem computacional, além de reestruturação da arquitetura atual.

Os resultados obtidos após a implantação e avaliação do Eliot em um ambiente de nuvem computacional comprovam a viabilidade, eficiência e confiabilidade dessa nova arquitetura proposta. Foi possível identificar melhora do desempenho através da redução nos tempos de resposta e aumento do volume de requisições processadas e trafegadas na rede além da redução nas falhas de conexão e de comunicação de dados.

Palavras-chave: Internet das Coisas. IoT. RFID. Middleware. EPCglobal. Escalabilidade. Elasticidade. Nuvem Computacional.

ABSTRACT

The digital universe is growing at significant rates in recent years. One of the main responsible for this increase in the volume of data is the Internet of Things, which in a simplistic definition, is to uniquely identify objects electronically, track them and store their information for later use. To deal with such data load, solutions are needed at software, hardware and architecture levels. Studies conducted in this work show that the architecture adopted currently has limitations, specially regarding scalability.

As scalability is a key feature to meet the growing demand for collection, processing and storage of data, this paper presents the architecture entitled Eliot, with proposals to precisely resolve the scalability and offers elasticity to the system. For this, is proposed the use of techniques of distributed databases, parallel processing and cloud computing, as well as restructuring of the current architecture.

The results obtained after the implementation and evaluation of Eliot in a cloud computing environment demonstrate the feasibility, efficiency and reliability of this new proposed architecture. It was possible to improve performance by reducing response times and increased volume of requisitions processed and trafficked in the network, in addition to the reduction in connection failures and data communication.

Keywords: Internet of Thinigs. IoT. RFID. Middleware. EPCglobal. Scalability. Elasticity. Cloud Computing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE SIGLAS

ALE	*Application Level Events*
API	*Application Programming Interface*
CPU	*Central Processing Unit*
EC2	*Elastic Compute Cloud*
EPC	*Electronic Product Code*
EPCIS	*Electronic Product Code Information Services*
HTTP	*Hypertext Transfer Protocol*
IoT	*Internet of Things*
IP	*Internet Protocol*
ISO	*International Organization for Standardization*
LLRP	*Low-Level Reader Protocol*
P2P	*Peer-to-Peer*
RAM	*Random Access Memory*
RFID	*Radio Frequency Identifier*
SOAP	*Simple Object Access Protocol*
SQL	*Structured Query Language*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*

SUMÁRIO

1 INTRODUÇÃO

> *"Os computadores de hoje, e portanto a Internet, são quase totalmente dependentes de seres humanos para obter informações. Quase todos os cerca de 50 petabytes de dados disponíveis na Internet foram primeiramente capturados ou criados por seres humanos, digitando, apertando um botão de gravação, tirando uma foto digital ou lendo um código de barras. O problema é que as pessoas têm tempo, atenção e precisão limitados, o que significa que não somos muito bons em capturar dados sobre as coisas do mundo real. Se tivéssemos computadores que soubessem tudo que havia para saber sobre as coisas, usando dados coletados sem qualquer ajuda humana, nós seríamos capazes de monitorar e contar tudo, reduzindo muito o desperdício, perdas e custos. Gostaríamos de saber quando as coisas precisassem de substituição, reparação ou revisão, e se elas estivessem frescas ou passadas do ponto."*
>
> Kevin Ashton[1]

A Internet das Coisas, também referenciada em inglês como *Internet of Things* ou simplesmente IoT, é um cenário em que os objetos, animais ou pessoas estão equipados com identificadores únicos, com capacidade de transferir automaticamente seus dados através de uma rede sem a necessidade de intervenção humana (ATZORI; IERA; MORABITO, 2010). A *coisa* no termo "Internet das Coisas" pode ser uma pessoa com um implante de monitor cardíaco, um animal de uma fazenda de criação com um biochip, um automóvel que foi construído com sensores para alertar o condutor quando algo não está bem, ou qualquer outro objeto natural ou feito pelo homem no qual se possa atribuir um endereço IP e que tenha a capacidade de transferir dados através de uma rede (XU; HE; LI, 2014).

A IoT vem promovendo um mundo cada vez mais conectado e já movimenta um mercado gigantesco. De acordo com a consultoria IDC[2], somente no Brasil cerca de US$ 2 bilhões devem girar em torno da IoT em 2014 e mais de US$ 8,9 trilhões serão gerados até 2020 em todo o mundo. Além dos valores monetários, o que impressiona são os volumes de dados envolvidos e as taxas de crescimento previstas. Conforme esse mesmo estudo, o universo digital vem dobrando de tamanho a cada 2 anos, e pode vir a ser multiplicado por 10 entre os anos de 2013 e 2020. Isso significa um crescimento dos atuais 4,4 trilhões de gigabytes para 44 trilhões de gigabytes em apenas 7 anos.

Dos quase 200 bilhões de objetos existentes atualmente com potencial capacidade de se conectar à Internet, apenas 7% já se comunicam e apenas 2% produzem dados que podem ser armazenados. A IDC também, afirma que o mercado digital atualmente é dominado por países

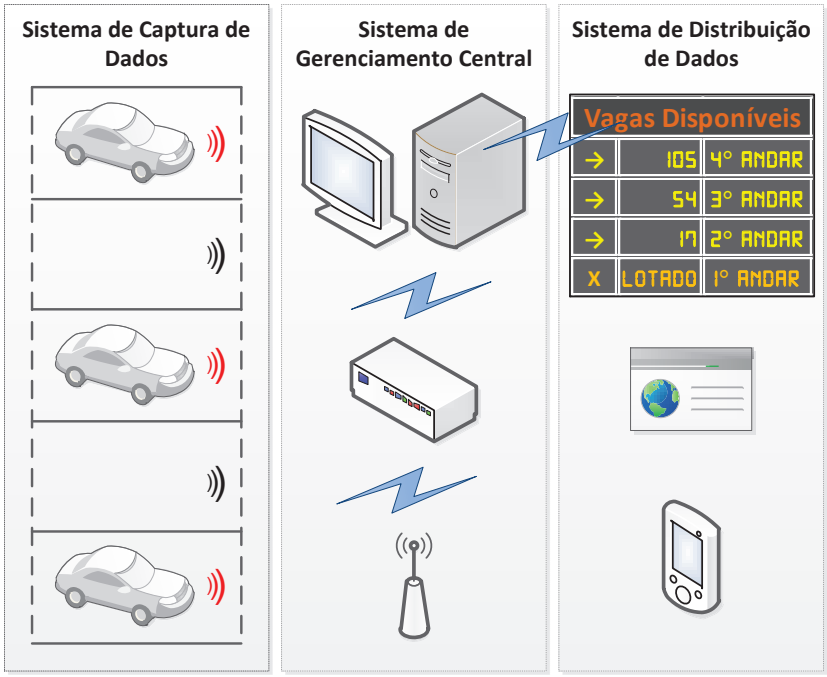
[1] A citação acima foi feita por Ashton (2009), co-fundador e diretor executivo do Auto-ID Center do MIT durante uma apresentação feita em 1999 para a empresa Procter & Gamble, mencionando pela primeira vez o conceito de "Internet das Coisas".

[2] Estudo conduzido pelo IDC (International Data Corporation) entitulado "*The Digital Universe of Opportunities: Rich Data and Increasing the Value of the Internet of Things*", publicado em abril de 2014, disponível em http://www.emc.com/leadership/digital-universe/2014iview/index.htm

industrializados, mas os países emergentes vão produzir rapidamente mais dados do que os países desenvolvidos. Esse estudo indica que a situação deve ser revertida em 2020, com o Brasil, China, Índia, México e Rússia desempenhando o papel central. A estimativa é que no ano de 2020, o número de dispositivos conectados à Internet vai crescer para 30 bilhões de unidades. Para finalizar, no estudo da IDC foi previsto um aumento considerável nos dados armazenados ou processados na nuvem. Em 2013, menos de 20% dos dados no universo digital passaram pela nuvem, mas em 2020 esse percentual deve aumentar para 40%. Dois terços de todos os dados produzidos no mundo são gerados por pessoas ou aplicativos pessoais, e 85% são relativos a processos empresariais.

Fora dos ambientes industriais ou empresariais, a IoT proporciona que pessoas forneçam e consumam informações das mais variadas formas. Como exemplo de aplicação da IoT, a Figura 1 dá uma ideia de como é possível ter sistemas de estacionamento inteligentes que informem em tempo real a disponibilidade de vagas em toda a cidade (PALA; INANC, 2007). Moradores poderiam identificar e reservar as vagas disponíveis mais próximas de suas casas, e os guardas de trânsito poderiam reconhecer carros estacionados em lugares indevidos. Além de melhorar o aproveitamento do uso dos estacionamentos, os congestionamentos também reduziriam, pois os motoristas saberiam exatamente onde estacionar os carros em vez de ficarem rodando por vários minutos até encontrarem uma vaga disponível. Ainda sobre trânsito, ao automatizar pagamentos de pedágio através da identificação automática dos veículos, seria possível melhorar as condições gerais do trânsito principalmente em situações de tráfego intenso, além de reduzir os custos com guichês, pessoas e transporte de valores.

Figura 1 – Exemplo de uso da IoT em estacionamentos

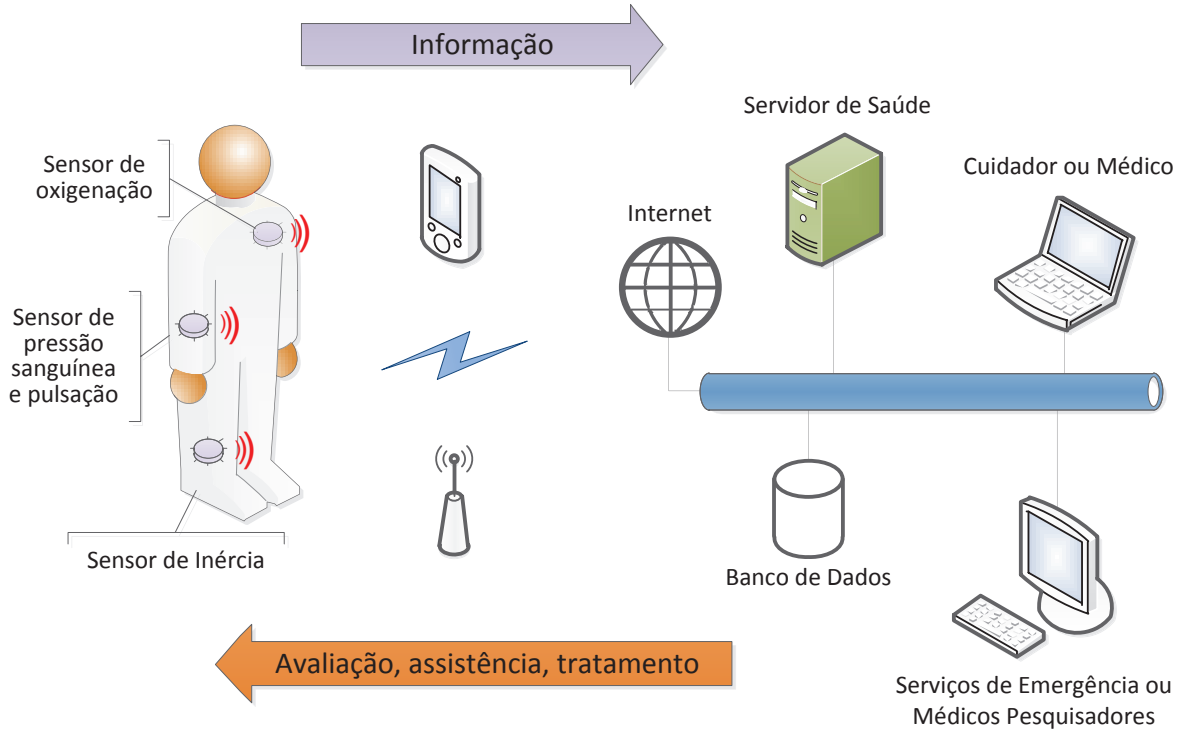


Fonte: Elaborado pelo autor

Também seria possível aplicar a IoT em sistemas de leitura de água, energia elétrica e gás utilizando equipamentos medidores de vazão com acesso a Internet (LI et al., 2009). Além de reduzir custos com leituras manuais nos relógios medidores, também aumentaria a precisão das medições e ofereceria valores de consumo em tempo real, podendo ser segmentado por bairro, rua e até por unidade consumidora. Isso permitira que as empresas fornecedoras conhecessem minuciosamente o perfil de consumo de cada cliente ou localidade, facilitando o planejamento de manutenções preventivas, detecção de vazamentos ou perdas e até servindo de apoio para tomada de decisão para novos investimentos em infraestrutura.

Na área da saúde, existem diversos estudos comprovando a possibilidade de realizar o monitoramento remoto residencial de pacientes com doenças crônicas como insuficiência cardíaca, doenças respiratórias ou diabetes (JIANG et al., 2008; DOKOVSKY; HALTEREN; WIDYA, 2004; LUBRIN; LAWRENCE; NAVARRO, 2005; LIM et al., 2010; LIN et al., 2012). Além de oferecer um monitoramento em tempo real, permitindo um pronto atendimento muito rápido em caso de emergência, o cenário ilustrado na Figura 2 também proporciona diminuir custos ao oferecer cuidados de saúde em ambientes domésticos mais baratos do que em clínicas ou hospitais. Outra vantagem seria ter um histórico de monitoramento constante dos sinais vitais do paciente como pressão sanguínea, frequência cardíaca, frequência respiratória, oxigenação, glicose, etc, facilitando o diagnóstico de novas doenças ou evolução do quadro atual.

Figura 2 – Exemplo de uso da IoT na saúde



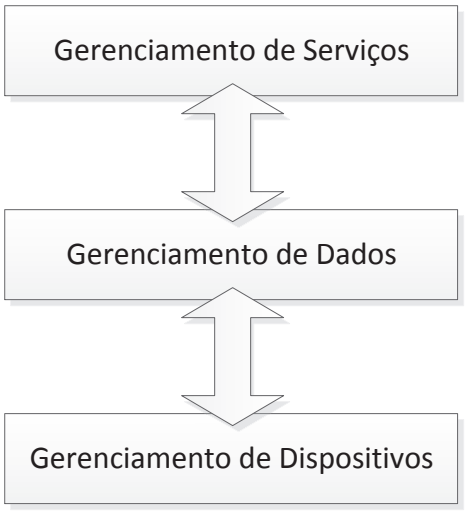
Fonte: Adaptação livre de Jiang et al. (2008)

Para que estes cenários descritos anteriormente possam se tornar realidade, são necessários basicamente três elementos: um identificador único, uma rede de transmissão e uma estrutura computacional para processamento, armazenamento e disponibilização das informações (CHEN; MINTUN, 1996). A associação internacional sem fins lucrativos EPCglobal[3], formada por instituições de mais de 100 países, define a etiqueta RFID de 96 bits como o identificador único padrão para a IoT. O número armazenado nessa etiqueta deve estar no padrão EPC (Electronic Product Code) (GS1, 2014), e possibilita a geração de 2^{96} combinações, o que equivale a aproximadamente 8×10^{28} identificadores únicos. Segundo Li (2006), as implantações de IoT vão gerar grandes quantidades de dados que precisam ser processados e analisados em tempo real. O autor também afirma que a coleta de dados, identificação e processamento das informações requerem soluções sofisticadas de hardware e software.

Devido a heterogeneidade de leitores, protocolos e padrões, é necessário um processo de padronização das informações para que sejam utilizadas de forma simples e adequada. Dessa forma, torna-se necessário um *middleware* para abstrair as particularidades do sistema físico RFID e processar os dados brutos, permitindo mais agilidade e flexibilidade para as aplicações que utilizam essas informações. A EPCglobal também define os padrões de interconexão entre esses elementos de software e hardware, como protocolos de comunicação entre equipamentos e interfaces de troca de dados entre sistemas. Esse padrão é chamado de EPCglobal Framework (EPCGLOBAL, 2014).

Conforme Al-Jaroodi, Aziz e Mohamed (2009), os *middlewares* RFID que seguem a especificação EPCglobal são implementados basicamente em três grandes níveis, conforme mostrado na Figura 3, e descrito a seguir:

Figura 3 – Estrutura típica de um *middleware* RFID



Fonte: Al-Jaroodi, Aziz e Mohamed (2009)

[3]http://www.gs1.org/epcglobal

- **Gerenciamento de dispositivos:** A camada de mais baixo nível é a de gerenciamento de dispositivos. Ela é responsável por gerenciar, monitorar e coletar dados dos leitores físicos implantados no sistema;

- **Gerenciamento de dados:** A camada intermediária é a de gerenciamento de dados, que reúne os dados brutos vindos dos leitores, filtra, limpa e os coloca em um formato utilizável para processamento posterior;

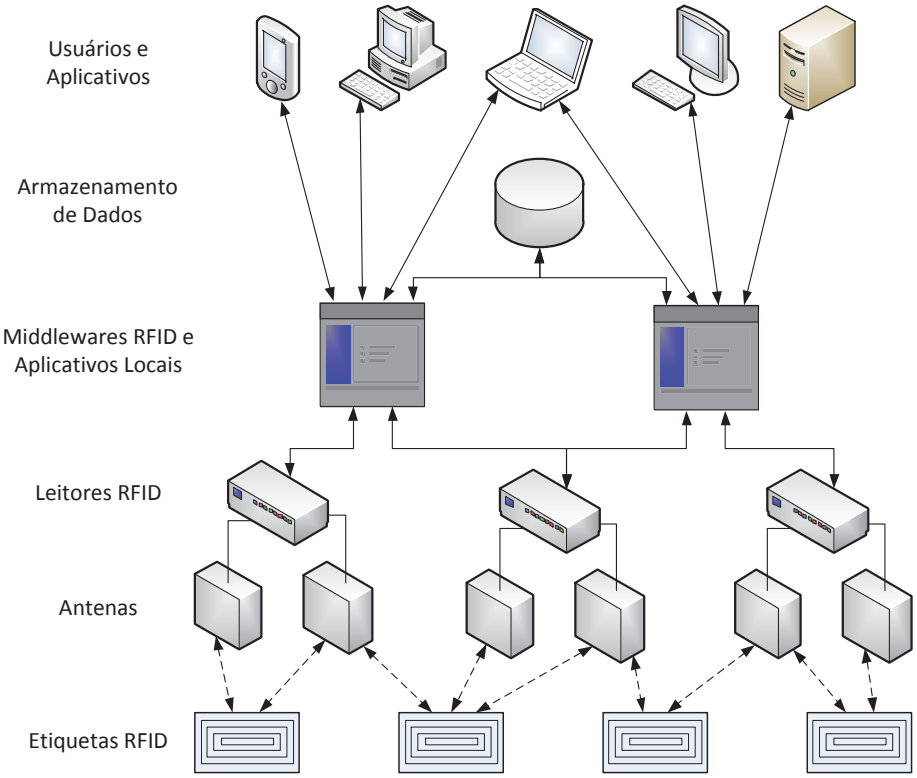
- **Gerenciamento de serviços:** A camada de mais alto nível é a de gerenciamento de serviços, que conecta o sistema RFID aos demais sistemas corporativos. A principal tarefa desta camada é receber consultas dos sistemas informatizados ou de outros *middlewares*, buscar o conteúdo armazenado em um repositório previamente alimentado com dados obtidos dos leitores e devolver uma resposta aos solicitantes.

Resumidamente, o principal objetivo do *middleware* RFID é coletar grandes quantidades de dados brutos provenientes de um ambiente RFID heterogêneo, filtrá-los, compilá-los em formato utilizável e enviá-los aos sistemas informatizados FLOERKEMEIER; LAMPE (2005); KANG; KIM (2011). O *middleware* se torna ainda mais necessário e vantajoso nos casos em que os dados precisam ser compartilhados em vários locais, como em sistemas de cadeia de suprimentos onde muitos leitores estão espalhados em fábricas, armazéns e centros de distribuição separados fisicamente.

No cenário ilustrado pela Figura 4, as etiquetas RFID podem estar coladas nos produtos finais ou até mesmo em peças que irão compor os produtos. Ao passarem pelas áreas de leitura, as antenas identificam a presença das etiquetas e retransmitem o código de identificação único para o leitor no qual estão conectadas. O leitor, por sua vez, transmite os dados coletados para os *middlewares* RFID conectados a ele. Nesse momento, os *middlewares* aplicam as regras de filtragem e agregação previamente definidas, armazenam os registros em banco de dados e notificam em alto nível as aplicações interessadas nas informações. Além da disseminação de dados através de notificação, o *middleware* também atende a requisições pontuais feitas pelas aplicações.

Como visto até aqui, um sistema de Internet das Coisas é bastante extenso, complexo e exige uma grande estrutura computacional, não apenas do ponto de vista da distribuição física dos equipamentos, mas também de conectividade de rede, poder de processamento, escalabilidade e alta disponibilidade. Uma alternativa para suportar a demanda mencionada consiste no emprego de computação em nuvem, ou *cloud computing* (MELL; GRANCE, 2011), para o gerenciamento dos componentes da arquitetura EPCglobal. A nuvem traz consigo a ideia de elasticidade, que aumenta ou diminui o número de máquinas virtuais, nós ou armazenamento em tempo de execução de acordo com a demanda, afirma Righi (2013). Tal ideia vai ao encontro do tráfego crescente e dinâmico de IoT.

Figura 4 – Arquitetura de um sistema RFID empresarial



Fonte: Elaborado pelo autor

1.1 Objetivos

O tratamento eficiente da escalabilidade quanto ao número de aplicações e leitores, com a adição e remoção automática de servidores e/ou componentes, ainda permanece como uma lacuna de pesquisa. Exatamente nessa oportunidade está embasada a ideia do presente trabalho, que aborda o uso da elasticidade em nuvem na área de IoT. A ideia é propor algoritmos para a gerência automática da escalabilidade do *middleware* IoT tirando proveito da elasticidade da nuvem, de modo a atender em tempo de execução demandas que podem ser dinâmicas.

Além das vantagens para o administrador e para os usuários da infraestrutura IoT, dado que não se envolvem com a gerência de recursos, as sugestões abordadas nessa proposta também trazem dois benefícios: (i) o uso de *templates*, fazendo com que a replicação da configuração de IoT seja facilitada, simplificando a difusão e manutenção de sistemas RFID; (ii) melhor aproveitamento de recursos, dado que pode haver consolidação deles no momento de pouca demanda, indo ao encontro das ideias de computação em nuvem.

Nesse âmbito, este trabalho tem como objetivo principal propor uma metodologia para identificar sobrecarga ou sub-utilização de servidores que hospedam um *middleware* IoT baseado na arquitetura EPCglobal, e automaticamente proporcionar elasticidade a essa rede, através de uma arquitetura escalável e de um sistema de gerenciamento e atuação. Para que o objetivo principal seja atingido, a pesquisa contempla os seguintes objetivos secundários:

- Desenvolver um método eficaz de comunicação entre um aplicativo *Gerente de Escalabilidade* e outro aplicativo *Gerente middleware EPCglobal*, bem como a interação entre esse *Gerente middleware EPCglobal* e as máquinas virtuais que executam as requisições;
- Mensurar o desempenho dos componentes do *middleware* EPCglobal quando executados em uma única máquina sem o uso da computação em nuvem. A ideia é verificar os limites de processamento e tempo de resposta em uma demanda combinando interações de aplicações e sensores RFID;
- Determinar quais informações são pertinentes para gerenciar a escalabilidade e os requisitos particulares de funcionamento do *middleware* EPCglobal. Nessa linha, informações de CPU, processamento, latência e largura de banda de rede, frequência de fluxo de dados e disco serão analisados;
- Definir um mecanismo de balanceamento de carga para o *middleware* EPCglobal, para gerir o despacho de requisições para as máquinas virtuais que executarão o trabalho de fato.

1.2 Questão de pesquisa

A partir dos desafios lançados pela nova e crescente área de IoT, a pergunta da pesquisa a ser explorada nesse trabalho é:

Como seria uma arquitetura computacional e algoritmos para gerenciar a escalabilidade de um middleware de Internet das Coisas padrão EPCglobal, a fim de garantir o desempenho vindo da demanda dinâmica de aplicações e sensores RFID?

Subentende-se por "gerenciar a escalabilidade" um mecanismo capaz de medir os recursos computacionais em uso, avaliar se atingiram os limites inferior ou superior de qualidade de serviço preestabelecidos e alocar ou desalocar recursos de forma automática. Para isso, pode-se utilizar uma estrutura de nuvem computacional, servidores *multicore* e técnicas de programação *multithread* e balanceamento de carga.

1.3 Estrutura do texto

A proposta de trabalho está organizada em 8 capítulos, com a seguinte estrutura: Logo após a introdução do assunto no capítulo 1, com a exposição dos objetivos e questões de pesquisa, o capítulo 2 apresenta os conceitos fundamentais e tecnologias necessárias para o entendimento do cenário a ser trabalhado. O capítulo 3 relaciona estudos para o conhecimento do estado da arte sobre IoT e alto desempenho, assuntos fundamentais para a tomada de decisão sobre a proposta de trabalho. O capítulo 4 discute a metodologia criada para avaliação de *middlewares* IoT e sua aplicação em um *middleware* padrão EPCglobal, além de identificar oportunidades a

serem trabalhadas. O capítulo 5 apresenta e discute a arquitetura proposta, definindo os passos a serem seguidos para a realização dos objetivos. A implementação dessa arquitetura proposta é descrita no capítulo 6, apresentando toda a estrutura configurada e *softwares* desenvolvidos. O capítulo 7 apresenta as avaliações realizadas, os resultados obtidos e discute as principais características identificadas. Para finalizar, a conclusão é apresentada no capítulo 8, onde são destacados os principais resultados, a contribuição científica e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos fundamentais para o entendimento da arquitetura proposta no trabalho. Iniciando com a tecnologia RFID e sistemas de identificação automática, segue com uma breve apresentação da Internet das Coisas. Depois, apresenta o conceito de *middleware* RFID, que é a solução proposta atualmente para lidar com a heterogeneidade, processamento, armazenamento e consulta de dados gerados por sensores e etiquetas RFID. Na sequência, são descritos os modelos padronizados e adotados mundialmente por diversos fabricantes e usuários de sistemas de identificação por rádio-frequência na tentativa de oferecer uma rede de serviços e troca de informações referentes a IoT. Para finalizar, são apresentados os conceitos de nuvem computacional, tecnologia que será utilizada como parte da proposta de solução para os problemas de desempenho, escalabilidade e disponibilidade de dados relacionados a Internet das Coisas.

2.1 RFID - Radio-Frequency Identification

Identificação por radiofrequência (RFID) é uma tecnologia capaz de identificar objetos através de ondas de rádio, e foi inicialmente utilizada durante a II Guerra Mundial para identificar aviões inimigos (LANDT, 2005). Embora RFID seja uma tecnologia relativamente antiga, os avanços mais recentes nos processos de fabricação de chips RFID acabaram reduzindo o custo das etiquetas e leitores, o que permite a aplicação em novas áreas como a identificação de produtos a nível de consumidor final. Segundo Want (2006), estes avanços têm o potencial de revolucionar a gestão da cadeia de suprimentos, controle de estoque e logística.

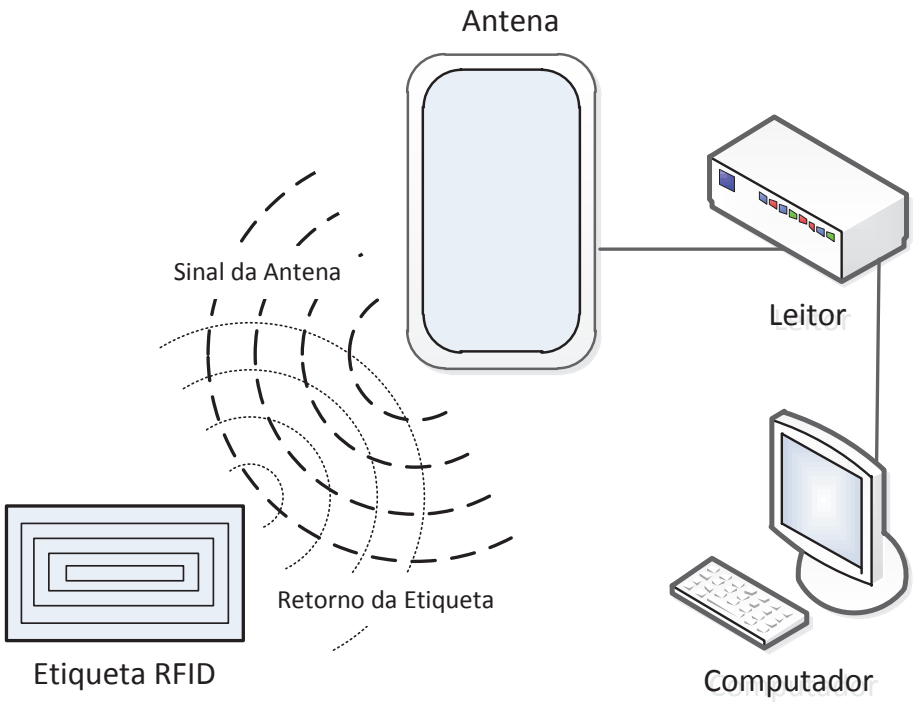
Want (2006) descreve que os sistemas RFID são constituídos basicamente por três elementos:

- **Antena** é o elemento de comunicação, que emite ondas de rádio em uma frequência específica, emitindo "perguntas" solicitadas pelo leitor. A antena também capta as ondas de "resposta" emitidas pela etiqueta e devolve ao leitor;
- **Etiqueta** (também conhecida como *transponder* ou *tag*) é o elemento que recebe a "pergunta" e responde com sua identificação única. As etiquetas podem ser elementos ativos (com bateria interna) ou passivos (sem bateria alguma);
- **Leitor** é o elemento de controle da comunicação entre antena e etiqueta. Ele é responsável por processar as informações relativas à etiqueta e disponibilizá-las para o midleware ou outros aplicativos dedicados.

