UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

JOÃO LUIZ GRAVE GROSS

Um modelo de custo eficiente para minimização da energia consumida e do tempo de processamento de tarefas IoT em um ambiente Mobile Edge Computing

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Orientador: Prof. Dr. Claudio Fernando Resin Geyer

Porto Alegre
2020

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Grave Gross, João Luiz

Um modelo de custo eficiente para minimização da energia consumida e do tempo de processamento de tarefas IoT em um ambiente Mobile Edge Computing / João Luiz Grave Gross. – Porto Alegre: PPGC da UFRGS, 2020.

163 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2020. Orientador: Claudio Fernando Resin Geyer.

1. Mobile Edge Computing. 2. Internet das Coisas. 3. Modelo de custo. 4. Green Computing. 5. Consumo energético. 6. Algoritmo de escalonamento. I. Geyer, Claudio Fernando Resin. II. Título.

“If I have seen further than others,
it is by standing upon the shoulders of giants.“
— Isaac Newton

AGRADECIMENTOS

Este trabalho é dedicada a todos aqueles que de alguma forma contribuíram para sua concretização. Agradeço ao meu orientador, Cláudio Geyer, pela paciência que teve comigo, bem como pelas dicas e contatos que disponibilizou com outros pesquisadores. Agradeço aos meus pais por estarem sempre me incentivando a estudar e por se interessarem pelo trabalho que estava desenvolvendo. Agradeço à minha namorada, Bruna Bernardo Schwarz, por estar ao meu lado quando mais precisei, me apoiando para que conseguisse terminar este trabalho e por valorizar meu esforço. Dedico um agradecimento especial ao meu amigo e colega de Graduação e Mestrado, Hélio Brauner, pelas incontáveis conversas que tivemos à respeito das nossas pesquisas, das dificuldades compartilhadas e pela motivação mútua para seguir e concluir a dissertação. Com sua ajuda pude progredir mais rapidamente e melhorar o texto final.

RESUMO

Com o avanço da quantidade de dispositivos móveis conectados à Internet, aplicações intensivas em dados e requisitos de QoS cada vez mais restritos, juntamente ao fato desses equipamentos serem alimentados por baterias com capacidade limitada, o consumo energético e a latência de resposta das aplicações para Internet das Coisas tornam-se um importante campo de pesquisa. A utilização da *Cloud* como única alternativa ao descarregamento de tarefas levanta dúvidas sobre sua efetividade neste cenário, visto a elevada latência das comunicações. *Mobile Edge Computing* (MEC) é uma arquitetura que introduz uma camada de computação intermediária entre os dispositivos finais e a *Cloud*. Ela proporciona às redes móveis recursos de *Cloud Computing* próximos ao usuário final, de modo que as aplicações dos dispositivos móveis possam descarregar suas tarefas, preservando seu nível de bateria e reduzindo a latência de resposta. Neste cenário é proposto um modelo de custo com diversidade de variáveis de consumo energético e tempo decorrido para a arquitetura MEC, com o objetivo de alocar as tarefas na melhor opção de custo disponível, visando reduzir tanto a energia total consumida pelo sistema como a latência total. O modelo de custo é implementado através do algoritmo de escalonamento TEMS (*Time and Energy Minimization Scheduler*) e validado com experimentos simulados. Os resultados mostram que é possível reduzir a energia consumida no sistema em até 51,61% e o tempo total decorrido em até 86,65% nos casos simulados com os parâmetros e características definidas em cada teste.

Palavras-chave: Mobile Edge Computing. Internet das Coisas. Modelo de custo. Green Computing. Consumo energético. Algoritmo de escalonamento.

A cost efficient model for minimizing energy consumption and processing time for IoT tasks in a Mobile Edge Computing environment

ABSTRACT

With increase in the number of mobile devices connected to the Internet, data-intensive applications and increasingly restricted QoS requirements, together with the fact that these devices are powered by batteries with limited capacity, energy consumption and response latency for applications in the Internet of Things become an important research field. The use of the Cloud as the only alternative to downloading tasks raises doubts about its effectiveness in this scenario, given the high latency of communications. Mobile Edge Computing (MEC) is an architecture that introduces an intermediate computing layer between end devices and the Cloud. It provides mobile networks with Cloud Computing capabilities close to the end user, so that mobile device applications can offload their tasks, preserving their battery level and reducing response latency. In this scenario, a cost model with diversity of energy consumption variables and time elapsed for the MEC architecture is proposed, with the objective of allocating tasks in the best available cost option, aiming to reduce both the total energy consumed by the system and the total latency. The cost model is implemented using the TEMS (Time and Energy Minimization Scheduler) scheduling algorithm and validated with simulated experiments. The results show that it is possible to reduce the energy consumed in the system by up to 51.61% and the total time elapsed by up to 86.65% in the simulated cases with the parameters and characteristics defined in each test.

Keywords: Mobile Edge Computing, Internet Of Things, Cost Model, Green Computing, Energy Consumption, Scheduling Algorithm.

LISTA DE ABREVIATURAS E SIGLAS

ACM	Association for Computing Machinery
BS	Base Station
CC	Cloudlet Computing
CDMA	Code Division Multiple Acces
CMOS	Complementary Metal-Oxide-Semiconductor
CPU	Central Process Unit
D2D	Device-to-device
DaaS	Desktop as a Service
DSRC	Dedicated Short-Range communications
DVFS	Dynamic Voltage and Frequency Scaling
EC	Edge Computing
EVDO	Evolution-Data Optimized
FAQ	Frequently Asked Questions
FC	Fog Computing
FDD	Frequency Division Duplex
GC	Green Computing
GPRS	General Packet Radio Services
GSM	Global Systems for Mobile Communications
HSDPA	High Speed Downlink Packet Access
HSUPA	High Speed Uplink Packet Access
IaaS	Infrastructure-as-a-Service
IDE	Integrated Development Environment
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
ILP	Integer Linear Programming

IoE	Internet of Everything
IoT	Internet of Things
IP	Internet Protocol
IPS	Intrusion Prevention System
ITU	International Telecommunication Union
LAN	Local Area Network
LIS	Limite Inferior de Segurança
LTE	Long Term Evolution
LTE-A	Long Term Evolution Advanced
M2M	Machine-to-machine
MBS	Macro Base Station
MDPI	Multidisciplinary Digital Publishing Institute
MEC	Mobile Edge Computing
NIST	National Institute of Standards and Technology
OFDM	Orthogonal Frequency-Division Multiplexing
ONU	Optical Network Unit
PaaS	Platform-as-a-Service
QoS	QUality of Service
RAM	Random Access Memory
RAN	Radio Access Network
SaaS	Software-as-a-Service
SLA	Service Level Agreements
SMS	Short Message Service
SoC	System-on-a-Chip
SSD	Solid-State Drive
TDD	Time Division Duplex

TEMS	Time and Energy Minimization Scheduler
TI	Tecnologia da Informação
V2I	Vehicle-To-Infrastructure
V2N	Vehicle-To-Network
V2P	Vehicle-to-Pedestrian
V2V	Vehicle-To-Vehicle
VM	Virtual Machine
VPN	Virtual Private Network
WCDMA	Wideband Code Division Multiple Access
WiFi	Wireless Fidelity
WiMAX	Worldwide Interoperability for Microwave Access
WSN	Wireless Sensor Networks

LISTA DE FIGURAS

Figura 2.1 Funcionamento do ambiente de *Cloud Computing*. 23
Figura 2.2 Fornecedores de serviço de diferentes tipos de *Cloud*. 24
Figura 2.3 Quantidade de dispositivos IoT esperados para 2020 e indústrias mais beneficiadas por IoT. 25
Figura 2.4 Complexidade dos dados gerados por dispositivos IoT. 26
Figura 2.5 Arquitetura de *Fog Computing* segundo o NIST. 28
Figura 2.6 Casos de uso dentro do ecossistema MEC. 29
Figura 2.7 Evolução das tecnologias sem fio. 31
Figura 2.8 O espaço de operação entre tensão de alimentação (V), velocidade de operação (T) e energia dinâmica (P). 38
Figura 2.9 Potência vs. tensão da CPU para o processador *Intel® Pentium® M* de 1.6GHz. 40

Figura 3.1 Quantidade de artigos pesquisados na primeira etapa de busca. 45
Figura 3.2 Quantidade de artigos selecionados na primeira etapa de busca. 46
Figura 3.3 Relação entre a quantidade de artigos pesquisados e selecionados. 47
Figura 3.4 Quantidade de artigos selecionados por área. 47
Figura 3.5 Quantidade de artigos por tema e período. 48
Figura 3.6 Quantidade de artigos selecionados por ano. 49
Figura 3.7 Popularidade dos assuntos *"Fog Computing"* e *"Edge Computing"* no mundo, utilizando o buscador do *Google*. 49
Figura 3.8 Quantidade de artigos pesquisados e selecionados por portal de busca. 51
Figura 3.9 Artigos selecionados nos portais de busca. 52
Figura 3.10 Quantidade de artigos por período de publicação. 53
Figura 3.11 Participação percentual do ano de publicação dos artigos selecionados. 53
Figura 3.12 Arquitetura de *Fog Computing* em uma fábrica inteligente (WAN et al., 2018). 55
Figura 3.13 Arquitetura IoT. 57
Figura 3.14 Arquitetura IoT de três camadas detalhada. 57
Figura 3.15 Equação de consumo de energia total minimizada pelo modelo. 59
Figura 3.16 Modelo do sistema. 61
Figura 3.17 Controlador e *Workers* do sistema. 64
Figura 3.18 Conjunto de dispositivos e *setup* utilizados para os experimentos. 65
Figura 3.19 Arquitetura do sistema distribuído com dispositivos móveis. 66
Figura 3.20 Arquitetura do ambiente MEC. 68

Figura 5.1 Resumo comparativo de características dos trabalhos analisados e da proposta deste trabalho. 82
Figura 5.2 Arquitetura do sistema. 83
Figura 5.3 Variáveis consideradas para o modelo de custo. 84
Figura 5.4 Nível de bateria do dispositivo e limite inferior de segurança. 96

Figura 7.1 Ecossistema veicular de aplicações inteligentes. 108
Figura 7.2 Cenários para comunicação veicular. 109
Figura 7.3 Exemplo de um cenário com estações base micro e macro para redes 5G. 113
Figura 7.4 Grafo para simulação da rede. 120
Figura 7.5 Diagrama UML de classes do simulador *ad-hoc*. Classes *IoTDevice*, *MECServer*, *Task*, *Scheduler*, *RAN-5G*, *FiberOptics* e *CloudDataCenter*. 121

Figura 7.6 Diagrama UML de classes do simulador *ad-hoc*. Classes *SimulatorExecution* e *Application*.....121
Figura 7.7 Variação de carga de processamento das tarefas utilizando a Aplicação 1. 126
Figura 7.8 Variação de carga de processamento das tarefas utilizando a Aplicação 2. 129
Figura 7.9 Variação do tamanho da entrada de dados usando a Aplicação 1 com carga definida em $200 * 10^6$ ciclos de CPU. 136
Figura 7.10 Custos totais percebidos pelo escalonador do sistema para a Aplicação 1 com carga definida em $200 * 10^6$ ciclos de CPU com diferentes tamanhos de entrada de dados. Entrada de dados em (1) 3,6 *MB*, (2) 36 *MB*, (3) 362 *MB* e (4) 3,6 *GB*.....137
Figura 7.11 Novo caso para experimentação de tamanho da entrada de dados usando a Aplicação 1 com carga definida em $200 * 10^6$ ciclos de CPU. 138
Figura 7.12 Variação da taxa de geração de tarefas utilizando a Aplicação 2. 140
Figura 7.13 Variação do tamanho do *deadline* utilizando a Aplicação 2.....143
Figura 7.14 Alocação de tarefas conforme o nível de bateria dos dispositivos IoT varia. Utilização da Aplicação 2. 145
Figura 7.15 Nível de bateria dos dispositivos IoT ao alocar tarefas utilizando a Aplicação 2. 145
Figura 7.16 Alocação de tarefas e nível de bateria de apenas um dispositivo IoT utilizando a Aplicação 2.....146
Figura 7.17 Progressão da energia total para dispositivos IoT com nível de bateria em (1.(a)) 3.6 *W*s* e (1.(b)) 36.000 *W*s*.....147

LISTA DE TABELAS

Tabela 4.1 Tipos de arquiteturas utilizadas. 72
Tabela 4.2 Locais de processamento das tarefas no sistema 73
Tabela 4.3 Variáveis consideradas para transmissões de dados. 74
Tabela 4.4 Variáveis de consumo energético utilizadas pelo modelo do sistema. 75
Tabela 4.5 Variáveis de consumo de tempo utilizadas pelo modelo do sistema. 77
Tabela 4.6 Características observadas nos modelos estudados. 78
Tabela 4.7 Formas de experimentação nos trabalhos estudados. 80

Tabela 7.1 Características das aplicações escolhidas. 111
Tabela 7.2 Velocidades das conexões 3G DC-HSPA+, 4G LTE-A Cat16, 5G. 112
Tabela 7.3 Consumo energético dos tipos de transmissões de dados. 114
Tabela 7.4 Potência dinâmica e tempo de execução para uma carga de trabalho no *Arduino Mega 2560*. 115
Tabela 7.5 Potência dinâmica e tempo de execução para um núcleo com uma carga de trabalho no *Raspberry Pi 4 Model B*. 117
Tabela 7.6 Somatório de custos de processamento e transmissões de dados para consumo energético e tempo decorrido. 127
Tabela 7.7 Custos calculados pelo escalonador do sistema para a Aplicação 1. 128
Tabela 7.8 Custos calculados pelo escalonador do sistema para a Aplicação 2. 130
Tabela 7.9 Variação dos coeficientes de custo do sistema utilizando a Aplicação 1. 131
Tabela 7.10 Variação dos coeficientes de custo do sistema utilizando a Aplicação 2. 132
Tabela 7.11 Menores custos calculados para cada caso da Tabela 7.10. 132
Tabela 7.12 Comportamento da alocação de tarefas para a Aplicação 2 ao variar os coeficientes de energia e tempo. 134
Tabela 7.13 Custos calculados pelo escalonador para a Aplicação 2, com coeficiente de energia em $4/5$. Valores ordenados por custo. 141
Tabela 7.14 Somatórios dos custos para diferentes taxas de geração de tarefas utilizando a Aplicação 2. 142
Tabela 7.15 Cenário sem uso de DVFS para a Aplicação 1 com carga de $200 * 10^6$ ciclos de CPU. 148
Tabela 7.16 Cenário com e sem uso de DVFS para a Aplicação 1 com carga de $200 * 10^6$ ciclos de CPU. 148

SUMÁRIO

1 INTRODUÇÃO 16
1.1 Motivação 16
1.2 Contribuições e Definição do Problema 18
1.3 Questão de Pesquisa 19
1.4 Objetivos 20
1.5 Organização do Texto 21
2 REFERENCIAL TEÓRICO 22
2.1 Cloud Computing 22
2.2 Internet das Coisas 24
2.3 Arquiteturas de Borda para Internet das Coisas 26
2.3.1 Edge computing 27
2.3.2 Fog computing 27
2.3.3 Mobile edge computing 29
2.3.3.1 Evolução das tecnologias de comunicação *mobile* 30
2.3.4 Cloudlet computing 33
2.4 Green Computing 34
2.5 Consumo de Energia em CPUs 34
2.5.1 Eletricidade básica: potência, tensão, corrente e resistência 35
2.5.2 Potência em núcleos de processamento 37
2.6 Tempo de execução em CPUs 39
2.7 Técnica DVFS 39
2.8 Algoritmos de Escalonamento de Tarefas 40
2.9 Considerações Sobre o Capítulo 42
3 TRABALHOS RELACIONADOS 43
3.1 Metodologia de Pesquisa 43
3.1.1 Escolhas iniciais 43
3.1.2 Primeira etapa 44
3.1.3 Segunda etapa 50
3.2 Trabalho 1: *Fog Computing for Energy-Aware Load Balancing and Scheduling in Smart Factory* 54
3.2.1 Críticas ao trabalho 1 55
3.3 Trabalho 2: *Energy efficient scheduling in IoT networks* 56
3.3.1 Críticas ao trabalho 2 59
3.4 Trabalho 3: *Energy Efficient Scheduling for Heterogeneous Fog Computing Architectures* 59
3.4.1 Críticas ao trabalho 3 60
3.5 Trabalho 4: *Energy-Delay Tradeoff for Dynamic Offloading in Mobile-Edge Computing System With Energy Harvesting Devices* 60
3.5.1 Críticas ao trabalho 4 62
3.6 Trabalho 5: *Awakening the Cloud Within Energy-Aware Task Scheduling on Edge IoT Devices* 63
3.6.1 Críticas ao trabalho 5 65
3.7 Trabalho 6: *Energy-efficient Offloading Policy for Resource Allocation in Distributed Mobile Edge Computing* 66
3.7.1 Críticas ao trabalho 6 68
3.8 Trabalho 7: *Energy-Efficient Task Offloading and Resource Scheduling for Mobile Edge Computing* 68
3.8.1 Críticas ao trabalho 7 70

