PGCOMP - Programa de Pós-Graduação em Ciência da Computação
Universidade Federal da Bahia (UFBA)
Av. Milton Santos, s/n - Ondina
Salvador, BA, Brasil, 40170-110

https://pgcomp.ufba.br
pgcomp@ufba.br

Ao longo dos anos, a cloud computing vem fornecendo infraestrutura para o processamento de aplicações e serviços. Entretanto, a distância entre os recursos computacionais da cloud e os dispositivos da Internet of Things (IoT) pode causar atrasos nos tempos de respostas para alguns serviços e aplicações que possuem sensibilidade a requisitos temporais. Nesse contexto, a fog computing surge como um paradigma computacional que busca aproximar os recursos da cloud computing dos dispositivos IoT, distribuindo geograficamente as Cloudlets e reduzindo o tempo de resposta. Estudos utilizando a fog computing apresentam soluções para a alocação das aplicações de forma que seja possível atender aos diferentes tipos de requisitos. Esses estudos avaliam determinados requisitos para tomada de decisão, tais como o tempo de resposta e o custo de comunicação. Entretanto, observa-se que à medida que certos requisito de uma aplicação são priorizados, outras aplicações podem ser afetadas. Dessa forma, este trabalho propõe uma abordagem para alocação de aplicações modulares em uma arquitetura de fog computing de múltiplos níveis hierárquicos, que busca reduzir o tempo de resposta de aplicações sensíveis a latência e reduzir o tráfego de dados na rede. Para alcançar esse objetivo, a abordagem proposta, Least Impact - X (LI-X), busca reduzir a ociosidade de recursos nos níveis mais baixos da fog levando em consideração a comunicação entre os módulos das aplicações quando precisa decidir sobre a alocação. O LI-X foi comparado a estudos predecessores em um ambiente simulado do iFogSim. Os resultados mostram que o LI-X conseguiu superar esses estudos na maioria dos cenários propostos, reduzindo o tempo de resposta e o custo com a comunicação de dados na rede.

Palavras-chave: cloud computing; fog computing; fog hierárquica; alocação; aplicações modulares.

MSC | 150 | 2022

Uma abordagem sensível à latência e ao custo de comunicação para alocação de Aplicações Modulares em uma Névoa Hierárquica

Leonan Teixeira de Oliveira

UFBA

Uma abordagem sensível à latência e ao custo de comunicação para alocação de Aplicações Modulares em uma Névoa Hierárquica

Leonan Teixeira de Oliveira

Dissertação de Mestrado

Universidade Federal da Bahia

Programa de Pós-Graduação em Ciência da Computação

Dezembro | 2022

Universidade Federal da Bahia
Instituto de Computação

Programa de Pós-Graduação em Ciência da Computação

UMA ABORDAGEM SENSÍVEL À LATÊNCIA E AO CUSTO DE COMUNICAÇÃO PARA ALOCAÇÃO DE APLICAÇÕES MODULARES EM UMA NÉVOA HIERÁRQUICA

Leonan Teixeira de Oliveira

DISSERTAÇÃO DE MESTRADO

Salvador
13 de Dezembro de 2022

LEONAN TEIXEIRA DE OLIVEIRA

UMA ABORDAGEM SENSÍVEL À LATÊNCIA E AO CUSTO DE COMUNICAÇÃO PARA ALOCAÇÃO DE APLICAÇÕES MODULARES EM UMA NÉVOA HIERÁRQUICA

Esta Dissertação de Mestrado foi apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Maycon Leone Maciel Peixoto

Salvador
13 de Dezembro de 2022

Ficha catalográfica elaborada pela Biblioteca Universitária de Ciências e Tecnologias Prof. Omar Catunda, SIBI – UFBA.

O48 Oliveira, Leonan Teixeira de

Uma abordagem sensível à latência e ao custo de comunicação para alocação de Aplicações Modulares em uma Névoa Hierárquica / Leonan Teixeira de Oliveira. – Salvador, 2022.

68 f.

Orientadora: Prof. Dr. Maycon Leone Maciel Peixoto

Dissertação (Mestrado) – Universidade Federal da Bahia. Instituto de Computação, 2022.

1. Computação. 2. Tecnologia da Informação. 3. Alocação. Peixoto, Maycon Leone Maciel. II. Universidade Federal da Bahia. III. Título.

CDU 004

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DA BAHIA
INSTITUTO DE COMPUTAÇÃO
PGCOMP - Programa de Pós-Graduação em Ciência da Computação
http://pgcomp.ufba.br

"*Uma abordagem sensível à latência e ao custo de comunicação para alocação de aplicações modulares em uma fog hierárquica*"

Leonan Teixeira de Oliveira

Dissertação apresentada ao Colegiado do Programa de Pós-Graduação em Ciência da Computação na Universidade Federal da Bahia, como requisito parcial para obtenção do Título de Mestre em Ciência da Computação.

Banca Examinadora

Prof. Dr. Maycon Leone Maciel Peixoto (Orientador PGCOMP)

Prof. Dr. Daniel Gouveia Costa (Universidade do Porto - Portugal)

Prof. Dr. Luiz Fernando Bittencourt (UNICAMP)

AGRADECIMENTOS

Todo pacote de dados que é enviado através da rede, precisa chegar a algum destino, seja lá onde esse destino for. O pacote que foi enviado no início dessa jornada, apesar dos saltos estranhos em meio a um emaranhado de *Backbones* de decisões, enfim encontra seu destino. Enquanto escrevo esses agradecimentos, ainda será necessário que esse pacote seja avaliado por rigorosos algoritmos de avaliação e detecção de erros. Contudo, ele chegou, e por isso preciso agradecer algumas pessoas que fizeram parte ou de alguma forma contribuíram para essa conquista.

Agradeço inicialmente ao meu pai Aldemar e minha mãe Marinália, que com atenção, conselhos e amor me guiaram a me tornar quem hoje sou. Agradeço ao meu irmão Alan, que também foi meu colega de mestrado e que juntamente com nosso colega Thiago, compartilhamos bons momentos no começo dessa trajetória.

Agradeço a namorada Simone, pela preocupação, paciência, afeto e amor, mesmo em momentos em que a minha atenção lhe foi negada, ou nos momentos em que deixamos de fazer uma viagem, um passeio ou mesmo tomar um sorvete em decorrências das minhas preocupações e anseios para a conclusão deste trabalho.

Agradeço aos amigos que conquistei nessa jornada, em especial ao Igo Romero, Ícaro Ramires e o Matheus Thiago, em muitos momentos, estar com vocês foi o que me manteve seguindo em frente. Agradeço também a todos do *Urban Computing Lab (UCL)*.

Agradeço ao meu orientador, Prof. Maycon Peixoto, não somente pelas orientações quanto ao desenvolvimento desse trabalho, mas também pelos conselhos, pela paciência e por ter acreditado que eu conseguiria concluir este trabalho.

E por fim, a todos os que de alguma forma contribuíram para que esse pacote de dados chamado mestrado chegasse até aqui, deixo os meus agradecimentos.

Se você quiser fazer uma torta de maçã do nada, você deve primeiro inventar o Universo

—CARL SAGAN

RESUMO

Ao longo dos anos, a *cloud computing* vem fornecendo infraestrutura para o processamento de aplicações e serviços. Entretanto, a distância entre os recursos computacionais da *cloud* e os dispositivos da Internet of Things (IoT) pode causar atrasos nos tempos de respostas para alguns serviços e aplicações que possuem sensibilidade a requisitos temporais. Nesse contexto, a *fog computing* surge como um paradigma computacional que busca aproximar os recursos da *cloud computing* dos dispositivos IoT, distribuindo geograficamente as *Cloudlets* e reduzindo o tempo de resposta. Estudos utilizando a *fog computing* apresentam soluções para a alocação das aplicações de forma que seja possível atender aos diferentes tipos de requisitos. Esses estudos avaliam determinados requisitos para tomada de decisão, tais como o tempo de resposta e o custo de comunicação. Entretanto, observa-se que à medida que certos requisito de uma aplicação são priorizados, outras aplicações podem ser afetadas. Dessa forma, este trabalho propõe uma abordagem para alocação de aplicações modulares em uma arquitetura de *fog computing* de múltiplos níveis hierárquicos, que busca reduzir o tempo de resposta de aplicações sensíveis a latência e reduzir o tráfego de dados na rede. Para alcançar esse objetivo, a abordagem proposta, *Least Impact - X (LI-X)*, busca reduzir a ociosidade de recursos nos níveis mais baixos da *fog* levando em consideração a comunicação entre os módulos das aplicações quando precisa decidir sobre a alocação. O *LI-X* foi comparado a estudos predecessores em um ambiente simulado do iFogSim. Os resultados mostram que o *LI-X* conseguiu superar esses estudos na maioria dos cenários propostos, reduzindo o tempo de resposta e o custo com a comunicação de dados na rede.

Palavras-chave: *cloud computing; fog computing; fog hierárquica; alocação; aplicações modulares.*

ABSTRACT

Over the years, cloud computing has provided infrastructure for processing applications and services. However, the distance between Cloud computing resources and Internet of Things (IoT) devices can cause delays in response times for some services and applications that have time-sensitive constraints. In this context, fog computing emerges as a computational paradigm that seeks to bring cloud computing resources closer to IoT devices, geographically distributing micro-data centers (Cloudlets) and reducing response time. Studies using fog computing present solutions for the application allocation, addressing that it is possible to meet different types of requirements. These studies evaluate certain requirements for decision making, such as response time and communication cost. However, it is observed that as a requirement is prioritized, another is neglected. Thus, this work proposes an approach for allocating modular applications in a fog computing architecture with multiple hierarchical levels, which seeks to reduce the response time of latency-sensitive applications and reduce data traffic on the network. To achieve this goal, the proposed approach, Least Impact - X (LI-X), seeks to reduce the idleness of resources at the lowest levels of the fog and takes into account the communication between application modules when deciding on the allocation. LI-X was compared to predecessor studies in a simulated iFogSim environment. The results show that the LI-X was able to overcome these studies in most of the proposed scenarios, reducing the response time and the cost of data communication on the network.

Keywords: cloud computing; fog computing; hierarchical fog; allocation; component-based applications

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

Capítulo 1

INTRODUÇÃO

Os últimos anos foram marcados por um forte crescimento do campo da *Internet of Things* (IoT), caracterizada pelo surgimento de aplicações de diversos tipos que procuram conectar coisas à Internet. A busca por uma computação ubíqua e pervasiva ainda está em estágio inicial, centenas de pesquisas têm como objetivo aprimorar as diversas áreas envolvidas por trás desse paradigma. Os chamados dispositivos IoT podem servir aos mais diferentes propósitos, equipados com sensores e atuadores, podem coletar informações e interagirem com o mundo real. Para isso, tais aplicações precisam muitas vezes processar grandes volumes de dados de forma a atender a diferentes requisitos (BONOMI et al., 2012; CHIANG; ZHANG, 2016; NGUYEN et al., 2021).

Os dispositivos IoT são elementos chave para o surgimento das chamadas Cidades Inteligentes, onde podem atuar, entre outras áreas, na melhoria da segurança, na melhora do fluxo de carros no trânsito, no transporte público, no gerenciamento inteligente de energia e de água potável. Dentro das casas, é possível conectar eletrodomésticos e eletrônicos, tornando possível uma maior automação residencial, desde o controle de iluminação, gerenciamento da segurança, temperatura ambiente e até o momento correto de regar as plantas. No campo da saúde, é possível vestir-se de sensores conectados capazes de monitorar o paciente durante a sua rotina diária e tomar decisões baseadas na análise dos dados coletados, como injetar insulina em um paciente diabético no momento adequado ou acionar uma ambulância em um momento de emergência.

Esses dispositivos muitas vezes não dispõem dos recursos necessários para processar ou mesmo não detêm todas as informações necessárias para tomar decisões. A IoT, por muitas vezes, faz uso da *cloud computing* e do seu poder computacional para armazenar e processar as informações coletadas que são produzidas por tais dispositivos (BORGIA, 2014; AL-FUQAHA et al., 2015). A *cloud computing* atua nesse conjunto, como, o ponto central de armazenamento e processamento dos dados. A *cloud computing* com seu poder computacional, e a sua característica de elasticidade, que permite o redimensionamento dos recursos de acordo a demanda, é capaz de processar os dados e executar os mais sofisticados algoritmos. Contudo, existe uma distância que os dados precisam percorrer entre os dispositivos IoT até um *datacenter* da *cloud computing* e algumas vezes é preciso

que algum dado retorne para o dispositivo de origem, ou mesmo que outro dispositivo também conectado na rede seja de alguma forma acionado. Percorrer essa distância custa algum tempo que não é tolerado por algumas aplicações que exigem respostas mais rápidas e que atrasos, mesmo que pequenos, não são aceitáveis (VAQUERO et al., 2008; GANGADHARAN, 2017; YOUSEFPOUR et al., 2019).

Neste sentido, surge a *fog computing* como um paradigma que busca aproximar os recursos computacionais da *cloud computing* para perto dos dispositivos finais, reduzindo desta forma o tempo necessário para que o dado trafegue até seu destino e em alguns casos retorne a sua origem, assim como também reduzindo o tráfego total na rede, visto que o dado precisa percorrer um caminho menor. Contudo, para fornecer essa capacidade computacional de maneira ampla aos dispositivos e usuários, os recursos da *fog computing* precisam estar geograficamente distribuídos, buscando dessa forma estar sempre próximos de onde o usuário ou dispositivo estará. Além disso, a *fog computing* permite uma organização dos seus recursos em termos de hierarquia, no qual é possível estabelecer níveis interconectados por onde os recursos estarão distribuídos. Essa característica, ao contrário da *cloud computing* que consegue colocar milhares de recursos computacionais em um só espaço físico, faz com que o poder computacional da *fog computing* fique distribuído em diferentes espaços físicos, tornando a capacidade de cada um desses pequenos centros limitada. (YOUSEFPOUR et al., 2019; CHIANG; ZHANG, 2016; BONOMI et al., 2012)

Os dispositivos IoT e suas aplicações podem ter diferentes requisitos. Algumas podem fazer uso mais intenso da rede, outras podem exigir tempos de respostas mais rápidas. Além disso, as aplicações podem ser construídas de forma modular de tal forma que cada um dos seus módulos possa ser executado de forma distribuída em diferentes dispositivos em uma rede. Com base nisso, provedores de serviços de *fog computing*, devido à capacidade limitada dos recursos e sua característica distribuída, precisam controlar e organizar a execução das aplicações e seus módulos, de forma a buscar um melhor aproveitamento dos recursos e atender aos requisitos individuais de cada aplicação, para isso, fazem uso dos algoritmos de alocação (PEIXOTO; GENEZ; BITTENCOURT, 2022; CHARÂNTOLA et al., 2019).

Visto que o crescente avanço da adoção da IoT em diversos segmentos provocará cada vez mais o aumento da demanda por recursos computacionais, e por ser um campo ainda emergente, diversas pesquisas nos últimos anos têm buscado aprimorar o paradigma de *fog computing* para que consiga suportar e atender da melhor forma a todos esses dispositivos. A questão principal que esse trabalho busca responder é: É possível aprimorar o uso eficiente dos recursos computacionais em uma arquitetura de *fog computing*, buscando reduzir o tempo de resposta de aplicações sensíveis ao atraso e ainda reduzir o tráfego total de dados na rede?

A partir dessa pergunta, este trabalho apresenta como hipótese a ideia de que: Reduzir a ociosidade de recursos nos níveis mais próximos dos dispositivos IoT em uma arquitetura de *fog computing*, considerando os requisitos de latência das aplicações e o trafego de dados entre os seus módulos, pode reduzir o tempo de resposta de aplicações sensíveis e o tráfego de dados na rede.

1.1 OBJETIVOS

Com base no contexto apresentado, o objetivo deste trabalho é construir uma abordagem para o problema de alocação de aplicações modulares em uma arquitetura de *fog computing* hierárquica. Espera-se que a abordagem seja capaz de organizar os módulos das aplicações entre os recursos computacionais da *fog* buscando reduzir o tempo de resposta das aplicações mais sensíveis ao atraso e, de forma secundária, reduzir o tráfego total de dados na rede.

Como objetivos específicos para alcançar o resultado esperado, foram definidos:

- Configurar um ambiente de simulação que permita a simulação de diferentes arquiteturas de *fog computing*;
- Projetar e implementar um algoritmo para alocação de aplicações modulares em uma arquitetura de *fog computing*;
- Comparar e avaliar o algoritmo proposto com outros algoritmos propostos na literatura por meio de experimentos em ambientes simulados;
- Apresentar os resultados obtidos com a pesquisa e divulgá-los no meio acadêmico.

1.2 PRINCIPAIS CONTRIBUIÇÕES

As contribuições deste trabalho são enumeradas a seguir:

- Desenvolvimento de um algoritmo para alocação de aplicações modulares da *fog computing*;
- Modificações no simulador iFogSim para a construção de topologias de *fog computing* de forma declarativa;
- Avaliação de diferentes algoritmos em diferentes topologias de *fog computing* utilizando o iFogSim.

Durante o decorrer da pesquisa, foi publicado o seguinte artigo:

- OLIVEIRA, L. et al. Arquitetura baseada em computação em névoa para sistemas de gerenciamento inteligente de Água. In: Anais do II Workshop de Computação Urbana. Porto Alegre, RS, Brasil: SBC, 2018. ISSN 2595-2706. Disponível em: https://sol.sbc.org.br/index.php/courb/article/view/2350

Ainda como resultado dessa pesquisa, submeteremos o artigo "LI-X: An approach to allocate component-based applications in a hierarchical fog computing architecture"para publicação na revista *IEEE Transactions on Services Computing*.

1.3 ORGANIZAÇÃO DA DISSERTAÇÃO

O Capítulo 2 apresenta o referencial teórico sobre *Internet of Things* (IoT), *cloud computing* e *fog computing*, destacando os principais conceitos necessários para a compreensão deste trabalho. No Capítulo 3 são apresentados alguns trabalhos que de alguma forma se relacionam com este. No Capítulo 4 é apresentada a proposta deste trabalho bem como os seus detalhes e algoritmos. O Capítulo 5 aborda a construção e a configuração do ambiente de experimentação, levando em consideração as aplicações, topologias e o simulador. Já no Capítulo 6 são expostos os principais resultados dos experimentos bem como a discussão e avaliação do que foi encontrado. Por fim, no Capítulo 7 apresentam-se as considerações finais e são apresentados alguns caminhos para os trabalhos futuros.

Capítulo 2

REFERENCIAL TEÓRICO

Neste capítulo são apresentados conceitos comumente citados ao longo do trabalho. Na subseção 2.1 é apresentada a definição de *Internet of Things* (IoT). Na 2.2 é apresentada a definição de *cloud computing*. A subseção 2.3 apresenta o conceito de *fog computing*. Na subseção 2.4 são apresentadas algumas questões relevantes para aplicações na *fog*. E por fim, na subseção 2.5 é apresentando o *iFogSim*, simulador baseado em eventos construído para simular aplicações e arquitetura de *fog computing*.

2.1 INTERNET DAS COISAS

Internet of Things (IoT) pode ser definida como paradigma que consiste na combinação de diversos aspectos e tecnologias, objetivando a construção de um sistema de interação entre o mundo real e o virtual, a ideia geral da IoT é que tudo pode ser conectado à internet. A partir dos conceitos de computação ubíqua e pervasiva, protocolos de Internet, tecnologias de sensoriamento, comunicação e sistemas embarcados, "coisas" devem ser capazes de interagir de forma autônoma e inteligente com o ambiente e com outras "coisas" por meio de atuadores e sensores através da rede de Internet (BORGIA, 2014; LI; XU; ZHAO, 2015).

Há uma variedade de novas aplicações inteligentes que podem ser desenvolvidas utilizando IoT, praticamente em todos os campos de conhecimento existe alguma forma de aplicá-la. Utensílios, redes de energia, equipamentos pessoais, sistemas agrícolas, sistemas de transporte, sistemas de saúde, sistemas industriais e até os nossos corpos podem ser utilizados como ambientes para aplicações IoT. Um dos objetivos dessas aplicações é melhorar e facilitar a vida das pessoas por meio da computação (AL-FUQAHA et al., 2015; BORGIA, 2014; GUBBI et al., 2013). Para Borgia (2014), às aplicações IoT podem ser classificadas em três domínios principais: Industrial, Cidades Inteligentes e Bem-Estar e Saúde. Cada domínio não está isolado, pois pode existir uma sobreposição entre eles dependendo da abrangência de atuação de cada aplicação. Para atender aos diversos requisitos de aplicações IoT, diferentes propostas de arquiteturas vêm sendo apresentadas nos últimos anos.

2.1.1 Arquitetura da IoT

Prover flexibilidade, escalabilidade, segurança, confiabilidade, Qualidade de Serviço (QoS) e a interoperabilidade entre os diversos tipos de dispositivo são requisitos críticos para a arquitetura de um sistema IoT. Em vista disso, diversas propostas de arquiteturas vêm sendo sugeridas ao longo dos últimos anos.

Wu et al. (2010) apresetam um modelo básico de uma arquitetura IoT dividida em três camadas, conforme representada na Figura 2.1. Na camada de percepção estão presentes os sensores e atuadores que trabalham na coleta de informações e na interação com o ambiente. A camada de rede é o cérebro da IoT, sendo responsável pela transmissão e pelo processamento de dados obtidos na camada de percepção. Por fim, na camada de aplicação há uma convergência das tecnologias envolvidas na IoT combinada com as necessidades da indústria. Ela é responsável por fornecer serviços específicos de aplicativos ao usuário. Nela são definidas as várias aplicações nas quais a Internet das Coisas pode ser implantada, por exemplo, casas inteligentes, cidades inteligentes e saúde inteligente (WU et al., 2010; AL-FUQAHA et al., 2015).

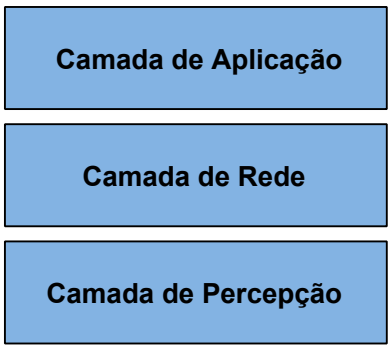


Figura 2.1 Arquitetura IoT em três camadas (WU et al., 2010).

As novas propostas de arquitetura estão surgindo com objetivo de abstrair cada vez mais os componentes do sistema IoT. A abordagem de Arquitetura Orientada a Serviços (SOA) tem sido a mais comumente utilizada. Isso, devido a SOA permitir a decomposição de sistemas complexos em pequenos serviços mais simples, bem definidos, com baixo acoplamento e independentes de plataforma, facilitando dessa forma a integração entre os diversos componentes, além de tornar mais simples a reutilização de software e hardware (LAN et al., 2015; AL-FUQAHA et al., 2015; WU et al., 2010; LI; XU; ZHAO, 2015; BORGIA, 2014).

A Figura 2.2 apresenta uma SOA para ambientes IoT comumente encontrada na literatura. As camadas de composição de serviço, gerenciamento de serviço e abstração de serviço, compõe o que é chamado de *middleware*. Essas camadas são as responsáveis por abstrair a interface de comunicação entre as diferentes tecnologias, eximindo dos desenvolvedores da aplicação a necessidade de terem todo o conhecimento técnico para

operar os diferentes tipos de dispositivos (BOTTA et al., 2016). Abaixo são apresentadas as características de cada uma das camadas.

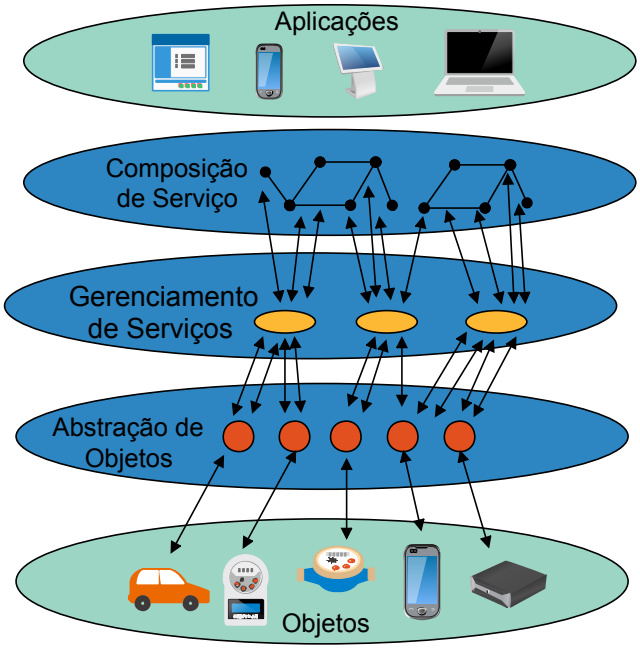


Figura 2.2 Arquitetura SOA para IoT em cinco camadas. Adaptado de Botta et al. (2016).

Objetos: Também chamada na literatura de camada de percepção ou de dispositivos. Essa camada é composta por objetos (coisas) equipadas com sensores e ou atuadores. Diversos aspectos devem ser levados em consideração no desenvolvimento dessa camada. Devido ao grande número de sensores necessários em aplicações complexas, os dispositivos inteligentes devem ser projetados visando minimizar os custos e os recursos necessários para seu funcionamento, outras características como a facilidade de desenvolvimento, implantação e comunicação também devem ser levadas em conta. As principais funções desta camada são: sensoriamento, atuação, identificação, interação e comunicação (BOTTA et al., 2016; LI; XU; ZHAO, 2015).

Abstração de Objetos: Devido à variedade de dispositivos conectados ao sistema, a diversidade de protocolos e aos tipos e estruturas de dados associados, o papel dessa camada é de abstrair a complexidade, tornando o acesso aos dispositivos uma tarefa simples para as próximas camadas. Cada objeto ou funcionalidade é abstraída em formato de serviço, fornecendo semântica e métodos para acesso aos objetos (AL-FUQAHA et al., 2015; BORGIA, 2014).

Gerenciamento de Serviço: Esta camada oferece serviços de descoberta automática, monitoramento e configurações dos dispositivos, além de fornecer serviços necessários pelos protocolos de rede (AL-FUQAHA et al., 2015; BONOMI et al., 2012).

Composição de Serviço: Esta camada fornece mecanismos para a composição de serviços, facilitando a integração entre os elementos conectados, gerenciando dinamicamente e em tempo real os novos serviços a partir de um único ou conjunto de serviços básicos (LI; XU; ZHAO, 2015; BORGIA, 2014).

Aplicação: Esta camada encontra-se no topo da arquitetura, nela as funcionalidades do sistema são expostas ao usuário final. A mesma apresenta capacidade de fornecer serviços inteligentes e de alta qualidade que atenda as necessidades do cliente e, através dela, o usuário pode acompanhar informações e interagir com todo o sistema (AL-FUQAHA et al., 2015).

2.1.2 Elementos da IoT

A Figura 2.3 mostra os seis elementos fundamentais para a entrega de serviços IoT definidos por Al-Fuqaha et al. (2015).

Figura 2.3 Elementos da IoT. Adaptado de Al-Fuqaha et al. (2015).