A Figura 5 ilustra um sistema básico de comunicação RFID. O leitor aciona a antena que emite as ondas de rádio para gerar um campo de leitura. As etiquetas RFID presentes nesse campo são energizadas e respondem com suas identificações únicas. Novamente a antena capta

as respostas emitidas pelas etiquetas e as retransmite para o leitor processá-las e enviá-las ao computador. Esse computador pode executar aplicativos dedicados ou um *middleware* RFID.

Figura 5 – Elementos básicos do sistema RFID



Fonte: Adaptação livre de Want (2006)

Em comparação com a tecnologia de leitura óptica dos códigos de barras, White et al. (2007) enumera algumas vantagens dos sistemas RFID:

- **Leitura dos dados sem necessidade de linha de visão entre leitor e etiqueta:** Como a comunicação entre o leitor e a etiqueta ocorre através de ondas de rádio, essas ondas são capazes de transpor diversos tipos de material. Isso possibilita leituras de etiquetas sobrepostas ou visualmente ocultas, dentro de embalagens ou caixas;

- **Sem necessidade de intervenção humana ou mecânica no processo de leitura:** É possível instalar antenas sobre esteiras de uma linha de produção, em portões de carga e descarga de materiais ou portas de almoxarifados. As antenas captam automaticamente a presença das etiquetas e retransmitem os dados aos leitores, sem a necessidade de comandos disparados por operadores humanos;

- **Paralelismo, permitindo leitura na faixa de centenas de itens por segundo:** Todas as etiquetas presentes no campo de leitura da antena se tornam ativas simultaneamente, podendo transmitir seus dados ao mesmo tempo. Isso possibilita, por exemplo, que uma empilhadeira transportando um *pallet* de produtos identificados com etiquetas RFID passe por um portão monitorado e centenas de etiquetas possam ser detectadas e registradas em questão de segundos.

Existem muitos tipos de sistemas RFID usados em diferentes aplicações e configurações. Estes sistemas têm diferentes fontes de energia, frequências de operação e funcionalidades. As propriedades de operação e as restrições impostas por órgãos de regulamentação irão determinar especificações técnicas e físicas, desempenho e os custos de fabricação e implantação do sistema.

2.2 Sistemas de identificação automática

Em termos de aplicações comerciais, Bose e Pal (2005) descreve que sistemas de RFID podem ser considerados uma instância de uma classe mais ampla de sistemas de identificação automática, conhecidos como "Auto-ID". Sistemas de Auto-ID essencialmente vinculam um nome ou identificador para um objeto físico e os leem automaticamente. Estes identificadores podem ser representados de forma óptica, eletromagnética ou até mesmo química. Os sistemas de Auto-ID mais bem sucedidos e amplamente difundidos comercialmente são o "Número Internacional de Artigo" (EAN) e o "Código Universal de Produtos" (UPC). O EAN/UPC[1] é um código de barras óptico e unidimensional, contendo informações do produto e do fabricante. Etiquetas EAN/UPC podem ser encontradas na maioria dos produtos de consumo em todo o mundo.

Como o código de barras óptico possui apenas informações do fabricante e produto, ele é inviável de ser utilizado em um ambiente que necessite identificação única por objeto, conforme White et al. (2007). Dessa forma, a associação internacional sem fins lucrativos EPCglobal[2] desenvolveu especificações para etiquetas RFID de baixo custo, chamadas *Electronic Product Code* (EPC) (GS1, 2014), como alternativa para substituir o onipresente e limitado EAN/UPC. A ideia do EPC é que cada objeto tenha sua identificação única, e não apenas o código do fabricante e a classe do produto conforme restrições do EAN/UPC.

O código EPC é representado na forma de uma URI (*Uniform Resource Identifier*)[3], contendo as informações do fabricante, classe do produto e número de série. Por exemplo, uma URI EPC pode ser visualizada como "urn:epc:id:sgtin:12345.67890.10479832", e sua estrutura binária pode ser vista na Tabela 1.

Tabela 1 – Estrutura de uma *tag* RFID EPC

	Decimal	Binário	Bits
Cabeçalho	48	00110000	8
Filtro	1	001	3
Partição	5	101	3
Empresa	12345	000000000011000000111001	24
Código Produto	67890	00010000100100110010	20
Número Serial	10479832	00000000000000100111111110100011011000	38

Fonte: Elaborada pelo autor

[1] http://www.gs1us.org/resources/standards/ean-upc
[2] http://www.gs1.org/epcglobal
[3] Definida na RFC 3986, disponível em http://tools.ietf.org/html/rfc3986

2.3 Internet das Coisas

A Internet das Coisas objetiva a criação de um sistema global de registro e informação de bens usando um sistema único de numeração chamado *Electronic Product Code*, ou EPC (GS1, 2014). A IoT é uma revolução tecnológica que impacta no futuro da computação e da comunicação e cujo desenvolvimento depende de pesquisa em campos importantes como os sensores *wireless* e a nanotecnologia, bem como desempenho e armazenamento em data centers. Em termos numéricos, IoT deve gerenciar uma rede de centenas de bilhões de objetos identificáveis e que poderão interoperar uns com os outros e com os data centers.

"O crescimento da Internet das Coisas será muito superior a de outros dispositivos conectados. Em 2020, o número de tablets, smartphones e PCs em uso chegará a cerca de 7,3 bilhões de unidades. Em contraste, a Internet das Coisas vai se expandir a uma taxa muito mais rápida, resultando em uma população de cerca de 26 bilhões de unidades nesse mesmo período", diz Peter Middleton (2013), diretor de pesquisas do Gartner.[4]

Para a viabilização dessa área, a tecnologia sem fio RFID (Identificação por Rádio Frequência) desponta como aquela capaz de capturar os dados EPC de dentro de *tags* usando leitores, que por sua vez alimentam o *middleware* ou sistema IoT para processamento e armazenamento de dados. O uso de IoT é fortemente relacionado com rastreamento de objetos. Em particular, no Brasil destaca-se o emprego de RFID e IoT para o rastreamento de rebanhos de gado através do projeto SISBOV do Ministério da Agricultura[5]. Em adição, o dueto também vem sendo usado em sistemas de controle de acesso a edifícios, aplicações de compra de bilhetes, cobrança de pedágio, controle de ativos e de acesso, compra de objetos no varejo, rastreamento e prevenção à falsificação no setor vestuário, localização de pacientes na área de saúde e cidades inteligentes no consumo racional de água (LI et al., 2009).

Como padronização para o crescente uso de IoT, a organização sem fins lucrativos GS1[6] desenvolveu uma padronização para códigos EPC e uma arquitetura de componentes e interfaces chamada EPCglobal, para uniformizar o uso de IoT na indústria. *Tags* EPC foram projetadas para identificar cada item fabricado, em oposição a apenas o fabricante e classe de produtos, como códigos de barra fazem hoje. Quanto a arquitetura de componentes, são necessários mecanismos para armazenar os dados EPC, filtrar os dados enviados por sensores (o que pode ser uma atividade intensiva quanto a processamento), protocolo uniforme entre diferentes leitores e um sistema de filtragem (KANG; KIM, 2011; JOSE et al., 2013).

[4] http://www.gartner.com/newsroom/id/2636073

[5] O Sistema Brasileiro de Identificação e Certificação de Bovinos e Bubalinos - SISBOV é utilizado para a identificação individual de bovinos e bubalinos em propriedades rurais que têm interesse em vender animais que serão utilizados para produção de carne para atender mercados que exigem identificação individual. http://www.agricultura.gov.br/animal/rastreabilidade/sisbov

[6] http://www.gs1.org

2.4 *Middleware* RFID

Middleware RFID é um software específico para o sistema de identificação por radiofrequência, que intermedeia a comunicação entre os sistemas empresariais e a infraestrutura de hardware do sistema RFID, formada por leitores e etiquetas ou sensores que estão acoplados à esta rede. Conforme Floerkemeier e Lampe (2005), o *middleware* para o sistema RFID coleta, filtra e agrega os dados únicos de identificação das etiquetas que o leitor RFID capturou. Portanto, podemos descrever uma arquitetura básica para o *middleware* como sendo a interface com o leitor, o processamento e gerenciamento de eventos, e a interface de integração com as aplicações que utilizam dados RFID.

Porém, nem todas as implementações RFID necessitam de um *middleware*. Será necessário um *middleware* RFID quando o número de solicitações dos dados das etiquetas for alto, quando o tráfego destas informações for intenso e quando a alta disponibilidade dos serviços que estão interagindo com o sistema RFID for primordial. Além disso, Hoag e Thompson (2006) também informa que o *middleware* pode gerenciar e monitorar a infraestrutura de hardware do sistema RFID.

A implantação de aplicações RFID em conjunto com um *middleware* tende a ser mais rápida, pois ele absorve as diferenças dos diversos leitores integrando os seus dados, construindo soluções RFID mais flexíveis e com escalabilidade. É sabido que quando o sistema RFID é implementado em uma organização, o fluxo de dados aumenta muito. Segundo Hansen e Gillert (2008), o uso de sistemas RFID leva a um enorme aumento na quantidade de dados a serem processados e são impostos requisitos em tempo real aos sistemas. Um *middleware* RFID implantado corretamente trará diversos benefícios, pois ele gerencia de forma centralizada e padronizada a coleta, filtragem, armazenamento e consulta ao dados.

2.5 EPCglobal

EPCglobal é uma *joint venture* entre a GS1 (anteriormente conhecida como *EAN International*) e GS1 EUA (anteriormente *Uniform Code Council, Inc.*). É uma organização criada para promover a padronização e a adoção mundial da tecnologia *Electronic Product Code* (EPC). O foco principal do grupo atualmente é criar tanto um padrão mundial para RFID quanto o uso da Internet para compartilhar dados através da rede EPCglobal. O conselho de administração da EPCglobal inclui representantes da EPCglobal, GS1, Auto-ID Labs[7] além de diversos fabricantes de equipamentos e empresas que utilizam a tecnologia RFID em grande escala na sua cadeia de produção e distribuição.

[7] http://www.autoidlabs.org/

2.5.1 EPCglobal Architecture Framework

A *EPCglobal Architecture Framework* é um conjunto de padrões inter-relacionados de hardware, software e interfaces de dados, juntamente com os serviços centrais que são operados pela EPCglobal e seus delegados, todos a serviço de um objetivo comum: melhorar a cadeia de abastecimento através da utilização de códigos eletrônicos de produtos (EPCs). Segundo a própria EPCglobal, essa padronização tem por objetivo:

- Enumerar, em alto nível, as padronizações de hardware, software e dados que fazem parte do *framework* EPCglobal, e mostrar como eles estão relacionados;
- Definir a arquitetura de alto nível de serviços básicos que são operados pela EPCglobal e seus delegados;
- Explicar os princípios que guiaram o projeto de normas individuais e componentes de serviços básicos dentro da arquitetura EPCglobal;
- Fornecer orientações de arquitetura para os usuários finais e fornecedores de tecnologia que buscam implementar padrões da EPCglobal e utilizar serviços essenciais EPCglobal.

Conforme é possível visualizar na Figura 6, a arquitetura do framework EPCglobal está dividida em 3 camadas: identificação, captura e troca de dados, detalhadas nas próximas seções:

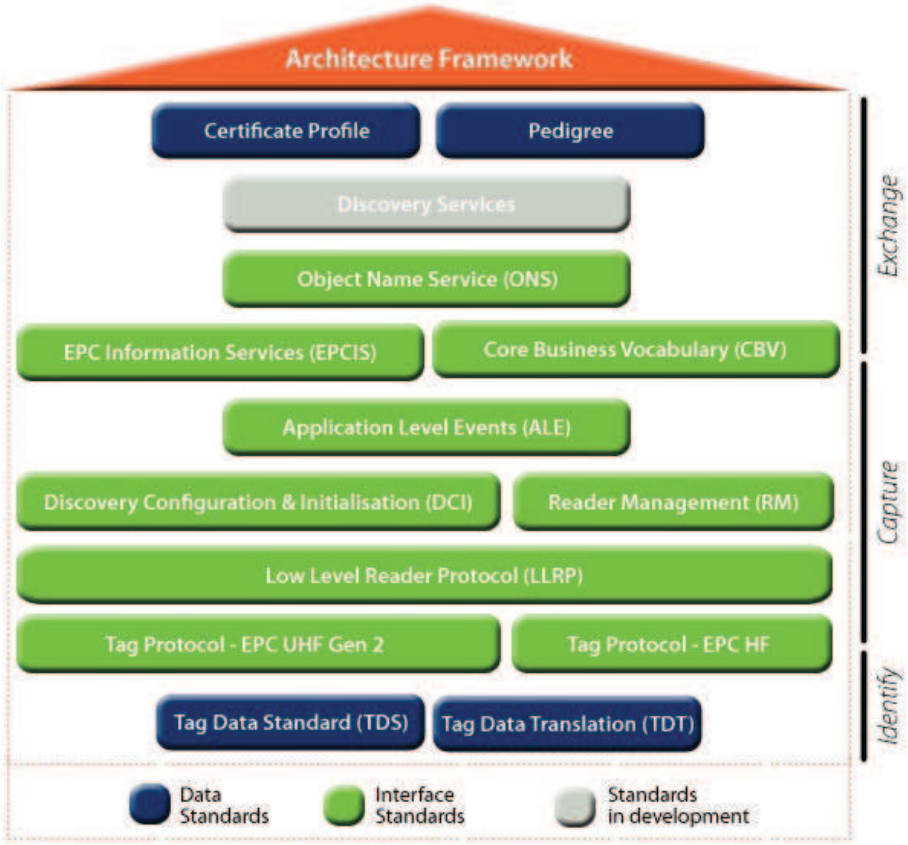
2.5.1.1 Camada de Identificação

Camada responsável pela definição das estruturas de dados e padrões de interfaceamento a nível de etiqueta EPC/RFID, descritas nos seguintes itens:

a) ***EPC Tag Data Standard (TDS)***
Define a estrutura de dados da etiqueta EPC/RFID, incluindo a codificação das chaves GS1 no formato EPC, assim como sua codificação no contexto do EPCIS.

b) ***EPC Tag Data Translation (TDT) Standard***
Define os padrões de interfaceamento para "leitura de máquina" dos dados EPC. A versão legível por máquina pode ser facilmente usada para validar formatos EPC, bem como a tradução entre os diferentes níveis de representação de uma forma consistente. Contém detalhes da estrutura e elementos dos arquivos de marcação, e fornece orientação sobre como ele pode ser usado em software de tradução ou validação automática, seja de forma autônoma ou incorporado em outros sistemas.

Figura 6 – Visão geral da arquitetura EPCglobal



Fonte: EPCglobal (2014)

2.5.1.2 Camada de Captura de Dados

Camada responsável pela definição dos protocolos de comunicação entre leitores e etiquetas RFID, diagnóstico e gerenciamento de leitores e obtenção de dados EPC filtrados e consolidados. Os serviços estão listados a seguir:

a) ***Tag Protocol - EPC UHF Gen 2***
Define os requisitos físicos e lógicos do protocolo de comunicação via rádio entre leitores e etiquetas RFID, operando na faixa de frequência ultra alta (UHF), entre 860 MHz a 960 MHz.

b) ***Tag Protocol - EPC HF***
Define os requisitos físicos e lógicos do protocolo de comunicação via rádio entre leitores e etiquetas RFID, operando na faixa de frequência alta (HF), a 13.56 MHz.

c) ***Low Level Reader Protocol (LLRP)***
Especifica a interface entre leitores RFID e clientes. É classificada como “baixo-nível” pois oferece controle sobre configurações de tempos e parâmetros do protocolo de comunicação via rádio.

d) ***Discovery Configuration & Initialisation (DCI)***
Define uma interface entre leitores RFID, controladores de acesso e a rede na qual estão conectados. Especifica as regras para se comunicar com outros dispositivos, as configurações obrigatórias e opcionais além de inicializar o funcionamento de cada leitor.

e) ***Reader Management (RM)***
Especifica o protocolo utilizado por softwares de gerenciamento para monitorar o estado de operação dos leitores RFID, por exemplo, EPCglobal SNMP RFID.

f) ***Application Level Events (ALE)***
Especifica uma interface através da qual os clientes podem obter os dados EPC de forma filtrada e consolidada, a partir de uma variedade de fontes.

2.5.1.3 Camada de Troca de Dados

Camada responsável pela definição das estruturas de dados e protocolos para troca de informações EPC entre sistemas. Possui serviços de identificação, tradução e localização, além de certificados de procedência. As padronizações estão listadas a seguir:

a) ***EPC Information Services (EPCIS)***
Especificação normativa com o objetivo de permitir que diferentes aplicativos EPCIS compartilhem dados relacionados ao EPC, tanto dentro quanto entre empresas. Em última análise, este compartilhamento tem por objetivo permitir que as empresas obtenham uma visão unificada da disposição dos objetos EPC dentro de um contexto de negócios relevante.

b) ***Core Business Vocabulary (CBV)***
Define vários elementos de vocabulário e seus valores para uso em conjunto com o padrão EPCIS. Isso permite garantir que todas as partes que trocam dados usando o CBV EPCIS terão um entendimento comum sobre o significado semântico dos dados EPC.

c) ***Object Name Service (ONS)***
Especifica como o *Domain Name System* (DNS) é usado para localizar os metadados e serviços associados com a parte SGTIN de um determinado código EPC.

d) ***Discovery Services***
Especificação ainda em desenvolvimento. Quando finalizada, irá permitir encontrar e obter informações relevantes sobre um produto a partir de seu código EPC. O proprietário do produto poderá definir quais informações estarão disponíveis e quem tem direito de acessá-las.

e) ***Certificate Profile***
Define o certificado digital para identificação de leitores RFID.

f) ***Pedigree***

Especifica uma arquitetura para a manutenção e troca de documentos eletrônicos com rastreamento de origem, para uso pelos participantes da cadeia de fornecimento de produtos farmacêuticos.

2.6 Nuvem computacional

O Instituto Nacional de Padrões e Tecnologia do Departamento de Comércio Norte Americano (NIST) define computação em nuvem como "um modelo para permitir acesso via rede de forma ubíqua, conveniente e sob demanda a um agrupamento compartilhado e configurável de recursos computacionais (por exemplo, redes, servidores, equipamentos de armazenamento, aplicações e serviços), que pode ser rapidamente fornecido e liberado com esforços mínimos de gerenciamento ou interação com o provedor de serviços" (MELL; GRANCE, 2011).

A computação em nuvem é um paradigma em evolução. A definição feita pelo NIST caracteriza aspectos importantes da computação em nuvem. Ela se destina a servir como um meio de comparação dos serviços oferecidos e estratégias de implantação, além de fornecer uma linha de base para a discussão do que é a computação em nuvem e qual sua melhor forma de utilização. Os modelos de serviço e implantação definidos formam uma taxonomia simples que não se destina a impor ou restringir qualquer método particular de implantação, prestação de serviços, ou operação do negócio.

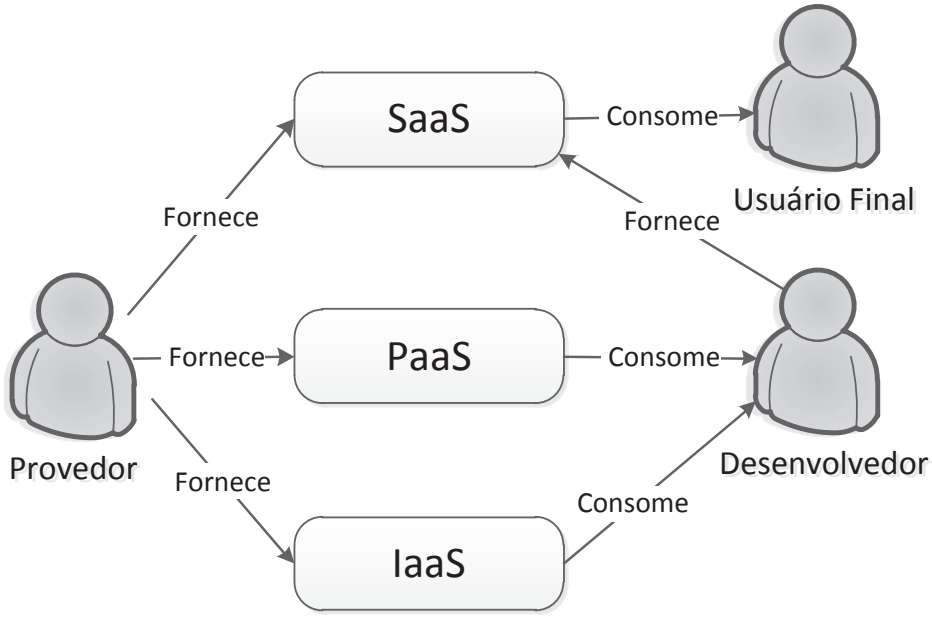
Segundo o NIST, o modelo de computação em nuvem possui cinco características essenciais (MELL; GRANCE, 2011), listadas a seguir:

- **Serviço automático sob demanda**: O consumidor deve ser capaz de alocar novos recursos automaticamente, sem interação humana com o provedor de serviços;
- **Acesso amplo via rede**: Os recursos devem estar disponíveis através da rede e devem ser acessíveis por mecanismos padrão, permitindo seu uso por diferentes dispositivos, tais como computadores pessoais, *smartphones*, *tablets*, etc;
- **Agrupamento de recursos**: Os recursos computacionais do provedor de serviços devem ser agrupados para servir a múltiplos consumidores, com recursos físicos e virtuais sendo arranjados e rearranjados dinamicamente conforme a demanda desses consumidores. Deve haver um senso de independência de localização, no qual o consumidor não tem um controle exato de onde os recursos utilizados estão localizados, mas deve ser possível especificar esse local em alto nível de abstração (país, unidade federativa ou data center);
- **Elasticidade rápida**: Os recursos devem ser alocados e liberados de forma elástica e automática em alguns casos, permitindo a rápida adaptação à demanda. Para o consumidor, os recursos disponíveis devem parecer ser ilimitados, sendo possível alocar a quantidade desejada desses recursos a qualquer momento;

- **Serviços mensurados**: Os serviços de computação em nuvem devem controlar e otimizar os recursos de maneira automática, disponibilizando mecanismos para medir esses recursos utilizando um sistema de medida apropriado para o tipo de recurso sendo utilizado (por exemplo, quantidade de espaço de armazenamento, velocidade de comunicação, capacidade de processamento, número de usuários ativos, etc.). Deve ser possível monitorar, controlar e consultar o uso dos recursos, provendo transparência para o consumidor e para o provedor dos serviços.

O NIST também define que a computação em nuvem deve ser oferecida em três modelos de serviço (MELL; GRANCE, 2011), visualizados na Figura 7 e descritos a seguir:

Figura 7 – Modelos de Serviço de Nuvem Computacional



Fonte: Elaborado pelo autor

SaaS (*Software* como Serviço): Oferece a possibilidade da utilização de aplicativos do provedor rodando em uma infraestrutura de nuvem. As aplicações são acessíveis a partir de vários dispositivos clientes, podendo ser utilizadas através de um navegador web (por exemplo, e-mail baseado na web), ou uma interface de programa compilado. O consumidor não gerencia nem controla a infraestrutura da nuvem, incluindo rede, servidores, sistemas operacionais ou armazenamento. A única liberdade para o usuário é permitir alterar configurações visuais e de usabilidade básica;

PaaS (Plataforma como Serviço): Possibilita o desenvolvimento e distribuição na nuvem de aplicativos utilizando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo provedor. O consumidor não gerencia nem controla a infraestrutura da nuvem, incluindo rede, servidores, sistemas operacionais ou armazenamento, mas tem

controle sobre os aplicativos implementados e definições de configuração para o ambiente de hospedagem de aplicativos;

IaaS (Infraestrutura como Serviço): Fornece recursos computacionais básicos, como processamento, armazenamento e serviços de rede. O consumidor não gerencia nem controla a infraestrutura de nuvem, mas tem controle sobre sistemas operacionais, armazenamento e aplicativos implementados. Em alguns casos é possível ter controle limitado sobre componentes do sistema, por exemplo, *firewalls*.

Os modelos de implantação descritos pelo NIST podem ser classificados quanto às restrições ou permissões de acesso aos serviços oferecidos pela nuvem (MELL; GRANCE, 2011), conforme descrito a seguir:

Nuvem privada: A infraestrutura da nuvem é oferecida para uso exclusivo por uma única organização que agrupa vários consumidores (por exemplo, várias unidades de negócio). Pode ser de propriedade, gerenciada e operada pela própria organização, um terceiro ou alguma combinação deles. Pode existir dentro ou fora das instalações da empresa;

Nuvem comunitária: A infraestrutura de nuvem é oferecida para uso exclusivo por uma comunidade específica de consumidores ou organizações que têm interesses em comum. Pode ser de propriedade, gerenciados e operados por uma ou mais das organizações da comunidade, um terceiro ou alguma combinação deles. Pode existir dentro ou fora das instalações da comunidade;

Nuvem pública: A infraestrutura de nuvem é fornecida para uso aberto ao público em geral. Pode ser de propriedade, gerenciada e operada por uma empresa, instituição acadêmica ou organização governamental ou ainda por alguma combinação entre elas. Fica instalada em um provedor de nuvem;

Nuvem híbrida: A infraestrutura de nuvem é uma composição de duas ou mais infraestruturas de nuvem distintas (privada, comunitária ou pública). São unidas por tecnologias padronizadas ou proprietárias, permitindo portabilidade de dados e aplicações.

3 TRABALHOS RELACIONADOS

Este capítulo relaciona alguns estudos para o conhecimento do estado da arte sobre *middlewares* IoT, suas arquiteturas, funcionalidades, aplicações e restrições. Tem como objetivo traçar um perfil de diversos *middlewares* disponíveis, e principalmente identificar suas características mais importantes, aplicações, tecnologias utilizadas e como gerenciam o balanceamento de carga e escalabilidade.

O critério de escolha foi a existência de trabalhos científicos publicados em congressos e revistas da área, apresentando as características e limitações de cada *middleware*. Esse critério foi adotado tendo como verdadeira a premissa de que se há pesquisadores os estudando, eles são relevantes para a comunidade científica. Foram consideradas tanto soluções compatíveis quanto não compatíveis com o modelo EPCglobal.

3.1 MARM

O *Middleware* RFID baseado em multi-agentes (MARM) (MASSAWE; AGHDASI; KINYUA, 2009) foi projetado utilizando o conceito de Engenharia de Software Orientada a Agentes (AOSE) (WOOLDRIDGEY; CIANCARINI, 2001). A principal motivação para o desenvolvimento do MARM foi elaborar um *middleware* para processamento e gerenciamento de dados gerados por sistemas RFID para gerenciamento de produtos, utilizado pela Universidade Central de Tecnologia (CUT), na África do Sul. MARM utiliza SMURF (limpeza adaptativa para fluxos de dados RFID) (JEFFERY; GAROFALAKIS; FRANKLIN, 2006) para tratar os dados não confiáveis capturados por leitores RFID e um modelo de dados temporal para facilitar o processamento de eventos da aplicação. A arquitetura MARM foi implementada utilizando a metodologia PASSI (CHELLA; COSSENTINO; SABATUCCI, 2004), que é uma das metodologias AOSE para projetar e desenvolver sistemas MAS (Sistemas Multi-Agentes).

A arquitetura é dividida principalmente em três camadas que são: camada de gerenciamento de dispositivo, camada de gerenciamento de dados e camada de interface. Os aplicativos cliente, no nível de interface, especificam os dados que eles desejam receber e o tipo de dados que eles querem ser relatados pela comunicação com os agentes de gerenciamento de dados, como ECReport-Request e ECReportSpec respectivamente. Os agentes de gerenciamento de dados, por sua vez se comunicam com os agentes leitores apropriados na camada do dispositivo, coletam os dados brutos, os processam, geram os relatórios solicitados e os enviam todos de volta para o aplicativo cliente. MARM foi projetado para fornecer não somente as funcionalidades como captura de dados, tratamento e processamento de eventos, mas também o processamento de eventos de nível de aplicação dentro do *middleware*.