3.9 Considerações Sobre o Capítulo 70
4 ANÁLISE COMPARATIVA 71
4.1 Tópicos de Análise 71
4.1.1 Tópico 1: Quanto ao tipo de arquitetura utilizado 71
4.1.2 Tópico 2: Quanto aos locais de processamento das tarefas 73
4.1.3 Tópico 3: Quanto às transmissões de dados 74
4.1.4 Tópico 4: Quanto ao consumo de energia 74
4.1.5 Tópico 5: Quanto ao tempo 77
4.1.6 Tópico 6: Quanto às características gerais do modelo 78
4.1.7 Tópico 7: Quanto à forma de experimentação 80
4.2 Considerações Sobre o Capítulo 80
5 PROPOSTA DE MODELO 81
5.1 Características do Sistema e Modelo Proposto 81
5.2 O Modelo de Custo do Sistema 85
5.2.1 Escolhas e premissas para o modelo 85
5.2.2 Modelo da rede 87
5.2.3 Computação local no dispositivo IoT 88
5.2.4 Computação local no servidor local (servidor MEC) 89
5.2.5 Custo em *idle* para dispositivos móveis e servidores locais 92
5.2.6 Computação remota na *Cloud* centralizada 92
5.2.7 Equação de custo do sistema - problema de *programação linear inteira* (ILP) 94
5.2.7.1 Coeficientes da equação de custo do sistema 95
5.2.7.2 Nível de bateria dos sispositivos IoT 96
5.3 Considerações Sobre o Capítulo 97
6 ALGORITMO DE ESCALONAMENTO 98
6.1 Descrição do Problema 98
6.2 Time and Energy Minimization Scheduler (TEMS) 99
6.3 Complexidade 104
6.4 Considerações Sobre o Capítulo 106
7 IMPLEMENTAÇÃO E AVALIAÇÃO 107
7.1 Características Gerais da Implementação 107
7.1.1 Aplicações veiculares 107
7.1.2 Aplicações escolhidas 110
7.1.3 Latência nas transmissões de dados 111
7.1.4 Consumo energético nas transmissões de dados 113
7.1.5 Frequências de operação e consumo energético nos dispositivos IoT 115
7.1.6 Frequências de operação e consumo energético nos servidores MEC 116
7.1.7 Frequência de operação e consumo energético na *Cloud* 117
7.1.8 Bateria dos dispositivos IoT 118
7.2 Simulador *Ad-hoc* 118
7.3 Detalhamento da Implementação 120
7.4 Descrição dos Experimentos 122
7.4.1 Cenários de teste com variação de carga de processamento das tarefas 123
7.4.2 Cenários de teste com variação do coeficiente de custo (relação tempo vs. energia) 123
7.4.3 Cenários de teste com variação do tamanho da entrada de dados da aplicação 123
7.4.4 Cenários de teste com variação da taxa de geração de tarefas 124
7.4.5 Cenários de teste com variação do tamanho do *deadline* 124
7.4.6 Cenários de teste com variação do nível de bateria dos dispositivos IoT 124
7.4.7 Cenários de teste com e sem DVFS 125

7.5 Análise dos Resultados 125
7.5.1 Variação de carga de processamento das tarefas 125
7.5.2 Variação dos coeficientes de custo do sistema 130
7.5.3 Variação do tamanho da entrada de dados 135
7.5.4 Variação da taxa de geração de tarefas 139
7.5.5 Variação do tamanho do *deadline* 142
7.5.6 Variação do nível de bateria dos dispositivos IoT 144
7.5.7 Cenário com e sem uso de DVFS 147
7.6 Considerações Sobre o Capítulo 149
8 CONCLUSÃO 151
8.1 Contribuições 154
8.1.1 Contribuições adicionais 154
8.2 Trabalhos Futuros 155
REFERÊNCIAS 157

1 INTRODUÇÃO

Esta dissertação tem como objetivo desenvolver um modelo de custo para um ambiente *Mobile Edge Computing* (MEC), de modo a reduzir o consumo energético do sistema e os tempos decorridos nas transmissões de dados e processamento de tarefas. Um algoritmo heurístico é desenvolvido para solucionar o modelo de custo e sua implementação é realizada no escalonador de tarefas do sistema. Nas seções seguintes são apresentados a motivação, contribuições e definição do problema, questão de pesquisa, objetivos e, por fim, a organização do trabalho, com uma breve descrição de cada capítulo.

1.1 Motivação

Bilhões de dispositivos inteligentes podem agora se conectar à Internet na forma de *Internet das Coisas* (*Internet of Things*, ou IoT) devido a avanços nas tecnologias de comunicação (AL-FUQAHA et al., 2015). De acordo com um relatório da Cisco, esses dispositivos irão gerar 507,9 *ZB* de dados em 2019 (HASSAN et al., 2018) e a receita global gerada por tecnologias de *Big Data* irá disparar dramaticamente de US$ 189 bilhões em 2019 para US$ 274 bilhões em 2022 (MARVIN, 2020). Porém, nos primórdios da computação, a situação era muito distinta da atual, quando computadores eram ainda grandes em tamanho, e altamente custosos e tarefas eram executadas em máquinas centralizadas. Desde então houve diversos ciclos de evolução, cada um com características que marcaram a evolução do seu ciclo em relação ao anterior.

No primeiro ciclo de computação os sistemas e aplicações eram desenvolvidos para uma máquina e para seu respectivo modo de operação em particular, executando em máquinas centralizadas de larga escala exclusivamente no modo *batch*. O projeto de cada sistema refletia a forma como o processamento de dados era realizado, com processamento *batch* e centralizado (ARMY, 1977). O segundo ciclo foi marcado pelo advento dos mini-computadores, ou computadores pessoais, com maior consideração quanto à performance das tarefas, para que fossem executadas da maneira mais eficiente possível, independentemente do processamento ser centralizado ou descentralizado, em tempo real ou em modo de operação *batch*. Dentre outras vantagens a liberdade de poder executar tarefas no próprio dispositivo sem a necessidade de uma unidade central de gerenciamento de recursos foi um marco deste ciclo. Porém, os computadores pessoais não estavam conectados uns aos outros, e assim, a capacidade computacional era limitada à disponível

no próprio dispositivo.

Os supercomputadores permitiram uma solução centralizada para aplicações de alta demanda computacional, marcando o início do terceiro ciclo de computação. Logo, *grids* e *clusters* proveram uma vasta gama de recursos computacionais de modo descentralizado. Múltiplas máquinas são conectadas umas às outras em grandes grupos colaborativos, chamados *clusters*, ofertando recursos ociosos para terceiros ou mesmo com máquinas dedicadas a esse propósito.

No início dos anos 2000 *Cloud Computing* tornou-se popular (HECK et al., 2018). Com acesso fácil a recursos de computação e armazenamento, interfaces intuitivas e modelos de negócio *pay-per-use* viáveis economicamente, *Cloud Computing* rapidamente ganhou espaço entres usuários domésticos e empresas. Pode-se dizer que *Cloud Computing* é uma evolução do modelo de consumo de recursos oferecido por *grids* e *clusters*, ofertando mais opções de recursos e customizações diversas ao usuário final.

O ciclo de computação atual que vivemos, o quarto ciclo de computação, é marcado por tecnologias inovadoras que permitiram a milhares de dispositivos ter conectividade com a Internet em qualquer local em que estejam. A Internet das Coisas lida com essa enorme gama de dispositivos com conectividade que demandam recursos computacionais para execução das suas tarefas. O aumento do número de dispositivos inteligentes conectados à Internet e a crescente geração de dados nestes dispositivos criam problemas que até então não existiam (BANGUI et al., 2018). O processamento dessas informações em servidores centralizados torna a operação proibitiva, pois o volume de dados e a velocidade com que são gerados, dificulta que o usuário tenha uma experiência de qualidade, com baixas latências de comunicação, caso o processamento seja exclusivamente remoto, longe do local onde os dados são criados (SATYANARAYANAN et al., 2009).

Problemas relativos à latência e privacidade surgem quando dispositivos na periferia da rede demandam recursos localizados em unidades centrais da rede, tais como os oferecidos pelos serviços de *Cloud Computing*. Além disso há um consumo de energia considerável, visto que *Data Centers* modernos implementam processamento paralelo de larga escala, que requer que grandes quantidades de dados sejam transferidas entre as VMs desses equipamentos. Como consequência, a energia consumida pela rede dos *Data Centers* pode representar mais de 20% de toda a energia consumida, enquanto a comunicação inter-VM pode gastar mais de 33% de todo o tempo de processamento (BACCARELLI et al., 2016).

Esse cenário com fluxos intensos de dados e milhares de dispositivos IoT conec-

tados à Internet demandou arquiteturas inovadoras que permitissem o processamento das tarefas próximo aos dispositivos, de modo a melhorar a *Qualidade de Serviço* (*Quality of Service*, ou QoS).

Fog Computing (FC) é uma arquitetura desenvolvida para estar próxima ao usuário final e tem por objetivo diminuir a latência e prover melhor QoS (HAOUARI; FARAJ; ALJA'AM, 2018). As altas velocidades de conexão de Internet com a *Cloud* e a proximidade física com os usuários também são outras características que permitem o funcionamento de aplicações em tempo real e serviços baseados em localização, além do suporte à mobilidade (STOJMENOVIC, 2014). Em um ambiente no qual a quantidade de dispositivos vem crescendo continuamente e suas demandas computacionais acompanham essa tendência de alta, FC é um paradigma essencial ao bom funcionamento das diversas aplicações que demandam recursos na periferia da rede.

Já *Mobile Edge Computing* (MEC), ou *Multi-acess Edge Computing*, é uma arquitetura de rede com objetivos semelhantes aos de FC, porém focada em redes móveis da era 5G. Atualmente a maioria das aplicações tende a descarregar o processamento de tarefas e o armazenamento de dados para servidores remotos, geralmente situados longe do usuário final e do dispositivo. MEC permite o provisionamento de serviços de *Cloud* próximos dos dispositivos móveis, trazendo processamento e armazenamento mais perto das estações celulares base (YU, 2016).

Entretanto, o consumo de energia em redes IoT complexas, como em ambientes FC e MEC, é um importante problema conforme relatado em trabalhos prévios (SARANGI; GOEL; SINGH, 2018). A maioria dos sensores IoT e atuadores são pequenos e possuem grandes limitantes de energia. Eles geralmente operam por baterias ou usam fontes intermitentes de energia como a energia solar. Dado o fato de que dispositivos IoT lidam com muitos dados, e requerem muita energia para processá-los, reduzir o consumo de energia em redes com dispositivos IoT é um objetivo que vale a pena ser explorado.

1.2 Contribuições e Definição do Problema

O trabalho possui como foco a elaboração de um modelo de custo detalhado com variáveis não previstas em modelos de custo de implementações prévias, tais como as variáveis de tempo e energia das transmissões de dados e código que ocorrem quando uma tarefa gerada em um dispositivo móvel é alocada para execução em um servidor local. Desse modo é possível ter uma percepção mais realista sobre o custo energético do sistema

e os tempos de execução, haja vista as transmissões de dados possuírem um impacto relevante na definição dos tempos totais de processamento, sempre que há transmissão de dados.

Devido ao grau de complexidade da solução ótima do modelo de custo, e seu elevado custo computacional em um sistema real (ZHANG et al., 2018), uma heurística foi desenvolvida de modo a obter uma solução semelhante à solução ótima para o problema, porém com processamento reduzido. Essa heurística faz parte da tomada de decisão do escalonador de tarefas do sistema, para que as tarefas sejam alocadas e processadas localmente no próprio dispositivo, localmente no servidor local, ou mesmo remotamente na *Cloud*. A inserção da *Cloud* no modelo de custo é outra contribuição deste trabalho, considerando a carência em propostas que contemplem a *Cloud* ao dimensionar o custo total dos sistemas. Em sistemas reais a *Cloud* funciona como uma válvula de escape toda vez que a rede local fica saturada. A *Cloud* conceitualmente possui recursos ilimitados; já o sistema local é limitado aos recursos de *hardware* disponíveis pelos próprios dispositivos móveis e servidores da rede local. Portanto, inserir recursos de *Cloud* no sistema, como mais uma alternativa de processamento, agrega confiabilidade à rede, pois desafogar a rede reduz a probabilidade de queda na Qualidade de Serviço.

A metodologia deste trabalho também se destaca como uma contribuição. As etapas definidas para a busca e seleção de artigos científicos foram descritas em detalhes e a utilização de diferentes plataformas de produção acadêmica foram fundamentais para encontrar referências bem citadas e diversidade de soluções semelhantes à proposta aqui apresentada. A busca por artigos também permitiu encontrar trabalhos para produzir um referencial teórico mais rico e a relevância desta pesquisa pôde ser atestada pela constante produção de novos trabalhos com mesmo viés de pesquisa, focado na redução do consumo de energia em sistemas de FC e MEC.

1.3 Questão de Pesquisa

Frente à carência de modelos de custo ricos em variáveis de tempo e consumo energético, que considerem as transmissões de dados e a *Cloud* como uma alternativa ao processamento de tarefas, além dos dispositivos móveis e dos servidores locais, busca-se responder à questão principal que fundamenta este trabalho: "É possível desenvolver um modelo de custo com múltiplas variáveis de energia e tempo para processamento de tarefas que permita a redução do consumo de energia e do tempo de processamento em um

sistema *Mobile Edge Computing* para dispositivos móveis, com a *Cloud* como alternativa ao descarregamento de tarefas?".

Esta questão é respondida, conforme será visto ao longo do desenvolvimento do trabalho, de forma positiva, visto que o modelo de custo criado e implementado através de um algoritmo heurístico para o escalonamento de tarefas do sistema reduziu o consumo energético e o tempo de processamento em diferentes cenários.

1.4 Objetivos

Este trabalho possui dois objetivos principais. O primeiro refere-se à minimização do consumo de energia global do sistema, de modo que uma mesma carga de trabalho possa ser executada com menor consumo de energia, devido à utilização de arquiteturas de sistema apropriadas, de *hardware* com microarquiteturas otimizadas e de um escalonador de tarefas eficaz. Já o segundo refere-se à minimização dos tempos de execução, mantendo a entrega de resultados abaixo do deadline de cada tarefa de forma a preservar a experiência do usuário. Os objetivos específicos do trabalho são:

- Desenvolver um modelo de custo do sistema que contemple variáveis de custo energético e de tempo de processamento, considerando as transmissões de dados e de código das aplicações entre os dispositivos móveis, servidores locais e a *Cloud*.
- Desenvolver uma heurística que solucione o modelo de custo do sistema e permita que o escalonador possua complexidade reduzida, com processamento mais ágil e menos custoso computacionalmente.
- Projetar um conjunto de cenários de teste em ambiente simulado para validar a heurística proposta e atestar sua eficácia.
- Disponibilizar uma metodologia de pesquisa que possa servir de referência a outros trabalhos.
- Esclarecer os conceitos de *Edge Computing*, *Fog Computing*, *Mobile Edge Computing* e *Green Computing*, visto a diversidade de definições encontradas na literatura e as constantes sobreposições de conceitos de cada tópico.