Os métodos de **identificação** são utilizados para fornecer identidades únicas para cada objeto dentro da rede. Endereçamentos IPv6 e IPv4 podem fornecer endereços únicos para o objeto, enquanto códigos, como o Código de Produtos Eletrônicos (EPC), pode fornecer um nome ao objeto. A utilização conjunta de nome e endereçamento na identificação dos dispositivos é importante, visto que os sistemas de nomenclatura podem não ser globalmente únicos. Dessa forma, mecanismos de endereçamento contribuem para a identificação única de cada objeto.

Sensores da IoT podem ser sensores inteligentes ou atuadores. Os mesmos são essenciais para o funcionamento da IoT, a partir deles os dados são coletados, processados e analisados para que sejam tomadas decisões baseadas em especificações do sistema. Normalmente, pequenos computadores e placas de circuitos integrados, como o Raspberry PI e o Arduino, equipados com sensores e componentes de comunicação são utilizados para as atividades de sensoriamento inteligente (AL-FUQAHA et al., 2015).

As **comunicações** em sistemas IoT contribuem para conectar diferentes tipos de objetos com o propósito de fornecer serviços inteligentes. Existem diversas tecnologias de comunicação utilizadas pelos dispositivos IoT, tais quais: Identificação por Radiofrequência (RFID), Comunicação por Campo de Proximidade (NFC), Bluetooth, IEEE 802.15.4 (ZigBee), WiFi, WiFiDirect e LTE (AL-FUQAHA et al., 2015).

O núcleo dos sistemas IoT é formado conjuntamente por hardware e softwares. Diversas plataformas de hardware são desenvolvidas para atender expressamente requisitos de sistemas IoT como o Arduino, Intel Galileo e Raspberry PI. Conjuntamente, plataformas de softwares também são produzidas com a finalidade de fornecer funcionalidades para sis-

temas IoT. Plataformas na *cloud* são outra forma de fornecer recursos **computacionais** ao sistemas IoT (AL-FUQAHA et al., 2015).

Os **serviços** em sistemas IoT são categorizados em quatro grupos: *Identity-Related*, *Information Aggregation Services*, *Collaborative-Aware* e *Ubiquitous Services*. O *Identity-Related* fornece o serviço básico de identificação dos objetos, enquanto os *Information Aggregation Services* atuam na coleta e sumarização dos dados. Os serviços de *Collaborative-Awere* utilizam os dados coletados pelo *Information Aggregation Services* para tomar decisões e executar ações e, por fim, os *Ubiquitous Services* têm o objetivo de fornecer acesso e controle completo a todo o ambiente através de um computador, telefone ou qualquer outra coisa (AL-FUQAHA et al., 2015; GIGLI; S., 2011).

A **semântica** é o elemento que representa a capacidade de extrair conhecimento de diferentes equipamentos do sistema IoT. Para isso, aplicam-se técnicas como reconhecimento de padrões e modelagem de informações para darem sentido aos dados obtidos, de modo que decisões possam ser tomadas (AL-FUQAHA et al., 2015).

2.2 CLOUD COMPUTING

Cloud computing é um paradigma que provê capacidades de computação de forma elástica e escalável em formato de serviço aos seus usuários (GANGADHARAN, 2017). A mesma é baseada em virtualização, SOA, computação utilitária e computação paralela. (NAMASUDRA; ROY; BALUSAMY, 2017). De acordo com Mell e Grance (2011), *cloud computing* é um modelo para habilitar o acesso por rede de forma ubíqua, conveniente e sob demanda a um conjunto compartilhado de recursos de computação (como redes, servidores, armazenamento, aplicações e serviços) que possam ser rapidamente provisionados e liberados com o mínimo de esforço de gerenciamento ou interação com o provedor de serviços. Esses recursos podem ser reconfigurados dinamicamente para ajustarem-se a uma carga de trabalho variável, permitindo dessa forma uma ótima utilização dos recursos (VAQUERO et al., 2008). Essa característica de reconfiguração permite que provedores de serviços de *cloud computing* ofereçam seus serviço aos seus clientes seguindo um modelo de *pay-to-use* (pague para usar), modelo que retira do usuário a necessidade de reservar quantidades adicionais de recursos, pagando somente pelo recurso utilizado em cada momento. Permitindo dessa forma que aplicações que tenham um consumo dinâmico de recursos sejam cobradas apenas pelos recursos consumidos (YOUSEFPOUR et al., 2019).

2.2.1 Cloud Computing e IoT

Com seus recursos virtualmente ilimitados, a *cloud computing* pode fornecer recursos necessários para suprir algumas limitações tecnológicas da IoT. Os dispositivos IoT, em geral, oferecem recursos bem limitados, como a reduzida capacidade de processamento e armazenamento. Além disso, a *cloud* pode fornecer diversas soluções para implementações do *middleware* da IoT, bem como para implementações de aplicações e serviços que exploram os dados produzidos pela IoT (BOTTA et al., 2016).

A *cloud* facilita o processo de integração e processamento dos dados da IoT bem como

a instalação de novos dispositivos e a integração entre eles. Na parte de comunicação, a *cloud* oferece soluções baratas para a conexão e gerenciamento, além de disponibilizar serviços de rede de alta velocidade facilitando o monitoramento e controle remoto dos dispositivos conectados. A IoT pode produzir uma quantidade enorme de dados dos mais variados tipos, enquanto a *cloud* fornece armazenamento virtual ilimitado a um baixo custo. Ademais, dispositivos IoT têm suas capacidades de processamento limitada e muitas aplicações exigem a execução de algoritmos que precisam de alto poder de processamento. Sendo assim, a *cloud* é uma excelente solução de baixo custo para tais aplicações (BOTTA et al., 2016).

Diversos são os benefícios da utilização da *cloud* em conjunto com IoT, contudo, há ainda alguns problemas críticos que são base de muitos estudos. Sistemas IoT podem requerer uma contínua transmissão de grande volume de dados produzidos por rede de sensores que acabam por consumir grande parte da largura de banda da rede, podendo dessa forma comprometer o seu funcionamento, além de elevar os custos relacionados à transmissão. Outro ponto importante em algumas aplicações IoT é a necessidade de tempos de resposta altamente reduzidos com valores próximos a alguns milissegundos e um sistema de tolerância a falhas eficaz, como é o caso por exemplo de sistemas de monitoramento da saúde, sistemas de comunicação entre veículos, sistemas de alarme de desastres e controle de voos. Além disso, alguns desses sistemas não podem ser interrompidos devido a falta de conectividade com a *cloud*, muitas vezes causada pela movimentação dos dispositivos por zonas sem conectividade, como no caso de veículos, ou em casos em que a conectividade é comprometida pela movimentação de satélites ou efeitos atmosféricos (YANNUZZI et al., 2014; CHIANG; ZHANG, 2016; BOTTA et al., 2016).

2.3 FOG COMPUTING

Fog computing é um paradigma distribuído que tem como propósito levar recursos computacionais para próximo dos usuários ou dispositivos da IoT. A *fog computing* pode ser vista como uma extensão da *cloud computing*, fornecendo recursos computacionais, armazenamento e serviços de rede para dispositivos na borda da rede de forma descentralizada. (Figura: 2.4) (BONOMI et al., 2012; CHIANG; ZHANG, 2016; DASTJERDI; BUYYA, 2016). A principal vantagem da *fog computing* em relação a *cloud computing* está na sua proximidade com o usuário final, essa característica proporciona tempos de respostas mais curtos para aplicações sensíveis ao atraso, dado que o caminho que a informação precisa percorrer é consideravelmente menor. A partir disso, tomadas de decisões, gerenciamento de dados e outras funções presentes em aplicações IoT não precisam ser executadas apenas na *cloud*, mas ao longo do caminho entre os dispositivos IoT e a *cloud* (YOUSEFPOUR et al., 2019).

fog e *cloud* se complementam, a fim de fornecer serviços de forma contínua em qualquer lugar. Ideal para cenários de aplicações IoT, a *fog* pode reduzir a latência de aplicações realizando tarefas analíticas e de controle próximas aos dispositivos. Por meio do processamento de dados dentro da *fog* é possível reduzir a quantidade de dados enviados para servidores da *cloud*, promovendo um melhor uso da rede. A *fog* pode operar de forma

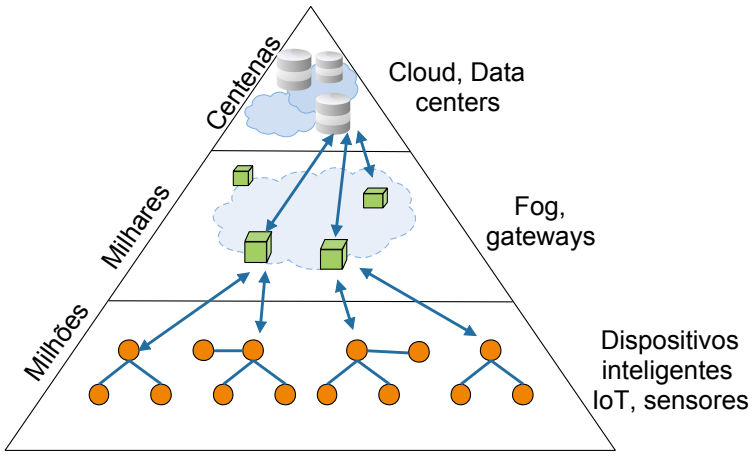


Figura 2.4 O papel da *cloud* e da *fog* na entrega de serviços IoT (AL-FUQAHA et al., 2015).

autônoma, não interrompendo os serviços em caso de falha de conectividade com a *cloud*, além disso, a *fog* pode ser escalada de acordo com a necessidade dos usuários (BONOMI et al., 2012; CHIANG; ZHANG, 2016).

A *fog* é apontada como uma solução para as aplicações IoT devido às diversas características, entre elas estão:

Tempo Real: Os recursos da *fog* são posicionados entre o dispositivos IoT e os servidores da *cloud*, promovendo melhorias no tempo de resposta das aplicações. Essa característica é essencial para aplicações que buscam trabalhar em tempo real. (AL-FUQAHA et al., 2015)

Distribuição e Escalabilidade: Em contraste com o modelo centralizado de *cloud*, o modelo da *fog* é baseado em *micro data centers* chamados de *cloudlets*, que são distribuídos geograficamente em diferentes regiões e são utilizados de acordo as necessidades das aplicações (AL-FUQAHA et al., 2015; BONOMI et al., 2012).

Alívio da Rede: O processamento dos dados de forma hierárquica pela *fog* reduz a quantidade de dados que precisam ser enviados para a *cloud* (CHIANG; ZHANG, 2016).

Não Interrupção dos serviços: Os sistemas *fog* são capazes de operar de forma autônoma, não interrompendo os serviços em casos cuja conectividade com a *cloud* é perdida (CHIANG; ZHANG, 2016).

2.3.1 Arquitetura em Camadas

Atualmente, trabalhos relevantes têm apresentando propostas de modelos de referências para arquiteturas de *fog* baseado em camadas. De modo geral, os modelos são baseados em uma estrutura de três camadas, tendo a *cloud* no topo, seguida pela camada de *fog* e, por fim, os dispositivos IoT na camada mais inferior. A camada de *fog* propõe a

redução da distância entre os dispositivos IoT e o poder computacional necessário para o processamento e execução das aplicações. Para isso, adiciona recursos computacionais entre a camada de *cloud* e os dispositivos IoT, tais recursos devem estar de fato próximos fisicamente aos dispositivos finais. Devido a isso, esses recursos devem estar distribuídos geograficamente para que seja possível ter uma maior área de abrangência (HU et al., 2017; VARSHNEY; SIMMHAN, 2017).

Os recursos da camada de *fog* podem ser organizados em um sistema de níveis, isso ajuda a construir uma espécie de hierarquia dentro da *fog*, geralmente partindo de níveis com recursos mais escassos, porém, mais próximos aos dispositivos IoT, até níveis da *fog* com mais recursos que podem estar um pouco mais distante dos dispositivos IoT, mas não tão distantes quanto a *cloud*. Dessa forma, a camada da *fog* pode ser composta por vários níveis, e os recursos podem ser consumidos de acordo com a necessidade a partir dos mais próximos ao dispositivo IoT, passando pelos níveis da *fog* até a *cloud* (VARSHNEY; SIMMHAN, 2017; CHARÂNTOLA et al., 2019).

Em cada nível, os recursos podem estar distribuídos entre diferentes *cloudlets*, ou como encontrado em alguns trabalhos, *micro data center, nano data center, local cloud*. Em cada *cloudlet*, diferentes recursos podem estar disponíveis para atender as demandas. Dentre os recursos é possível destacar o poder de computação (CPU), memória RAM, armazenamento, processamento gráfico (GPU) e comunicação (rede) (VARSHNEY; SIMMHAN, 2017; PEIXOTO; GENEZ; BITTENCOURT, 2022; CHARÂNTOLA et al., 2019).

Assim como em uma *cloud*, de forma a melhorar a eficiência no uso dos recursos, diferentes aplicações podem compartilhar recursos entre si, visto que, cada *cloudlet* possui recursos limitados. Em alguns casos, é necessário que aplicações sejam encaminhadas para serem executadas em outros níveis da arquitetura da *fog*. Nessa situação, é preciso tomar decisões sobre qual aplicação ou qual parte da aplicação precisa ser executada em uma outra *cloudlet*, ou mesmo, o que poderia ser executado diretamente na *cloud* (PEIXOTO; GENEZ; BITTENCOURT, 2022; CHARÂNTOLA et al., 2019).

2.3.2 Gerenciamento de Recursos

O gerenciamento de recursos é tarefa fundamental para tornar a *fog computing* uma realidade. Os recursos limitados e os diferentes requisitos de QoS das aplicações fazem com que diferentes estudos sejam guiados ao aprimoramento de técnicas e ferramentas para uso mais eficiente dos recursos, entre os tipos de recursos podemos destacar o poder computacional de processamento e a comunicação de rede, objetos de estudo deste trabalho, além de consumo energético e capacidade de armazenamento (GHOBAEI-ARANI; SOURI; RAHMANIAN, 2020; TOCZÉ; NADJM-TEHRANI, 2018; HU et al., 2017).

Entre os objetivos do gerenciamento de recursos está a alocação de recursos, que determina onde e com que recursos cada aplicação será executada. Essa decisão poderá levar em consideração o consumo de energia, o tráfego de dados pela rede, o processamento necessário, entre outras questões. Tais decisões afetam diretamente o tempo de resposta das aplicações. Diferentes estudos buscam encontrar boas soluções, ou um conjunto de soluções para esse problema. A virtualização dos recursos é algo que tem apresentado

bons resultados. O compartilhamento de recursos entre diferentes aplicações por meio da virtualização permite que aplicações façam um melhor uso dos recursos disponíveis (HU et al., 2017; GHOBAEI-ARANI; SOURI; RAHMANIAN, 2020; **??**).

Considerando a natureza das arquiteturas da *fog*, levando em consideração suas camadas e níveis hierárquicos com suas *cloudlets* heterogêneas, considerando também a possibilidade do compartilhamento de recursos entre aplicações e as implicações das decisões de onde devem ser executadas cada aplicação, políticas são utilizadas para auxiliar no processo de decisão e alocação dos recursos, definindo regras e protocolos que devem ser seguidos buscando maior eficiência e uma melhor aderências aos requisitos de cada aplicação (PEIXOTO; GENEZ; BITTENCOURT, 2022; GUPTA et al., 2017).

2.4 APLICAÇÕES NA FOG

Assim como na *cloud*, aplicações de diferentes naturezas podem ser executadas na *fog*, de maneira geral, qualquer aplicação que seja executada em *cloud* pode ser executada em uma *fog*. Contudo, as características da *fog* são mais atrativas para aplicações com certos requisitos. O tempo total de resposta é uma das métricas que a *fog* busca melhorar, aplicações que exigem tempos curtos de resposta, são beneficiadas pelo uso da *fog*, uma vez que a distância necessária para trafegar os dados entre o dispositivo e uma *cloudlet* tende a ser menor que a distância necessária para trafegar o mesmo conjunto de dados entre o dispositivo e a *cloud*, dessa forma é possível reduzir o tempo total entre a requisição e a chegada da resposta.

Outra característica que pode ser obtida pelo uso da *fog* é a redução do tráfego de dados pela rede, visto que, uma vez que o dado precisa percorrer uma distância menor, o uso da rede de modo geral, tende a ser menor. Isso beneficia diretamente aplicações com alta taxa de transmissão de dados, em que o uso da *fog* pode aliviar o fluxo de dados, e consequentemente reduzir os custos com a transmissão.

2.5 IFOGSIM

O *iFogSim* é um simulador de ambientes de *fog computing* que permite a modelagem e simulação de cenários, bem como, avaliar e observar o comportamento do ambiente quanto ao gerenciamento e aplicações de políticas de agendamento de recursos. Foi desenvolvido baseado no *CloudSim*, um simulador de ambientes *cloud* amplamente utilizado na literatura. Com o *iFogSim* é possível coletar métricas como atrasos e tráfego de dados na rede, custos operacionais e consumo de energia (GUPTA et al., 2017).

No *iFogSim*, as aplicações são modeladas baseando-se em um modelo de fluxo de distribuição de dados, constituídas por grupos de módulos interdependentes que recebem dados, processam e imputam os dados resultantes do processamento em outros módulos. O modelo da aplicação pode ser representado por um grafo direcionado, tendo os módulos como seus vértices e a dependência entre os módulos como as arestas do grafo. Além disso, as aplicações podem estar conectadas a sensores que podem ser modelados para emitirem um fluxo de dados contínuo e atuadores que podem ser acionados por outros módulos, produzindo assim um modelo do tipo *Sense-Process-Actuare* cujo dado é coletada pelos

sensores, enviada para os módulos de processamento de dados que podem ocasionalmente enviar comandos para os atuadores (GUPTA et al., 2017).

A Figura 2.5 apresenta um diagrama de algumas das classes do *iFogSim*. Um cenário é formado por um modelo físico dos dispositivos e topologia da rede, um modelo lógico das aplicações e um modelo de gerenciamento e política de alocação de recursos. Para o modelo físico é disponibilizada a classe *FogDevice* que deve especificar as características de *hardware* dos dispositivos e suas conexões, a classe *Sensor* que simula sensores IoT emitindo dados para *FogDevices* e a classe *Actuator* que permite a modelagem de atuadores. Além disso, a classe *Tuple* representa uma unidade fundamental de comunicação, durante a simulação *Tuples* são geradas a todo momento representando a comunicação entre entidades. Elas carregam informações sobre tamanho de dados, requisitos de processamento, origem e destino dos dados (GUPTA et al., 2017).

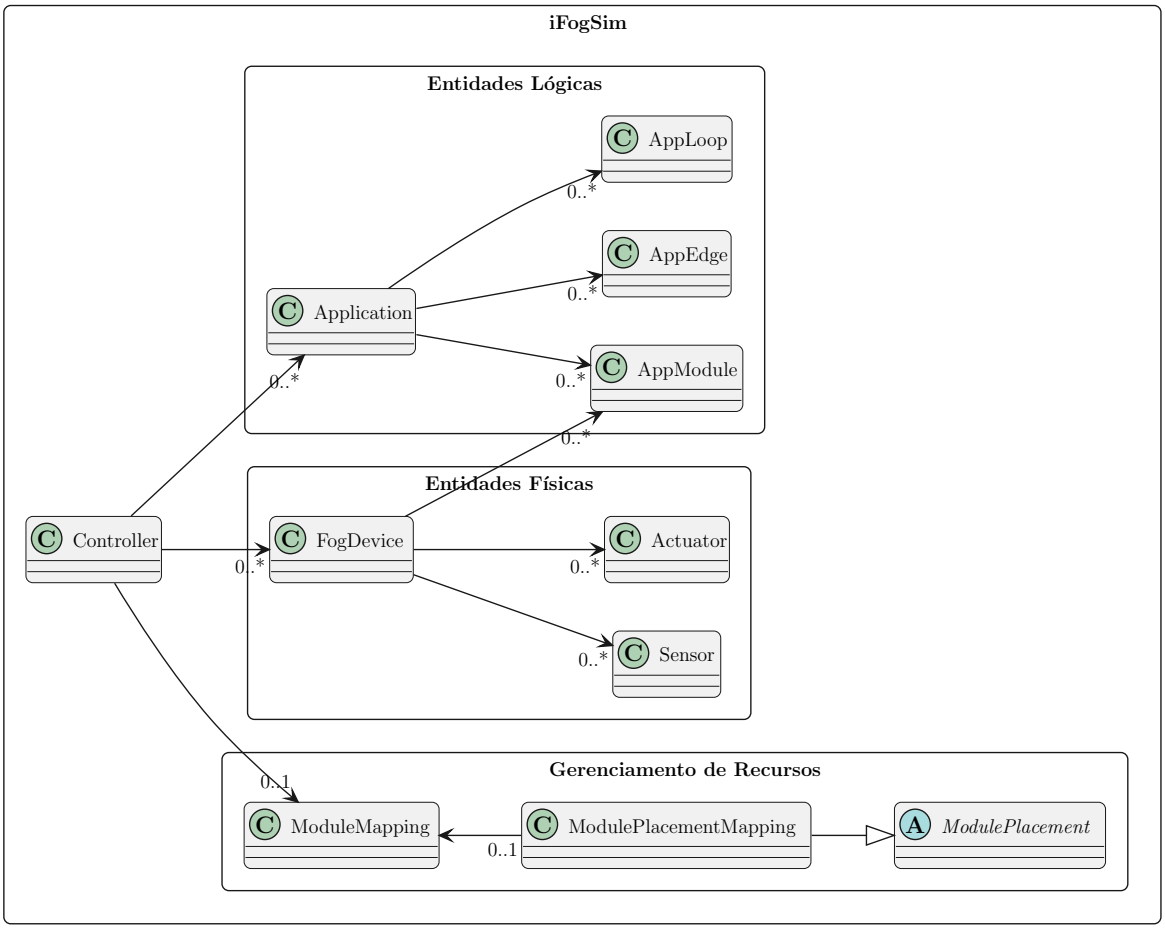


Figura 2.5 Diagrama UML - Principais classes do *iFogSim*.

Como visto, as aplicações podem ser modeladas como grafos direcionados, para isso, o *iFogSim* fornece um conjunto de classes para auxiliar nessa modelagem (Figura: 2.5). Uma *Application* é formada por um conjunto de *AppModule* e *AppEdge*, a classe *AppModule* é utilizada para representar instâncias de módulos de processamento de dados da aplicação, assim que um elemento do tipo *Tuple* chega no dispositivo que contém o

módulo de destino, o *AppModule* processa e gera um ou mais novos elementos do tipo *Tuple* e encaminha a outro módulo ou a um *Actuator*. Com a classe *AppEdge* é possível modelar as arestas do grafo direcionado, em outras palavras, modela a dependência entre os módulos da aplicação, são utilizadas para configurar o comportamento do *AppModule* no que diz respeito a emissão de elementos do tipo *Tuple*, que podem emiti-los na ocorrência de algum evento específico ou de acordo um determinado período de tempo.

A classe *AppLoop* é uma classe adicional utilizada para medir atrasos de uma requisição, permitindo especificar o caminho do grafo que se deseja medir (GUPTA et al., 2017). Para a alocação de aplicações nos dispositivos, o *iFogSim* fornece a classe *ModulePlacement*, além de algumas implementações concretas dessa classe. As classes dessa hierarquia servem para definir em qual dispositivo (*FogDevice*) os módulos (*AppModule*) das aplicações (*Application*) serão alocados (GUPTA et al., 2017).

2.6 CONSIDERAÇÕES FINAIS

Este capítulo introduziu os diversos conceitos abordados neste trabalho, foram apresentados os conceitos de IoT, *cloud computing* e *fog computing* bem como a relação de complementação entre cada um desses paradigmas, acompanhada da motivação para o surgimento da *fog*. Além disso, foi destacada a arquitetura de múltiplas camadas da *fog*, bem como a importância do gerenciamento dos recursos disponíveis. Por fim, foi apresentada uma visão geral do simulador de ambientes de *fog computing* que será utilizado para a simulação e avaliação da proposta deste trabalho. No próximo capítulo será apresentado o conjunto de trabalhos relacionados que serviram de base para esta pesquisa.

Capítulo 3

TRABALHOS RELACIONADOS

Devido às diversas limitações e desafios associados ao uso da *cloud computing* para algumas aplicações, principalmente aquelas que fazem uso do paradigma de *Internet of Things* (IoT), a *fog computing* surge como uma opção fornecendo poder computacional nas proximidades do usuário. Diversos tipos de aplicações podem ser beneficiadas por esse paradigma, principalmente aplicações que precisam de respostas rápidas, sistemas que trabalham com reconhecimento facial, reconhecimento de voz e tradução, aplicações voltadas para a saúde, sistemas de vigilância e emergência, jogos em tempo real entre outras aplicações. (CHAMOLA; THAM; CHALAPATHI, 2017). Contudo, devido a natureza da *fog computing* e das aplicações que a utilizam, uma série de outros desafios foram apresentados e começaram a ser estudados. Rejiba, Masip-Bruin e Marín-Tordera (2019) apresentam um série de oportunidades de pesquisa em seu trabalho, entre elas, aponta o processo de tomada de decisões quanto ao dispositivo em que as aplicações devem ser executadas ou para onde devem ser migradas de forma a melhorar alguma métrica de Qualidade de Serviço (QoS), onde essa decisão deve ser tomada e quais os impactos no processo.

Yangui et al. (2016) propõem uma arquitetura de uma Plataforma como um Serviço (PaaS) para automatizar o provisionamento de aplicativos em ambientes híbridos *cloud-fog*. A arquitetura PaaS proposta, permite desenvolver aplicações de acordo com o domínio alvo, configurar e dimensionar recursos para implantar e executar os componentes das aplicações, gerenciar o fluxo de execução, efetuar o monitoramento de *Service Level Agreement* (*SLA*), efetuar migração de componentes, além de fornecer interfaces para o gerenciamento de recursos e componentes. Diferentes cenários de alocação de aplicações foram observados e diferentes resultados foram encontrados para cada arranjo de alocação. O autor conclui que a organização ideal dos componentes das aplicações não é uma tarefa trivial e exige o uso de algoritmos sofisticados para conseguir melhores resultados.

Chamola, Tham e Chalapathi (2017) utiliza uma rede definida por software (SDN) contendo várias *cloudlets* para executar serviços na proximidade dos usuários de dispositivos móveis. Para Chamola, Tham e Chalapathi (2017), executar tarefas na rede de

cloudlets pode ser uma solução para melhorar a QoS no que diz respeito ao tempo de resposta dos serviços. Com base na política proposta, se um *cloudlet* ficar sobrecarregada, as tarefas devem ser transferidas e processadas em outra *cloudlet* da rede que esteja com recursos disponíveis. O gerenciamento e a distribuição das tarefas é executado por um serviço centralizado chamado pelo autor de *Cloudlet Central Manager*.

Taneja e Davy (2017) apresenta um algoritmo para mapeamento de módulos de aplicativos IoT baseado em uma arquitetura de *fog computing* que considera os requisitos de cada módulo da aplicação, como CPU, Memória e largura de banda. O autor busca alocar os módulos das aplicações o mais próximo dos usuários e dispositivos finais. Para isso considera os recursos disponíveis em cada nó da *fog*, e busca reduzir a ociosidade dos recursos que estão mais próximos da borda da rede. O autor chama atenção para a relevância da utilização de uma arquitetura baseada em *fog computing* e *cloud computing* quando se busca a redução do tempo de resposta das aplicações. Apesar de considerar que os módulos das aplicações podem ser alocados em diferentes dispositivos, Taneja e Davy (2017) não considera a prioridade das aplicações nem o impacto da comunicação entre esses módulos.

