Programa Interdisciplinar de Pós-Graduação em

Computação Aplicada

Mestrado Acadêmico

Ivam Guilherme Wendt

PSElastic - Um Modelo de Elasticidade Multinível para *Brokers Publish/Subscribe*.

São Leopoldo, 2017

Ivam Guilherme Wendt

PSELASTIC - UM MODELO DE ELASTICIDADE MULTINÍVEL PARA *BROKERS PUBLISH/SUBSCRIBE*

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. Rodrigo da Rosa Righi

Co-orientador:
Prof. Dr. Jorge Luis Victoria Barbosa

São Leopoldo
2017

W473p Wendt, Ivam Guilherme.
PSElastic : um modelo de elasticidade multinível para Brokers Publish/Subscribe / Ivam Guilherme Wendt. – 2017.
82 f. : il. ; 30 cm.

Dissertação (mestrado) – Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, 2017.
"Orientador: Prof. Dr. Rodrigo da Rosa Righi ; co-orientador: Prof. Dr. Jorge Luis Victoria Barbosa."

1. Elasticidade. 2. Publish/Subscribe. 3. Broker. 4. Internet das coisas. I. Título.

CDU 004

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

Ivam Guilherme Wendt

PSElastic - Um Modelo de Elasticidade Multinível para *Brokers Publish/Subscribe*.

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 23 de Fevereiro de 2017.

BANCA EXAMINADORA

Prof. Dr. Rodrigo da Rosa Righi – UNISINOS

Prof. Dr. Jorge Luis Victória Barbosa – UNISINOS

Prof. Dr. Cristiano André da Costa – UNISINOS

Prof. Dr. Antônio Marcos Alberti – INATEL

Prof. Dr. Rodrigo da Rosa Righi

Visto e permitida a impressão
São Leopoldo,

Prof. Dr. Sandro José Rigo
Coordenador PPG em Computação Aplicada

Aos meus familiares.

As vezes a Vida é como andar de bicicleta:

No início não sabemos nada, com o tempo vamos aprendendo como tudo funciona, ganhando confiança, andando cada vez mais rápido e sem medos.

Quando menos se espera, aparece um buraco em um lugar inesperado e nos derruba.

Agora, mesmo com as feridas ardendo, a bicicleta torta e sem toda aquela confiança; levantamos e tentamos seguir nosso caminho, pois sabemos que não tem como voltar no tempo e que ficarmos parados não nos levará a lugar nenhum.
— Ivam Guilherme Wendt

AGRADECIMENTOS

Inicialmente agradeço a meus pais Paulo Rogério Wendt e Adreani Elena Wendt, pela minha criação, educação, apoio e incentivo. Sempre serão exemplos de dedicação, persistência e honestidade. Citando Sir Isaac Newton, "Se eu vi mais longe do que outros, é porque eu estava sobre os ombros de gigantes.".

Agradeço a meu orientador Rodrigo da Rosa Righi e co-orientador Jorge Luis Victoria Barbosa, que me guiaram na busca dos conhecimentos necessários para o desenvolvimento deste trabalho, conduzindo com maestria cada passo realizado. Agradeço, também, ao órgão de pesquisa CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) que tornou possível a realização deste trabalho. E aos meus amigos e colegas, sempre presentes e apoiando meus passos em cada momento durante a realização deste trabalho.

E por fim, gostaria de fazer um agradecimento em especial à minha noiva Vanessa Rodrigues Bender, que esta sempre ao meu lado, superando as dificuldades da vida, compartilhando os momentos felizes, e principalmente, por ser um ponto de referência em minha vida.

RESUMO

A Internet das Coisas (IoT - *Internet of Things*) é considerada a próxima grande evolução na era da computação após a Internet. Na Internet das Coisas muitos objetos que nos rodeiam serão conectados em redes e se comunicarão entre sí sem auxílio ou intervenção humana. Implementações de grande escala têm potencial para criar um enorme tráfego agregado. Muitas vezes *Brokers* ou Intermediários, são utilizados para possibilitar esta comunicação *machine-to-machine* (M2M), utilizando o modelo de comunicação *Publish/-Subscribe* que possui diversas bibliotecas particulares para uso na Internet das Coisas. Alguns trabalhos desenvolvidos pela comunidade visam tratar o desempenho no ambiente de *Brokers Publish/Subscribe*, no entanto impõem modificações e restrições quanto aos *Brokers* e arquitetura utilizada. Neste cenário, visando atender a necessidade de escalabilidade citada, é proposto um modelo de elasticidade chamado PSElastic. PSElastic fornece um modelo de elasticidade multinível para *Brokers Publish/Subscribe*, fazendo uso da elasticidade provida pela Computação em Nuvem, sem afetar o desempenho do sistema e sem a necessidade de investimentos ou ajustes por parte dos clientes e aplicações que fazem uso deste modelo. O modelo oferece a elasticidade de forma automática, reativa e multinível, onde multinível refere-se aos níveis de *Broker* e de Orquestrador do modelo. Buscando avaliar o modelo PSElastic foram definidos três cenários diferentes para a execução dos testes: Cenário 1, sem elasticidade; Cenário 2, elasticidade na camada de *Brokers* e Cenário 3, elasticidade multinível. Diferentes limites de réplicas de máquinas virtuais foram utilizados, assim como combinações de *thresholds* para avaliar o Tempo de Execução, quantidade de Mensagens por Segundo tratadas e a Eficiência na utilização dos recursos computacionais. PSElastic apresentou excelentes resultados ao reduzir em até 81,2% o tempo necessário para o tratamento de uma carga de dados e aumentar o *throughput* do ambiente com *Broker* Mosquitto de 54,95 mensagens por segundo (Mps) para 235,29 Mps e de 38,02 Mps para 202,02 Mps no ambiente com *Broker* RabbitMQ.

Palavras-chave: Elasticidade. Publish/Subscribe. Broker. Internet das Coisas.

ABSTRACT

The Internet of Things (IoT) is considered the next big evolution in the computing era after the Internet. In Internet of Things many objects around us will be connected in networks and will communicate among themselves without assistance or human intervention. Large scale implementations has potential to create a huge aggregate traffic. Often brokers or intermediaries, are used to enable this machine-to-machine (M2M) communication using the Publish/Subscribe communication model that has several private libraries for use in Internet of Things. Some works developed by the community aim to treat performance in Publish/Subscribe brokers environment, however impose modifications and restrictions regarding brokers and architecture used. In this scenario, aiming to fulfill the need for cited scalability, we propose a elasticity model called PSElastic. The model provides an model with multilevel elasticity for Publish/Subscribe Brokers, making use of the elasticity promoted by cloud computing, not affecting system performance and without the need for investments or adjustments by customers and applications that make use of this model. PSElastic offers elasticity in the following manners: automatic, reactive and uses the multilevel elasticity, where multilevel refers to Broker level and Orchestrator model. In order to evaluate the model PSElastic, three different scenarios for execution of tests were defined: Scenario 1, without elasticity; Scenario 2, elasticity in the layer of Broker and Scenario 3, multilevel elasticity. Different limits of replica of virtual machines were used, as well as thresholds to evaluate Execution Time, number of Messages per Second handled and Efficiency in the use of computational resources. PSElastic showed excellent results by reducing the time needed to handle a data load by up to 81.2% and increase the throughput of the environment with Mosquitto Broker from 54.95 messages per second (Mps) to 235.29 Mps and from 38.02 Mps to 202.02 Mps in the environment with RabbitMQ Broker.

Keywords: Elasticity. Publish/Subscribe. Broker. Internet of Things.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A Internet das Coisas (IoT - *Internet of Things*) é considerada a próxima grande evolução na era da computação após a Internet (GUBBI et al., 2013). Na China, no ano de 2014 haviam pelo menos 9 bilhões de dispositivos interconectados, e espera-se que este número chegue a 24 bilhões até 2020 (CHEN et al., 2014). No paradigma da Internet das Coisas, muitos objetos que nos rodeiam serão conectados em redes e se comunicarão entre si sem auxílio ou intervenção humana. Uma grande quantidade de sensores inteligentes será incorporada em uma variedade de aplicações. Esta incorporação de sensores resultará em grandes quantidades de dados que precisarão ser armazenados, processados e apresentados de forma eficiente e facilmente interpretável pelo usuário.

A Computação em Nuvem pode fornecer a infraestrutura virtual para tal utilidade integrando dispositivos de monitoramento, armazenamento, ferramentas de análise, plataformas de visualização e acesso de clientes. Entre as principais características da Computação em Nuvem está a elasticidade. Esta característica possibilita aos usuários alterar a capacidade da nuvem a qualquer momento de forma dinâmica, aumentando ou diminuindo a disponibilidade de recursos de acordo com a demanda ou qualidade de serviço desejada (LORIDO-BOTRÁN; MIGUEL-ALONSO; LOZANO, 2012; RAVEENDRAN; BICER; AGRAWAL, 2011). Através do princípio de provisionamento sob demanda e do modelo *pay-as-you-go* ou pague pelo que usa, o interesse na capacidade de elasticidade está relacionado aos benefícios que ela pode proporcionar: melhor desempenho, melhor utilização de recursos e redução de custos. Outra vantagem é a possibilidade de alocar uma pequena quantidade de recursos no início da execução da aplicação (SAH; JOSHI, 2014). Nesse contexto, uma possibilidade é a exploração da elasticidade multinível, que corresponde a aplicação de elasticidade em mais de uma camada ou nível na arquitetura do sistema, por exemplo *Front-End*, Processamento em *Batch*, Banco de Dados, etc.

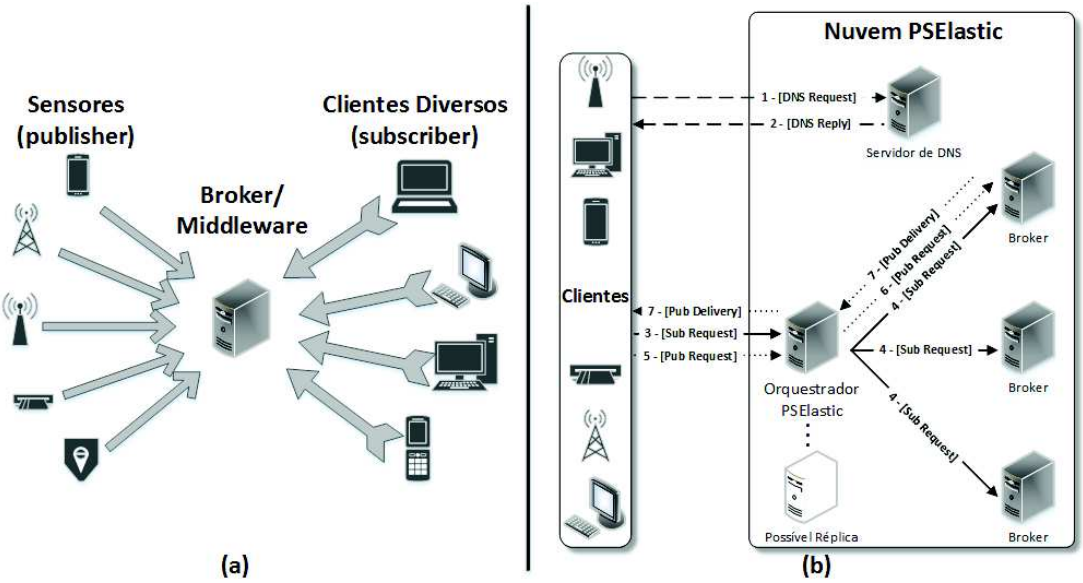
Considerando o cenário de Internet das Coisas, o sistema pode precisar aumentar a quantidade de recursos computacionais disponíveis, ampliando a capacidade de processamento da nuvem, com o objetivo de atender mais requisições simultaneamente sem afetar o nível de serviço desejado. Caso a demanda do sistema reduza, a quantidade de recursos também pode ser reduzida. Isso resulta em um menor custo e menor consumo de energia (ROSA RIGHI et al., 2015). A elasticidade em nuvem é largamente explorada em arquiteturas cliente servidor, como vídeo sob demanda, lojas online, *e-governance* e *Web Services* (RAVEENDRAN; BICER; AGRAWAL, 2011). Uma estratégia típica neste contexto, é a elasticidade horizontal da infraestrutura de nuvem com replicação de instâncias de máquinas virtuais (WARD; BARKER, 2014).

Segundo Khan et al. (2012) os dispositivos pertencentes à Internet das Coisas implementam diferentes tipos de serviços para aplicações voltadas à saúde inteligente (*smart health*), agricultura inteligente (*smart farming*), cidade inteligente (*smart city*), entre ou-

tras. Estes serviços utilizam uma camada chamada de Camada de *Middleware* que pode receber as informações, armazena-las, processa-las e tomar decisões automáticas com base nos resultados. Muitas vezes *Brokers* ou Intermediários, são utilizados nesta camada para possibilitar comunicação *machine-to-machine* (M2M), onde um dispositivo pode se comunicar com outro para enviar alertas e informações, utilizando o modelo de comunicação *Publish/Subscribe* (KHAN et al., 2012). A abordagem **(a)** da Figura 1 apresenta o modelo tradicional de implementação de *Brokers Publish/Subscribe*, sem elasticidade e sem balanceamento de carga, onde todas as mensagens são tratadas pelo mesmo *Broker*, tornando-o um ponto único de falhas e um limitador do desempenho da arquitetura.

Neste contexto, existem recursos de elasticidade em nuvem, no entanto os mesmos são pouco empregados no contexto de Internet das Coisas (TRAN; SKHIRI; ZIMÁNYI, 2011; LI et al., 2011; BARAZZUTTI et al., 2014). Tendo em vista que a Camada de *Middleware* centraliza grande parte da comunicação entre dispositivos (KHAN et al., 2012), pesquisas aplicando elasticidade a esta camada são pertinentes para o desenvolvimento da Internet das Coisas. A camada citada será abordada com mais detalhes na Seção 2.2 do Capítulo 2.

Figura 1: Modelo Tradicional e Modelo PSElastic.



Fonte: elaborado pelo autor.

1.1 Questão de Pesquisa

O modelo desenvolvido nesta dissertação busca responder a seguinte questão de pesquisa:

Como seria um modelo de elasticidade multinível em nuvem para Brokers de aplicações Publish/Subscribe, que seja transparente para usuários, aplicações e o Broker escolhido?

A ideia desta dissertação é desenvolver um modelo capaz de fornecer elasticidade para *Brokers Publish/Subscribe* de forma transparente para usuários e aplicações. A elasticidade foi utilizada devido a suas vantagens relacionadas a custos e desempenho. A palavra-chave multinível, deve-se ao fato de que a elasticidade foi utiliza em dois níveis da arquitetura. Sendo um nível na camada de *Brokers*, para permitir a escalabilidade no processamento e tratamento de mensagens; e outro nível no Orquestrador PSElastic para evitar que o mesmo se torne o gargalo do modelo. O modelo deve ser transparente no sentido de não necessitar de ajustes ou desenvolvimentos adicionais nas aplicações; e agnóstico quanto ao *Broker*, no sentido de suportar a utilização de qualquer *Broker* na camada de *Middleware*. Desta forma, o modelo propõe uma arquitetura com elasticidade para *Brokers Publish/Subscribe*, aproveitando o melhor da Computação em Nuvem, pagando pelo que usa, sem afetar o desempenho do sistema e sem a necessidade de investimentos ou ajustes por parte dos clientes e aplicações que utilizam esta infraestrutura.

A abordagem **(b)** apresentada na Figura 1 corresponde a uma visão de alto nível do modelo PSElastic, onde a carga de processamento é balanceada entre vários *Brokers Publish/Subscribe* e a arquitetura é capaz de ampliar sua capacidade de processamento utilizando elasticidade automática. A visão de alto nível do modelo PSElastic será retomada na Seção 4.1 do Capítulo 4.

1.2 Objetivos

Este trabalho tem como objetivo geral desenvolver o modelo chamado **PSElastic**; um modelo de elasticidade multinível para *Brokers* de aplicações que utilizam comunicação *Publish/Subscribe*. PSElastic busca proporcionar elasticidade automática para *Brokers Publish/Subscribe* de forma transparente para usuários, aplicações e implementações de *Brokers*. Para tal, fornece um arcabouço (*framework*) com um Gerenciador de operações de elasticidade e um Orquestrador, que trata o encaminhamento de mensagens sem a necessidade de modificações ou adaptações nas aplicações.

Para atingir o objetivo geral, foram definidos os seguintes objetivos específicos:

(i) Realizar um estudo sobre o estado da arte relacionado a modelos e técnicas que visam tratar desempenho em *Publish/Subscribe*;

(ii) Comparar os trabalhos encontrados neste estudo com base em critérios definidos e encontrar lacunas;

(iii) Desenvolver um modelo de elasticidade que preencha as lacunas encontradas e um protótipo deste modelo para a realização de testes;

(iv) Avaliar o protótipo dentro do ambiente universitário, com base em um conjunto de dados coletados de uma aplicação de Internet das Coisas;

(v) Analisar os resultados obtidos e comparar estes resultados com as diferentes configurações de elasticidade.

1.3 Organização do Texto

Esta dissertação está organizada em sete capítulos. Após a introdução apresentada no Capítulo 1, os conceitos fundamentais para a compreensão do restante do trabalho e da pesquisa proposta são apresentados no Capítulo 2; o Capítulo 3 apresenta uma lista com trabalhos relacionados ao tema desta pesquisa, bem como um comparativo e identificação das lacunas encontradas; o modelo proposto para preencher as lacunas identificadas no Capítulo 3 e que responde a questão de pesquisa é apresentado no Capítulo 4; a metodologia utilizada para avaliação é abordada no Capítulo 5 e os resultados obtidos utilizando o modelo PSElastic são abordados no Capítulo 6; por fim, o Capítulo 7 apresenta as considerações finais do presente trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo apresentar conceitos considerados fundamentais para o desenvolvimento deste trabalho. A Seção 2.1 apresenta o modelo de comunicação *Publish/Subscribe*. A Seção 2.2 aborda alguns conceitos e definições de Internet das Coisas (IoT - *Internet of Things*), assim como o método de comunicação *machine-to-machine* (M2M) e os meios utilizados para possibilitar esta comunicação. Os principais conceitos relacionados à Computação em Nuvem são apresentados na Seção 2.3, incluindo os modelos de serviço, modelos de implantação e elasticidade. Os principais conceitos relacionados a Balanceamento de Carga são abordados na Seção 2.4. Por fim, são realizadas considerações parciais sobre os assuntos tratados neste capítulo na Seção 2.5.

2.1 Modelo *Publish/Subscribe*

Serviços *Publish/Subscribe* provaram ser uma solução adequada para a construção de grandes infraestruturas de comunicações, graças às suas propriedades de comunicação assíncrona e de desacoplamento, garantindo dissociação entre *publishers* e *subscribers* no espaço e no tempo (EUGSTER et al., 2003) sem requerer qualquer ligação anterior (MARTINS; DUARTE, 2010). Na verdade, essas propriedades removem a necessidade de estabelecer explicitamente todas as dependências entre as entidades que interagem, de modo a tornar a infraestrutura de comunicação, mais escalável, flexível e de fácil manutenção. Por esta razão, serviços *Publish/Subscribe* são atualmente utilizados como o componente principal na construção de vários projetos industriais buscando inovação em infraestruturas de grande escala empregadas a diferentes domínios de aplicação (como em aplicações de saúde (ESPOSITO; CIAMPI; PIETRO, 2014), gestão do tráfego aéreo (CINQUE; MARTINO; ESPOSITO, 2012), ou a produção industrial (DELAMER; LASTRA, 2006)).

Serviços *Publish/Subscribe* representam uma solução de *Middleware*, onde as mensagens são entregues para destinatários interessados de uma maneira centrada em dados, ou seja, sem indicar o endereço desses destinatários, mas com base em alguns atributos das mensagens e um estilo anônimo de interação. Desta forma, os emissores e receptores não necessitam estar cientes de suas respectivas identidades (EUGSTER et al., 2003; ESPOSITO; COTRONEO; RUSSO, 2013).

Essas arquiteturas são baseadas no padrão de *design* orientado a objetos chamado *Subject-Observer* (ou Assunto-Observador) (GAMMA et al., 1994): um Ator, chamado de Observador, é automaticamente notificado e atualizado sobre as mudanças que ocorrem no estado de outro Ator, chamado de Assunto. O Observador deve informar os eventos que está interessado por meio de uma chamada de *Subscription* (ou assinatura). Então, se o Assunto detecta a ocorrência de um desses eventos, ele notifica o Observador invocando um

dos seus métodos. Tal padrão de projeto apresenta os seguintes benefícios: (i) acoplamento mínimo entre os Assuntos e os Observadores, (ii) apoio à comunicação *multicast*, e (iii) de filtragem de notificações com base no interesse do Observador. No entanto, ele também tem a desvantagem de cada Assunto ter que manter uma lista de Observadores, com os seus interesses relacionados, e chamá-los se necessário (ESPOSITO; CIAMPI, 2015).

Para melhorar a dissociação e, assim, escalabilidade, o padrão Assunto-Observador foi combinado com o Mediador (GAMMA et al., 1994), dando origem ao padrão de Notificação de Eventos (RIEHLE, 1996): o processo de gerenciamento e notificação de Observadores não é colocado localmente com os Assuntos, mas centralizado em uma entidade terceira mediadora, por vezes referido como o Serviço de Notificação. Uma forte dissociação entre aplicações é obtida utilizando Invocações Implícitas (GARLAN; NOTKIN, 1991) dos métodos dos Observadores quando os eventos ocorrem no Assunto. Este padrão, combinado com o padrão reator, que despacha notificações recebidas para vários manipuladores, e com o padrão proator (SCHMIDT et al., 2013), que utiliza operações assíncronas, tornou possível um novo modelo de interação que foi formalizado no padrão *Publish/Subscribe* (EUGSTER et al., 2003).

Tal padrão define a arquitetura que caracteriza um serviço típico de *Publish/Subscribe* (ou publicação/assinatura) e é composto por três entidades principais: (i) um *publisher* (ou editor) que divulga as notificações geradas pelos eventos que ocorrem na aplicação; (ii) um *subscriber* (ou assinante) que recebe as notificações de seu interesse e as passa para a aplicação; e (iii) o serviço de notificação que interliga os *publishers* e *subscribers*, e tem a função de armazenar as notificações de entrada e de roteá-las para os *subscribers* interessados. O serviço de notificação permite uma dissociação entre a produção e consumo de eventos, o que resulta em um aumento de escalabilidade (SCHLOSSNAGLE, 2006) uma vez que todas as dependências explícitas são removidas (EUGSTER et al., 2003).

Sistemas *Publish/Subscribe* podem ser baseados em tópicos ou baseados em conteúdo. Em sistemas baseados em tópico, cada espaço de informação está associada com um tópico, grupo ou canal assim chamado. *Publishers* publicam as suas notificações para um tópico. *Subscribers* assinam os tópicos que estão interessados. Para fins de eficiência e conveniência, alguns sistemas permitem aos *subscribers* fornecer um filtro de semântica sobre o conteúdo das notificações. Sistemas com esta funcionalidade extra são designados como baseados em conteúdo. Na presença de um grande número de publicações, os filtros podem facilitar a vida dos *subscribers* e tornar o sistema mais eficiente por não entregar notificações inúteis (MARTINS; DUARTE, 2010).