1.5 Organização do Texto

Este trabalho foi dividido em 8 capítulos. O Capítulo 1 contempla a Introdução. O Capítulo 2 apresenta o referencial teórico, com conceitos e definições sobre os diferentes paradigmas de computação de borda para Internet das Coisas, bem como explicações sobre como é calculado o consumo energético e o tempo de execução em núcleos de processamento, uso da técnica de DVFS, esclarecimento do conceito de *Green Computing* e algoritmos de escalonamento utilziados em ambientes estáticos. O Capítulo 3 apresenta a metodologia de busca de artigos e um detalhamento sobre os artigos selecionados que englobam o estado da arte de modelos de minimização de custo em ambientes FC e MEC. O Capítulo 4 apresenta um comparativo entre os trabalhos relacionados e discute sobre características de interesse à proposta de modelo do sistema. O Capítulo 5 apresenta a proposta de modelo do sistema e seu modelo de custo. O Capítulo 6 apresenta a heurística utilizada no algoritmo de escalonamento que implementa o modelo de custo, com a descrição de seu funcionamento e o cálculo de complexidade. O Capítulo 7 apresenta um detalhamento sobre a implementação do algoritmo heurístico do Capítulo 6, a forma de simulação e o simulador desenvolvido, as características do *hardware* do sistema e das tecnologias de transmissão de dados, a definição do conjunto de experimentos e os dados coletados, o ambiente de execução do simulador, os resultados e sua avaliação. E, por fim, no Capítulo 8 são apresentadas as conclusões da pesquisa realizada, as contribuições realizadas e os trabalhos futuros.

2 REFERENCIAL TEÓRICO

O objetivo deste capítulo é apresentar diferentes arquiteturas de rede que proveem recursos para dispositivos ligados à Internet, tais como os dispositivos IoT, de modo a discorrer sobre as diferentes características de cada arquitetura, recursos oferecidos e proximidade dos dispositivos na borda da rede. Além disso, são apresentados diferentes conceitos encontrados na literatura a respeito de *Green Computing* e, por fim, uma abordagem sobre métricas de energia e tempo utilizadas neste trabalho.

2.1 Cloud Computing

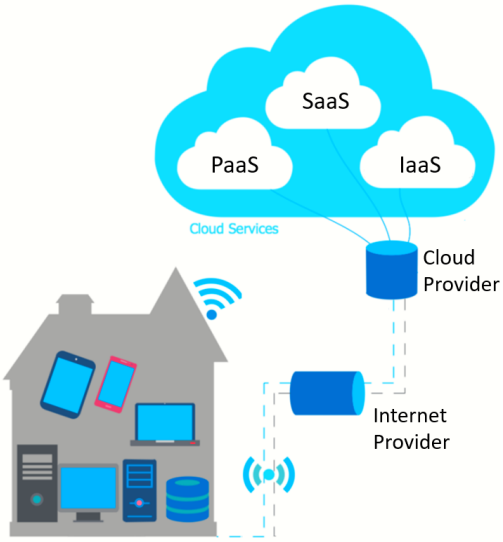
A partir dos anos 2000 avanços em tecnologias para computação e virtualização permitiram o surgimento de grandes *Data Centers* de baixo custo que executam grande parte das aplicações de Internet e processamento de *backend* (JENNINGS; STADLER, 2014). A economia de escala que surgiu permitiu à infraestrutura de *Data Centers* ser rentável para terceiros. Assim surgiu o paradigma de *Cloud Computing*, um modelo de entrega de serviço e acesso na qual recursos dinâmicos escaláveis e virtualizados são oferecidos como um serviço pela Internet. O objetivo principal de *Cloud Computing* é prover serviços computacionais *on-demand* com alta confiabilidade, escalabilidade e disponibilidade em ambientes distribuídos. Um dos importantes requerimentos de um ambiente *Cloud Computing* é prover QoS confiável. Ele pode ser definido em termos de *Acordos de Nível de Serviço* (*Service Level Agreements*, ou SLA), que descrevem características como mínimo *throughput*, máximo tempo de resposta ou latência entregue pelo sistema (PRIYA; PILLI; JOSHI, 2013).

Segundo o *Instituto Nacional de Padrões e Tecnologias* (*National Institute of Standards and Technology*, ou NIST), *Cloud Computing* pode ser definido como "um modelo para permitir acesso de rede ubíquo, conveniente e *on-demand* para um conjunto compartilhado de recursos computacionais configuráveis (e.g., redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados e executados com o mínimo de esforço de gerenciamento ou interação com o provedor de serviço. Esse modelo de *Cloud* é composto por três modelos de serviço, quatro modelos de implantação e cinco características essenciais."(MELL; GRANCE, 2011). Já segundo (DOLUI; DATTA, 2017) *Cloud Computing* é um paradigma baseado em *Data Centers* capazes de lidar com armazenamento e processamento para grandes volumes de dados. Esses *Data*

Centers são conectados uns aos outros por redes de fibra óptica para formar redes de *Data Centers* aparecendo como um único recurso ao usuário final, com baixa latência de comunicação entre eles.

Os principais modelos de serviço de *Cloud Computing* são classificados como SaaS (*Software-as-a-Service*), PaaS (*Platform-as-a-Service*) e IaaS (*Infrastructure-as-a-Service*), porém há modelos novos, como por exemplo, DaaS (*Desktop as a Service*). Os modelos de implantação são categorizados em *Clouds* públicos, privadas, comunitárias e híbridas. As características incluem serviços *on-demand*, acesso amplo à rede, agrupamento de recursos, elasticidade rápida, e serviços de medição (PRIYA; PILLI; JOSHI, 2013). A Figura 2.1 demonstra o funcionamento da *Cloud* em um ambiente residencial ou comercial, no qual aplicações de diferentes equipamentos solicitam recurso à *Cloud* centralizada para satisfazer suas necessidades computacionais e de armazenamento.

Figura 2.1: Funcionamento do ambiente de *Cloud Computing*.

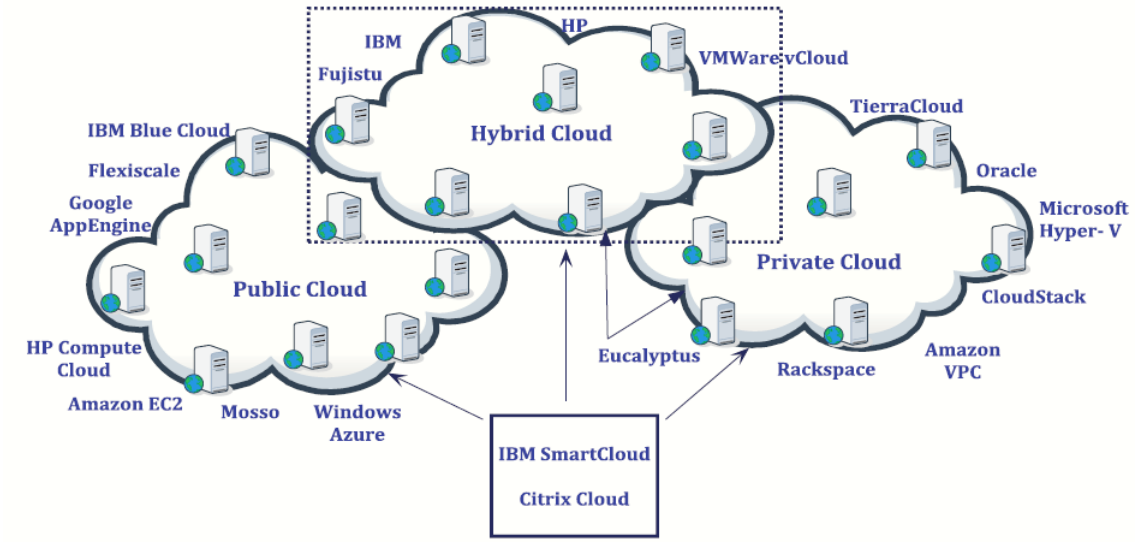


Fonte: (CONCEPTDRAW, 2019), adaptada.

Cloud Computing tem se tornado um termo amplo e popular, utilizado não somente pela comunidade tecnológica, mas também pelo público em geral. A crescente demanda por serviços de *Cloud* incentivou o surgimento acelerado de empresas ofertando tais serviços, conforme mostra a Figura 2.2. Com diferentes modelos de entrega de recursos, suprindo as necessidades de aplicações entregues pela Internet, assim como *hardware* e sistemas de software que residem nos *Data Centers* e hospedam essas aplicações, esse modo de computação é usado em larga escala.

Recentemente, com o advento de recursos de virtualização, *Cloud Computing* é considerada uma das melhores escolhas para satisfazer as necessidades de aplicações de alto desempenho (KAUR; CHANA, 2015). As diferentes configurações de *Data Cen-*

Figura 2.2: Fornecedores de serviço de diferentes tipos de *Cloud*.



Fonte: (KAUR; CHANA, 2015).

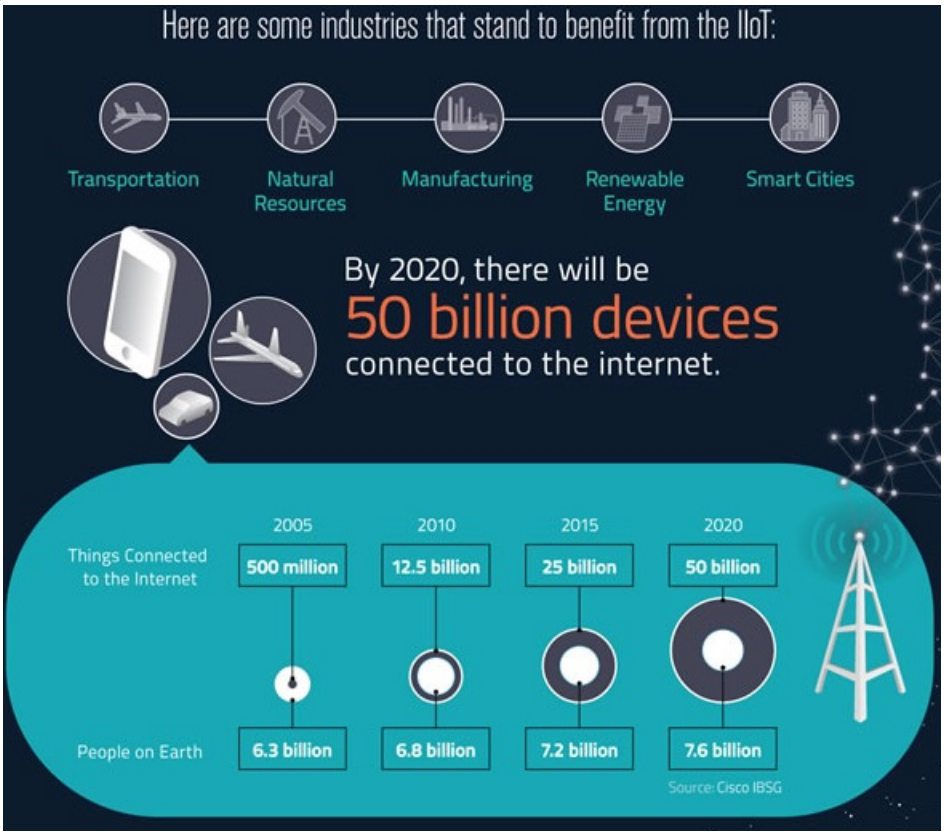
ters para *Cloud Computing*, embora hospedem servidores com alto poder computacional e armazenamento para volumes imensos de dados, dissipam grande quantidade de calor e precisam ser mantidos em ambientes com ar condicionado em tempo integral (EECS, 2012), possuindo elevado consumo de energia. O fato de serem centralizadas e geralmente distantes do usuário final também é um agravante, em um cenário no qual cada vez mais dispositivos conetados à Internet demandam recursos com necessidade de latência reduzida.

2.2 Internet das Coisas

Nos próximos anos, é esperado um número crescente de dispositivos físicos conectados à Internet a uma taxa sem precedentes, à medida que a indústria e a sociedade implementam IoT. A Figura 2.3 mostra como esta evolução está acelerada. Para 2020 são esperados mais de 50 bilhões de dispositivos IoT e este é só o começo da revolução, visto que muitas indústrias e usuários domésticos ainda nem começaram o processo.

O objetivo principal de IoT é permitir que objetos físicos colaborem entre si sem intervenção humana para compartilhar informação; que coordenem inteligência e decisões em tempo real e, principalmente, que entreguem serviços ubíquos através de todo o espectro de atividades humanas (JALALI et al., 2017). IoT está impulsionando muitas aplicações como videomonitoramento inteligente, sensoriamento veicular e análise do comportamento humano. No geral, uma aplicação IoT requer um conjunto de disposi-

Figura 2.3: Quantidade de dispositivos IoT esperados para 2020 e indústrias mais beneficiadas por IoT.

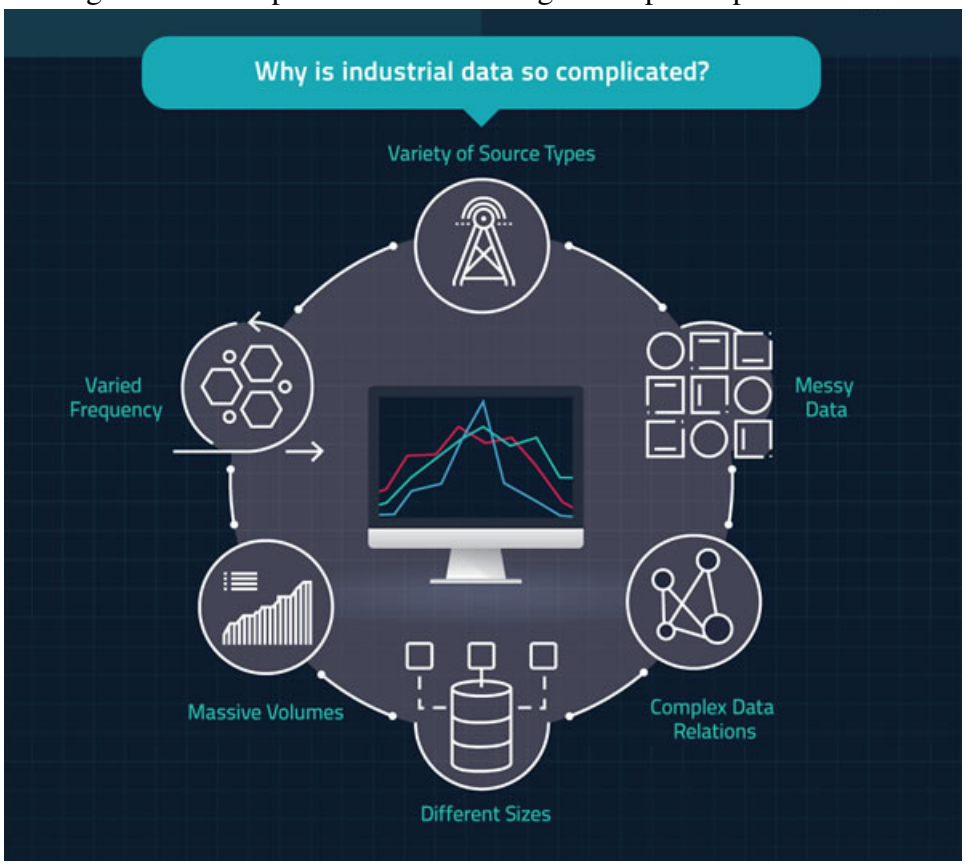


Fonte: (VISUALCAPITALIST, 2019).

tivos IoT que capturam dados dos arredores, executam algoritmos computacionalmente intensos de *machine learning* (e.g. *deep learning*) e geram conhecimento que facilita a vida diária das pessoas e o gerenciamento governamental (YU et al., 2018). Com a era de *Big Data* e inteligência artificial em constante evolução, dispositivos IoT com poder de processamento e recursos energéticos limitados apresentam dificuldade em atender uma crescente demanda por diferentes aplicações envolvendo os pilares de variedade, volume e velocidade (BACCARELLI et al., 2016). A Figura 2.4 exibe a variedade e complexidade dos dados gerados por dispositivos IoT.