Shah-Mansouri e Wong (2018) fazem um estudo sobre a alocação de tarefas em uma arquitetura de *fog computing* hierárquica, com propósito de maximizar a qualidade de experiência do usuário buscando reduzir o tempo de resposta das aplicações, bem como a redução de custo com energia. Para isso, fazem uso de algoritmos e abordagens e abordagens baseadas na teoria dos jogos . Segundo os autores, os resultados dos experimentos mostraram que os usuários em geral obtiveram uma melhor *Quality of Experience* (*QoE*) quando comparada a outras propostas presentes na literatura. Contudo, neste trabalho os autores não atacam a ideia de aplicações modulares, mas sim, um sistema de execução de tarefas, onde as aplicações podem ser executadas a qualquer momento em qualquer dispositivo, desde que a tarefa seja direcionada àquele dispositivo.

Aburukba et al. (2020) propõe a utilização de uma algoritmo genético customizado para resolver o problema de escalonamento de requisições recebidas de dispositivos IoT em uma arquitetura de *fog computing*. Com objetivo principal de minimizar a latência das aplicações, a proposta consegue apresentar bons resultados em comparação a outras técnicas como *round robin* ou fila de prioridade. Ali et al. (2020) apresenta uma adaptação do algoritmo genético *Non-dominated Sorting Genetic Algorithm (NSGA-II)* que busca reduzir o tempo de resposta e custo de execução de aplicações. Ambas as propostas utilizam um sistema de decisão centralizado.

Kaur, Auluck e Rana (2022) apresenta o RTH^2S - *Real Time Heterogeneous Hierarchical Scheduling*, um algoritmo para escalonamento de tarefas em tempo real projetados para operar em uma arquitetura de *fog computing* multinível. A premissa do trabalho é que exista um grupo de tarefas com diferentes tamanhos, prioridades e prazos, que precisam ser executadas, dessa forma, o algoritmo tentará agendar a execução do grupo de tarefas considerando seus diferentes perfis, privilegiando aquelas com mais prioridades e de prazos mais curtos. Para avaliar o RTH^2S, Kaur, Auluck e Rana (2022) utilizaram o *iFogSim* e uma base real de registros de execução de tarefas para comparar a performance da proposta com a execução diretas das tarefas em um *cloud*, assim como, com o algoritmo de agendamento de tarefas *LTF - Longest Time Firt*, que tenta alocar tarefas

maiores em nós da *fog* em que seriam executados mais rapidamente. Em seu experimento o RTH^2S conseguiu atingir melhores taxas de sucesso ao entregar a maior quantidade de tarefas respeitando o prazo, além disso conseguiu um menor custos para a execução das tarefas em relação ao *LTF*. Contudo, neste trabalho não são consideradas aplicações modulares e a alocação de recursos é baseada em tarefas individuais.

Bittencourt et al. (2017) discute o problema de alocação de aplicações em dispositivos que compõem a *fog* de nível único utilizando o simulador *iFogSim*. O trabalho faz um estudo do comportamento de duas aplicações distintas na *fog*, com diferentes quantidade de usuários movendo-se entre regiões, levando em consideração três diferentes estratégias de alocação de recursos: Concorrente, onde as aplicações requisitadas são alocadas concorrentemente em uma mesma *cloudlet* sem considerar o limite de uso de recursos, *First Come-First Served* (*FCFS*) que consiste em disponibilizar os recursos de acordo a ordem de chegada das requisições de alocação e quando não houver mais recursos, direcionar as requisições para a *cloud* e por fim, *Delay-Priority* (*DP*), que leva em conta a prioridade da aplicação, nesse caso, aplicações com requisitos de tempo de resposta mais apertados são alocadas com prioridade em relação às demais aplicações, quando não houver mais recursos na *cloudlet* as aplicações são direcionadas para a *cloud*. Conforme a quantidade de usuário aumentou em uma determinada *cloudlet DP* apresentou um melhor desempenho quanto ao tempo de resposta das aplicações sensíveis a latência ao custo do aumento de tráfego na rede. A abordagem Concorrente, apesar de apresentar um tráfego de rede mínimo, não apresentou bons resultados quanto a latência das aplicações, isso devido ao fato de todas as aplicações serem executadas em uma única *cloudlet* de forma concorrente, comprometendo dessa forma a execução das tarefas. Bittencourt et al. (2017) não levou em consideração um cenário de uma *fog* com múltiplos níveis, além disso, considerou os módulos das aplicações como um grupo único que precisava ser posicionado em conjunto no mesmo dispositivo.

Charântola et al. (2019) propõe o escalonador *Delay-Priority & Individual* (*DP-I*) para alocação de recursos em uma arquitetura de uma *fog* de nível único. O estudo leva em consideração que as aplicações podem ser divididas em módulos que comunicam-se entre si e podem estar posicionadas em diferentes dispositivos. Quando há a necessidade de alocar recursos para uma nova aplicação, ou para novos módulos de uma aplicação já em execução em uma determinada Cloudlet, e na ausência de recursos suficientes, o algoritmo ao invés de elevar toda a aplicação para a *cloud*, seleciona apenas os módulos dependentes, que comunicam-se entre si, e encaminha-os para a *cloud*. Além disso, o algoritmo prioriza manter na *cloudlet* aplicações com requisitos de latência menores. Charântola et al. (2019) Compara a abordagem proposta com os algoritmos avaliados por Bittencourt et al. (2017). Os resultados sugerem que com a nova proposta foi possível reduzir a média de atrasos em aplicações com requisitos de latência menores e que usuários podem sentir diferenças de QoS na medida em que os serviços são migrados para o serviço de nuvem. Contudo, não é considerada uma fog de múltiplos níveis.

Peixoto, Genez e Bittencourt (2022) propõe o *Communication Based & Edgewards* (*CB-E*), uma abordagem que busca reduzir o tráfego de rede tentando atender aos requisitos de latência das aplicações. Para isso, quando na necessidade de alocação de recursos para novas aplicações em uma Cloudlet, e não havendo recursos suficientes, o algoritmo

avalia o grupo de módulos que devem ser migrados, para isso considera o aumento do tráfego de rede que poderá ser produzido quando determinados módulos ou grupos de módulos dependentes forem enviados para níveis mais elevados de uma fog. Então envia para camadas superiores da fog todos os módulos de um determinado tipo ou grupos que provoquem o menor impacto na rede. Peixoto, Genez e Bittencourt (2022) avaliar seus resultados em três cenários diferentes, considerando cenários com fog de apenas um nível e com dois níveis, também variando a capacidade computacional das Cloudlets. Peixoto, Genez e Bittencourt (2022) Comparam a a proposta com os algoritmos avaliados por Bittencourt et al. (2017) e a abordagem proposta em Charântola et al. (2019). Como resultado, o *CB-E* mostrou-se capaz de reduzir o tráfego total de dados enquanto manteve atrasos aceitáveis em aplicações sensíveis.

3.1 DISCUSSÃO DOS TRABALHOS RELACIONADOS

Como pode ser visto na Tabela 3.1, diferentes estudos vêm sendo conduzidos em torno da alocação de recursos para aplicações em arquiteturas de rede baseadas em *fog computing*. Dentre os trabalhos relacionados considerados na revisão, a maioria deles não levaram em consideração que a camada da *fog* pode ter vários níveis, além disso, apenas três levaram em consideração o tráfego de dados gerado pela comunicação ou mesmo o atraso causado pela distância entre os dispositivos.

Outra questão importante é a natureza modular das aplicações, que só foi considerada em quatro trabalhos, ainda que alguns trabalhos tratem apenas de execução de tarefas isoladas e não da reserva dos recursos para determinadas aplicações. Destaca-se que apenas três trabalhos utilizam uma estratégia distribuída para a tomada de decisão quanto ao local de alocação de recursos para as aplicações ou para os seus módulos.

Trabalho	Fog	Multinível	Avaliação		Aplicação	Gerenciamento
			CPU	Comunicação	Modular	Distribuído
Shah-Mansouri e Wong (2018)	✓	✓	✓			
Yangui et al. (2016)	✓	✓			✓	
Chamola, Tham e Chalapathi (2017)	✓		✓	✓		
Taneja e Davy (2017)	✓		✓			
Aburukba et al. (2020)	✓		✓	✓		
Ali et al. (2020)	✓		✓			
Kaur, Auluck e Rana (2022)	✓	✓	✓			
Bittencourt et al. (2017)	✓		✓		✓	✓
Charântola et al. (2019)	✓		✓		✓	✓
Peixoto, Genez e Bittencourt (2022)	✓	✓	✓	✓	✓	✓
Least Impact - X (LI-X)	✓	✓	✓	✓	✓	✓

Tabela 3.1 Resumos dos trabalhos relacionados

Este trabalho é baseado nas pesquisas de outros autores da literatura. A figura 3.1 apresenta a sequência dos trabalhos diretamente relacionados a esta pesquisa. No pri-

meiro trabalho, Bittencourt et al. (2017) apresenta e avalia a alocação de aplicações utilizando os algoritmos *FCFS* e *DP* que considera a CPU disponíveis no dispositivo para a alocação de novas aplicações, mas não considera a modularidade das aplicações. Na sequência, Charântola et al. (2019) apresentam o algoritmo CB, que leva em consideração a modularidade das aplicações e as suas prioridades, buscando reduzir o seu tempo resposta. Por fim, Peixoto, Genez e Bittencourt (2022) apresenta o *CB-E*, que além considerar a modularidade das aplicações também avalia o impacto da comunicação entre cada um dos seus módulos e apresenta uma *fog* de dois níveis hierárquicos.

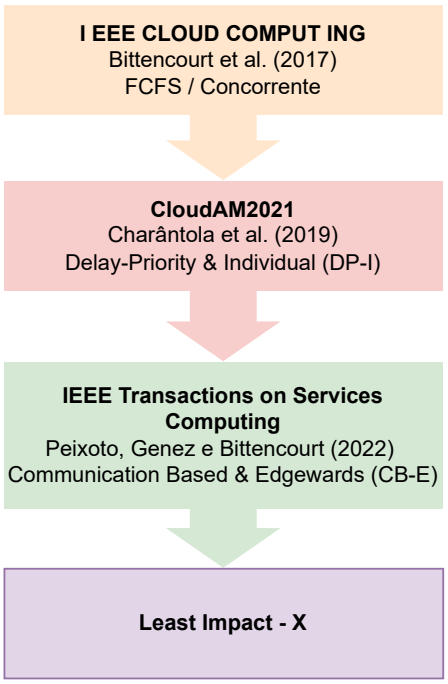


Figura 3.1 Ordem Cronológica dos Trabalhos.

3.2 CONSIDERAÇÕES FINAIS

Apesar do trabalho de Peixoto, Genez e Bittencourt (2022) apresentar bons resultados, no que se refere a redução do tráfego de dados e no atendimento aos requisitos de latência das aplicações, a proposta deste trabalho traz uma nova abordagem para a política de decisão de alocação de recursos, que ao contrário do que é visto em Peixoto, Genez e Bittencourt (2022), onde, ao decidir e identificar o grupo de módulos que deverá ser elevado para camadas superiores da *fog*, eleva todas as instâncias em execução de cada um desses módulos, o LI-X busca elevar, desse grupo, apenas os módulos que considera necessários para abrir espaço para os módulos da nova aplicação. Além disso, na avaliação sobre a necessidade de recursos de cada módulo o LI-X leva em consideração a média de utilização de recursos pelo módulo. Com isso, é possível reduzir a ociosidade dos recursos nos níveis mais baixos da *fog*. Este trabalho também busca avaliar o comportamento dos algoritmos e das aplicações em uma *fog* com um quarto nível hierárquico.

Capítulo 4

LEAST IMPACT - X

Neste capítulo será apresentada a proposta de um algoritmo para alocação de aplicações modulares em uma arquitetura hierárquica de *fog computing*.

4.1 VISÃO GERAL

Em uma arquitetura de *fog computing*, os recursos e dispositivos das *cloudlets* estão distribuídos geograficamente, sempre buscando estar próximos dos dispositivos finais. A *fog* foi proposta com objetivo de encurtar a distância entre o poder computacional presente na *cloud* e os dispositivos que fazem uso desses recursos, dessa forma, conseguindo tempos de resposta mais rápidos. Como visto anteriormente, os recursos presentes na camada de *fog* estão organizados em *cloudlets*, que podem ser heterogêneas entre si, ou seja, podem fornecer diferentes quantidades e tipos de recursos computacionais, e ainda podem estar hierarquicamente organizados em níveis, onde as *cloudlets* mais próximas aos dispositivos da *Internet of Things* (IoT), geralmente oferecem menos recursos e as que estão mais distantes dos dispositivos IoT, fornecem geralmente mais recursos (Figura: 4.1). Neste trabalho, o nível mais próximos aos dispositivos é considerando o nível 1, o nível mais próximo a *cloud* é considerando o nível n e os níveis intermediários dependerá de cada arquitetura, podendo ser representando pelo conjunto $níveis = \{1, 2, ..., n-1, n\}$.

Diversos dispositivos IoT podem usar essa arquitetura e consequentemente diversas aplicações podem ser executadas. As aplicações podem ser de diferentes naturezas e organização. Neste trabalho, consideramos que as aplicações podem ser modulares, ou seja, a aplicação é composta por módulos que podem ser executados individualmente em diferentes *cloudlets*. Além disso, cada aplicação pode ter diferentes níveis de prioridade quanto ao tempo de resposta e cada módulo pode fazer uso em maior ou menor grau de diferentes recursos computacionais. Nesse sentido, faz-se necessário escolher em qual *cloudlet* cada módulo da aplicação será executado a fim de atender melhor os requisitos de cada aplicação.

Neste trabalho, os módulos das aplicações são alocados em recursos disponíveis em uma arquitetura de *fog computing* de forma a reduzir o atraso em aplicações sensíveis e ainda procura reduzir o tráfego de dados pela rede. A ideia geral deste trabalho é

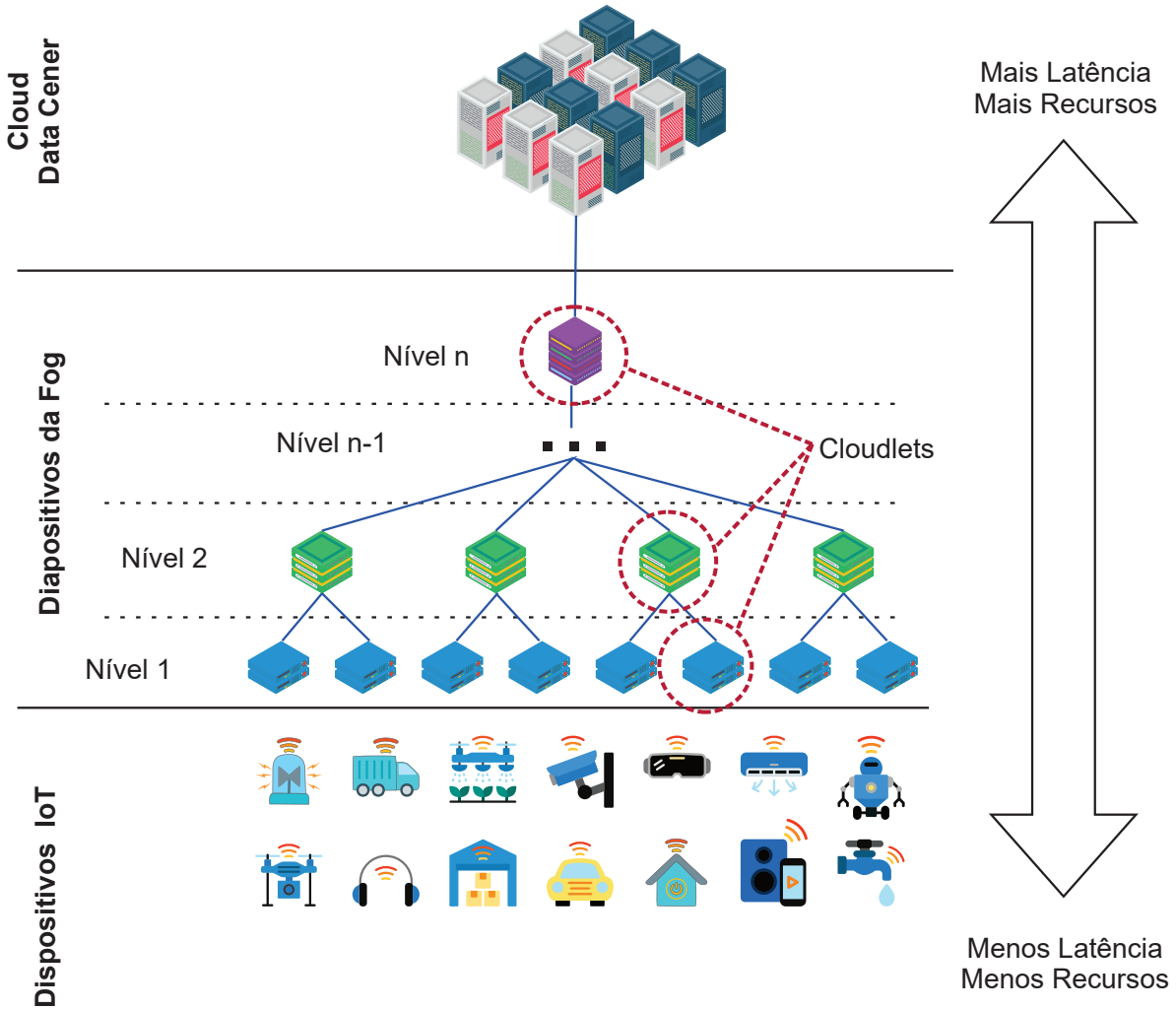


Figura 4.1 Visão Geral da Arquitetura de *fog computing*

semelhante a propostas de outros três trabalhos da literatura: (i) Bittencourt et al. (2017) (ii) Charântola et al. (2019) e (iii) Peixoto, Genez e Bittencourt (2022).

Cloudlet

Tendo em vista que na arquitetura apresentada na Figura 4.1 a camada de *fog* é composta de diferentes *cloudlets* distribuídas por diversos níveis, cada *cloudlet* em um determinado nível l se conecta com uma outra *cloudlet* do nível imediatamente acima $l-1$, até que em seu último nível n, há uma conexão com a *cloud*. Partindo dessa premissa, definimos uma *cloudlet* como um conjunto de quatro componentes básicos (Figura 4.2):(i) a unidade de armazenamento, responsável por fornecer persistência de dados para as aplicações; (ii) a unidade de comunicação, responsável por realizar e gerenciar a conexão com outros dispositivos;(iii) a unidade computacional, responsável por fornecer recursos como memória e poder de processamento para módulos das aplicações, representados na

figura por contêineres ou máquinas virtuais; (iv) e por fim a unidade controladora, responsável por gerenciar as aplicações e o uso dos recursos. Então, cada *cloudlet* conta com o conjunto básico desses componentes.

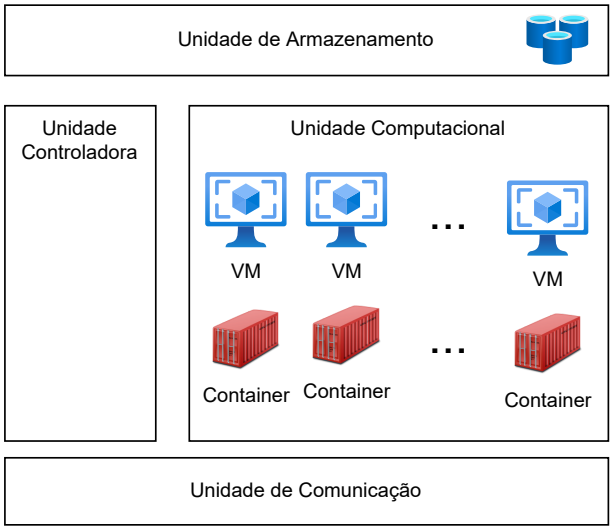


Figura 4.2 Componentes básicos de uma *cloudlet*.

Neste trabalho, utilizamos uma abordagem distribuída para a execução da tarefa de decidir o dispositivo onde cada módulo das aplicações devem ser executados. Cada *cloudlet* tem autonomia para decidir, através de sua unidade controladora, se e quais módulos das aplicações recém chegadas devem ser executadas por ela, e em alguns casos, efetuar o encaminhamento das módulos para a próxima *cloudlet* no nível imediatamente acima.

A Figura 4.3 apresenta o fluxo geral de funcionamento do LI-X. O LI-X é executado em cada unidade controladora de cada *cloudlet*. Assim que requisições de alocação de recursos para uma aplicação chegam e novas instâncias de módulos precisam ser instanciadas em uma *cloudlet*, uma avaliação com base no histórico de execuções anteriores desse módulo é executada a fim de identificar o consumo médio de recursos necessários para a sua execução, para isso, é considerada a frequência média de chegada de requisições para módulos do mesmo tipo, bem como, o consumo médio de recursos para responder a cada requisição. Neste trabalho, por utilizarmos um simulador, utilizamos os valores de médias configuradas para as distribuições de frequência de emissão de dados dos sensores e as médias configuradas nas distribuições da quantidade de recursos necessários para processamento de cada tipo de requisição. Após a obtenção do consumo médio de recursos por cada um dos módulos da aplicação, os módulos são inseridos em uma lista de módulos pendentes. A decisão de alocar recursos ou não para cada um dos módulos deverá ser feita isoladamente, inicialmente será considerado se há recursos suficientes para o módulo de acordo com a análise de recursos necessários e a disponibilidade dos recursos na *cloudlet* atual, caso existam, o módulo é marcado para ser instanciado e o próximo módulo disponível na lista começará a ser analisado, caso não existam recursos disponíveis, o algoritmo irá efetuar a verificação de quais módulos da aplicação atual

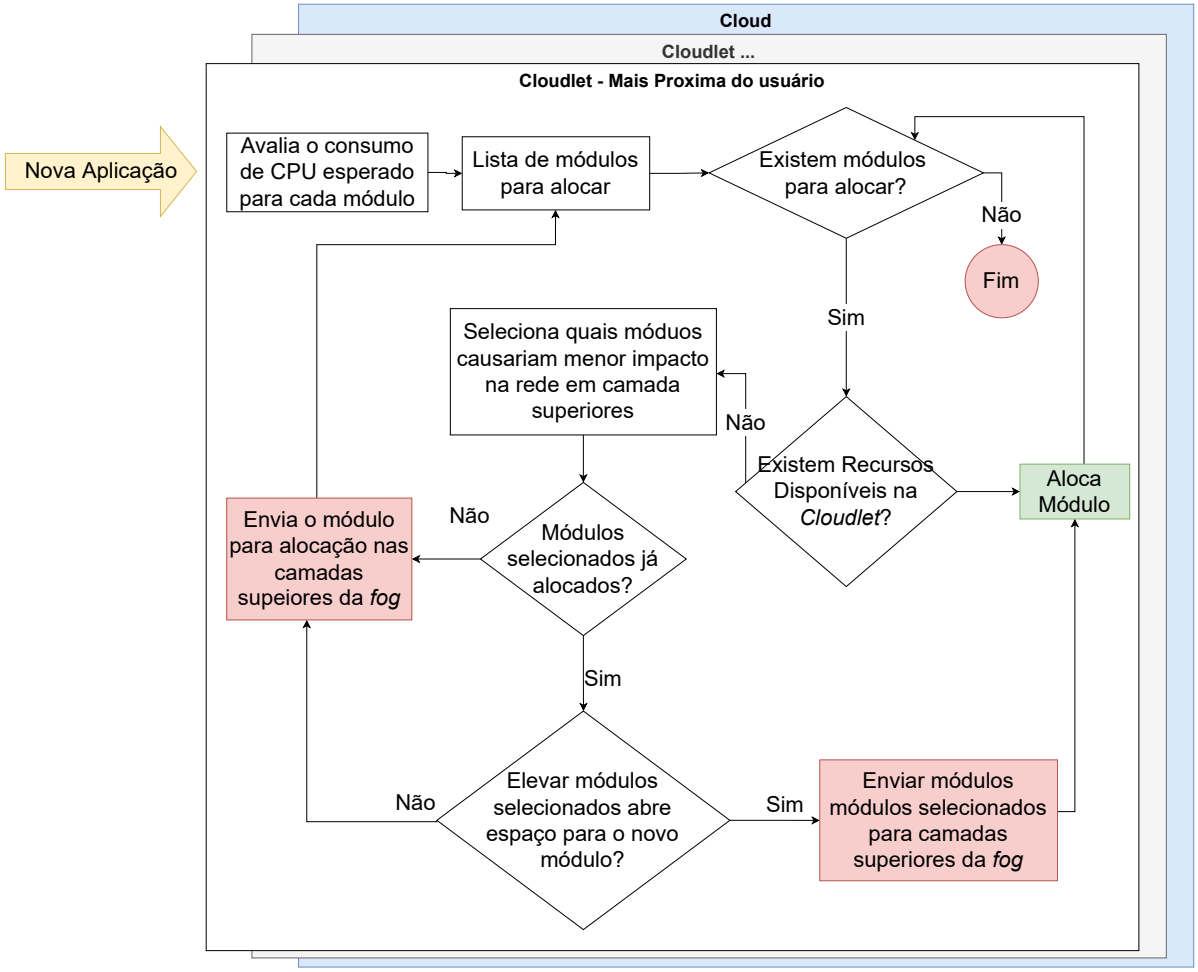


Figura 4.3 Fluxo geral de funcionamento do *Least Impact - X* (LI-X)

poderiam ser movidos para níveis superiores da *fog*, incluindo o próprio módulo, caso existam módulos candidatos a serem movidos e que possuam recursos alocados e em quantidade suficiente para abrirem espaço para a instanciação do módulo atual, esses módulos candidatos serão removidos do dispositivo atual e encaminhado para a camada superior da *fog* ou a *cloud* e o módulo atual será instanciado na *cloudlet*, caso contrário este módulo é encaminhado para ser instanciado nas camadas superiores. Quando os dispositivos de camadas superiores recebem os módulos, todo o processo de verificação é feito novamente, até que todos os módulos estejam com recursos alocados em algum dispositivo. Detalhes da implementação das partes desse processo serão apresentados na seção a seguir.

4.2 O LEAST IMPACT - X

A proposta do *Least Impact - X* (LI-X) é encontrar uma boa organização de alocação de recursos para os módulos das aplicações de forma a reduzir a latência das aplicações sensíveis bem como reduzir o impacto causado na rede pelo tráfego de dados. Para

alcançar esse objetivo, o algoritmo proposto em Peixoto, Genez e Bittencourt (2022) que identifica conjuntos de módulos com alto grau de comunicação foi utilizado como base da proposta. As demais partes do algoritmo foram desenvolvidas no decorrer deste trabalho.

A seguir, são descritas algumas definições que contribuirão para o entendimento dos algoritmos que serão apresentados.