3.2 Fosstrak

Fosstrak (FOSSTRAK, 2014; FLOERKEMEIER; RODUNER; LAMPE, 2007) é uma plataforma RFID de código aberto. Sua implementação é baseada no EPCglobal Network Specifications. Fosstrak foi desenvolvido após um extenso trabalho de pesquisa feito na ETH de Zurique. Ele fornece uma infraestrutura de RFID à comunidade RFID, beneficiando os pesquisadores e desenvolvedores de aplicativos com a implementação de mensagens de baixo nível com os componentes da Rede EPC. Ele favorece o uso da rede EPC e disponibiliza componentes de software modificáveis de acordo com as necessidades da aplicação. A plataforma Fosstrak consiste em três módulos: leitor, *middleware* de filtragem e coleta e serviço de informações EPC (EPCIS). O módulo leitor realiza filtragem e disseminação de dados síncrona e assíncrona, conforme especificação da EPCglobal.

O módulo leitor pode ser utilizado em três modos diferentes. Na primeira modalidade, ele encapsula o protocolo proprietário do leitor RFID, e é executado em um servidor separado. No segundo modo, o leitor Fosstrak é usado no modo de simulação, fazendo uso de simuladores de leitores de RFID pertencentes a diferentes fornecedores. Isto pode ser especialmente útil no caso de não se ter nenhum leitor RFID físico disponível. No terceiro modo, o Fosstrak é embutido dentro do leitor de RFID para fornecer filtragem e disseminação de dados.

Uma vez que os leitores capturam os dados das *tags* relevantes, eles notificam o *middleware* de filtragem e coleta do Fosstrak que combina os dados que chegam de diferentes leitores em um único relatório que é enviado para as aplicações subscritas de acordo com um cronograma pré-determinado. A interface entre o *middleware* de filtragem e coleta e um aplicativo *host* é baseado na especificação EPCglobal Application Level Events (ALE) (ALE, 2014). Uma vez que também fornece a funcionalidade de agregação, pode omitir os eventos de leitura redundantes de diferentes leitores originários do mesmo local.

O *EPC Information Services* (EPCIS) (EPCIS, 2014) é o componente responsável por receber os dados RFID do *middleware* de filtragem e coleta, traduzindo-os em eventos de negócios e os tornando disponíveis para a aplicação. É composto de três módulos diferentes: repositório EPCIS, módulo de captura EPCIS que recebe os dados RFID e módulo de consulta EPCIS, que oferece a facilidade de recuperar os eventos gravados no repositório. A atual implementação do Fosstrak não fornece persistência de dados RFID do módulo de filtragem e coleta, e fornece suporte limitado para protocolos proprietários de leitores.

3.3 WinRFID

WinRFID (PRABHU et al., 2006) é um *middleware* RFID projetado e desenvolvido na Universidade da Califórnia, Estados Unidos. Sua arquitetura é baseada em cinco camadas: camada de hardware RFID, camada de protocolo, camada de processamento de dados, estrutura XML, e camada de apresentação de dados. A camada de hardware RFID lida com as questões

relacionadas com a implementação, configuração e comunicação com os leitores e módulo I/O. Existem três componentes desta camada: leitor, *tag* e I/O. A camada de protocolo abstrai os protocolos mais comuns para leitura de *tags*, como a ISO 15693, ICODE, EPC Classe 0 e EPC Classe 1. Possui uma *engine* de protocolos, cuja responsabilidade é analisar e processar os dados brutos como se fosse qualquer um dos protocolos padrão mencionado anteriormente.

A camada de processamento de dados implementa as regras para processar os dados brutos provenientes da camada de protocolo, removendo as leituras duplicadas e conferindo as *tags* lidas. Qualquer anormalidade nessa camada é tratada por meio de diferentes formas de alertas, tais como e-mails, mensagens e gatilhos definidos pelo usuário. Em seguida, os dados limpos e formatados das *tags* são encaminhados para a camada *framework xml*, para serem apresentados em formato xml. Isso facilita que os dados sejam utilizados por uma grande variedade de aplicações empresariais, bem como por sistemas de gestão da cadeia de suprimentos e gestão de estoque.

A camada de apresentação de dados é uma camada de aplicação que utiliza os dados fornecidos a partir da camada framework xml, para visualização e tomada de decisões. WinRFID foi concebido através do framework .NET, que facilita a adição de funcionalidades para as aplicações por meio de plug-ins em tempo de execução. Para aproveitar esse benefício, WinRFID tem um mecanismo de regras adaptável, portanto, os usuários finais podem facilmente adicionar suas próprias regras por meio de plug-ins.

3.4 Hybrid

O Hybrid (CERVANTES et al., 2007) é um *middleware* baseado em comunicação em grupo, em redes peer-to-peer (P2P). Foi puramente concebido para ser um sistema de gestão de estacionamento eletrônico (EPMS), em um campus universitário. O gerenciamento de membros do grupo, como criar, destruir ou alterar os membros do grupo é tratado pelo nó do gerenciador de membros do grupo. O nó coordenador de comunicação do grupo assegura a coordenação afetiva e oportuna entre os nós membros, de modo que qualquer mudança no status do estacionamento seja detectada com atraso mínimo.

O encaminhamento de todos os eventos do estacionamento para o serviço de banco de dados é o trabalho do nó do gerenciador de eventos de estacionamento. Conectores de serviços executam o controle de gerenciamento de nós individuais. O *middleware* Hybrid está atualmente projetado para lidar com dois tipos de eventos de estacionamento: entrada e saída, que ocorrem quando um veículo entra na vaga de estacionamento ou sai da vaga de estacionamento, respectivamente.

3.5 RF^{2}ID

RF^{2}ID (Reliable Framework for Radio Frequency Identification) (AHMED et al., 2007) foi concebido e implementado com o intuito de fornecer um *middleware* com características como confiabilidade, balanceamento de carga, alto rendimento, escalabilidade e organização de dados. A arquitetura do *middleware* é construída sobre o conceito de leitor virtual (VR) e caminho virtual (VP). Cada VR está associado a um conjunto de leitores RFID físicos (PR), em uma região física.

As principais responsabilidades de cada VR são as seguintes: filtrar os dados provenientes do PRs conectados e de outros VRs, *timestamping* de dados, gerenciamento de caminhos virtuais e gerenciamento de consulta. Caminhos virtuais (VPs) são canais lógicos criados dinamicamente por VRs para fornecer funcionalidades como alta confiabilidade de escala dos dados, respostas de consultas eficientes e balanceamento de carga entre os leitores.

3.6 LIT

LIT (Logistics Information Technology) (KABIR et al., 2008) é um *middleware* que aborda tanto as necessidades de aplicação quanto as limitações da tecnologia RFID. Ele foi projetado utilizando os conceitos de Application Level Events (ALE) (ALE, 2014) e EPC Information Services (EPCIS) (EPCIS, 2014) propostos pela EPCglobal. A arquitetura de software ALE do *middleware* é composta por quatro camadas: abstração de aplicativos, execução baseada em estado, consulta contínua e camada de abstração de leitores.

A fim de alcançar um elevado desempenho, foi usado o modelo de execução baseado em estado. A arquitetura de software EPCIS consiste em três camadas: serviço de consulta, repositório e serviço de captura. O componente de acesso aos dados na camada do repositório foi projetado para fornecer persistência e conexões para diferentes bancos de dados, como Oracle, MS SQL Server, MySQL, etc.

3.7 REFiLL

REFiLL (ANAGNOSTOPOULOS; SOLDATOS; MICHALAKOS, 2009) é um *middleware* leve para filtragem e coleta, com base na arquitetura EPCglobal network. Ele foi projetado como um ambiente programável, que oferece flexibilidade para os desenvolvedores de aplicações RFID para plataformas de hardware heterogêneas. REFiLL está posicionado entre a camada de virtualização de leitores e a Application Level Events (ALE) (ALE, 2014).

Sua engine é composta por filtros e gestores de saída. Os filtros são usados para receber dados vindos dos leitores ou outros filtros. Cada filtro tem os seguintes atributos: portas de entrada, portas de saída e processadores de eventos. Os filtros são definidos e interligados com a ajuda de arquivos de marcação baseados em XML editáveis, os quais estão compilados em

código Java para fornecer a funcionalidade de filtragem e coleta dos eventos RFID. Gerenciadores de saída fornecem a funcionalidade de receber os dados processados a partir dos filtros e abstrações para os sistemas de saída, tais como bancos de dados relacionais, arquivos, *Sockets* e *Web Services*.

3.8 Comparativo entre *middlewares* RFID

Após o estudo dos diversos *middlewares* RFID, foi possível identificar as principais características de cada um deles. Nenhum implementa o modelo EPCglobal por completo, porém, o objetivo principal desse estudo foi identificar diferentes formas de lidar com os problemas da escalabilidade e balanceamento de carga. As seções a seguir apresentam uma discussão a respeito das características encontradas nos *middlewares*.

3.8.1 Confiabilidade

Todos os middlewares estudados implementam algum recurso de confiabilidade na leitura, filtragem ou disseminação dos dados. Esses recursos garantem que uma etiqueta RFID que foi detectada no leitor será devidamente processada, filtrada ou agregada e depois disponibilizada para consulta pelas aplicações clientes.

3.8.2 Escalabilidade

No quesito escalabilidade, novamente todos os 7 *middlewares* oferecem algum recurso para instanciar seus módulos mais de uma vez, o que permite aumentar a capacidade de processamento para tarefas específicas. Essa característica é fundamental para que mais a frente seja possível projetar uma solução que ofereça elasticidade ao *middleware*, monitorando os recursos em uso e gerenciando a alocação ou liberação de módulos e a distribuição do processamento de dados entre eles.

3.8.3 Balanceamento de carga

Apenas 4 dos 7 *middlewares* estudados oferecem a possibilidade de realizar balanceamento de carga. O correto funcionamento desse recurso é muito importante para o desempenho geral do sistema, pois enquanto um módulo pode estar trabalhando sobrecarregado, paralelamente outra instância desse mesmo módulo pode estar subutilizado.

3.8.4 Gerenciamento de dados

O gerenciamento de dados é implementado de forma particular por cada um dos *middlewares* estudados. Porém, apenas 2 deles implementam o padrão EPCglobal. Nesse quesito, também foi possível identificar serviços baseados em modelos de dados temporais e dados distribuídos em bancos *peer-to-peer*.

3.8.5 Segurança

A segurança dos dados trafegados entre os diferentes componentes foi a característica menos presente, apenas 3 dos 7 *middlewares* estudados implementam algum tipo de segurança. Isso é uma fragilidade que merece ser melhor resolvida, pois a exposição de dados em redes não seguras pode resultar em vazamento de informações confidenciais.

3.8.6 Avaliação geral

As soluções de *middleware* apresentadas são fortemente concentradas em encontrar maneiras de lidar com o processamento e gerenciamento de dados brutos provenientes de leitores RFID de forma eficaz. Isto é conseguido depois de aplicar as seguintes etapas: remoção de leituras duplicadas, combinação de dados vindos de diversos leitores e disseminação eficiente de dados.

Já que um *middleware* RFID típico lida com grandes quantidades de dados provenientes de vários leitores RFID e os compartilha com as aplicações empresariais, é necessário ter um sistema confiável que garanta um fluxo ininterrupto e correto de dados RFID. Isto é conseguido por meio de diferentes técnicas de gestão de eventos. Com base nas pesquisas realizadas, a Tabela 2 apresenta uma comparação das soluções dos *middlewares* em discussão tendo com parâmetros a confiabilidade, escalabilidade, balanceamento de carga, gerenciamento de dados e segurança.

Uma rede típica RFID pode ter centenas de leitores, enquanto em alguns casos, este número pode aumentar para milhares. Os dados provenientes de todos esses leitores vão se reunir em um determinado ponto e, assim, criar um grande volume de dados. Lidar com tais quantidades de dados brutos pode representar um sério desafio para o desempenho do *middleware*. Portanto, a fim de facilitar este processo, uma técnica de rede distribuída é fortemente recomendada, em que cada nó seja responsável por reunir os dados brutos a partir da sua fonte de dados designada, de acordo com os seus limites de carga pré-definidos.

Lidar com essa grande quantidade de dados também representa ameaças de segurança, tais como processamento inseguro de dados e ameaças à privacidade, por exemplo, utilizando os dados RFID para atividades maliciosas. Esses argumentos justificam claramente a necessidade de se ter um *middleware* com um nível de segurança de protocolo e segurança em nível de mensagem.

Tabela 2 – Comparação entre *middlewares* RFID

***Middleware* RFID**	**Confiabilidade**	**Escalabilidade**	**Balancea-mento de Carga**	**Gerenciamento de Dados**	**Segurança**
MARM	SMURF (Statistical smoothing for unreliable RFID data)	Sistema multi-agentes	Não abordado	Modelo de dados temporal para dados RFID	Não abordado
Fosstrak	Implementação do EPC Reader Protocol	Servidor dedicado, modo simulação e embarcado no leitor RFID	Subscrição de leitores	Especificações da EPCglobal Application Level Events	Assinatura / Publicação
WinRFID	Regras de processamento	Módulos distribuídos	Não abordado	Persistência de dados e mecanismo de regras de negócios customizáveis	Autentica-ção e restrição de acesso
Hybrid	Modelo de comunicação em grupo peer-to-peer	Rede peer-to-peer multi-ring	Sistemas peer-to-peer	Infraestrutura de notificação de eventos descentralizados em grande escala	Subscrição por nível de nodo
RF^2ID	Leitores Virtuais	Caminhos virtuais entre leitores virtuais e leitores físicos	Gerencia-mento de caminhos	Servidor de nomes e servidor de caminhos	Não abordado
LIT	Implementação do EPC Reader Protocol	Interface de gerenciamento de leitores	Modelo de execução baseado em estados	Especificações da EPCglobal Application Level Events	Não abordado
REFiLL	REFiLL Engine e EPC Reader Protocol	Framework programável leve	Não abordado	REFiLL Filters	Não abordado

Fonte: Elaborado pelo autor

Embora a tecnologia RFID já esteja em uso há vários anos, o surgimento de *middlewares* RFID é um evento relativamente recente. Nota-se uma grande variedade de *middlewares* para diversas aplicações, desde propósitos específicos até propósitos genéricos. Não há um consenso nas estruturas, módulos e funcionalidades, embora a EPCglobal sugira um modelo a ser seguido.

3.9 Lacunas de atuação

Com base no estudo realizado, percebe-se a necessidade de um *middleware* mais escalável, robusto, eficiente e seguro. O objetivo desse estudo não é necessariamente propor soluções para todas as limitações identificadas, mas tomar conhecimento de que elas existem para poderem ser trabalhadas no futuro, caso haja interesse ou necessidade. Segue abaixo a relação das lacunas identificadas, e por que é necessário que o *middleware* IoT as atenda:

- **Escalabilidade:** característica necessária para oferecer a possibilidade de expandir ou contrair a estrutura computacional de coleta, processamento, armazenamento e consulta

dos dados de acordo com a demanda. A essa expansão/contração automática de forma transparente ao usuário final se denomina “elasticidade”;

- **Robustez:** o sistema deve ser capaz de se manter operando sem falhas, mesmo que um dos componentes fique indisponível. Deve prever tolerância a falhas de forma automática, sem que o usuário final perceba;
- **Eficiência:** devido à necessidade de processamento de grandes volumes de dados, o sistema deve ser eficiente quanto ao tempo de resposta às requisições e notificações. Além disso, IoT possui diversas aplicações de tempo real. Então, desempenho é peça chave;
- **Segurança:** visto que os dados podem ter origem pessoal ou necessitarem de sigilo, é muito importante que o sistema ofereça processamento, armazenamento e troca de mensagens de forma segura;

4 METODOLOGIA E JUSTIFICATIVA

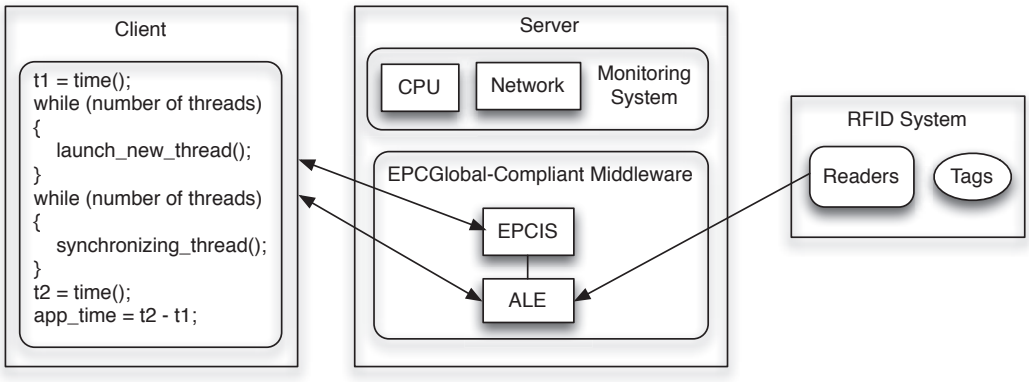
Um ponto chave para o desenvolvimento desse trabalho é identificar as limitações, sobrecarga e subutilização de *middlewares* IoT que implementam o padrão EPCglobal. Quando uma sobrecarga for identificada, deve ser possível alocar mais recursos computacionais para garantir o desempenho do sistema. Quando o sistema estiver ocioso ou subutilizado, deve ser possível desalocar recursos computacionais para economizar energia elétrica, rede e manutenção em geral. Dessa forma, torna-se essencial desenvolver uma metodologia para avaliação do desempenho de *middlewares* IoT. Este capítulo apresenta a metodologia de avaliação, os testes realizados sobre o *middleware* padrão EPCglobal, os resultados obtidos e como essa metodologia será aplicada novamente sobre a arquitetura proposta para comprovar a sua eficiência.

4.1 Metodologia proposta para a pesquisa

Para a avaliação de desempenho e limitações de *middlewares* IoT padrão EPCglobal, foi desenvolvido o *software "MIB: Micro Benchmark para Avaliação de Middlewares de Internet das Coisas"*. A utilização de *benchmarks* vem sendo feita em diversos estudos para levantamento de perfis de desempenho e limitações, conforme Gillam et al. (2013); Gall et al. (2013); Sung e Hsu (2013).

A metodologia proposta consiste em aplicar um *micro benchmark* para obter resultados experimentais de desempenho e uso de recursos computacionais durante a execução do *middleware* escolhido. A ideia é avaliar o comportamento do sistema através da combinação de: (i) avaliação individual dos módulos EPCIS e ALE; (ii) diversas combinações de consultas paralelas e sequenciais; (iii) diferentes situações de carga, variando a quantidade de leitores RFID e *tags* ativas por leitor. Do ponto de vista da arquitetura, a metodologia de avaliação pode ser dividida em 3 módulos, visualizados na Figura 8.

Figura 8 – Metodologia para avaliação de *middlewares* IoT



Fonte: Elaborado pelo autor

O módulo cliente é responsável por registrar o tempo $t1$, disparar as *threads* que realizam as requisições ao servidor, sincronizá-las e obter o tempo decorrido, calculando $t2 - t1$. Os tempos inicial $t1$ e final $t2$ devem ser cuidadosamente capturados na mesma máquina cliente, para evitar problemas de sincronismo de *clock*. O processo de sincronismo das *threads* tem comportamento de barreira bloqueante, aguardando a finalização de todas as requisições para somente após esse momento tomar o tempo $t2$. Cada teste deve ser feito tendo como alvo individualmente o EPCIS ou ALE, nunca os dois módulos simultaneamente. Essa separação tem por objetivo medir cada módulo de forma isolada, pois como estão executando em um mesmo servidor, as requisições processadas por um módulo influenciam no desempenho do outro módulo. Avaliando de forma individual, podemos assegurar que o processo de medição não irá interferir no resultado medido.

O módulo servidor executa um *middleware* padrão EPCglobal, no qual todos os seus componentes devem residir em uma mesma máquina. Além disso, este servidor também executa um sistema de monitoramento que armazena periodicamente dados relacionados com a carga da CPU, utilização de memória e tráfego de rede de entrada e saída. A ideia é observar o comportamento desses recursos durante os acessos aos módulos ALE ou EPCIS.

O ciclo de eventos (*event cycle*) do módulo ALE do *middleware* deve ser definido para enviar os dados coletados das etiquetas RFID no menor período de tempo possível. Isso é necessário para gerar um grande fluxo de dados, visto que uma das métricas da metodologia é avaliar o comportamento do sistema em relação à carga de dados aplicada. Para a metodologia, esse parâmetro da periodicidade do ciclo de eventos deve ser flexível.

Além disso, as consultas ao módulo EPCIS devem ser configuradas para buscar os dados gravados no banco em um intervalo configurável. Deve ser possível definir o intervalo da janela de tempo e o deslocamento em relação ao *clock* do sistema. Essas configurações são necessárias para eliminar ou pelo menos minimizar questões de atraso natural na sequência de processamento dos dados pelo *middleware*, visto que o modelo EPCglobal não especifica aplicações de tempo real.

Finalmente, a metodologia proposta varia os parâmetros de quantidades de *threads*, número de requisições por *thread*, quantidade de leitores RFID e número de *tags* ativas por leitor. Essa variação tem por objetivo encontrar um limite, caso exista, em termos de falhas, grande atraso no retorno da resposta, falta de resposta às solicitações do cliente, perda de pedidos ou má interpretação dos pedidos em relação à carga de dados e solicitações realizadas.

4.2 Proposta de *micro benchmark*

A realização de um *micro benchmark* tem como objetivo traçar um perfil da aplicação para identificar as limitações, sobrecarga ou subutilização do *middleware* IoT, caso existam. Aplicando-se a metodologia descrita anteriormente através do aplicativo *MIB*, espera-se obter respostas para as seguintes questões:

- Existe alguma situação de falha do sistema?
- Qual a relação entre a carga aplicada e o consumo de recursos?
- Qual o comportamento do sistema ao atingir os limites de uso de CPU, rede ou memória?
- É possível identificar *thresholds* de sobrecarga ou subutilização com essa metodologia de avaliação?

A situação de falha pode ser detectada através de uma parada do sistema, resposta mal formada ou retorno de mensagem de exceção vinda do *middleware* ou sistema operacional. A relação entre a carga aplicada e o consumo de recursos pode ser identificada através de gráficos cruzando essas informações coletadas durante os testes. O comportamento ao atingir os limites de uso dos recursos computacionais também pode ser visto através de gráficos, identificando saturações, picos, quedas ou mudanças de inclinação nas curvas de CPU, memória ou rede.

Por fim, a identificação do *threshold* de sobrecarga pode ser feita avaliando a situação dos recursos computacionais em uso no momento em que ocorre um aumento significativo no tempo de resposta ou quando é recebida uma resposta sem dados. A subutilização é um critério mais subjetivo, mas pode ser definido como sendo o instante em que a diminuição da carga não reflete em redução dos recursos computacionais em uso pelo sistema.

4.3 Escolha do *middleware* RFID a ser trabalhado

A escolha do *middleware* RFID a ser trabalhado teve os seguintes critérios:

- Implementação para uso geral: o *middleware* deve atender às mais variadas demandas da IoT, por isso, seu uso deve ser de propósito geral;
- Baseado na EPCglobal: deve seguir os padrões estrutural e de interfaceamento aceitos pelo mercado, para garantir o maior nível possível de utilização por diversas aplicações;
- Disponibilidade de acesso ao código fonte: deve ter distribuição gratuita e código aberto, para o caso de ser necessário verificar questões de implementação a nível de programação;
- Possibilidade de implantação modular, de forma distribuída: como a proposta desse trabalho envolve diretamente nuvem computacional, é essencial que o *middleware* suporte de forma nativa a instalação em ambientes distribuídos;
- Utilização por pesquisadores na área acadêmica: para que seja possível tomar ciência de trabalhos sendo realizados por outros pesquisadores com objetivos semelhantes e para compartilhar os resultados atingidos no estudo desse trabalho;
- Utilização comercial: a utilização pelo mercado, tanto na indústria quanto no comércio dão credibilidade ao *middleware*, pois o coloca à prova em ambientes de uso real.

Entre todos os *middlewares* avaliados, o Fosstrak foi o escolhido pois atende a todos os critérios. Porém, a metodologia de avaliação e a proposta de solução deve ter como alvo qualquer *middleware* que implemente o padrão EPCglobal, não apenas o Fosstrak.

4.4 Configuração para avaliação

Uma vez escolhido o *middleware* para aplicação da metodologia desenvolvida, foi definido o ambiente de avaliação. O ambiente foi composto por 3 computadores interligados por uma rede *Fast Ethernet*.

O módulo servidor consiste em um único computador Intel Core 2 Duo 2.1 GHz 4GB RAM com Windows 7 32 bits executando o *midleware* Fosstrak, o gerenciador de banco de dados MySQL para armazenamento dos identificadores EPC e um monitor de recursos. O monitor de recursos identifica a carga de CPU, utilização de memória e tráfego de rede a cada 1 segundo, e salva esses dados em um arquivo CSV localmente.

O ciclo de eventos (*event cycle*) do módulo ALE foi configurado para enviar os dados coletados das etiquetas RFID a cada 2 segundos. Esse período foi definido como sendo o mínimo possível após testes com o Fosstrak, no qual o processamento dos dados se tornou estável e confiável. Nos testes, intervalos mais curtos geravam perda de pacotes.

Para representar o sistema RFID, foi utilizado o emulador Rifidi[1] rodando em uma máquina Intel CoreDuo 2.16 GHz com 2GB RAM e Windows XP 32 bits. É necessário que o sistema RFID seja executado em uma máquina específica para que os recursos consumidos pelo processo de geração dos dados das etiquetas RFID não interfiram no desempenho do servidor nem sobrecarreguem ou tendenciem os indicadores de CPU e tráfego de rede. O emulador foi configurado para executar 3 cenários de carga de dados, descritos a seguir:

- 4x0: Rifidi executando 4 leitores mas sem nenhuma etiqueta ativa, resultando assim em uma carga de dados nula;
- 4x1: Rifidi executando 4 leitores com apenas 1 etiqueta RFID ativa por leitor, enviando os dados de 4 etiquetas por ciclo;
- 4x4: Rifidi executando 4 leitores com 4 etiquetas RFID ativas por leitor, gerando dados de 16 etiquetas por ciclo.

O módulo cliente utiliza uma máquina Intel Core i3 2.4 GHz, com 4GB RAM e Windows 7 64 bits. As consultas ao módulo EPCIS foram configuradas para buscar os dados gravados no banco no intervalo [instante atual - 10 segundos, instante atual - 5 segundos]. Por exemplo, uma requisição de consulta de dados realizada às 10h 20min e 30seg irá solicitar ao EPCIS os registros gravados entre 10h 20min e 20seg e 10h 20min e 25seg.

[1] http://www.rifidi.org

Considerando o número de *threads*, planeja-se usar desde uma única *thread* até 512, em um intervalo crescente regido por potência de 2. Assim, serão criadas 10 configurações, partindo de 2^0 até 2^9 requisições paralelas para avaliar os módulos EPCIS e ALE. Além disso, cada *thread* realizará 1, 2, 4, 8 ou 16 requisições seriais para cada um dos módulos do *middleware*. Cada combinação "*threads* x requisições" deve ser executada 5 vezes diante dos 3 cenários de carga de dados descritos anteriormente para aumentar a confiabilidade das medições. No final dos testes, será feita uma média aritmética e desvio padrão das medições de cada conjunto de parâmetros avaliado, para se ter um índice de qualidade dos valores obtidos.

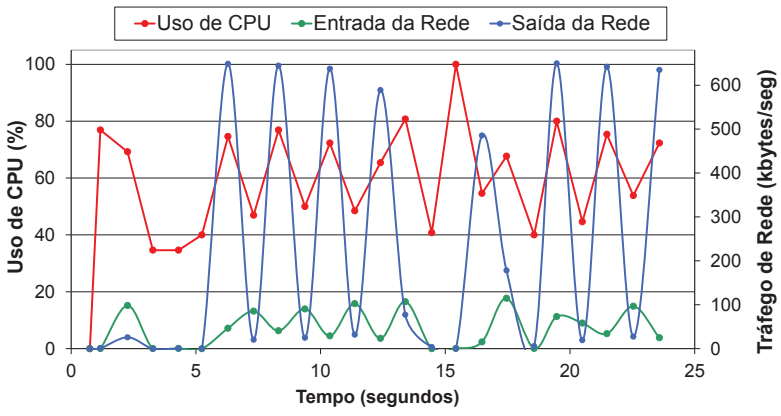
4.5 Aplicação do *micro benchmark* no *middleware* Fosstrak

Após a aplicação do *micro benchmark MIB* no Fosstrak com a metodologia e cenários descritos anteriormente, foi feito o agrupamento e a consolidação dos dados obtidos. Com base nessas informações, foi possível levantar o perfil de comportamento de cada um dos módulos do *middleware* e identificar alguns pontos críticos que precisaram ser melhor estudados através da avaliação dos dados pontuais de cada teste. Os resultados são apresentados a seguir.