Devido à diversidade de dados dos dispositivos IoT, conforme Figura 2.4, seja em volume, complexidade ou variedade, inicialmente as tecnologias de *Cloud* foram utilizadas para fornecer recursos computacionais e de armazenamento dos quais esses dispositivos careciam. Porém as longas distâncias entre dispositivos IoT e a *Cloud* geram altas latências de transmissão, principalmente em aplicações com grandes volumes de dados, a exemplo do videomonitoramento (envio de imagem e vídeo). A latência típica pode chegar a centenas de milisegundos (VICTOR, 2015). Além disso, a explosão de dispositivos IoT eleva a competição por largura de banda e recursos de *Cloud*, e as vantagens dessa

Figura 2.4: Complexidade dos dados gerados por dispositivos IoT.



Fonte: (VISUALCAPITALIST, 2019).

tecnologia são reduzidas (PU et al., 2019). Portanto, aplicações IoT demandam novas abordagens para consumo de recursos, reduzindo latência e consumo energético.

2.3 Arquiteturas de Borda para Internet das Coisas

Para lidar com o problema enfrentado pelos dispositivos IoT ao realizar transmissões com a *Cloud*, diferentes arquiteturas que atuam na periferia da rede foram desenvolvidas para aproximar a computação desses dispositivos. Essas abordagens de computação na borda da rede aumentam as capacidades de armazenamento e processamento dos dispositivos IoT conectados à Internet e proveem recursos em uma camada intermediária entre os dispositivos finais e a *Cloud* (DOLUI; DATTA, 2017).

Com a presença dos *edge nodes*, ou equipamentos na periferia da rede, o volume de computação realizado nos *Data Centers* é reduzido, pois são executados na camada intermediária, e não na *Cloud*. Isso permite a redução da latência e o suporte à mobilidade, visto a natureza geodistribuída desses equipamentos.

A camada intermediária, entre dispositivos IoT e a *Cloud*, é implementada de diferentes maneiras variando conforme a atuação dos *edge nodes*, os protocolos de comunicação, a rede e os serviços oferecidos. A implementação dessa camada pode ser classificada em quatro tipos principais: *Edge Computing* (EC), *Mobile Edge Computing* (MEC), *Fog Computing* (FC) e *Cloudlet Computing* (CC) (DOLUI; DATTA, 2017) (HECK et al., 2018).

2.3.1 Edge computing

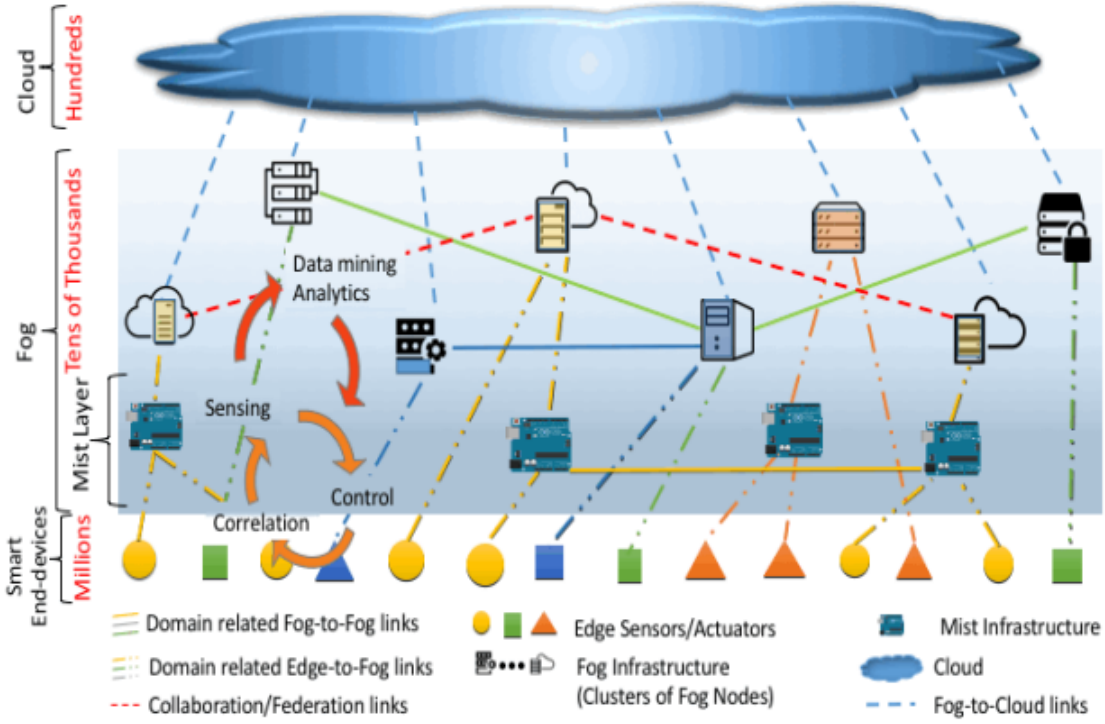
Edge Computing (EC) é um dos paradigmas de computação na borda de rede mais controversos. Frequentemente é citado de modo genérico para se referir a todos os tipos de computação realizados próximos aos dispositivos IoT (DOLUI; DATTA, 2017), porém pesquisas mais recentes classificam EC como um paradigma independente (SHI et al., 2016) (LOPEZ et al., 2015).

A primeira descrição de EC, como um paradigma isolado, foi feita por (VAQUERO; RODERO-MERINO, 2014), na qual foi delineado um cenário distribuído com dispositivos finais ubíquos, equipamentos de rede e servidores de borda (*edge servers*) formando uma *mini-cloud* para a troca de recursos sem a intervenção de *Data Centers* centralizados. Segundo o NIST (IORGA; FELDMAN; BARTON, 2011), EC é uma camada de rede que contempla os dispositivos inteligentes e seus usuários para prover computação local, sistemas de medição ou outros tipos de serviços acessíveis pela rede, sem apresentar comunicação com a camada de FC ou *Cloud*. Pode-se dizer que EC prima pela interação entre os dispositivos IoT sem, necessariamente, oferecer equipamentos de rede de alto poder computacional (HECK et al., 2018).

2.3.2 Fog computing

Fog Computing (FC) é um paradigma de computação de borda de rede descentralizado que preenche a lacuna entre a nuvem e os dispositivos finais. Ela compõe uma camada intermediária entre dispositivos IoT e a *Cloud*, estendendo os serviços de *Cloud* próximo aos dispositivos IoT, permitindo a computação, armazenamento, tomada de decisão e gerenciamento de dados em nós da rede próximos destes dispositivos (BANGUI et al., 2018) (YOUSEFPOUR et al., 2019).

Figura 2.5: Arquitetura de *Fog Computing* segundo o NIST.



Fonte: (IORGA; FELDMAN; BARTON, 2011).

Para o NIST, FC é um paradigma horizontal, físico ou virtual que reside entre os dispositivos inteligentes e a *Cloud* tradicional. Esse paradigma suporta aplicações sensíveis à latência ao prover computação distribuída, ubíqua, escalável e em camadas, armazenamento e conectividade de rede. A Figura 2.5 apresenta este paradigma com suas três camadas. A primeira consiste nos dispositivos IoT que produzem dados e os enviam para o *fog node* mais próximo via comunicação de curta distância (3G/4G, LTE, Wi-Fi). A segunda camada possui os *fog nodes*, que alocam recursos e processam dados colaborativamente. E a terceira camada representa a *Cloud*, utilizada para agregação de dados e *analytics* (BONOMI et al., 2014).

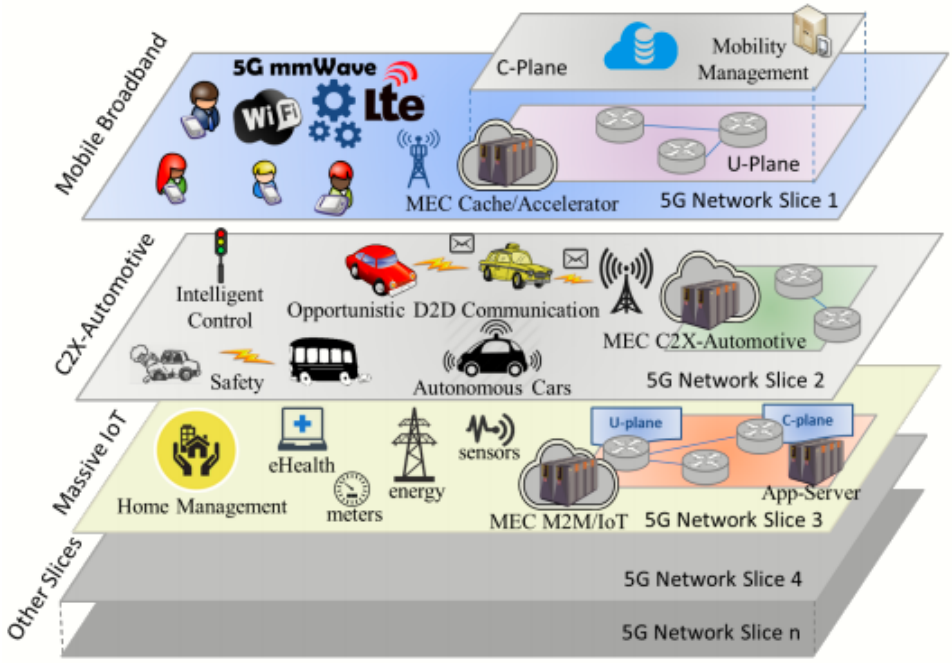
A empresa de consultoria *WinSystems* (WINSYSTEMS, 2019), por sua vez, traz uma diferenciação sobre FC e EC referente à inteligência trazida para a periferia da rede. Em um ambiente FC a inteligência é posicionada na rede local (*Local Area Network*, ou LAN), na rede local. Os dados são transmitidos a um *gateway*, de onde é enviado para nodos de processamento, também locais, e os resultados são retornados à rede. Já em um ambiente EC a inteligência e poder de processamento são posicionados no próprio dispositivo. Para redes com dezenas de milhares de dispositivos IoT, a abordagem FC agrega ao ofertar mais serviços na periferia da rede, sempre permitindo que, em caso de necessidade de recursos adicionais ou serviços não oferecidos na camada intermediária, a

Cloud possa ser acessada.

2.3.3 Mobile edge computing

Mobile Edge Computing (MEC) é um paradigma que leva computação e armazenamento próximo aos dispositivos IoT, porém dedicado a redes com conexão mobile (3G, 4G, LTE ou 5G), podendo ser considerado um caso especial de FC. Os servidores MEC oferecem serviços de *Cloud* dentro da RAN (*Radio Access Network*) e próximo aos dispositivos conectados à RAN (HECK et al., 2018), de modo a reduzir as latências de comunicação. Eles geralmente estão localizados junto de estações base (*Base Station*, ou BS) ou a células agregadoras de sinal de rede móvel. MEC pode se beneficiar da cobertura ubíqua de redes celulares, tornando-se uma peça chave ao suporte de comunicações *machine-to-machine* (M2M) e serviços IoT maduros o suficiente para lidar com aplicações de utilidades, energia, automotivas e cidades inteligentes (TALEB et al., 2017). Os servidores MEC, além de executar as tarefas localmente, podem também encaminhar as solicitações de recursos para a *Cloud*, em *Data Centers* centralizados (MAHMUD; BUYYA, 2016), sempre que necessário.

Figura 2.6: Casos de uso dentro do ecossistema MEC.



Fonte: (TALEB et al., 2017).

A Figura 2.6 apresenta um exemplo de diferentes ecossistemas de rede presen-

tes dentro de uma infraestrutura de rede MEC comum, considerando o papel de MEC em banda larga móvel, sistemas automotivos e serviços IoT com grande volume de dados. A banda larga móvel requer alta capacidade, assegurando que uma aplicação receba a performance apropriada. A plataforma MEC pode armazenar conteúdo na borda da rede, aumentando a capacidade de serviços móveis via descarregamento de tráfego para os equipamentos de borda locais. MEC também pode oferecer diversos outros serviços como aceleração de vídeo ou aplicações baseadas em otimização de performance para assegurar a melhor experiência de banda larga móvel. Já para as aplicações automotivas há a necessidade de latências extremamente baixas e escalabilidade com a rede de borda provendo funções e recursos para que diferentes serviços estejam habilitados próximo aos dispositivos IoT.

Ao mesmo tempo que o paradigma MEC é limitado à infraestrutura de rede móvel, ele traz diversas vantagens. Dentre elas está a integração da computação na arquitetura da rede móvel e a disponibilidade em tempo real de informações sobre a rede que garanta QoS e habilitam aplicações e serviços para o ambiente *mobile* (DOLUI; DATTA, 2017). MEC também tem um forte apelo e apreciação comercial, principalmente pelas operadoras de telefonia, que ao agregar servidores MEC junto às suas estações base de transmissão de dados móveis, podem cobrar seus clientes com base no uso de recursos (PATEL; HU; HéDé, 2014).

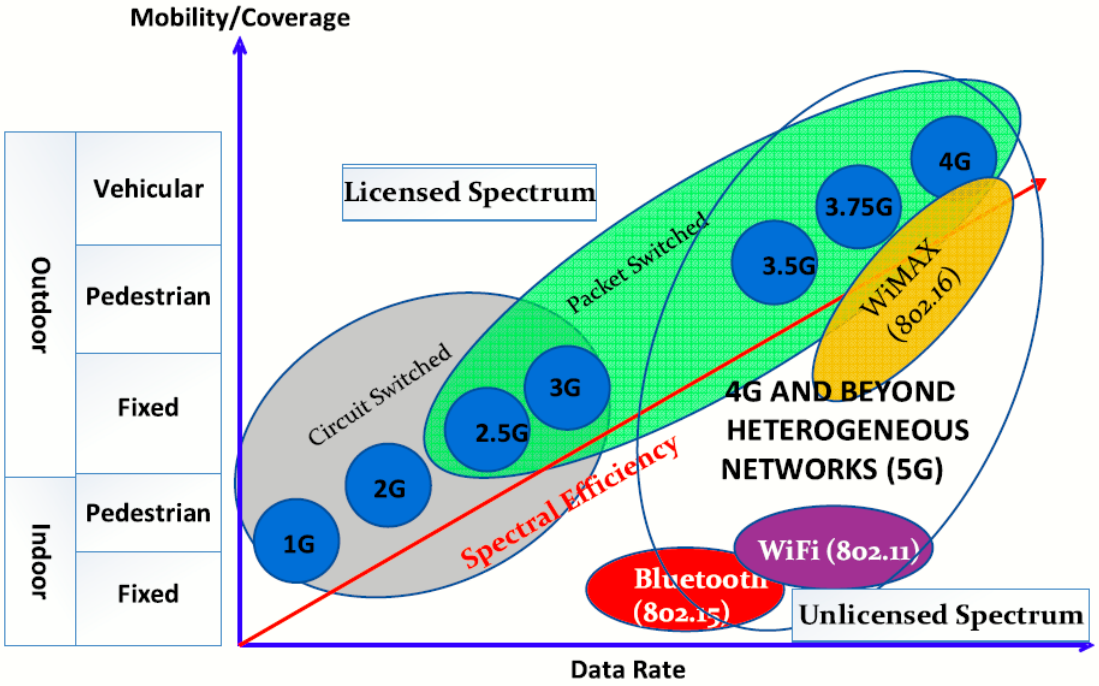
2.3.3.1 Evolução das tecnologias de comunicação mobile

As comunicações sem fio se tornaram uma importante parte do nosso dia a dia. Desde as comunicações por satélite, as transmissões de televisão e rádio avançaram para telefones móveis pervasivos, e as comunicações sem fio transformaram a forma como nossa sociedade funciona. Na Figura 2.7 é apresentada a evolução das tecnologias sem fio, com suas diferentes gerações, em termos de velocidade de transmissão de dados, cobertura do sinal e eficiência espectral (GUPTA; JHA, 2015). À medida que as tecnologias sem fio evoluem, a taxa de transmissão, mobilidade, cobertura e eficiência espectral aumentam.