No âmbito de Internet das Coisas, *Brokers* são os componentes principais para mediar todas as trocas de mensagens entre *publishers* e *subscribers*, de certa forma eles são um repositório central de todos os dados publicados. As aplicações de Internet das Coisas, são tipicamente idealizadas para que gerem e publiquem dados de forma esporádica. No

entanto, implementações de grande escala têm potencial para criar um enorme tráfego com destino aos *Brokers*, causando congestionamento e reduzindo assim a taxa de transferência (mensagens por segundo) do ambiente (XU; MAHENDRAN; RADHAKRISHNAN, 2016).

Um conjunto de protocolos é adotado por serviços *Publish/Subscribe* para realizar a distribuição das notificações dos *publishers* para os *subscribers* interessados (ESPOSITO; CIAMPI, 2015). Na verdade, esses serviços podem ser realizados, explorando as capacidades de comunicação *unicast* tradicional de protocolos a nível da camada de transporte, tais como UDP ou TCP; mas eles também podem usar *multicast* ou *broadcast* a nível da camada de rede para atingir este objetivo. Dentro da literatura atual, várias soluções têm sido propostas para possibilitar esta comunicação 1 para N (BALDONI et al., 2009; CARZANIGA; PAPALINI; WOLF, 2011; MARTINS; DUARTE, 2010). Estas fazem parte de pesquisas relacionadas a grupos de comunicação.

2.2 Internet das Coisas (IoT)

Weiser (1999) previu que as pessoas iriam interagir com centenas de computadores em vez de apenas um, e computadores seriam incorporados no ambiente ou em objetos portáteis da vida cotidiana. Esta visão indica desaparecimento do computador como uma entidade distinta e estabelece o conceito de computação ubíqua. Em Computação Ubíqua as interfaces para interagir com dispositivos de computação se tornam mais amigáveis, naturais, e nós somos capazes de nos manter conectados devido à existência de vários tipos de redes em diversos locais e contextos, sem perceber.

Após duas décadas de evolução significativa em termos de software, hardware e especialmente em redes de telecomunicações, surgiu o conceito de Internet das Coisas. Internet das Coisas pode ser vista como uma combinação dos conceitos de computação ubíqua (WEISER, 1999), computação pervasiva (SATYANARAYANAN, 2001), "coisas que pensam"(GERSHENFELD, 1999) e "ambiente inteligente"(FERGUSON, 2002). Todos esses conceitos têm em comum a visão de que haverá um mundo com objetos físicos de nossa vida cotidiana, equipados com uma lógica digital, sensores e uma capacidade de conexão com a Internet (FLEISCH; SARMA; THIESSE, 2009). A ideia básica da Internet das Coisas é que praticamente todas as coisas físicas em todo o mundo podem se tornar computadores que se conectam à Internet, então as coisas começam a ter algumas características de pequenos computadores, tornando-se em seguida, objetos inteligentes (FLEISCH, 2010).

O campo de pesquisa da Internet das Coisas engloba várias áreas do conhecimento, como computação, engenharia, telecomunicações, *design*, economia e negócios. Os campos de pesquisa incluem, por exemplo, *Radio Frequency Identification* (RFID), comunicação *machine-to-machine* (M2M), *machine-type communication* (MTC), sensor sem fio e redes de atuadores (WSAN), computação ubíqua e web-de-coisas (WoT) (ATZORI; IERA;

MORABITO, 2010). Há também estudos relacionados com Internet das Coisas e interação humano-computador (HCI), a fim de verificar as tendências e oportunidades no uso da Internet das Coisas no *design* do produto (KORESHOFF; ROBERTSON; LEONG, 2013).

É interessante notar que até agora os esforços de computação ubíqua, pervasiva, tangível ou "vestível"(*wearable*) consistem em um único dispositivo conectado a uma única fonte de dados, enquanto que a Internet das Coisas promove o conceito de um ecossistema onde um dispositivo é conectado a muitos outros objetos (KORESHOFF; ROBERTSON; LEONG, 2013). Internet das Coisas difere da Internet como a conhecemos hoje, pois permite que as coisas, lugares e mundo físico gerem dados automaticamente (FLEISCH, 2010). De acordo com Mattern e Floerkemeier (2010), Internet das Coisas não é o resultado de uma única tecnologia, é a combinação de várias tecnologias de desenvolvimento complementares que oferecem recursos, que ajudam a preencher a lacuna entre o mundo virtual e o físico.

Entre as definições da Tabela 1 a mais próxima para coincidir com a atual visão da Internet das Coisas, incluindo objetos inteligentes é a proposta por Fleisch (2010). Ele afirma que praticamente todas as coisas físicas em todo o mundo podem se tornar computadores que se conectam à Internet, tornando-se em seguida, objetos inteligentes.

Segundo Khan et al. (2012), a arquitetura de Internet das Coisas geralmente é dividida em cinco camadas, como mostrado na Figura 2. Estas camadas são descritas abaixo:

- Camada de Percepção (*Perception Layer*): A camada de percepção é também conhecida como "camada dos dispositivos". Consiste de objetos físicos e sensores. Os sensores podem ser RFID, código de barras 2D, ou sensor infravermelho, dependendo método de identificação de objetos. Esta camada basicamente lida com a identificação e coleta de informações específicas de objetos. Dependendo do tipo de sensores, a informação pode ser sobre a localização, temperatura, orientação, movimento, vibração, aceleração, humidade, alterações químicas no ar, etc. A informação recolhida é então passada para a camada de rede para a transmissão segura até o sistema de processamento da informações.

- Camada de Rede (*Network Layer*): A camada de rede também pode ser chamada de "camada de transmissão". Esta camada transfere de forma segura a informação a partir de dispositivos sensores até o sistema de processamento de informações. O meio de transmissão pode ser com ou sem fio e tecnologia pode ser 3G, UMTS, Wi-Fi, Bluetooth, infravermelho, ZigBee, etc, dependendo dos dispositivos sensores. Assim, a camada de rede transfere as informações da camada percepção para a Camada de *Middleware*.

- Camada de *Middleware* (*Middleware Layer*): Os dispositivos pertencentes à Internet das Coisas implementam diferentes tipos de serviços. Esta camada pode receber

Tabela 1: Principais definições da Internet das Coisas de diferentes áreas do conhecimento.

Autor	Conceito	Área do Conhecimento
Union (2005)	Internet das Coisas engloba a **conexão de objetos e aparelhos de uso diário** em todos os tipos de redes (intranets, *peer-to-peer* e Internet).	Telecomunicações
Fleisch (2010)	**Todas as coisas físicas em todo o mundo podem se tornar computadores que se conectam à Internet**, então as coisas começam a ter algumas características de pequenos computadores, tornando-se em seguida, objetos inteligentes.	Administração e economia
Mattern e Floerkemeier (2010)	Internet das Coisas é uma visão em que a **Internet se estende para o mundo real através de objetos** do cotidiano.	Computação pervasiva
Tan e Wang (2010)	Internet das Coisas será a próxima geração da Internet, onde **todos os objetos estão ligados**. Ela representa uma nova era de computação ubíqua.	Computação
Atzori, Iera e Morabito (2010)	Internet das Coisas é um novo paradigma onde existe a **presença sútil de objetos e "coisas" inteligentes em torno de nós** — como etiquetas RFID, sensores, atuadores, telefones móveis, etc. — que estarão **prontos para interagir e cooperar** uns com os outros a fim de alcançar um objetivo específico.	Rede de Computadores
Koreshoff, Robertson e Leong (2013)	Internet das Coisas refere-se a uma visão mais ampla, onde "coisas" são objetos, lugares e ambientes do quotidiano. **Todas estas "coisas" são interligadas umas com as outras através da Internet**.	Interação humano-computador

Fonte: elaborado pelo autor.

as informações da camada de rede e armazenar no banco de dados, executar o processamento da informação e tomar a decisão automática com base nos resultados. Esta camada é muito utilizada para possibilitar a comunicação *machine-to-machine* (M2M), onde um dispositivo pode se comunicar com outro para enviar alertas e informações, utilizando o modelo de comunicação *Publish/Subscribe*. Os *Brokers*, citados na Seção 2.1, pertencem a esta camada.

- Camada de Aplicação (*Application Layer*): Esta camada proporciona uma gestão global do aplicativo com base nas informações de objetos processadas na camada de *Middleware*. As aplicações implementadas pela Internet das Coisas podem ser voltadas à saúde inteligente (*smart health*), agricultura inteligente (*smart farming*), casa inteligente (*smart home*), cidade inteligente (*smart city*), transporte inteligente (*intelligent transportation*), etc.

- Camada de negócio (*Business Layer*): Esta camada é responsável pela gestão do sistema global de Internet das Coisas, incluindo as aplicações e serviços. Com base nos dados recebidos da camada de aplicação ela constrói modelos de negócio, gráficos, fluxogramas, etc. O verdadeiro sucesso da tecnologia da Internet das Coisas também depende dos bons modelos de negócios. Com base na análise dos resultados, esta camada irá ajudar a determinar as futuras ações e estratégias de negócios.

Figura 2: Arquitetura em camadas para Internet das Coisas.

Fonte: traduzido livremente de Khan et al. (2012).

Tendo em vista que a camada de *Middleware* é a responsável pelo processamento das informações utilizadas pela aplicação (KHAN et al., 2012), ela pode afetar o desempenho

do sistema como um todo caso seja sobrecarregada. Sendo assim, é pertinente que sejam desenvolvidos trabalhos que tratem a elasticidade de recursos computacionais nesta camada.

2.3 Computação em Nuvem

O *National Institute of Standards and Technology* (NIST) (MELL; GRANCE, 2011) define a Computação em Nuvem como um modelo que permite, de maneira conveniente e sob demanda, o acesso através de rede de computadores a um conjunto configurável de recursos computacionais (redes, servidores, armazenamento, aplicações, e serviços) que podem rapidamente ser provisionados e liberados com esforço mínimo de gerenciamento ou iteração do provedor de serviço. Segundo o NIST, este modelo de computação promove disponibilidade e é composto por (MELL; GRANCE, 2011):

- Cinco características essenciais: Auto-previsionamento sob demanda, acesso amplo a rede, agrupamento de recursos, rápida elasticidade e medição de serviço;
- Três modelos de serviço: Infraestrutura como Serviço ou *Infrastructure as a Service* (IaaS), Plataforma como Serviço ou *Platform as a Service* (PaaS) e Software como Serviço ou *Software as a Service* (SaaS);
- Quatro modelos de implantação: Nuvem Privada, Nuvem Pública, Nuvem Híbrida e Nuvem Comunitária.

2.3.1 Modelos de Serviço

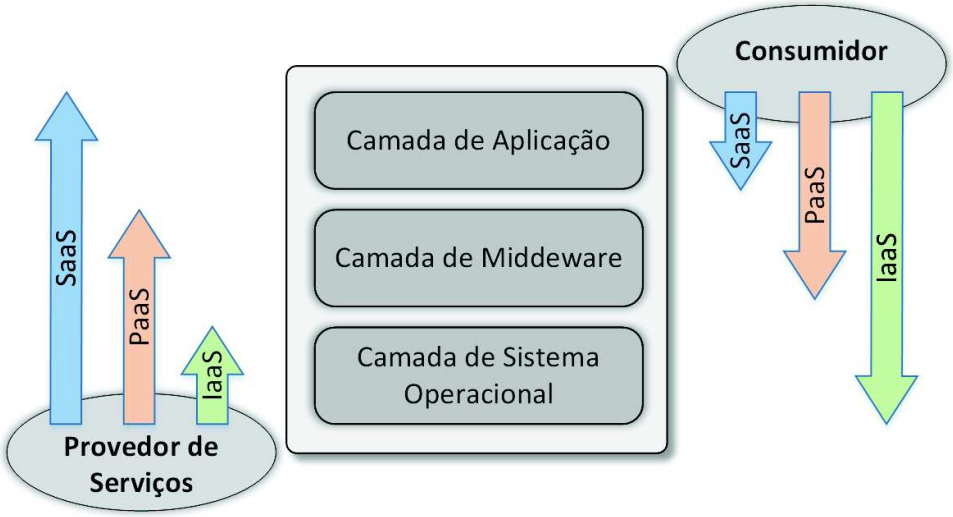
Em Computação em Nuvem, os recursos oferecidos podem ser tanto recursos de *hardware* (memória e CPU (*Central Processing Unit*)), quanto recursos de *software*. Os modelos de serviço oferecidos por uma nuvem variam de acordo com o tipo de recursos que são disponibilizados. A Figura 3 exibe os modelos, que são classificados em três categorias: (MELL; GRANCE, 2011; PUTHAL et al., 2015)

- **Infraestrutura como Serviço (IaaS)**: Neste modelo, são oferecidos ao usuário recursos fundamentais de hardware como CPU, memória, armazenamento e rede, permitindo ao usuário a implantação de *softwares* conforme sua necessidade. É realizada a virtualização dos recursos computacionais através de um *hypervisor* e os mesmos são disponibilizados ao usuário através de máquinas virtuais. A virtualização de recursos é a característica chave deste modelo permitindo ao usuário executar seu próprio sistema operacional e gerenciar máquinas virtuais que rodam na infraestrutura oferecida. Conforme a Figura 3, é de responsabilidade do Provedor de Serviços de Nuvem apenas a primeira camada referente ao sistema operacional. As demais camadas ficam sob responsabilidade do Consumidor;

- **Plataforma como Serviço (PaaS)**: Aqui, o usuário não têm o controle do sistema operacional e *softwares* instalados nas máquinas virtuais. Essa tarefa é de responsabilidade do provedor da nuvem, que fica encarregado de administrar as máquinas virtuais além da infraestrutura. Deste modo, é oferecido ao usuário um conjunto de recursos como sistema operacional, linguagens de programação, plataformas para desenvolvimento de *softwares* e ferramentas para implantação de aplicações. O usuário deste modelo tem acesso a estas plataformas e ferramentas com o objetivo do desenvolvimento e implantação de aplicações;

- **Software como Serviço (SaaS)**: Neste modelo, é oferecido ao usuário o acesso a aplicações que executam sobre uma infraestrutura de nuvem. O usuário deste modelo utiliza apenas a aplicação, sem se envolver com a infraestrutura e os serviços necessários para que a aplicação execute. A implementação e administração de toda a infraestrutura é de responsabilidade do provedor.

Figura 3: Escopo de controle entre provedor e consumidor conforme o modelo de serviço.



Fonte: traduzido livremente de Liu et al. (2011).

2.3.2 Modelos de Implantação

Uma infraestrutura de nuvem pode ser criada e mantida pela própria organização ou ser contratada através de um provedor de nuvem. O modo como uma infraestrutura de nuvem é disponibilizada e mantida nos leva a quatro modelos de implantação diferentes (MELL; GRANCE, 2011; PUTHAL et al., 2015):

- **Nuvem Privada**: Em uma nuvem privada, toda a infraestrutura de computação é disponibilizada para uso exclusivo de uma única organização. Neste modelo, a

organização pode ser dona da infraestrutura ou também terceirizá-la, porém toda a infraestrutura é alocada apenas para a organização, sem haver o compartilhamento dos recursos físicos e lógicos com outros usuários;

- **Nuvem Pública**: Neste modelo, a infraestrutura é provisionada para o uso de qualquer usuário. Toda a infraestrutura é compartilhada por diferentes usuários, podendo ser cobrada ou de graça. Neste modelo, existe a figura de um provedor que mantém a infraestrutura e oferece diferentes serviços para diferentes usuários;

- **Nuvem Comunitária**: Neste modelo, a infraestrutura é provisionada exclusivamente para um grupo específico de usuários que compartilham os mesmos requisitos e interesses. Como exemplo, podemos citar um grupo de universidades que mantém suas nuvens privadas e decidem uni-las em uma única nuvem comunitária;

- **Nuvem Híbrida**: Por fim, uma nuvem híbrida é formada pela combinação de duas ou mais diferentes infraestruturas dos demais modelos de implantação. As infraestruturas que compõe uma nuvem híbrida continuam distintas, porém elas são unidas por protocolos específicos que permitem a portabilidade da aplicação entre uma infraestrutura e outra.

2.3.3 Elasticidade

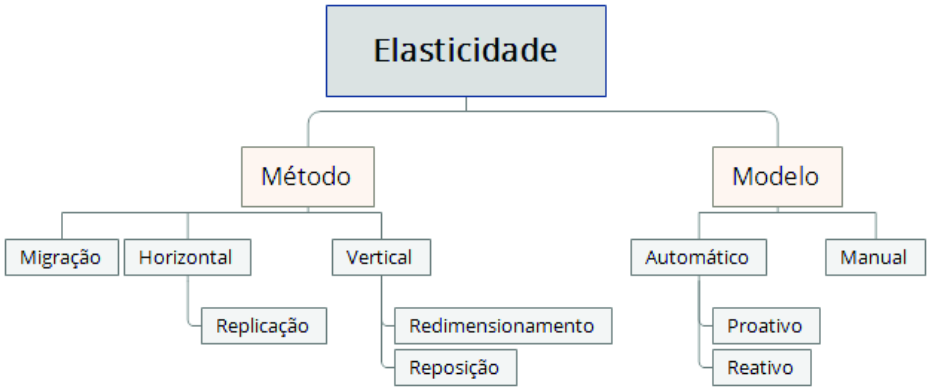
A elasticidade de recursos é uma das funcionalidades mais importantes da Computação em Nuvem, através dela é possível aumentar ou diminuir a quantidade de recursos disponíveis sem a interrupção do serviço para atender variações de demanda (COUTINHO et al., 2014; LEHRIG; EIKERLING; BECKER, 2015). Existem diferentes métodos para oferecer a elasticidade, cada um definindo diferentes operações para adição ou remoção de recursos. Além dos métodos, existem diferentes modelos utilizados, que definem a maneira como as operações são realizadas. A Figura 4 apresenta os diferentes métodos e modelos de elasticidade, bem como as operações que podem ser realizadas.

Conforme Galante e Bona (2012), existem três métodos diferentes para oferecer a elasticidade (GALANTE; BONA, 2012; COUTINHO et al., 2014; RODRIGUES, 2015):

- Elasticidade **horizontal**: Consiste em adicionar ou remover instâncias de componentes do ambiente virtual do usuário. Essas instâncias podem ser máquinas virtuais, contêineres, ou módulos de aplicações que são replicados no ambiente de nuvem e disponibilizados para o usuário. O método mais utilizado atualmente para oferecer a elasticidade horizontal é através da **replicação** de máquinas virtuais;

- Elasticidade **vertical**: Consiste em adicionar ou remover recursos (processamento, memória, rede e armazenamento) das instâncias virtuais em execução. Neste caso,

Figura 4: Métodos e modelos de elasticidade de recursos. O modelo apresenta como operações podem ser disparadas enquanto o método apresenta os tipos de operações que podem ser realizadas.



Fonte: adaptado de Galante e Bona (2012); Coutinho et al. (2014).

máquinas virtuais têm seus atributos alterados podendo ter seus recursos físicos aumentados ou diminuídos. Existem duas técnicas principais de elasticidade vertical: **redimensionamento** e **reposição**. A primeira, e mais comumente utilizada, consiste em alterar a quantidade de recursos físicos disponíveis para a máquina virtual. Porém, atualmente poucos sistemas operacionais suportam que a quantidade de recursos seja alterada sem a necessidade de reinicialização da máquina virtual. Quanto a técnica de reposição, esta consiste em substituir um recurso físico por outro com maior ou menor poder computacional. Neste caso um servidor que comporta máquinas virtuais pode ser substituído por um mais ou menos potente dependendo da necessidade;

- **Migração**: recursos como máquinas virtuais e aplicações podem ser migrados entre os servidores que compõe a infraestrutura da nuvem. Essa técnica pode ser utilizada em substituição a técnica de reposição. Através de migrações, máquinas virtuais podem ser transferidas de uma máquina para outra com maior ou menor poder computacional. Além disso, migrações são pertinentes para a consolidação de servidores visando diminuir o consumo energético com menos equipamentos ligados.

Os diferentes métodos de elasticidade são classificados conforme as interações necessárias dos usuários para a execução das operações de elasticidade. Galante e Bona (2012) apresentam políticas que definem dois modelos de elasticidade diferentes (GALANTE; BONA, 2012):

- **Manual**: Neste modelo o usuário realiza todas as operações de gerenciamento de recursos manualmente através de ferramentas de linha de comando, interface gráfica ou *Application Programming Interfaces* (APIs) disponibilizadas pelo provedor ou sistema de nuvem.

- **Automático**: a elasticidade é realizada através de ferramentas de monitoramento do sistema de nuvem ou por alguma aplicação específica. Neste caso, decisões de elasticidade são tomadas através de algoritmos de avaliação de métricas monitoradas. Apesar de ser chamado automático, o usuário ainda tem a tarefa de configurar manualmente as métricas e os algoritmos de avaliação que serão utilizados (GALANTE; BONA, 2012). As operações no modo automático podem ser classificadas como **reativas** ou **proativas** (GALANTE; BONA, 2012; COUTINHO et al., 2014). Na forma reativa, as operações de elasticidade são tomadas baseando-se em nas métricas atuais coletadas do ambiente, que servem para disparar ações em caso de os dados coletados informarem a violação de algum *thresholds* (GALANTE; BONA, 2012). Na forma proativa, os dados coletados são utilizados para a realização de previsões de futuros eventos. Essas previsões são levadas em consideração para a tomada de decisão e a realização ou não de operações de elasticidade. Técnicas de previsão de carga dos recursos são comumente utilizadas para a reorganização proativa dos recursos (GALANTE; BONA, 2012; COUTINHO et al., 2014).

2.4 Balanceamento de Carga

A ideia básica do Balanceamento de Carga ou *Load Balancing* é distribuir a carga em proporções iguais de modo que cada recurso não seja sobrecarregado ou subutilizado em um sistema (MANDAL et al., 2013), sem perturbar o funcionamento da tarefa (SHOJA; NAHID; AZIZI, 2014). O serviço de balanceamento de carga usa vários algoritmos de escalonamento para determinar qual servidor deve tratar o pedido e o encaminha para o servidor selecionado (SHARMA; BANGA, 2013). Estes algoritmos podem ser classificados em dois tipos: balanceamento dinâmico de carga e balanceamento estático de carga.

Algoritmos de balanceamento estáticos são adequados principalmente para ambientes homogêneos e estáveis e podem produzir resultados muito bons nestes ambientes. No entanto, eles geralmente não são flexíveis e não podem coincidir com as mudanças dinâmicas nos atributos durante o tempo de execução (FARRAG; MAHMOUD; EL-HORBATY, 2015). Algoritmos de balanceamento dinâmicos são mais flexíveis e levam em consideração diferentes tipos de atributos do sistema, antes e durante o tempo de execução (RIMAL; CHOI; LUMB, 2009). Esses algoritmos podem se adaptar às mudanças e proporcionar melhores resultados em ambientes heterogêneos e dinâmicos. No entanto, os atributos de distribuição tornam-se mais complexos e dinâmicos. Como resultado alguns desses algoritmos podem tornar-se ineficiente e causar mais sobrecarga do que o necessário, resultando em uma degradação global do desempenho dos serviços (NUAIMI et al., 2012).

Os autores Patel, Patel e Patel (2012) e Shoja, Nahid e Azizi (2014) citam ainda como metas do balanceamento de carga: melhoria substancial no desempenho, manutenção da estabilidade do sistema, aumento da flexibilidade do sistema, de modo a adaptar-se às

modificações; e construir um sistema tolerante a falhas através da criação de backups.