4.5.1 Identificação dos comportamentos padrão

Como o módulo ALE é regido pelas configurações do *event cycle*, que foi definido para ocorrer a cada 2 segundos, é possível visualizar na Figura 9 um comportamento cíclico tanto para uso de CPU quanto para tráfego de rede. Claramente, é possível observar um período de cerca de 2 segundos entre quaisquer atividades no ALE, em que a fase de processamento ocorre no final do ciclo. Tanto o uso da CPU quanto o tráfego de rede de saída estão sincronizados, enquanto os dados de entrada estão deslocados em 1 segundo. Este comportamento foi capturado ao aplicar a seguinte configuração: 128 *threads*, cada uma solicitando 8 consultas sequenciais e 16 etiquetas RFID ativadas no emulador Rifidi. Porém, esse comportamento cíclico foi observado em todos os cenários testados.

Figura 9 – Comportamento padrão do módulo ALE

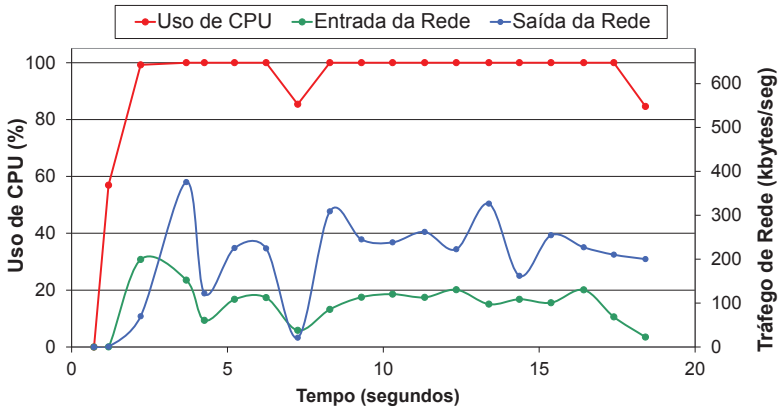


Fonte: Elaborado pelo autor

Os testes para o componente EPCIS consistem basicamente em realizar uma requisição em um banco de dados relacional através de uma consulta SOAP parametrizada. Ao contrário do ALE, o EPCIS não é regido por ciclos, mas por uso de CPU e comunicação via rede. Considerando-se que o acesso ao EPCIS é em suma um acesso a uma base de dados, não há impedimentos para que a resposta seja a mais rápida possível. Assim, são esperados níveis mais elevados e constantes de carga da CPU e rede.

A Figura 10 ilustra o comportamento padrão do módulo EPCIS. O uso da CPU sobe rapidamente e se mantém estável em torno de 100%, enquanto os tráfegos de rede permanecem em oscilação. O tráfego de rede de entrada é menor do que o de saída, porque as requisições são menores do que as respostas. A Figura 10 foi capturada ao testar o mesmo cenário descrito anteriormente, executando 128 *threads*, 8 consultas sequenciais e 16 etiquetas RFID ativas. Porém, comportamento semelhante foi observado em todos os cenários testados.

Figura 10 – Comportamento padrão do módulo EPCIS



Fonte: Elaborado pelo autor

4.5.2 Análise de tempo de resposta e carga de CPU

As Tabelas 3, 4, 5 e 6 a seguir apresentam os resultados em relação ao tempo de resposta e uso da CPU para os módulos ALE e EPCIS separadamente. Em relação aos tempos de resposta do módulo ALE, podemos destacar a primeira e a última coluna da Tabela 3. Nelas é possível verificar que o tempo médio de resposta por requisição ao ALE é substancialmente igual, tanto com carga mínima quanto com carga máxima. A configuração com uma requisição e nenhuma *tag* RFID obteve um tempo médio de 2,84 segundos, enquanto foram mensurados 2,66 segundos testando com 16 requisições e 16 etiquetas.

Aplicando-se a mesma metodologia de avaliação, a Tabela 4 ilustra o uso de CPU destacando os maiores valores por teste, o que aconteceu 93,33% dos casos quando eram executadas 200 *threads*. Em uma visão geral, é possível analisar as seguintes diferenças entre as Tabelas 3 e 4: ao contrário do tempo de resposta, a carga de CPU cresce diretamente proporcional à quantidade de *tags* utilizadas. Além disso, observa-se que o tempo de resposta cresce em uma taxa menor que o crescimento do uso de CPU. Por exemplo, temos um incremento de 9,65% do tempo quando se passa de 1 para 200 *threads* em um cenário com 16 etiquetas. No entanto, a mesma configuração implica no aumento de 32,34% de carga na CPU.

Tabela 3 – ALE - Tempo de resposta (segundos)

Número de Threads	1 Requisição			2 Requisições			4 Requisições			8 Requisições			16 Requisições		
	0 tag	4 tags	16 tags	0 tag	4 tags	16 tags	0 tag	4 tags	16 tags	0 tag	4 tags	16 tags	0 tag	4 tags	16 tags
1 thread	3.01	2.45	3.25	5.22	5.28	4.86	10.51	10.84	10.25	20.22	20.48	18.83	41.57	40.47	38.33
2 threads	2.15	2.36	2.68	4.22	5.29	5.76	9.12	10.33	11.01	19.68	21.08	21.11	37.60	41.43	41.30
4 threads	3.62	4.01	3.34	6.32	5.76	5.21	10.65	10.14	10.57	20.22	20.26	20.49	39.02	40.32	41.01
8 threads	2.06	2.75	2.72	5.22	5.38	5.06	11.71	11.99	11.61	20.54	21.92	25.36	43.71	41.97	42.53
16 threads	3.50	2.94	3.12	4.72	6.30	6.23	11.17	11.49	11.48	21.77	21.18	21.48	37.30	39.81	43.53
32 threads	2.32	3.17	3.05	5.61	6.46	5.71	10.15	10.81	10.43	19.79	21.57	21.53	42.48	39.14	41.35
64 threads	2.79	3.32	4.15	6.19	6.69	6.72	12.10	11.28	11.37	21.66	22.92	21.53	35.80	40.60	47.95
128 threads	3.09	4.03	4.15	6.49	7.17	8.27	11.43	11.37	14.71	19.79	19.85	24.01	40.79	44.61	44.30
200 threads	3.03	2.92	4.41	5.45	4.95	5.29	10.23	12.26	12.71	21.24	24.18	22.71	41.93	45.71	42.03

Tabela 4 – ALE - Uso de CPU (%)

Número de Threads	1 Requisição			2 Requisições			4 Requisições			8 Requisições			16 Requisições		
	0 tag	4 tags	16 tags	0 tag	4 tags	16 tags	0 tag	4 tags	16 tags	0 tag	4 tags	16 tags	0 tag	4 tags	16 tags
1 thread	23.57	26.15	40.84	19.43	30.54	39.83	21.10	35.11	47.47	23.31	39.58	46.39	22.93	34.48	48.32
2 threads	16.10	27.35	37.86	20.67	29.18	39.21	21.57	32.57	45.69	20.42	34.10	48.21	23.66	34.58	47.61
4 threads	17.81	29.36	52.97	20.61	36.85	38.90	19.56	42.59	46.18	23.03	35.79	46.14	23.29	35.09	48.49
8 threads	17.46	26.18	38.53	19.26	29.44	38.53	22.61	33.52	43.93	23.69	34.78	47.97	23.51	34.87	48.74
16 threads	18.81	27.60	39.61	19.08	29.96	42.58	25.23	32.41	46.51	22.12	33.08	46.72	22.10	35.40	49.52
32 threads	17.09	27.36	32.06	21.87	34.47	41.35	24.02	35.78	50.67	22.83	36.59	49.43	24.60	37.63	49.80
64 threads	21.58	27.11	42.45	23.87	33.34	46.21	26.38	38.69	51.02	29.60	38.89	52.18	28.10	39.78	51.37
128 threads	26.04	38.04	48.09	26.84	39.47	48.86	26.89	40.70	53.08	29.89	43.74	55.73	29.34	42.26	58.15
200 threads	32.08	42.08	48.70	29.53	45.67	57.69	32.96	44.52	59.19	33.58	46.03	65.21	32.96	46.69	63.95

Tabela 5 – EPCIS - Tempo de resposta (segundos)

Número de Threads	1 Requisição			2 Requisições			4 Requisições			8 Requisições			16 Requisições		
	0 tag	4 tags	16 tags	0 tag	4 tags	16 tags	0 tag	4 tags	16 tags	0 tag	4 tags	16 tags	0 tag	4 tags	16 tags
1 thread	1.00	1.01	1.35	1.51	1.51	1.51	2.51	2.61	2.51	4.51	4.63	4.62	8.53	8.75	8.63
2 threads	1.20	1.01	1.01	1.51	1.64	1.51	2.51	2.51	2.52	4.52	4.64	4.63	8.55	8.55	8.57
4 threads	1.01	1.01	1.01	1.51	1.51	1.63	2.54	2.53	2.53	4.54	4.74	4.64	8.56	8.86	8.76
8 threads	1.01	1.01	1.02	1.53	1.52	1.52	2.54	2.65	2.91	4.55	4.79	4.86	8.63	8.80	8.91
16 threads	1.03	1.03	1.02	1.56	1.54	1.66	2.56	2.91	3.12	4.61	5.33	5.27	8.71	9.56	9.56
32 threads	1.07	1.22	1.49	1.61	2.16	2.56	2.88	3.77	4.01	4.84	6.77	7.07	9.28	12.48	13.92
64 threads	1.38	1.90	1.78	1.99	2.80	3.08	3.81	4.57	5.26	6.58	8.76	9.18	12.71	16.79	18.04
128 threads	1.77	2.92	2.71	3.08	4.40	5.50	4.91	8.17	8.88	9.25	15.43	17.51	18.03	28.45	34.53
256 threads	2.73	4.06	5.85	4.87	7.88	9.28	9.04	14.97	17.17	18.02	29.61	32.65	33.75	52.57	58.95
512 threads	5.02	7.87	9.97	9.13	15.37	17.25	17.62	28.59	30.95	34.79	48.68	53.36	68.44	88.76	100.66

Tabela 6 – EPCIS - Uso de CPU (%)

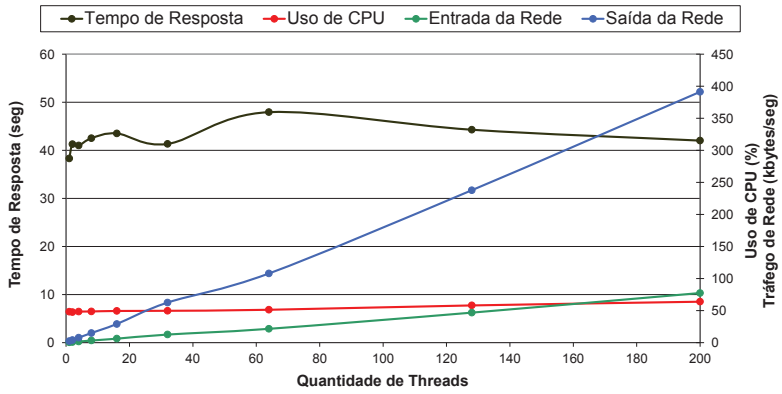
Número de Threads	1 Requisição			2 Requisições			4 Requisições			8 Requisições			16 Requisições		
	0 tag	4 tags	16 tags	0 tag	4 tags	16 tags	0 tag	4 tags	16 tags	0 tag	4 tags	16 tags	0 tag	4 tags	16 tags
1 thread	13.65	48.00	66.94	15.15	22.76	53.56	17.85	21.38	41.81	18.89	31.51	43.74	21.42	32.32	45.88
2 threads	14.36	26.15	37.35	20.92	28.94	23.73	22.64	29.04	46.76	22.65	34.56	49.76	19.62	34.55	53.89
4 threads	18.03	21.46	40.58	20.62	25.46	42.90	19.33	26.92	38.91	27.05	34.88	48.79	23.81	37.39	55.00
8 threads	18.08	25.04	31.05	19.00	24.31	34.38	22.10	35.17	47.69	26.19	41.90	53.68	27.85	43.88	59.70
16 threads	22.54	18.81	30.14	21.75	34.56	41.12	22.01	41.88	52.74	30.46	51.00	61.97	37.19	58.77	67.40
32 threads	18.82	32.87	38.08	30.00	45.15	47.66	35.75	51.55	64.46	42.77	60.80	68.07	50.91	67.43	75.41
64 threads	16.36	46.77	64.84	35.38	57.26	63.25	48.70	67.47	76.69	60.47	80.98	83.87	66.28	85.08	88.67
128 threads	37.91	54.78	55.19	51.78	66.89	75.23	68.78	80.82	84.31	77.84	89.40	90.32	84.24	93.02	93.90
256 threads	59.42	66.48	81.95	70.08	81.45	84.41	81.15	89.75	92.87	89.28	93.72	94.14	93.89	95.72	96.50
512 threads	69.58	86.29	91.34	81.63	89.40	92.03	89.20	93.01	94.55	93.99	96.57	96.98	96.32	97.65	97.73

Da mesma forma que foram analisados os dados do ALE, também foi possível destacar algumas células nas tabelas do EPCIS. As células marcadas na Tabela 5 enfatizam o momento em que ocorre um súbito aumento no tempo de resposta, o que pode indicar uma sobrecarga do sistema. No entanto, uma análise da Tabela 5 permite concluir que não há problemas de escalabilidade quando se considera a métrica do tempo. Em outras palavras, o incremento no número de *threads* não implica em um desempenho pior por requisição. Também foram marcadas na Tabela 6 todas as células com o uso de CPU superior a 80%. Este índice, por exemplo, poderia ser usado como um *threshold* para a seleção dos candidatos ao balanceamento de carga. Além disso, observa-se uma sinergia entre as células marcadas nas Tabelas 5 e 6. Isso indica que ocorre um aumento considerável no tempo de resposta quando o uso de CPU ultrapassa 80%.

4.5.3 Resultados e discussões

Antes de executar os testes com o módulo ALE, havia uma expectativa de que o uso da CPU, tráfego de rede e tempo de resposta deveriam aumentar à medida que o número de dados a serem processados crescesse também. Analisando os dados do ALE com 16 requisições e 16 etiquetas, isso realmente acontece com o uso da CPU e tráfego de rede em uma taxa linear, como ilustrado na Figura 11. No entanto, o módulo ALE tende a apresentar um tempo de resposta constante, com uma pequena variação, enquanto o número de *threads* está aumentando.

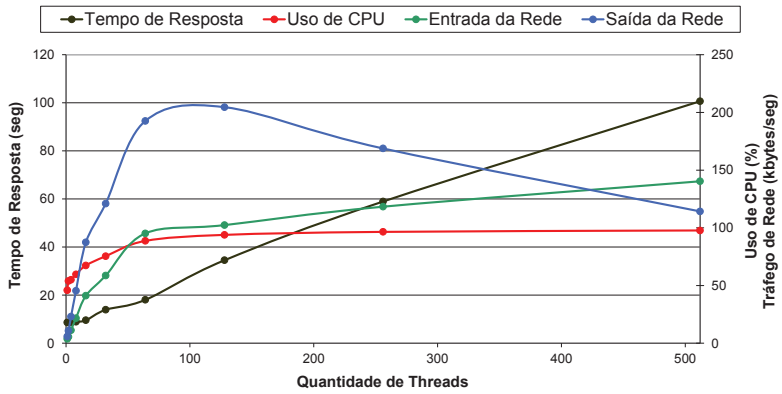
Figura 11 – Comportamento médio do módulo ALE



Fonte: Elaborado pelo autor

O componente EPCIS, por sua vez, tem um comportamento diferente quando comparado ao ALE. Conforme mostrado na Figura 12, o uso de CPU, tráfego de rede e tempo de resposta aumentam quando a quantidade de *threads* e o número de requisições crescem também. Perto de 90% da carga de CPU, observa-se que a inclinação das curvas de CPU e tráfego de rede de entrada não é tão acentuada. Particularmente, o desenho da curva de CPU apresenta o mesmo comportamento clássico de um gráfico de transferência de rede, onde os valores crescem rapidamente e se mantêm próximos do limite da tecnologia. Além disso, o tráfego de rede de saída começa a diminuir de repente, quando o uso da CPU ultrapassa o valor de 95%.

Figura 12 – Comportamento médio do módulo EPCIS



Fonte: Elaborado pelo autor

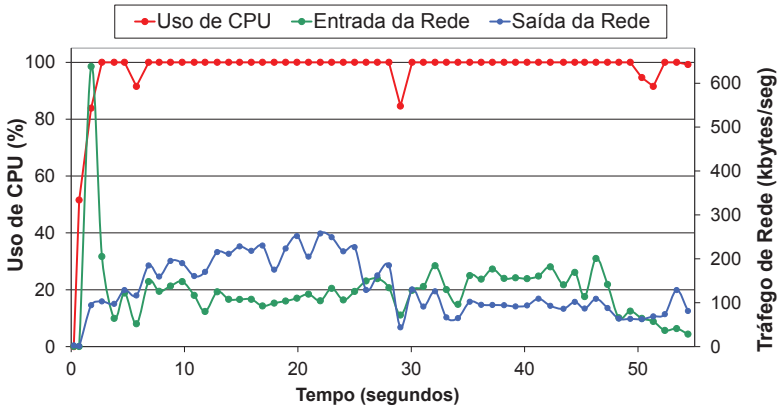
4.5.4 Gargalos e limitações

O módulo ALE apresentou uma limitação na qual era possível processar até um limite de 200 requisições simultaneamente. Caso o módulo cliente da ferramenta de testes executasse 201 *threads* ou mais solicitando consultas SOAP ao mesmo tempo, ocorria falha de *timeout* e o módulo ALE parava de funcionar. Após essa falha, até mesmo requisições para o módulo EPCIS não respondiam mais, sendo necessário reiniciar o Tomcat (ambiente de execução do *middleware*) para fornecer um sistema estável novamente. Analisando o tempo de resposta e uso da CPU na Figura 11, não é possível visualizar uma tendência de saturação que justifique a limitação de 200 requisições simultâneas. Dessa forma, essa limitação parece ser um problema de implementação do *middleware* ou restrição do sistema operacional.

A Figura 13 ilustra um caso de teste com 512 *threads*, 8 requisições por *thread* e 16 *tags* RFID ativas em que é possível explicar o comportamento médio descrito anteriormente na Figura 12. Em primeiro lugar, a Figura 13 apresenta um pico no tráfego de rede de entrada por causa do acúmulo de pedidos feitos por 512 *threads* ao mesmo tempo, e uma forte diminuição em seguida. O uso da CPU atinge cerca de 100% em poucos segundos, mantendo-se próximo a este nível até o final do teste. De 4 a 26 segundos, o tráfego de rede de saída é maior do que o de entrada, indicando que os identificadores das etiquetas RFID enviados pelo emulador Rifidi estavam sendo capturados corretamente pelo módulo ALE, salvos no MySQL e retornados como resposta às consultas ao EPCIS.

No entanto, aproximadamente 26 segundos após o início da execução do teste, o tráfego de rede de saída começa a diminuir, mantendo os valores abaixo do tráfego de entrada. Enquanto isso, o tráfego de entrada mantém um valor flutuante médio normal. Ao analisar o conteúdo da resposta às consultas, observou-se um conjunto de dados vazio, no qual não retornava nenhum dado das *tags* RFID. Isto pode ser explicado analisando-se o uso de CPU, que mostrou-se esgotado por um longo período de tempo. Essa exaustão fez com que os dados enviados pelo Rifidi não pudessem ser processados em tempo hábil, resultando em uma diferença maior que 10 segundos entre a requisição e a resposta.

Figura 13 – Comportamento do módulo EPCIS em situação de sobrecarga



Fonte: Elaborado pelo autor

4.6 Justificativa da proposta

Embora os resultados obtidos na aplicação do *micro benchmark MIB* sobre o Fosstrak sejam confiáveis apenas para as configurações e cenários utilizados nos testes, assume-se como sendo válidos para traçar o perfil de comportamento padrão para *middlewares* baseados na arquitetura EPCglobal.

Todas as questões levantadas na proposta de avaliação foram respondidas. Foi possível identificar os padrões de comportamento de cada módulo, como respondem a diferentes cenários de carga e o comportamento após os limites dos recursos computacionais terem sido alcançados. Foi possível identificar os *thresholds* de sobrecarga e subutilização com essa metodologia de avaliação.

Tendo em vista a pesquisa realizada nessa etapa, juntamente com a identificação de limitações do *middleware* avaliado, identifica-se a oportunidade de propor modelos alternativos para a infraestrutura computacional da Internet das Coisas.

Considerando o crescimento previsto para a quantidade de usuários, número de objetos rastreados e volume de dados armazenados, a possibilidade da tecnologia atual não conseguir atender as demandas é bastante grande. Dessa forma, é questão primordial desenvolver novas técnicas de gerenciamento desses serviços, principalmente em relação à escalabilidade.

5 ARQUITETURA PROPOSTA

A arquitetura proposta tem por objetivo principal oferecer escalabilidade e elasticidade ao *middleware* IoT de forma transparente do ponto de vista do usuário final. Consequentemente, deve-se obter melhores resultados de desempenho e confiabilidade do sistema em relação à tradicional arquitetura não distribuída nem escalável, visto que agora o processamento e armazenamento das informações passaria a ser realizado em máquinas distribuídas.

A solução deve ser implementada sobre um *middleware* IoT padrão EPCglobal, executado em uma estrutura de nuvem computacional. A estrutura de nuvem é fortemente recomendada devido as suas características intrínsecas de elasticidade e ausência de necessidade de manutenção por parte do usuário final. A decisão sobre alocar, realocar ou desalocar recursos deve ser tomada após análise de indicadores reais de desempenho e recursos computacionais em uso pelo sistema.

5.1 Decisões de projeto

A arquitetura proposta envolve decisões tomadas para resolver problemas e limitações identificados durante a pesquisa e experimentos realizados no decorrer do trabalho. Para este propósito, algumas premissas foram elaboradas a fim de auxiliar no desenvolvimento da arquitetura e validação funcional da proposta. Os dois primeiros itens a seguir representam alterações na arquitetura, enquanto os dois últimos itens são recomendações.

a) **Divisão do módulo EPCIS para atender demandas distintas:** o módulo EPCIS atende tanto demandas internas da camada ALE quanto externas vindas de consultas solicitadas por usuários e aplicações. As requisições vindas do ALE são para gravação de dados capturados, enquanto as requisições externas são puramente para realizar consultas a dados armazenados no banco. A divisão do EPCIS de acordo com a natureza da operação realizada é essencial para gerenciar a escalabilidade do sistema;

b) **Escalabilidade e elasticidade do EPCIS:** a demanda de consultas ao EPCIS vinda de usuários e aplicações é dinâmica e variada. Por isso, é necessário que esse módulo seja escalável para atender o aumento de demanda e elástico para liberar recursos alocados anteriormente e que estejam subutilizados;

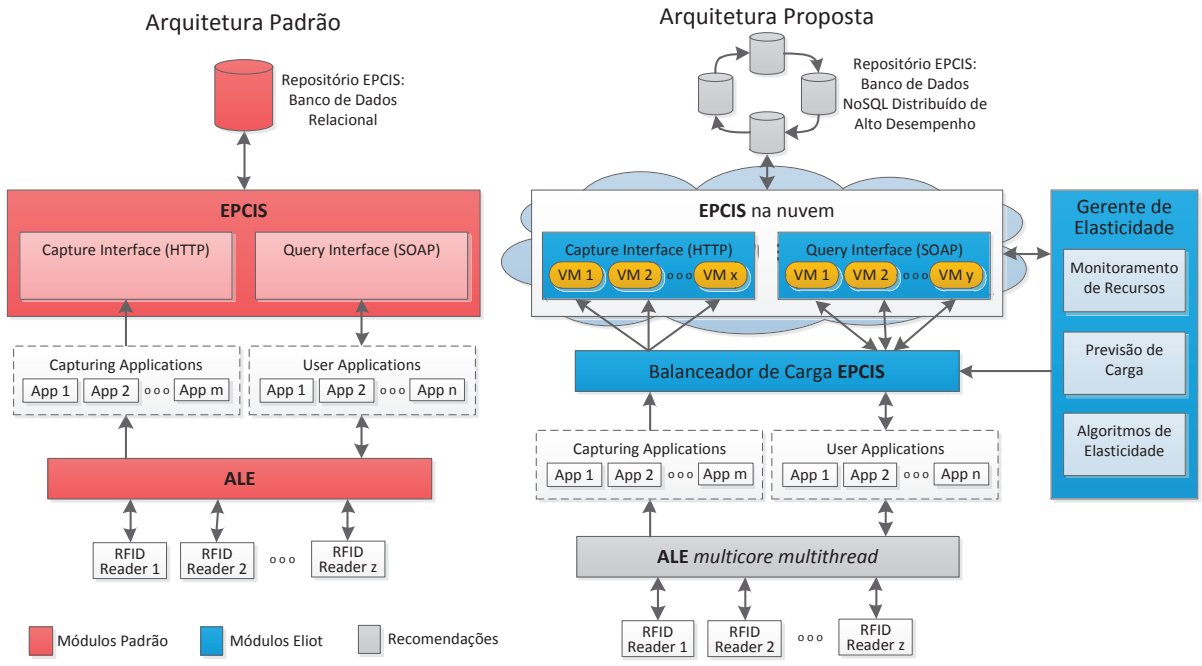
c) **Processamento paralelo para o módulo ALE:** visto que o módulo ALE é responsável pela captura e processamento básico dos dados vindos de diversas etiquetas RFID e de sensores, é necessário oferecer uma solução de processamento em paralelo para não serializar demandas que ocorrem paralelamente no “mundo real”;

d) **Alta disponibilidade e tolerância a falhas para o banco de dados:** os dados coletados de sensores e etiquetas são armazenados em um repositório para posterior consulta. Como o banco de dados é um elemento virtualmente centralizado, é essencial propor uma solução de alta disponibilidade e tolerante a falhas. Se o banco de dados não operar corretamente, todo o sistema IoT fica comprometido.

5.2 Arquitetura Eliot - *Elasticity-Driven Internet of Things*

A proposta consiste em alterações na arquitetura padrão do *middleware* e recomendações de alteração do repositório de dados e da máquina que executa o ALE, como pode ser visualizado na Figura 14. No diagrama à esquerda, é possível verificar como a arquitetura EPCglobal padrão é organizada. Nos blocos de cor vermelha estão concentrados os pontos críticos identificados durante a aplicação do *MIB* sobre o Fosstrak. Nesse diagrama é possível visualizar que não há indicação alguma de processamento paralelo para o módulo ALE. E para o EPCIS, tanto *Capture Interface* quanto *Query Interface* trabalham dentro do mesmo processo, competindo pelos mesmos recursos computacionais. No diagrama à direita é apresentada a arquitetura proposta, intitulada *Eliot*, com as modificações do padrão EPCglobal destacadas na cor azul e as recomendações de uso na cor cinza. Cada item é apresentado em detalhes nas próximas seções:

Figura 14 – Arquitetura *Eliot* para elasticidade da infraestrutura de IoT



Fonte: Elaborado pelo autor

5.2.1 Repositório de dados

Para o repositório de dados, recomenda-se utilizar um banco não relacional padrão NoSQL de forma distribuída, através de uma arquitetura P2P, semelhante ao que foi desenvolvido por Veen, Waaij e Meijer (2012). Essa família de bancos de dados tem na sua essência o fornecimento de soluções para grandes volumes de dados, tolerância a falhas, escalabilidade e alta disponibilidade. Os candidatos a serem utilizados são o Apache Cassandra[1] e o MongoDB[2].

5.2.2 Processamento paralelo para o módulo ALE

Conforme o modelo EPCglobal, o módulo ALE é responsável pelo gerenciamento e troca de dados com os leitores físicos. O protocolo de comunicação padrão é o LLRP, que utiliza sockets. Segundo esse protocolo, a comunicação de dados entre os equipamentos e o módulo ALE precisa ser pré-configurada e preestabelecida, e deve permanecer ativa enquanto os equipamentos estiverem ligados. Além das propriedades do equipamento, deve-se definir previamente através de uma interface de configuração do ALE o endereço IP e a porta de comunicação de cada equipamento em uso. Assim, fica evidente que essa interface entre ALE e equipamento é bastante estática, o que torna inviável propor um modelo de balanceamento de carga dinâmico.

Mas a característica principal que deve ser considerada para esse módulo ALE é que ele atende demandas do "mundo real", no qual dezenas ou centenas de leitores podem identificar objetos rastreáveis ao mesmo tempo e transmitir os dados para o módulo no mesmo instante. Isso caracteriza uma forte concorrência e necessidade de processamento paralelo. Para corresponder a essa demanda, a recomendação é reorganizar o módulo ALE para ser executado com bibliotecas *multithread* em um servidor *multicore*, para que possa rodar mais rápido e de forma paralela.

5.2.3 Divisão do módulo EPCIS

O módulo EPCIS possui duas interfaces de comunicação com aplicações externas: (i) a interface *Capture Interface*, responsável pelo armazenamento das informações vindas das *Capturing Applications* e que realiza unicamente operações de escrita; (ii) a interface *Query Interface*, responsável pela realização de consultas ao banco de dados solicitadas pelos usuários e aplicações cliente, realizando unicamente operações de leitura.