Conforme a Figura 2.7 as tecnologias 1G e 2G utilizam chaveamento de circuito, enquanto 2.5G e 3G utilizam chaveamento por circuito e chaveamento por pacote e as tecnologias 3.5G até 5G utilizam chaveamento por pacote. Além disso, a figura também diferencia entre espectros licenciados, que requerem participação em licitação para concessão de uso pelo poder público, e não licenciados, espectros de frequência de uso livre.

Todas as gerações em evolução utilizam espectros de frequência licenciados, enquanto *WiFi*, *Bluetooth* e *WiMAX* utilizam espectros não licenciados.

Figura 2.7: Evolução das tecnologias sem fio.



Fonte: (GUPTA; JHA, 2015).

A primeira geração (1G) de tecnologias para telefones móveis foi anunciada no início dos anos 1980. Sua taxa de transferência de dados é de até 2.4kbps. Todas as comunicações sem fio eram centradas em voz e a tecnologia utilizada era analógica, já que à época os sistemas de rádio digitais eram muito caros para construção (RODRIGUEZ, 2015). Ela possuía muitos problemas de qualidade de sinal e segurança, visto que as conversas eram gravadas e reproduzidas em torres de rádio, sujeitas à escuta por terceiros (GUPTA; JHA, 2015).

A segunda geração (2G) foi introduzida no início dos anos 1990. Esta foi a primeira geração na qual foi utilizada a tecnologia digital para telefones móveis, ou seja, migrar do 1G para o 2G significou migrar do sistema analógico para o sistema digital. GSM (*Global Systems for Mobile Communications*) foi o primeiro sistema 2G, usado principalmente para comunicação em voz e com taxa de transmissão de até 64kbps. A bateria de telefones com tecnologia 2G durava mais, pois os sinais de rádio possuíam menor potência. Essa geração também proporcionou serviços de troca de mensagens, como SMS (*Short Message Service*) e *e-mail*, embora as comunicações sem fio focassem majoritariamente em transmissão de voz. Algumas tecnologias foram introduzidos posteriormente, como o GPRS (*General Packet Radio Services*), utilizado na Europa, e o CDMA (*Code*

Division Multiple Access), nos EUA, que aliado à tecnologia 2G pôde prover taxas de transmissão de até 144kbps. Essa melhoria ficou conhecida como 2.5G, pois utilizava sistemas 2G juntamente com uma mescla entre chaveamento por pacotes e chaveamento por circuitos.

A terceira geração (3G) começou a ser utilizada no início dos anos 2000 e foi a primeira a oferecer serviços para voz e dados móveis, com transmissões de dados de até 2Mbps. Para essa geração foi desenvolvido pela primeira vez um padrão internacional pela ITU (*International Telecommunication Union*), em contraste às gerações anteriores. Melhorias adicionais foram o *roaming* global e a qualidade do sinal de voz. Sua principal desvantagem é o maior consumo energético em relação a modelos 2G. 3G explora a tecnologia WCDMA (*Wideband Code Division Multiple Access*), operando nos modos FDD (*Frequency Division Duplex*) e TDD (*Time Division Duplex*), e é possível dizer que com a migração de 2G para 3G houve a evolução de sistemas baseados em voz para sistemas baseados em dados (RODRIGUEZ, 2015). Com o surgimento das redes 3G, novas tecnologias como HSUPA/HSDPA (*High Speed Uplink/Downlink Packet Access*) e EVDO (*Evolution-Data Optimized*) estabeleceram umas geração intermediária entre 3G e 4G, chamada de 3.5G, que melhorou as transmissões de dados entre 5-30Mbps (ROMERO et al., 2004).

A quarta geração (4G) chegou em 2011 e trouxe consigo melhora nas comunicações sem fio ao implementar uma solução de alta velocidade de transmissão de dados, completa e confiável, baseada no IP (*Internet Protocol*). Vozes, dados e multimídia são entregues a velocidades muito superiores às do 3G, além da melhor cobertura do sinal, espectro de frequências suplementar e acesso de alta velocidade a serviços de vídeo sob demanda, compartilhamento de arquivos *peer to peer* e serviços *web*, inviáveis nas gerações predecessoras. Dois sistemas 4G foram desenvolvidos, o WiMAX (*Worldwide Interoperability for Microwave Access*) nos EUA, usando OFDM (*Orthogonal Frequency-Division Multiplexing*), uma evolução do WiFi, e o outro foi o LTE, desenvolvido após o WiMAX. Ambos são muito semelhantes, mas o LTE teve mais aceitação pelas companhias telefônicas e se consolidou. Ele apresenta taxas de transmissão de dados de até 100-200Mbps, mas por não ser uma implementação pura do 4G e apresentar características inferiores, alguns autores referem-se à tecnologia como uma geração intermediária entre o 3G e o 4G, chamada de 3.75G (GUPTA; JHA, 2015). Melhorias no LTE levaram ao LTE-A (*Long Term Evolution Advanced*) com taxas de *download* de 100Mbps para casos de altíssima mobilidade (até 360 km/h) e de 1Gbps para uso estacionário ou para uso

por pedestres. A migração do 3G para o 4G permitiu uma mudança de taxas de transmissão de baixa velocidade para Internet para taxas de transmissão de alta velocidade para consumo de vídeo móvel.

Uma nova geração de sistemas celulares aparece a cada 10 anos, e espera-se que a próxima geração de sistemas celulares, a quinta geração (5G), seja padronizada entre em plena produção ao longo de 2021 (RODRIGUEZ, 2015). É consenso na academia e na indústria que o 5G deve entregar experiência de Internet aos dispositivos móveis semelhante à de fibra óptica, com taxas de transmissão de até 10Gbps para condições estáticas ou de pouco movimento e de 1Gbps para dispositivos móveis em alta velocidade (com velocidades superiores a 300 km/h). Com o 5G as comunicações sem fio em redes móveis terão performance significativamente mais elevada, em termos de taxas de transmissão de dados, latências e eficiência energética (BASSOLI; RENZO; GRANELLI, 2017).

Para o futuro de uma sociedade conectada, todos e tudo será interconectado - pela *Internet de Tudo* (*Internet of Everything*, ou IoE) - e dezenas ou centenas de dispositivos IoE servirão a cada pessoa. A infraestrutura celular do 5G e seu inerente suporte para *Big Data* permitirá que as cidades sejam definitivamente inteligentes. Dados serão gerados por pessoas e máquinas e analisados em tempo real, gerando inferências sobre os hábitos e preferências de trânsito das pessoas nas estradas, e monitoramento de saúde para pacientes e idosos. 5G é a tecnologia habilitadora para o pleno funcionamento dos carros autônomos, permitindo comunicação rápida, segura e eficiente entre os veículos em movimento.

2.3.4 Cloudlet computing

Não tão difundido na literatura, mas também presente, o paradigma de *Cloudlet Computing* foi originalmente desenvolvido por (SATYANARAYANAN et al., 2009) como um computador ou conjunto de computadores de alta performance conectados à Internet e fixados na periferia da rede para descarregar a execução de aplicações. Porém, ele apresenta certas limitações se comparado a FC. *Cloudlets* são geralmente acessadas por redes WiFi, dando a eles cobertura e suporte à mobilidade limitados, além de apenas trazerem os recursos de *Cloud* para perto da periferia da rede, ao invés de interagir com a *Cloud* (HECK et al., 2018). Para solucionar esse problema, Satyanarayanan et al. (2015) modificaram a estrutura de *Cloudlets* para uma nova configuração de três camadas, compostas por dispositivos IoT, *Cloudlets* interconectados e a *Cloud*. Com a nova versão os

Cloudlets passam a trabalhar como *fog nodes*.

2.4 Green Computing

Green Computing (GC) refere-se à prática de utilizar recursos computacionais mais eficientemente enquanto a performance de um equipamento ou sistema é mantida ou melhorada (HARMON; AUSEKLIS, 2009) (ROSSI et al., 2014). Práticas como gerenciamento de consumo, virtualização, melhoramento dos sistemas de refrigeração, reciclagem, descarte apropriado de resíduos eletrônicos e otimização da infraestrutura de TI são alguns requisitos de sustentabilidade para GC.

O rápido crescimento de modelos de negócios baseados na Internet e o uso em larga escala de serviços virtualizados em nuvem foram gatilhos importantes para o surgimento de GC. Nos últimos anos houve um expressivo aumento no consumo de energia elétrica devido à expansão dos *Data Centers* e a preocupação com os impactos ambientais e a geração de carbono impulsionaram estudos na área. Estudos recentes mostram, que os custos com a energia utilizada para departamentos de TI pode chegar a 50% do total de custos com energia de uma organização (HARMON; AUSEKLIS, 2009). Estratégias *green* trazem o benefício de diminuir os custos financeiros e o impacto no meio ambiente, agregando valor aos clientes, ao negócio e à sociedade.

Há também estudos que associam GC a tão somente o uso de fontes renováveis de energia, como em (ZHANG et al., 2018). Porém, neste trabalho a definição de GC utilizada é aquela que visa a redução dos custos energéticos do sistema, de modo a tornar a arquitetura mais eficiente e com menor geração de emissões atmosféricas.

2.5 Consumo de Energia em CPUs

Nesta sessão são apresentados alguns conceitos sobre eletricidade, importantes para entender as principais relações entre corrente, tensão e potência de um sistema. Esses conceitos formam a base para explorar com mais detalhes como se dá o consumo de energia elétrica nos núcleos de processamento de uma CPU. Posteriormente é explicado como ocorre a dissipação de energia em nos núcleos de uma CPU, e qual equação é utilizada para calcular a potência de dissipação energética. Por fim, a técnica de DVFS é apresentada como uma alternativa para minimizar a potência energética de um núcleo de

processamento quando em operação.

2.5.1 Eletricidade básica: potência, tensão, corrente e resistência

A potência (P) mede a taxa de energia transferida em um sistema, ou seja, mede quanto de energia é transferida. A energia transferida é o *watt*, que em termos elétricos corresponde à taxa com que um trabalho é realizado quando uma corrente elétrica (A) de um *ampere* flui por sistema cuja tensão (V), ou diferença de potencial, seja de um *volt* (SEARS, 2009). A potência padrão é dada em *watt*, que é uma unidade de potência. A equação de potência de um equipamento ou sistema considera sua tensão de alimentação elétrica e a corrente que passa por ele:

$$P = A * V \ [Watt] \tag{2.1}$$

Em sistemas residenciais é muito comum utilizar uma outra unidade de medida, o kWh. Diferentemente do *watt*, que é uma unidade de potência, o kWh mede consumo de energia. Seu consumo é dado no tempo e a base de tempo é uma hora. Por exemplo, se um chuveiro elétrico possui potência de 5.500 *W* e for utilizado por 3 horas continuamente, o consumo total será de 16,5 *kWh*, ou seja, $(5.500W/1.000) * 3h$. Há sistemas, entretanto, que utilizam a unidade de medida Ws, que é a quantidade de potência consumida em um segundo. Essa unidade de medida é utilizada na análise de resultados no Capítulo 7.

De modo geral, o *watt* mede a quantidade de trabalho realizado por segundo, também sendo representada pelo trabalho em *joules* por segundo (SEARS, 2009). Tomando por base as unidades de *joules* por segundo para o *watt*, podemos entender sua relação com a corrente e a tensão. A corrente é a quantidade de carga elétrica transportada de um ponto a outro em um segundo. Um *ampere* corresponde ao transporte de $6,25 * 10^{18}$ elétrons em um segundo, ou um *coulomb* (C) em um segundo. Já a tensão corresponde à transmissão entre dois pontos de energia, em *joules*, por carga elétrica, em *coulombs*. Ou seja, um *volt* transmite um *joule* de energia para cada transporte de um *coulomb* entre dois pontos. Desse modo, potência, corrente e tensão podem ser representados por:

$$P = \frac{Energia}{Tempo} \ [Joule/segundo] \tag{2.2}$$

$$A = \frac{Carga}{Tempo} \ [Coulomb/segundo] \tag{2.3}$$

$$V = \frac{Energia}{Carga}\ [Joule/Coulomb] \tag{2.4}$$

Aplicando as Equações 2.3 e 2.4 na Equação 2.1, as unidades restantes são em *joule* por segundo, ou seja, *watts*.

A resistência elétrica (R) é outra componente com relação direta na potência de um sistema. Em termos técnicos, a resistência elétrica é a capacidade de um corpo se opor à passagem de corrente elétrica, mesmo quando há um diferencial de potencial (tensão) aplicado ao sistema. Todo material possui uma resistência elétrica própria e havendo passagem de corrente pelo material ocorre o *Efeito Joule*, no qual parte da energia é dissipada na forma de calor (SEARS, 2009). A dissipação de energia, ou potência, é proporcional à resistência. Sua unidade de medida é o *Ohm*.

A relação entre corrente, tensão e resistência é expressa pela *Lei de Ohm*, que afirma que a resistência elétrica em um condutor corresponde à divisão do diferencial de potencial elétrico aplicado entre os terminais do material pela corrente elétrica que flui por ele. Assim, também podemos dizer que a tensão aplicada entre dois terminais é diretamente proporcional à corrente elétrica que o percorre. As equações que expressam essas relações são dadas por:

$$R = \frac{V}{A}\ [Ohm] \tag{2.5}$$

$$A = \frac{V}{R}\ [Ampere] \tag{2.6}$$

$$V = \frac{R}{A}\ [Volt] \tag{2.7}$$

Por fim, podemos aplicar a Lei de Ohm à potência, expressa na Equação 2.1. Desse modo, a potência de um sistema pode ser determinada em termos da tensão e da resistência, conforme a Equação 2.8:

$$P = \frac{V^2}{R}\ [Watt] \tag{2.8}$$

2.5.2 Potência em núcleos de processamento

O consumo de energia tem sido um assunto de grande preocupação em relação a dispositivos móveis alimentados por baterias, tais como os *smartphones*, e a CPU é uma das principais fontes de consumo de energia nestes dispositivos (CARROLL; HEISER, 2010). À medida que as tecnologias *multicore* e *multicpu* se tornaram mais populares nos *smartphones*, o consumo de energia em CPUs ganhou ainda mais significância, por contribuir com um percentual cada vez maior no consumo total (ZHANG et al., 2015). Portanto, mensurar e gerenciar o consumo energético de uma CPU é um das tarefas mais importantes em dispositivos *multicore*.