O balanceamento de carga pode ser feito em duas camadas distintas. Podemos fazer o balanceamento de carga a nível processos e aplicação. Neste modelo a aplicação possui uma carga de trabalho definida e a divide entre os nós de processamento disponíveis (CASAVANT; KUHL, 1988; WARAICH, 2008; NAVAUX; RIGHI, 2009). E a nível de rede, onde novas conexões TCP são direcionadas por um *Middleware* para algum servidor selecionado por um algoritmo (ASLAM; SHAH, 2015), após o encaminhamento da conexão, a carga de trabalho passa a ser tratada pelo servidor que receber a conexão.

2.5 Considerações Parciais

Este capítulo apresentou conceitos fundamentais para a compreensão das tecnologias e decisões do modelo proposto. O modelo de comunicação *Publish/Subscribe* e suas entidades foi apresentado na Seção 2.1. As características, modelos de serviço (IaaS, PaaS e Saas) e modelos de implementação (Nuvem Privada, Nuvem Pública, Nuvem Híbrida e Nuvem Comunitária) da Computação em Nuvem foram apresentadas Seção 2.3. Os conceitos, métodos (migração, horizontal e vertical) e modelos (automático e manual) de elasticidade foram abordados na Subseção 2.3.3, assim como os conceitos de balanceamento de carga na Seção 2.4. Além disso o paradigma de Internet das Coisas é explorado na Seção 2.2, devido a forte influência no tema, tendo em vista que utiliza o modelo de comunicação abordado no trabalho e foi utilizado na fase de implantação.

3 TRABALHOS RELACIONADOS

Este capítulo tem como objetivo apresentar os trabalhos relacionados ao modelo proposto. Nesta análise foram incluídos trabalhos que aplicam elasticidade a *Brokers* e aplicações *Publish/Subscribe*. Os critérios utilizados na seleção são especificados na seção 3.1. Os trabalhos selecionados são apresentados nas Seções 3.2, 3.3 e 3.4. Por fim, estão presentes as oportunidades de pesquisa e considerações parciais na Seção 3.6.

3.1 Metodologia de Pesquisa e Escolha dos Trabalhos Relacionados

A pesquisa dos trabalhos relacionados foi realizada utilizando o Portal de Periódicos CAPES/MEC[1] e o portal Google Scholar[2], por indexarem diversos periódicos como IEEE-Xplorer[3], ACM - Digital Library[4] e Springer[5].

Para a realização da pesquisa, os principais termos utilizados foram "*cloud computing publish subscribe*", "*publish subscribe elasticity*", "*elastic publish subscribe*"e "*autoscaling publish subscribe*". Os resultados das pesquisas foram analisados com o objetivo de encontrar trabalhos que explorassem a elasticidade em aplicações *Publish/Subscribe* como forma de incrementar a escalabilidade do sistema. Trabalhos sem suporte a elasticidade ou com publicação anterior ao ano de 2011 foram desconsiderados.

3.2 E-STREAMHuB

Barazzutti et al. (2014) abrange um mecanismo para ampliar e reduzir recursos de forma automática para serviços *Publish/Subscribe* com filtragem baseada em conteúdo e executado em ambientes de nuvem. O sistema E-STREAMHuB é uma extensão do STREAMHuB (BARAZZUTTI et al., 2013), um *engine Publish/Subscribe* escalável porém estático.

A arquitetura de E-STREAMHuB é apresentada na Figura 5. Esta arquitetura é dividida camadas com componentes dedicados para cada operação. Desta forma, busca obter um melhor desempenho e permitir que o sistema escale cada camada individualmente. E-STREAMHuB divide o serviço *Publish/Subscribe* em três camadas fundamentais. Cada uma destas camadas pode ter mais de uma entidade de processamento, estas entidades de processamento são chamadas de *operator slices*.

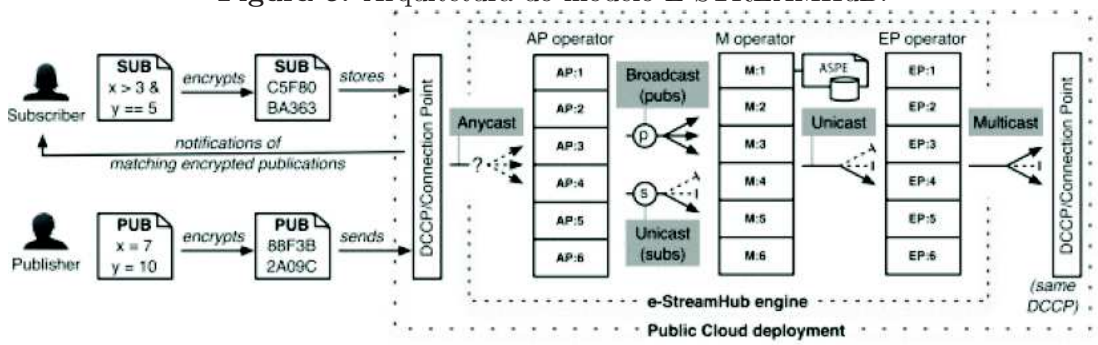
A primeira camada é chamada de Ponto de Acesso ou *Access Point* (AP). AP tem a função de dividir a carga de trabalho para a próxima camada em conjuntos que não se

[1]http://periodicos.capes.gov.br.ez101.periodicos.capes.gov.br
[2]https://scholar.google.com.br/
[3]http://ieeexplore.ieee.org/Xplore/home.jsp
[4]http://dl.acm.org/
[5]http://www.springer.com/

sobrepõem, utilizando para isso modulo *hash* para identificar cada *subscription*. Mensagens *subscribe* são tratadas direto na AP. Ela cria o *hash* das informações da mensagem associando a um operador da próxima camada, insere em sua tabela e envia via *unicast* a mensagem para o operador selecionado. Mensagens *publish* recebidas são transmitidas através de *broadcast* a todos os operadores da próxima camada.

A segunda camada é a de *Matching* (M), basicamente os operadores desta camada validam se existem *subscribers* interessados em receber a mensagem *publish* que esta sendo processada. Se existirem, o operador coloca os *subscribers* interessados em uma lista e repassa esta informação para um dos operadores da próxima camada. A terceira e última camada é chamada de Ponto de Saída ou *Exit Point* (EP), os operadores desta camada recebem a lista de *subscribers* gerada pela camada anterior e encaminham a mensagem *publish* que esta sendo processada aos interessados.

Figura 5: Arquitetura do modelo E-STREAMHuB.

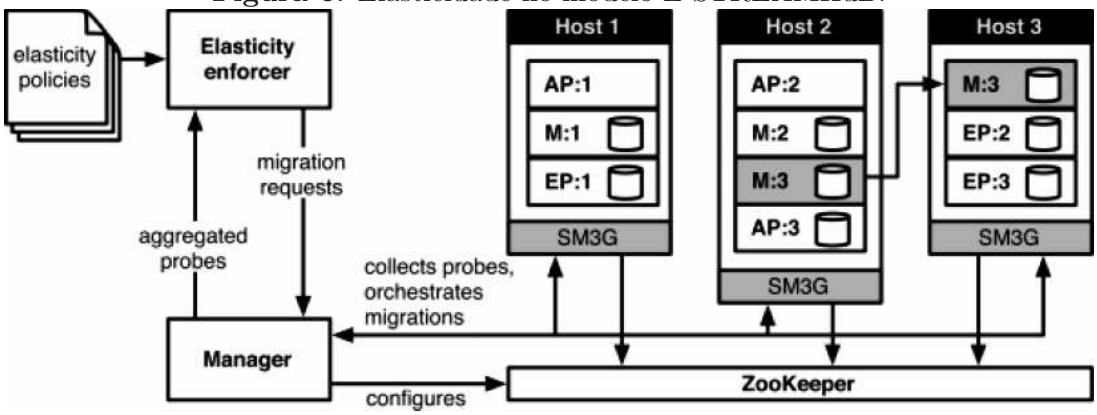


Fonte: adaptado de Barazzutti et al. (2014).

A Figura 6 apresenta um exemplo do processo de elasticidade do modelo E-STREAMHuB. A elasticidade é realizada através da migração de processos de Operadores entre nós ativos. Caso os nós ativos não sejam suficientes, o sistema solicita a criação de um novo nó. No entanto, a maneira como esta solicitação é feita, e como o novo nó será adicionado na solução não fica clara no artigo. Não descrevem se será uma réplica de máquinas virtuais (VMs), se será necessária a instalação de um novo servidor (Sistema Operacional e plataformas necessárias) ou ainda se novas máquinas físicas devem ser adicionadas à infraestrutura. O mesmo acontece com o processo de desativação de nós subutilizados.

A abordagem apresentada por E-STREAMHuB é interessante pois introduz o paralelismo de dados e permite o processamento de várias mensagens *publish* recebidas em paralelo contra todas as assinaturas registradas. No entanto, se o número de mensagens *publish* crescer muito, toda a solução será sobrecarregada tendo em vista que estas mensagens são enviadas para todos os operadores de *Matching*.

Figura 6: Elasticidade no modelo E-STREAMHuB.



Fonte: adaptado de Barazzutti et al. (2014).

3.3 EQS

EQS (TRAN; SKHIRI; ZIMÁNYI, 2011) é uma arquitetura de fila de mensagens que podem ser classificada como *Publish/Subscribe* baseada em tópicos, desenhado para permitir a escalabilidade elástica, desempenho e alta disponibilidade. Seu projeto é direcionado para uma implantação de uma infraestrutura de Computação em Nuvem usando a Infraestrutura como Serviço.

A Figura 7 apresenta a arquitetura do modelo EQS.

- Produtor e Ouvinte (*Producer and listener*): São equivalentes aos *publishers* e *subscribers* do modelo *Publish/Subscribe*. Um ouvinte é um cliente que se conecta a uma fila específica, a fim de recuperar uma mensagem. E um produtor é um cliente que se conecta a uma fila específica, a fim de entregar uma mensagem.

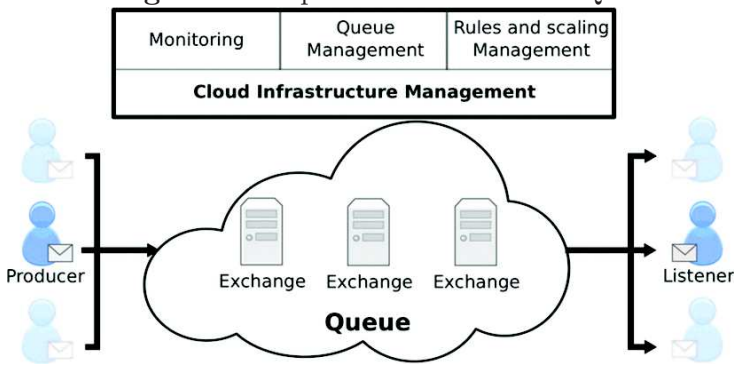
- Troca (*Exchange*): é o componente de baixo nível que faz a ligação entre produtores e ouvintes. Ele recebe uma mensagem na sua interface e transmite-as para os ouvintes numa relação N para N. Graças à sua natureza sem estado *stateless*, *exchanges* são os componentes EQS adequados para o aumento de escala, a fim de se adaptar à carga.

- Fila (*Queue*): Uma fila é o conceito lógico da entidade usada para transmitir mensagens relativas a um determinado assunto entre produtores e ouvinte. É o equivalente aos tópicos do modelo *Publish/Subscribe*. Cada fila possui um conjunto de limites de indicadores chave de desempenho (KPIs) emitidos a partir de um conjunto de definições Acordo de Nível de Serviço (SLA).

- Gestão de Filas (*Queue Management*): Ele mantém um registro permanente de todas as filas existentes, incluindo o seu nome, suas URLs, o SLA associado, a lista de *Exchanges* que compõem a fila e sua localização física. Estas informações constituem o metadados associado a uma fila.

- Monitoração (*Monitoring*): O componente de monitoração agrega informações sobre a saúde do serviço de EQS. Ele monitora os KPIs no nível EQS e ao nível de infraestrutura. É capaz de tomar ações em casos de falha de um *Exchange*, como por exemplo migrar filas para um *exchanges* livre ou criar uma nova instancia dele.
- Gerenciamento de regras e escalonamento (*Rules and scaling management*): este componente gerencia as regras e as ações desencadeadas em relação ao SLA definido.

Figura 7: Arquitetura do modelo EQS.



Fonte: adaptado de Tran, Skhiri e Zimányi (2011).

A elasticidade na EQS é obtida através da migração de tópicos entre nós de processamento, para isto a carga de cada tópico é monitorada. Caso a migração não seja o suficiente o modelo pode instanciar novas réplicas dos *Exchanges* para atender a demanda. A migração de tópicos entre nós pode gerar um custo e potencial interrupção do serviço.

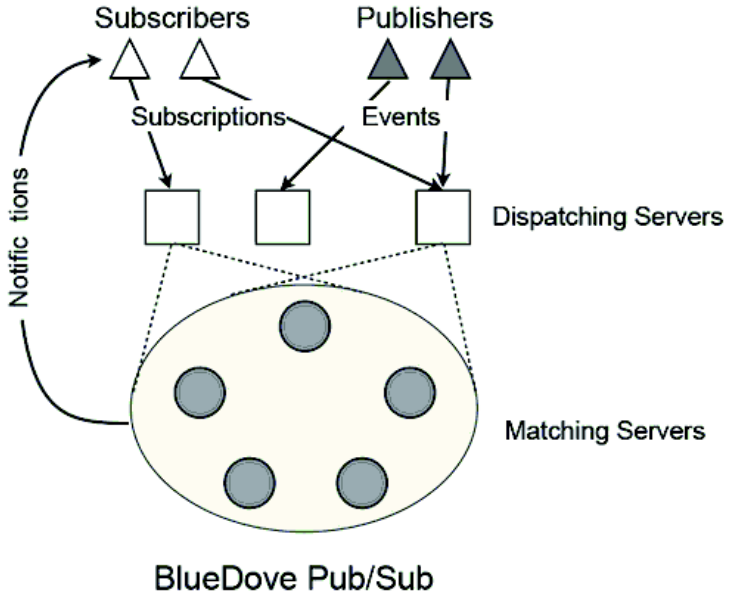
3.4 Blue Dove

Blue Dove (LI et al., 2011) visa a implantação de serviços *Publish/Subscribe* baseados em atributos no ambiente de nuvem pública. O fluxo de mensagens no Blue Dove é estritamente dependente do modelo de filtragem baseada em atributo usado para assinaturas. O nós disponiveis são divididos em regiões, e os atributos são associados a estas regiões. Mensagens *Subscribes* e *Publishs* são enviadas para os servidores correspondentes que estão no comando de uma determinada região.

A Figura 8 apresenta a arquitetura do modelo Blue Dove. Ela possui duas camadas, sendo a primeira dos *dispatchers servers*, que exerce o papel de *front-end*, direcionando os *publishers* e *subscribers* para os nós no comando de determinada região. A segunda camada do modelo é dos *matchers servers*, eles atuam como *back-end* do sistema, processando as mensagens recebidas e encaminhando-as aos *subscribers* quando necessário.

A abordagem apresentada por de Blue Dove não permite a utilização de sistemas de filtragem que não se baseiam em atributos ou que não permitam examinar o conteúdo das mensagem de *subscribe* no lado do servidor, como por exemplo mensagens criptografada.

Figura 8: Arquitetura do modelo Blue Dove.



Fonte: adaptado de Li et al. (2011).

Quanto a elasticidade, Blue Dove a suporta parcialmente, permitindo apenas escalar recursos, sem suportar a liberação dos mesmos caso sua utilização esteja baixa.

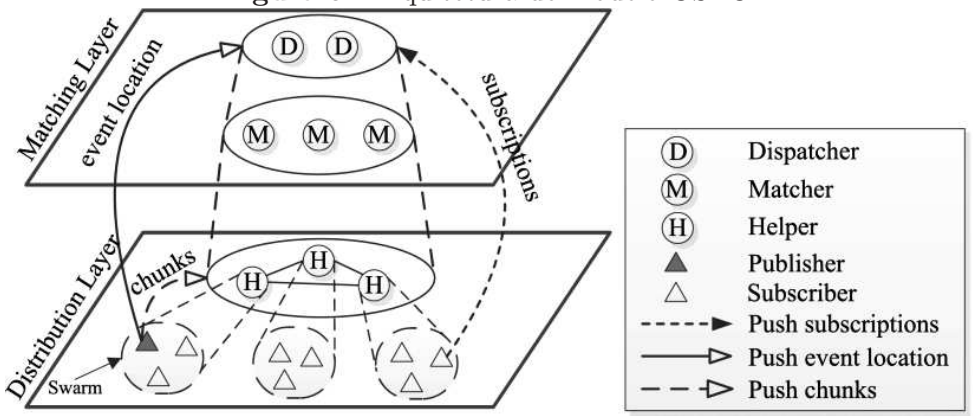
3.5 GSEC

Wang e Ma (2015) apresenta um serviço de *Publish/Subscribe* baseado em conteúdo, escalável e elástico chamado de GSEC (*A General Scalable and Elastic Content-Based Publish/Subscribe Service*). GSEC propõe um *framework* de duas camadas, que utiliza uma técnica de particionamento de espaço híbrido para alcançar uma taxa de transferência de correspondência elevada, dividindo as assinaturas (*subscriptions*) em múltiplos *clusters* de uma forma hierárquica. Além disso, uma técnica de distribuição de conteúdo baseada em auxiliares (*helpers*) é proposta para alcançar alta largura de banda de *upload*, onde os servidores atuam como provedores e coordenadores para explorar completamente a capacidade de *upload* do sistema.

A Figura 9 apresenta a arquitetura do modelo GSEC, que em uma visão de alto nível, consiste em duas camadas: a camada de correspondência (*matching layer*) e a camada de distribuição(*distribution layer*). A camada de correspondência é responsável pela correspondência de eventos com as assinaturas (*subscriptions*) de forma paralela. Nesta camada, um conjunto de servidores despachantes (chamados *dispatchers*) são expostos à Internet, e podem ser acessados por qualquer assinante (*subscriber*) e editor (*publisher*). Os distribuidores dividem as assinaturas em vários clusters que são gerenciados por um grupo de servidores de correspondência paralelos (chamados *matchers*). A camada de distribuição é responsável por distribuir o conteúdo do evento para os assinantes correspondentes.

Nessa camada, um grupo de servidores auxiliares (chamados *helpers*) são organizados em uma floresta de árvores *multicast*, que são usadas para aumentar a capacidade total de largura de banda de *upload* do sistema.

Figura 9: Arquitetura do modelo GSEC.



Fonte: adaptado de Wang e Ma (2015).

3.6 Análise e Oportunidades de Pesquisa

Os trabalhos apresentados foram avaliados de acordo com sete quesitos definidos como desejáveis. Os quesitos avaliados estão descritos em detalhes abaixo, e dispostos na Tabela 2.

- **Filtragem**: refere-se ao método de filtragem suportado pelo modelo;
- **Elasticidade**: define se o modelo suporta elasticidade de recursos, e ainda em quantos níveis da arquitetura a elasticidade está presente;
- **Método**: refere-se ao método de elasticidade, apresentando como novos recursos são adicionados ou removidos quando operações de elasticidade são necessárias. Os métodos possíveis são: migração, horizontal e vertical;
- **Modelo**: refere-se ao modelo de elasticidade, considerando a maneira como são executadas as operações do método de elasticidade Os modelos possíveis são: manual, automático reativo e automático proativo;
- **Operação**: define as técnicas e operações envolvidas para a realização da elasticidade relacionadas ao método de elasticidade;
- **Modificações**: define se o modelo exige adaptações por parte dos programadores, usuários ou aplicações para que seja utilizado; e qual item deve ser ajustado;

Tabela 2: Avaliação dos trabalhos relacionados com base nos quesitos selecionados.

Trabalho	**Filtragem**	**Suporta Elasticidade?**	**Método**	**Modelo**	**Operação**	**Modificações**	**Restrições de Plataforma**
E-STREAM HuB (BARAZZUTTI et al., 2014)	Conteúdo	Sim, multinível	Horizontal e Migração	Automático Reativo	Migração de processos e replicação de Máquinas Virtuais (VMs)	Sim, no *Middleware/Broker*	Se aplica apenas ao *Middleware* STREAMHuB
EQS (TRAN; SKHIRI; ZIMÁNYI, 2011)	Tópicos	Sim, somente em um nível	Horizontal e Migração	Automático Reativo	Migração de processos e replicação de VMs	Sim, no *Middleware/Broker*	o *Middleware* deve suportar migrações de processos
Blue Dove (LI et al., 2011)	Atributos	Somente Adição de Recursos, multinível	Horizontal	Automático Reativo	Replicação de VMs	Sim, no *Middleware/Broker*	Se aplica apenas a um *Middleware*
GSEC (WANG; MA, 2015)	Conteúdo	Sim, multinível	Horizontal	Automático Reativo	Replicação de VMs	Sim, no *Middleware/Broker*	Se aplica apenas a um *Middleware*

Fonte: elaborado pelo autor.

- **Restrições de Plataforma**: refere-se as restrições impostas pelos modelos analisado.

Analisando os trabalhos, podem ser destacados as seguintes lacunas em relação aos sistemas estudados nas seções anteriores:

(i) Necessidade de alterar o código fonte da aplicação ou desenvolver *scripts* adicionais (BARAZZUTTI et al., 2014; TRAN; SKHIRI; ZIMÁNYI, 2011; LI et al., 2011; WANG; MA, 2015);

(ii) Utilização de componentes proprietários que não estão disponíveis (BARAZZUTTI et al., 2014; WANG; MA, 2015);

(iii) Restrições quanto ao *Middleware* utilizado (BARAZZUTTI et al., 2014; TRAN; SKHIRI; ZIMÁNYI, 2011; LI et al., 2011; WANG; MA, 2015);

(iv) Capacidade de filtragem restrita a apenas um método (BARAZZUTTI et al., 2014; TRAN; SKHIRI; ZIMÁNYI, 2011; LI et al., 2011; WANG; MA, 2015);

(v) Nenhuma análise de situações de pico esporádicos quando se atinge um *threshold* (BARAZZUTTI et al., 2014; TRAN; SKHIRI; ZIMÁNYI, 2011; LI et al., 2011; WANG; MA, 2015);

Considerando os trabalhos analisados, é possível encontrar soluções pontuais que promovem elasticidade para *Brokers Publish/Subscribe*. No entanto, todas elas impõem restrições quanto ao *Middleware* utilizado e capacidades de filtragem. Além disto as situações de pico esporádicos não são tratadas.

4 O MODELO PSELASTIC

Este Capítulo descreve o modelo PSElastic, que é um modelo de elasticidade multinível para *Brokers Publish/Subscribe*. A Seção 4.1 apresenta as decisões de projeto. A arquitetura será abordada na Seção 4.2, que define cada um dos componentes como Orquestrador e Gerenciador; nas Subseções 4.2.1 e 4.2.2 serão apresentados os fluxos de tratamento de mensagens, e os de adição e remoção de recursos. Por fim, a decisão de elasticidade será apresentada na Seção 4.3 e as considerações parciais na Seção 4.4.