Considerando essa fronteira que divide os serviços oferecidos pelo módulo EPCIS, a proposta consiste em dividir o EPCIS em submódulos e executá-los em máquinas distintas. Um módulo deve executar unicamente a *Capture Interface* e outro módulo deve executar unicamente a *Query Interface*. Com essa divisão, fica mais simples monitorar os indicadores de desempenho do sistema, identificar os *thresholds* e tomar decisões quanto a balanceamento de carga.

[1] http://cassandra.apache.org
[2] http://www.mongodb.org

5.2.4 Disponibilização de *templates* de máquinas virtuais

Dando sequência à divisão do módulo EPCIS, outra alteração sugerida na arquitetura é a criação de dois *templates* pré-configurados para máquinas virtuais, um com a *Capture Interface* e outro com a *Query Interface*. Esses *templates* devem ser utilizados por um *Gerente de Escalabilidade* para instanciar novas máquinas virtuais que irão atender a sobrecarga de processamento vindo de novas demandas. *Templates* de máquinas virtuais são um recurso amplamente utilizado por gerenciadores de nuvem, pois proporcionam a disponibilização de máquinas virtuais pré-configuradas e ajustadas para fins específicos em um curto espaço de tempo.

5.2.5 Escalabilidade e elasticidade do EPCIS

O último item proposto é aplicar elasticidade reativa ao módulo EPCIS, implementando um *Gerente de Escalabilidade* para identificar sobrecarga ou subutilização do sistema e instanciar ou desalocar máquinas virtuais conforme a necessidade. Normalmente são utilizadas instruções do tipo *regra-condição-ação* e *thresholds* mínimos e máximos preestabelecidos para o gerenciamento da elasticidade.

Um algoritmo de elasticidade deve ser executado periodicamente para analisar uma previsão de carga *lp* (*load prediction*) para cada métrica avaliada (CPU, rede, disco ou memória). A previsão de carga *lp* utiliza o conceito de *aging*, com o objetivo de detectar falsos-positivos e falsos-negativos nas ações de alocação e desalocação (TANENBAUM, 2010). Basicamente, a técnica de *aging* atribui um peso mais elevado para a observação mais recente, dividindo por uma potência de 2 em cada elemento subsequente da série histórica.

Por exemplo, considerando um *threshold* máximo de 80% e observações como 72, 70, 78, 71 e 81, sendo essa última a mais recente, a previsão de carga *lp* irá informar o valor 74,62 como resultado da seguinte expressão: $lp = \frac{81}{2} + \frac{71}{4} + \frac{78}{8} + \frac{70}{16} + \frac{72}{32}$. Dessa maneira, o último valor "81" não irá disparar nenhuma ação de elasticidade, uma vez que é avaliado como falso-positivo. Essa estratégia consegue amortizar a importância dos picos, uma vez que uma alocação errônea não irá ocorrer se forem analisados os dados históricos do sistema.

Depois de testar a alocação, as mesmas métricas devem ser aplicadas em relação ao *threshold* mínimo com o intuito de eventualmente desalocar máquinas virtuais. Por último, o algoritmo tentará migrar para outros nodos as máquinas virtuais previamente alocadas, realizando uma consolidação da infraestrutura oferecida. Essa última parte é especialmente útil por questões de custos e economia de energia, uma vez que os nodos podem ser reduzidos na sua quantidade.

5.3 Algoritmos do *Gerente de Escalabilidade*

O *Gerente de Escalabilidade* deverá implementar e executar alguns algoritmos. As variáveis de controle utilizadas por eles são descritas a seguir:

- **occurrence[]:** vetor para armazenar a série temporal do parâmetro analisado. Deve ser inicializada com zero em todos os seus elementos, e a cada período de tempo preestabelecido deve ser atualizada com o valor do recurso computacional monitorado;
- **lp_epcis:** variável que armazena a previsão de carga momentânea do parâmetro analisado. É atualizada a cada período de tempo preestabelecido aplicando-se a técnica de *aging* sobre o vetor *occurrence[]*;
- **high_threshold_epcis:** limite superior do parâmetro analisado. Deve ser definido com o valor encontrado ao se interpretar os dados do *micro benchmark* sobre o *middleware* IoT;
- **low_threshold_epcis:** limite inferior do parâmetro analisado. Deve ser definido com o valor encontrado ao se interpretar os dados do *micro benchmark* sobre o *middleware* IoT.

A primeira ação é calcular a previsão de carga, conforme visualizado no Algoritmo 1. O vetor *occurrence[]* armazena os últimos indicadores dos recursos computacionais sendo avaliados. Então, deve-se aplicar a técnica de *aging* para se obter a previsão de carga e armazená-la na variável *lp_epcis*.

Algoritmo 1: Cálculo da previsão de carga com método de *aging*

Entrada: Vetor $occurrence[\,]$ contendo o valor dos recursos computacionais
Saída: Previsão de carga conforme técnica de *aging*

1: $lp\_epcis \leftarrow 0$;
2: **for** $cada\ elemento\ de\ occurrence[\,]\ variando\ i$ **do**
3: $\quad lp\_epcis \leftarrow lp\_epcis + \frac{occurrence[i]}{2^{(i+1)}}$;
4: **end for**

Em seguida, o Algoritmo 2 avalia a necessidade de alocação de uma nova máquina virtual comparando a previsão de carga *lp_epcis* calculada anteriormente em relação ao *threshold* superior *high_threshold_epcis*. Caso a previsão de carga *lp_epcis* seja maior que o *threshold* superior, uma nova máquina virtual EPCIS é lançada. Depois, avalia-se a possibilidade de liberação de alguma máquina virtual existente, comparando a mesma previsão de carga *lp_epcis* com o *threshold* inferior *low_threshold_epcis*.

Ainda no Algoritmo 2, caso a previsão de carga *lp_epcis* seja menor que o *threshold* inferior e caso exista mais de uma máquina virtual EPCIS alocada, a máquina virtual EPCIS com menor carga é desalocada. Para finalizar o algoritmo, os processos de alocação e liberação de máquina virtual se comunicam de forma síncrona com o módulo balanceador de carga para que não

ocorra de uma máquina recém alocada ficar ociosa ou então uma requisição ser direcionada para uma máquina recém liberada, gerando falha.

Algoritmo 2: Alocação e liberação de máquina virtual

Entrada: Previsão de carga lp_epcis e seus limites inferior e superior
Saída: Alocação ou liberação de máquina virtual

1: **if** $(lp\_epcis > high\_threshold\_epcis)$ **then**
2: $vm \leftarrow$ cria uma nova máquina virtual EPCIS
3: notifica balanceador de carga EPCIS (adiciona vm)
4: **else if** $(lp\_epcis < low\_threshold\_epcis)$ **and** $(count\_allocated\_vm > 1$ **then**
5: $vm \leftarrow$ pega máquina virtual com menor carga
6: notifica balanceador de carga EPCIS (remove vm)
7: libera a máquina virtual EPCIS (vm)
8: **end if**

Por último, no Algoritmo 3 é feita uma avaliação da necessidade de reorganização das máquinas virtuais alocadas. Inicialmente, faz-se uma varredura em cada nodo em uso para verificar se possui alguma máquina virtual instanciada. Caso exista, é feita uma nova varredura nos demais nodos para identificar se algum deles possui carga livre suficiente para acomodar as máquinas virtuais do nodo original. Caso afirmativo, as máquinas virtuais do nodo original são migradas para o outro nodo com recursos disponíveis. Então, o nodo original é identificado como não contendo máquinas virtuais, para que possa ser liberado na rotina seguinte que realiza a consolidação da estrutura oferecida.

Algoritmo 3: Reorganização das máquinas virtuais

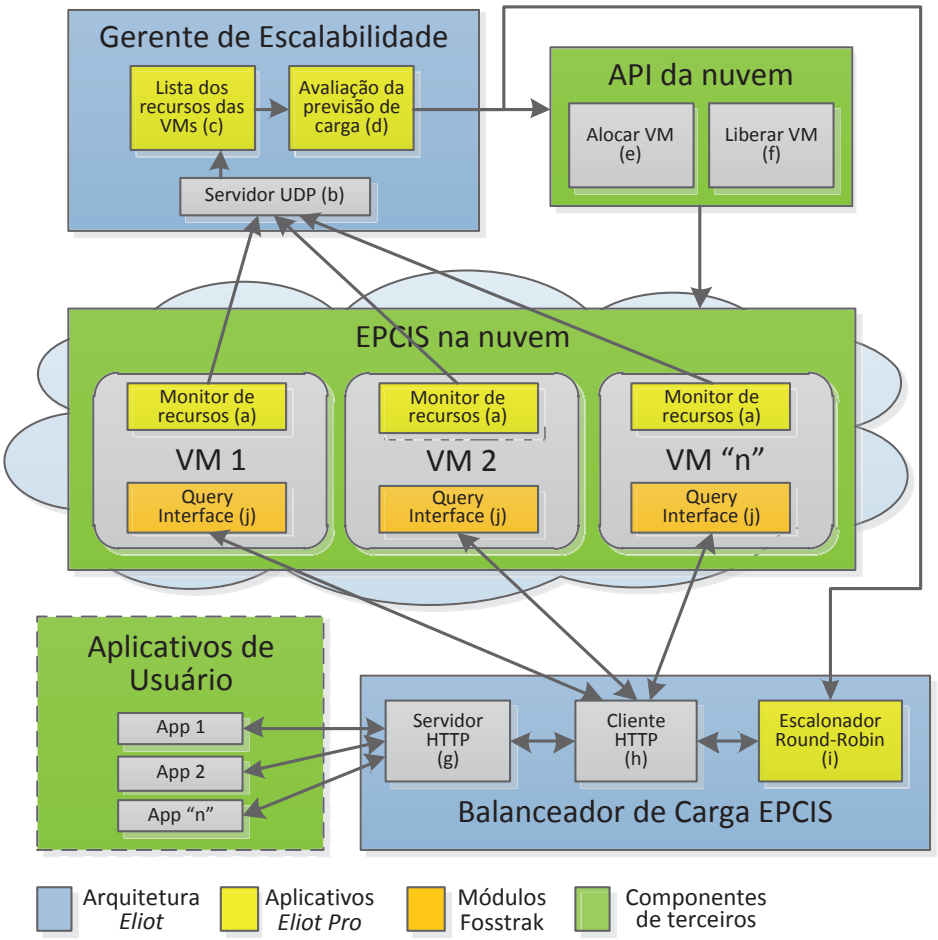
Entrada: Lista dos nodos em uso
Saída: Consolidação dos *hosts* e liberação de nodos

1: **for** $cada\ nodo\ em\ uso,\ variando\ i$ **do**
2: **if** $nodo\ i\ possui\ maquinas\ virtuais$ **then**
3: **for** $cada\ nodo\ em\ uso,\ variando\ j$ **do**
4: **if** $(i \neq j)$ **and** $(carga\ em\ uso\ do\ nodo\ j < carga\ livre\ do\ nodo\ i)$ **then**
5: migra todas as máquinas virtuais de j para i;
6: marca host j como não contendo máquinas virtuais;
7: consolida host j;
8: **end if**
9: **end for**
10: **end if**
11: **end for**

6 IMPLEMENTAÇÃO

Para monitorar e gerenciar as máquinas virtuais configuradas conforme arquitetura *Eliot* proposta no capítulo anterior, foi desenvolvido um conjunto de aplicativos intitulado "*Eliot Pro - Elasticity-Driven Internet of Things Prototype*". Conforme ilustrado na Figura 15, esse conjunto de aplicativos tem como objetivo oferecer um *middleware* IoT elástico de forma transparente para as aplicações clientes que utilizam o *middleware*. Entre os algoritmos descritos no capítulo anterior, no *Eliot Pro* foram implementados os códigos para "Cálculo da previsão de carga com método de *aging*" e "Alocação e liberação de máquina virtual".

Figura 15 – Protótipo implementado para avaliação da arquitetura *Eliot*



Fonte: Elaborado pelo autor

6.1 O sistema *Eliot Pro - Elasticity-Driven Internet of Things Prototype*

O módulo de monitoramento de recursos do *Eliot Pro* (Figura 15.a) consiste em um serviço do sistema operacional que monitora o uso de CPU e tráfego de rede de entrada e saída de cada máquina virtual em uso. A cada 1 segundo, esse módulo envia os dados monitorados para o

Gerente de Escalabilidade (Figura 15.b) através do protocolo UDP. Com os dados recebidos, o módulo de gerenciamento alimenta uma lista (Figura 15.c) sobre a qual irá realizar o cálculo da previsão de carga do sistema (Figura 15.d). Essa previsão de carga é então utilizada para a tomada de decisão de alocar (Figura 15.e) ou desalocar (Figura 15.f) máquinas virtuais através da API de gerenciamento do ambiente da nuvem computacional.

O Algoritmo 4 ilustra como estão organizadas as rotinas de monitoramento dos recursos utilizados pelas máquinas virtuais ativas e a tomada de decisão para alocar ou desalocar uma máquina virtual. Inicialmente na linha 1, é recebida uma *string* via protocolo UDP contendo os dados de CPU e tráfego de rede em uso por uma determinada máquina virtual e são armazenados em um *buffer*. Em seguida, é extraído desse *buffer* o índice correspondente à máquina virtual que será utilizado como identificador na matriz que armazena os dados de todas as máquinas virtuais.

Nas linhas 3, 4 e 5 são atualizados os dados de CPU, tráfego de rede de entrada e de saída da máquina virtual previamente identificada. Na linha 6 é feito o cálculo da previsão de carga com o método de *aging*, conforme descrito no Algoritmo 1 apresentado na Seção 5.3. Na linha 7 é calculada a previsão de carga média de todas as máquinas virtuais. Após obter esse valor de carga média, na linha 8 é feito um comparativo em relação aos *thresholds* superior e inferior para identificar a necessidade ajustar a quantidade de máquinas virtuais em uso. Na linha 9 é realizada uma chamada ao Algoritmo 2 também descrito na Seção 5.3, que realiza a alocação ou liberação de máquinas virtuais, quando necessário.

Algoritmo 4: Monitoramento dos recursos das máquinas virtuais e tomada de decisão para alocar ou desalocar máquina virtual

Entrada: *String* de dados contendo os recursos monitorados nas máquinas virtuais
Saída: Alocação ou liberação de máquina virtual, se necessário

1: $buffer \leftarrow udp\_receive\_resources$;
2: $vm\_index \leftarrow extract\_vm\_index(buffer)$;
3: $vm[vm\_index].cpu \leftarrow extract\_cpu(buffer)$;
4: $vm[vm\_index].net\_in \leftarrow extract\_net\_in(buffer)$;
5: $vm[vm\_index].net\_out \leftarrow extract\_net\_out(buffer)$;
6: $vm[vm\_index].lp \leftarrow calculate\_lp(vm[vm\_index])$;
7: $lp\_epcis \leftarrow calculate\_system\_lp(vm[])$;
8: realiza avaliação do lp_epcis em relação aos $thresholds$
9: realiza alocação ou liberação de máquina virtual, se necessário

6.2 Encapsulamento das diversas EPCIS *Query Interfaces*

Além dos módulos de monitoramento e gerenciamento descritos anteriormente, foi implementado o módulo para balanceamento de carga das requisições de consultas, feitas originalmente para uma única *EPCIS Query Interface*. Como agora podem existir diversas máquinas

virtuais disponibilizando o serviço de consulta aos dados EPC, esse módulo fica responsável por oferecer toda a estrutura de máquinas virtuais para as aplicações clientes de forma transparente, pois implementa uma interface SOAP unificada de consulta, redireciona as requisições para as máquinas virtuais instanciadas e administra o balanceamento de carga entre elas.

O módulo consiste no encapsulamento das diversas interfaces de consulta disponíveis e as centraliza em um único endereço. Ele possui um servidor HTTP (Figura 15.g) escutando as requisições vindas das aplicações clientes e as redireciona através de um cliente HTTP (Figura 15.h) para a máquina virtual que irá processar a requisição. Para escolha da máquina virtual de destino (Figura 15.j) foi implementado um escalonador (Figura 15.i) com um algoritmo *Round-Robin*.

A técnica de escalonamento *Round-Robin* descrita no Algoritmo 5 consiste em uma simples fila circular. Cada vez que o escalonador é chamado, ele retorna o próximo servidor da fila, tendo como referência o servidor retornado anteriormente. Caso a fila chegue no final, o escalonador retorna para o início, devolvendo o primeiro servidor disponível na fila.

Por se tratar de um algoritmo simples, uma vantagem de se utilizar o *Round-Robin* é que ele é muito rápido e consome poucos recursos computacionais. Porém, não há uma regra mais elaborada para a escolha do servidor que deverá processar a requisição. Conforme estudos realizados por Zhan et al. (2014) e Pascual et al. (2014), isso pode ser uma desvantagem caso o servidor escolhido já esteja sobrecarregado, enquanto outro servidor poderia estar com mais recursos disponíveis para processamento.

Algoritmo 5: Escalonador *Round-Robin*

Entrada: Lista de *hosts* ativos ($list\_host[]$) e índice do último *host* escalonado ($host\_idx$)
Saída: Próximo $host$ da lista

1: $host\_idx \leftarrow host\_idx + 1$;
2: **if** $host\_idx > length(list\_host[]) - 1$ **then**
3: $\quad host\_idx \leftarrow 0$;
4: **end if**
5: $return\ list\_host[host\_idx]$;

O Algoritmo 6 ilustra como é realizado o redirecionamento e processamento das requisições. Um servidor HTTP único fica na escuta e recebe as requisições SOAP vindas das aplicações clientes. Em seguida, através de uma técnica chamada *Round-Robin* é feita a escolha do servidor EPCIS de destino que irá processar a consulta propriamente dita. Após a escolha do servidor de destino, um cliente HTTP dispara a requisição previamente solicitada pela aplicação, aguarda o retorno do servidor EPCIS de destino e devolve para a mesma aplicação cliente a resposta recebida do servidor. Para a aplicação cliente, esse processo é totalmente transparente, ela não percebe que o endereço onde ela solicitou a consulta não é o servidor EPCIS real. Também não é necessário realizar nenhuma alteração na aplicação cliente.

Algoritmo 6: Redirecionamento das requisições de consulta

Entrada: Requisição http
Saída: Resposta da requisição http redirecionada para o *host* de destino

1: $request \leftarrow http\_receive\_query$;
2: $host \leftarrow scheduler\_get\_host()$;
3: $response \leftarrow http\_post(request, host)$;
4: $return\ response$

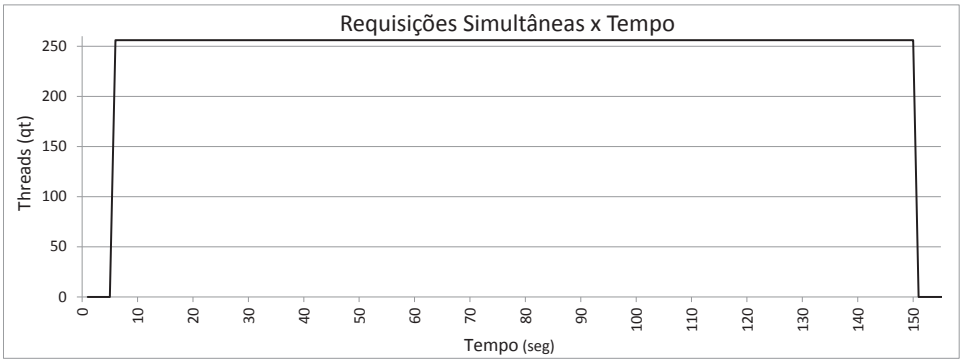
6.3 Aplicação cliente para testes

Tendo como base o *software "MIB: Micro Benchmark para Avaliação de Middlewares de Internet das Coisas"* descrito na Seção 4.1 com o objetivo de avaliar *middlewares* IoT, foi desenvolvida uma aplicação cliente para testar a implementação da arquitetura *Eliot*. Essa aplicação consiste em diversas *threads* realizando consultas ao EPCIS simultaneamente, com perfis de demanda de requisições constante, ascendente, descendente e onda, detalhados a seguir:

6.3.1 Constante

Nesse perfil ilustrado na Figura 16, as requisições são feitas de forma constante, sendo possível apenas definir a quantidade de requisições simultâneas. O Algoritmo 7 que representa esse perfil é bastante simples, e ilustra como é realizada a rotina de requisições com quantidade constante. Na linha 1 são criadas todas as *threads*, e na linha 2 elas são disparadas para execução. Durante o teste, não há ajuste na quantidade de *threads* ativas.

Figura 16 – Requisições constantes



Fonte: Elaborado pelo autor

6.3.2 Ascendente

Nesse perfil, as requisições são feitas com a quantidade simultânea variando de forma ascendente constante iniciando em 1, como pode ser visto na Figura 17. É possível configurar o número máximo de requisições simultâneas, o intervalo de tempo e a quantidade em que ocorre o acréscimo.

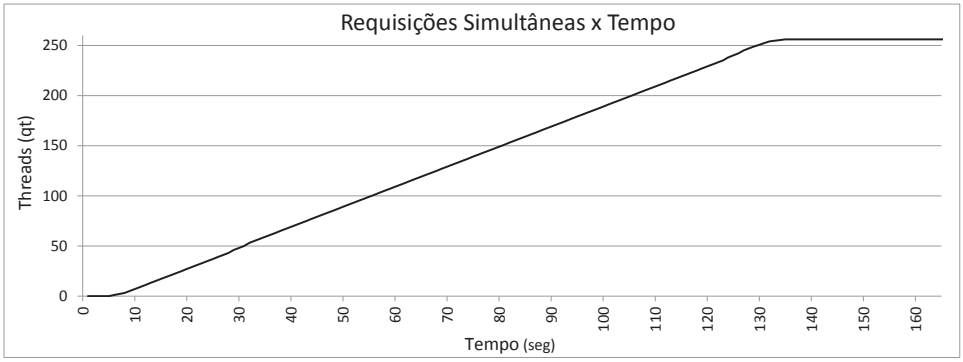
Algoritmo 7: Requisições constantes

Entrada: Quantidade de *threads* a serem criadas ($cfg\_max\_threads$)
Saída: *Threads* criadas e executadas

1: $create\_threads(cfg\_max\_threads)$;
2: $run\_threads()$;

O Algoritmo 8 demonstra como é feito o controle da quantidade de requisições simultâneas ascendentes. Na linha 1, a variável de controle x é inicializada com o valor 1. Essa variável armazena a quantidade de *threads* ativas. Em seguida, na linha 2 é iniciado um laço que fica ativo enquanto o usuário não solicitar o encerramento do teste. Na linha 3, é feito o ajuste da quantidade de *threads* ativas. Na primeira iteração do laço é criada apenas uma *thread*, e a cada iteração futura esse valor é incrementado.

Figura 17 – Requisições ascendentes



Fonte: Elaborado pelo autor

Algoritmo 8: Requisições ascendentes

Entrada: Quantidade máxima de *threads* a serem criadas ($cfg\_max\_threads$), intervalo de tempo entre a criação das *threads* ($cfg\_wait\_interval$) e quantidade de *threads* a serem criadas a cada intervalo de tempo (cfg_step)
Saída: Quantidade de *threads* crescente no decorrer do tempo

1: $x \leftarrow 1$;
2: **while** $not\ terminated$ **do**
3: $adjust\_running\_threads(x)$;
4: $sleep(cfg\_wait\_interval)$;
5: $x \leftarrow x + cfg\_step$;
6: **if** $x > cfg\_max\_threads$ **then**
7: $x \leftarrow cfg\_max\_threads$
8: **end if**
9: **end while**

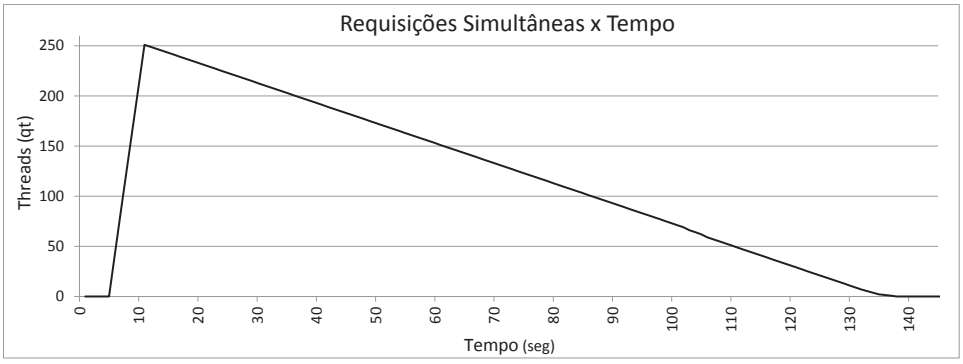
Na linha 4 é aplicado um tempo de espera no laço para que as *threads* possam ser executadas e realizarem as requisições. Em seguida na linha 5, é calculada a nova quantidade de *threads*, conforme configurado pelo usuário. Na linha 6 é verificado se essa quantidade atingiu o valor máximo de requisições simultâneas também definido pelo usuário. Caso afirmativo, o valor é reajustado para esse limite pré-configurado. Na próxima iteração do laço, a quantidade de *threads* ativas é ajustada conforme o cálculo realizado anteriormente.

6.3.3 Descendente

Nesse perfil representado pela Figura 18, as requisições são feitas com a quantidade simultânea variando de forma descendente constante, encerrando em 0. É possível configurar o número inicial de requisições simultâneas, o intervalo de tempo e a quantidade em que ocorre o decréscimo.

O Algoritmo 9 demonstra como é feito o controle da quantidade de requisições simultâneas descendentes. Na linha 1, a variável de controle x é inicializada com o máximo de requisições simultâneas definido pelo usuário. Essa variável armazena a quantidade de *threads* ativas. Em seguida, na linha 2 é iniciado um laço que fica ativo enquanto o usuário não solicitar o encerramento do teste. Na linha 3, é feito o ajuste da quantidade de *threads* ativas. Na primeira iteração do laço, são criadas todas as *threads* configuradas, e a cada iteração futura esse valor é decrementado. Na linha 4 é aplicado um tempo de espera no laço para que as *threads* possam ser executadas e realizarem as requisições. Em seguida na linha 5, é calculada a nova quantidade de *threads*, conforme configurado pelo usuário. Na linha 6 é verificado se essa quantidade atingiu um valor inferior a zero. Caso afirmativo, o valor é reajustado para zero. Na próxima iteração do laço, a quantidade de *threads* ativas é ajustada conforme o cálculo realizado anteriormente.

Figura 18 – Requisições descendentes



Fonte: Elaborado pelo autor

Algoritmo 9: Requisições descendentes

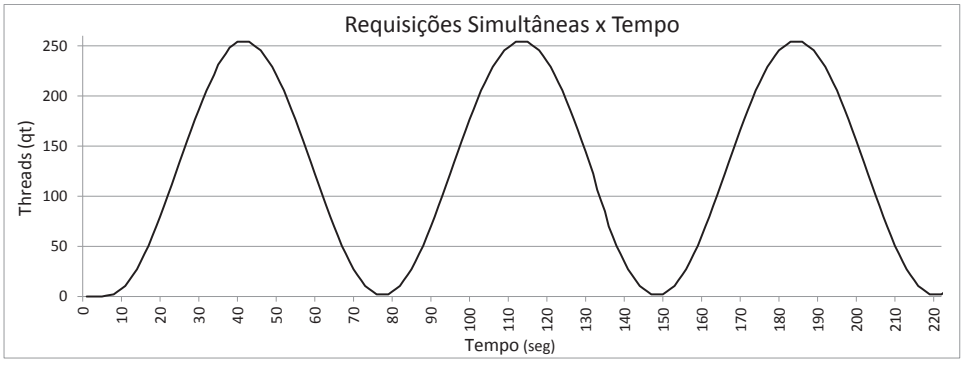
Entrada: Quantidade inicial de *threads* a serem criadas ($cfg\_max\_threads$), intervalo de tempo entre a liberação das *threads* ($cfg\_wait\_interval$) e quantidade de *threads* a serem liberadas a cada intervalo de tempo (cfg_step)
Saída: Quantidade de *threads* decrescente no decorrer do tempo

```
1: x ← cfg_max_threads;
2: while not terminated do
3:     adjust_running_threads(x);
4:     sleep(cfg_wait_interval);
5:     x ← x − cfg_step;
6:     if x < 0 then
7:         x ← 0
8:     end if
9: end while
```

6.3.4 Onda

No último perfil proposto, as requisições são feitas de forma cíclica, com a quantidade simultânea variando conforme uma função senoidal ilustrada na Figura 19. É possível configurar o número máximo de requisições simultâneas (pico da senoide), o intervalo de tempo e a quantidade de graus em que o número de *threads* é ajustado.