Em circuitos CMOS (*Complementary Metal-Oxide-Semiconductor*), conjuntos de transistores presentes nas CPUs, a dissipação de potência pode ser tanto estática como dinâmica. A potência dinâmica é a energia consumida pela CPU para realizar as mudanças de estado dos transistores durante a computação. Já a potência estática provêm majoritariamente de correntes de fuga, originária dos vazamentos de corrente inerentes aos transistores de silício. Portanto:

$$P_{CPU} = P_{dinamica} + P_{estatica} \tag{2.9}$$

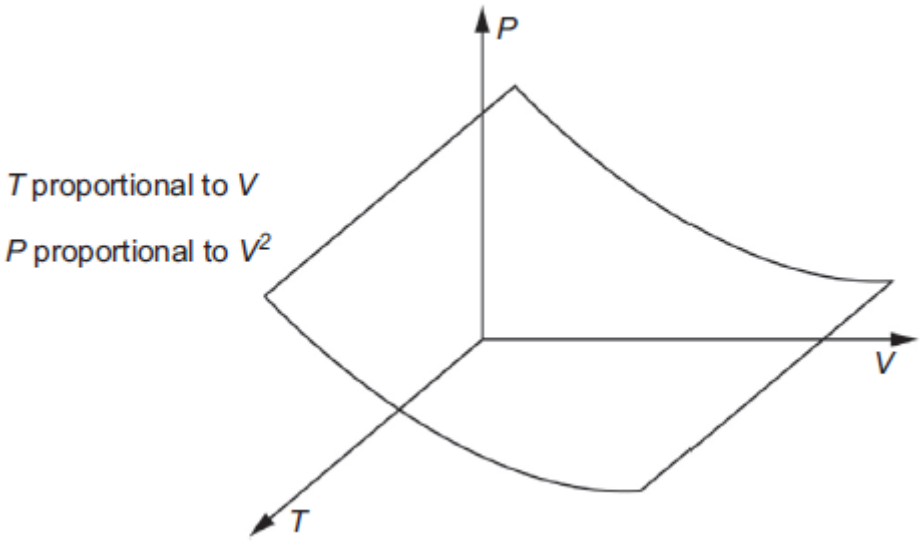
A potência estática é bastante dependente de técnicas de projeto de circuitos e tecnologias de processo, sendo que projetos bem desenhados podem tornar a contribuição desse tipo de energia negligenciável (BURD; BRODERSEN, 1996) (JIN; GOTO, 2012a). A potência dinâmica pode ser dividida em potência de curto-circuito e potência de carga. A potência de curto-circuito é gasta pelos transistores quando há uma mudança de estado lógico e por um breve instante o transistor aterra, causando um curto-circuito entre a fonte de alimentação e o terra do sistema (VOGELEER et al., 2014). Para circuitos bem projetados a potência de curto-circuito é muito pequena, representando de 5 a 10% da potência dinâmica e pode ser omitida em análises de consumo (BURD; BRODERSEN, 1996).

$$P_{CPU} = P_{carga} + P_{curto-circuito} + P_{fuga} \tag{2.10}$$

A potência de carga é gerada ao carregar as portas dos capacitores, de modo a permitir a comutação de estado das portas lógicas em circuitos CMOS, e custa ao sistema a maior parte da potência dissipada na CPU. Ela é aproximadamente proporcional ao

quadrado da tensão (voltagem) e a tensão é aproximadamente proporcional à frequência de operação. As principais relações existentes entre as variáveis são $P \propto V^2$ e $V \propto f$, no qual P é a potência dinâmica, V é a tensão e f é a frequência de operação (SARANGI; GOEL; SINGH, 2018).

Figura 2.8: O espaço de operação entre tensão de alimentação (V), velocidade de operação (T) e energia dinâmica (P).



Fonte: (GUPTA; JHA, 2015).

Na Figura 2.8 é apresentado um gráfico sobre os pontos de operação de uma CPU, conforme as relações expressas anteriormente. É possível observar que o tempo de processamento é diretamente proporcional à tensão de alimentação da CPU. Como a tensão é proporcional à frequência de operação, pode-se afirmar que o tempo de processamento também varia proporcionalmente à frequência da CPU. Já a potência de carga, expressa pela letra P varia quadraticamente à tensão da CPU, e, consequentemente, à frequência de operação.

Devido à pequena contribuição no consumo energético da potência de curto-circuito e da potência de corrente de fuga, é possível definir a potência aproximada da CPU com a potência de carga, uma fonte de potência dinâmica. Na literatura, a potência de carga é definida como $\propto CV^2f$, na qual C é capacitância de chaveamento dos transistores, V é a tensão de alimentação e f é a frequência de operação da CPU (INTEL, 2004) (LIU et al., 2007):

$$P_{dinamica} = C * V^2 * f \ [Watt] \tag{2.11}$$

De posse da equação de energia dinâmica representada por 2.11, podemos saber

qual o efetivo consumo da CPU, bastando escolher uma janela de tempo para monitorar o consumo. A Equação 2.12 expressa essa relação, na qual $E_{consumida}$ é a energia consumida, $P_{dinamica}$ é a potência dinâmica da CPU em *watts* e T_{exec} é o tempo de execução da tarefa em segundos (LIU et al., 2007).

$$E_{consumida} = P_{dinamica} * T_{exec} \, [Watt * segundo] \tag{2.12}$$

2.6 Tempo de execução em CPUs

A equação de performance, utilizada para o cálculo do tempo decorrido em uma execução, é um produto de três fatores, a quantidade de instruções do programa (I), a média de ciclos de *clock* por instrução (CPI) e o tempo do ciclo de *clock* (T) (TANENBAUM; AUSTIN, 2012). A equação para o cálculo de tempo é dada por:

$$T_{tarefa} = I * CPI * T_{clock} \tag{2.13}$$

Para (SARANGI; GOEL; SINGH, 2018) a Equação 2.13 pode ser reescrita com a quantidade total de ciclos de processamento (CT) para a execução da tarefa e a frequência de *clock* (f) da CPU, que é inversamente proporcional ao tempo de ciclo de *clock*. A nova equação é dada por:

$$T_{tarefa} = \frac{CT}{f} \tag{2.14}$$

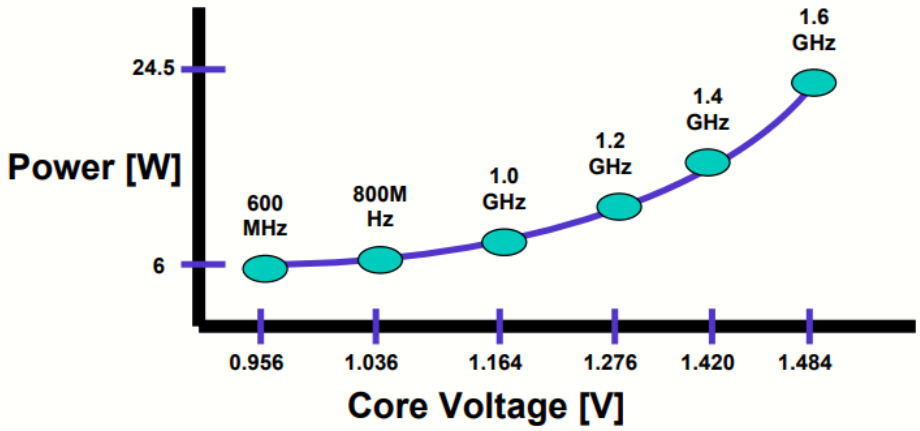
A Equação 2.14 é utilizada no modelo de custo do sistema para calcular o tempo de execução das tarefas em um núcleo de processamento.

2.7 Técnica DVFS

DVFS (*Dynamic Voltage and Frequency Scaling*) é um método bem difundido para reduzir o consumo energético em dispositivos móveis modernos que atua ao nível de um único núcleo de processamento (SARANGI; GOEL; SINGH, 2018). DVFS pode ser facilmente implementado em sistemas de tempo real sob restrições de tempo. Ele atua durante a execução das tarefas mantendo frequência de núcleo e tensão reduzidas com o objetivo de reduzir a dissipação de potência, e, consequentemente, o consumo de energia

(CHEN et al., 2018).

Figura 2.9: Potência vs. tensão da CPU para o processador *Intel® Pentium® M* de 1.6GHz.



Fonte: (INTEL, 2004).

Na Figura 2.9 é possível observar um estudo feito pela empresa Intel, na qual a técnica de DVFS foi aplicada. Ao alterar tensão e frequência a potência altera de acordo. Valores mais altos geram maior potência de operação, porém a tarefa é executada mais rapidamente. É um *trade-off* que deve ser considerado de acordo com as necessidades da aplicação, dando prioridade ao consumo ou ao tempo de execução.

Há diferentes técnicas de DVFS que são muito eficazes quando possuem informações diversas sobre as aplicações, como tempo de chegada, *deadline* e carga de trabalho (JIN; GOTO, 2012b). Porém, estudos mostram que a potência dinâmica em um núcleo de processamento é dependente principalmente da frequência de *clock*. Ela apresenta um comportamento convexo, ou seja, há um ponto de operação ótimo da frequência do núcleo de modo que o consumo energético seja mínimo (VOGELEER et al., 2014). Desse modo, reduzir a frequência e a tensão são objetivos da técnica de DVFS para reduzir a energia dinâmica, daí o nome *"dynamic voltage and frequency scaling"*.

2.8 Algoritmos de Escalonamento de Tarefas

O escalonamento de tarefas é uma das partes mais importantes de ambientes Cloud Computing, Fog Computing e Mobile Edge Computing (WADHONKAR; THENG, 2016) (PHAM; HUH, 2016) (YU; WANG; GUO, 2018). O objetivo do escalonamento de tarefas é, principalmente, otimizar a utilização de recursos e reduzir o tempo de finalização das tarefas, alocando às tarefas os recursos necessários para que execução ocorra na menor

janela de tempo possível. Os algoritmos de escalonamento podem ser divididos em dois grupos, os escalonadores estáticos e os escalonadores dinâmicos.

- Escalonamento Estático: Todas as informações das tarefas e recursos do sistema são de conhecimento do escalonador antes da execução. Ele possui um menor *overhead* de operação (WADHONKAR; THENG, 2016).
- Escalonamento Dinâmico: As informações sobre as características das tarefas não são de conhecimento prévio do escalonador. O tempo de execução das tarefas pode não ser conhecido. Possui um maior *overhead* de operação (WADHONKAR; THENG, 2016).

Neste trabalho é utilizado o escalonamento estático, visto que o escalonador proposto no Capítulo 6 possui conhecimento prévio sobre o tamanho das tarefas, seu tempo de execução até a finalização, bem como sobre os recursos de processamento disponíveis na rede e suas características.

A seguir são apresentados alguns dos principais algoritmos de escalonamento descritos na literatura para o escalonamento de tarefas em ambientes Cloud, Fog e MEC segundo (WADHONKAR; THENG, 2016), (GHANBARI; OTHMAN, 2012), (SELVARANI; SADHASIVAM, 2010) e (SINGH JAGBEER; SINGH, 2010).

- *Improved Cost Based Algorithm*: Apresenta uma melhoria ao algoritmo de escalonamento tradicional baseado em custo para realizar a adequada alocação de recursos. As tarefas são agrupadas de acordo com o poder de processamento dos recursos disponíveis na rede;
- *Earliest Feasible Deadline First*: A tarefa com o menor *deadline* é escalonada. Toda vez que uma tarefa é concluída ou uma nova é criada a fila de tarefas é varrida até encontrar o processo com o *deadline* mais próximo. Esse processo será o próximo a ser escalonado para execução;
- *A Priority based Job Scheduling Algorithm*: A prioridade da tarefa é considerada para a alocação de recursos, conforme as características das tarefas e dos recursos;
- *Priority Based Earliest Deadline First Scheduling Algorithm*: Aqui são utilizados dois algoritmos, o primeiro é o *Earliest Deadline First* e o segundo é o *Priority Based Scheduling Algorithm*. O algoritmo foca na alocação de recursos e utilização de memória. A abordagem utilizada permite reduzir o tempo de execução de tarefas interrompidas antecipadamente (*preempted tasks*) e elas podem ser escalonadas de modo eficiente. Este algoritmo resolve o problema de tempo de espera de tarefas

interrompidas antecipadamente. A fila de tarefas em espera é introduzida e processa as tarefas preemptadas.

Conforme os algoritmos listados, é possível perceber diferentes abordagens para tarefas que possuem *deadline* de conclusão e necessitam de recursos mais rapidamente, bem como as prioridades estabelecidas para cada tarefa e recursos, que ditam o destino das alocações realizadas pelo algoritmo de escalonamento.

2.9 Considerações Sobre o Capítulo

Neste capítulo foram apresentados diversos conceitos que são utilizados ao longo deste trabalho. As arquiteturas de borda permitem que o leitor tenha uma visão ampla sobre seus tipos, definições, características, utilização e objetivos. Uma delas, a arquitetura *Mobile Edge Computing*, é utilizada na arquitetura do sistema, juntamente com a *Cloud*.

Os conceitos de eletricidade, cálculo de potência e energia consumida, bem como a forma como é calculado o tempo de execução de tarefas com base na carga computacional, são conceitos utilizados nas equações do modelo de custo do sistema. Além destas, a técnica de DVFS permite que o modelo tenha flexibilidade, alterando variáveis de frequência de operação e tensão de alimentação das CPUs de modo a reduzir o consumo energético ou agilizar a execução de uma tarefa.

Por fim, os algoritmos de escalonamento apresentados formam a base para desenvolver a dinâmica de funcionamento do escalonador do tarefas do sistema, proposto no Capítulo 6.

3 TRABALHOS RELACIONADOS

Neste capítulo é apresentada a metodologia de pesquisa em duas etapas utilizada para busca de artigos, bem como a seleção final de um conjunto de trabalhos relacionados a esta dissertação e seus detalhamentos. O detalhamento dos trabalhos relacionados é posteriormente analisado no Capítulo 4 para o levantamento de características que irão auxiliar na definição da proposta de modelo do sistema.

3.1 Metodologia de Pesquisa

A metodologia utilizada para a seleção de artigos do estado da arte alinhados com a proposta que é feita neste trabalho é apresentada com detalhes a seguir. Ela foi desenvolvida em duas etapas, a primeira a focada na busca de *surveys* com temas mais amplos e a segunda focada na busca de artigos com temas semelhantes aos da proposta deste trabalho. Na segunda etapa é realizada a seleção final dos artigos do estado da arte.

A construção dessa metodologia em duas etapas foi baseada nas metodologias adotadas nos trabalhos (KITCHENHAM; CHARTERS, 2007), (KESHAV, 2007) e (THAKUR; GORAYA, 2017).

3.1.1 Escolhas iniciais

A seleção de artigos iniciou com a escolha de editoras conceituadas para a realização das buscas. As editoras escolhidas foram Elsevier, Springer, IEEE e ACM, devido a contatos prévios que este autor teve com as referidas editoras e também visando otimizar a quantidade de locais de busca. Além destas, também foram realizadas buscas na plataforma *Google Scholar*, em busca de artigos não listados nas editoras citadas e que pudessem apresentar informações relevantes à pesquisa desenvolvida nesta dissertação.

Foram estabelecidas algumas etapas de pesquisa de artigos, nas quais, objetivos e conjuntos de palavras de busca específicos foram definidos.

Para todas as etapas de busca foram estabelecidos alguns critérios, de modo a uniformizar a metodologia de busca e limitar os resultados obtidos. Foi definido o ano de 2013 para iniciar a varredura por artigos, pois em buscas prévias observou-se que a partir desse ano houve aumento no número de publicações nos temas pesquisados. A

busca ocorreu com a utilização do critério de relevância da publicação para o ordenamento dos resultados e a escolha dos artigos se deu com base no seu título, e, posteriormente, resumo e palavras chaves. Apenas as 5 a 8 primeiras páginas de resultados de busca foram consideradas em cada local pesquisado.