- $A = \{a_1, a_2, \ldots, a_j\}$ representa o conjunto de aplicações modulares a serem alocadas em uma fog.
- $M(a_j) = \{m_{(j,1)}, m_{(j,2)}, \ldots, m_{(j,n)}\}$ representa os módulos da aplicação a_j.
- $M(\{a_1, ..a_j\})) = \{m_{(1,1)}, m_{(1,2)}, \ldots, m_{(j,n)}\}$ representa o conjunto de módulos das aplicações selecionadas.
- $R(c)$ representa a capacidade computacional de uma determinada *cloudlet* dado em Milhões de Instruções por Segundo (MIPS).
- $r(c)$ representa a quantidade de recursos computacionais disponíveis em uma determina *cloudlet* dados em MIPS
- $n(m)$ representa a quantidade média de MIPS necessários para a execução de um determinado módulo m sem que haja formação de filas de requisições.
- $P(c)$ representa a *cloudlet* do nível imediatamente acima da *cloudlet* c. Ou seja, caso a *cloudlet* c esteja em um determinado nível l, a *cloudlet* $P(c)$ encontra-se no nível $l + 1$ ou na *cloud*, caso l seja o maior nível na hierarquia da arquitetura tratada.

A equação 4.1 foi utilizada para calcular $n(m)$, r_i é a quantidade de recursos utilizadas para o processamento uma determinada requisição i que o módulo m processou anteriormente. A média de recursos necessárias para cada requisição é dada pelo somatório da quantidade de recursos utilizadas por cada requisição dividida pela quantidade de requisições observadas. Enquanto a quantidade de recursos necessários para o módulo é dada pela média de recursos utilizadas para atender cada requisição dividida pela frequência de requisições recebidas pelo módulo m. Dessa forma é possível prever os recursos necessários para a execução do módulo m em qualquer que seja a *cloudlet*.

$$n(m) = \frac{\frac{1}{j}\sum_{i=1}^{j} r_i}{freq(m)} \quad (4.1)$$

O Algoritmo 1 apresenta o pseudocódigo da inicialização do LI-X, inicialmente o grupo de aplicações que precisam ser instanciadas é extraído do conjunto de todas as aplicações A. Na linha 2, as aplicações são ordenadas considerando a prioridade de cada uma, aplicações sensíveis a latência tem maior prioridade sobre as demais. Na linha 3,

Algorithm 1 start (Cloudlet c)

Ensure: Cloudlet c
1: $applications \leftarrow Subset\ of\ A\ near\ c$
2: $sort(a)$
3: $modules \leftarrow M(a)$
4: **for** m in $modules$ **do**
5: $allocateOnDevice(c, m)$ (Algorithm 2)
6: **end for**

os módulos de cada uma das aplicações são extraídos e adicionados em uma lista que posteriormente terá seus elementos avaliados individualmente pelos algoritmos a seguir, na tentativa de alocar recursos para cada um deles na *fog*, partindo da *cloudlet* atual *c*.

O processo de decisão de alocação é detalhado no pseudocódigo apresentado no Algoritmo 2. Ao receber uma solicitação de alocação de recursos para um módulo m em uma *cloudlet* c é verificado se existem recursos disponíveis para alocar, se existirem, a alocação será feita (linha 1-3) e o algoritmo será encerrado, caso contrário, o Algoritmo 3 é acionado e deve retornar uma lista M_{up} dos módulos da aplicação atual já alocados na *cloudlet* c, incluindo o módulo atual m, que provocariam o menor impacto de comunicação se forem realocados para níveis superiores da *fog*. Uma vez que se tenha o retorno do Algoritmo 3, caso o módulo atual m seja o único elemento na lista M_{up}, ele será encaminhado para alocação na *cloudlet* logo acima da *cloudlet* c na hierarquia da *fog* (linhas 5-7).

Algorithm 2 allocateOnDevice ($cloudlet, module$)

Ensure: Cloudlet c, Module m
1: $hasSpace \leftarrow r(c) \leq n(m)$
2: **if** $hasSpace = true$ **then**
3: $allocate(c, m)$
4: **else**
5: $M_{up} \leftarrow modulesToUp(m, c)$ (Algorithm 3)
6: $c_{parent} \leftarrow P(c)$
7: **if** $\{modulo\} = M_{up}$ **then**
8: $allocateOnDevice(c_{parent}, m)$
9: **else**
10: $cpuToDeploy \leftarrow r(c) - n(m)$
11: $cpuToRelease \leftarrow n(M_{up})$
12: $totalInstances \leftarrow roundUp(cpuToDeploy \div cpuToRelease)$
13: **if** $countSet(c, M_{up}) \geq totalInstances$ **then**
14: **for all** $moduleToUp\ in\ M_{up}$ **do**
15: **for** $i \leftarrow 0;\ i < count;\ i \leftarrow i + 1$ **do**
16: $deallocate(c, moduleToUp)$
17: $allocateOnDevice(c_{parent}, moduleToUp)$
18: **end for**
19: **end for**
20: $allocateOnDevice(c, m)$
21: **else**
22: $allocateOnDevice(c_{parent}, m)$
23: **end if**
24: **end if**
25: **end if**

Caso o módulo m não esteja na lista ou existam outros módulos na lista M_{up}, calculam-se os recursos adicionais necessários para instanciar o módulo m na *cloudlet* c (linhas

10-12). Primeiro é verificada a diferença entre a quantidade de recursos disponíveis na *cloudlet* c em relação a quantidade de recursos necessários para a execução do módulo m. Na sequência, verifica-se a quantidade de recursos que seriam disponibilizadas se uma instância de cada um dos módulos da lista M_{up} forem movidos para outra camada da *fog*. Com base nas duas informações calcula-se quantas instâncias de cada módulo deveriam ser movidas para que recursos para o módulo m possa ser alocado na *cloudlet* c. Na sequência é verificado se existe a quantidade necessário de módulos da lista M_{up} na *cloudlet* c para serem movidos, caso exista, os módulos são removidos da *cloudlet* c e serão encaminhados para a *cloudlet* logo acima, em seguida os recursos para o módulo m são alocados na *cloudlet* atual c. Caso não existam módulos suficientes para liberar recursos com a realocação, o módulo m será encaminhado para a *cloudlet* logo acima.

Após a solicitação de alocação de um módulo em uma *cloudlet* c, o Algoritmo 3 é responsável por selecionar o grupo de módulos candidatos a serem enviados para as camadas superiores da *fog*. O primeiro passo deste algoritmo será encontrar os possíveis arranjos S de módulos que poderiam ser movidos para o nível logo acima da *fog*, esta seleção é baseada no Algoritmo 4.

Algorithm 3 modulesToUp (module, cloudlet)

Require: Cloudlet c
Ensure: Set of modules M moved upwards
1: $S \leftarrow$ *Components sets in c* (Algorithm 4)
2: **for all** *component set*$M \in S$ **do**
3: $Cost_m \leftarrow 0$
4: **for all** *component* $m \in M$ **do**
5: $Cost_m \leftarrow \text{Cost}_m + impact\ of\ M$
6: **end for**
7: **end for**
8: *Select M whose* $Cost_m$ *is minimum*

Para cada um dos conjuntos encontrados, calcula-se o possível impacto causado na rede caso o grupo de módulos sejam elevados. Esse cálculo é executado pelo Algoritmo 5. O grupo com o menor impacto será selecionado como candidato para ser enviado para as camadas superiores.

O Algoritmo 4, proposto em Peixoto, Genez e Bittencourt (2022), gera arranjos de possíveis módulos que podem ser movidos para níveis mais altos da hierarquia da *fog*. Partindo da análise dos módulos presentes em uma *cloudlet* c, o algoritmo seleciona e cria um arranjo M de apenas um elemento para cada módulo presente na *cloudlet* c que não inicia uma comunicação com outros módulos já presentes na mesma *cloudlet* c e os adicionam a um conjunto de arranjos S (linhas 1-9). Na segunda parte do algoritmo, para cada arranjo M presente em S, procura-se por módulos responsáveis por iniciarem a comunicação com algum dos módulos de M. Quando um módulo que satisfaça esse critério é encontrado, um novo arranjo M é criado, adicionando o novo módulo e então é adicionado em S. O processo é interativo e se repete até que não haja mais a criação de novos arranjos (linhas 10-20).

Algorithm 4 generateComponentsSet (Cloudlet c)

Ensure: Cloudlet c

$S \leftarrow \emptyset$
for *component* $m_s \in C$ **do**
  for *component* $m_t \in c$ **do**
    if $\notin$ *Up arc from* m_s *to* m_t **then**
      $M \leftarrow \{m_s\}$
      $S \leftarrow S \cup \{M\}$
    end if
  end for
end for
$S_{old} = \emptyset$
while $S_{old} \neq S$ **do**
  for *set* $M_s \in S$ **do**
    for *component* $m_s \in M_s$ **do**
      for m_s *incoming edge* e **do**
        $M_{m_e} \leftarrow \{m_s\} \cup \{source(e)\}$
        $S \leftarrow S \cup M_{m_e}$
      end for
    end for
  end for
end while

Por fim, o Algoritmo 5, também proposto por Peixoto, Genez e Bittencourt (2022), será utilizado para calcular o impacto que um módulo m poderá causar na rede ao ser movido para níveis superiores da *fog* em conjunto com um arranjo de módulos M. Para isso, para cada aresta da aplicação A a qual o módulo m pertence, calcula-se a Equação 4.2, cujo b é quantidade de bytes transmitidos do módulo de origem da aresta para um módulo de destino, s é a seletividade da aresta, T_a é a periodicidade na qual a aplicação transmite os dados e p é a periodicidade da aresta. Se a aresta tiver como módulo de origem o módulo m, o módulo de destino da aresta não estiver nas *cloudlets* em níveis superiores e m não pertencer ao arranjo M, então o valor da Equação 4.2 é incrementado ao impacto resultante, mas, se o módulo de destino estiver em níveis superiores, o valor da Equação 4.2 é decrementado do impacto resultante. Por outro lado, caso m seja o dispositivo de destino da aresta, o seu módulo de destino não for encontrado em dispositivos em níveis hierárquicos superiores da *fog* e m não pertencer ao arranjo M o incremento é feito. Entretanto, o decremento será realizado se o módulo de destino estiver alocado em níveis superiores da hierarquia.

$$I_e \leftarrow b \times s \times \frac{T_a}{p} \quad (4.2)$$

Algorithm 5 calcImpact (componentSet M, module m)

Require: *Component set* $M; m \in M;$ *Application arcs* E
Ensure: $totalIimpact_m$
for all *application edge* $e \in E$ **do**
  $I_e \leftarrow b \times s \times \frac{T_a}{p}$
  if m is the source of e **then**
    if No upward device has the target component of e **then**
      if $target\ component\ of\ e \notin M$ **then**
        $totalImpact_m \leftarrow totalImpact + I_e$
      end if
    else
      $totalImpact_m \leftarrow totalImpact - I_e$
    end if
  end if
  if m is the target of e **then**
    if No upward device has the source component of e **then**
      if $source\ component\ of\ e \notin M$ **then**
        $totalImpact_m \leftarrow totalImpact + I_e$
      end if
    else
      $totalImpact_m \leftarrow totalImpact - I_e$
    end if
  end if
end for

Com isso, espera-se que os recursos das *cloudlets* em níveis mais inferiores da *fog* sejam mais bem utilizados, fazendo com que a execução da aplicação tenda a ser feita próxima ao dispositivo IoT, quando houver recursos, dessa forma além de reduzir o tempo de resposta, é possível reduzir a quantidade de dados trafegados na rede, além disso, quando os níveis mais inferiores estiverem sobrecarregados, espera-se que os módulos a serem movidos para camadas superiores sejam aqueles que causem o menor impacto na rede de dados. A próxima seção apresenta uma demonstração ilustrada do que é esperado para essa proposta e a comparação com as demais abordagens utilizadas como comparativo.

4.3 EXEMPLO ILUSTRATIVO

As Figuras 4.4 e 4.5 são apresentadas como forma de ilustrar o comportamento esperado do LI-X em detrimento a outros algoritmos. Na figura, cada instância da aplicação são identificadas pelas cores, a comunicação entre os módulos entre os módulos se dá nesse sentido.

Considerando que os recursos necessários para alocação de um determinado módulo é um espaço representado nas figuras por um retângulo. Na figura 4.4 é apresentado um caso em que uma *cloudlet* (cloudlet-0-0) de menor nível está totalmente ocupada e há então a necessidade de alocação de dois novos módulos.

A figura 4.5(a) apresenta uma possibilidade de resposta do algoritmo *Delay-Priority*

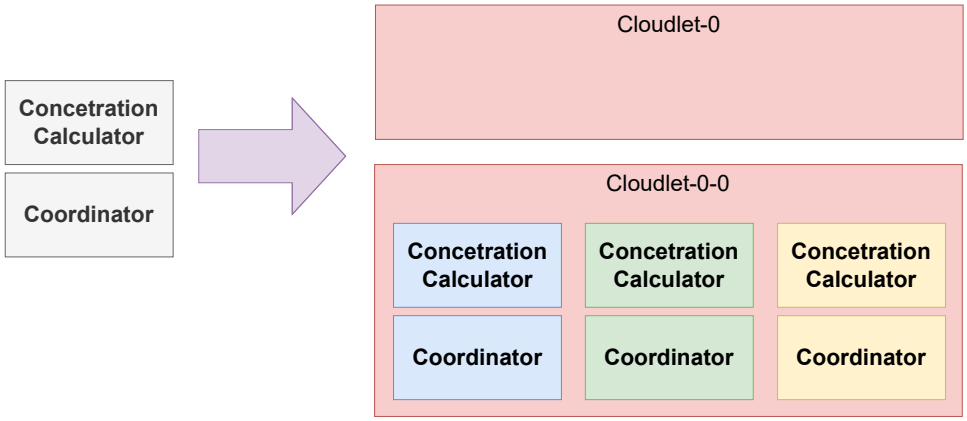
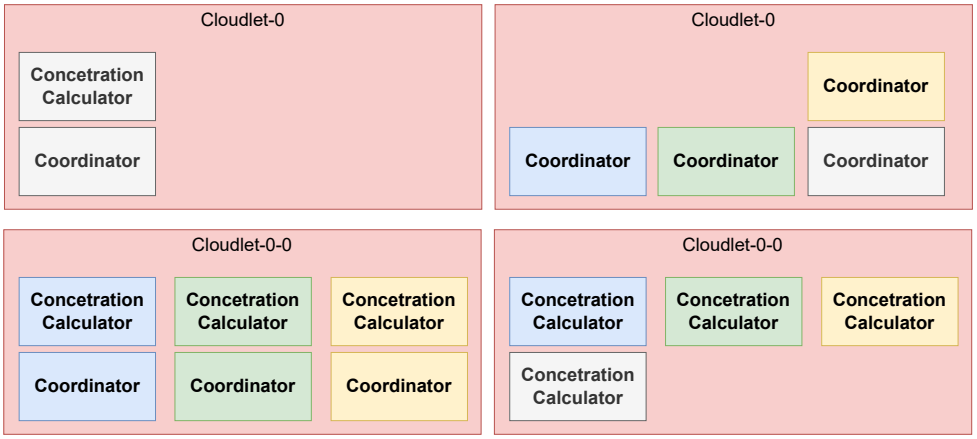
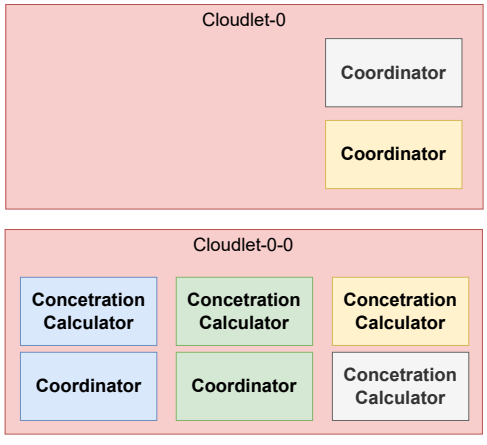


Figura 4.4 Exemplo de Requisição.



(a) DP-I (b) CB-E



(c) LI-X

Figura 4.5 Exemplo de Resultado.

& Individual (*DP-I*). Que após a chegada de novos módulos, caso não haja espaço suficiente para alocação em menor nível, os novos módulos são então encaminhados para

as camadas superiores da *fog computing*, neste caso a *Cloudlet-0*.

O resultado obtido pelo algoritmo *Communication Based & Edgewards* (*CB-E*) é ilustrado na figura 4.5(b). Neste caso, o algoritmo ao analisar os módulos presentes, identificou que a elevação do módulo *Coordinator* seria a melhor escolha, visto que a elevação deste módulo causaria o menor impacto de comunicação quanto ao tráfego de dados executado. Então, eleva todos os módulos desse tipo para o nível acima da *fog computing*.

Já a figura 4.5(c), apresenta o resultado obtido quando os módulos são alocados utilizando o algoritmo LI-X. Neste caso, apesar do módulo *Coordinator* ser identificado como o módulo que causaria o menor impacto quando elevado, o algoritmo identifica que não há necessidade da elevação de todos os módulos desse tipo, e eleva apenas a quantidade necessária para que seja possível a alocação dos novos módulos, inclusive, escolhe alocar o módulos *Coordinator* que acabar de chegar já na camada superior.

4.4 CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentado o *LI-X*, uma proposta para alocação distribuída de aplicações modulares em uma arquitetura de *fog computing*. Foi apresentada a visão geral do ambiente, os componentes essenciais para o funcionamento da *cloudlet* e os detalhes do funcionamento do algoritmo proposto. Além disso, uma ilustração do resultado esperado foi apresentado por meio de um exemplo. No capítulo seguinte será apresentado o ambiente de experimentação e as devidas configurações e ajustes realizadas a avaliação da proposta deste trabalho.

Capítulo 5

DEFINIÇÃO E CONFIGURAÇÃO DO AMBIENTE DE EXPERIMENTAÇÃO

Neste capítulo serão descritos os passos metodológicos executados para a definição e configuração do ambiente de execução dos experimentos.

A fim avaliar e comparar a proposta apresentada nesse trabalho, foi necessária a definição e configuração de um ambiente de simulação. Para o ambiente de simulação foi utilizado o *iFogSim*, simulador amplamente utilizado em outras pesquisas. De forma a configurar o simulador, foi necessária a definição e configuração das aplicações e das diferentes topologias de *fog computing*, a seguir, serão apresentados todos os aspectos essenciais para configuração do simulador.

5.1 APLICAÇÕES

As aplicações a serem executadas no simulador são determinantes para a avaliação dos resultados. Neste trabalho foram utilizadas duas aplicações, uma sensível ao atraso e outra aplicação que produz um alto tráfego de dados, mas que não tem requisito de tempo de resposta muito apertado. Nas subseções seguintes serão apresentados detalhes de cada uma.

5.1.1 EEG Tractor Beam Game

O *EEG Tractor Beam Game* trata-se de um jogo multijogador baseado em *Brain Computer Interface* (BCI) em que o usuário utiliza, com apoio de leitores de ondas cerebrais, o cérebro para interagir com o jogo, além disso é uma aplicação do tipo *latency-critical*, ou seja, deve operar com limites bem reduzidos de atrasos, próximo ao tempo real. Para jogar, os jogadores devem utilizar um headset Eletroencefalograma (EEG) conectado com seus *smartphones* (Figura 5.1). O objetivo é coletar alguns itens utilizando a concentração, quanto mais concentrado o jogador estiver, mais vai atrair os itens para perto de si (ZAO et al., 2014; GUPTA et al., 2017; PEIXOTO; GENEZ; BITTENCOURT, 2022; BITTENCOURT et al., 2017).

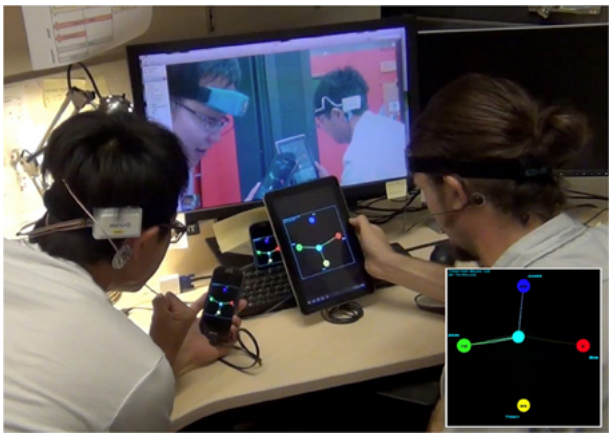


Figura 5.1 Jogadores em uma partida do EEG Tractor Beam (ZAO et al., 2014).

Processamento rápido e tempos de resposta curtos são essenciais para atender aos requisitos do jogo. O poder computacional fornecido e a redução do atraso devido a proximidade com o usuário torna a *fog* uma alternativa capaz de atender a esses requisitos, Gupta et al. (2017) modelou o *EEG Tractor Beam Game* no *iFogSim*, a Figura 5.2 apresenta um grafo com a representação dos módulos e as dependências existentes entre eles, Bittencourt et al. (2017), Charântola et al. (2019) e Peixoto, Genez e Bittencourt (2022) utilizaram em seus trabalhos e por isso foi escolhido para ser utilizado neste trabalho.

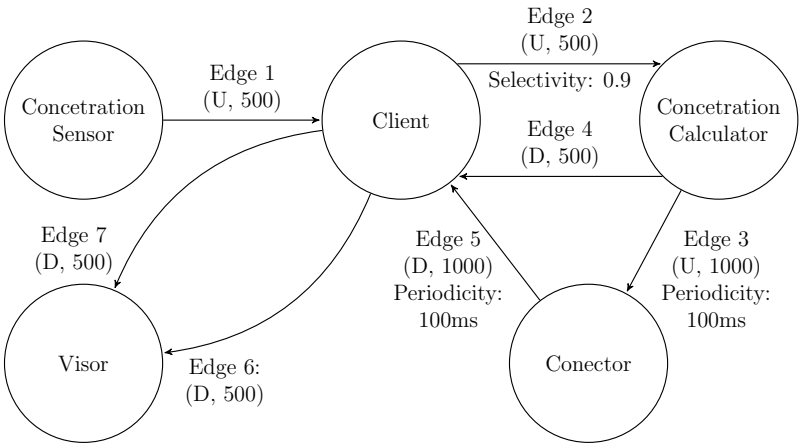


Figura 5.2 *EEG Tractor Beam Game* e seus Módulos

O jogo é composto por três módulos principais que executam o processamento: o *Client*, o *Concetration Calculator* e o *Coordinator*. No *iFogSim*, cada um dos módulos foi modelado utilizando a classe *AppModule* utilizando os requisitos de processamento descritos na Tabela 5.1, também utilizadas nos trabalhos de Bittencourt et al. (2017), Charântola et al. (2019) e Peixoto, Genez e Bittencourt (2022). As dependências entre os módulos foram modeladas com a classe *AppEdge*, utilizando as definições de periodicidade, seletividade e de tamanho dos dados conforme exposta no grafo da Figura 5.2. Por

exemplo, uma tupla de tamanho 1 kbyte é enviada a partir do *concentration calculator* para o *connector* a cada 100 ms ou ainda, para cada tupla recebida pelo *concentration calculator*, uma tupla de tamanho 0,5 kbyte é enviada para o *cliente*.

EEG Bean Tractor Game			VSOT			
client	concentration calculator	coordinator	object detector	motion detector	object tracker	user intreface
200	350	100	550	550	300	200

Tabela 5.1 Recursos estimados em MIPS para cada módulo das aplicações *fog*.

Como visto, a classe *AppEdge* (Figura: 2.5) é utilizada para definir a dependência entre os módulos, para cada *edge* apresentada no grafo da Figura 5.2 um *AppEdge* foi criado utilizando os parâmetros definidos na Tabela 5.2. Para essa aplicação, configuramos um único elemento do tipo *AppLoop* para mensuramos o tempo total da saída de uma unidade de dado produzida no *sensor* até o retorno para o *visor*, definida pela sequência:

concetration sensor → *client* → *concentration calculator* → *client* → ***visor***

Como pode ser visto na Tabela 5.2, um dos parâmetros do *AppEdge* é a quantidade de recursos necessária para processamento do dado. Esse valor no *iFogSim* é estaticamente definido e não sofria nenhuma variação durante a simulação. A classe foi então alterada para que esse valor fosse tratado utilizando uma distribuição uniforme, sofrendo uma variação de 10% para mais ou para menos. A ideia foi produzir um ambiente um pouco mais dinâmico para as simulações.

Para auxiliar no processo de modelagem das aplicações e permitir separar melhor as responsabilidades da criação das aplicações no simulador, foi desenvolvido o conjunto de classes do diagrama *Unified Modeling Language* (*UML*) apresentado na Figura 5.3. A classe *ApplicationBuilder* é uma classe abstrata que define dois métodos abstratos, *createApplication* em que deve ser definida toda a estrutura lógica da aplicação, e o método *createEdgeDevices* que deve criar os sensores e atuadores e associá-los ao *FogDevices* sobrejacentes. A classe *VRGameApplication* foi implementada a partir da classe

Edge	Tipo da Tupla	Tamanho de CPU (MIPS)	Tamanho Dado (bytes)
Edge 1	EEG	3000	500
Edge 2	SENSOR	3500	500
Edge 3	PLAYER_GAME_STATE	1000	1000
Edge 4	CONCETRATION	14	500
Edge 5	GLOBAL_GAME_STATE	1000	1000
Edge 6	GLOBAL_STATE_UPDATE	1000	500
Edge 7	SELF_STATE_UPDATE	1000	500

Tabela 5.2 Configurações de dependência entre os módulos do EEG Tractor Game.

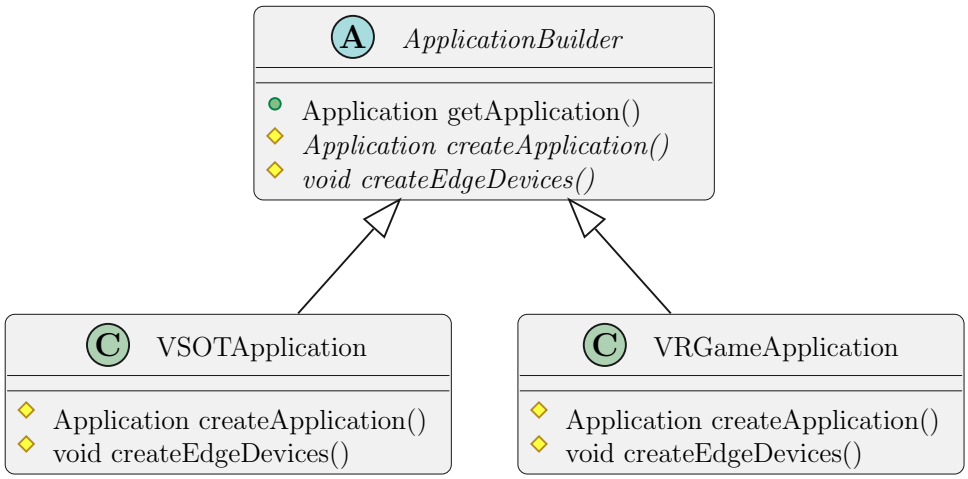


Figura 5.3 Classes auxiliares para a criação das aplicações.

ApplicationBuilder definindo toda a estrutura lógica da aplicação *EEG Tractor Beam Game*.