Figura 19 – Requisições com comportamento de onda



Fonte: Elaborado pelo autor

O Algoritmo 10 ilustra como é realizado o ajuste da quantidade de *threads* que realizam as requisições simultâneas. A variável de controle x é inicializada com o valor zero, e armazena a posição correspondente à quantidade de *threads* ativas. Um laço é realizado enquanto o usuário não solicitar o encerramento do teste. A cada iteração do laço, na linha 3 é feito o ajuste do valor da variável x incrementando o intervalo configurado, em graus. Na linha 4, esse valor é convertido para radianos e aplicado um deslocamento de fase na ordem de $-(\pi/2)$. Esse deslocamento é necessário para que a onda inicie com quantidade de *threads* igual a zero. Na

linha 5 é aplicada a função senoiodal e feita a soma de uma unidade na amplitude, para evitar valores negativos. Nessa mesma linha, ainda é feita a multiplicação pelo valor definido nas configurações como sendo a máxima quantidade de requisições simultâneas. Seguindo na linha 6, é feito o juste da quantidade de requisições, alocando ou liberando as *threads* conforme necessidade. Para finalizar a iteração do laço, na linha 7 é realizada uma espera de tempo conforme definido nas configurações, para que a quantidade de requisições recém calculada possa permanecer ativa por um intervalo de tempo antes de ser recalculada e reajustada.

Algoritmo 10: Requisições com comportamento de onda

Entrada: Quantidade de *threads* no pico da onda ($cfg\_max\_threads$), intervalo de tempo entre os ajustes na quantidade de *threads* ($cfg\_wait\_interval$) e incremento angular da função senoidal ser realizado a cada intervalo de tempo (cfg_step)
Saída: Quantidade de *threads* em formato de onda no decorrer do tempo

1: $x \leftarrow 0$;
2: **while** $not\ terminated$ **do**
3: $\quad x \leftarrow x + cfg\_step$;
4: $\quad x\_rad \leftarrow x/360 * 2 * \pi - (\pi/2)$;
5: $\quad y \leftarrow round((sin(x\_rad) + 1)/2 * cfg\_max\_threads)$;
6: $\quad adjust\_running\_threads(y)$;
7: $\quad sleep(cfg\_wait\_interval)$;
8: **end while**

6.4 Restrições da implementação

A arquitetura proposta na Seção 5.2 não foi integralmente implementada. Como o foco do trabalho é a escalabilidade do *middleware* IoT, optou-se direcionar todos os esforços para a realização de tarefas intimamente ligadas ao monitoramento e processamento dos indicadores de desempenho, assim como alocação e liberação de máquinas virtuais. Mesmo reconhecendo a importância dos itens não implementados, eles não são indispensáveis para a realização das avaliações da arquitetura escalável.

6.4.1 Banco de dados

Conforme proposto originalmente, seria utilizado um banco de dados NoSQL distribuído como objetivo de oferecer tolerância a falhas e balanceamento de carga, resultando em alto desempenho. Para montar essa estrutura de banco distribuído seria necessário um bom esforço de instalação e configuração, além de alocação de várias máquinas no ambiente de nuvem. Dessa forma, para minimizar as chances de ocorrerem gargalos, tentou-se compensar com uma máquina com muito recurso computacional. Então, o banco de dados MySQL acabou sendo instalado em um *hardware* contendo 8 CPUs Intel Xeon E5-2680 v2, 15 GB RAM e placa de rede *gigabit*.

6.4.2 Reorganização das máquinas virtuais

O algoritmo “Reorganização das máquinas virtuais” não foi implementado pois não é essencial para a avaliação da arquitetura escalável. Ele tem por objetivo apenas otimizar o consumo de energia da nuvem computacional, realizando a consolidação das máquinas físicas que executam as máquinas virtuais. Sua implementação fica como sugestão de melhoria futura.

7 AVALIAÇÃO

Este capítulo apresenta a avaliação da arquitetura proposta através da instalação e configuração do *middleware* IoT em uma ambiente de nuvem computacional juntamente com o sistema *Eliot Pro*, e a realização de diversos testes com cenários distintos de carga de dados e consultas ao repositório. O capítulo descreve o ambiente de testes, a metodologia de avaliação, apresenta os resultados obtidos e faz uma discussão sobre os comportamentos identificados.

7.1 Ambiente de testes

Para avaliar a arquitetura *Eliot* proposta, novamente foi utilizado o *middleware* Fosstrak por se tratar de uma implementação do padrão EPCglobal. Além disso, utilizando o mesmo *middleware* no qual foi aplicado o *micro benchmark MIB* apresentado no Capítulo 4, é possível fazer comparativos mais confiáveis de desempenho entre as 3 arquiteturas computacionais: (i) não distribuída; (ii) distribuída e não escalável; (iii) distribuída e escalável.

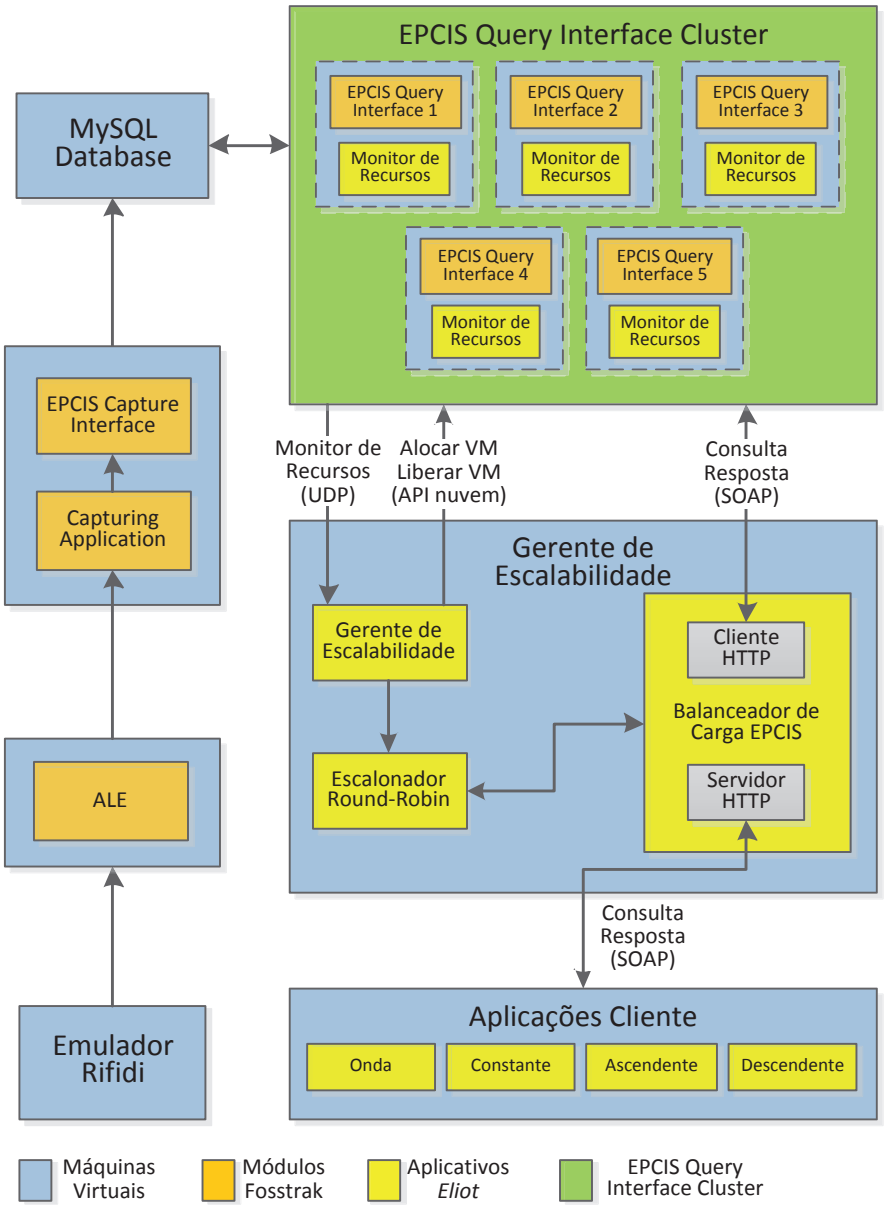
O ambiente de nuvem computacional utilizado foi o EC2 da Amazon[1]. Esse ambiente foi escolhido pois é reconhecido mundialmente e permite organizar facilmente a arquitetura sobre a qual a aplicação será executada, com uma vasta opção de máquinas virtuais e sistemas operacionais pré-configurados. A Figura 20 ilustra a estrutura de *hardware* utilizada e como o *middleware* Fosstrak foi configurado para os testes juntamente com o conjunto de aplicativos *Eliot Pro*, que proporcionam elasticidade ao sistema.

Cada elemento na cor azul da Figura 20 representa uma máquina virtual, no total de 11 máquinas. Todas elas executam o sistema operacional Windows 2008 Server R2. A configuração de *hardware* básica é 1 CPU Intel Xeon 2.5 GHz e 1 GB RAM. Como na arquitetura proposta foi previsto processamento paralelo para o módulo ALE, foi utilizada uma máquina com configuração de *hardware* contendo 8 CPUs Intel Xeon E5-2680 v2 e 15 GB RAM. Conforme descrito na Seção 6.4.1, o banco de dados MySQL também foi instalado em uma máquina com as mesmas configurações da máquina que executa o módulo ALE.

Os elementos na cor amarela da Figura 20 representam os aplicativos que compõem o *Eliot Pro*, sistema desenvolvido para gerenciar a escalabilidade e proporcionar elasticidade ao *middleware* IoT. A cor laranja representa os módulos padrão do *middleware* Fosstrak. Na lateral esquerda da figura, temos os módulos dispostos de forma distribuída, porém sem controle de elasticidade. O elemento na cor verde da Figura 20 representa o *cluster* de máquinas virtuais contendo os módulos *EPCIS Query Interface* do Fosstrak, que são controlados pelo *Gerente de Escalabilidade* do *Eliot Pro*. O acesso a esse *cluster* ocorre de forma transparente através do módulo *EPCIS Query Interface Wrapper*. O *cluster* foi configurado para ter no mínimo uma e no máximo cinco máquinas virtuais. A quantidade de máquinas alocadas irá depender da carga de consultas solicitadas pelas aplicações clientes e dos limites inferior e superior configurados para alocação e liberação de recursos computacionais.

[1] http://aws.amazon.com/ec2

Figura 20 – Estrutura do *middleware* IoT configurado na nuvem



Fonte: Elaborado pelo autor

7.2 Metodologia de avaliação

Para realizar a avaliação da estrutura distribuída e escalável implementada na nuvem, foi posto em prática um fluxo de escrita e outro de leitura de dados EPC. O fluxo de escrita tem por objetivo garantir que o *middleware* seja capaz de capturar os dados das etiquetas e gravar no banco de dados de forma contínua e sem falhas. O fluxo de leitura tem com objetivo gerar as mais variadas demandas de consulta aos dados EPC, refletindo em diversos cenários de uso dos recursos computacionais do *cluster* de máquinas *EPCIS Query Interface* e consequentemente a alocação e liberação dessas máquinas virtuais de forma automática, eficiente e transparente para as aplicações clientes.

7.2.1 Escrita de dados

O processo de escrita segue as mesmas configurações utilizadas durante aplicação do *micro benchmark* no Fosstrak, descrito no Capítulo 4. Uma máquina executa o *software* Rifidi emulando 4 leitores RFID. Cada leitor possui 4 etiquetas ativas, gerando uma carga de dados equivalente a 16 etiquetas por ciclo de escrita. Uma outra máquina executa exclusivamente o módulo ALE para captura dos dados das etiquetas. Ainda seguindo as mesmas configurações utilizadas no *micro benchmark*, a cada 2 segundos o módulo ALE envia o *Event Cycle Report* com os dados das etiquetas capturadas para uma terceira máquina, que executa a *Capturing Application* e o *EPCIS Capturing Interface*. Então, essa terceira máquina envia os dados para o MySQL que roda em uma quarta máquina.

7.2.2 Leitura de dados

O fluxo de leitura de dados é mais complexo que o de escrita, e foi alterado em relação à configuração padrão do Fosstrak para oferecer escalabilidade e elasticidade ao módulo *EPCIS Query Interface*. O funcionamento detalhado desse fluxo foi descrito anteriormente na Seção 6.3. Os procedimentos de leitura consistem em gerar carga de requisições de consultas ao *EPCIS Query Interface* nas mais variadas formas e quantidades simultâneas, com os seguintes comportamentos:

- **Constante:** Gera carga constante de requisições simultâneas. Com esse perfil é possível realizar dezenas ou centenas de consultas simultaneamente, com o objetivo de avaliar a estabilidade do sistema com carga alta por um longo período de tempo. Como resultado, é esperada uma sobrecarga inicial devido a grande quantidade simultânea de requisições disparadas no início do teste. Em seguida, deve-se observar a alocação de uma ou mais máquinas virtuais até que o nível de utilização médio dos recursos computacionais do *cluster* de máquinas *EPCIS Query Interface* estejam dentro dos limites configurados pelo usuário;

- **Ascendente:** Gera carga crescente de requisições simultâneas variando no tempo, iniciando em uma requisição até um limite máximo definido pelo usuário. Tem por objetivo avaliar o comportamento do sistema em situações de crescimento constante da demanda de recursos computacionais. O resultado esperado é a alocação automática de máquinas *EPCIS Query Interface* conforme a demanda de requisições cresce;

- **Descendente:** Gera carga decrescente de requisições simultâneas variando no tempo, iniciando em uma quantidade máxima definida pelo usuário e encerrando sem nenhuma requisição. Tem por objetivo avaliar o comportamento do sistema em situações de queda constante da demanda de recursos computacionais. O resultado esperado inicial é semelhante ao cenário de requisições constantes, ou seja, uma alta demanda inicial resultando

na alocação repentina de uma ou mais máquinas virtuais. Em seguida, com a queda da demanda de requisições, é esperado que os recursos computacionais em uso sejam reduzidos e ocorra a liberação gradual de máquinas *EPCIS Query Interface* conforme a demanda de requisições diminui;

- **Onda:** Gera carga de requisições simultâneas variando no tempo conforme uma função senoidal. Nesse perfil é possível realizar diversas consultas variando a quantidade simultânea em um formato de onda. Tem como objetivo avaliar o aumento e diminuição do uso de recursos computacionais conforme a carga varia no tempo. Como resultado, é esperada a alocação e liberação de máquinas virtuais automaticamente, acompanhando a taxa de utilização dos recursos.

7.2.3 Avaliação da arquitetura distribuída sem elasticidade

Avaliar a arquitetura distribuída sem suporte à elasticidade do módulo *EPCIS Query Interface* é importante para se obter resultados referentes unicamente à arquitetura distribuída. Estes resultados serão depois comparados com os testes realizados na estrutura escalável e também em relação à arquitetura não distribuída, para avaliação de ganho de desempenho.

Para realizar os testes sem elasticidade do módulo *EPCIS Query Interface*, o *Eliot* foi configurado com o *threshold* inferior igual a zero e o *threshold* superior igual a 100. Com esses limites, sempre existirá somente uma máquina virtual no *cluster* de *EPCIS Query Interface*, pois a previsão de carga *lp* jamais ultrapassará esses valores e o *Eliot* não irá executar os comandos para alocação ou liberação de máquina virtual.

7.2.4 Avaliação da arquitetura distribuída com elasticidade

A arquitetura distribuída com suporte à elasticidade do módulo *EPCIS Query Interface* é o objeto principal desse estudo. Com a realização dos testes, espera-se avaliar a implementação e identificar vantagens em relação à arquitetura distribuída sem suporte à elasticidade, e detectar ainda mais vantagens em relação à arquitetura não distribuída. Para realizar os testes com elasticidade do módulo *EPCIS Query Interface*, o *Eliot* foi configurado com o *threshold* inferior igual a 30 e o *threshold* superior igual a 70. Esses valores foram escolhidos com base nos estudos realizados no Capítulo 4, onde foi identificado que o sistema apresentava instabilidades quando o uso de CPU ultrapassava 80%. Portanto, para que o *middleware* IoT escalável trabalhasse dentro de uma faixa de operação garantida, foram definidos os *thresholds* descritos anteriormente.

Com esses limites, sempre que a previsão de carga *lp* ultrapassar o *threshold* superior, será alocada uma máquina virtual no *cluster* de *EPCIS Query Interface*, com o limite máximo de 5 máquinas. E sempre que a previsão de carga *lp* ficar abaixo do *threshold* inferior, será liberada uma máquina virtual no *cluster* de *EPCIS Query Interface*, com o limite mínimo de 1 máquina.

7.3 Resultados esperados

Ao aplicar os quatro perfis de leitura de dados (constante, ascendente, descendente e onda) sobre a arquitetura distribuída sem e com suporte à elasticidade, espera-se:

- Verificar se a arquitetura distribuída funciona na sua forma mais básica, ou seja, sem elasticidade, com apenas uma máquina no *cluster* de *EPCIS Query Interface*;
- Comparar a arquitetura distribuída básica em relação à não distribuída, para identificar diferenças no uso de recursos computacionais e tempo de resposta;
- Levantar os perfis de comportamento da arquitetura distribuída básica e identificar os pontos críticos por não estar utilizando o recurso de elasticidade;
- Levantar os perfis de comportamento da arquitetura distribuída completa, utilizando o recurso de elasticidade do *cluster* de *EPCIS Query Interface*;
- Comparar o desempenho da arquitetura distribuída sem utilizar o recurso de elasticidade em relação à mesma arquitetura, só que agora com o recurso de elasticidade habilitado.

7.4 Realização dos testes

Após toda a estrutura ter sido instalada e configurada no ambiente de nuvem, foi feito o primeiro teste para garantir o funcionamento do fluxo de escrita de dados, ou seja, assegurar que as etiquetas RFID estivessem sendo capturadas pelo ALE, processadas pela *Capturing Application* e gravadas no banco de dados pela *EPCIS Capturing Interface*. E de fato, consultando o banco de dados após alguns segundos, foi possível identificar registros gravados nas tabelas do EPCIS, confirmando que o fluxo de escrita estava funcionando perfeitamente na estrutura distribuída. Em seguida, foi feito o segundo teste para garantir o funcionamento do fluxo de leitura de dados, ou seja, validar o funcionamento da aplicação cliente e do *EPCIS Query Interface Wrapper*, oferecendo acesso de forma transparente ao *cluster* de máquinas que executam a *EPCIS Query Interface*. Esse segundo teste também resultou em sucesso, retornando para a aplicação cliente os códigos EPC das etiquetas que foram capturadas e gravadas no banco de dados anteriormente. Com o sucesso destes testes comprovando o correto funcionamento do *middleware* Fosstrak na estrutura distribuída e escalável, partiu-se para a aplicação dos cenários de teste descritos anteriormente.

7.4.1 Organização da apresentação dos resultados

Os resultados obtidos em cada cenário de teste estão organizados da seguinte maneira: primeiro, são apresentados os gráficos com o perfil do comportamento de cada arquitetura avaliada:

(a) não distribuída; (b) distribuída e não escalável; (c) distribuída e escalável. Em seguida, é possível visualizar um gráfico de barras (d) com a comparação de desempenho entre as arquiteturas, para os seguintes indicadores:

- **Fluxo médio de saída de rede (kbytes/segundo)**: informa o volume médio de dados que foi despachado pelo *EPCIS Query Interface Wrapper* durante a realização do teste;
- **Tempo de resposta médio por requisição (milissegundos)**: indica o tempo médio entre o envio das requisições para o *EPCIS Query Interface Wrapper* e o retorno das respostas;
- **Quantidade média de requisições por segundo**: representa o total de requisições realizadas durante o teste dividido pelo tempo total da execução.

Ainda nos gráficos com o perfil do comportamento de cada arquitetura, o eixo horizontal está expresso em segundos decorridos desde o início do teste. Já o eixo vertical foi convertido em valores percentuais para que todos os indicadores pudessem ser expressos no mesmo gráfico. O valor absoluto de cada indicador não é importante para entendimento do comportamento dos cenários estudados, mas sim a relação de variação entre eles.

Na sequência, é apresentada uma tabela com a comparação de desempenho dos testes em relação às arquiteturas avaliadas. Para o cálculo da variação percentual entre as arquiteturas, foi utilizada a seguinte fórmula: $var\_percentual = (\frac{arq\_comparada}{arq\_referencia} - 1) * 100$, onde $var\_percentual$ é a variação percentual do indicador, $arq\_comparada$ é a arquitetura que se pretende comparar e $arq\_referencia$ é a arquitetura referência para comparação.

Além dos indicadores citados anteriormente, essa tabela é complementada com as seguintes informações:

- **Quantidade média de CPUs utilizadas**: Informa a média da quantidade de CPUs utilizadas durante os testes;
- **Previsão de carga média (%)**: Indica a média da previsão de carga do *cluster EPCIS Query Interface* durante a realização dos testes.

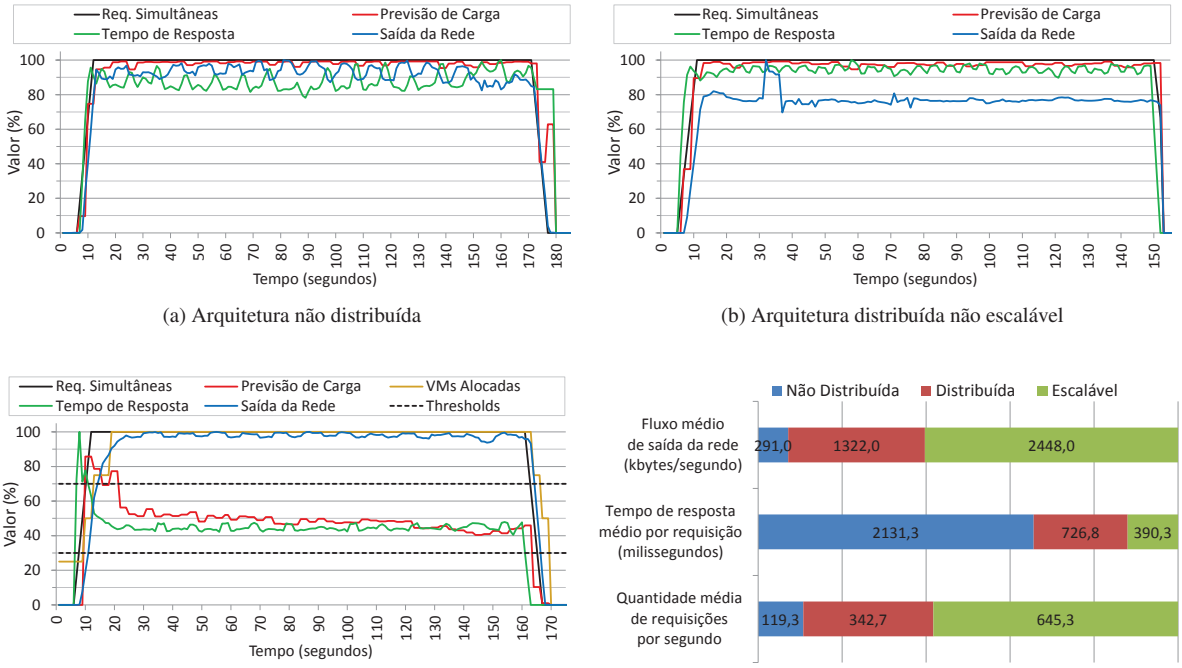
7.4.2 Carga Constante

As Figuras 21 (a), (b) e (c) apresentam o comportamento das avaliações de 256 requisições simultâneas e constantes. Para as arquiteturas não distribuída (a) e distribuída e não escalável (b), todos os indicadores se mantêm constantes com uma pequena flutuação. Porém, na arquitetura distribuída e escalável (c), pode-se identificar dois padrões de comportamento: (i) o tempo de resposta das requisições é diretamente relacionado à previsão de carga e (ii) o tráfego de saída da rede é diretamente relacionado à quantidade de máquinas virtuais alocadas. Também na Figura 21 (c) é possível identificar a alocação de máquinas virtuais quando a previsão de

carga ultrapassa o *threshold* superior, e a liberação de máquinas virtuais quando a previsão de carga fica abaixo do *threshold* inferior.

A Tabela 7 e a Figura 21 (d) apresentam os indicadores e comparativos de desempenho para os mesmos testes de 256 requisições simultâneas e constantes. Destaque para a redução do tempo de resposta em 81,7% da arquitetura distribuída e escalável em relação a não distribuída, e também redução de 65,9% da arquitetura distribuída e não escalável em relação a não distribuída. Entre as arquiteturas distribuída e escalável e distribuída e não escalável houve uma redução de 46,3% no tempo de resposta da primeira em relação à segunda.

Figura 21 – Avaliação com 256 requisições simultâneas constantes



(a) Arquitetura não distribuída

(b) Arquitetura distribuída não escalável

(c) Arquitetura distribuída escalável

(d) Comparação de desempenho entre as arquiteturas

Fonte: Elaborado pelo autor

Tabela 7 – Comparação de desempenho - Constante 256 threads

Arquitetura / Indicador	Não distribuída	Distribuída	Escalável	Variação (%) Distribuída / Não distribuída	Variação (%) Escalável / Não distribuída	Variação (%) Escalável / Distribuída
Quantidade média de CPUs utilizadas	1,0	1,0	3,9	0,0	290,0	290,0
Previsão de carga média (%)	96,3	96,3	50,3	0,0	-47,7	-47,7
Fluxo médio de saída da rede (kbytes/segundo)	291,0	1322,0	2448,0	354,3	741,2	85,2
Tempo de resposta médio por requisição (milissegundos)	2131,3	726,8	390,3	-65,9	-81,7	-46,3
Quantidade média de requisições por segundo	119,3	342,7	645,3	187,2	440,8	88,3

Fonte: Elaborada pelo autor

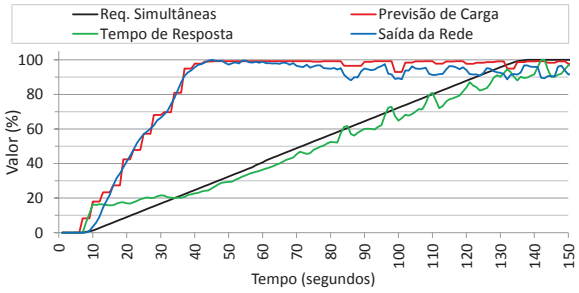
7.4.3 Carga Ascendente

As Figuras 22 (a), (b) e (c) apresentam o comportamento das avaliações com fluxo ascendente com limite final de 256 requisições simultâneas. Para as arquiteturas não distribuída (a) e distribuída e não escalável (b), os indicadores de tráfego de saída da rede e previsão de carga se mantêm com variações muito próximas. Esse mesmo comportamento de equivalência se percebe também entre o tempo de resposta das consultas e a quantidade de requisições simultâneas.

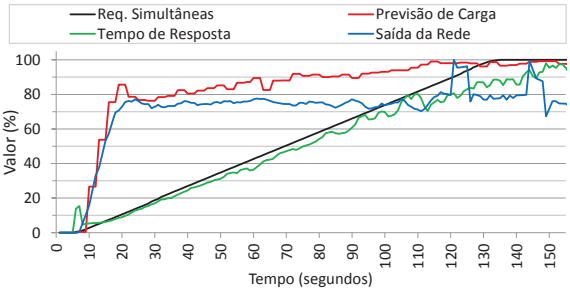
Porém, na arquitetura distribuída e escalável ilustrada na Figura 22 (c), o indicador de tráfego de saída da rede se desprende da linha de previsão de carga e passa a acompanhar a tendência da quantidade de máquinas virtuais alocadas. O indicador de previsão de carga passa a ser independente, e não dita mais o comportamento padrão das demais medições. Também nessa mesma Figura 22 (c) é possível identificar a alocação de máquinas virtuais quando a previsão de carga ultrapassa o *threshold* superior.

A Tabela 8 e a Figura 22 (d) apresentam os indicadores e comparativos de desempenho para os mesmos testes ascendentes com limite final de 256 requisições simultâneas. Destaque para a redução do tempo de resposta em 81,1% da arquitetura distribuída e escalável em relação a não distribuída, e também redução de 62,3% da arquitetura distribuída e não escalável em relação a não distribuída. Entre as arquiteturas distribuída e escalável e distribuída e não escalável houve uma redução de 49,8% no tempo de resposta da primeira em relação à segunda.