3.1.2 Primeira etapa

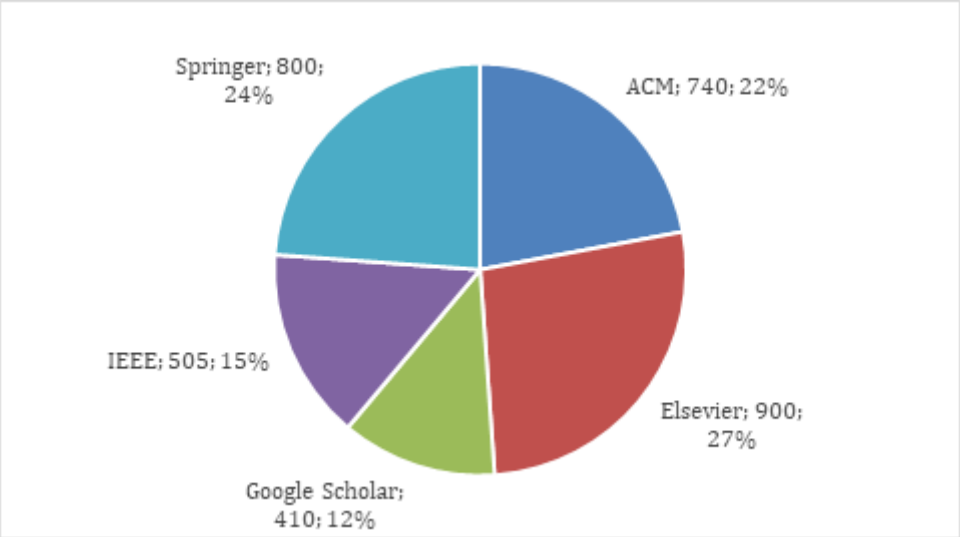
Esta primeira busca teve como objetivo encontrar *surveys* sobre diferentes tópicos vinculados a temas como consumo de energia, eficiência no uso e alocação de recursos e latência em *Cloud*, de modo a aprofundar os conhecimentos do autor nas referidas áreas. Para a definição dos primeiros conjuntos de palavras de busca foi realizada uma pesquisa prévia sobre tecnologias vinculadas a dispositivos presentes na periferia da rede que tenham como premissa o uso racional de recursos, melhora de QoS e redução no consumo de energia. As tecnologias que lidam atualmente com tais problemas situam-se majoritariamente nos campos de pesquisa de *Cloud Computing*, *Edge Computing*, *Fog Computing* e *Mobile Edge Computing*. Também há citações a *Micro Cloud* para determinadas aplicações, e, portanto, também foram consideradas. Os conjuntos de palavras dos referidos campos de pesquisa foram incluídos no conjunto de busca. As buscas foram realizadas nas plataformas das editoras, bem como no buscador do *Google Scholar*. A primeira etapa de busca utilizou o seguinte conjunto de palavras:

- *Micro cloud survey*;
- *Cloud computing survey*;
- *Edge computing survey*;
- *Fog computing survey*;
- *Mobile edge computing survey*.

Optou-se, inicialmente, pela busca de *surveys* e não artigos simples somente, pois os *surveys* já pressupõe uma pesquisa extensiva pelo autor na área e aglutina muitas informações relevantes em um só documento. Desse modo, um conjunto restrito de *surveys* foi selecionado.

No total foram pesquisados 3.355 artigos distribuídos entre as cinco fontes buscadas, conforme Figura 3.1. A quantidade de artigos pesquisados em cada editora e no *Google Scholar* foram diferentes, pois a quantidade de resultados de busca por página de cada editora varia. Para as editoras Springer e ACM a quantidade de resultados por

Figura 3.1: Quantidade de artigos pesquisados na primeira etapa de busca.



página foi de 25, para Elsevier e IEEE foi 20 e para o *Google Scholar* foram 10. Assim, como foram consideradas as primeiras 5 a 8 páginas de busca, as quantidades de artigos retornados pelos buscadores foram levemente diferentes, variando entre 100 a 160 para editoras com 20 resultados por página, 125 a 200 para editoras com 25 resultados por página e de 50 a 80 para o *Google Scholar*.

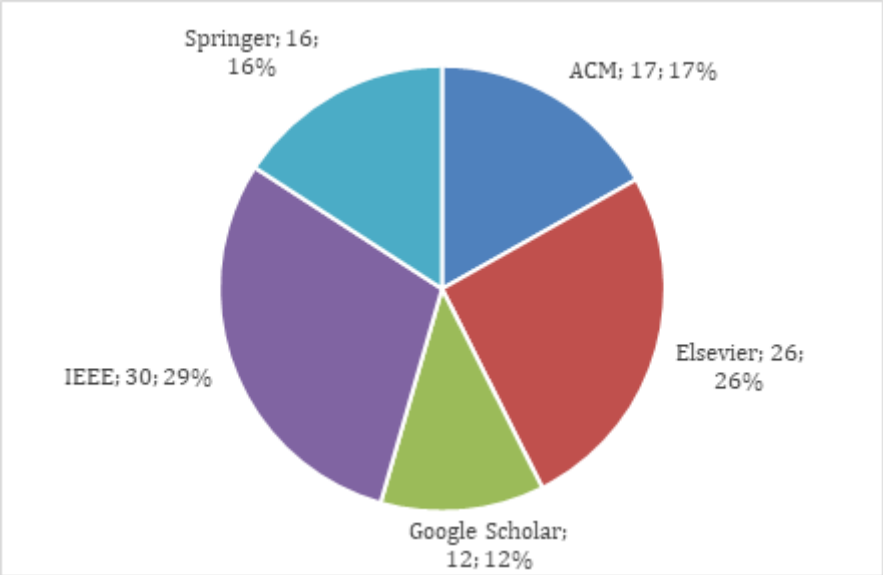
Vale notar que o buscador do IEEE foi o que retornou a melhor qualidade de resultados de busca, pois, embora tenha, para alguns conjuntos de palavras de busca, retornado poucos resultados, estes tinham forte afinidade com as palavras pesquisadas, visto os artigos retornados na busca, fenômeno menos aparente nos buscadores das demais editoras. O buscador da editora Elsevier também é bastante otimizado e apresentou muitos resultados interessantes, além de possuir na página específica de cada artigo uma *feature* de artigos relacionados que auxiliam a obter mais artigos pertinentes ao assunto pesquisado. Em contrapartida, o buscador da Springer apresentou os resultados com a menor afinidade com as palavras de busca. Dentre os resultados listados havia diversos capítulos de livros com conteúdo abrangente e pouco relacionados às palavras de busca. O buscador do *Google Scholar*, por sua vez, apresentou uma boa qualidade de resultados, porém a maioria dos resultados obtidos redirecionava para as páginas das quatro editoras. Assim, artigos encontrados via buscador do *Google Scholar*, mas hospedados na página de uma das editoras, foram contabilizados para a respectiva editora. Utilizando o buscador do *Google Scholar* foram encontrados muitos artigos na plataforma *arXiv.org* da *Cornell University Library*[1], que no momento da consulta possuía acesso aberto a 1.463.669 impressões eletrônicas em Física, Matemática, Ciência da Computação, Biologia Quantitativa, Finança

[1](ARXIV.ORG, 2019). Acessado em: 08/11/2019.

Quantitativa, Estatística, Engenharia Elétrica e Sistemas Científicos e Economia. Os artigos encontrados na plataforma *arXiv.org* não estavam presentes nos portais das editoras.

Dentre os artigos avaliados foram selecionados apenas *surveys* que abordavam os conteúdos descritos nas palavras de busca, totalizando 101 artigos. Na Figura 3.2 é possível observar a quantidade final de artigos selecionados nesta primeira etapa de busca, separados por origem. Os artigos indicados como sendo encontrados pelo *Google Scholar* incluem apenas aqueles que não são originários das outras 4 editoras categorizadas, o que explica a baixa participação no resultado. O IEEE foi o portal no qual foi obtida a maior quantidade de artigos, visto a forte correlação entre as palavras de busca e os resultados do buscador, bem como alguns artigos que foram adicionados à sua contagem final devido as buscas no *Google Scholar*.

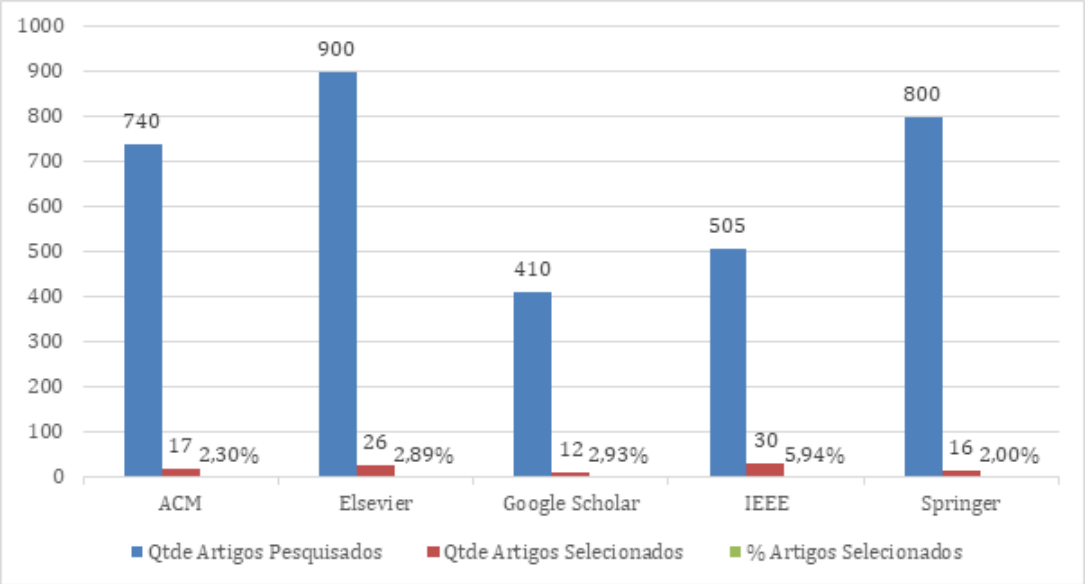
Figura 3.2: Quantidade de artigos selecionados na primeira etapa de busca.



Na Figura 3.3 é apresentada a relação entre a quantidade de artigos pesquisados em cada fonte e a quantidade de artigos selecionados. O percentual de artigos selecionados foi calculado com relação à quantidade total de artigos pesquisados em cada fonte. Observa-se que o portal do IEEE foi o que resultou em mais artigos selecionados, seja em valor absoluto ou percentual frente ao total de artigos pesquisados. Isto apenas reforça a constatação já mencionada, sobre a qualidade do buscador em associar os resultados de busca às palavras de busca pesquisadas.

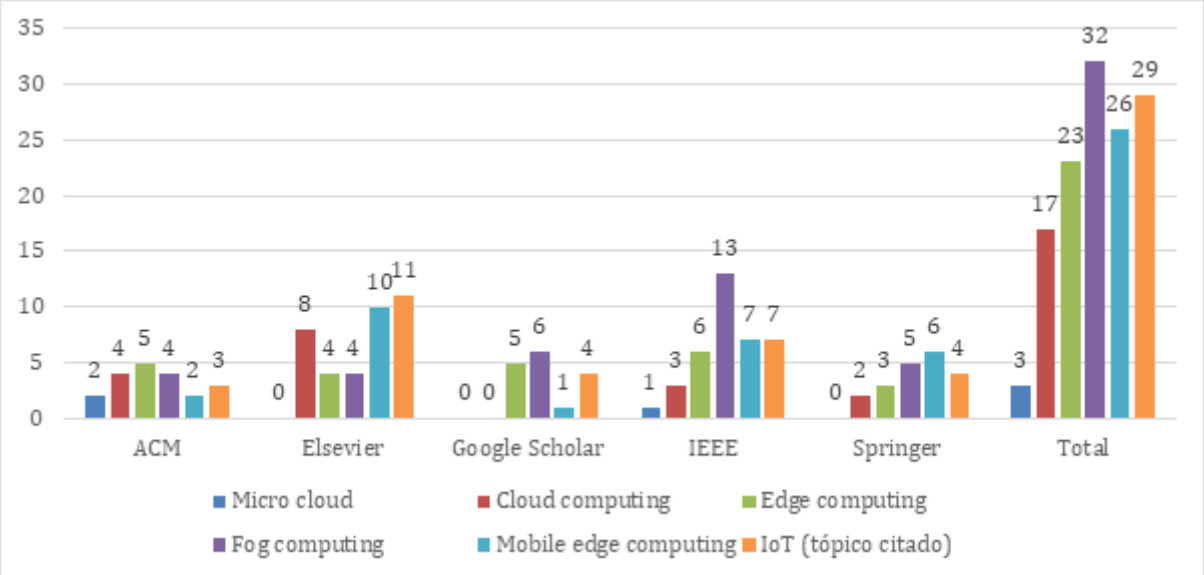
Na Figura 3.4 são apresentadas as quantidades de artigos selecionados de cada editora com base nas palavras de busca. Nota-se que no acumulado total, representado pelas barras verticais mais à direita do gráfico, os tópicos vinculados a FC, EC, IoT e MEC são os assuntos mais presentes encontrados a partir de 2013 nos resultados de busca.

Figura 3.3: Relação entre a quantidade de artigos pesquisados e selecionados.



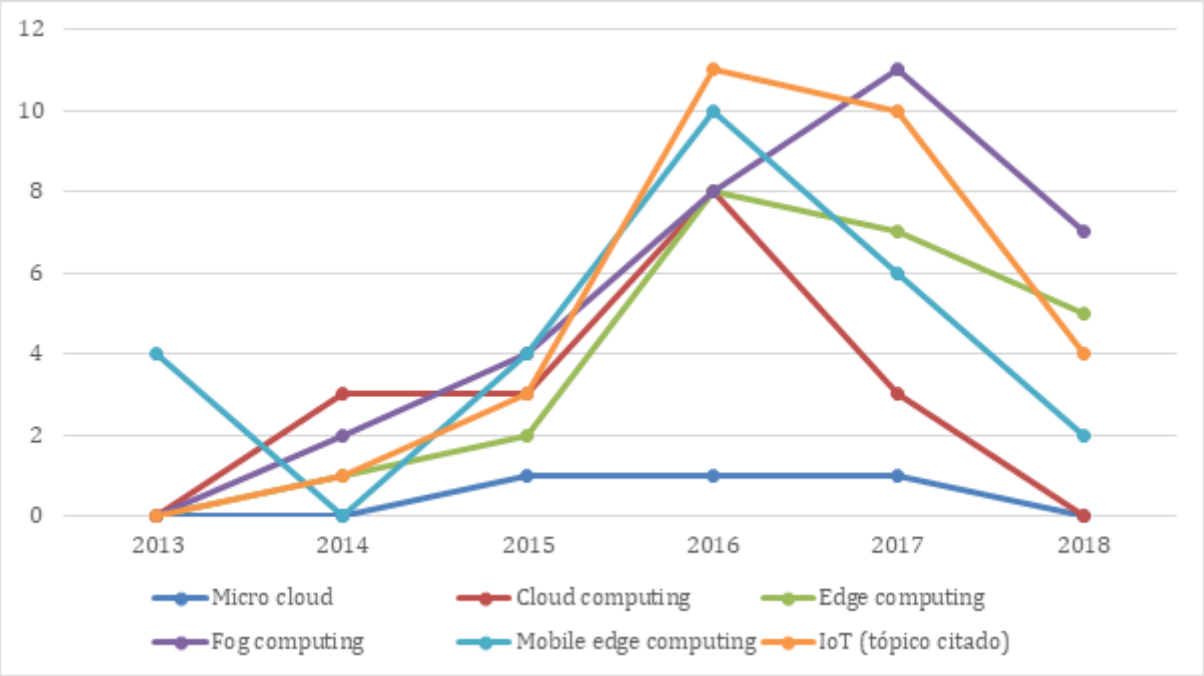
Foi criado um tópico extra para a palavra “IoT”, mesmo que este termo não tenha sido utilizado nas buscas, visto que muitos dos artigos encontrados elaboram seus trabalhos sobre essa tecnologia. Ou seja, dentre os artigos selecionados para as palavras de busca da primeira etapa, as colunas referentes a IoT indicam a quantidade dos artigos selecionados que também citam este tema.

Figura 3.4: Quantidade de artigos selecionados por área.