5.1.2 Video Surveillance Object Tracking (VSOT)

Os sistemas de vigilância por vídeo são tradicionalmente projetados para armazenar o fluxo de dados em mídias que posteriormente podem ser analisados por humanos. Contudo, analisar horas e horas de vídeos de vigilância de diversas câmeras não é uma tarefa trivial. Diferentes propostas de sistemas de videomonitoramento inteligente foram apresentadas nos últimos anos, sistemas modernos são capazes de detectar e rastrear objetos, detectar ações, efetuar descrições de cenas entre outras funcionalidades (Figura 5.4, esses sistemas conseguem reduzir ou mesmo eliminar o trabalho humano de análise de vídeos. Entretanto, executar essa análise de forma automática requer recursos computacionais, além de capacidade de processamento e de armazenamento, fluxos de vídeo podem produzir uma alto tráfego de dados, a *cloud* pode fornecer recursos para o processamento dos dados, contudo levar todo o fluxo de câmeras de vídeo para a *cloud* pode elevar a sobrecarga da rede e também elevar também o custo do transporte dessa informação, por outro lado, utilizar recursos da *fog* tem potencial de reduzir o tráfego de dados na rede e pode proporcionar melhores tempos de resposta (GUPTA et al., 2017; XU et al., 2018; GAWANDE; HAJARI; GOLHAR, 2020).

Gupta et al. (2017) apresenta o modelo de um sistema de videomonitoramento baseado em componentes (Figura: 5.5). O sistema proposto é composto por sensores, atuadores e cinco módulos que fazem a tarefa de detectar e rastrear movimento de objetos, além de fornecer uma interface para o usuário: *Motion Detector*,*Object Detector*, *Object Tracker*, *PTZ Control* e *User Interface*. O mesmo sistema também foi utilizado nos trabalhos de Bittencourt et al. (2017), Charântola et al. (2019) e Peixoto, Genez e Bittencourt (2022).

O *Motion Detector* é um módulo que funciona internamente as câmeras que ao detectar algum movimento envia o fluxo de vídeo para ser analisado pelo módulo *Object Detector*. O módulo *Object Detector* ao recebe o vídeo, compara com imagens anterio-

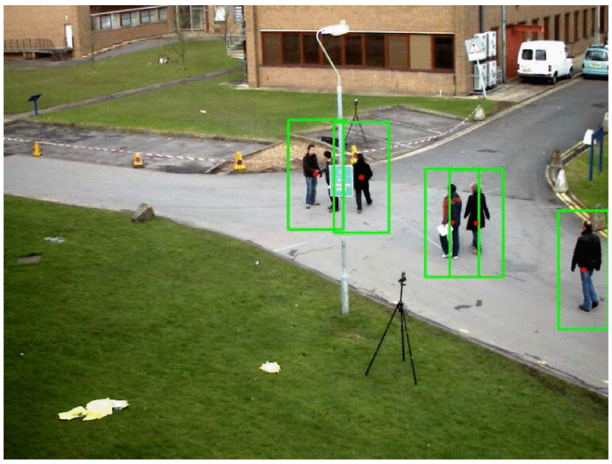


Figura 5.4 Exemplo de aplicação de videomonitoramento com detecção e rastreamento de objetos (XU et al., 2018).

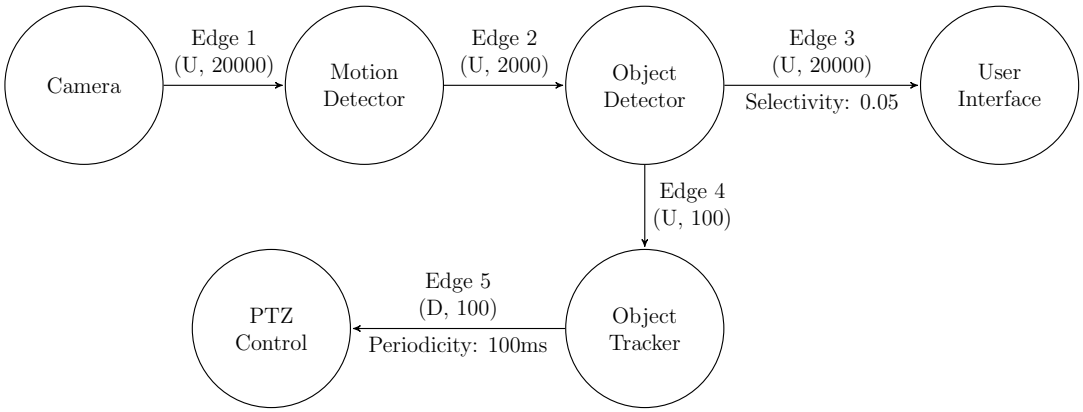


Figura 5.5 Aplicação de videomonitoramento e seus Módulos

res e verifica movimentos de um determinado objeto, caso seja detectado, encaminha a requisição para o módulo *Object Tracker* que é responsável por calcular as coordenadas do objeto, a direção em que se movimenta e calcular o novo posicionamento da câmera para que possa continuar a seguir o objeto. Após efetuar os devidos cálculos o novo posicionamento da câmera é enviado para o módulo *PTZ Control* que é um módulo que também funciona internamente na câmera e irá acionar atuadores que reposicionarão a câmera com os novos parâmetros. O módulo *User Interface* vai receber trechos de vídeos relevantes identificados pelo sistema e fornecerá uma interface de acesso para usuários (GUPTA et al., 2017).

Os módulos foram configurados no *iFogSim* com uso da classe *AppModule* (Figura: 2.5), os recursos necessários para cada módulo foi ajustado de acordo a Tabela 5.1, assim como na aplicação anterior, os valores de tamanho de CPU, varia em 10% para mais ou para menos utilizando uma distribuição uniforme. As dependências entre os módulos foram modeladas com a classe *AppEdge* utilizando os parâmetros apresentados na Tabela 5.3. A aplicação como um todo, foi configurada a partir da implementação da classe

Edge	Tipo da Tupla	Tamanho de CPU (MIPS)	Tamanho Dado (bytes)
Edge 1	RAW VIDEO STREAM	1000	20000
Edge 2	MOTION VIDEO STREAM	2000	2000
Edge 3	DETECTED OBJECT	500	2000
Edge 4	OBJECT LOCATION	1000	100
Edge 5	PTZ PARAMS	100	100

Tabela 5.3 Configurações de dependência entre os módulos da aplicação *VSOT*.

VSOTApplication (Figura 5.3). Para avaliar o atraso na comunicação entre os módulos, o *AppLoop* foi configurado para avaliar o tempo total entre a saída de um fluxo de vídeo da câmera até a solicitação de movimentação da câmera no *PTZ Control*:

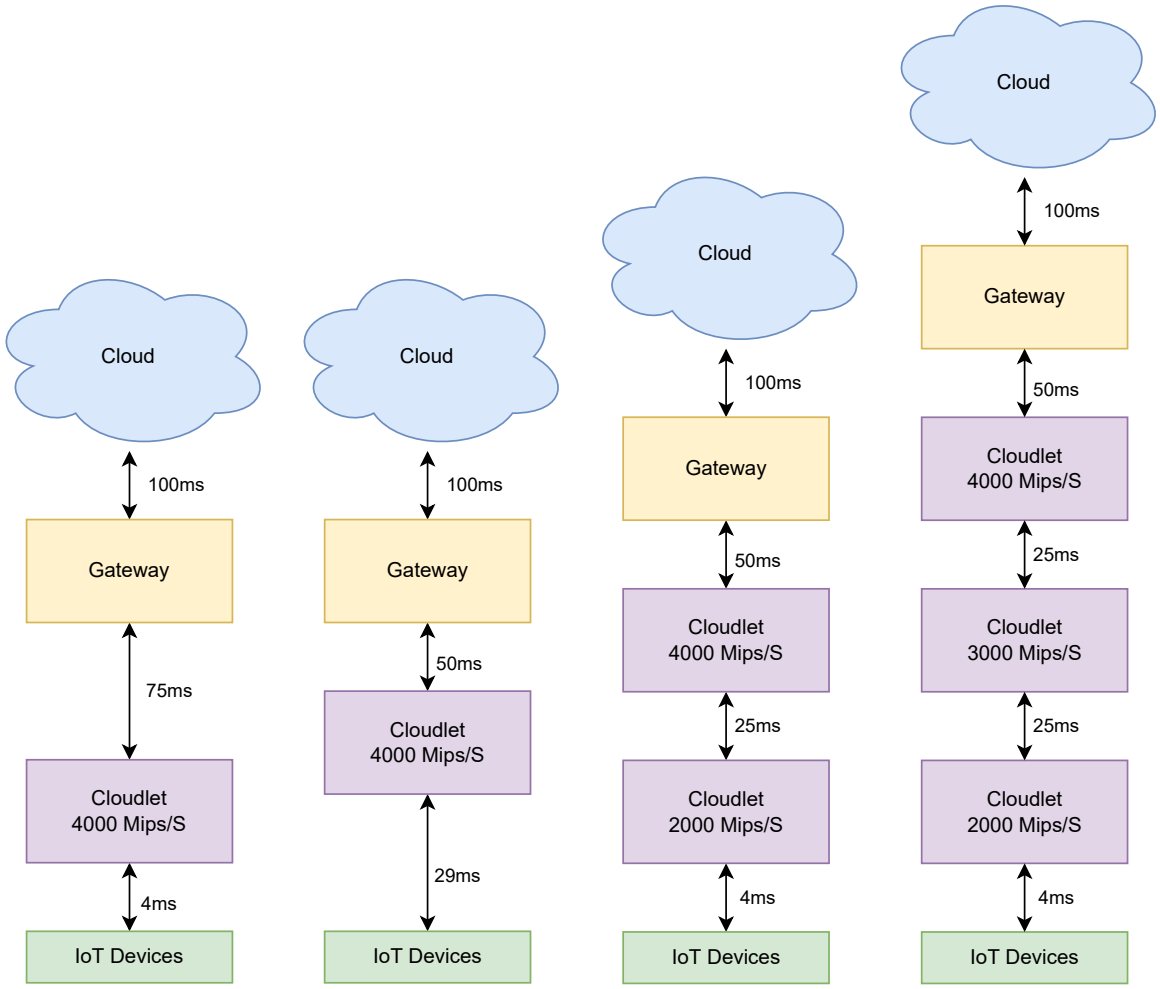
camera → *motion detector* → *object tracker* → ***PTZ Control***

5.2 CONFIGURAÇÃO DO AMBIENTE

Para a execução dos experimentos, foram definidos quatro diferentes topologias de *fog* (Figura: 5.6). Todas as topologias têm em sua camada superior a *cloud*, seguida por um *Gateway* cuja comunicação sofre de um atraso de 100ms. Além disso, todas as topologias avaliadas são compostas por uma ou mais camadas de *cloudlets*, tendo ao final a camada de dispositivos *Internet of Things* (IoT). A comunicação entre cada uma das camadas sofre com atrasos em algum nível. Nos experimentos é suposto que a *cloud* tem poder computacional suficiente para rodar todas as aplicações de uma só vez.

Na topologia do Tipo A (Figura: 5.6(a)) o *Gateway* é seguido por uma *cloudlet* com poder computacional de 4000 mips, com atraso de 75 ms e por fim a camada de dispositivos da IoT que sofrem com o atraso de 4 ms. Na topologia do Tipo B (Figura: 5.6(b)) o *Gateway* é seguido por uma *cloudlet* com poder computacional de 4000 mips, com atraso de 50 ms, seguida pela camada de dispositivos IoT com atraso de 29 ms. Na topologia do Tipo C (Figura: 5.6(c)) o *Gateway* é seguido por uma *cloudlet* com poder computacional de 4000 mips, com atraso de 50 ms, seguida por uma outra *cloudlet* com poder computacional de 2000 mips com atraso de 25 ms que por fim é seguida pela camada de dispositivos com atraso de 4 ms. Na topologia do Tipo D (Figura: 5.6(d)) o *Gateway* seguido é por uma *cloudlet* com poder computacional de 4000 mips, com atraso de 50 ms, seguida por uma outra *cloudlet* com poder computacional de 3000 mips, com atraso de 25 ms que por sua vez é seguida por uma última *cloudlet* com poder computacional de 2000 mips, com atraso de 25 ms, que por fim é seguida pela camada de dispositivos com atraso de 4 ms.

As topologias do Tipo A (Figura: 5.6(a)), Tipo B (Figura: 5.6(b)) e Tipo C (Figura: 5.6(c)) são as mesmas utilizadas nos trabalhos de Charântola et al. (2019),(PEIXOTO; GENEZ; BITTENCOURT, 2022) e (BITTENCOURT et al., 2017). A topologia do Tipo D é uma variação com 3 camadas de *cloudlet* proposta por este trabalho. A utilização de



(a) Tipo A (b) Tipo B (c) Tipo C (d) Tipo D

Figura 5.6 Topologias *fog*.

diferentes topologias neste trabalho objetiva avaliar o desempenho de diferentes métodos de alocação das aplicações em cenários diferentes.

Para todas as topologias, na camada de IoT *Devices* (Figura: 5.6) foram adicionados os dispositivos para quatro instâncias da aplicações de videomonitoramento (Figura: 5.5). Quanto a aplicação VRGame (Figura: 5.2) foram criados 16 cenários diferentes com objetivo de simular a mobilidade dos usuários do jogo. Nesse sentido foram avaliados cenários simulando desde a ausência de jogadores até a chegada de 15 jogadores nas proximidades de *cloudlet* (Figura: 5.7).

Os módulos da Câmera, Motion Detector e PTZ control da aplicação de videomonitoramento sempre serão alocados na própria Câmera e o módulo *User Interface* sempre será alocado na *cloud* (Figura: 5.8). Os demais módulos serão alocados na *cloud* ou *fog* de acordo com a decisão do algoritmo.

Os módulos do EEG, Display e Client da aplicação VRGame sempre estarão alocados

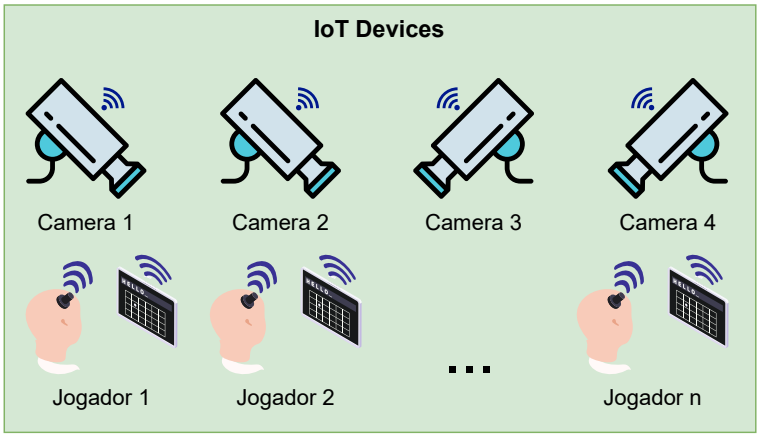


Figura 5.7 Camada de Dispositivos IoT.

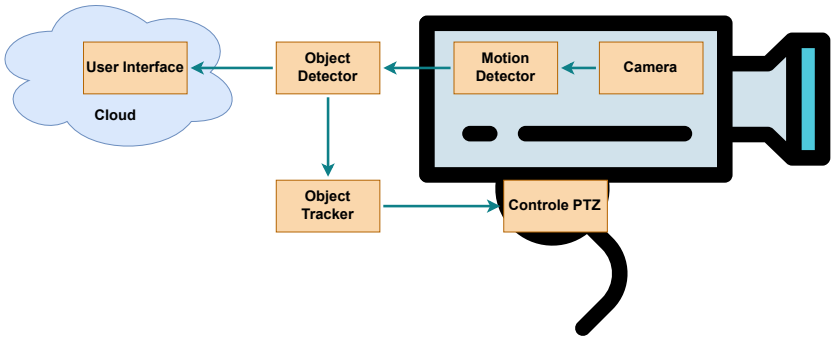


Figura 5.8 Módulos videomonitoramento.

junto ao Jogador. Os demais módulos serão alocados na *fog* ou *cloud* (Figura: 5.9).

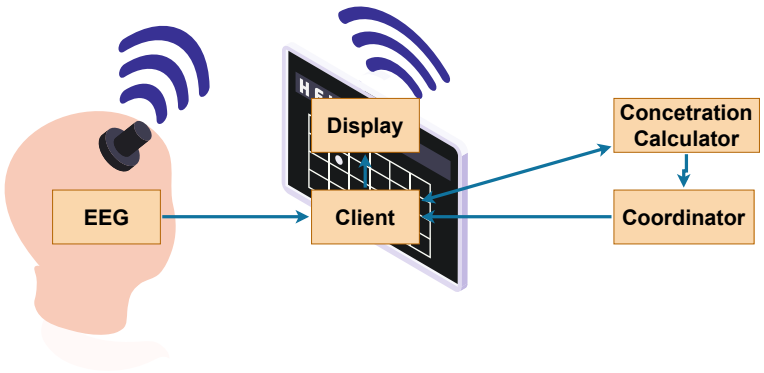


Figura 5.9 Módulos EEG VRGame.

Para modelar os cenários no *iFogSim*, cada *cloudlet* foi modelada com a classe *FogDevice*, sensores e atuadores foram modelados respectivamente com o uso das classes *Sensor*

e *Actuator* (Figura: 2.5). Para facilitar a definição das topologias, foi desenvolvido um conjunto de classes para tornar o processo mais simples. Para definir cada nível da *fog* e a relação entre os níveis foi criada a classe *Level* (Figura: 5.10), cada instância dessa classe recebe informações a respeito da quantidade de *cloudlets* presentes no nível, informações sobre os recursos disponíveis em cada uma, o atraso para a camada superior e também pode receber o próximo nível, para compor dessa forma, uma estrutura hierárquica. Um exemplo do uso da classe *Level* pode ser visto na 5.11.

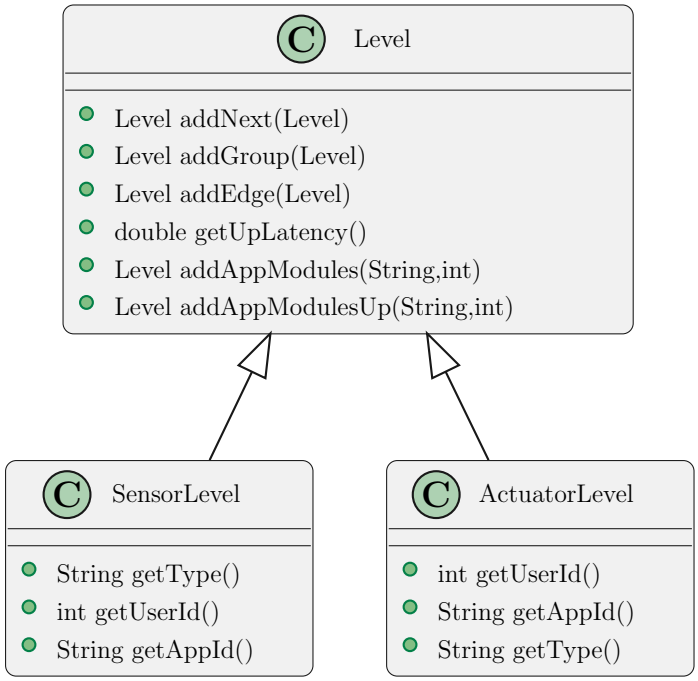


Figura 5.10 Classes auxiliares para a criação de cada camada da topologia.

A classe abstrata *Topology* (Figura 5.12) for desenvolvida com objetivo de facilitar a criação de diferentes topologias, generalizando o processo de configuração do simulador restando ao desenvolvedor declarar com uso da classe *Level* os níveis da *fog*. Para este trabalho, para cada uma das topologias apresentadas na Figura 5.6, foi criada uma especialização da classe *Topology* conforme diagrama apresentado na Figura 5.12. A Figura 5.11 apresenta o código da topologia do Tipo D, composta pela *cloud* em sua camada superior, seguidas de um *proxy* sem poder de processamento relevante, afastado da *fog* por um atraso de 100 ms, seguida de uma camada de *fog* com poder computacional de 4000 mips, afastado do *proxy* por 25 ms de atraso, seguida por uma segunda camada de *fog* com poder computacional de 3000 mips, afastada da camada anterior por 25 ms de atraso e por fim, seguida por mais uma camada com dispositivo com poder computacional de 2000 mips, afastada da camada anterior por 25 ms de atraso. Além disso, foi definido que os dispositivos de borda sofreriam um atraso de 4 ms em relação a essa última camada.

A responsabilidade de anexar os dispositivos IoT de cada instância das aplicações na topologia desenvolvida ficou a cargo das mesmas classes responsáveis pela criação das aplicações (Figura 5.3), neste caso o método *createEdgeDevices()* das subclasses da *Ap-*

```java
public class TopologyD extends Topology {

  private void configNetwork(){
    Level top = new Level(1, new HostConfig(44800, 40000, 10000, 10000, 0.011), "cloud",0);
      Level bottom = top
        .addNext(new Level(1, new HostConfig(1, 4000, 10000, 10000, 0.0), "proxy-server",100))
        .addNext(new Level(3, new HostConfig(4000, 4000, 10000, 10000,0.0), "cloudlet",25))
        .addNext(new Level(1, new HostConfig(3000, 4000, 10000, 10000,0.0), "cloudlet",25))
        .addNext(new Level(1, new HostConfig(2000, 4000, 10000, 10000,0.0), "cloudlet",25));
      setTop(top);
      setBottom(bottom);
  }

  public TopologyD() {
    super("D",4);
      configNetwork();
  }
```

Figura 5.11 Topologia C

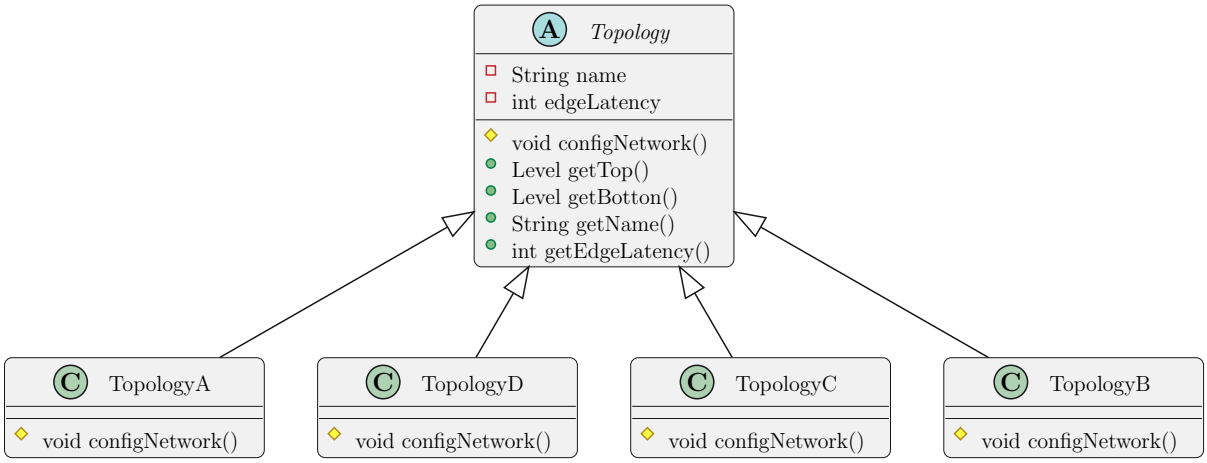


Figura 5.12 Classes auxiliares para a criação de topologias.

plicationBuilder deve adicionar os dispositivos necessários para a simulação da aplicação. A Figura 5.13 apresenta o código do método *createEdgeDevices()* da classe *VSOTApplication*, responsável pela configuração da aplicação *VSOT*. Neste método foi adicionado após a camada mais baixa da *fog* um dispositivo de borda com poder computacional de 1000 mips, um sensor do tipo *CAMERA* que simula a emissão de fluxos de dados de vídeo e o atuador que recebe dados do tipo *PTZ_CONTROL*, que carrega os dados para o redirecionamento da câmera.

Para a configuração e execução da simulação, a classe *Experimento* (Figura 5.14) foi desenvolvida com a responsabilidade de configurar de gerar as classes do *iFogSim* a partir das declarações de topologias e aplicações, antes de iniciar a simulação, as aplicações são anexadas as topologias e a classe *MakeFog* é acionada para criar as classes e configurar a topologia de acordo as classes do *iFogSim*. Na sequência a simulação é iniciada. Além

```java
protected void createEdgeDevices() {
    this.topology.getBotton().addEdge(
        new Level(this.dist, new HostConfig(1000, 10000, 10000, 270, 0), "m-DCNS", topology.getEdgeLatency())
            .addAppModulesUp("object_detector", 1)
            .addAppModulesUp("object_tracker", 1)
            .addAppModules("motion_detector", 1)
            .addGroup(
                new SensorLevel(1, "s-m-DCNS", "CAMERA",
                    new NormalDistribution(DCNS_TRANSMISSION_TIME, SD_DCNS_TRANSMISSION_TIME, SEED), 1,
                    fogBroker.getId(), appId),
                new ActuatorLevel(1, "ptz-m-DCNS", "PTZ_CONTROL", 1, fogBroker.getId(), appId)));
}
```

Figura 5.13 Código do método *createEdgeDevices()* da classe VSOTApplication.

disso, o método *showTopoloy()* pode ser acionado para se obter uma visualização da topologia configurada, a visualização faz uso de um recurso disponível no *iFogSim* para visualização de topologias (Figura 5.15).

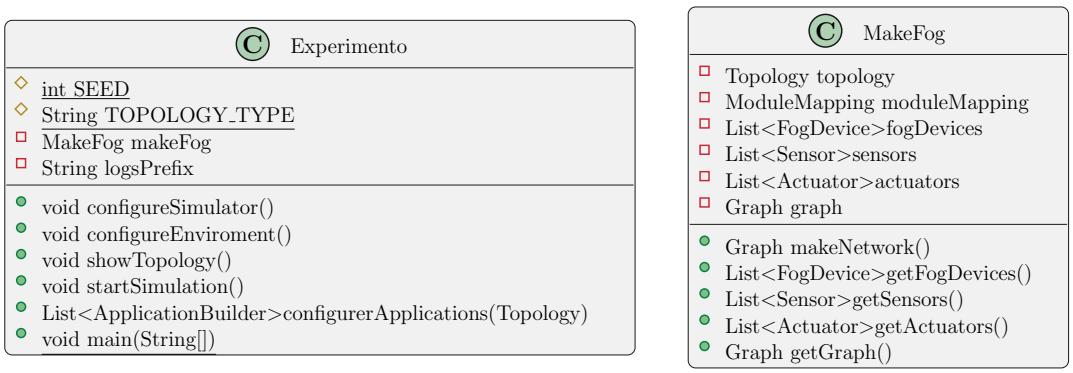


Figura 5.14 Classes auxiliares para configuração e execução do experimento

Para servir de base comparativa para essa proposta, utilizamos o algoritmo *Delay-Priority & Individual* (*DP-I*), proposto por Charântola et al. (2019) e o algoritmo *Communication Based & Edgewards* (*CB-E*), proposto por Peixoto, Genez e Bittencourt (2022).

O *DP-I* é uma política de alocação de recursos que, para definir onde cada aplicação será executada, considera inicialmente as aplicações sensíveis a latência, aloca todos os módulos possíveis dessa aplicação na *cloudlet* de níveis mais baixos e quando não houver mais recursos, começa a alocar os módulos nos próximos níveis, até que, se necessário, utilize a *cloud* (CHARÂNTOLA et al., 2019) .