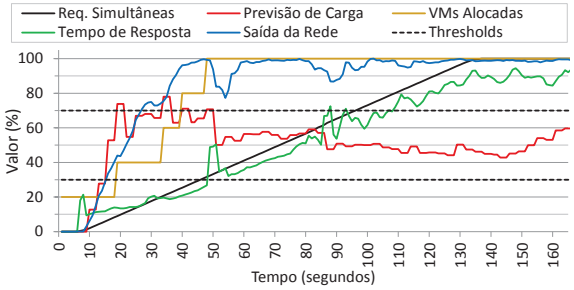
Figura 22 – Avaliação ascendente com máximo de 256 requisições simultâneas



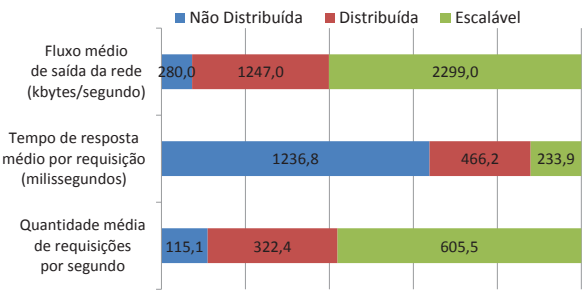
(a) Arquitetura não distribuída



(b) Arquitetura distribuída não escalável



(c) Arquitetura distribuída escalável



(d) Comparação de desempenho entre as arquiteturas

Fonte: Elaborado pelo autor

Tabela 8 – Comparação de desempenho - Ascendente 256 threads

Arquitetura / Indicador	Não distribuída	Distribuída	Escalável	Variação (%) Distribuída / Não distribuída	Variação (%) Escalável / Não distribuída	Variação (%) Escalável / Distribuída
Quantidade média de CPUs utilizadas	1,0	1,0	4,2	0,0	320,0	320,0
Previsão de carga média (%)	89,2	87,5	53,3	-2,0	-40,3	-39,1
Fluxo médio de saída da rede (kbytes/segundo)	280,0	1247,0	2299,0	345,4	721,1	84,4
Tempo de resposta médio por requisição (milissegundos)	1236,8	466,2	233,9	-62,3	-81,1	-49,8
Quantidade média de requisições por segundo	115,1	322,4	605,5	180,0	425,9	87,8

Fonte: Elaborada pelo autor

7.4.4 Carga Descendente

As Figuras 23 (a), (b) e (c) apresentam o comportamento das avaliações com fluxo descendente com quantidade inicial de 256 requisições simultâneas. Para as arquiteturas não distribuída (a) e distribuída e não escalável (b), os indicadores de tráfego de saída da rede e previsão de carga se mantêm com variações muito próximas. Esse mesmo comportamento de equivalência se percebe também entre o tempo de resposta das consultas e a quantidade de requisições simultâneas.

Porém, na arquitetura distribuída e escalável ilustrada na Figura 23 (c), o indicador de tráfego de saída da rede se desprende da linha de previsão de carga e passa a acompanhar a tendência da quantidade de máquinas virtuais alocadas. O indicador de previsão de carga passa a ser independente, e não dita mais o comportamento padrão das demais medições. Também nessa mesma Figura 23 (c) é possível identificar a liberação de máquinas virtuais quando a previsão de carga fica abaixo do *threshold* inferior.

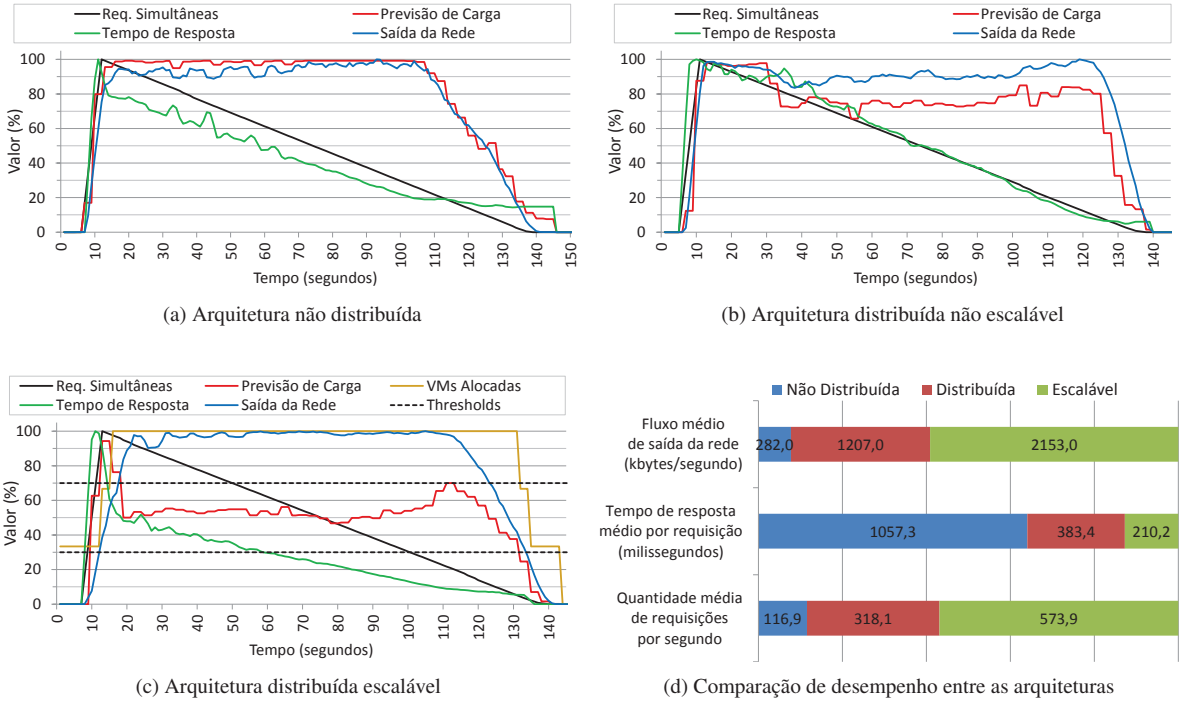
A Tabela 9 e a Figura 23 (d) apresentam os indicadores e comparativos de desempenho para os mesmos testes descendentes, iniciando em 256 requisições simultâneas. Destaque para a redução do tempo de resposta em 80,1% da arquitetura distribuída e escalável em relação a não distribuída, e também redução de 63,7% da arquitetura distribuída e não escalável em relação a não distribuída. Entre as arquiteturas distribuída e escalável e distribuída e não escalável houve uma redução de 45,2% no tempo de resposta da primeira em relação à segunda.

Tabela 9 – Comparação de desempenho - Descendente 256 threads

Arquitetura / Indicador	Não distribuída	Distribuída	Escalável	Variação (%) Distribuída / Não distribuída	Variação (%) Escalável / Não distribuída	Variação (%) Escalável / Distribuída
Quantidade média de CPUs utilizadas	1,0	1,0	2,9	0,0	190,0	190,0
Previsão de carga média (%)	87,0	75,3	53,2	-13,5	-38,8	-29,3
Fluxo médio de saída da rede (kbytes/segundo)	282,0	1207,0	2153,0	328,0	663,5	78,4
Tempo de resposta médio por requisição (milissegundos)	1057,3	383,4	210,2	-63,7	-80,1	-45,2
Quantidade média de requisições por segundo	116,9	318,1	573,9	172,2	391,0	80,4

Fonte: Elaborada pelo autor

Figura 23 – Avaliação descendente iniciando com 256 requisições simultâneas



(a) Arquitetura não distribuída

(b) Arquitetura distribuída não escalável

(c) Arquitetura distribuída escalável

(d) Comparação de desempenho entre as arquiteturas

Fonte: Elaborado pelo autor

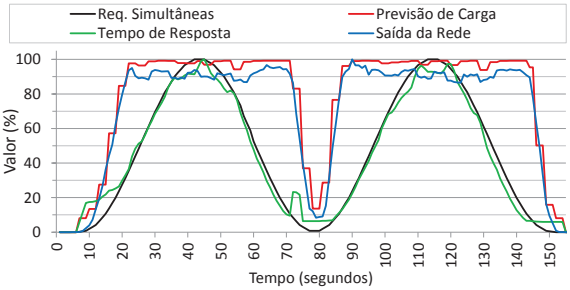
7.4.5 Carga em Onda

As Figuras 24 (a), (b) e (c) apresentam o comportamento das avaliações com fluxo de carga em onda, com pico de 256 requisições simultâneas. Para as arquiteturas não distribuída (a) e distribuída e não escalável (b), os indicadores de tráfego de saída da rede e previsão de carga se mantêm com variações muito próximas. Esse mesmo comportamento de equivalência se percebe também entre o tempo de resposta das consultas e a quantidade de requisições simultâneas.

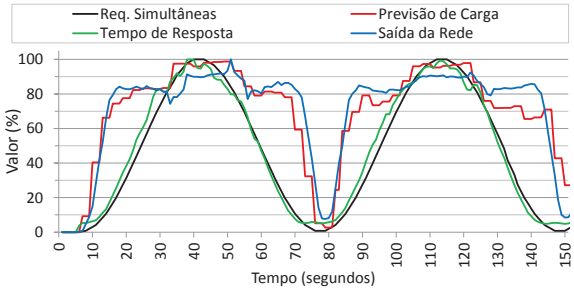
Porém, na arquitetura distribuída e escalável ilustrada na Figura 24 (c), o indicador de tráfego de saída da rede se desprende da linha de previsão de carga, e passa a acompanhar a tendência da quantidade de máquinas virtuais alocadas. O indicador de previsão de carga passa a ser independente, e não dita mais o comportamento padrão das demais medições. Também nessa mesma Figura 24 (c) é possível identificar a alocação de máquinas virtuais quando a previsão de carga ultrapassa o *threshold* superior. E quando a previsão de carga fica abaixo do *threshold* inferior, podemos visualizar a liberação de máquinas virtuais.

A Tabela 10 e a Figura 24 (d) apresentam os indicadores e comparativos de desempenho para os mesmos testes de comportamento de onda, com pico de 256 requisições simultâneas. Destaque para a redução do tempo de resposta em 80,0% da arquitetura distribuída e escalável em relação a não distribuída, e também redução de 62,5% da arquitetura distribuída e não escalável em relação a não distribuída. Entre as arquiteturas distribuída e escalável e distribuída e não escalável houve uma redução de 46,5% no tempo de resposta da primeira em relação à segunda.

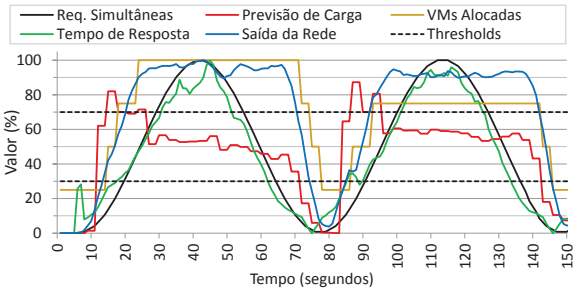
Figura 24 – Avaliação em onda com pico de 256 requisições simultâneas



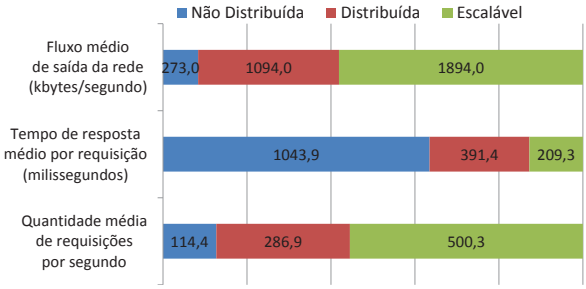
(a) Arquitetura não distribuída



(b) Arquitetura distribuída não escalável



(c) Arquitetura distribuída escalável



(d) Comparação de desempenho entre as arquiteturas

Fonte: Elaborado pelo autor

Tabela 10 – Comparação de desempenho - Onda 256 threads

Arquitetura / Indicador	Não distribuída	Distribuída	Escalável	Variação (%) Distribuída / Não distribuída	Variação (%) Escalável / Não distribuída	Variação (%) Escalável / Distribuída
Quantidade média de CPUs utilizadas	1,0	1,0	2,7	0,0	170,0	170,0
Previsão de carga média (%)	85,5	79,8	50,8	-6,7	-40,6	-36,3
Fluxo médio de saída da rede (kbytes/segundo)	273,0	1094,0	1894,0	300,7	593,8	73,1
Tempo de resposta médio por requisição (milissegundos)	1043,9	391,4	209,3	-62,5	-80,0	-46,5
Quantidade média de requisições por segundo	114,4	286,9	500,3	150,7	337,1	74,4

Fonte: Elaborada pelo autor

7.5 Avaliação geral dos resultados obtidos

Embora cada cenário de teste apresente suas particularidades quanto a uso de CPU, fluxo de saída da rede e tempo de resposta, é possível identificar padrões de comportamento comuns entre eles. Em todas as 3 arquiteturas avaliadas e para os 4 cenários, sempre o tempo de resposta das consultas está intimamente ligado à quantidade de requisições simultâneas em execução.

O uso de CPU também aumenta de acordo com a quantidade de requisições solicitadas. Além do processamento dos dados propriamente ditos, para comunicação entre os módulos está sendo utilizado HTTP sobre TCP/IP por questões de confiabilidade. Esses protocolos são executados em software, tanto pela aplicação quanto pelo sistema operacional. Ele faz cópias de memórias, controle de *checksum*, mantém temporizadores de retransmissão e janela deslizante.

Toda essa computação é feita em software, utilizando ciclos de CPU do processador. Dessa forma, quanto mais requisições forem solicitadas, mais fluxo de rede e consequentemente mais utilização de CPU.

Já o volume do tráfego de saída da rede é influenciado diretamente pela previsão de carga nas arquiteturas não distribuída e distribuída e não escalável. Porém, na arquitetura distribuída e escalável, o tráfego de saída da rede acompanha a quantidade de máquinas virtuais alocadas no *cluster EPCIS Query Interface*. Dessa forma, podemos identificar um comportamento único para a arquitetura *Eliot* implementada nesse estudo, distribuída e com escalabilidade do módulo *EPCIS Query Interface*.

7.5.1 Perfil de desempenho geral

Além de identificar os padrões de comportamento dos indicadores medidos nas respectivas arquiteturas computacionais, também foi possível identificar valores muito próximos de desempenho destes indicadores entre os testes realizados. A Tabela 11 apresenta as variações percentuais médias de cada indicador em relação às arquiteturas avaliadas. Para que os resultados fossem coerentes, os valores foram convertidos para percentual a cada teste realizado. No final, foram feitas as médias gerais sobre esses percentuais.

O fluxo de saída da rede teve um crescimento médio de 332,1% quando comparada a arquitetura distribuída e não escalável em relação a não distribuída. O ganho foi ainda maior entre a arquitetura distribuída e escalável em relação à não distribuída, apontando um acréscimo de 679,9%. Na arquitetura distribuída e escalável, o fluxo de saída da rede teve um aumento médio de 80,3% comparado com a distribuída e não escalável.

O tempo de resposta médio por requisição teve uma redução média de 63,6% quando comparada a arquitetura distribuída e não escalável em relação a não distribuída. A redução foi ainda maior entre a arquitetura distribuída e escalável em relação à não distribuída, apontando um decréscimo de 80,7%. Na arquitetura distribuída e escalável, o tempo de resposta médio por requisição teve uma diminuição média de 47,0% comparado com a distribuída e não escalável.

A quantidade média de requisições por segundo teve um crescimento médio de 172,5% quando comparada a arquitetura distribuída e não escalável em relação a não distribuída. O ganho foi ainda maior entre a arquitetura distribuída e escalável em relação à não distribuída, apontando um acréscimo de 398,7%. Na arquitetura distribuída e escalável, a quantidade média de requisições por segundo teve um aumento médio de 82,7% comparado com a distribuída e não escalável.

7.5.2 Ganho de desempenho com a escalabilidade

Avaliando unicamente a arquitetura distribuída e escalável, foi possível identificar diferentes perfis de desempenho aplicando a mesma carga de requisições, porém, com variadas quantida-

Tabela 11 – Comparação de desempenho entre todas as arquiteturas e todos os cenários de teste

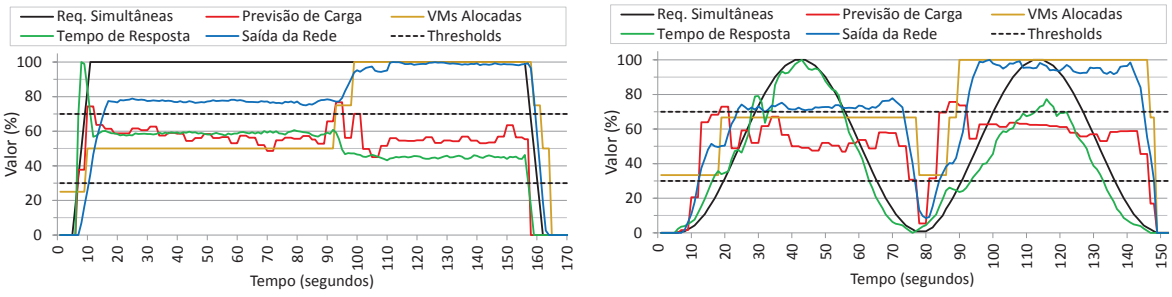
Arquitetura / Indicador	Variação (%) Distribuída / Não distribuída	Variação (%) Escalável / Não distribuída	Variação (%) Escalável / Distribuída
Fluxo médio de saída da rede	332,1	679,9	80,3
Tempo de resposta médio por requisição	-63,6	-80,7	-47,0
Quantidade média de requisições por segundo	172,5	398,7	82,7

Fonte: Elaborada pelo autor

des de máquinas virtuais alocadas no *cluster EPCIS Query Interface*.

A Figura 25 (a) representa um cenário de carga com 128 requisições simultâneas. Desde o início do teste até aproximadamente 90 segundos, estão alocadas apenas 2 máquinas virtuais *EPCIS Query Interface*. Isso está permitindo um fluxo de saída de rede médio de 1988 kilobytes por segundo, e um tempo de resposta médio por requisição de 230 milissegundos. Porém, logo após os 90 segundos ocorre um aumento na previsão de carga e são alocadas mais duas máquinas virtuais, atingindo um total de 4 *EPCIS Query Interfaces* disponíveis no *cluster*. Como agora há mais recursos computacionais disponíveis para processar as requisições, o desempenho do sistema melhora, obtendo um aumento de 28,1% no tráfego de saída da rede, atingindo 2547 kilobytes por segundo. O tempo de resposta também teve melhora de desempenho, sendo reduzido em 23,5%, passando para 176 milissegundos por requisição.

Figura 25 – Ganho de desempenho da estrutura escalável



(a) Ganho de desempenho com 128 requisições simultâneas e constantes

(b) Ganho de desempenho em onda com pico de 512 requisições simultâneas

Fonte: Elaborado pelo autor

Outro exemplo de ganho de desempenho pode ser visto na Figura 25 (b), que representa um cenário de teste com comportamento de onda, com pico de 512 requisições simultâneas. Durante o primeiro ciclo do teste, que vai até aproximadamente 80 segundos, foram alocadas apenas 2 máquinas virtuais *EPCIS Query Interface*. Isso permitiu um fluxo de saída de rede médio de 1805 kilobytes por segundo, e um tempo de resposta médio por requisição de 1120 milissegundos. Porém, no segundo ciclo que se inicia logo após os 80 segundos, foram alocadas 3 máquinas *EPCIS Query Interfaces* no *cluster*. Como agora também há mais recursos computacionais disponíveis para processar as requisições, o desempenho do sistema melhora,

obtendo um aumento médio de 30,5% no tráfego de saída da rede, atingindo 2355 kilobytes por segundo. O tempo de resposta também teve melhora de desempenho, sendo reduzido em 22,8%, passando para 865 milissegundos por requisição.

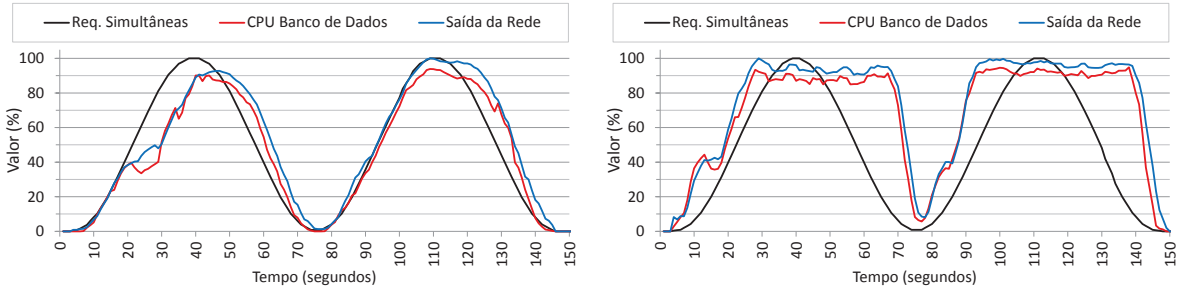
7.5.3 Limitações identificadas

Os resultados das avaliações indicam ganhos de desempenho e confiabilidade da arquitetura distribuída e escalável em relação às demais arquiteturas testadas. Mesmo assim, algumas limitações foram identificadas e são apresentadas a seguir.

7.5.3.1 Saturação da CPU do banco de dados

Durante os testes com grande volume de requisições simultâneas foi identificada saturação no indicador de fluxo de saída da rede. Mesmo com o aumento na quantidade de máquinas virtuais disponíveis no *cluster EPCIS Query Interface*, o tráfego de rede não aumentava proporcionalmente. Analisando os resultados de diversos cenários de teste, foi possível observar que quando o uso de CPU da máquina do banco de dados atingia um valor muito elevado, o fluxo de saída da rede estabilizava.

Figura 26 – Avaliação da saturação da CPU do banco de dados



(a) Sem saturação de CPU, fluxo de saída da rede normal em teste de onda com pico de 64 requisições simultâneas

(b) Com saturação de CPU, fluxo de saída da rede truncado em teste de onda com pico de 512 requisições simultâneas

Fonte: Elaborado pelo autor

Para o cenário de testes com comportamento de onda e pico de 64 requisições simultâneas, a Figura 26 (a) apresenta a curva de uso de CPU pelo banco de dados atingindo um valor máximo de aproximadamente 90%, durante um pequeno intervalo de tempo. Esse valor elevado de uso de CPU não representou limitação no fluxo de saída da rede. Porém, no cenário com comportamento de onda e pico de 512 requisições simultâneas ilustrado na Figura 26 (b), é possível identificar que o uso de CPU da máquina do banco de dados atingiu 90% e se manteve flutuando próximo desse valor durante um período significativo de tempo. Paralelamente, é possível verificar que o fluxo de saída da rede também ficou limitado, e só voltou a variar após a redução do uso de CPU.

Dessa forma, podemos identificar uma relação muito próxima entre o uso de CPU da máquina do banco de dados e o fluxo de saída da rede. Isso comprova que a recomendação de se utilizar um banco de dados NoSQL distribuído traria ainda mais benefícios ao *middleware* implementado, desde que esse banco oferecesse alto desempenho no retorno às requisições de consultas realizadas. Porém, mesmo que essa configuração de banco de dados não tenha sido utilizada, isso não invalida a arquitetura *Eliot* implementada, que apresentou muitas vantagens durante a realização dos testes.

7.5.3.2 *Timeout* das requisições

As avaliações realizadas inicialmente com a arquitetura não distribuída apresentadas no Capítulo 4 demonstraram que para quantidades elevadas de consultas simultâneas realizadas durante um longo período de tempo, o sistema ficava tão sobrecarregado que ocorriam *timeouts* nas requisições, e consequentemente o fluxo de saída da rede caía. Na avaliação dos resultados dos testes realizados na arquitetura distribuída, foi dada uma atenção especial para tentar identificar se esse problema havia sido eliminado, ou pelo menos minimizado.

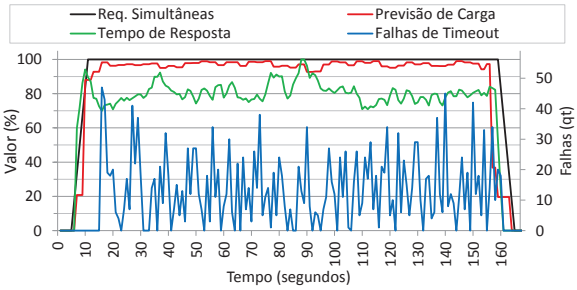
Conforme pode ser visualizado na Figura 27, os *timeouts* das requisições ainda acontecem, porém, foi possível identificar melhorias significativas. Para a arquitetura distribuída e não escalável, é possível identificar na Figura 27 (a) o comportamento do teste com 512 solicitações simultâneas e constantes. Nesse cenário, as ocorrências de *timeout* das requisições se distribuem igualmente desde o início até o fim do teste.

A Figura 27 (b) apresenta o resultado do mesmo cenário de teste de 512 requisições simultâneas e constantes, porém agora utilizando a arquitetura distribuída e escalável. Dessa vez, é possível identificar que as ocorrências de *timeout* das requisições se concentram no início do teste. A concentração ocorre dessa forma pois nesse momento existe apenas uma máquina virtual *EPCIS Query Interface* para servir o sistema. Como a previsão de carga sobe rapidamente, em questão de segundos já são alocadas mais 2 máquinas virtuais no *cluster*. Como consequência da disponibilidade de mais recursos computacionais, o sistema deixa de ficar sobrecarregado e consegue atender à alta demanda de requisições, sem que ocorram mais casos de *timeout*.

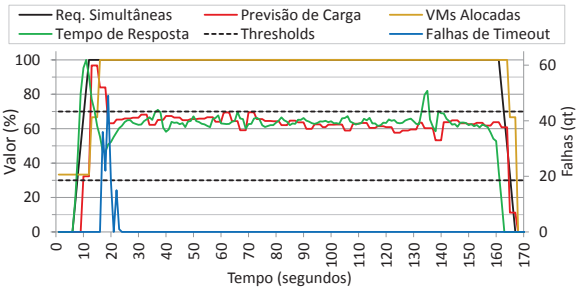
A Figura 27 (c) ilustra o comportamento do teste com carga descendente iniciando em 512 requisições simultâneas, executado na arquitetura distribuída e não escalável. Nesse cenário, as ocorrências de *timeout* se concentram no início do teste devido à alta concentração inicial de requisições simultâneas. Por se tratar de um teste com comportamento de carga descendente, com o passar do tempo a quantidade de requisições simultâneas vai diminuindo naturalmente, reduzindo também a sobrecarga do sistema. Passado pouco mais de 60 segundos do início do teste, o *middleware* consegue trabalhar sem problemas, eliminando por completo os casos de falha por *timeout*.

Para encerrar o estudo do cenário de carga descendente iniciando em 512 requisições simultâneas, a Figura 27 (d) apresenta o comportamento do teste realizado na arquitetura distribuída

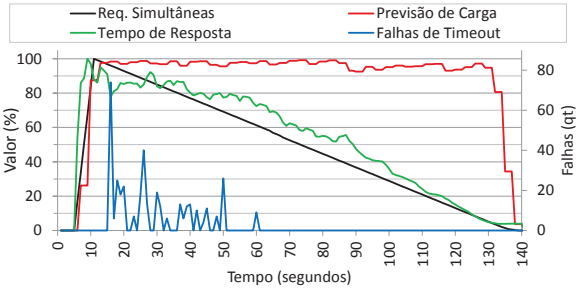
Figura 27 – Ocorrência de *timeout* durante testes de carga elevada



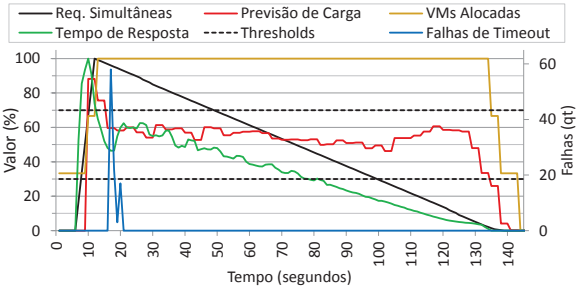
(a) Arquitetura distribuída e não escalável, com 512 requisições simultâneas e constantes



(b) Arquitetura distribuída e escalável, com 512 requisições simultâneas e constantes



(c) Arquitetura distribuída e não escalável, com carga descendente iniciando em 512 requisições simultâneas



(d) Arquitetura distribuída e escalável, com carga descendente iniciando em 512 requisições simultâneas

Fonte: Elaborado pelo autor

e escalável. Aqui, também é possível identificar que a ocorrência de *timeout* se concentra no início do teste, devido ao grande volume de requisições simultâneas. Nesse instante só existe uma máquina virtual alocada no *cluster EPCIS Query Interface*, e recai sobre ela toda a carga de processamento. Consequentemente, a previsão de carga sobe acima do *threshold* superior e duas máquinas virtuais *EPCIS Query Interface* são alocadas, distribuindo a carga entre elas e reduzindo a previsão de carga do *cluster*. Aliado à queda na quantidade de requisições simultâneas, característica do cenário de teste descendente, nessa mesma Figura 27 (d) é possível verificar que as falhas de *timeout* encerram logo após 20 segundos de teste.

Aprofundando a avaliação dos casos de *timeout*, foi possível identificar que o *Eliot* conseguiu adaptar a arquitetura distribuída e escalável para os cenários de teste em que a variação da demanda de requisições é pequena, que foi o caso dos testes ascendente e onda. Nesses cenários, a ocorrência de *timeout* foi completamente eliminada. Já para os cenários de teste constante e descendente, em que ocorre variação repentina na demanda de requisições simultâneas, houve redução mas não eliminação de falhas de *timeout*.