4.1 Decisões de Projeto

PSElastic fornece elasticidade horizontal e reativa para *Brokers Publish/Subscribe* de forma transparente para aplicações e usuários. Foi adotada a elasticidade horizontal devido ao fato de que, com elasticidade vertical, a quantidade de recursos fica limitada sempre aos recursos disponíveis em uma única máquina física e, além disso, a maioria dos sistemas operacionais comuns não permitem que sejam adicionados recursos em tempo de execução (DUTTA et al., 2012; LORIDO-BOTRAN; MIGUEL-ALONSO; LOZANO, 2014; RODRIGUES, 2015). A elasticidade reativa foi utilizada devido a simplicidade e velocidade no momento da detecção das ações de elasticidade. Ainda, PSElastic opera com a granularidade de máquinas virtuais, ciente da sobrecarga de inicializar uma máquina virtual, levando em conta esse conhecimento para oferecer esse recurso. Para o desenvolvimento do modelo PSElastic, foram definidas as decisões de projeto abaixo:

(i) Usuários não precisam configurar o mecanismo de elasticidade; porém, há a possibilidade de informar os *thresholds* que devem ser utilizados;

(ii) Desenvolvedores não precisam reescrever o código da aplicação a fim de tirar proveito da elasticidade;

(iii) O modelo será genérico a ponto de abranger qualquer *Broker* utilizado;

(iv) O *Broker* provê autenticação para os diversos atores que se conectam a ele;

(v) O cenário de investigação utilizará uma nuvem privada, composta por recursos homogêneos não compartilhados entre usuários;

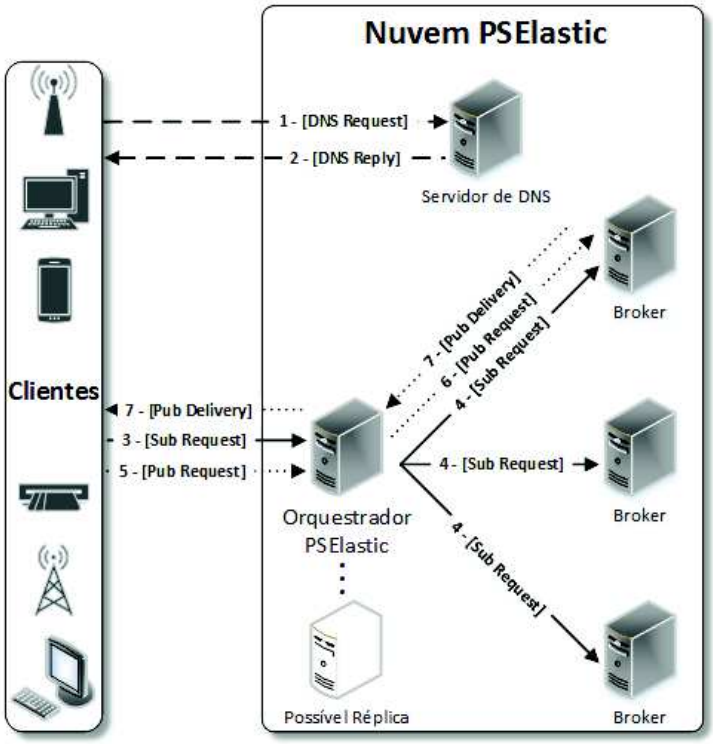
(vi) A estratégia de elasticidade adotada será de forma reativa, automática e horizontal com base em *thresholds*, utilizando a replicação de máquinas virtuais;

(vii) A elasticidade se dará em dois níveis, o primeiro instanciando réplicas de *Brokers* e o segundo instanciando réplicas de Orquestrador.

(viii) Qualquer aplicação que utilize o modelo de comunicação apresentado na Seção 2.1 pode usufruir dos benefícios do modelo PSElastic.

Na Figura 10 é apresentada uma visão de alto nível do modelo PSElastic. Primeiramente, os clientes consultam o Servidor de DNS para saber com qual das réplicas de Orquestrador PSElastic devem se comunicar. Em seguida iniciam a comunicação com o Orquestrador selecionado, que por sua vez direciona a comunicação para os *Brokers* após validação do tipo de mensagem (*Publish* ou *Subscribe*) e consulta ao algoritmo de balanceamento (caso a mensagem seja de *Publish*). Após o processamento, o *Broker* envia a mensagem *Publish Delivery* para o Orquestrador que a encaminha aos *Subscribers* interessados na mensagem *Publish*. A arquitetura do modelo será mais explorada na Seção 4.2 e o fluxograma de tratamento de mensagens será abordado na Subseção 4.2.1.

Figura 10: Visão de alto nível do modelo PSElastic. As setas com linha tracejada correspondem a mensagens de consulta e resposta do serviço de DNS. As setas com linha contínua correspondem a mensagens de assinatura (*Subscribe*). As setas com linha pontilhada por sua vez, correspondem a mensagens de publicação (*Publish*).

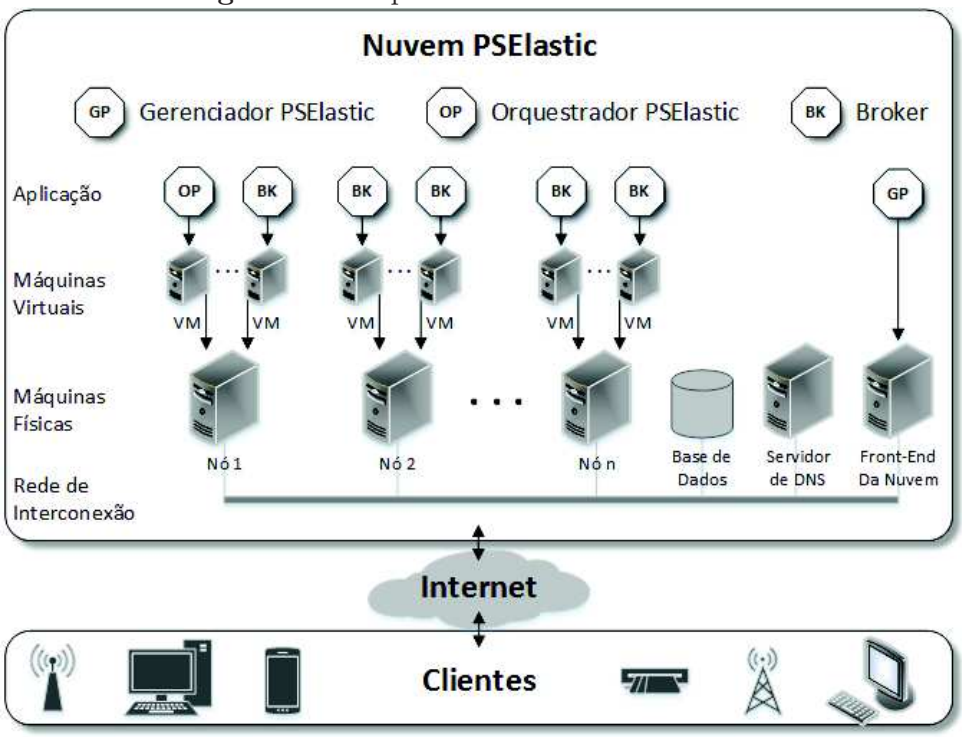


Fonte: elaborado pelo autor.

4.2 Arquitetura

PSElastic é um modelo que opera no nível de PaaS permitindo a *Brokers Publish/-Subscribe* obterem vantagem da elasticidade em nuvem sem a necessidade de alterações no código fonte. Para fornecer elasticidade, o modelo atua com operações de alocação e

Figura 11: Arquitetura do modelo PSElastic.



Fonte: elaborado pelo autor.

consolidação de instâncias de máquinas virtuais. A Figura 11 apresenta a arquitetura do modelo PSElastic. Seus componentes e funcionalidades são definidos abaixo:

- Gerenciador PSElastic: é o componente responsável por gerenciar a elasticidade e controlar as reconfigurações de recursos. O Gerenciador PSElastic considera dados de carga de trabalho das máquinas físicas e virtuais que fazem parte da infraestrutura de nuvem como entrada para análise de necessidades de reconfigurações de recursos. Além disso, implementa estratégias que possibilitam ao Orquestrador PSElastic estar ciente da adição ou remoção de recursos sem a necessidade de envolvimento do usuário.

- Orquestrador PSElastic: realiza o tratamento inicial das mensagens recebidas, direcionando a comunicação para os *Brokers* após validação do tipo de mensagem e consulta ao algoritmo de balanceamento. Desta forma, acaba fazendo o papel de Balanceador de Carga ou *Load Balancer* para a camada de *Brokers*. O fluxograma de tratamento de mensagens será explorado na Subseção 4.2.1.

- *Broker*: é o componente responsável por processar as mensagens. Pertence a Camada de *Middleware* apresentada na Figura 2 da Seção 2.2. Seu motor de análise gera uma lista, e encaminha a mensagem de *Publish* para os *Subscribers* que devem recebe-la. Este componente não é desenvolvido ou definido pelo modelo, PSElastic é genérico a ponto de abranger o maior número de *Brokers* possíveis.

- *Front-End* da nuvem: este componente é responsável por gerenciar os recursos físicos e lógicos no ambiente da nuvem privada.

- Base de Dados: componente utilizado para armazenar as informações dos *Brokers*.

- Servidor de DNS: componente utilizado para prover balanceamento de carga a nível de Orquestradores PSElastic, caso mais de um Orquestrador PSElastic seja instanciado.

- Nós: máquinas físicas presentes no ambiente de nuvem que são responsáveis pelo fornecimento de recursos de *hardware* para as máquinas virtuais.

- VMs: máquinas virtuais ou *virtual machines*, que são executadas dentro dos nós físicos.

4.2.1 Tratamento de Mensagens

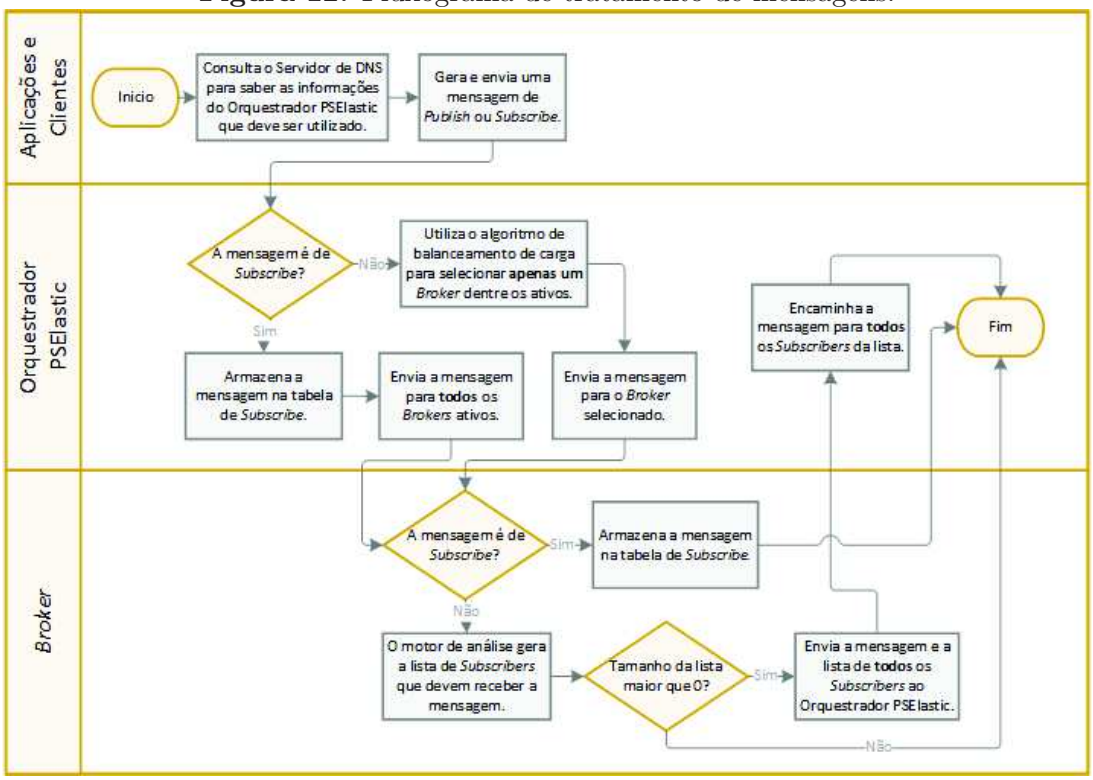
A Figura 12 apresenta o fluxograma de tratamento das mensagens que chegam ao modelo. O fluxo de tratamento de mensagens se inicia com o cliente, que consulta o Servidor DNS para saber qual réplica do Orquestrador PSElastic deve ser utilizada. Em seguida uma mensagem *Publish* ou *Subscribe* é gerada e enviada.

Ao receber a mensagem o Orquestrador PSElastic valida seu tipo, se for *Subscribe* ele armazena as informações da mensagem em uma tabela de *Subscribe* e então encaminha a mensagem para todos os *Brokers* ativos. Este armazenamento é necessário pois ao subir uma nova réplica de *Broker* o Orquestrador deverá informar a lista completa de *Subscribes* recebidos até o momento, para que este novo ativo possua todas as regras de tratamento de mensagens necessárias.

Caso a mensagem seja *Publish*, o Orquestrador utiliza o algoritmo de balanceamento de carga para selecionar apenas um dos *Brokers* ativos e então encaminha a mensagem apenas para o *Broker* selecionado. Esta estratégia é utilizada para que as mensagens *Publish* sejam distribuídas entre os *Brokers* ativos e evitar que dois *Brokers* processem a mesma mensagem, aumentando assim a capacidade de processamento de mensagens do modelo.

O *Broker* também valida o tipo de mensagem, se for *Subscribe* ele armazena suas informações na tabela de *Subscribe* e o fluxo de tratamento de mensagens chega a seu fim. Se for *Publish* ele utiliza seu motor de análise para gerar a lista de *Subscribers* que devem receber a mensagem. Se o tamanho da lista for maior que zero, o *Broker* envia a mensagem ao Orquestrador para que seja encaminhada a todos os *Subscribers* da lista e então chega ao fim do fluxo. Se não for maior que zero, ele vai direto para o final do fluxo.

Figura 12: Fluxograma de tratamento de mensagens.



Fonte: elaborado pelo autor.

4.2.2 Adição e Remoção de Recursos

Na Figura 13 é apresentado o fluxograma seguido para instanciar novos recursos na nuvem. O Gerenciador PSElastic é quem inicia o fluxo após identificar sobrecarga no sistema. O primeiro passo é identificar qual camada esta sobrecarregada, *Broker* ou Orquestrador PSElastic, e solicitar ao *Front-End* da nuvem um novo recurso para esta camada. O *Front-End* da nuvem instancia um novo recurso com o perfil solicitado e quando a máquina estiver acessível na rede, envia uma notificação ao Gerenciador utilizam uma área de dados compartilhada.

O Gerenciador PSElastic valida se todos os serviços necessários estão em execução no novo recurso, se não estiverem ele conecta no terminal deste recurso e realiza a inicialização antes de continuar o fluxo. Após esta confirmação o Gerenciador envia as informações da tabela de *Subscribe* para o processo de *Broker* ou Orquestrador PSElastic em execução no novo recurso, que por sua vez insere estas informações em sua tabela local. O fluxo volta então para o Gerenciador, que valida o tipo de recurso instanciado.

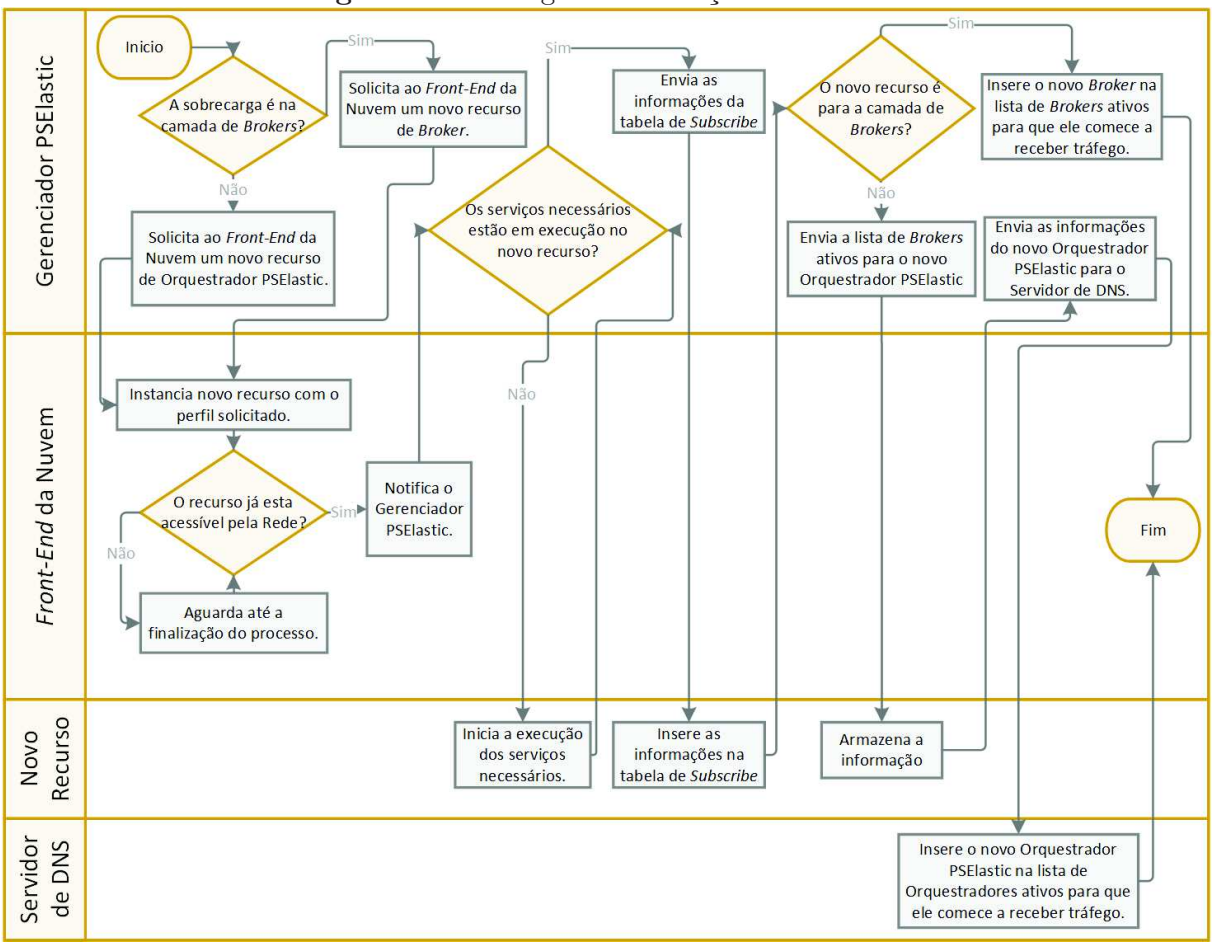
Caso o novo recurso seja destinado à Camada de *Brokers*, o Gerenciador adiciona o novo *Broker* na lista de *Brokers* ativos para que ele comece a receber tráfego e então o fluxo chega a seu final.

Caso o recurso seja um novo Orquestrador PSElastic, o Gerenciador envia a lista de *Brokers* ativos para que o processo de Orquestrador PSElastic em execução no recurso

a armazene em sua tabela local. Após isso, o Gerenciador envia as informações do novo Orquestrador ao Servidor de DNS. O Servidor DNS insere as informações na lista de Orquestradores ativos para que ele comece a receber tráfego e o fluxo chega a seu final.

É importante destacar que o modelo possibilita a criação de réplicas apenas de *Brokers* e de Osquestradores PSElastic. Réplicas do Gerenciador nunca serão instanciadas, este é um servidor único para todo o modelo.

Figura 13: Fluxograma de adição de recursos.

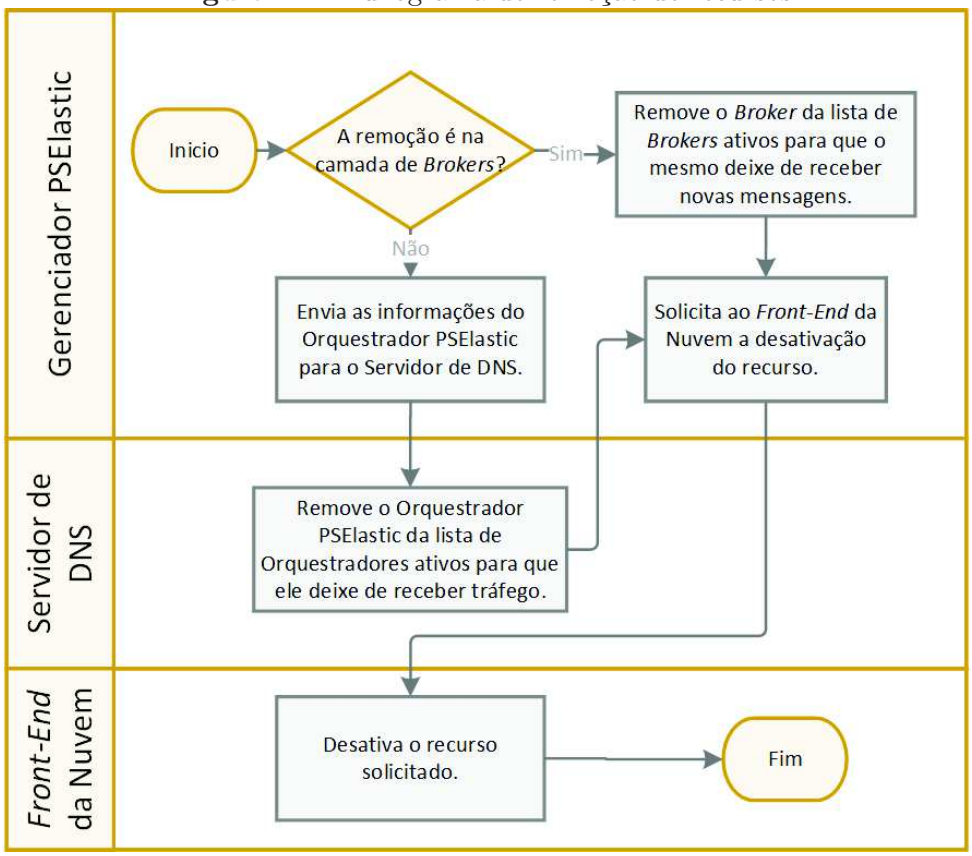


Fonte: elaborado pelo autor.

A Figura 14 por sua vez exibe o fluxograma de remoção de recursos na arquitetura. Assim como no processo de adição, quem inicia este processo é o Gerenciador PSElastic após identificar recurso subutilizados no sistema. O primeiro passo é identificar qual camada que deve ser alterada, *Broker* ou Orquestrador PSElastic. Caso o recurso a ser desativado seja um Orquestrador PSElastic, o Gerenciador solicita ao Servidor DNS a remoção deste recurso da lista de ativos, para que ele deixe de receber novas mensagens. Após o termino do tratamento das mensagens já recebidas, o Gerenciador solicita a desativação do recurso ao *Front-End* da nuvem e o fluxo chega ao fim. Caso o recurso a ser desativado seja um *Broker*, o gerenciador remove o recurso da lista de *Brokers* ativos, para que ele deixe de receber novas mensagens. Após o termino do tratamento das mensagens

pendentes o Gerenciador solicita a desativação do recurso ao *Front-End* da nuvem e o fluxo chega ao fim.

Figura 14: Fluxograma de remoção de recursos.



Fonte: elaborado pelo autor.

4.3 Decisão de Elasticidade

O Gerenciador de elasticidade do *middleware* AutoElastic (RODRIGUES, 2015) foi utilizado como Gerenciador do modelo PSElastic e todas suas características, técnicas de monitoramento e lançamento da elasticidade foram mantidas. A presente Seção apresenta como é definida a carga do ambiente e como esse valor é utilizado para realizar o lançamento de operações de elasticidade.