Por outro lado, *CB-E* é uma política de alocação que, dado um grupo de aplicações modulares, irá priorizar a alocação de aplicações sensíveis a latência nos níveis mais baixos da *fog*, assim como na *DP-I*. Contudo, quando os recursos ficarem escassos, avaliará quais módulos da aplicação poderiam ser realocados para camadas superiores da *fog* e que causariam o menor impacto no tráfego de dados na rede. Ao selecionar tais módulos, o algoritmo envia todos os módulos dos tipos selecionados para as camadas subsequentes da arquitetura (PEIXOTO; GENEZ; BITTENCOURT, 2022).

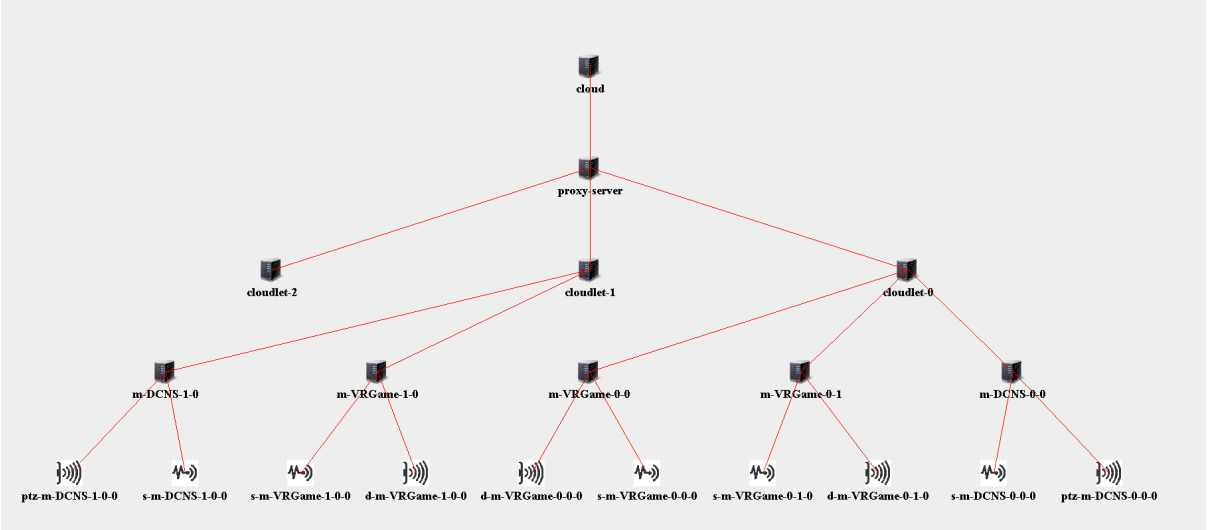


Figura 5.15 Exemplo de topologia gerada.

Para cada combinação de topologia (Figura: 5.6) com cada um dos algoritmos (*CB-E*, *DP-I* e LI-X) e as diferentes quantidade de usuários da aplicação VRGame, foram feitas dez execuções da simulação, variando-se a semente do simulador para cada conjunto. A variação da semente tem influência direta sobre a frequência de geração de requisições e os recursos necessários para o processamento de cada uma.

5.3 CONSIDERAÇÕES FINAIS

Este capítulo apresentou a definição do ambiente de experimentação, começando da definição e configuração de cada uma das aplicações envolvidas, apresentando em seguida as diferentes organizações de topologias de *fog* experimentadas e por fim os parâmetros de execução da simulação. Além disso, alguns aspectos importantes a respeito do simulador foram apresentados. Na próxima seção serão apresentados os resultados dos experimentos bem como a discussão desses resultados.

Capítulo 6

RESULTADOS

Neste capítulo são apresentados os principais resultados obtidos após a realização dos experimentos. Os resultados para cada uma das Topologias apontadas na Figura 5.6 serão apresentadas em seções distintas.

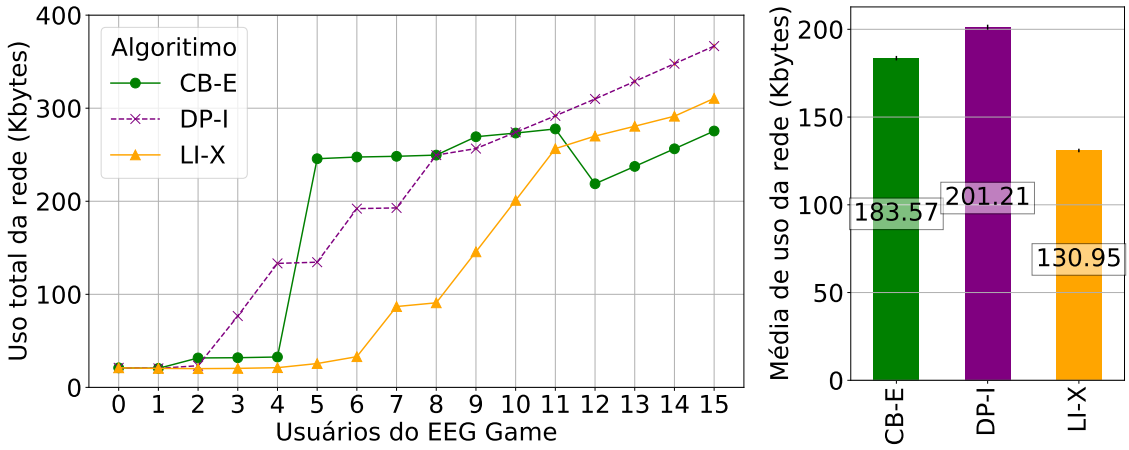
6.1 TOPOLOGIA A

O gráfico da Figura 6.1(a) apresenta o tráfego de rede total de acordo a chegada de novos usuários e a Figura 6.1(b) apresenta a média de tráfego de rede para todas as quantidades de usuários para a Topologia do Tipo A (Figura: 5.6(a)). A Tabela 6.1 apresenta o desvio padrão para o gráfico da Figura 6.1(a) encontrado a partir das médias das replicações do experimento. A partir da análise dos dados é possível perceber uma ligeira vantagem do *Least Impact - X* (LI-X) em comparação aos demais algoritmos, enquanto o LI-X obteve uma média de 130,95 Kbytes de dados trafegados, o *Communication Based & Edgewards* (*CB-E*) teve uma média 183,57 Kbytes e o *Delay-Priority & Individual* (*DP-I*) teve uma média de 201,21 Kbytes. Ainda no na Figura 6.1(a), é possível observar que a partir da chegada do 12º usuário o *CB-E* apresenta um ganho em relação ao LI-X, mas na média do experimento, o LI-X teve um ganho de 28,7% em relação ao *CB-E* e de 34.9% em relação ao *DP-I*.

Users	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CB-E	0,66	0,67	0,67	0,67	0,68	1,67	1,69	1,70	1,72	1,72	1,74	1,76	1,53	1,65	1,74	1,66
DP-I	0,66	0,67	0,68	1,04	1,23	1,22	1,53	1,53	1,72	1,72	1,77	1,98	1,94	2,21	2,44	2,50
LI-X	0,66	0,67	0,67	0,67	0,67	0,68	0,67	1,06	1,09	1,27	1,58	1,75	1,78	1,80	1,79	1,97

Tabela 6.1 Desvio Padrão: Uso de Rede - Topologia A

Quanto ao atraso das aplicações no cenário da Topologia A (Figura: 6.2), foi observado que na aplicação VRGame o LI-X conseguiu um tempo médio de atraso de 30,22 ms, enquanto o *CB-E* obteve um tempo médio de atraso de 117,87 ms e o *DP-I* conseguiu um atraso médio de 26,93 ms. Observando o gráfico do atraso em função da quantidade de usuários, é possível notar um comportamento parecido entre os algoritmos até o 9º



(a) Usuários Movidos (b) Média

Figura 6.1 Uso da Rede: Topologia Tipo A

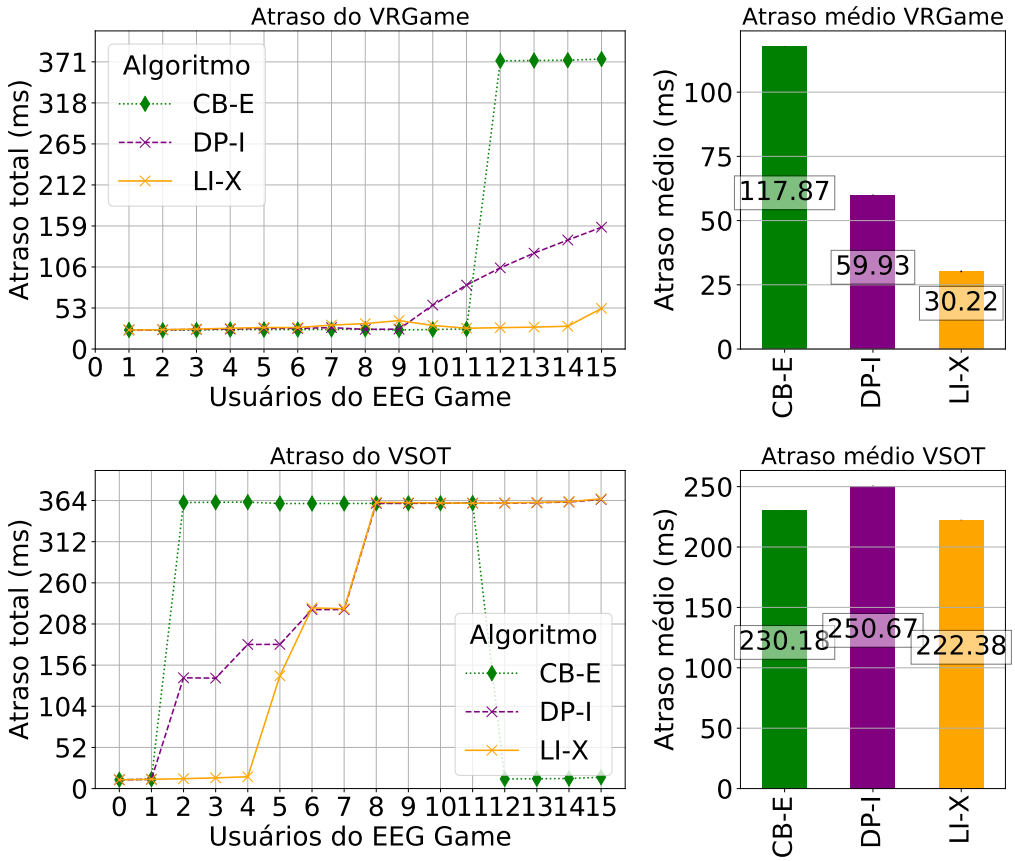


Figura 6.2 Atraso das Aplicações: Topologia Tipo A

usuário. A partir do 12º usuário o *CB-E* começa a destoar dos demais algoritmos. Ao

	Users	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CB-E	VSOT	0,0	0,1	0,0	0,1	0,0	0,0	0,0	0,0	0,0	0,1	0,1	0,1	0,1	0,1	0,2	0,4
	VRGame	0,0	0,3	0,3	0,2	0,1	0,1	0,1	0,1	0,1	0,2	0,2	0,1	0,1	0,1	0,1	0,4
DP-I	VSOT	0,0	0,1	0,4	0,4	0,6	0,6	0,5	0,5	0,1	0,0	0,1	0,1	0,1	0,2	0,2	1,1
	VRGame	0,0	0,3	0,3	0,2	0,2	0,2	0,2	0,2	0,1	0,1	0,5	0,8	0,9	0,6	0,9	1,5
LI-X	VSOT	0,0	0,1	0,2	0,2	0,3	0,5	0,5	0,5	0,1	0,1	0,1	0,1	0,1	0,3	0,3	1,5
	VRGame	0,0	0,3	0,4	0,2	0,2	0,2	0,2	0,5	1,1	1,8	0,3	0,2	0,1	0,2	0,1	1,5

Tabela 6.2 Desvio Padrão: Atraso - Topologia A.

avaliar o comportamento da aplicação VSOT percebe-se que o LI-X também obteve um resultado melhor, apresentando um tempo médio de atraso de 222,38 ms, *CB-E* apresentou o segundo melhor resultado com o tempo de atraso de 230,18 ms e por último o *DP-I* que teve um tempo médio de resposta de 250,67 ms, com isso, o LI-X conseguiu ser melhor que o *CB-E* 3,39% e que o *DP-I* em 11,29%. A Tabela 6.2 apresenta os desvios padrões das médias das replicações do experimento para os gráficos da Figura 6.2.

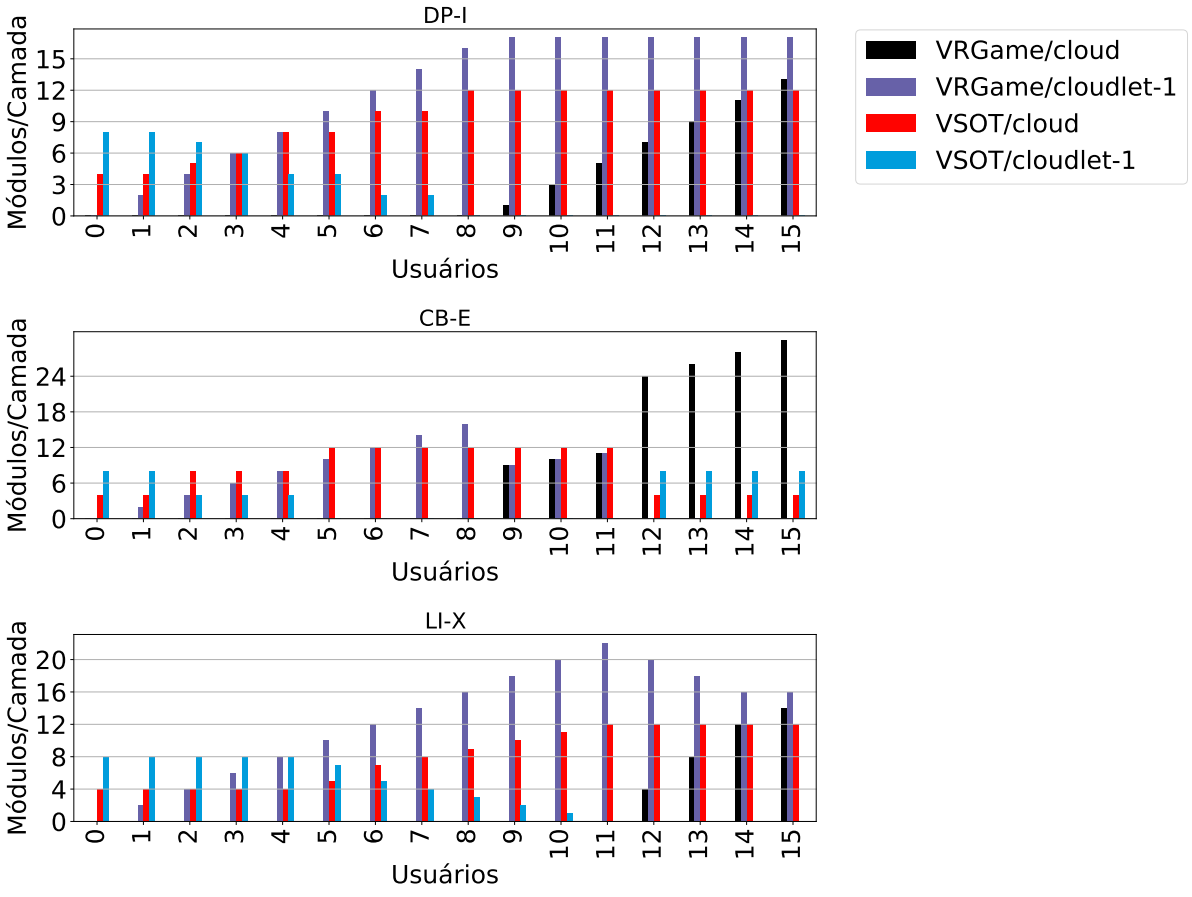


Figura 6.3 Contagem de módulos por dispositivo: Topologia Tipo A

A fim de entender os motivos do comportamento apresentado acima, a Figura 6.3 apresenta a distribuição dos módulos das aplicações entre os dispositivos. Como apresen-

tado na Figura 5.6, a Topologia A é composta de apenas uma camada de *fog*, representada na Figura 6.3 pela *cloudlet-1* e pela *cloud* como camada superior. No gráfico é possível notar o motivo pela qual a média de atraso da aplicação VRGame ser a maior para o *CB-E*, a partir da chegada do 12º usuário, todos os módulos do VRGame são alocados na *cloud*, com isso, o atraso tende a ser maior por conta da distância entre os dispositivos. Por outro lado, parte dos módulos da aplicação VSOT foi alocada na *cloudlet-1*, ficando mais próxima do usuário, reduzindo o atraso e também o tráfego total de dados pela rede dessa aplicação, o que também explica o comportamento identificado nos gráficos da Figura 6.1.

Nos gráficos da Figura 6.3, o *DP-I* e o LI-X apresentam um comportamento parecido, principalmente quando em quantidades maiores de usuários. Para entender melhor esse comportamentos a Figura 6.4 apresenta um recorte mais granular da distribuição dos módulos da aplicação VRGame entre os dispositivos a partir da chegada do 10º usuário, nela é possível observar que a partir da chegada do 12º usuário, o *CB-E* aloca todos os módulos do VRGame na *cloud*, o LI-X deu preferência por alocar uma maior quantidade de instâncias do módulo *concentration_calculator* na *cloudlet-1*, módulo este, identificado como potencialmente causador de maior impacto na comunicação quando alocado mais distante do usuário, e alocou o quanto pôde do módulo *connector* com os recursos restantes, as demais instâncias dos módulos foram elevados para *cloud* devido a falta de recursos suficientes na *cloudlet-1*, comportamento este diferente do observado no *DP-I*, que produziu uma distribuição mais igualitária entre os módulos da aplicação.

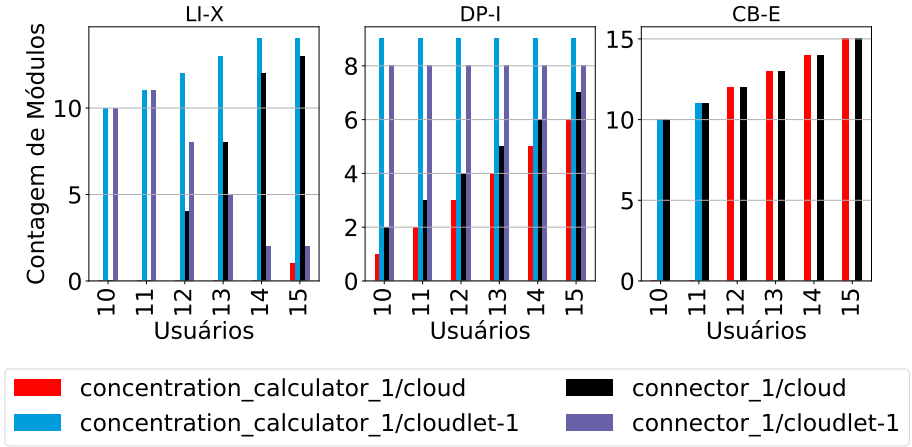
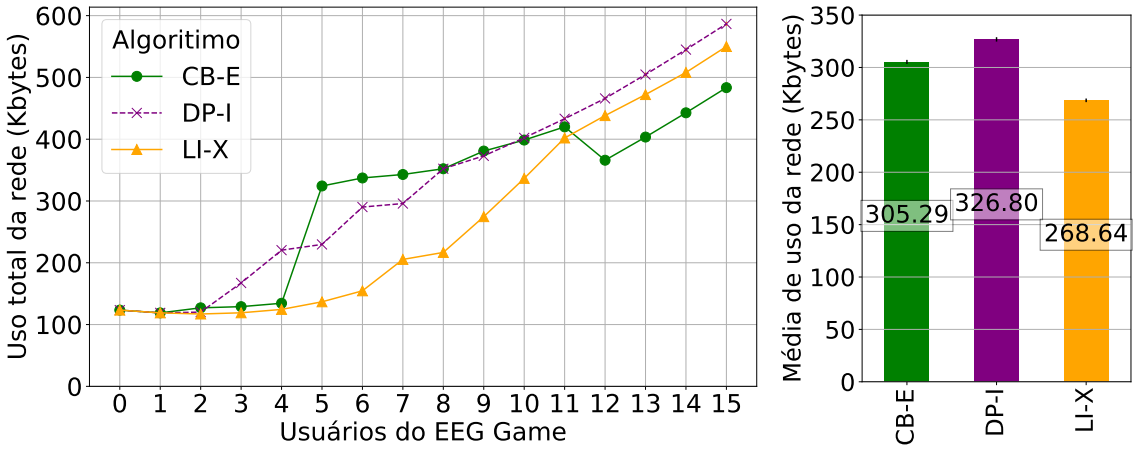


Figura 6.4 Contagem de módulos por dispositivo - VRGame: Topologia Tipo A

6.2 TOPOLOGIA B

Na Figura 6.5(a) é apresentado o tráfego de rede total de acordo a chegada de novos usuários e na Figura 6.5(b) é apresentada a média de tráfego de rede para todas as quantidades de usuários para a Topologia do Tipo B (Figura: 5.6(b)). A Tabela 6.3 apresenta o desvio padrão para o gráfico da Figura 6.5(a) encontrado a partir das médias das replicações do experimento. Nessa Topologia, apesar da diferença ter sido menor, o

LI-X se manteve à frente na média, enquanto o LI-X obteve uma média de 268,64 Kbytes, o *CB-E* teve uma média 305,29 Kbytes e o *DP-I* teve uma média de 326,80 Kbytes. Assim como na Topologia do Tipo A, é possível observar na Figura 6.5(a) que a partir da chegada do 12º usuário o *CB-E* apresenta um ganho em relação ao LI-X, mas na média do experimento, o LI-X teve um ganho de 12,01% em relação ao *CB-E* e de 17,80% em relação ao *DP-I*.



(a) Usuários Movidos (b) Média

Figura 6.5 Uso da Rede: Topologia Tipo B

Users	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CB-E	0,99	0,99	1,07	1,06	1,10	2,15	2,23	2,25	2,33	2,40	2,51	2,67	2,40	2,59	2,81	2,76
DP-I	0,97	1,00	1,05	1,35	1,59	1,63	1,99	2,04	2,33	2,35	2,58	2,80	2,83	3,24	3,40	3,43
LI-X	0,97	1,00	1,03	1,04	1,05	1,06	1,13	1,54	1,63	1,84	2,31	2,64	2,78	2,93	2,93	3,10

Tabela 6.3 Desvio Padrão: Uso de Rede - Topologia B

	Users	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CB-E	VSOT	0,0	0,1	0,1	0,1	0,1	0,0	0,0	0,0	0,0	0,1	0,1	0,2	0,1	0,1	0,2	0,4
	VRGame	0,0	0,3	0,3	0,2	0,2	0,1	0,2	0,1	0,1	0,1	0,1	0,2	0,1	0,1	0,1	0,4
DP-I	VSOT	0,0	0,1	0,3	0,4	0,5	0,5	0,4	0,4	0,0	0,1	0,1	0,1	0,1	0,2	0,2	1,4
	VRGame	0,0	0,3	0,4	0,2	0,2	0,2	0,2	0,2	0,1	0,2	0,5	0,6	0,8	0,5	0,7	1,7
LI-X	VSOT	0,0	0,1	0,1	0,2	0,4	0,3	0,4	0,4	0,1	0,1	0,1	0,1	0,2	0,3	0,3	1,2
	VRGame	0,0	0,3	0,4	0,3	0,3	0,2	0,3	0,5	0,9	1,6	0,5	0,1	0,2	0,2	0,2	1,4

Tabela 6.4 Desvio Padrão: Atraso - Topologia B.

Quanto a avaliação do tempo médio de atraso na Topologia B (Figura: 6.6), para a aplicação VRGame, o LI-X obteve um tempo médio de atraso de 76,61 ms, o *CB-E* obteve uma média de 154,22 ms e o *DP-I* teve uma média de 104,95 ms. Neste cenário o ganho do LI-X em relação ao *CB-E* foi de 48,4% e de 24,2% para o *DP-I*. Quanto a aplicação VSOT o *CB-E* apresentou o tempo médio de atraso de 248,93 ms, o *DP-I* apresentou o tempo médio de 266,59 ms e por fim o LI-X apresentou o tempo médio de atraso de

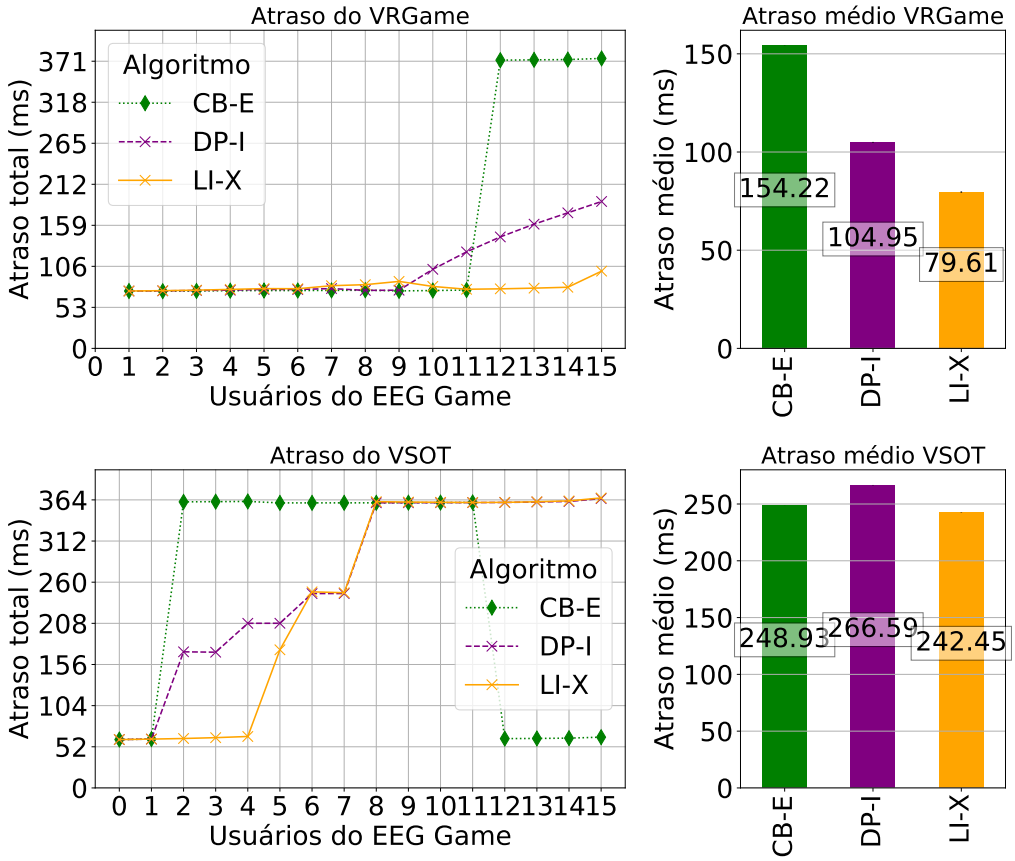


Figura 6.6 Atraso das Aplicações: Topologia Tipo B

242,45 ms o que representa uma ganho de 2.6,7% em relação ao *CB-E* e um ganho de 9,1% em relação ao *DP-I*. Dessa forma o LI-X também conseguiu atingir o objetivo nesse cenário. A Tabela 6.4 apresenta os desvios padrões das médias das replicações do experimento para os gráficos da Figura 6.6.