A Tabela 12 apresenta os indicadores de falha de *timeout* detalhados por cenário de teste e arquitetura utilizada. Os valores são apresentados em percentual de requisições com falha em relação ao total de consultas realizadas. Na arquitetura distrtibuída e não escalável, todos os cenários de teste com 512 requisições simultâneas apresentaram casos de tempo de resposta maior que 10 segundos, gerando *timeout* nas requisições. Já na arquitetura distrtibuída e escalável, foi

possível reduzir a quantidade de requisições que apresentaram *timeout* para os cenários de teste constante e descendente.

Tabela 12 – Percentual de requisições com tempo de resposta maior que 10 segundos

Arquitetura / Cenário de Teste	Não Escalável (%)	Escalável (%)	Variação (%)
Constante	3,57	0,16	-92,4
Ascendente	1,05	0,00	-100,0
Descendente	0,86	0,13	-73,0
Onda	1,37	0,00	-100,0

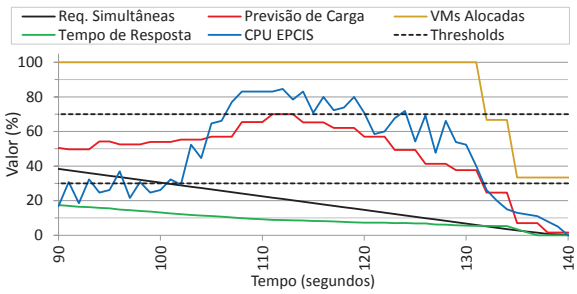
Fonte: Elaborada pelo autor

Para resolver ou minimizar as falhas de *timeout* geradas por aumento repentino e significativo na quantidade de requisições simultâneas, uma possibilidade seria reduzir o valor da configuração do *threshold* superior. Dessa forma, as máquinas *EPCIS Query Interface* seriam alocadas com uma previsão de carga menor, e o *cluster* trabalharia com menos sobrecarga de CPU. Outra possibilidade seria manter mais máquinas ativas no *cluster EPCIS Query Interface* mesmo com baixa ou nenhuma quantidade de requisições. Com isso, o *cluster* trabalharia subutilizado em alguns momentos, porém, estaria melhor preparado para atender uma demanda com oscilações mais fortes e imprevisíveis. Essas configurações são apenas sugestões de melhoria futura, não foi pensada a melhor forma de desenvolvê-las nem sequer foram implementadas ou testadas.

7.5.4 Avaliação da previsão de carga com método de *aging*

Em vez de avaliar simplesmente o uso instantâneo de CPU para a tomada de decisão sobre alocar ou liberar máquinas virtuais, a Figura 28 ilustra um caso em que se comprova a eficiência do método de *aging*. O exemplo ilustrado se refere ao teste descendente com 256 requisições simultâneas, já próximo do seu encerramento. Decorridos 100 segundos desde o início do teste, ainda estão alocadas 3 máquinas virtuais *EPCIS Query Interface*. Nesse instante, estão sendo realizadas aproximadamente 70 requisições simultâneas, cada uma atingindo um tempo médio de resposta na ordem de 100 milissegundos.

Figura 28 – Previsão de carga com método de *aging* em relação ao uso de CPU



Fonte: Elaborado pelo autor

Logo em seguida, percebe-se um aumento significativo no uso de CPU, e consequentemente,

aumento no indicador de previsão de carga. A quantidade de requisições simultâneas vem decaindo constantemente, e por consequência, o tempo de resposta médio por requisição também decresce. Próximo dos 110 segundos, o uso de CPU ultrapassa o *threshold* superior, porém, a previsão de carga não supera o limite estipulado em 70% devido à técnica de *aging*. Isso faz com que não seja alocada nova máquina virtual no *cluster* de *EPCIS Query Interface*. E realmente não seria necessária a alocação, pois o tempo de resposta nesse instante estava próximo dos 60 milissegundos, ou seja, o sistema estava mais eficiente que há instantes atrás.

Com o decorrer do teste, um pouco além dos 130 segundos, estão sendo realizadas aproximadamente 10 requisições simultâneas, com tempo de resposta em torno dos 40 milissegundos por requisição. Nesse instante, o uso de CPU cai drasticamente, fazendo com que a previsão de carga com o método de *aging* fique abaixo do *threshold* inferior, e consequentemente, liberando as máquinas virtuais previamente alocadas.

7.5.5 Consideração do tempo de resposta no cálculo da previsão de carga

Conforme apresentado no decorrer desse trabalho, o cálculo da previsão de carga das máquinas do *cluster EPCIS Query Interface* implementa a técnica de *aging*, que consiste no somatório ponderado de valores de uma série temporal. A ponderação é maior para os valores mais recentes, e os valores mais antigos possuem pesos menores.

A previsão de carga utilizada em todos os testes realizados consistiu na aplicação da técnica de *aging* unicamente sobre o indicador de uso de CPU das máquinas do *cluster EPCIS Query Interface*. Porém, ao interpretar e avaliar os diversos resultados obtidos nos experimentos, identificou-se a possibilidade de utilizar uma outra variável para compor a previsão de carga, e aprimorar a métrica para tomada de decisão sobre alocação e liberação de máquinas virtuais. Essa variável adicional seria o tempo de resposta desejado.

Avaliando também o tempo de resposta das requisições, seria possível adicionar um componente de qualidade de serviço ao *middleware* IoT. Conforme ilustrado anteriormente na Figura 25, o simples incremento na quantidade de máquinas *EPCIS Query Interface* disponíveis proporciona uma redução no tempo de resposta, com uma carga de requisições similar.

Dessa forma, caso a previsão de carga baseada em CPU ainda não tivesse atingido o *threshold* superior, seria possível alocar mais uma máquina virtual no *cluster* caso o tempo de resposta das requisições estivesse acima de um limite preestabelecido. A lógica inversa também é verdadeira, ou seja, embora a previsão de carga baseada em CPU ainda não estivesse abaixo do *threshold* inferior, seria possível liberar uma máquina virtual do *cluster* caso a medição do tempo de resposta das requisições estivesse abaixo de um limite preestabelecido. Baixo tempo de resposta significa que o *cluster* de máquinas *EPCIS Query Interface* está subutilizado, sendo possível liberar algumas máquinas por questões de economia. Essa nova métrica é apenas uma sugestão de melhoria futura, não foi pensada a melhor maneira de desenvolvê-la nem sequer foi implementada ou testada.

7.5.6 Alocação e liberação de máquinas virtuais na nuvem

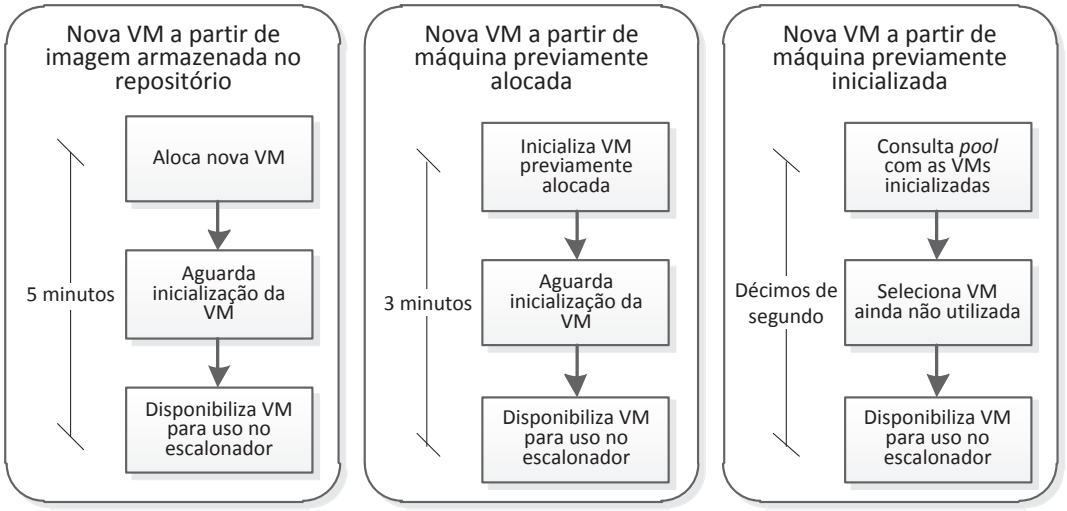
Quando a previsão de carga ultrapassa o valor do *threshold* superior, o *Eliot* deve alocar uma máquina virtual para distribuir a carga de processamento. Quando a previsão de carga fica abaixo do valor do *threshold* inferior, o *Eliot* deve liberar uma máquina virtual para reduzir a subutilização do *cluster* de máquinas *EPCIS Query Interface*. Estas rotinas foram implementadas e funcionaram perfeitamente, fazendo chamadas à API do EC2 da Amazon para alocação e liberação de máquinas virtuais.

Porém, o tempo necessário para que uma máquina EPCIS pudesse ser utilizada após ter dado o comando de inicialização era bastante elevado, em torno de 5 minutos. Esse tempo engloba alocar a máquina, copiar a imagem do repositório para o *hard-disk*, inicializar o sistema operacional e executar os aplicativos do *middleware* Fosstrak e do *Eliot*.

Foram feitas tentativas de reduzir esse tempo deixando as máquinas no estado *stopped*, ou seja, as máquinas já estavam pré-alocadas com as imagens carregadas no *hard-disk*, sendo necessário apenas inicializar o sistema operacional e executar os aplicativos do *middleware* Fosstrak e do *Eliot*, eliminando o tempo de cópia da imagem. Com isso, foi possível reduzir o tempo de inicialização para uma faixa entre 2 a 3 minutos.

Mesmo com essa redução, o tempo para se obter uma máquina EPCIS nova e em funcionamento ainda era bastante elevado em relação ao tempo de execução dos testes. Então, optou-se por manter sempre 5 máquinas virtuais em funcionamento no *cluster EPCIS Query Interface*, porém, somente disponibilizá-las para uso conforme as mesmas regras de alocação e liberação implementadas no *Eliot*. As rotinas de acesso à API do Amazon EC2 para alocação e liberação de máquinas virtuais foram substituídas por um *pool* de máquinas *EPCIS Query Interface* e por um controle interno para utilização delas por parte do escalonador.

Figura 29 – Obtenção de nova máquina virtual *EPCIS Query Interface*



Fonte: Elaborado pelo autor

No bloco mais à esquerda da Figura 29 é ilustrado o procedimento de alocação de uma máquina virtual a partir de uma imagem armazenada no repositório de imagens da Amazon. Esse diagrama representa o procedimento previsto na arquitetura original. No bloco central da Figura 29 é ilustrado o procedimento de obtenção de uma máquina virtual a partir de uma máquina previamente alocada, mas ainda não inicializada. Esse diagrama representa uma tentativa de obtenção de mais recursos computacionais em um tempo reduzido. Para finalizar, no bloco mais à direita da Figura 29 é ilustrado o procedimento de obtenção de uma máquina virtual a partir de um *pool* de máquinas previamente alocadas e inicializadas. Esse diagrama representa o ambiente real utilizado nas avaliações.

O procedimento utilizado na prática para obtenção de nova máquina virtual *EPCIS Query Interface* não é o mais correto do ponto de vista de otimização de recursos computacionais, porém, foi o mais adequado para a viabilização dos testes. O tempo despendido na inicialização das máquinas virtuais, que foi praticamente eliminado com a utilização do *pool*, não deve ser desconsiderado na arquitetura proposta. Seguramente esse tempo de espera influencia no resultado final de desempenho do *middleware* IoT escalável, porém, ele não interfere na validação do controle da elasticidade do sistema.

8 CONCLUSÃO

Este trabalho apresenta a arquitetura *Eliot* para *middlewares* IoT padrão EPCglobal, com foco na elasticidade do sistema. Também aborda a implementação, avaliação e discussão dos resultados obtidos. Inicialmente, foi feito um levantamento detalhado sobre os mais diversos *middlewares* IoT disponíveis no mercado. Nesse estudo, foi possível identificar as principais características de cada um, aplicações e tecnologias utilizadas, como gerenciam problemas de escalabilidade e balanceamento de carga.

Para justificar a arquitetura proposta, foram desenvolvidas uma metodologia e uma ferramenta de avaliação de desempenho e escalabilidade, chamada "*MIB: Micro Benchmark para Avaliação de Middlewares de Internet das Coisas*". Para implementar a nova arquitetura, foi utilizada a estrutura de nuvem computacional EC2 da Amazon, na qual foi instalado e configurado o *middleware* Fosstrak.

Para gerenciar a escalabilidade do *middleware* de forma transparente para as aplicações clientes, foi desenvolvido um conjunto de aplicativos intitulado "*Eliot Pro - Elasticity-Driven Internet of Things Prototype*", contendo módulos para monitoramento de recursos (CPU e rede), gerenciamento da escalabilidade (avaliação de desempenho, alocação e liberação de máquinas virtuais) e um balanceador de carga, realizando também o redirecionamento de requisições de consultas.

Para avaliar o desempenho do sistema IoT, um *micro benchmark* com o aplicativo *MIB* foi aplicado sobre o *middleware* Fosstrak, onde a metodologia proposta foi colocada em prática. Foram avaliadas as arquiteturas não distribuída, distribuída e não escalável e distribuída e escalável. Após uma análise detalhada dos dados registrados durante os testes da arquitetura não distribuída, foi possível identificar comportamentos padrão, limitações e gargalos do sistema. Esse estudo foi fundamental para o entendimento do modelo padrão EPCglobal e para o desenvolvimento da arquitetura proposta.

Analisando minuciosamente os dados registrados durante os testes nas arquiteturas distribuída e não escalável e distribuída e escalável, foi possível comprovar que o *Eliot* funciona corretamente. Além de identificar os comportamentos padrão ao executar o sistema na arquitetura distribuída, foi feito um estudo comparativo de desempenho entre as arquiteturas padrão, distribuída e escalável.

Comparando a arquitetura distribuída em relação a padrão, foi identificada uma redução no tempo de resposta das requisições de consulta ao EPCIS, e consequentemente, um aumento na quantidade processada dessas requisições por parte do *middleware*. Já o volume do tráfego de rede teve um incremento considerável, como reflexo da maior capacidade de atendimento às solicitações vindas das aplicações clientes. A quantidade de falhas por *timeout* de conexão detectadas na arquitetura distribuída também teve redução drástica comparada com as falhas registradas na arquitetura padrão.

Com a comprovação da redução das falhas aliada ao ganho de desempenho geral identi-

ficado durante as avaliações, pode-se afirmar que a arquitetura *Eliot*, distribuída e escalável, juntamente com o gerenciamento da elasticidade realizado pelo sistema *Eliot Pro*, respondem à questão de pesquisa levantada na Seção 1.2: "*Como seria uma arquitetura computacional e algoritmos para gerenciar a escalabilidade de um middleware de Internet das Coisas padrão EPCglobal, a fim de garantir o desempenho vindo da demanda dinâmica de aplicações e sensores RFID?*"

8.1 Contribuição científica

A primeira contribuição científica desse trabalho consiste em uma metodologia de avaliação para *middlewares* IoT padrão EPCglobal, chamada *"MIB: Micro Benchmark para Avaliação de Middlewares de Internet das Coisas"*. Essa metodologia permite avaliar o desempenho geral do *middleware*, tempos de resposta, uso de recursos computacionais como CPU e tráfego de rede e também identificar falhas de comunicação de dados ou respostas mal formadas.

A segunda contribuição científica é a arquitetura "*Eliot - Elasticity-Driven Internet of Things*", que permite que *middlewares* IoT padrão EPCglobal possam ser instalados de forma distribuída e escalável, explorando a elasticidade da computação em nuvem com alto desempenho.

8.2 Trabalhos futuros

Como sugestão de trabalhos futuros está a implementação completa da arquitetura *Eliot* inicialmente proposta, com a utilização de um banco de dados NoSQL distribuído para avaliação de desempenho. Quanto ao ambiente de nuvem computacional, além do EC2 da Amazon, também há espaço para desenvolvimento do suporte a APIs de outros gerenciadores, como por exemplo OpenNebula, OpenStack e CloudStack.

Outra melhoria sugerida seria utilizar mais indicadores de desempenho na definição da previsão de carga. Por exemplo, seria possível considerar também o tempo de resposta das requisições aliado ao uso de CPU para a tomada de decisão sobre alocação ou liberação de máquinas virtuais.

Para finalizar, poderia ser aprimorada a técnica do escalonador que determina qual máquina irá realizar o processamento das requisições de consultas ao EPCIS, avaliando o uso de CPU, tráfego de rede e tempo de resposta de cada uma das máquinas virtuais disponíveis.

8.3 Artigos publicados

Durante o desenvolvimento deste trabalho, o artigo científico *Future Directions for Providing Better IoT Infrastructure* (GOMES; RIGHI; COSTA, 2014) foi aprovado em um dos mais importantes congressos da área, o UbiComp edição 2014, classificado pela CAPES como A1. Outro artigo *Internet of Things Scalability: Analyzing the Bottlenecks and Proposing Alterna-*

tives (GOMES; ROSA RIGHI; COSTA, 2014) foi aprovado para o ICUMT edição 2014, com classificação B4. Isso confirma a qualidade da pesquisa realizada e o interesse da comunidade científica no desenvolvimento do assunto proposto.

REFERÊNCIAS

AHMED, N.; KUMAR, R.; FRENCH, R. S.; RAMACHANDRAN, U. RF2ID: a reliable middleware framework for rfid deployment. In: IPDPS, 2007. **Anais...** [S.l.: s.n.], 2007. p. 1–10.

AL-JAROODI, J.; AZIZ, J.; MOHAMED, N. Middleware for RFID Systems: an overview. **2009 33rd Annual IEEE International Computer Software and Applications Conference**, [S.l.], p. 154–159, 2009.

ALE, E. **Application Level Events (ALE) Standard - V 1.1.1**. Disponível em: <http://www.gs1.org/gsmp/kc/epcglobal/ale>. Acesso em: jun. 2014.

ANAGNOSTOPOULOS, A. P.; SOLDATOS, J. K.; MICHALAKOS, S. G. REFiLL: a lightweight programmable middleware platform for cost effective rfid application development. **Pervasive and Mobile Computing**, [S.l.], v. 5, n. 1, p. 49–63, 2009.

ASHTON, K. That Internet of Things Thing. **RFID Journal**, [S.l.], June 2009.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: a survey. **Computer networks**, [S.l.], v. 54, n. 15, p. 2787–2805, 2010.

BOSE, I.; PAL, R. Auto-ID: managing anything, anywhere, anytime in the supply chain. **Communications of the ACM**, [S.l.], v. 48, n. 8, p. 100–106, 2005.

CERVANTES, L. F.; LEE, Y.-S.; YANG, H.; LEE, J. A hybrid middleware for RFID-based parking management system using group communication in overlay networks. In: INTELLIGENT PERVASIVE COMPUTING, 2007. IPC. THE 2007 INTERNATIONAL CONFERENCE ON, 2007. **Anais...** [S.l.: s.n.], 2007. p. 521–526.

CHELLA, A.; COSSENTINO, M.; SABATUCCI, L. Tools and patterns in designing multi-agent systems with PASSI. **WSEAS Transactions on Communications**, [S.l.], v. 3, n. 1, p. 352–358, 2004.

CHEN, Y.; MINTUN, T. **Intelligent remote visual monitoring system for home health care service**. US Patent 5,553,609.

DOKOVSKY, N.; HALTEREN, A. V.; WIDYA, I. BANip: enabling remote healthcare monitoring with body area networks. **Scientific Engineering of Distributed ...**, [S.l.], p. 62–72, 2004.

EPCGLOBAL. **EPCglobal Knowledge Centre**. Disponível em: <http://www.gs1.org/gsmp/kc/epcglobal>. Acesso em: jun. 2014.

EPCIS, E. **EPCIS - EPC Information Services Standard - V 1.1**. Disponível em: <http://www.gs1.org/gsmp/kc/epcglobal/epcis>. Acesso em: jun. 2014.

FLOERKEMEIER, C.; LAMPE, M. RFID middleware design: addressing application requirements and rfid constraints. **Proceedings of the 2005 joint conference on ...**, [S.l.], p. 219–224, october 2005.

FLOERKEMEIER, C.; RODUNER, C.; LAMPE, M. RFID application development with the Accada middleware platform. **Systems Journal, IEEE**, [S.l.], v. 1, n. 2, p. 82–94, 2007.

FOSSTRAK. **Fosstrak - Open Source RFID Platform**. Disponível em: <https://code.google.com/p/fosstrak>. Acesso em: jun. 2014.

GALL, F. L.; CHEVILLARD, S. V.; GLUHAK, A.; XUELI, Z. Benchmarking Internet of Things Deployments in Smart Cities. **2013 27th International Conference on Advanced Information Networking and Applications Workshops**, Los Alamitos, CA, USA, v. 0, p. 1319–1324, 2013.

GILLAM, L.; LI, B.; JOHN, O.; TOMAR, A. P. et al. Fair benchmarking for cloud computing systems. **Journal of Cloud Computing: Advances, Systems and Applications**, [S.l.], v. 2, n. 1, p. 6, 2013.

GOMES, M. M.; RIGHI, R. d. R.; COSTA, C. A. da. Future directions for providing better IoT infrastructure. In: ACM INTERNATIONAL JOINT CONFERENCE ON PERVASIVE AND UBIQUITOUS COMPUTING: ADJUNCT PUBLICATION, 2014., 2014. **Proceedings...** [S.l.: s.n.], 2014. p. 51–54.

GOMES, M.; ROSA RIGHI, R. da; COSTA, C. A. da. Internet of things scalability: analyzing the bottlenecks and proposing alternatives. In: ULTRA MODERN TELECOMMUNICATIONS AND CONTROL SYSTEMS AND WORKSHOPS (ICUMT), 2014 6TH INTERNATIONAL CONGRESS ON, 2014. **Anais...** [S.l.: s.n.], 2014. p. 269–276.

GS1. **EPC Tag Data Standard Version 1.8**. Disponível em: <http://www.gs1.org/sites/default/files/docs/tds/TDS_1_8_Standard_20140203.pdf>. Acesso em: jun. 2014.

HANSEN, W.-R.; GILLERT, F. **RFID for the Optimization of Business Processes**. [S.l.]: John Wiley & Sons, 2008.

HOAG, J. E.; THOMPSON, C. W. Architecting RFID middleware. **Internet Computing, IEEE**, [S.l.], v. 10, n. 5, p. 88–92, 2006.

JEFFERY, S. R.; GAROFALAKIS, M.; FRANKLIN, M. J. Adaptive cleaning for RFID data streams. In: VERY LARGE DATA BASES, 32., 2006. **Proceedings...** [S.l.: s.n.], 2006. p. 163–174.

JIANG, S.; CAO, Y.; IYENGAR, S.; KURYLOSKI, P.; JAFARI, R.; XUE, Y.; BAJCSY, R.; WICKER, S. CareNet: an integrated wireless sensor networking environment for remote healthcare. In: ICST 3RD INTERNATIONAL CONFERENCE ON BODY AREA NETWORKS, 2008. **Proceedings...** [S.l.: s.n.], 2008. p. 9.

JOSE, J. I. S.; PASTOR, J. M.; ZANGRONIZ, R.; DIOS, J. J. de. RFID Tracking for urban transportation using EPCGlobal-based WebServices. In: ADVANCED INFORMATION NETWORKING AND APPLICATIONS WORKSHOPS (WAINA), 2013 27TH INTERNATIONAL CONFERENCE ON, 2013. **Anais...** [S.l.: s.n.], 2013. p. 1295–1300.

KABIR, A.; HONG, B.; RYU, W.; AHN, S. LIT Middleware: design and implementation of rfid middleware based on the epc network architecture. In: **Dynamics in Logistics**. [S.l.]: Springer, 2008. p. 221–229.

KANG, M.; KIM, D.-H. A Real-Time Distributed Architecture for RFID Push Service in Large-Scale EPCglobal Networks. In: KIM, T.-h.; ADELI, H.; CHO, H.-s.; GERVASI, O.; YAU, S.; KANG, B.-H.; VILLALBA, J. (Ed.). **Grid and Distributed Computing**. [S.l.]: Springer Berlin Heidelberg, 2011. p. 489–495. (Communications in Computer and Information Science, v. 261).

LANDT, J. The history of RFID. **Potentials, IEEE**, [S.l.], v. 24, n. 4, p. 8–11, Oct 2005.

LI, C.-M. An integrated software platform for RFID-enabled application development. **IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC'06)**, [S.l.], v. 1, 2006.

LI, M.; MORI, T.; NOGUCHI, H.; SHIMOSAKA, M.; SATO, T. Use of Active RFID and Environment-embedded Sensors for Indoor Object Location Estimation. In: INTERNATIONAL UNIVERSAL COMMUNICATION SYMPOSIUM, 3., 2009, New York, NY, USA. **Proceedings...** ACM, 2009. p. 93–99. (IUCS '09).

LIM, S.; OH, T. H.; CHOI, Y.; LAKSHMAN, T. Security Issues on Wireless Body Area Network for Remote Healthcare Monitoring. In: SENSOR NETWORKS, UBIQUITOUS, AND TRUSTWORTHY COMPUTING (SUTC), 2010 IEEE INTERNATIONAL CONFERENCE ON, 2010. **Anais...** [S.l.: s.n.], 2010. p. 327–332.

LIN, S.-S.; HUNG, M.-H.; TSAI, C.-L.; CHOU, L.-P. Development of an Ease-of-Use Remote Healthcare System Architecture Using RFID and Networking Technologies. **Journal of Medical Systems**, [S.l.], v. 36, n. 6, p. 3605–3619, 2012.

LUBRIN, E.; LAWRENCE, E.; NAVARRO, K. Wireless remote healthcare monitoring with Motes. In: MOBILE BUSINESS, 2005. ICMB 2005. INTERNATIONAL CONFERENCE ON, 2005. **Anais...** [S.l.: s.n.], 2005. p. 235–241.

MASSAWE, L. V.; AGHDASI, F.; KINYUA, J. The Development of a Multi-Agent Based Middleware for RFID Asset Management System Using the PASSI Methodology. In: INFORMATION TECHNOLOGY: NEW GENERATIONS, 2009. ITNG'09. SIXTH INTERNATIONAL CONFERENCE ON, 2009. **Anais...** [S.l.: s.n.], 2009. p. 1042–1048.

MELL, P.; GRANCE, T. The NIST definition of cloud computing. **National Institute of Standards and Technology**, [S.l.], sep 2011.

PALA, Z.; INANC, N. Smart parking applications using RFID technology. In: RFID EURASIA, 2007 1ST ANNUAL, 2007. **Anais...** [S.l.: s.n.], 2007. p. 1–3.

PASCUAL, J. A.; LORIDO-BOTRÁN, T.; MIGUEL-ALONSO, J.; LOZANO, J. A. Towards a Greener Cloud Infrastructure Management using Optimized Placement Policies. **Journal of Grid Computing**, [S.l.], p. 1–15, 2014.

PRABHU, B.; SU, X.; RAMAMURTHY, H.; CHU, C.-C.; GADH, R. WinRFID: a middleware for the enablement of radiofrequency identification (rfid)-based applications. **Mobile, wireless, and sensor networks: Technology, applications, and future directions**, [S.l.], p. 331–336, 2006.

RIGHI, R. D. R. Elasticidade em cloud computing: conceito, estado da arte e novos desafios. **Revista Brasileira de Computação Aplicada**, [S.l.], v. 5, n. 2, p. 2–17, nov 2013.

SUNG, W.-T.; HSU, C.-C. IOT system environmental monitoring using IPSO weight factor estimation. **Sensor Review**, [S.l.], v. 33, n. 3, p. 246–256, 2013.

TANENBAUM, A. S. **Computer Networks, 5-th Edition**. [S.l.]: Prentice Hall, 2010.

VEEN, J. S. van der; WAAIJ, B. van der; MEIJER, R. J. Sensor data storage performance: sql or nosql, physical or virtual. In: CLOUD COMPUTING (CLOUD), 2012 IEEE 5TH INTERNATIONAL CONFERENCE ON, 2012. **Anais...** [S.l.: s.n.], 2012. p. 431–438.

WANT, R. An introduction to RFID technology. **Pervasive Computing, IEEE**, [S.l.], v. 5, n. 1, p. 25–33, Jan 2006.

WHITE, G.; GARDINER, G.; PRABHAKAR, G. P.; ABD RAZAK, A. A comparison of barcoding and RFID technologies in practice. **Journal of information, information technology and organizations**, [S.l.], v. 2, p. 119–132, 2007.

WOOLDRIDGEY, M.; CIANCARINI, P. Agent-oriented software engineering: the state of the art. In: AGENT-ORIENTED SOFTWARE ENGINEERING, 2001. **Anais...** [S.l.: s.n.], 2001. p. 1–28.

XU, L.; HE, W.; LI, S. Internet of Things in Industries: a survey. **IEEE Transactions on Industrial Informatics**, [S.l.], v. 3203, n. c, p. 1–1, 2014.

ZHAN, R. et al. Comparative Analysis and Simulation of Load Balancing Scheduling Algorithm Based on Cloud Resource. In: INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND INFORMATION TECHNOLOGY, 2014. **Proceedings...** [S.l.: s.n.], 2014. p. 449–456.