Assim como Imai, Chestna e Varela (2012) e Chiu e Agrawal (2010), a atividade de monitoramento do Gerenciador PSElastic é feita de forma periódica. O Gerenciador captura os valores de carga de CPU de cada máquina virtual e aplica um cálculo de séries temporais, considerando também valores coletados anteriormente, para obter a carga geral, aqui chamada de carga de processamento do sistema. Cada ciclo de monitoramento consiste em uma observação, em que os valores são coletados e armazenados (permitindo o uso de informações temporais) para serem utilizados posteriormente. PSElastic implementa a técnica de elasticidade utilizando regras baseadas em *thresholds* (LORIDO-BOTRAN;

MIGUEL-ALONSO; LOZANO, 2014), do mesmo modo como plataformas de nuvem comerciais como Amazon AWS[1], Microsoft Azure[2]. O valor resultante do cálculo da carga de processamento do sistema é utilizado em comparação com os *thresholds* superior e inferior para a tomada de decisões de elasticidade (JAMSHIDI; AHMAD; PAHL, 2014). A esses *thresholds*, podem ser atribuídos valores entre 0 e 100.

A carga de processamento do sistema é calculada através da Equação 4.1, em que a função chamada $CPS(o)$ calcula este valor na observação o. Esta função consiste em uma média aritmética da carga de CPU de cada uma das máquinas virtuais, em que v é a quantidade de máquinas virtuais. A carga de CPU de cada máquina virtual é calculada através da função $SES(e, o)$, representada na Equação 4.2. Esta equação aplica o método SES do modelo ARIMA, realizando a suavização exponencial simples da carga de CPU, representada por $CPU(e, o)$, de uma determinada máquina virtual e, considerando todo o histórico de observações, sendo o a observação atual. Por fim, o valor de CPS obtido é utilizado em comparação com os *thresholds* superior T_s e inferior T_i disparando ações de elasticidade se ocorrerem violações. A Figura 15 apresenta o modelo de elasticidade reativa empregado por PSElastic utilizando a carga de processamento do sistema. Em relação aos parâmetros utilizados para o cálculo das Equações 4.1 e 4.2, a Tabela 3 sumariza todos os parâmetros e funções apresentadas.

Figura 15: Modelo de elasticidade reativa PSElastic baseado em *thresholds* superior (T_s) e inferior (T_i).

REGRA1: se CONDIÇÃO1 então AÇÃO1
REGRA2: se CONDIÇÃO2 então AÇÃO2

CONDIÇÃO1: CPS(o) $> T_s$, em que o é a última observação de monitoramento.
CONDIÇÃO2: CPS(o) $< T_i$, em que o é a última observação de monitoramento.

AÇÃO1: Aloca um novo nó e aloca n máquinas virtuais nele, em que n é a quantidade de núcleos de processamento do nó.
AÇÃO2: Finaliza as máquinas virtuais que estão executando dentro do nó e remove este nó após isso.

Fonte: adaptado de Rodrigues (2015).

$$CPS(o) = \frac{\sum_{e=0}^{v-1} SES(e, o)}{v} \tag{4.1}$$

$$SES(e, o) = \begin{cases} \frac{CPU(e,o)}{2} & se\ o = 0 \\ \frac{SES(e,o-1)}{2} + \frac{CPU(e,o)}{2} & se\ o \neq 0 \end{cases} \tag{4.2}$$

em que $CPU(e, o)$ se refere a carga de CPU da máquina virtual e na observação de monitoramento de índice o.

[1]https://aws.amazon.com/pt/
[2]https://azure.microsoft.com/pt-br/

Tabela 3: Descrição dos parâmetros e funções utilizados nas equações de definição do mecanismo de elasticidade.

Parâmetro/Função	Descrição
e	Índice de uma máquina virtual executando um processo escravo.
o	Observação de carga mais atual do Gerenciador PSElastic.
v	Quantidade de máquinas virtuais executando processos escravos.
T_i	Valor do *threshold* inferior.
T_s	Valor do *threshold* superior.
$CPU(e, o)$	Carga de CPU da máquina virtual e na observação o.
$SES(e, o)$	Carga de processamento da máquina virtual e na observação o.
$CPS(o)$	Carga de processamento do sistema na observação o.

Fonte: adaptado de Rodrigues (2015).

SES opera com o conceito de *Aging* (TANENBAUM, 2003), empregando uma suavização exponencial simples em que a última medida tem a maior influência sobre o índice de carga. A ideia de *Aging* é empregada no modelo PSElastic buscando tratar situações de picos de carga, especialmente para evitar ações de elasticidade ocasionadas por falsos-negativos ou falsos-positivos. Por exemplo, assumindo um *threshold* superior de 80%, valores de monitoramento de CPU para uma única máquina virtual tais como 82, 78, 81, 80, 77 e 93 (valor mais recente), tem-se $SES(e, o) = \frac{1}{2}.93 + \frac{1}{4}.77 + \frac{1}{8}.80 + \frac{1}{16}.81 + \frac{1}{32}.78 + \frac{1}{64}.82 = 84.53$. Esse valor permite a reconfiguração de recursos.

Em contraste com PSElastic, a abordagem de Imai, Chestna e Varela (2012) espera por x observações consecutivas fora da margem do *threshold*. Neste caso, o uso de x igual a 2, 3 ou 4 não resultaria em ações de elasticidade. Em adição a esse cenário de falso-negativo, um falso-positivo ocorre quando uma aplicação apresenta um pico ou queda de forma inesperada, desencadeando ações de elasticidade desnecessárias. Assim, a maneira de PSElastic evitar *thrashing* contempla a utilização de séries temporais e da técnica de suavização exponencial simples para suavizar possíveis ruídos nas observações de carga.

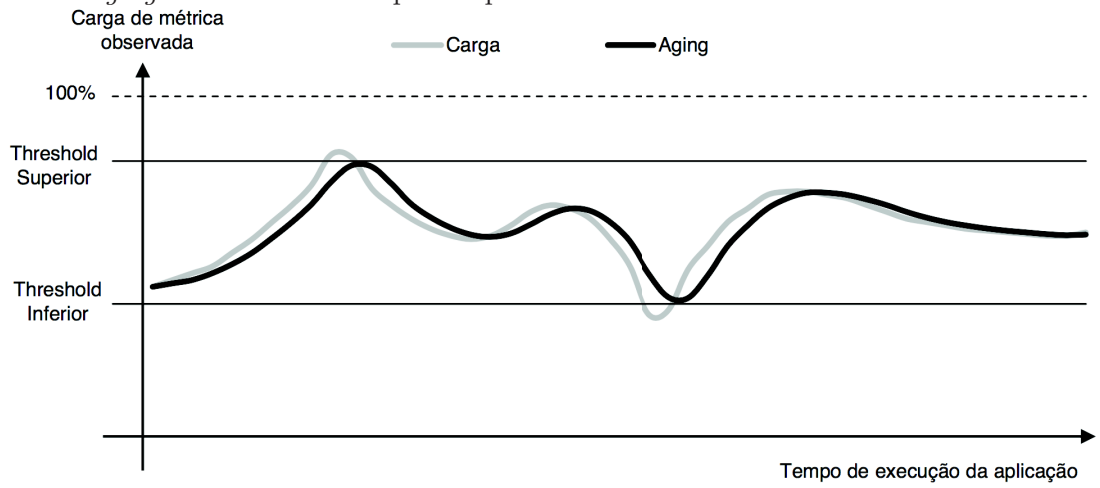
A Figura 16 demonstra uma situação em que ações de elasticidade ocorrem quando é considerado apenas a carga real da métrica monitorada. Entretanto, com a utilização do conceito de *Aging* os picos e quedas da métrica monitorada são amortizados levando-se em conta os valores coletados nas observações passadas, evitando operações desnecessárias.

4.4 Considerações Parciais

O modelo PSElastic apresentado, visa proporcionar elasticidade multinível para *Brokers* de aplicações que utilizam o modelo de comunicação *Publish/Subscribe*. Esta elasticidade deve ser transparente e não pode afetar os serviços oferecidos.

A arquitetura do modelo foi apresentada na Seção 4.2. Ela descreve o Gerenciador PSElastic e suas funções, assim como as funções dos componentes Orquestrador PSElastic,

Figura 16: Cenário em que a alocação e desalocação de recursos são evitadas com o uso da técnica de *Aging* em momentos de pico e queda dos valores da métrica monitorada.



Fonte: adaptado de Rodrigues (2015).

Broker, *Front-End* da nuvem, Base de Dados, Servidor de DNS, Nós e VMs. O fluxo de tratamento de mensagens é apresentado na Subseção 4.2.1, assim como os de adição e remoção de recursos no modelo na Subseção 4.2.2. Por fim, a decisão de elasticidade utilizando *thresholds* e *Aging* foi apresentada na Seção 4.3.

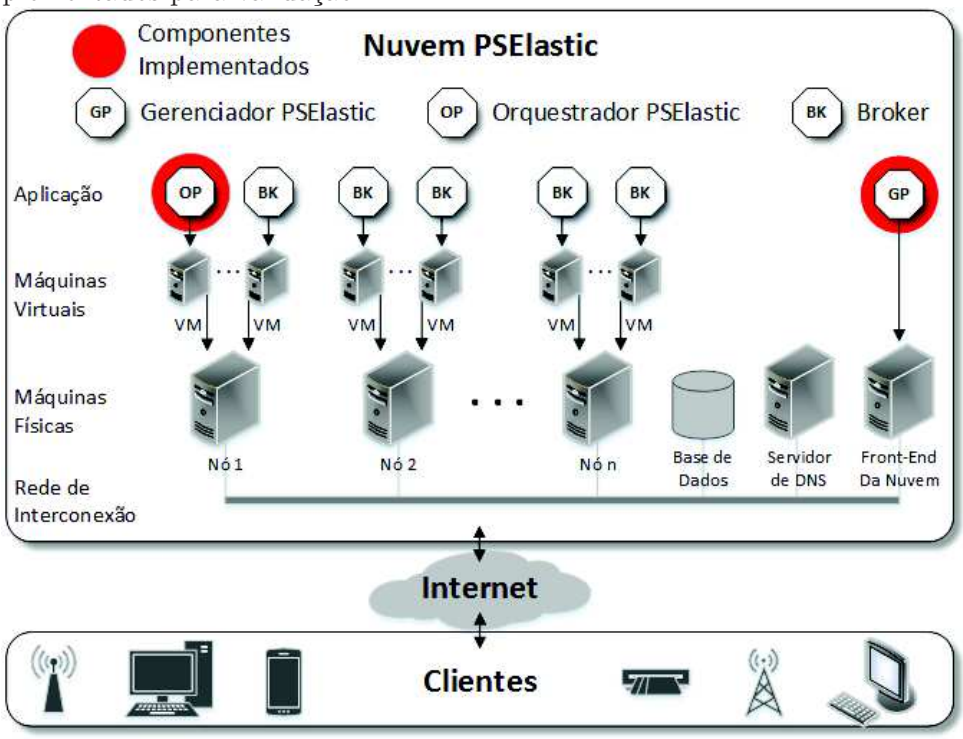
5 METODOLOGIA DE AVALIAÇÃO

Conforme abordado na Seção 4.2 o modelo PSElastic é composto por um total de 8 componentes. Levando em consideração que o objetivo é de proporcionar elasticidade a *Brokers Publish/Subscribe*, os componentes necessários para isto foram desenvolvidos e avaliados. Mais detalhes sobre os componentes que foram implementados são apresentados na Seção 5.1. A infraestrutura utilizada para as avaliações é abordada na Seção 5.2 e os cenários de testes na Seção 5.3. Por fim, as métricas de avaliação são abordadas na Seção 5.4 e as considerações parciais na Seção 5.5

5.1 Protótipo

A Figura 17 apresenta a arquitetura do PSElastic, ressaltando com círculos vermelhos o Gerenciador e Orquestrador PSElastic, todas as alterações, algoritmos e implementações de protótipo serão restritas a estes componentes do modelo.

Figura 17: Arquitetura proposta, destacando com círculos vermelhos (OP e GP) os componentes implementados para validação.



Fonte: elaborado pelo autor.

5.1.1 Orquestrador PSElastic

A implementação do Orquestrador PSElastic foi baseada na aplicação Python Director[1], um balanceador de carga escrito em Python que recebe as conexões TCP de entrada e as conecta a um dos vários servidores *back-end*. Os algoritmos de balanceamento de carga suportados são: *Random, Round Robin, Leastconns*.

Foram realizadas modificações na aplicação para que ela trate de forma diferente mensagens *Publish* e *Subscribe*. Mensagens *Publish* são encaminhadas para apenas um *Broker* dentre os disponíveis, foi utilizado o algoritmo *Round Robin* para seleção do servidor devido a sua simplicidade e eficiência. Quando uma mensagem *Subscribe* é recebida, o Orquestrador abre um *socket* TCP para cada *Broker* disponível e encaminha a mensagem, simulando assim uma conexão TCP de 1 (cliente) para N (*Brokers*), funcionalidade não suportada por padrão no protocolo TCP.

5.1.2 Gerenciador PSElastic

Conforme citado na Seção 4.3, o Gerenciador de elasticidade do *middleware* AutoElastic (RODRIGUES, 2015) foi utilizado como Gerenciador do modelo PSElastic e todas suas características foram mantidas. As principais funções de monitoramento e gerenciamento da elasticidade são realizadas pelo Gerenciador. Ele é o responsável por realizar a coleta de dados dos recursos do ambiente, aplicar a avaliação e gerenciar as operações de elasticidade.

A Figura 18 apresenta o seu ciclo de monitoramento, demonstrando as principais operações realizadas. A cada intervalo de tempo, o gerenciador verifica se existem recursos inicializados anteriormente que podem estar disponíveis para a utilização. Isso ocorre pois, quando novos recursos são adicionados ao ambiente, eles estarão disponíveis apenas após um determinado tempo. Após essa verificação, o Gerenciador coleta os dados de monitoramento dos recursos aos quais aplica um algoritmo de avaliação buscando obter uma única métrica que define a carga do ambiente. O resultado desse cálculo é utilizado para comparação com os *thresholds* buscando identificar situações que possam ocasionar operações de elasticidade. Após a realização ou não dessas operações, o gerenciador aguarda novamente um intervalo de tempo para iniciar o próximo ciclo de monitoramento (RODRIGUES, 2015).

O Gerenciador considera dados de carga de trabalho das máquinas físicas que compõem a infraestrutura de nuvem como entrada para análise de necessidades de reconfigurações de recursos. Ele utiliza uma área de dados compartilhada, privada para as máquinas virtuais e as máquinas físicas dentro do ambiente da nuvem PSElastic para viabilizar a

[1] *Python Director - pure Python tcp load balancer*: https://github.com/skyline75489/pythondirector e http://pythondirector.sourceforge.net

Figura 18: Principais operações realizadas pelo Gerenciador em cada ciclo de monitoramento.

Fonte: adaptado de Rodrigues (2015).

comunicação entre os processos da aplicação e o Gerenciador. A troca de mensagens pode ser realizada através de três tipos de notificações para viabilizar a elasticidade assíncrona, conforme demonstra a Tabela 4.

Tabela 4: Notificações fornecidas através da área de dados compartilhada a fim de viabilizar a comunicação entre o Gerenciador e o Processo Integrador.

Notificação	Direção	Descrição
Notificação 1	Gerenciador → Processo Integrador	Disponível n novos recursos.
Notificação 2	Gerenciador → Processo Integrador	Solicitação de permissão para consolidação/desativação de recursos específicos.
Notificação 3	Processo Integrador → Gerenciador	Resposta à Notificação 2 permitindo a consolidação/desativação dos recursos.

Fonte: adaptado de Rodrigues (2015).

Considerando o método de elasticidade, PSElastic utiliza a replicação horizontal de máquinas virtuais (KOUKI et al., 2014). As operações de elasticidade consistem sempre na adição de uma máquina física com n máquinas virtuais ou a remoção de uma máquina física e suas máquinas virtuais. A inicialização de novos recursos é realizada pelo Gerenciador que, somente após as novas máquinas virtuais terminarem seu processo de inicialização, realiza o processo de notificação de novos recursos.

Quanto à consolidação de recursos, o grão de trabalho é sempre uma máquina física e não uma máquina virtual. Isso se deve ao fato do uso eficiente dos recursos (não usar parcialmente os núcleos disponíveis) e melhor gerenciamento no consumo de energia elétrica. Em especial, Baliga et al. (2011) afirmam que o número de máquinas virtuais em uma máquina física não é um fator tão influente para consumo energético, mas sim se a máquina física está ligada ou não.

A técnica de *Aging* foi empregada no modelo PSElastic buscando tratar situações de picos de carga, especialmente para evitar ações de elasticidade ocasionadas por falsos-negativos ou falsos-positivos. *Aging* realiza suavização exponencial simples em que a última medida tem a maior influência sobre o índice de carga (RODRIGUES, 2015).

Para possibilitar a integração entre as diferentes aplicações presentes no modelo PSElastic foi implementado o Algoritmo 1. Ele é responsável por identificar as operações de elasticidade e fazer os ajustes necessários no Orquestrador PSElastic e na solução de DNS utilizada na elasticidade da camada de Orquestradores.

Algoritmo 1: Pseudo-linguagem do processo integrador.

```
   Entrada: arquivo com conteúdo a ser enviado
   Saída: assinaturas destinadas ao modelo
 1 início
 2     enquanto (verdadeiro) faça
 3         topico = divide_linha(linha_do_arquivo[i], coluna_1);
 4         se (arquivo_existe (adicionar_broker)) então
 5             acao=adicionar
 6             para (i=0; i < quantidade_servidores_orquestrador; i++) faça
 7                 conecta_servidor(servidor_orquestrador[i]);
 8                 altera_configuracao_orquestrador (acao, ip_servidor_adicionar);
 9                 reinicia_processo_osquestrador();
10                 desconecta_servidor(servidor_orquestrador[i]);
11             fim
12         fim
13         se (arquivo_existe (remover_broker)) então
14             acao=remover
15             para (i=0; i < quantidade_servidores_orquestrador; i++) faça
16                 conecta_servidor(servidor_orquestrador[i]);
17                 altera_configuracao_orquestrador (acao, ip_servidor_remover);
18                 reinicia_processo_osquestrador();
19                 desconecta_servidor(servidor_orquestrador[i]);
20             fim
21         fim
22         se (arquivo_existe (adicionar_orquestrador)) então
23             acao=adicionar
24             conecta_servidor(servidor_dns);
25             altera_configuracao_dns (acao, ip_servidor_adicionar);
26             reinicia_processo_dns();
27             desconecta_servidor(servidor_dns);
28         fim
29         se (arquivo_existe (remover_orquestrador)) então
30             acao=remover
31             conecta_servidor(servidor_dns);
32             altera_configuracao_dns (acao, ip_servidor_remover);
33             reinicia_processo_dns();
34             desconecta_servidor(servidor_dns);
35         fim
36     fim
37 fim
```

5.1.3 Clientes

Os Algoritmos 2 e 3 foram desenvolvidos para gerar tráfego para o modelo, eles são executados em máquinas virtuais (VMs) exclusivas, simulando os clientes da solução. Estas VMs não são monitoradas pelo Gerenciador PSElastic e não são consideradas pelo mecanismo de elasticidade. O algoritmo 2 recebe como entrada um arquivo com o conteúdo a ser publicado no modelo, cada linha é dividida de forma que a primeira coluna seja utilizada como tópico e as colunas 2 e 3 sejam concatenadas para formar o corpo da mensagem, a mensagem é publicada e o loop recomeça até que o arquivo chegue ao fim. O algoritmo 3 por sua vez, recebe o mesmo arquivo, gera uma lista de tópicos únicos e assina cada um deles. Desta forma, o algoritmo 2 envia dados e o 3 recebe dados das assinaturas realizadas, simulando o funcionamento esperado de uma arquitetura *Publish/Subscribe*.

Algoritmo 2: Pseudo-linguagem do processo gerador de tráfego.

```
   Entrada: arquivo com conteúdo a ser enviado
   Saída: publicações destinadas ao modelo
1  início
2  |  para (i=0; i < tamanho_arquivo; i++) faça
3  |  |  topico = divide_linha(linha_do_arquivo[i], coluna_1);
4  |  |  mensagem = divide_linha(linha_do_arquivo[i], coluna_2+coluna_3);
5  |  |  publica_mensagem(topico, mensagem, ip_do_servidor);
6  |  fim
7  fim
```

Algoritmo 3: Pseudo-linguagem do processo assinante.

```
   Entrada: arquivo com conteúdo a ser enviado
   Saída: assinaturas destinadas ao modelo
1  início
2  |  para (i=0; i < tamanho_arquivo; i++) faça
3  |  |  topico = divide_linha(linha_do_arquivo[i], coluna_1);
4  |  |  se (valida_lista_topicos_ja_publicados (topico) == 0) então
5  |  |  |  publica_mensagem(topico, mensagem, ip_do_servidor);
6  |  |  |  insere_lista_topicos_ja_publicados (topico);
7  |  |  fim
8  |  fim
9  fim
```

5.2 Infraestrutura de Nuvem

O ambiente selecionado para a realização dos testes foi o laboratório C01 413 localizado no Programa de Pós-Graduação em Computação Aplicada (PIPCA) da Universidade do Vale do Rio dos Sinos (Unisinos)[2]. Este laboratório possui 11 equipamentos homogêneos com processadores de dois núcleos de 2.9 GHz e 4 GB de memória, interconectados através

[2]http://www.unisinos.br/

de uma rede 100 Mbps. A plataforma de nuvem OpenNebula[3] versão 4.12.1 foi configurada nestes equipamentos, dos quais em um deles foi instalado o servidor OpenNebula para servir como o *Front-End* da nuvem. Os 10 equipamentos restantes foram configurados como nós OpenNebula para receberem e executarem máquinas virtuais. Além disso, o *Front-End* também serviu como repositório de arquivos e imagens de máquinas virtuais. A versão 9.9.5 do Bind foi utilizada no servidor DNS.

5.3 Cenários

Buscando avaliar o impacto de diferentes configurações de *thresholds* no desempenho e consumo de recursos das aplicações, foram definidos três cenários diferentes para a execução dos testes:

- Cenário 1, sem elasticidade: Inicialmente foram realizados testes para analisar o desempenho da solução sem utilizar elasticidade, ou seja, utilizando apenas uma réplica/instância de cada componente. Este cenário foi utilizado para permitir análises que buscam identificar o ganho de desempenho obtido com o modelo PSElastic.