Para entender um pouco mais desse comportamento, a Figura 6.7 apresenta a distribuição dos módulos entre os dispositivos na Topologia B. A Topologia B se assemelha a Topologia A, ambas contém apenas um nível de *fog* com a mesma quantidade de recursos computacionais, diferenciando-se apenas pelas distâncias entre os dispositivos. Devido a isso, os resultados quanto ao local de alocação de cada instância dos módulos da aplicação também apresentaram semelhanças significativas, o atraso médio apresenta um aumento devido a essa distância adicional entre os dispositivos e o primeiro nível da *fog*. A partir da chegada do 9º usuário, o *CB-E* começa a alocar todos as instâncias dos módulos da aplicação VRGame na *cloud* o que provoca uma aumento significativo no tempo médio de atraso e a partir da chegada do 12º usuário o *CB-E* alocar todos os módulos da aplicação VSOT na *cloudlet-1* o que acaba por reduzir o atraso médio para essa aplicação e também a o tráfego total da rede, visto que o VSOT é a aplicação que produz e trafega mais dados. Assim como na Topologia A, na Topologia B o comportamento do *DP-I* e do LI-X são

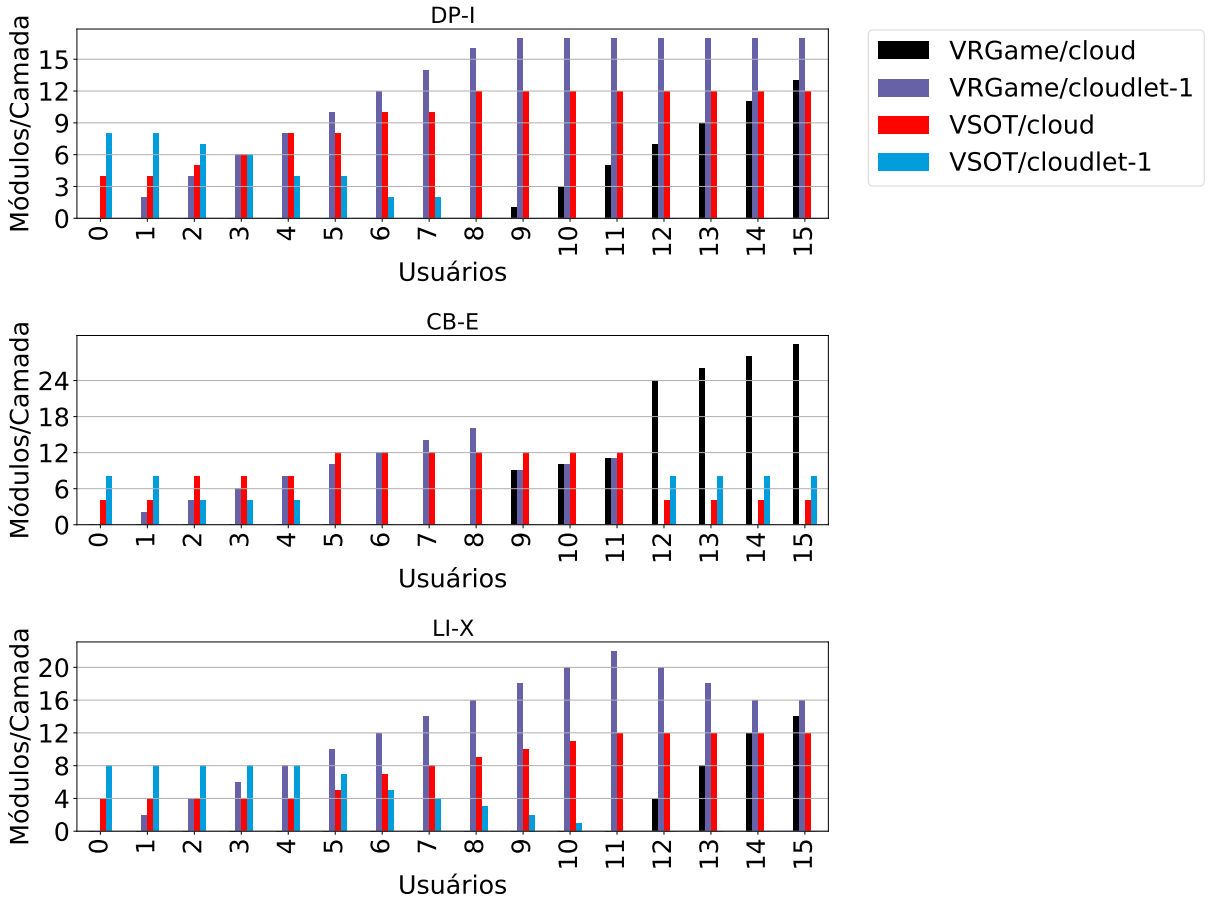


Figura 6.7 Contagem de módulos por dispositivo: Topologia Tipo B

parecidos e uma análise mais detalhada precisa ser apresentada.

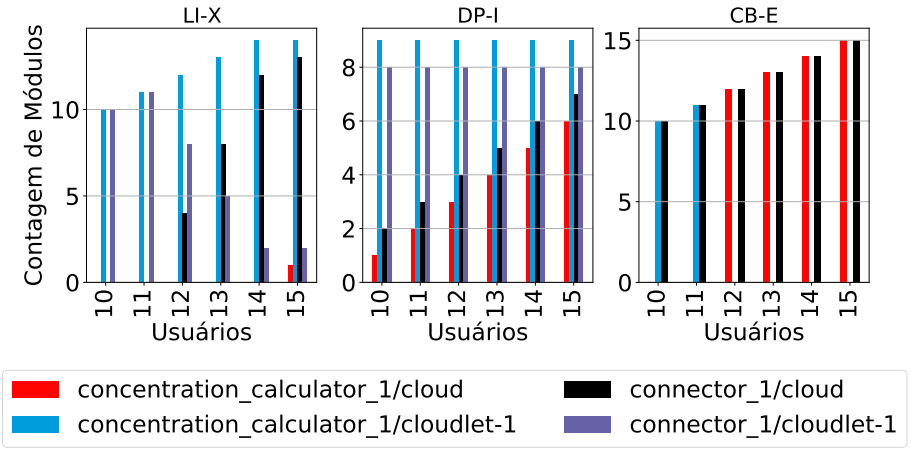
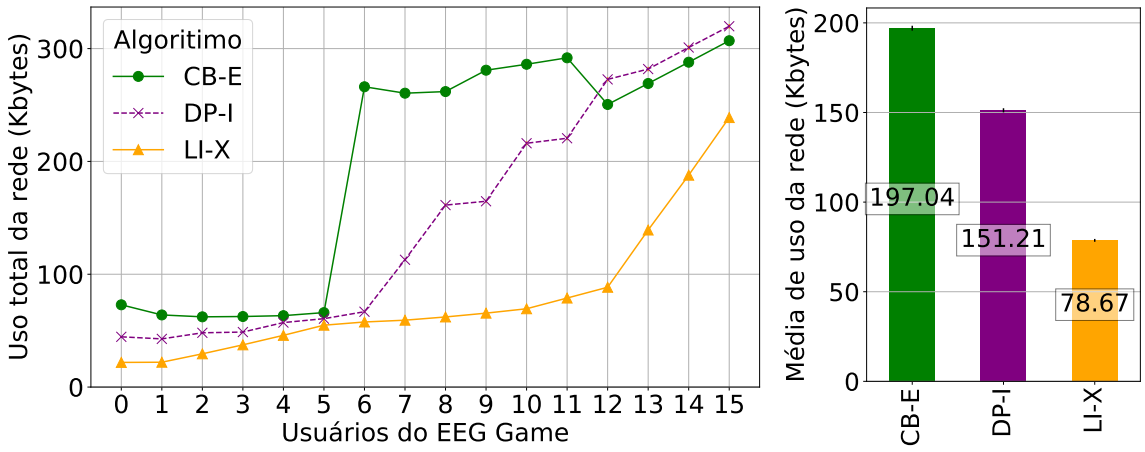


Figura 6.8 Contagem de módulos por dispositivo - VRGame: Topologia Tipo B

Os gráficos da Figura 6.8 apresenta em detalhes a distribuição das instâncias dos

módulos da aplicação VRGame entre os dispositivos a partir da chegada do 12º usuário. Novamente percebe-se que o *CB-E* aloca todos os módulos da aplicação na *cloud*, o *DP-I* distribui os módulos de forma proporcional respeito o limite de recursos e o LI-X dá preferência pela alocação do *concentration_calculator* na *cloudlet-1* o que acaba reduzindo o tempo médio de atraso para a aplicação VRGame visto que o módulo *concentration_calculator* é o que causa maior impacto na comunicação.

6.3 TOPOLOGIA C



(a) Usuários Movidos (b) Média

Figura 6.9 Uso da Rede: Topologia Tipo C.

Users	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CB-E	0,76	0,73	0,74	0,74	0,74	0,74	1,83	1,80	1,82	1,82	1,85	1,90	1,72	1,80	1,91	1,80
DP-I	0,66	0,68	0,71	0,71	0,72	0,73	0,74	1,13	1,38	1,39	1,70	1,71	1,89	1,91	2,12	2,02
LI-X	0,66	0,67	0,69	0,66	0,70	0,71	0,71	0,72	0,75	0,72	0,73	0,80	0,82	1,25	1,48	1,72

Tabela 6.5 Desvio Padrão: Uso de Rede - Topologia C.

A Topologia C acrescenta uma nova camada de *fog* ao experimento, a Figura 6.9(a) apresenta o tráfego de rede total de acordo a chegada de novos usuários e na Figura 6.9(b) é apresento a média de tráfego de rede para todas as quantidades de usuários. A Tabela 6.5 apresenta o desvio padrão para o gráfico da Figura 6.9(a) encontrado a partir das médias das replicações do experimento. Para essa Topologia, o LI-X apresentou um resultado significativamente melhor, enquanto o LI-X obteve uma média de 78,67 Kbytes, o *CB-E* teve uma média 197,04 Kbytes e o *DP-I* teve uma média de 151,21 Kbytes. Nessa topologia, o LI-X foi melhor que os demais algoritmo para todas as quantidades de usuários, como pode ser observado na Figura 6.9(a). Na média do experimento, o LI-X teve um ganho de 60,08% em relação ao *CB-E* e de 47,98% em relação ao *DP-I*.

Quando observado o local onde os módulos das aplicações foram alocados nos gráficos da Figura 6.11, percebe-se que o *CB-E* não consegue alocar nenhum módulo da aplicação

Ao avaliar o atraso para essa Topologia (Figura: 6.10), foi observado um tempo médio de atraso de 40,27 ms para a aplicação VRGame com o uso do LI-X, de 139,33 ms com o uso do *CB-E* e de 52 ms com o uso do *DP-I*, que corresponde a um ganho de 71,1% do LI-X em relação ao *CB-E* e de 22,6% em relação ao *DP-I*. Ao avaliar a aplicação VSOT, o LI-X também apresentou um melhor desempenho quanto a redução do tempo médio de atraso, neste cenário o uso do LI-X provocou uma redução de 14,1% do tempo médio de atraso em relação ao uso do *CB-E* e de 30,2% em relação ao *DP-I*. Dessa forma, para esse cenário, o LI-X também conseguiu o melhor resultado. A Tabela 6.6 apresenta os desvios padrões das médias das replicações do experimento para os gráficos da Figura 6.10.

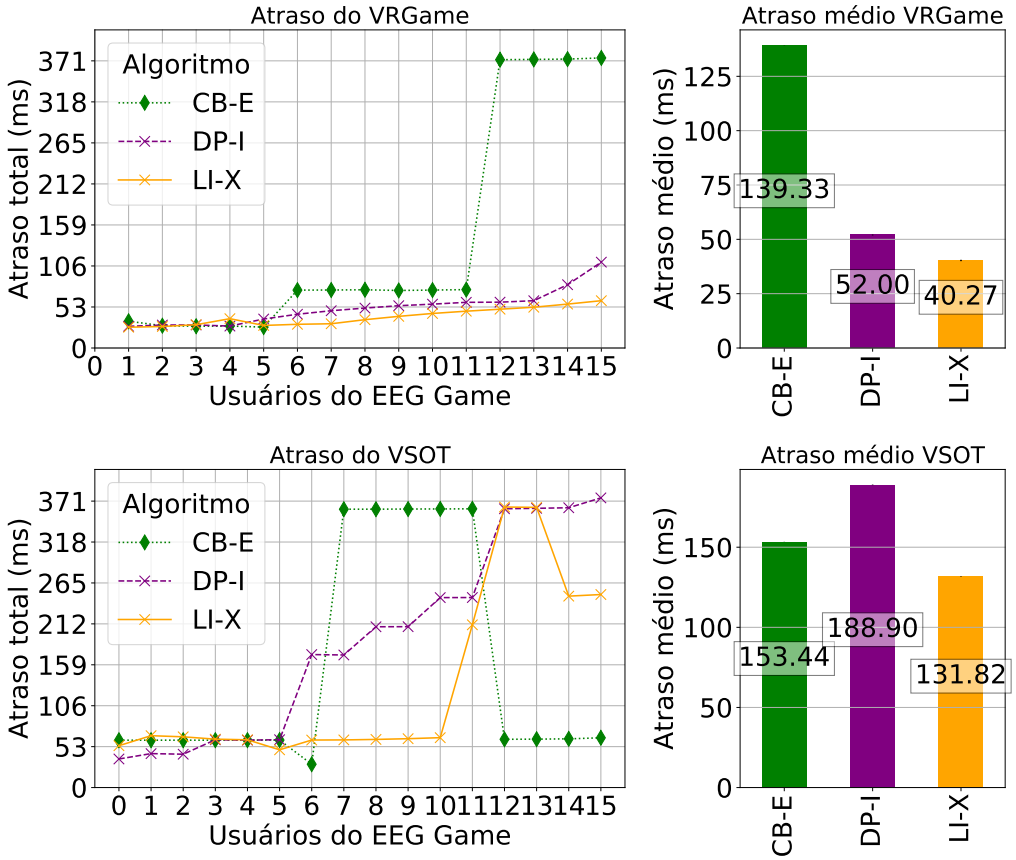


Figura 6.10 Atraso das Aplicações: Topologia Tipo C

VSTO no primeiro nível de *fog*, que nesse cenário possui metade dos recursos disponíveis nos primeiros níveis de *fog* das Topologia A e B, o mesmo ocorre com a aplicação VRGame a partir da chegada do 6º usuário, esse comportamento ocorre pois o *CB-E* ao avaliar os módulos da aplicação que deveriam ser enviados para camadas superiores decide por elevar todos as instâncias de um módulo quando não há espaço suficiente para alocar todos, dessa forma o primeiro nível da *fog* acaba ficando ocioso, o mesmo comportamento ocorre para o nível superior conforme novos usuários vão chegando, e a partir do 12º

	Users	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CB-E	VSOT	0,0	0,0	0,0	0,0	0,0	0,1	0,0	0,0	0,0	0,1	0,1	0,1	0,1	0,1	0,3	0,4
	VRGame	0,0	1,1	0,4	0,2	0,3	0,1	0,1	0,1	0,1	0,1	0,1	0,2	0,1	0,1	0,1	0,4
DP-I	VSOT	0,1	0,1	0,1	0,0	0,0	0,1	0,3	0,3	0,5	0,5	0,4	0,4	0,1	0,2	0,3	7,3
	VRGame	0,0	0,4	0,3	0,3	0,2	0,3	0,3	0,2	0,2	0,2	0,2	0,3	0,2	0,1	0,3	7,0
LI-X	VSOT	3,0	0,4	0,3	0,1	0,0	0,0	0,1	0,1	0,2	0,2	0,3	0,5	0,1	0,2	0,5	0,5
	VRGame	0,0	0,4	0,3	0,3	1,3	0,2	0,4	0,3	0,3	0,3	0,2	0,2	0,2	0,2	0,4	0,8

Tabela 6.6 Desvio Padrão: Atraso - Topologia C.

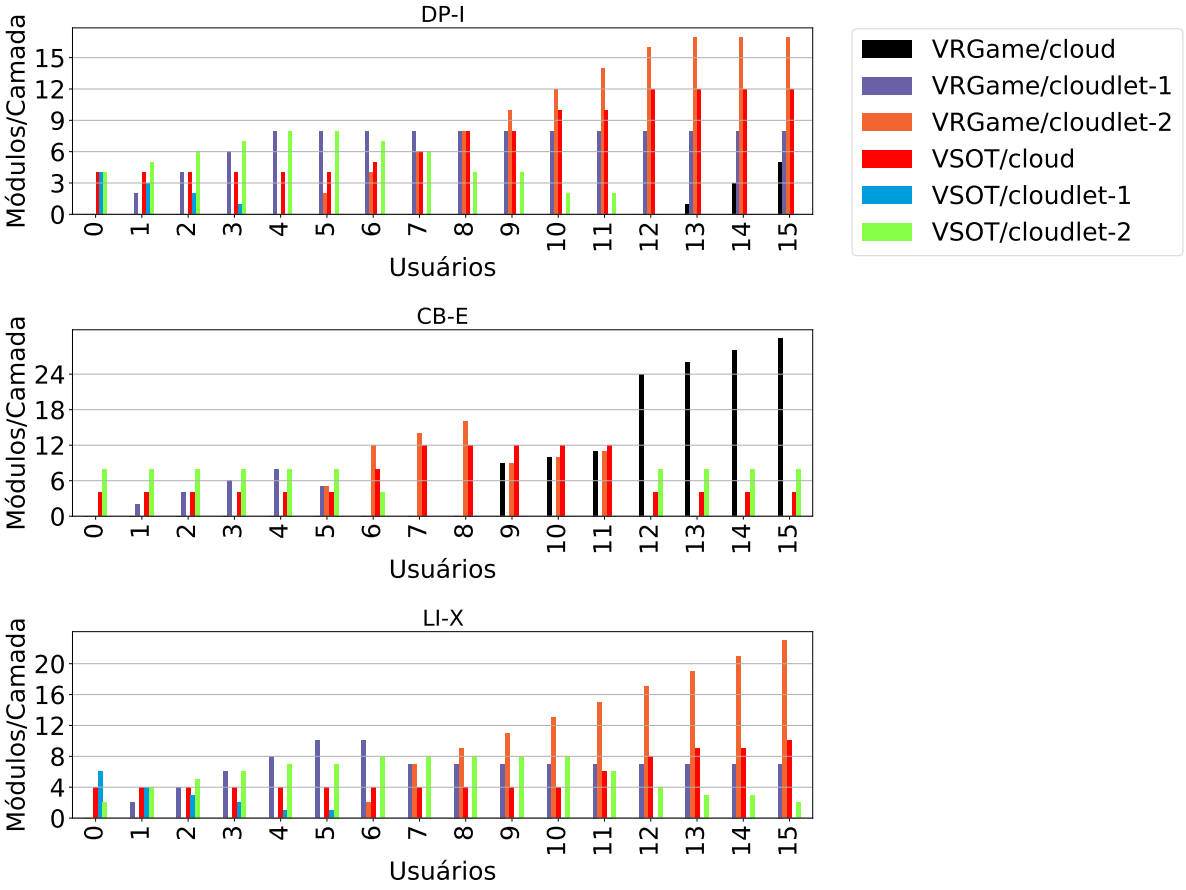


Figura 6.11 Contagem de módulos por dispositivo: Topologia Tipo C

usuário, a aplicação VRGame acaba por ser alocada totalmente na *cloud* elevando dessa forma a média de atraso.

Ainda observando a Figura 6.11, o *DP-I* e o LI-X ao contrário do *CB-E* conseguem utilizar todos os níveis da *fog* conforme os usuários do VRGame começam a chegar, distribuindo os módulos das aplicações a partir dos níveis mais baixos até a *cloud* na ausência de recursos nas camadas inferiores. Contudo, as duas abordagens distribuem os módulos das aplicações de forma diferentes, observando o recorte da distribuição dos módulos da aplicação VRGame a partir da chegada do 10º usuário (Figura 6.12), nota-se que *DP-I* distribui os módulos de forma mais igual, alocando no mesmo dispositivo sempre

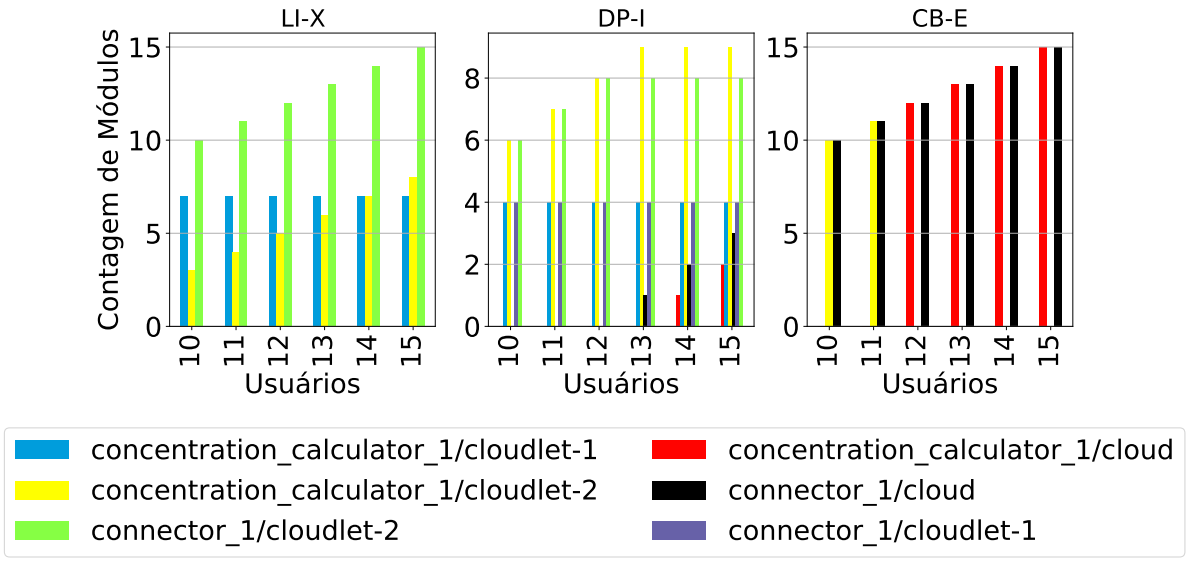


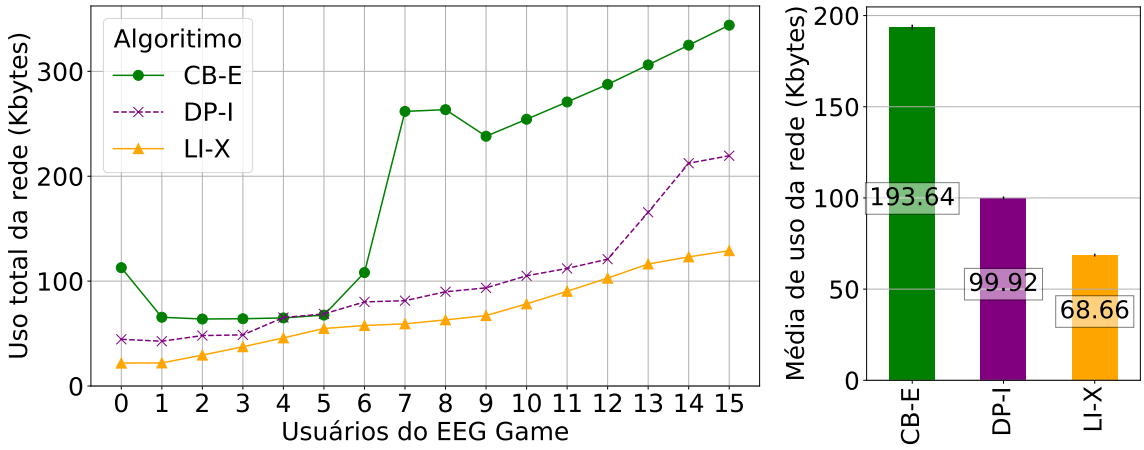
Figura 6.12 Contagem de módulos por dispositivo - VRGame: Topologia Tipo C

que consegue todos os módulos da aplicação para o novo usuário que acaba de chegar, já o LI-X tenta alocar o *concentration_calculator* sempre nas camadas mais baixas e o *connector* quando não consegue alocar na mesma camada, encaminha-o para as camadas superiores, com isso é possível reduzir o atraso ainda mais, visto que dentre os dois módulos este último é o que menos se comunica com os demais módulos da aplicação e também o que gera menos tráfego de dados, contribuindo dessa forma também para a redução do tráfego de dados.

6.4 TOPOLOGIA D

A Topologia D adiciona um terceiro nível da *fog*, na Figura 6.13(a) é apresento o tráfego de rede total de acordo a chegada de novos usuários e na Figura 6.13(b) é apresento a média de tráfego de rede para todas as quantidades de usuários para a Topologia do Tipo D (Figura: 5.6(d)). A Tabela 6.7 apresenta o desvio padrão para o gráfico da Figura 6.13(a) encontrado a partir das médias das replicações do experimento. Para essa Topologia, o LI-X também apresentou um resultado significativamente melhor que os demais algoritmos, enquanto o LI-X obteve uma média de 68,66 Kbytes, o *CB-E* teve uma média 193,64 Kbytes e o *DP-I* teve uma média de 99,66 Kbytes. Também nessa topologia, o LI-X foi melhor que os demais algoritmo para todas as quantidades de usuários, como pode ser observado na Figura 6.13(a). Na média do experimento, o LI-X teve um ganho de 68,55% em relação ao *CB-E* e de 31,29% em relação ao *DP-I*.

Avaliando o tráfego de dados pela rede (Figura: 6.14), o uso do LI-X conseguiu reduzir o tempo médio de atraso em 79,7% em relação ao *CB-E* e 22,9% em relação ao *DP-I* para a aplicação VRGame. Os tempos médios de atraso foram de 40,42 ms, 198,93 ms e 52,42 ms respectivamente. Para este mesmo cenário, observando comportamento da aplicação VSOT, o *CB-E* obteve um tempo médio de atraso de 64,13 ms, o *DP-I* obteve



(a) Usuários Movidos (b) Média

Figura 6.13 Uso da Rede: Topologia Tipo D.

Users	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CB-E	0,93	0,73	0,74	0,74	0,74	0,74	0,90	1,78	1,81	1,64	1,71	1,86	1,95	2,03	2,15	2,06
DP-I	0,66	0,67	0,71	0,70	0,76	0,77	0,77	0,80	0,87	0,83	0,90	0,95	0,99	1,36	1,59	1,41
LI-X	0,66	0,67	0,69	0,66	0,70	0,71	0,71	0,72	0,76	0,72	0,78	0,84	0,90	0,97	0,97	0,95

Tabela 6.7 Desvio Padrão: Uso de Rede - Topologia D.

o tempo de 116,07 ms e o LI-X obteve um tempo de 81,76ms que corresponde a uma perda de 27,5% da performance em relação ao *CB-E* e um ganho de 29,6% em relação ao *DP-I*. Com isso, mesmo não conseguindo o melhor tempo para a aplicação VSOT, o LI-X conseguiu atingir o objetivo de reduzir o tempo médio de atraso da aplicação sensível. A Tabela 6.8 apresenta os desvios padrões das médias das replicações do experimento para os gráficos da Figura 6.14.