- Cenário 2, elasticidade na camada de *Brokers*: o objetivo deste cenário é validar o funcionamento da elasticidade na camada de *Brokers* com diferentes *thresholds* e duas implementações de *Brokers* diferentes. Os seguintes ambientes foram utilizados:

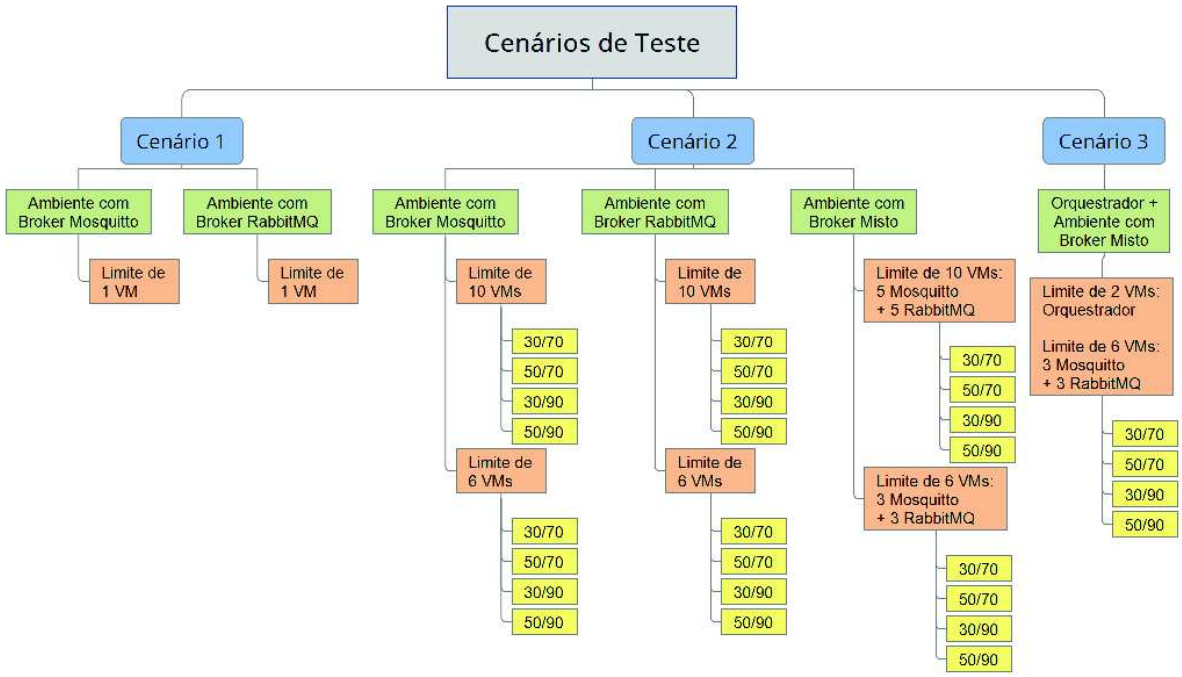
  - Ambiente com *Broker* Mosquitto: neste ambiente foram realizados testes permitindo a criação de 6 e depois 10 réplicas de VMs de *Broker* Mosquitto.
  - Ambiente com *Broker* RabbitMQ: neste ambiente foram realizados testes permitindo a criação de 6 e depois 10 réplicas de VMs de *Broker* RabbitMQ.
  - Ambiente misto: neste ambiente 50% das réplicas criadas foram de *Broker* Mosquitto e os outros 50% de *Broker* RabbitMQ.

- Cenário 3, elasticidade multinível: Tendo em vista que a elasticidade da camada de *Brokers* foi validada no Cenário 2, o principal objetivo deste cenário é validar o funcionamento da elasticidade na camada do Orquestrador. Para isto, foram utilizados duas réplicas de VMs de Orquestrador, assim como três réplicas de *Broker* Mosquitto e três de *Broker* RabbitMQ.

A Figura 19 apresenta de uma forma visual os cenários de teste. Onde os retângulos azuis correspondem aos cenários, retângulos verdes ao tipo de ambiente e *Broker* utilizado e retângulos laranjas correspondem ao limite de réplicas estabelecido. Por fim, retângulos

[3]http://opennebula.org/

Figura 19: Cenários de teste utilizados. Retângulos azuis correspondem aos cenários, retângulos verdes ao tipo de ambiente e *Broker* utilizado, retângulos laranjas correspondem ao limite de réplicas estabelecido e retângulos amarelos correspondem aos *thresholds* utilizados.



Fonte: elaborado pelo autor.

amarelos correspondem aos *thresholds* utilizados, sendo o primeiro valor correspondente ao *threshold* inferior (T_i) e o segundo ao *threshold* superior (T_s).

Para elasticidade nos Cenários 2 e 3 foram utilizados *thresholds* de 70% e 90% para o limite superior e 30% e 50% para o limite inferior[4]. Todas as execuções iniciaram a partir do mesmo cenário que consiste em um único nó com duas máquinas virtuais (igual ao número de núcleos da máquina). As principais métricas de comparação foram, quantidade de mensagens por segundo que cada cenário tratou e o tempo despendido para processar o conjunto de dados utilizado.

Para simular a entrada de dados no modelo, nos Cenários 1 e 2 foram utilizados quatro servidores virtuais e em cada um deles foram instanciados cinco processos geradores de tráfego (*Publishers*) e cinco processos assinantes (*Subscribers*), estes processos foram definidos nos Algoritmos 2 e 3 na Subseção 5.1.3. Como entrada dos processos geradores de tráfego foram utilizadas partes de um conjunto de dados (BRACCIALE et al., 2014) com mais de 21 milhões de registros com a seguinte estrutura: identificação do usuário, localização e um *timestamp* (data/hora). Cada registro descreve a localização precisa de um dos 316 motoristas de táxi na cidade de Roma em um determinado instante de tempo durante o período de 30 dias, sendo entre 1 de fevereiro de 2014 e 2 de março de 2014.

[4]Estes são considerados representativos para validar elasticidade em ambientes ambientes de nuvem (AL-SHISHTAWY; VLASSOV, 2013; RIGHI et al., 2015; SHARMA et al., 2011).

Os primeiros 400 mil registros do conjunto de dados foram distribuídos entre todos os processos geradores de tráfego, resultando em 20 mil registros para cada processo. O campo de identificação do usuário foi utilizado como tópico das publicações e o restante do registro (localização e *timestamp*) foi utilizado como mensagem. Os processos coletores por sua vez recebem a mesma fatia do conjunto de dados, geram uma lista de usuários existentes nesta fatia e fazem assinaturas baseadas nas identificações de usuários listadas.

No Cenário 3 a mesma estratégia de simulação foi utilizada, no entanto uma taxa de transferência (*throughput*) maior foi necessária para tornar a elasticidade necessária. Neste cenário foram utilizados oito servidores virtuais e em cada um deles foram instanciados cinco processos geradores de tráfego (*Publishers*) e cinco processos assinantes (*Subscribers*). Os primeiros 800 mil registros do conjunto de dados foram distribuídos entre todos os processos geradores de tráfego, resultando em 20 mil registros para cada processo.

5.4 Métricas

Para a avaliação dos resultados obtidos em cada ambiente, foram definidas métricas para avaliação de questões de desempenho da aplicação bem como eficiência na utilização dos recursos. As métricas utilizadas são:

- Tempo de Execução: esta métrica corresponde ao tempo total, em segundos, despendido pela aplicação para tratamento da carga de dados.

- Vazão (Mensagens por Segundo): corresponde ao total de mensagens enviadas para cada ambiente divido pelo tempo de execução do mesmo. Seria o equivalente a taxa de transferência (*throughput*) da solução.

- Custo Total de Execução: A Equação 5.1 apresenta a métrica Cte, onde j corresponde ao número de máquinas virtuais utilizadas, $pt_e(j)$ é a fatia de tempo em que a aplicação foi executada nestas j máquinas virtuais. Sendo assim, o custo corresponde a quantidade total de recursos utilizados durante uma execução.

- Eficiência: A Equação 5.2 apresenta a métrica Efc, onde j corresponde ao número de máquinas virtuais utilizadas, $pt_e(j)$ é a fatia de tempo em que a aplicação foi executada nestas j máquinas virtuais e t_e corresponde ao tempo total de execução do ambiente. Sendo assim, eficiência corresponde a quantidade de recursos utilizados em cada segundo de execução.

$$Cte = \sum_{j=1}^{s} j \times pt_e(j) \tag{5.1}$$

$$Efc = \sum_{j=1}^{s}(\frac{j \times pt_e(j)}{t_e}) \tag{5.2}$$

5.5 Considerações Parciais

Os protótipos de Orquestrador, Gerenciador e Clientes foram abordados na Seção 5.1, assim como a infraestrutura de nuvem utilizada na Seção 5.2. Na Seção 5.3 foram apresentados os cenários de avaliação sem elasticidade, com elasticidade na camada de *Broker* e com elasticidade multinível. Por fim, as métricas tempo de execução, mensagens por segundo, custo e eficiência foram apresentadas na Seção 5.4.

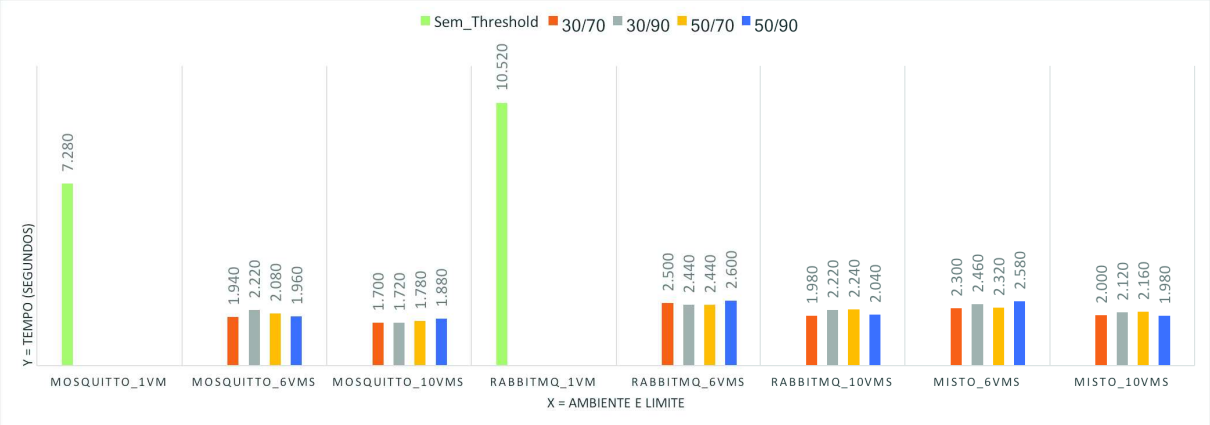
6 RESULTADOS OBTIDOS

Este Capítulo descreve os resultados obtidos em todos os testes, com todos ambientes de todos cenários. Os resultados associados a métrica Tempo de Execução são apresentados nas Seções 6.1 e 6.2. Na Seção 6.3 são abordados os resultados utilizando a métrica de Mensagens por Segundo. As métricas Custo (Cte) e Eficiência (Efc) são utilizadas nos resultados da Seção 6.4. A Seção 6.5 apresenta exclusivamente os resultados obtidos no cenário 3, que utiliza a elasticidade multinível. As considerações parciais são abordadas na Seção 6.6.

6.1 Análise de Desempenho: Tempo de execução

A Figura 20 apresenta o tempo despendido para o tratamento da carga de trabalho em cada configuração de ambiente. O eixo X refere-se ao tipo de *Broker* utilizado e limite de réplicas permitido. O eixo Y corresponde ao tempo contabilizado em segundos e cada barra do gráfico corresponde a uma configuração de *threshold* utilizada. Onde o primeiro valor é o *threshold* inferior (T_i) e o segundo valor o *threshold* superior (T_s).

Figura 20: Tempo total de execução dos testes.



Fonte: elaborado pelo autor.

O ambiente Mosquitto_1VM obteve o melhor resultado dentre as execuções do cenário de avaliação 1 (sem elasticidade). Este ambiente levou 7.280 segundos (121,3 minutos) para ser finalizado e corresponde a execução dos testes utilizando apenas uma réplica de máquina virtual do *Broker* Mosquitto. Neste caso, nenhum *threshold* foi utilizado pois a elasticidade estava desabilitada. O ambiente RabbitMQ_1VM corresponde a execução dos testes utilizando apenas uma réplica de máquina virtual do *Broker* RabbitMQ e levou 10.520 segundos (175,3 minutos) para ser finalizado, sendo 44,5% mais lento que o ambiente Mosquitto.

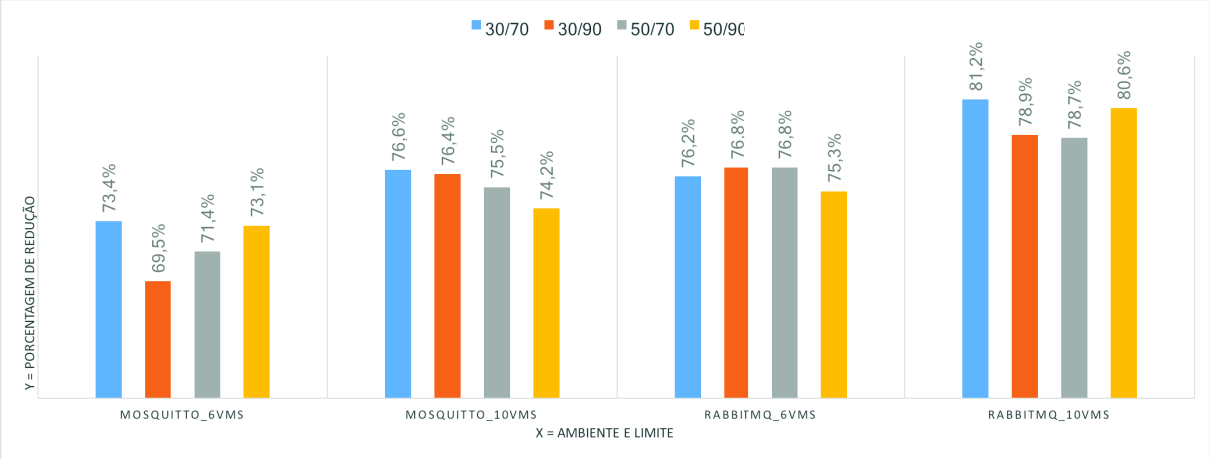
Com a utilização do limite de criação de 6 réplicas de VMs, o melhor resultado foi obtido pelo ambiente Mosquitto_6VM que levou 1.940 segundos (32,3 minutos) para

ser finalizado utilizando T_i de 30 e T_s de 70. O pior caso foi obtido no ambiente RabbitMQ_6VM utilizando T_i de 50 e T_s de 90 que levou 2.600 segundos (43,3 minutos) para ser finalizado, sendo 34% mais lento que o ambiente Mosquitto_6VM. Ainda assim o ambiente RabbitMQ_6VM obteve um resultado 64,3% mais rápido que o ambiente Mosquitto_1VM, melhor ambiente do cenário 1.

Com o limite de criação de réplicas configurado para 10 VMs, a execução foi finalizada em 1.700 segundos (28,3 minutos) no ambiente Mosquitto_10VM utilizando T_i de 30 e T_s de 70. O pior resultado foi registrado no ambiente RabbitMQ_10VM, finalizado em 2.240 segundos (37,3 minutos) utilizando T_i de 50 e T_s de 70, sendo 31,7% mais lento que o ambiente Mosquitto_10VM.

O porcentagem de redução no tempo de execução utilizando o modelo PSElastic é apresentada na Figura 21. Os ambientes Mosquitto_6VMs e Mosquitto_10VMs são comparados ao ambiente Mosquitto_1VM que utiliza o mesmo *Broker*, assim como os ambientes RabbitMQ_6VMs e RabbitMQ_10VMs são comparados ao ambiente RabbitMQ_1VM. Conforme informações exibidas na Figura, O modelo PSElastic reduziu em até 76,6% o tempo necessário para o tratamento da carga de dados em ambientes executando o *Broker* Mosquitto. A vantagem foi ainda maior para ambiente executando o *Broker* RabbitMQ, chegando a 81,2% de redução.

Figura 21: Redução no tempo de execução.



Fonte: elaborado pelo autor.

De modo geral, o melhor resultado obtido utilizando elasticidade foi no ambiente Mosquitto_10VM com T_i de 30 e T_s de 70 onde a execução foi finalizada em 1.700 segundos (28,3 minutos). O pior caso foi obtido no ambiente RabbitMQ_6VM utilizando T_i de 50 e T_s de 90 que levou 2.600 segundos (43,3 minutos) para ser finalizado.

6.2 Alocação de Recursos

A Tabela 5 apresenta o tempo de utilização dos recursos virtuais em cada configuração de ambiente. Por exemplo, o ambiente Moquitto com limite de 10 VM utilizando T_i 50 e T_s 90 utilizou 2 VMs durante 160 segundos, 4 VMs durante 440 segundos, 6 VMs durante 180, 8 VMs durante 200 segundos e 10 VMs durante 900 segundos, totalizando 1.880 segundos de execução.

Tabela 5: Tempo x Quantidade de Máquinas Virtuais.

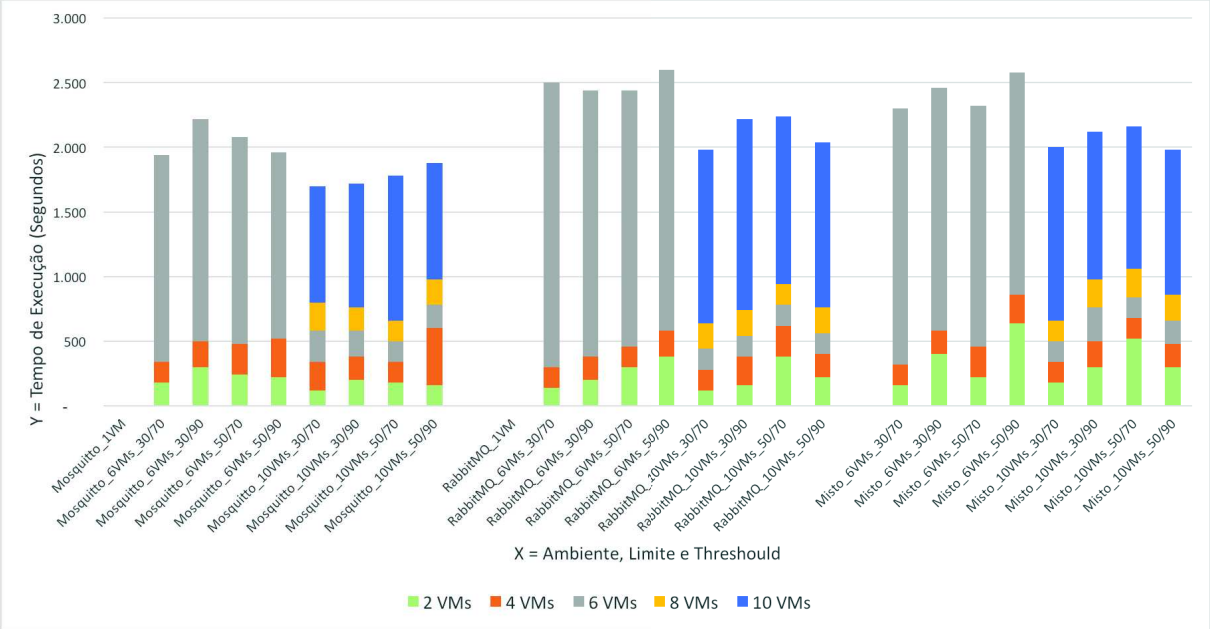
Ambiente	Limite	Threshold	1 VM	2 VMs	4 VMs	6 VMs	8 VMs	10 VMs	Total (Seg)
Mosquitto	1 VM	-	7.280	-	-	-	-	-	7.280
	6 VMs	30/70	-	180	160	1.600	-	-	1.940
		30/90	-	300	200	1.720	-	-	**2.220**
		50/70	-	240	240	1.600	-	-	2.080
		50/90	-	220	300	1.440	-	-	1.960
	10 VMs	30/70	-	120	220	240	220	900	**1.700**
		30/90	-	200	180	200	180	960	1.720
		50/70	-	180	160	160	160	1.120	1.780
		50/90	-	160	440	180	200	900	1.880
RabbitMQ	1 VM	-	10.520	-	-	-	-	-	10.520
	6 VMs	30/70	-	140	160	2.200	-	-	2.500
		30/90	-	200	180	2.060	-	-	2.440
		50/70	-	300	160	1.980	-	-	2.440
		50/90	-	380	200	2.020	-	-	**2.600**
	10 VMs	30/70	-	120	160	160	200	1.340	**1.980**
		30/90	-	160	220	160	200	1.480	2.220
		50/70	-	380	240	160	160	1.300	2.240
		50/90	-	220	180	160	200	1.280	2.040
Misto	6 VMs	30/70	-	160	160	1.980	-	-	2.300
		30/90	-	400	180	1.880	-	-	2.460
		50/70	-	220	240	1.860	-	-	2.320
		50/90	-	640	220	1.720	-	-	**2.580**
	10 VMs	30/70	-	180	160	160	160	1.340	**2.000**
		30/90	-	300	200	260	220	1.140	2.120
		50/70	-	520	160	160	220	1.100	2.160
		50/90	-	300	180	180	200	1.120	1.980

A Figura 22 exibe de forma gráfica o tempo de utilização dos recursos virtuais em cada configuração de ambiente. O eixo X refere-se ao tipo de *Broker* utilizado, limite de réplicas permitido e *thresholds* utilizados. O eixo Y corresponde ao tempo contabilizado em segundos e cada barra empilhada do gráfico corresponde a quantidade de recursos ocupados.

A Figura 23 apresenta o estado do *threshold* em cada leitura realizada durante a execução em cada configuração de ambiente. O eixo X refere-se ao tipo de *Broker* utilizado, limite de réplicas permitido e *thresholds* utilizados. O eixo Y corresponde a quantidade de coletas realizadas e cada barra empilhada do gráfico corresponde estado dos *thresholds*.

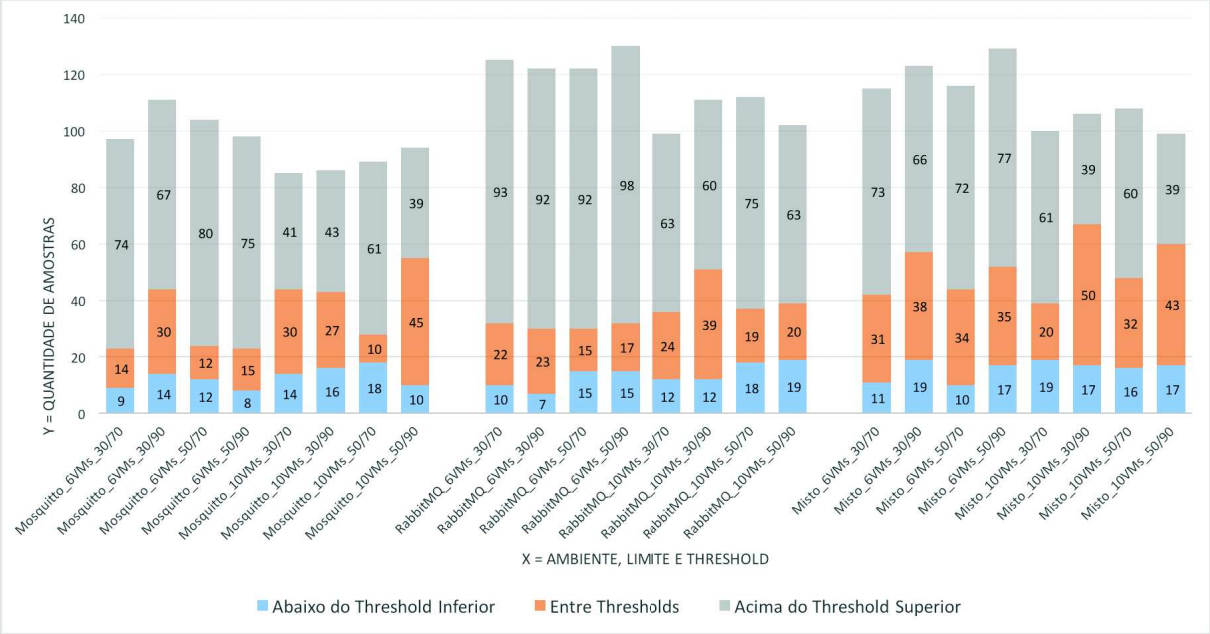
Os ambientes Mosquitto_10Vms com T_i 50 e T_s 90, Misto_10VMs com T_i 30 e T_s 90 e Misto_10VMs com T_i 50 e T_s 90 obtiveram 39 leituras acima do T_s, o menor número registrado nos testes. Demonstrando assim, que são os ambientes onde a aplicação passou menos tempo sobrecarregada. Por outro lado, o ambiente com maior sobrecarga foi o RabbitMQ_6VMs com T_i 50 e T_s 90, registrando 98 leituras acima do T_s.

Figura 22: Alocação de recursos.



Fonte: elaborado pelo autor.

Figura 23: Estado do *threshold*.



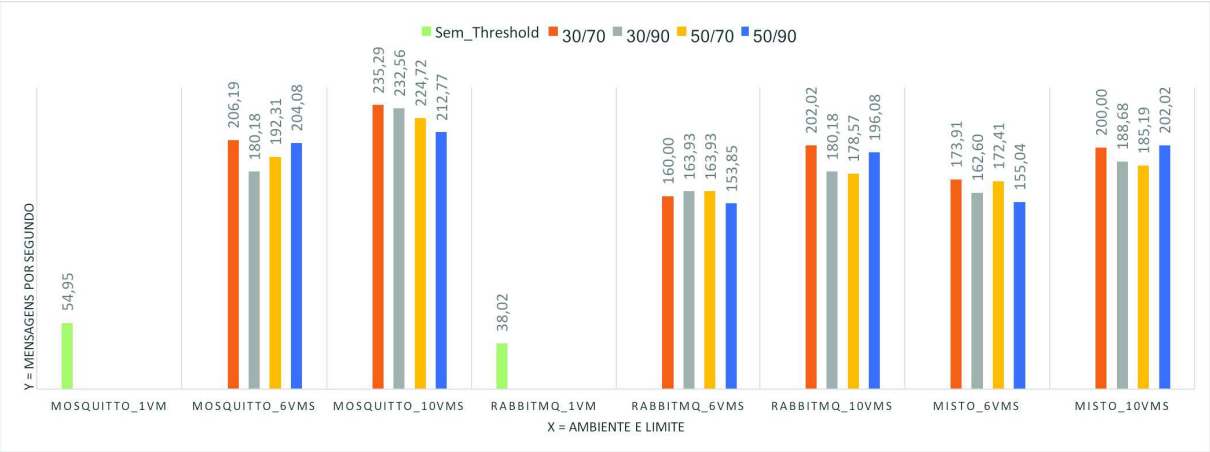
Fonte: elaborado pelo autor.

6.3 Análise de Vazão: Mensagens por Segundo

A Figura 24 apresenta a quantidade de mensagens por segundo tratadas em cada configuração de ambiente. O eixo X refere-se ao tipo de *Broker* utilizado e limite de réplicas permitido. O eixo Y corresponde a quantidade de mensagens por segundo (Mps) e cada barra do gráfico corresponde a uma configuração de *threshold* utilizada. Onde o

primeiro valor é o *threshold* inferior (T_i) e o segundo valor o *threshold* superior (T_s).

Figura 24: Quantidade de mensagens por segundo tratadas em cada ambiente.



Fonte: elaborado pelo autor.

O ambiente Mosquitto_1VM obteve o melhor resultado dentre as execuções do cenário de avaliação 1 (sem elasticidade), este ambiente tratou 54,95 Mps. O ambiente RabbitMQ_1VM obteve o desempenho de 38,02 Mps, 30,8% menor que o anterior.

Com a utilização do limite de criação de 6 réplicas de VMs, o melhor resultado foi obtido pelo ambiente Mosquitto_6VM utilizando T_i de 30 e T_s de 70 que tratou 206,19 Mps. O pior caso foi obtido no ambiente RabbitMQ_6VM utilizando T_i de 50 e T_s de 90 com desempenho de 153,85 Mps.

Com o limite de criação de réplicas configurado para 10 VMs, 235,29 Mps foram tratadas no ambiente Mosquitto_10VM utilizando T_i de 30 e T_s de 70. O pior resultado foi registrado no ambiente RabbitMQ_10VM, 178,57 Mps foram tratadas utilizando T_i de 50 e T_s de 70, sendo 24,1% menor que o ambiente Mosquitto_10VM.

De modo geral, o melhor resultado obtido utilizando elasticidade foi no ambiente Mosquitto_10VM com T_i de 30 e T_s de 70 que tratou 235,29 Mps. O pior caso foi obtido no ambiente RabbitMQ_6VM utilizando T_i de 50 e T_s de 90 com desempenho de 153,85 Mps.

6.4 Análise de Custo e Eficiência na Utilização de Recursos

Na Tabela 6 são apresentados os resultados do cálculo de custo de execução de cada ambiente testado. O cálculo de custo corresponde ao total de recursos utilizados durante a execução dos testes. Sendo assim, enquanto menor o custo, melhor.

A Figura 25 apresenta de forma gráfica as informações contidas na Tabela 6. O eixo X refere-se ao tipo de *Broker* utilizado e limite de réplicas permitido. O eixo Y corresponde ao valor de custo calculado e cada barra do gráfico corresponde a uma configuração de *threshold* utilizada. Onde o primeiro valor é o *threshold* inferior (T_i) e o segundo valor o

Tabela 6: Tabela de custo dos ambiente testados.

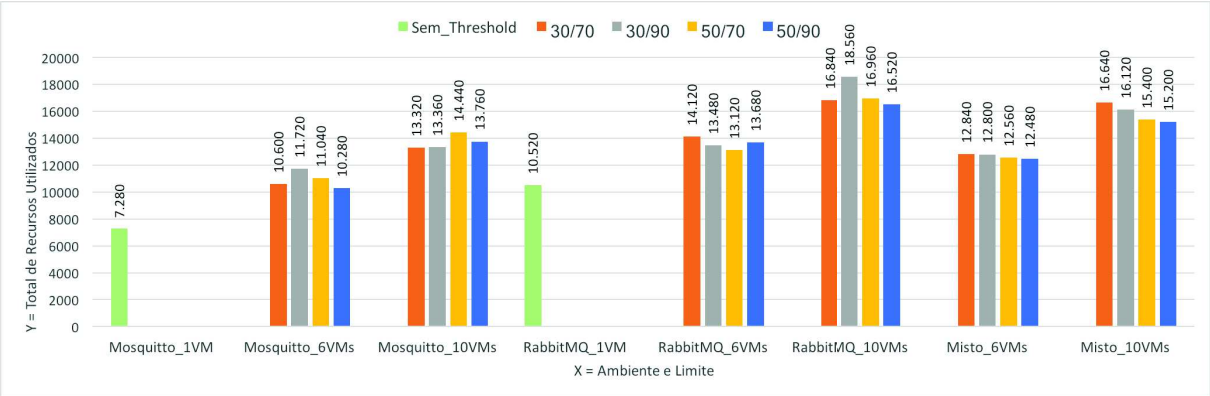
Ambiente	Recursos	Sem_TS	30/70	30/90	50/70	50/90
Mosquito	1VM	7.280	-	-	-	-
	6VMs	-	10.600	11.720	11.040	**10.280**
	10VMs	-	13.320	13.360	14.440	13.760
RabbitMQ	1VM	10.520	-	-	-	-
	6VMs	-	14.120	13.480	13.120	13.680
	10VMs	-	16.840	**18.560**	16.960	16.520
Misto	6VMs	-	12.840	12.800	12.560	12.480
	10VMs	-	16.640	16.120	15.400	15.200

Fonte: elaborado pelo autor.

threshold superior (T_s).

O melhor resultado associado ao custo foi obtido pelo ambiente Mosquitto_6VM com T_i de 50 e T_s de 90, utilizando 10.280 recursos para finalizar a execução. O pior caso foi obtido no ambiente RabbitMQ_10VM com T_i de 30 e T_s de 90 utilizando 18.560 recursos durante a execução.

Figura 25: Custo total de execução dos ambientes testados.



Fonte: elaborado pelo autor.

A Tabela 7 por sua vez apresenta os resultados do cálculo de eficiência de cada ambiente testado. O cálculo de eficiência corresponde a quantidade de recursos (Máquinas Virtuais) utilizados em cada segundo da execução dos testes. Sendo assim, enquanto menor a quantidade de recursos, mais eficiente é o ambiente.

O melhor resultado foi obtido pelo ambiente Mosquitto_6VM com T_i de 50 e T_s de 90, utilizando uma média de 5,245 VMs em cada segundo de execução. O pior caso foi obtido no ambiente RabbitMQ_10VM com T_i de 30 e T_s de 70 utilizando uma média de 8,505 VMs em cada segundo de execução.

Tabela 7: Tabela de eficiência dos ambiente testados.

Ambiente	Recursos	Sem_TS	30/70	30/90	50/70	50/90
Mosquito	1VM	1,00	-	-	-	-
	6VMs	-	5,464	5,279	5,308	**5,245**
	10VMs	-	7,835	7,767	8,112	7,319
RabbitMQ	1VM	1,00	-	-	-	-
	6VMs	-	5,648	5,525	5,377	5,262
	10VMs	-	**8,505**	8,360	7,571	8,098
Misto	6VMs	-	5,583	5,203	5,414	4,837
	10VMs	-	8,320	7,604	7,130	7,677

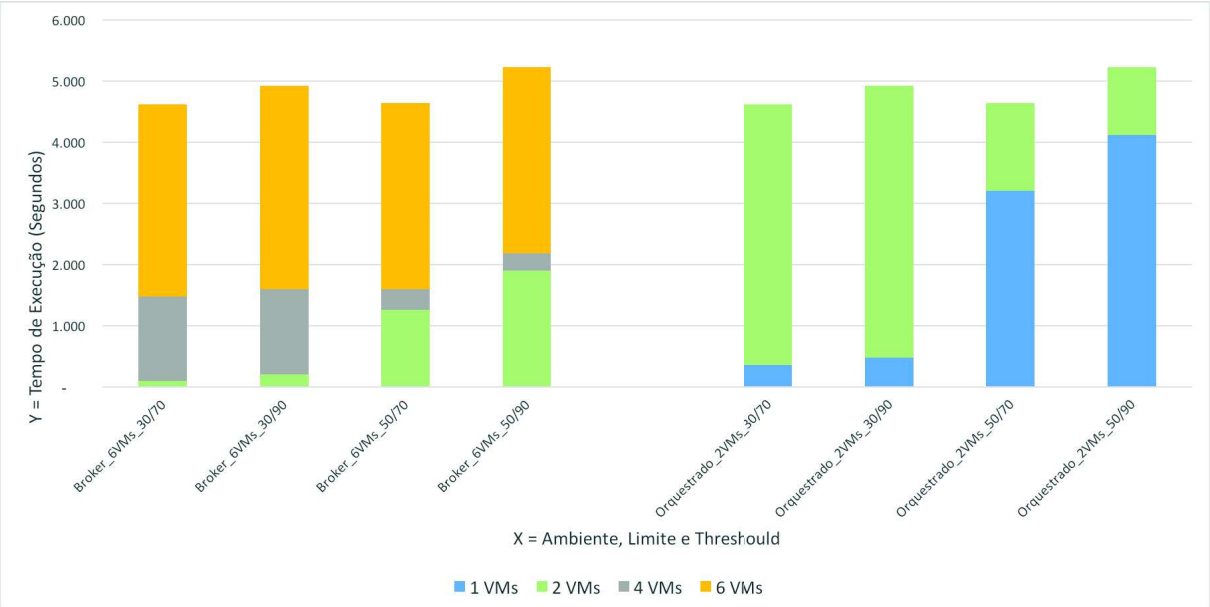
Fonte: elaborado pelo autor.

6.5 Alocação de Recursos no Cenário com Elasticidade Multinível

Conforme definição apresentada na Seção 5.3 do Capítulo 5.3, o principal objetivo do cenário 3 é validar o funcionamento da elasticidade na camada do Orquestrador. Para isto, foram utilizados duas réplicas de VMs de Orquestrador, assim como três réplicas de *Broker* Mosquitto e três de *Broker* RabbitMQ.

A Tabela 8 apresenta o tempo de utilização dos recursos virtuais em cada configuração de ambiente. Por exemplo, o ambiente de *Broker* com limite de 6 VM utilizando T_i 50 e T_s 90 utilizou 2 VMs durante 1.900 segundos, 4 VMs durante 280 segundos, 6 VMs durante 3.040, totalizando 5.220 segundos de execução. Durante o mesmo teste, a camada de Osquestrador utilizou 1 VM durante 4.120 segundos e 2 VMs durante 1.100, totalizando os mesmos 5.220 segundos de execução. A Figura 26 corresponde ao gráfico de alocação de recursos dos testes do cenário 3.

Figura 26: Alocação de recursos no Cenário 3.



Fonte: elaborado pelo autor.

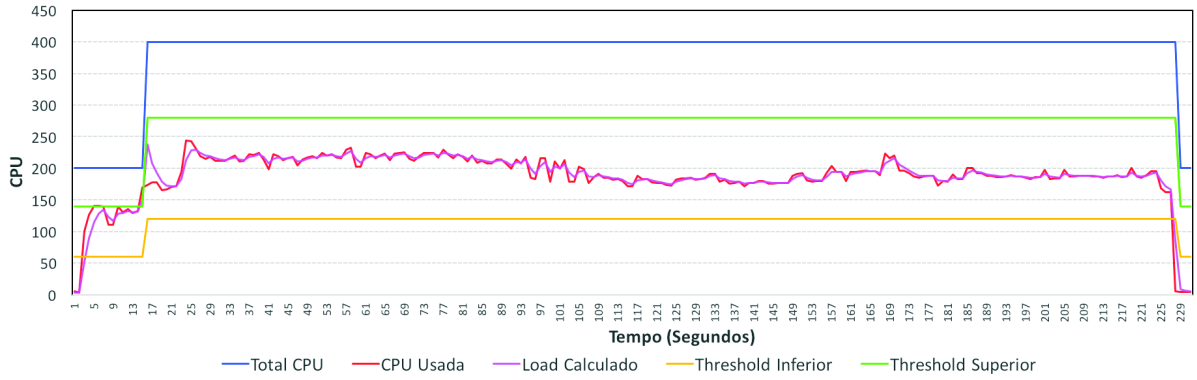
Tabela 8: Cenário 3: Tempo x Quantidade de Máquinas Virtuais.

Ambiente	Limite	Threshold	1 VM	2 VMs	4 VMs	6 VMs	Total (Seg)
Broker	6 VMs	30/70	-	100	1.380	3.140	**4.620**
		30/90	-	200	1.400	3.320	4.920
		50/70	-	1.260	340	3.040	4.640
		50/90	-	1.900	280	3.040	**5.220**
Osquestrador	2 VMs	30/70	360	4.260	-	-	**4.620**
		30/90	480	4.440	-	-	4.920
		50/70	3.200	1.440	-	-	4.640
		50/90	4.120	1.100	-	-	**5.220**

Fonte: elaborado pelo autor.

A Figura 27 apresenta o gráfico de utilização de CPU dos Orquestradores durante a execução do teste do Cenário 3 com T_i 30 e T_s 70, onde a linha azul representa o total de CPU alocado no instante da coleta e a linha vermelha a utilização total de CPU do ambiente. A linha rosa corresponde ao *Load* calculado (utilizando *Aging*) do ambiente. As linhas amarela e verde correspondem ao T_i e T_s respectivamente.

Figura 27: Utilização CPU da camada de Orquestrador com T_i de 30 e T_s de 70.



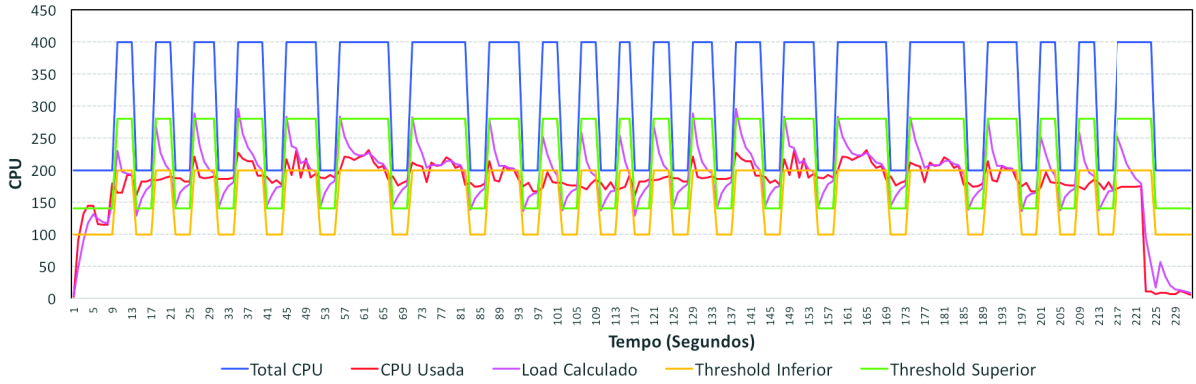
Fonte: elaborado pelo autor.

A Figura 28 por sua vez, apresenta a mesma informação que a Figura 27, mas referente ao Cenário 3 com T_i 50 e T_s 70. Estas duas figuras foram selecionadas para demonstrar o impacto que a configuração de *thresholds* pode causar ao ambiente com elasticidade.

No teste representado pela Figura 27, a segunda réplica de Orquestrador é instanciada por volta da observação 13. Esta ação é executada pois o processamento total do ambiente ultrapassa o valor de T_s configurado. O ambiente se mantem com duas réplicas até a finalização do teste.

Durante a execução representada pela Figura 28 a segunda réplica de Orquestrador também é instanciada por volta da observação 13. No entanto, a segunda réplica é desativada logo após sua criação, pois o processamento total do ambiente fica abaixo do T_i. As ações de ativação e desativação da segunda réplica são repetidas até o final do teste. Este comportamento foi registrado pois o processamento total do ambiente com uma réplica fica acima do T_s e com duas réplicas fica abaixo do T_i.

Figura 28: Utilização CPU da camada de Orquestrador com T_i de 50 e T_s de 70.



Fonte: elaborado pelo autor.

6.6 Considerações Parciais

Levando em consideração a métrica Tempo de Execução, o melhor resultado obtido foi no ambiente Mosquitto_10VM com T_i de 30 e T_s de 70 onde a execução foi finalizada em 1.700 segundos (28,3 minutos). O pior caso foi obtido no ambiente RabbitMQ_6VM utilizando T_i de 50 e T_s de 90 que levou 2.600 segundos (43,3 minutos) para ser finalizado. Conforme informações exibidas na Figura 21, o modelo PSElastic reduziu em até 76,6% o tempo necessário para o tratamento da carga de dados em ambientes executando o *Broker* Mosquitto. A vantagem foi ainda maior para ambiente executando o *Broker* RabbitMQ, chegando a 81,2% de redução.

Se a avaliação for baseada no estado dos *thresholds*, os ambientes Mosquitto_10VMs com T_i 50 e T_s 90, Misto_10VMs com T_i 30 e T_s 90 e Misto_10VMs com T_i 50 e T_s 90 obtiveram 39 leituras acima do T_s, o menor número registrado nos testes. Demonstrando assim, que são os ambientes onde a aplicação passou menos tempo sobrecarregada. Por outro lado, o ambiente com maior sobrecarga foi o RabbitMQ_6VMs com T_i 50 e T_s 90, registrando 98 leituras acima do T_s.

No quesito Mensagens por Segundo, o melhor resultado foi obtido no ambiente Mosquitto_10VM com T_i de 30 e T_s de 70 que tratou 235,29 Mps. O pior caso foi obtido no ambiente RabbitMQ_6VM utilizando T_i de 50 e T_s de 90 com desempenho de 153,85 Mps.

Considerando a métrica referente ao custo total de execução, o melhor resultado foi obtido pelo ambiente Mosquitto_6VM com T_i de 50 e T_s de 90, utilizando 10.280 recursos para finalizar a execução. O pior caso foi obtido no ambiente RabbitMQ_10VM com T_i de 30 e T_s de 90 utilizando 18.560 recursos durante a execução.

O melhor resultado em Eficiência foi obtido pelo ambiente Mosquitto_6VM com T_i de 50 e T_s de 90, utilizando uma média de 5,245 VMs em cada segundo de execução. O pior caso foi obtido no ambiente RabbitMQ_10VM com T_i de 30 e T_s de 70 utilizando

uma média de 8,505 VMs em cada segundo de execução. Como observado nos testes, o ambiente mais rápido não necessariamente será o mais eficiente.

Levando em consideração o cenário 3 de testes, a quantidade de requisições por segundo geradas pelos clientes disponíveis foi suficiente para validar o correto funcionamento da Elasticidade Multinível. No teste utilizando T_i 30 e T_s 70 a camada de Orquestrador alocou a segunda VM no início da execução e a liberou somente no final do teste. Por outro lado, com a utilização de T_i 50 e T_s 70 ações de ativação e desativação da segunda réplica são repetidas com frequência no decorrer de todo teste.

7 CONSIDERAÇÕES FINAIS

A elasticidade em nuvem é largamente explorada em arquiteturas cliente servidor, como vídeo sob demanda, lojas online, *e-governance* e *Web Services* (RAVEENDRAN; BICER; AGRAWAL, 2011). No entanto este recurso é pouco empregado no contexto de Internet das Coisas (TRAN; SKHIRI; ZIMÁNYI, 2011; LI et al., 2011; BARAZZUTTI et al., 2014). Conforme informações apresentadas na Seção 2.2 do Capítulo 2, os *Brokers* presentes na Camada de *Middleware* da Internet das Coisas centralizam grande parte da comunicação entre dispositivos (KHAN et al., 2012). Desta forma, implementações de grande escala têm potencial para criar um enorme tráfego agregado com destino aos *Brokers*, causando congestionamento e reduzindo assim a taxa de transferência (mensagens por segundo) do ambiente (XU; MAHENDRAN; RADHAKRISHNAN, 2016).

O presente trabalho apresentou um modelo de elasticidade multinível para *Brokers* de aplicações que utilizam o modelo de comunicação *Publish/Subscribe* chamado **PSElastic**. Ele busca proporcionar elasticidade automática reativa para *Brokers Publish/Subscribe* de forma transparente para usuários, aplicações e implementações de *Brokers*, tratando assim a escalabilidade e desempenho da solução. Para tal, fornece um arcabouço (*framework*) com um Gerenciador de operações de elasticidade horizontal, e um Orquestrador que trata o encaminhamento de mensagens sem a necessidade de modificações ou adaptações nas aplicações.

A arquitetura de PSElastic foi apresentada no Capítulo 4, assim como a metodologia utilizada para avaliação no Capítulo 5. Os conceitos fundamentais para a compreensão do trabalho e da pesquisa proposta foram abordados no Capítulo 2; o Capítulo 3 por sua vez apresentou uma lista com trabalhos relacionados ao tema desta pesquisa. Os resultados obtidos utilizando o modelo PSElastic foram abordados no Capítulo 6, assim como as contribuições do modelo na Seção 7.1 e os trabalhos futuros na Seção 7.2 do presente Capítulo.

Conforme resultados apresentados no Capítulo 6, o modelo PSElastic mostrou-se eficiente ao reduzir em até 76,6% o tempo necessário para o tratamento da carga de dados em ambientes executando o *Broker* Mosquitto. A vantagem foi ainda maior para ambiente executando o *Broker* RabbitMQ, chegando a 81,2% de redução.