	Users	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CB-E	VSOT	0,0	0,0	0,0	0,0	0,1	0,1	0,1	0,0	0,1	0,1	0,1	0,1	0,2	0,1	0,2	0,3
	VRGame	0,0	1,1	0,4	0,2	0,3	0,1	0,2	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,3
DP-I	VSOT	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,3	0,3	0,4	7,9
	VRGame	0,0	0,5	0,4	0,3	0,2	0,2	0,3	0,2	0,3	0,2	0,2	0,2	0,2	0,2	0,2	7,4
LI-X	VSOT	3,0	0,4	0,3	0,1	0,1	0,1	0,2	0,3	0,3	0,1	0,2	0,1	0,1	0,1	0,2	0,8
	VRGame	0,0	0,4	0,3	0,3	1,3	0,2	0,4	0,3	0,3	0,2	0,3	0,2	0,4	0,2	0,2	1,0

Tabela 6.8 Desvio Padrão: Atraso - Topologia D.

Avaliando a alocação dos módulos das aplicações entre os níveis da *fog* na Topologia D. É possível observar nos gráficos da Figura 6.15, que mesmo na ausência de usuários do VRGame, o *CB-E* não consegue alocar nenhum módulo da aplicação VSOT no primeiro nível da *fog*, e divide os módulos do VSOT para quase todas as quantidade de usuários entre níveis superiores da *fog* e a *cloud*, isso devido ao algoritmo não alocar instâncias de um mesmo módulo individualmente em níveis distintos, quanto a aplicação VRGame conforme a chegada de usuários, todas as instâncias das aplicações começam a

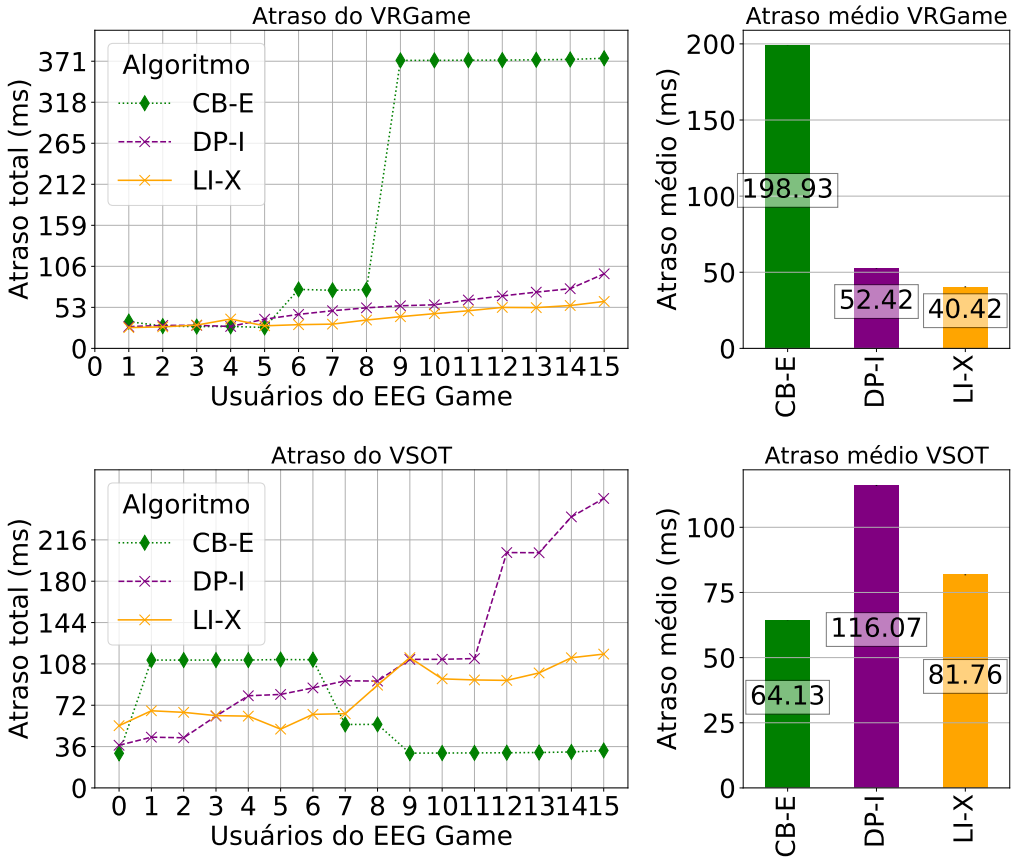


Figura 6.14 Atraso das Aplicações: Topologia Tipo D

ser migradas para níveis superiores de acordo a quantidade de recursos disponíveis, esse comportamento provoca a ociosidade de recurso em níveis mais baixos da *fog* também provoca o aumento do atraso em cada interação, pois as instâncias dos módulos acabam sendo todas alocadas mais distantes do usuário o que reflete o observado na Figura 6.14.

Também é perceptível o comportamento esperado pelo *DP-I*, de alocar as cada instâncias da aplicações dos usuários que vão chegando de acordo a disponibilidade de recursos nos níveis inferiores, conforme os recursos de um nível se torna escasso, os módulos que não foram alocados são direcionados para níveis superiores, esse comportamento evita a ociosidade de recursos, mas não leva em conta o comportamento de cada módulo, o que é feito pelo LI-X, que ao alocar os módulos dá preferência aos que provocam menor impacto, como pode ser observado nos gráficos da Figura 6.16, que apresenta a distribuição dos módulos da aplicação entre os níveis da *fog* a partir da chegada do 10º usuário, nele é possível verificar que o *concetration_calculator* é alocados em camadas mais baixas, enquanto o *connector* é alocado nas camadas mais superiores de acordo a disponibilidade de recursos, dessa forma o LI-X consegue reduzir o tempo médio de atraso, uma vez que o *concetration_calculator* é o módulo da aplicação VRGame que causa maior impacto estando mais longe do usuário.

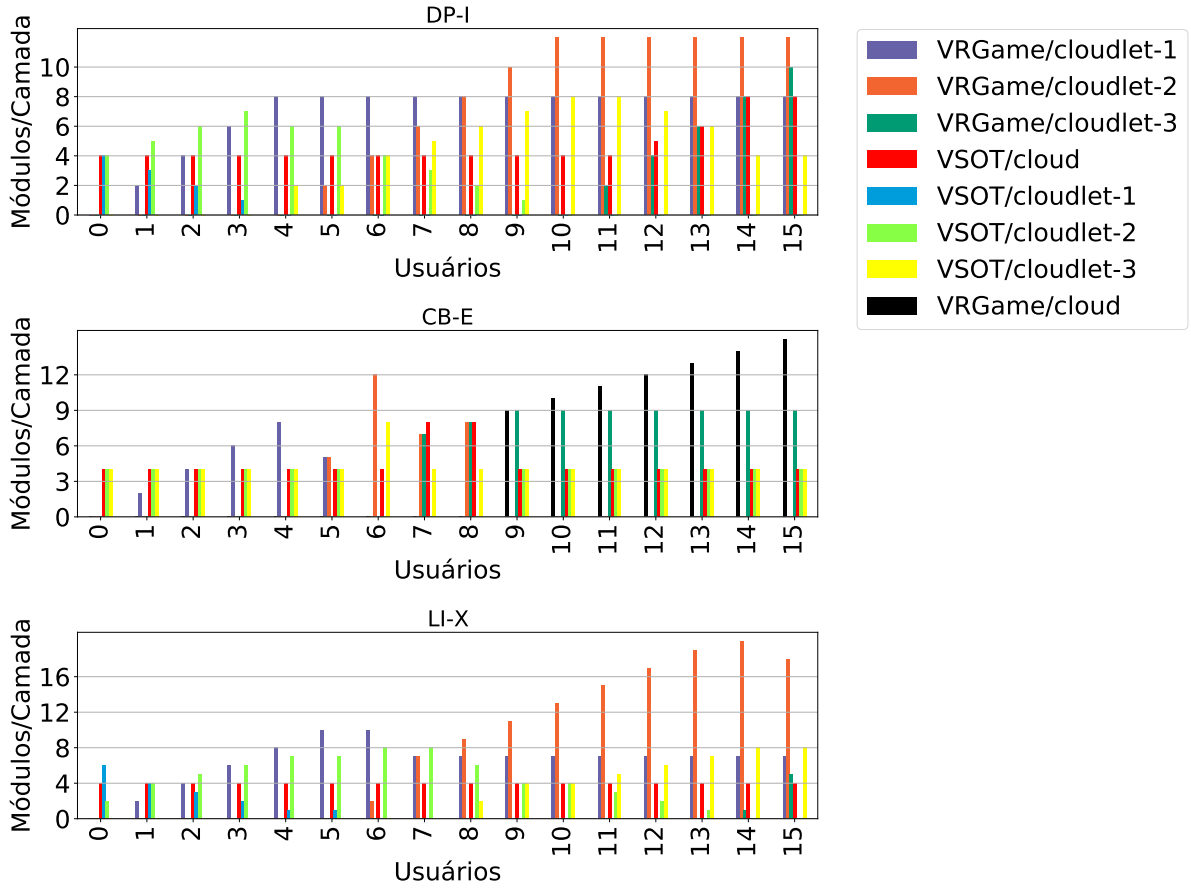


Figura 6.15 Contagem de módulos por dispositivo: Topologia Tipo D

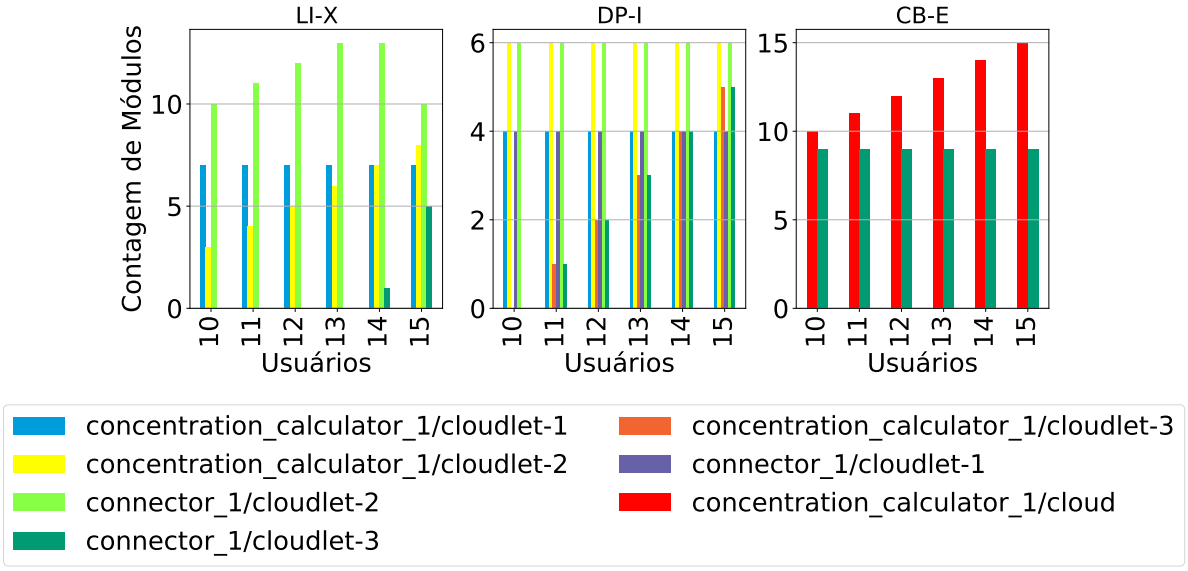


Figura 6.16 Contagem de módulos por dispositivo - VRGame: Topologia Tipo D

6.5 CONSIDERAÇÕES FINAIS

Na Tabela 6.9 é apresentado o resumo do uso da rede pelos algoritmos e a diferença do LI-X em relação aos demais algoritmos. Nesse quesito, o *CB-E* apresentou resultados melhores que o *DP-I* nos cenários em que as *cloudlets* dos níveis mais baixos da *fog* apresentavam recursos suficientes para alocar todos as instâncias de grupos de módulos específicos, pois na ausência de recursos para todos, o *CB-E* acaba elevando grupos inteiros de instâncias de módulos para *cloudlets* das comandas superiores, fazendo com que dessa forma, mesmo que o tráfego de dados entre alguns módulos seja reduzido por estarem um mesmo *cloudlet*, a distância até o dispositivo *Internet of Things* (IoT) aumentou, fazendo com que em alguns casos o tráfego de dados seja afetado.

Para todos as topologias o LI-X obteve um resultado melhor, isso devido ao fato do LI-X buscar a redução do uso da rede posicionado o grupos de instâncias de módulos que causam mais impacto na comunicação em uma mesma *cloudlet*, estratégia semelhante à proposta da *CB-E*, e de buscar reduzir a ociosidade dos recursos das *cloudlets* nos níveis mais baixos da *fog*, estratégia semelhante à proposta do *DP-I*. Este comportamento foi observado em todos os cenários apresentados. Mostrando que para estes cenários, o *LI-X* apresenta ser uma boa solução.

Ainda sobre o tráfego de dados na rede, o *CB-E* apresentou melhores resultados frente ao *DP-I* nas topologias em que as *cloudlets* dos níveis mais baixos da *fog* apresentavam recursos suficientes para alocar todos as instâncias de módulos específicos, pois quando não há recursos suficientes para todos, o *CB-E* acaba elevando grupos inteiros de instâncias de módulos para *cloudlets* em comandas superiores, fazendo com que dessa forma, mesmo que o tráfego de dados entre alguns módulos seja reduzido por estarem uma mesma *cloudlet* a distância até o dispositivo IoT ou a outros módulos aumenta, provocando em alguns casos o aumento do tráfego de dados. O *DP-I* apesar de não fazer essa análise conseguiu ser melhor que o *CB-E* em alguns cenários devido ao melhor aproveitamento dos recursos das *cloudlets* de níveis inferiores, conseguindo em alguns casos colocar mais módulos nessas camadas e reduzir a distância necessária para a comunicação entre os módulos e o dispositivo IoT.

Topologia	CB-E	DP-I	LI-X	Diferença *DP-I*	Diferença *CB-E*
A	183,57 Kb	201,21 Kb	130,95 Kb	34,9%	28,7%
B	305,29 Kb	326,80 Kb	268,64 Kb	17,80%	12,01%
C	197,04 Kb	151,21 Kb	78,67 Kb	47,98%	60,08%
D	193.64 Kb	99,92Kb	68,66Kb	31,29%	68,55%

Tabela 6.9 Resumo: Uso de rede.

A Tabela 6.10 apresenta de forma resumida os resultados da média de atraso das requisições das aplicações para todos os cenários do experimento. Como pode ser visto, o LI-X conseguiu reduzir o tempo médio de atraso da aplicação sensível em todos os casos, e ainda conseguiu reduzir o atraso para a aplicação não sensível em quase todos os cenários, apenas para a topologia do tipo D houve um perda de 27,49%.

O fato de tentar alocar recursos para grupos de módulos de maior impacto na comu-

nicação próximos uns dos outros reduziu o tempo necessário para a comunicação entre esses grupos, aliado a isso, buscar a redução da ociosidade dos recursos das *cloudlets* em níveis inferiores e ser mais seletivo no momento da movimentação dos grupos de módulos, provocou o aumento do posicionamento de módulos relacionados nas camadas mais inferiores da *fog*. Fazendo com que mais instâncias de módulos das aplicações fossem executadas próximos aos dispositivos IoT provocando também a redução dos atrasos devido a comunicação.

Topologia	Aplicação	CB-E	DP-I	LI-X	Difereça DP-I	Diferença CB-E
A	VSOT	230,18	250,67	222,38	11,29%	3,39%
	VRGame	117,87	59,93	30,22	49,57%	74,36%
B	VSOT	248,93	266,59	242,45	9,06%	2,60%
	VRGame	154,22	104,95	79,61	24,14%	48,38%
C	VSOT	153,44	188,90	131,82	30,22%	14,09%
	VRGame	139,33	52,00	40,27	22,56%	71,10%
D	VSOT	64,13	116,07	81,76	29,56%	-27,49%
	VRGame	198,93	52,42	40,42	22,89%	79,68%

Tabela 6.10 Resumo: Atraso médio das aplicações.

Para todos os experimentos, os desvios padrões observados foram pequenos o suficiente para não interferirem nas conclusões apresentadas. O LI-X se apresentou como uma solução mais eficiente para os cenários apresentados frente aos trabalhos comparados.

Capítulo 7

CONCLUSÕES

A *fog computing* busca literalmente aproximar o poder computacional da *cloud computing* dos seus usuários finais, para isso, é necessário que esses recursos estejam distribuídos geograficamente. Dessa forma, é possível reduzir de maneira significativa o tempo de resposta das aplicações que a utilizam, visto que o tempo necessário para a transmissão das requisições e das repostas das aplicações é encurtado, pois o caminho que os pacotes de dados precisam percorrer é menor. Contudo, diferente da *cloud computing*, a *fog computing* não consegue fornecer a mesma capacidade computacional que a *cloud* fornece. devido a isso, o uso de seus recurso precisa ser gerenciado de forma eficiente, de modo a atender da melhor forma a necessidade dos usuários.

Assim como em uma *cloud*, na *fog* os recursos são alocados para aplicações de diferentes naturezas. Este trabalho explorou o uso da *fog* por aplicações modulares. Durante os experimentos, foram utilizadas duas aplicações, uma sensível a latência, que exige respostas rápidas e outra aplicação que tem uma tolerância maior ao tempo de resposta, mas que consome uma quantidade significativa de recursos da rede.

A literatura propõe diferentes algoritmos para alocação de recursos em *cloud* ou em *fog*. Esse trabalho explorou a alocação de aplicações modulares em cenários onde há diferentes níveis de *fog*. O *Least Impact - X* (LI-X) aqui proposto, teve como objetivo ser uma política de alocação de aplicações modulares, que busca inicialmente atender as aplicações mais sensíveis a latências e de forma secundário, reduzir o tráfego de dados na rede.

De forma a avaliar a performance da proposta, o LI-X foi comparado com outros dois algoritmos encontrados na literatura, o *Delay-Priority & Individual* (*DP-I*) e o *Communication Based & Edgewards* (*CB-E*). Para isso, foi montado um ambiente simulado, utilizando o simulador iFogSim em quatro diferentes topologias de arquiteturas *fog*. Foram utilizadas duas diferentes aplicações com diferentes características e foram avaliados o tempo de resposta das aplicações em cada cenário simulado, bem como, a quantidade de dados trafegados pela rede.

Ao fim dos experimentos, o LI-X apresentou melhores resultados em todos os cenários em relação aos demais algoritmos. Dessa forma, o LI-X se apresenta como uma solução alternativa e atrativa, confirmando dessa forma a hipótese desse trabalho.

7.1 TRABALHOS FUTUROS

Como possíveis trabalhos futuros é possível apontar:

- Avaliar o impacto das migrações que ocorrem nos cenários de mobilidade dos usuários quanto ao consumo de rede, eficiência energética, tempo de resposta e custo operacional.
- Ampliar e avaliar os cenários, levando em consideração novos tipos de aplicações e diferentes topologias de organização da *fog*.
- Considerar os custos e a eficiência energética durante o processo de alocação.
- Considerar aplicações ou tarefas que não necessitem de alocação constante de recursos e trabalham sob demanda como as aplicações *serverless*.

REFERÊNCIAS BIBLIOGRÁFICAS

ABURUKBA, R. O. et al. Scheduling internet of things requests to minimize latency in hybrid fog–cloud computing. *Future Generation Computer Systems*, v. 111, p. 539–551, 2020. ISSN 0167-739X. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0167739X18303327⟩.

AL-FUQAHA, A. et al. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys and Tutorials*, v. 17, n. 4, p. 2347–2376, 2015. ISSN 1553877X.

ALI, I. M. et al. An automated task scheduling model using non-dominated sorting genetic algorithm ii for fog-cloud systems. *IEEE Transactions on Cloud Computing*, p. 1–1, 2020.

BITTENCOURT, L. F. et al. Mobility-Aware Application Scheduling in Fog Computing. *IEEE Cloud Computing*, Published by the IEEE Computer Society, v. 4, n. 2, p. 26–35, 2017. ISSN 23256095.

BONOMI, F. et al. Fog computing and its role in the internet of things. In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC '12*. New York, New York, USA: ACM Press, 2012. p. 13. ISBN 9781450315197. ISSN 978-1-4503-1519-7. Disponível em: ⟨http://dl.acm.org/citation.cfm?doid=2342509.2342513⟩.

BORGIA, E. The internet of things vision: Key features, applications and open issues. *Computer Communications*, Elsevier B.V., v. 54, p. 1–31, 2014. ISSN 01403664. Disponível em: ⟨http://dx.doi.org/10.1016/j.comcom.2014.09.008⟩.

BOTTA, A. et al. Integration of Cloud computing and Internet of Things: A survey. *Future Generation Computer Systems*, Elsevier B.V., v. 56, p. 684–700, 2016. ISSN 0167739X. Disponível em: ⟨http://dx.doi.org/10.1016/j.future.2015.09.021⟩.

CHAMOLA, V.; THAM, C.-K.; CHALAPATHI, G. S. S. Latency aware mobile task assignment and load balancing for edge cloudlets. In: *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. [S.l.: s.n.], 2017. p. 587–592.

CHARÂNTOLA, D. et al. Component-based Scheduling for Fog Computing. *UCC 2019 Companion - Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, p. 3–8, 2019.

CHIANG, M.; ZHANG, T. Fog and IoT: An Overview of Research Opportunities. *IEEE Internet of Things Journal*, v. 3, n. 6, p. 854–864, 2016. ISSN 23274662.

DASTJERDI, A. V.; BUYYA, R. Fog Computing: Helping the Internet of Things Realize Its Potential. *Computer*, v. 49, n. 8, p. 112–116, aug 2016. ISSN 0018-9162. Disponível em: ⟨http://ieeexplore.ieee.org/document/7543455/⟩.

GANGADHARAN, G. Open Source Solutions for Cloud Computing. *Computer*, v. 50, n. 1, p. 66–70, jan 2017. ISSN 0018-9162. Disponível em: ⟨http://ieeexplore.ieee.org/document/7807195/⟩.

GAWANDE, U.; HAJARI, K.; GOLHAR, Y. Pedestrian detection and tracking in video surveillance system: issues, comprehensive review, and challenges. *Recent Trends in Computational Intelligence*, IntechOpen, p. 1–24, 2020.

GHOBAEI-ARANI, M.; SOURI, A.; RAHMANIAN, A. A. Resource management approaches in fog computing: a comprehensive review. *Journal of Grid Computing*, Springer, v. 18, n. 1, p. 1–42, 2020.

GIGLI, M.; S., K. Internet of Things: Services and Applications Categorization, Advances in Internet of Things. *Scientific Research*, v. 1 No. 2, p. 27–31, 2011.

GUBBI, J. et al. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, Elsevier B.V., v. 29, n. 7, p. 1645–1660, 2013. ISSN 0167739X. Disponível em: ⟨http://dx.doi.org/10.1016/j.future.2013.01.010⟩.

GUPTA, H. et al. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, Wiley Online Library, v. 47, n. 9, p. 1275–1296, 2017.

HU, P. et al. Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of Network and Computer Applications*, v. 98, p. 27–42, 2017. ISSN 1084-8045. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S1084804517302953⟩.

KAUR, A.; AULUCK, N.; RANA, O. Real-time scheduling on hierarchical heterogeneous fog networks. *IEEE Transactions on Services Computing*, IEEE, 2022.

LAN, L. et al. An event-driven service-oriented architecture for the internet of things. *Proceedings - 2014 Asia-Pacific Services Computing Conference, APSCC 2014*, p. 68–73, 2015.

LI, S.; XU, L. D.; ZHAO, S. The internet of things: a survey. *Information Systems Frontiers*, v. 17, n. 2, p. 243–259, 2015. ISSN 13873326.

MELL, P.; GRANCE, T. The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology. *National Institute of Standards and Technology, Information Technology Laboratory*, v. 145, p. 7, 2011. ISSN 1472-0213. Disponível em: ⟨http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf⟩.

NAMASUDRA, S.; ROY, P.; BALUSAMY, B. Cloud Computing: Fundamentals and Research Issues. *2017 Second International Conference on Recent Trends and Challenges in Computational Models (ICRTCCM)*, p. 7–12, 2017. Disponível em: ⟨http://ieeexplore.ieee.org/document/8057500/⟩.

NGUYEN, D. C. et al. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys Tutorials*, v. 23, n. 3, p. 1622–1658, 2021.

PEIXOTO, M. M.; GENEZ, T. A. L.; BITTENCOURT, L. F. Hierarchical scheduling mechanisms in multi-level fog computing. *IEEE Transactions on Services Computing*, v. 15, n. 5, p. 2824–2837, 2022.

REJIBA, Z.; MASIP-BRUIN, X.; MARÍN-TORDERA, E. A survey on mobility-induced service migration in the fog, edge, and related computing paradigms. *ACM Computing Surveys*, v. 52, n. 5, 2019. ISSN 15577341.

SHAH-MANSOURI, H.; WONG, V. W. Hierarchical fog-cloud computing for iot systems: A computation offloading game. *IEEE Internet of Things Journal*, IEEE, v. 5, n. 4, p. 3246–3257, 2018.

TANEJA, M.; DAVY, A. Resource aware placement of iot application modules in fog-cloud computing paradigm. In: *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. [S.l.: s.n.], 2017. p. 1222–1228.

TOCZÉ, K.; NADJM-TEHRANI, S. A taxonomy for management and optimization of multiple resources in edge computing. *Wireless Communications and Mobile Computing*, Hindawi, v. 2018, 2018.

VAQUERO, L. M. et al. A break in the clouds. *ACM SIGCOMM Computer Communication Review*, v. 39, n. 1, p. 50, 2008. ISSN 01464833. Disponível em: ⟨http://portal.acm.org/citation.cfm?doid=1496091.1496100⟩.

VARSHNEY, P.; SIMMHAN, Y. Demystifying fog computing: Characterizing architectures, applications and abstractions. In: *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*. IEEE, 2017. Disponível em: ⟨https://doi.org/10.11092Ficfec.2017.20⟩.

WU, M. et al. Research on the architecture of Internet of Things. *ICACTE 2010 - 2010 3rd International Conference on Advanced Computer Theory and Engineering, Proceedings*, v. 5, p. 484–487, 2010. ISSN 2154-7491.

XU, R. et al. Real-time human objects tracking for smart surveillance at the edge. In: IEEE. *2018 IEEE International conference on communications (ICC)*. [S.l.], 2018. p. 1–6.

YANGUI, S. et al. A platform as-a-service for hybrid cloud/fog environments. In: *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. [S.l.: s.n.], 2016. p. 1–7.

YANNUZZI, M. et al. Key ingredients in an IoT recipe: Fog Computing, Cloud computing, and more Fog Computing. *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks, CAMAD 2014*, p. 325–329, 2014. ISSN 2378-4865.

YOUSEFPOUR, A. et al. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, Elsevier B.V., v. 98, n. December 2018, p. 289–330, 2019. ISSN 13837621. Disponível em: ⟨https://doi.org/10.1016/j.sysarc.2019.02.009⟩.

ZAO, J. K. et al. Augmented brain computer interaction based on fog computing and linked data. In: IEEE. *2014 International conference on intelligent environments*. [S.l.], 2014. p. 374–377.