No quesito Mensagens por Segundo, o modelo PSElastic aumentou o *throughput* do ambiente Mosquitto de 54,95 Mps para 235,29 Mps. No ambiente RabbitMQ o *throughput* aumentou de 38,02 Mps para 202,02 Mps.

Levando em consideração a métrica referente ao custo total de execução, o melhor resultado foi obtido pelo ambiente Mosquitto_6VM com T_i de 50 e T_s de 90, utilizando 10.280 recursos para finalizar a execução. O pior caso foi obtido no ambiente RabbitMQ_10VM com T_i de 30 e T_s de 90 utilizando 18.560 recursos durante a execução.

Utilizando a métrica Eficiência, o melhor resultado foi obtido pelo ambiente Mos-

quitto_6VM com T_i de 50 e T_s de 90, utilizando uma média de 5,245 VMs em cada segundo de execução. O pior caso foi obtido no ambiente RabbitMQ_10VM com T_i de 30 e T_s de 70 utilizando uma média de 8,505 VMs em cada segundo de execução. Como observado nos testes, o ambiente mais rápido não necessariamente será o mais eficiente.

Levando em consideração o cenário 3 de testes, a quantidade de requisições por segundo geradas pelos clientes disponíveis foi suficiente para validar o correto funcionamento da Elasticidade Multinível. Foi possível identificar alterações no comportamento do ambiente elástico de acordo com os *thresholds* utilizados. Por exemplo, o teste utilizando T_i de 30 e T_s de 70 manteve a alocação de VMs estável, já o teste com T_i de 50 e T_s de 70 executou ações de ativação e desativação da segunda réplica repetidas vezes até o fim da execução.

Ainda, baseado em todos resultados obtidos pode-se afirmar que a implementação de *Broker* Mosquitto é mais rápida (Tempo de Execução), possui maior *throughput* (Mensagens por Segundo) e é mais eficiente (Eficiência) do que a implementação de *Broker* RabbitMQ.

7.1 Contribuições

As contribuições do modelo PSElastic focam em oferecer a elasticidade de recursos para *Brokers Publish/Subscribe* sem impor modificações no código das aplicações nem a alteração do *Broker* já utilizado no ambiente. Dessa maneira, oferece um ambiente elástico em que não seja necessária a configuração de regras e ações. Ainda, a elasticidade oferecida pelo modelo ocorre de forma transparente e assíncrona não bloqueando a execução das aplicações. Podem ser destacadas as seguintes contribuições do modelo PSElastic:

(i) Utilização no contexto de *Publish/Subscribe*, da técnica de **Aging** aplicada ao histórico da carga de processamento da aplicação para as tomadas de decisão do mecanismo de elasticidade;

(ii) **Modelo agnóstico** a *Broker*, possibilitando implementações menos impactantes aos serviços;

(iii) **Arquitetura** aberta e genérica, possibilitando sua adaptação para outros modelos de comunicação;

(iv) Utilização de **elasticidade multinível**, ampliando assim a escalabilidade da arquitetura.

A Tabela 9 ressalta as contribuições do modelo PSElastic comparando com os demais trabalhos analisados. Pode ser observado que o modelo suporta filtragens por tópico, conteúdo e atributos e possui suporte à elasticidade com método horizontal, modelo automático reativo e operações de replicação de VMs. Por fim, o PSElastic não exige alterações em código fonte de aplicações e não possui restrições quanto ao *Middleware* utilizado.

Tabela 9: Contribuições do modelo PSElastic comparado com os trabalhos relacionados com base nos quesitos previamente definidos.

Trabalho	**Filtragem**	**Suporta Elasticidade?**	**Método**	**Modelo**	**Operação**	**Modificações**	**Restrições de Plataforma**
E-STREAM HuB (BARAZZUTTI et al., 2014)	Conteúdo	Sim, multinível	Horizontal e Migração	Automático Reativo	Migração de processos e replicação de VMs	Sim, no *Middleware/Broker*	Se aplica apenas ao *Middleware* STREAMHuB
EQS (TRAN; SKHIRI; ZIMÁNYI, 2011)	Tópicos	Sim, somente em um nível	Horizontal e Migração	Automático Reativo	Migração de processos e replicação de VMs	Sim, no *Middleware/Broker*	o *Middleware* deve suportar migrações de processos
Blue Dove (LI et al., 2011)	Atributos	Somente Adição de Recursos, multinível	Horizontal	Automático Reativo	Replicação de VMs	Sim, no *Middleware/Broker*	Se aplica apenas a um *Middleware*
GSEC (WANG; MA, 2015)	Conteúdo	Sim, multinível	Horizontal	Automático Reativo	Replicação de VMs	Sim, no *Middleware/Broker*	Se aplica apenas a um *Middleware*
PSElastic	Tópicos, Conteúdo e Atributos	Sim	Horizontal	Automático Reativo	Replicação de VMs	Não	Não

Fonte: elaborado pelo autor.

7.2 Trabalhos Futuros

Algumas limitações do modelo PSElastic, que podem ser vistas como oportunidades de otimizações para trabalhos futuros são destacadas abaixo:

(i) Avaliar o modelo PSElastic em ambientes reais com uma demanda maior que a gerada em laboratório;

(ii) Avaliar a utilização do Orquestrador PSElastic em outros modelos de comunicação;

(iii) Acrescentar um método que avalie de forma proativa as decisões de elasticidade e comparar com o modelo atual.

REFERÊNCIAS

AL-SHISHTAWY, A.; VLASSOV, V. ElastMan: elasticity manager for elastic key-value stores in the cloud. In: ACM CLOUD AND AUTONOMIC COMPUTING CONFERENCE, 2013., 2013, New York, NY, USA. **Proceedings...** ACM, 2013. p. 7:1–7:10. (CAC '13).

ASLAM, S.; SHAH, M. A. Load balancing algorithms in cloud computing: a survey of modern techniques. In: NATIONAL SOFTWARE ENGINEERING CONFERENCE (NSEC), 2015., 2015. **Anais...** IEEE, 2015. p. 30–35.

ATZORI, L.; IERA, A.; MORABITO, G. The Internet of Things: a survey. **Computer Networks**, ., v. 54, n. 15, p. 2787 – 2805, 2010.

GARBINATO, B.; MIRANDA, H.; RODRIGUES, L. (Ed.). **Distributed Event Routing in Publish/Subscribe Systems**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. 219–244 p.

BALIGA, J.; AYRE, R.; HINTON, K.; TUCKER, R. Green Cloud Computing: balancing energy in processing, storage, and transport. **Proceedings of the IEEE**, New York, NY, USA, v. 99, n. 1, p. 149–167, 2011.

BARAZZUTTI, R.; FELBER, P.; FETZER, C.; ONICA, E.; PINEAU, J.-F.; PASIN, M.; RIVIÈRE, E.; WEIGERT, S. Streamhub: a massively parallel architecture for high-performance content-based publish/subscribe. In: ACM INTERNATIONAL CONFERENCE ON DISTRIBUTED EVENT-BASED SYSTEMS, 7., 2013, . **Proceedings...** ACM, 2013. p. 63–74.

BARAZZUTTI, R.; HEINZE, T.; MARTIN, A.; ONICA, E.; FELBER, P.; FETZER, C.; JERZAK, Z.; PASIN, M.; RIVIèRE, E. Elastic Scaling of a High-Throughput Content-Based Publish/Subscribe Engine. In: DISTRIBUTED COMPUTING SYSTEMS (ICDCS), 2014 IEEE 34TH INTERNATIONAL CONFERENCE ON, 2014. **Anais...** IEEE, 2014. p. 567–576.

BRACCIALE, L.; BONOLA, M.; LORETI, P.; BIANCHI, G.; AMICI, R.; RABUFFI, A. **CRAWDAD dataset roma/taxi (v. 2014-07-17)**. 2014.

CARZANIGA, A.; PAPALINI, M.; WOLF, A. L. Content-based Publish/Subscribe Networking and Information-centric Networking. In: ACM SIGCOMM WORKSHOP ON INFORMATION-CENTRIC NETWORKING, 2011, New York, NY, USA. **Proceedings...** ACM, 2011. p. 56–61. (ICN '11).

CASAVANT, T. L.; KUHL, J. G. A taxonomy of scheduling in general-purpose distributed computing systems. **IEEE Transactions on Software Engineering**, ., v. 14, n. 2, p. 141–154, Feb 1988.

CHEN, S.; XU, H.; LIU, D.; HU, B.; WANG, H. A Vision of IoT: applications, challenges, and opportunities with china perspective. **IEEE Internet of Things Journal**, ., v. 1, n. 4, p. 349–359, Aug 2014.

CHIU, D.; AGRAWAL, G. Evaluating caching and storage options on the Amazon Web Services Cloud. In: GRID COMPUTING (GRID), 2010 11TH IEEE/ACM INTERNATIONAL CONFERENCE ON, 2010. **Anais...** IEEE, 2010. p. 17–24.

CINQUE, M.; MARTINO, C. D.; ESPOSITO, C. On data dissemination for large-scale complex critical infrastructures. **Computer Networks**, ., v. 56, n. 4, p. 1215 – 1235, 2012.

COUTINHO, E. F.; CARVALHO SOUSA, F. R. de; REGO, P. A. L.; GOMES, D. G.; SOUZA, J. N. de. Elasticity in cloud computing: a survey. **Annals of Telecommunications - Annales des Télécommunications**, Paris, França, p. 1–21, 2014.

DELAMER, I. M.; LASTRA, J. L. M. Service-Oriented Architecture for Distributed Publish/Subscribe Middleware in Electronics Production. **IEEE Transactions on Industrial Informatics**, ., v. 2, n. 4, p. 281–294, Nov 2006.

DUTTA, S.; GERA, S.; VERMA, A.; VISWANATHAN, B. SmartScale: automatic application scaling in enterprise clouds. In: CLOUD COMPUTING (CLOUD), 2012 IEEE 5TH INTERNATIONAL CONFERENCE ON, 2012. **Anais...** IEEE, 2012. p. 221–228.

ESPOSITO, C.; CIAMPI, M. On Security in Publish/Subscribe Services: a survey. **IEEE Communications Surveys Tutorials**, ., v. 17, n. 2, p. 966–997, Secondquarter 2015.

ESPOSITO, C.; CIAMPI, M.; PIETRO, G. D. An event-based notification approach for the delivery of patient medical information. **Information Systems**, ., v. 39, p. 22 – 44, 2014.

ESPOSITO, C.; COTRONEO, D.; RUSSO, S. On reliability in publish/subscribe services. **Computer Networks**, ., v. 57, n. 5, p. 1318 – 1343, 2013.

EUGSTER, P. T.; FELBER, P. A.; GUERRAOUI, R.; KERMARREC, A.-M. The Many Faces of Publish/Subscribe. **ACM Comput. Surv.**, New York, NY, USA, v. 35, n. 2, p. 114–131, June 2003.

FARRAG, A. A. S.; MAHMOUD, S. A.; EL-HORBATY, E. S. M. Intelligent cloud algorithms for load balancing problems: a survey. In: IEEE SEVENTH INTERNATIONAL CONFERENCE ON INTELLIGENT COMPUTING AND INFORMATION SYSTEMS (ICICIS), 2015., 2015, . **Anais...** IEEE, 2015. p. 210–216.

FERGUSON, G. T. Have your objects call my objects. **Harvard business review**, ., v. 80, n. 6, p. 138–144, 2002.

FLEISCH, E. What is the internet of things? An economic perspective. **Economics, Management and Financial Markets**, ., v. 5, n. 2, p. 125, 2010.

FLEISCH, E.; SARMA, S.; THIESSE, F. Preface to the focus theme section: 'internet of things'. **Electronic Markets**, ., v. 19, n. 2, p. 99–102, 2009.

GALANTE, G.; BONA, L. C. E. de. A survey on cloud computing elasticity. In: UTILITY AND CLOUD COMPUTING (UCC), 2012 IEEE FIFTH INTERNATIONAL CONFERENCE ON, 2012, . **Anais...** IEEE, 2012. p. 263–270.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design patterns**: elements of. .: Addison-Wesley, 1994.

PREHN, S.; TOETENEL, W. J. (Ed.). **Formalizing design spaces**: implicit invocation mechanisms. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991. 31–44 p.

GERSHENFELD, N. **When Things Start to Think**. New York, NY, USA: Henry Holt and Co., Inc., 1999.

GUBBI, J.; BUYYA, R.; MARUSIC, S.; PALANISWAMI, M. Internet of Things (IoT): a vision, architectural elements, and future directions. **Future Generation Computer Systems**, ., v. 29, n. 7, p. 1645 – 1660, 2013. Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services; Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond.

IMAI, S.; CHESTNA, T.; VARELA, C. A. Elastic Scalable Cloud Computing Using Application-Level Migration. In: IEEE/ACM FIFTH INTERNATIONAL CONFERENCE ON UTILITY AND CLOUD COMPUTING, 2012., 2012, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2012. p. 91–98. (UCC '12).

JAMSHIDI, P.; AHMAD, A.; PAHL, C. Autonomic Resource Provisioning for Cloud-based Software. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE ENGINEERING FOR ADAPTIVE AND SELF-MANAGING SYSTEMS, 9., 2014, New York, NY, USA. **Proceedings...** ACM, 2014. p. 95–104. (SEAMS 2014).

KHAN, R.; KHAN, S. U.; ZAHEER, R.; KHAN, S. Future Internet: the internet of things architecture, possible applications and key challenges. In: FRONTIERS OF INFORMATION TECHNOLOGY (FIT), 2012 10TH INTERNATIONAL CONFERENCE ON, 2012, . **Anais...** ., 2012. p. 257–260.

KORESHOFF, T. L.; ROBERTSON, T.; LEONG, T. W. Internet of Things: a review of literature and products. In: AUSTRALIAN COMPUTER-HUMAN INTERACTION CONFERENCE: AUGMENTATION, APPLICATION, INNOVATION, COLLABORATION, 25., 2013, New York, NY, USA. **Proceedings...** ACM, 2013. p. 335–344. (OzCHI '13).

KOUKI, Y.; OLIVEIRA, F. A. d.; DUPONT, S.; LEDOUX, T. A Language Support for Cloud Elasticity Management. In: CLUSTER, CLOUD AND GRID COMPUTING (CCGRID), 2014 14TH IEEE/ACM INTERNATIONAL SYMPOSIUM ON, 2014. **Anais...** IEEE, 2014. p. 206–215.

LEHRIG, S.; EIKERLING, H.; BECKER, S. Scalability, Elasticity, and Efficiency in Cloud Computing: a systematic literature review of definitions and metrics. In: INTERNATIONAL ACM SIGSOFT CONFERENCE ON QUALITY OF SOFTWARE ARCHITECTURES, 11., 2015, . **Proceedings...** ACM, 2015. p. 83–92.

LI, M.; YE, F.; KIM, M.; CHEN, H.; LEI, H. A Scalable and Elastic Publish/Subscribe Service. In: PARALLEL DISTRIBUTED PROCESSING SYMPOSIUM (IPDPS), 2011 IEEE INTERNATIONAL, 2011, . **Anais...** IEEE, 2011. p. 1254–1265.

LIU, F.; TONG, J.; MAO, J.; BOHN, R.; MESSINA, J.; BADGER, L.; LEAF, D. NIST cloud computing reference architecture. **NIST special publication**, ., v. 500, n. 2011, p. 292, 2011.

LORIDO-BOTRÁN, T.; MIGUEL-ALONSO, J.; LOZANO, J. A. Auto-scaling techniques for elastic applications in cloud environments. **Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09**, ., v. 12, p. 2012, 2012.

LORIDO-BOTRAN, T.; MIGUEL-ALONSO, J.; LOZANO, J. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments. **Journal of Grid Computing**, Houten, Netherlands, v. 12, n. 4, p. 559–592, 2014.

MANDAL, J. K.; MUKHOPADHYAY, A.; DASGUPTA, K.; MANDAL, B.; DUTTA, P.; MANDAL, J. K.; DAM, S. First International Conference on Computational Intelligence: modeling techniques and applications (cimta) 2013 a genetic algorithm (ga) based load balancing strategy for cloud computing. **Procedia Technology**, ., v. 10, p. 340 – 347, 2013.

MARTINS, J. L.; DUARTE, S. Routing algorithms for content-based publish/subscribe systems. **IEEE Communications Surveys Tutorials**, ., v. 12, n. 1, p. 39–58, First 2010.

MATTERN, F.; FLOERKEMEIER, C. **From Active Data Management to Event-Based Systems and More**: papers in honor of alejandro buchmann on the occasion of his 60th birthday. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. 242–259 p.

MELL, P. M.; GRANCE, T. **SP 800-145. The NIST Definition of Cloud Computing**. Gaithersburg, MD, United States: Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011.

NAVAUX, P. O. A.; RIGHI, R. d. R. **A New Approach for Processes Rescheduling Management on Bulk Synchronous Parallel Applications**. 2009. Tese (Doutorado em Ciência da Computação) — UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, 2009.

NUAIMI, K. A.; MOHAMED, N.; NUAIMI, M. A.; AL-JAROODI, J. A Survey of Load Balancing in Cloud Computing: challenges and algorithms. In: NETWORK CLOUD COMPUTING AND APPLICATIONS (NCCA), 2012 SECOND SYMPOSIUM ON, 2012, . **Anais...** ., 2012. p. 137–142.

PATEL, M. P. V.; PATEL, M. H. D.; PATEL, A. P. P. J. A Survey on Load Balancing in Cloud Computing. In: INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH AND TECHNOLOGY, 2012, . **Anais...** ESRSA Publications, 2012. v. 1, n. 9 (November-2012).

PUTHAL, D.; SAHOO, B.; MISHRA, S.; SWAIN, S. Cloud Computing Features, Issues, and Challenges: a big picture. In: COMPUTATIONAL INTELLIGENCE AND NETWORKS (CINE), 2015 INTERNATIONAL CONFERENCE ON, 2015. **Anais...** IEEE, 2015. p. 116–123.

RAVEENDRAN, A.; BICER, T.; AGRAWAL, G. A framework for elastic execution of existing mpi programs. In: PARALLEL AND DISTRIBUTED PROCESSING

WORKSHOPS AND PHD FORUM (IPDPSW), 2011 IEEE INTERNATIONAL SYMPOSIUM ON, 2011, . **Anais...** IEEE, 2011. p. 940–947.

RIEHLE, D. The Event Notification Pattern - Integrating Implicit Invocation with Object-Orientation. **TAPOS**, ., v. 2, n. 1, p. 43–52, 1996.

RIGHI, R.; RODRIGUES, V.; ANDRE DACOSTA, C.; GALANTE, G.; BONA, L.; FERRETO, T. AutoElastic: automatic resource elasticity for high performance applications in the cloud. **Cloud Computing, IEEE Transactions on**, ., v. PP, n. 99, p. 1–1, 2015.

RIMAL, B. P.; CHOI, E.; LUMB, I. A Taxonomy and Survey of Cloud Computing Systems. In: INC, IMS AND IDC, 2009. NCM '09. FIFTH INTERNATIONAL JOINT CONFERENCE ON, 2009, . **Anais...** ., 2009. p. 44–51.

RODRIGUES, V. F. **AutoElastic**: explorando a elasticidade de recursos em nuvem para aplicações de alto desempenho iterativas. 2015. Tese (Doutorado em Ciência da Computação) — Universidade do Vale do Rio dos Sinos, São Leopoldo, Rio Grande do Sul, Brasil, 2015.

ROSA RIGHI, R.; COSTA, C. A.; RODRIGUES, V. F.; ROSTIROLLA, G. Joint-analysis of performance and energy consumption when enabling cloud elasticity for synchronous HPC applications. **Concurrency and Computation: Practice and Experience**, ., 2015.

SAH, S. K.; JOSHI, S. R. Scalability of efficient and dynamic workload distribution in autonomic cloud computing. In: ISSUES AND CHALLENGES IN INTELLIGENT COMPUTING TECHNIQUES (ICICT), 2014 INTERNATIONAL CONFERENCE ON, 2014, . **Anais...** IEEE, 2014. p. 12–18.

SATYANARAYANAN, M. Pervasive computing: vision and challenges. **Personal Communications, IEEE**, ., v. 8, n. 4, p. 10–17, Aug 2001.

SCHLOSSNAGLE, T. **Scalable internet architectures**. .: Pearson Education, 2006.

SCHMIDT, D. C.; STAL, M.; ROHNERT, H.; BUSCHMANN, F. **Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects**. .: John Wiley & Sons, 2013. v. 2.

SHARMA, T.; BANGA, V. K. Efficient and Enhanced Algorithm in Cloud Computing. **International Journal of Soft Computing and Engineering (IJSCE) ISSN**, ., p. 2231–2307, 2013.

SHARMA, U.; SHENOY, P.; SAHU, S.; SHAIKH, A. A Cost-Aware Elasticity Provisioning System for the Cloud. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, 2011., 2011, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2011. p. 559–570. (ICDCS '11).

SHOJA, H.; NAHID, H.; AZIZI, R. A comparative survey on load balancing algorithms in cloud computing. In: COMPUTING, COMMUNICATION AND NETWORKING TECHNOLOGIES (ICCCNT), 2014 INTERNATIONAL CONFERENCE ON, 2014, . **Anais...** ., 2014. p. 1–5.

TAN, L.; WANG, N. Future internet: the internet of things. In: ADVANCED COMPUTER THEORY AND ENGINEERING (ICACTE), 2010 3RD INTERNATIONAL CONFERENCE ON, 2010, . **Anais...** ., 2010. v. 5, p. V5–376–V5–380.

TANENBAUM, A. **Computer Networks**. 4th. ed. Upper Saddle River, New Jersey: Prentice Hall PTR, 2003. 912 p.

TRAN, N.-L.; SKHIRI, S.; ZIMÁNYI, E. EQS: an elastic and scalable message queue for the cloud. In: CLOUD COMPUTING TECHNOLOGY AND SCIENCE (CLOUDCOM), 2011 IEEE THIRD INTERNATIONAL CONFERENCE ON, 2011, . **Anais...** IEEE, 2011. p. 391–398.

UNION, I. T. **ITU Internet Reports 2005**: the internet of things. [Online; accessed 12-January-2017].

WANG, Y.; MA, X. A General Scalable and Elastic Content-Based Publish/Subscribe Service. **IEEE Transactions on Parallel and Distributed Systems**, ., v. 26, n. 8, p. 2100–2113, Aug 2015.

WARAICH, S. S. Classification of Dynamic Load Balancing Strategies in a Network of Workstations. In: INFORMATION TECHNOLOGY: NEW GENERATIONS, 2008. ITNG 2008. FIFTH INTERNATIONAL CONFERENCE ON, 2008, . **Anais...** ., 2008. p. 1263–1265.

WARD, J. S.; BARKER, A. Self managing monitoring for highly elastic large scale cloud deployments. In: DATA INTENSIVE DISTRIBUTED COMPUTING, 2014, . **Proceedings...** ACM, 2014. p. 3–10.

WEISER, M. The Computer for the 21st Century. **SIGMOBILE Mob. Comput. Commun. Rev.**, New York, NY, USA, v. 3, n. 3, p. 3–11, July 1999.

XU, Y.; MAHENDRAN, V.; RADHAKRISHNAN, S. Towards SDN-based fog computing: mqtt broker virtualization for effective and reliable delivery. In: INTERNATIONAL CONFERENCE ON COMMUNICATION SYSTEMS AND NETWORKS (COMSNETS), 2016., 2016, . **Anais...** ., 2016. p. 1–6.