Também foi realizada uma análise da distribuição dos artigos por período. Conforme apresentado nas Figuras 3.5 e 3.6, a maioria dos artigos concentra-se nos anos 2016, 2017 e 2018, totalizando 76% do total de artigos, somados estes 3 anos. Não foram encontrados artigos para 2019, pois essa primeira etapa foi realizada próximo ao início

Figura 3.5: Quantidade de artigos por tema e período.



de 2019, e, portanto, ainda não havia publicações até o momento das pesquisas para esse ano. Os anos de 2016 e 2017 são os que concentram mais artigos, e embora a produção científica nas áreas de FC e EC tenha aumentado no tempo, os artigos publicados em 2018 ainda não possuem relevância suficiente para serem priorizados pelos motores de busca, e, portanto, acabaram aparecendo com menor incidência nos resultados, e, consequentemente, na escolha final de artigos. Esta afirmação também se respalda pelo interesse que os temas de EC e FC vêm apresentando nos últimos anos. A Figura 3.7 apresenta o interesse das palavras de busca *"Fog Computing"* e *"Edge Computing"* de janeiro de 2013 até novembro de 2018 no buscador Google. De acordo com o *Google*, a escala de popularidade deve ser interpretada da seguinte forma:

> "Os números representam o interesse de pesquisa relativo ao ponto mais alto no gráfico de uma determinada região em um dado período. Um valor de 100 representa o pico de popularidade de um termo. Um valor de 50 significa que o termo teve metade da popularidade. Uma pontuação de 0 significa que não havia dados suficientes sobre o termo."

Desse modo, conforme a tendência das pesquisas globais dos referidos termos, o interesse (popularidade) sobre ambos vem crescendo, chegando a picos de 100 em períodos de 2018, nos dois casos.

Essa primeira etapa foi importante para entender as definições e funcionamento das tecnologias, aplicações, problemas que enfrentam e desafios. Foi possível ter uma

Figura 3.6: Quantidade de artigos selecionados por ano.

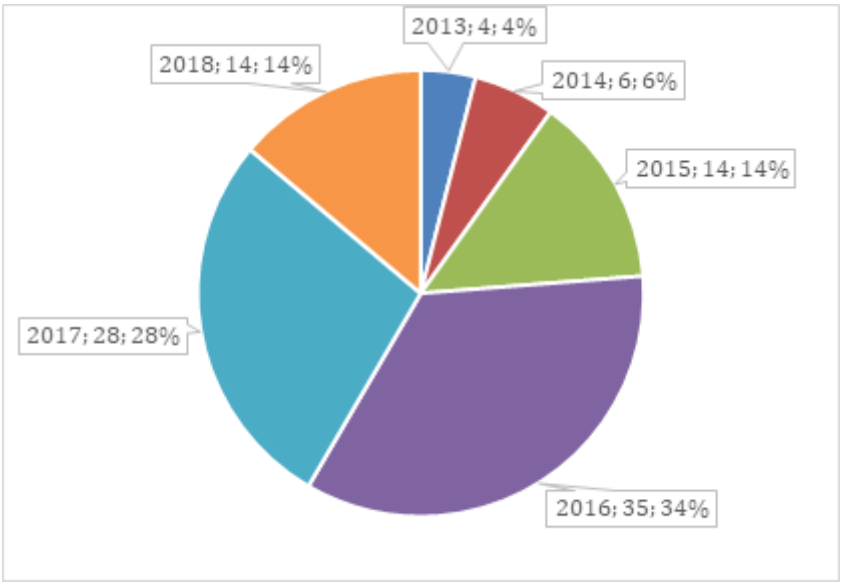
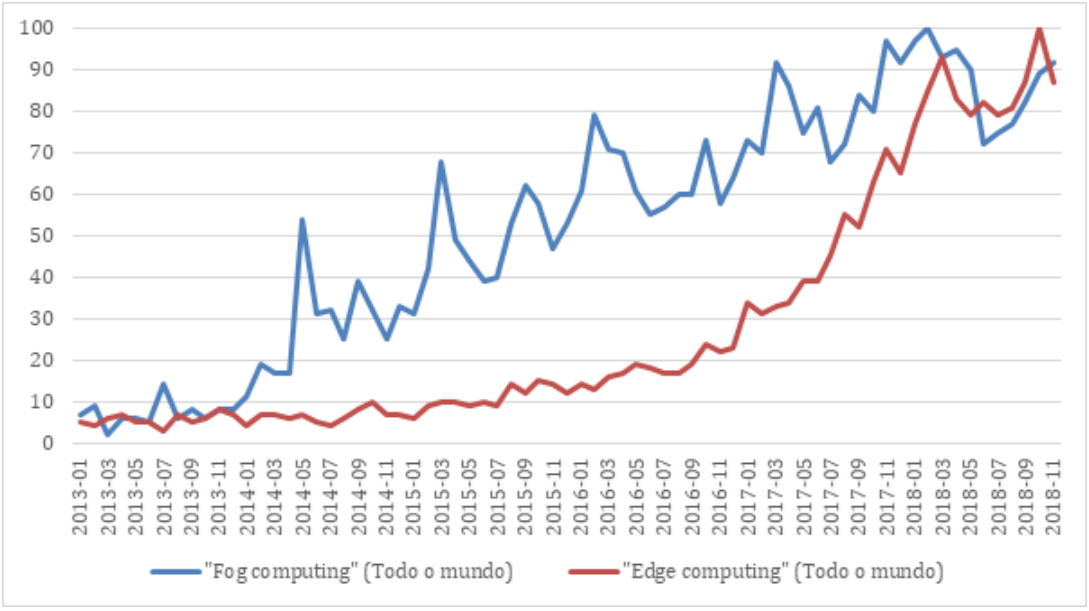


Figura 3.7: Popularidade dos assuntos *"Fog Computing"* e *"Edge Computing"* no mundo, utilizando o buscador do *Google*.



visão mais ampla sobre o papel das tecnologias de *Cloud Computing*, EC, FC e MEC, bem como a correlação existente entre elas e IoT. Além disso, o conjunto de artigos obtidos forneceu referências a artigos bastante citados por outros trabalhos nas referidas áreas de pesquisa.

3.1.3 Segunda etapa

Com base nas conclusões obtidas da análise de artigos da primeira etapa, foi elaborada uma segunda etapa de pesquisa, com foco em EC, FC e IoT, em busca de trabalhos desenvolvidos com o objetivo de reduzir o consumo de energia do sistema, utilizar de modo mais eficaz os recursos ou melhorar a qualidade de serviço, reduzindo a latência de comunicação. Assim, tendo como objetivo da segunda etapa encontrar artigos com trabalhos desenvolvidos considerando aspectos de energia, foram escolhidas as palavras *"green"* e *"energy"*, agregadas a EC, FC e IoT. Os conjuntos de palavras escolhidos para esta etapa foram:

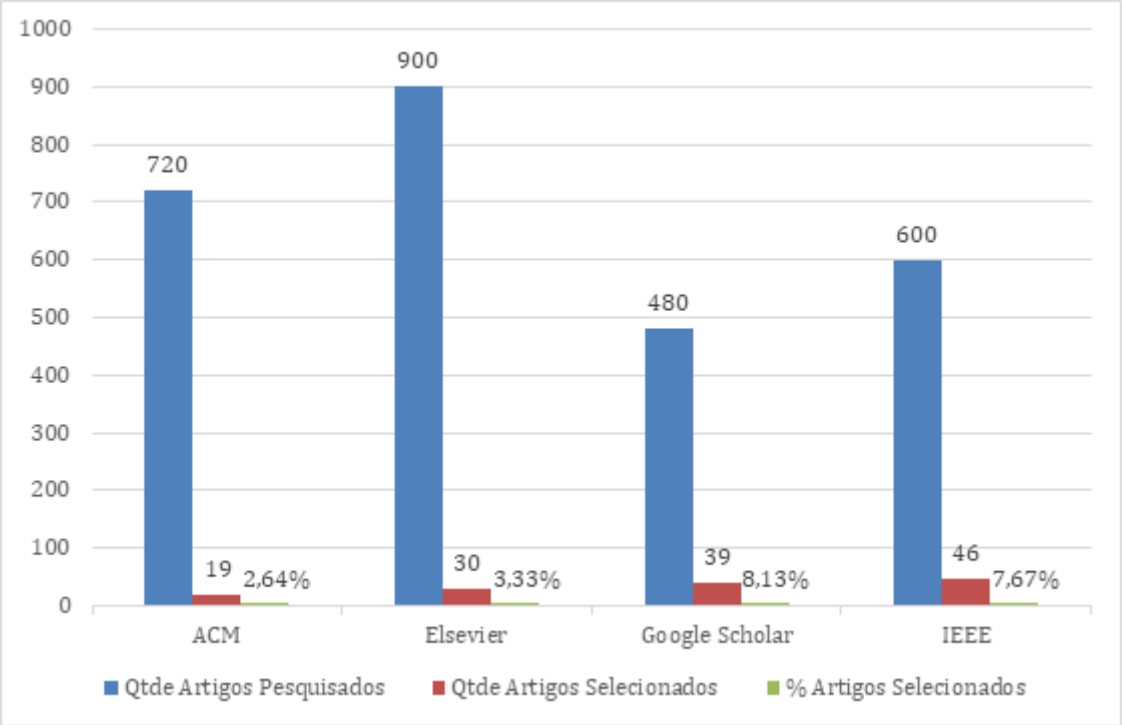
- *Green Edge Computing*;
- *Energy Edge Computing*;
- *Green Fog Computing*;
- *Energy Fog Computing*;
- *Green* IoT;
- *Energy* IoT.

As buscas ocorreram para artigos publicados a partir de 2013 e apenas os artigos que desenvolveram pesquisas para redução do consumo de energia em sistemas FC, EC e IoT foram considerados. A introdução da palavra *"green"* no conjunto de busca ocorreu pois em alguns *surveys* esta palavra estava vinculada a trabalhos que lidavam com aspectos de energia, sendo assim uma tentativa de encontrar artigos interessantes sobre o tema de redução do consumo de energia em sistemas com dispositivos móveis.

Na Figura 3.8 há um resumo das quantidades de artigos pesquisados em cada um dos portais e as quantidades de artigos selecionados, ou seja, com conteúdo relevante. Nessa etapa não foram realizadas buscas no portal Springer, pois, conforme observado na primeira etapa, o portal apresentou os piores resultados e pouco agregou.

Nas buscas realizadas no portal da ACM verificou-se que o conceito de *green energy* está muito mais associado a fontes renováveis e sustentáveis de energia do que

Figura 3.8: Quantidade de artigos pesquisados e selecionados por portal de busca.



efetivamente em consumo de energia. É um tópico importante atualmente, visto que as emissões de CO_2 (ROSSI; GAETANI; DEFINA, 2016) são crescentes a cada ano e o aquecimento global torna as intempéries cada vez mais agressivas. Inicialmente este trabalho possuía como premissa o uso de tecnologias *green* para auxiliar no consumo energético na origem, ou seja, junto ao dispositivo, e, por conseguinte, na infraestrutura periférica que dá auxílio ao processamento de tarefas, i.e. a infraestrutura de FC. Porém, tecnologias *green* não necessariamente trazem este benefício, e, portanto, nesta etapa de busca, após consolidar melhor o entendimento sobre GC, as atenções do autor voltaram-se a trabalhos com foco em redução do consumo de energia, ou operação mais eficaz de tarefas, não necessariamente utilizando estratégias *green*. Para pesquisas sobre artigos vinculados a IoT, utilizando a palavra agregada *"energy"*, foi encontrada uma maior gama de artigos.

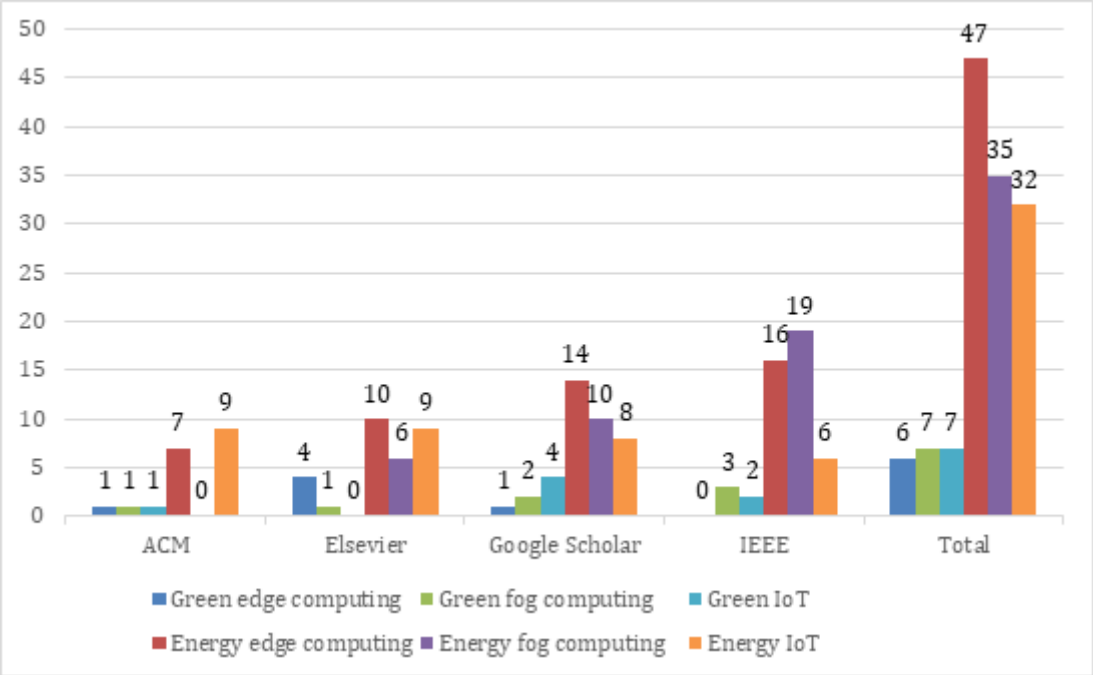
No portal de busca da Elsevier foi encontrada a maior quantidade de artigos relacionados ao tema *green* para aplicações de otimização do consumo de energia, porém envolvendo sustentabilidade e utilização de equipamentos de menor consumo energético. Por outro lado, a quantidade de artigos focados em arquiteturas e modelos de baixo consumo energético para FC e EC foi muito maior.

No *Google Scholar* também foram encontrados artigos sobre GC vinculados à sustentabilidade dos sistemas, mas não necessariamente à redução do consumo de ener-

gia. Nesse portal foram obtidos muitos artigos com propostas de novas arquiteturas para ambientes FC e EC, bem como para MEC. Alguns trabalhos apresentaram formulações matemáticas para otimização de recursos energéticos e redução da latência nas comunicações para tarefas críticas, com tempo de conclusão curto. Os artigos referenciados no *Google Scholar* foram originários principalmente do portal IEEE, mas também foram encontrados artigos da Springer, Elsevier, ACM e MDPI, nesta ordem, por quantidade decrescente de artigos.

Por fim o portal do IEEE apresentou os resultados mais relevantes, com poucos artigos sobre o tema *green* e muitos artigos sobre os demais tópicos. Novamente, o motor de busca do IEEE se mostrou mais eficiente em apresentar bons resultados.

Figura 3.9: Artigos selecionados nos portais de busca.



Na Figura 3.9 é possível observar que a relevância de artigos com temas *green* é pequena, e, portanto, poucos artigos são apresentados nos motores de busca. Já para as palavras "*Energy Fog*", "*Energy Edge*" e "*Energy* IoT", há mais produção acadêmica.

Nas Figuras 3.10 e 3.11 é possível observar que a quantidade de artigos selecionados escala à medida que se aproxima o ano de 2019. Com o tempo a produção acadêmica de projetos com foco na otimização de energia de arquiteturas e sistemas para EC, FC e MEC aumentaram, conforme observado na quantidade de resultados de busca nos portais das editoras. Assim como na primeira etapa de busca, 2019 não teve a maior quantidade de artigos selecionados, principalmente pois no momento das buscas o ano de 2019 estava em andamento e os motores de busca não retornaram os artigos recentemente publicados

Figura 3.10: Quantidade de artigos por período de publicação.

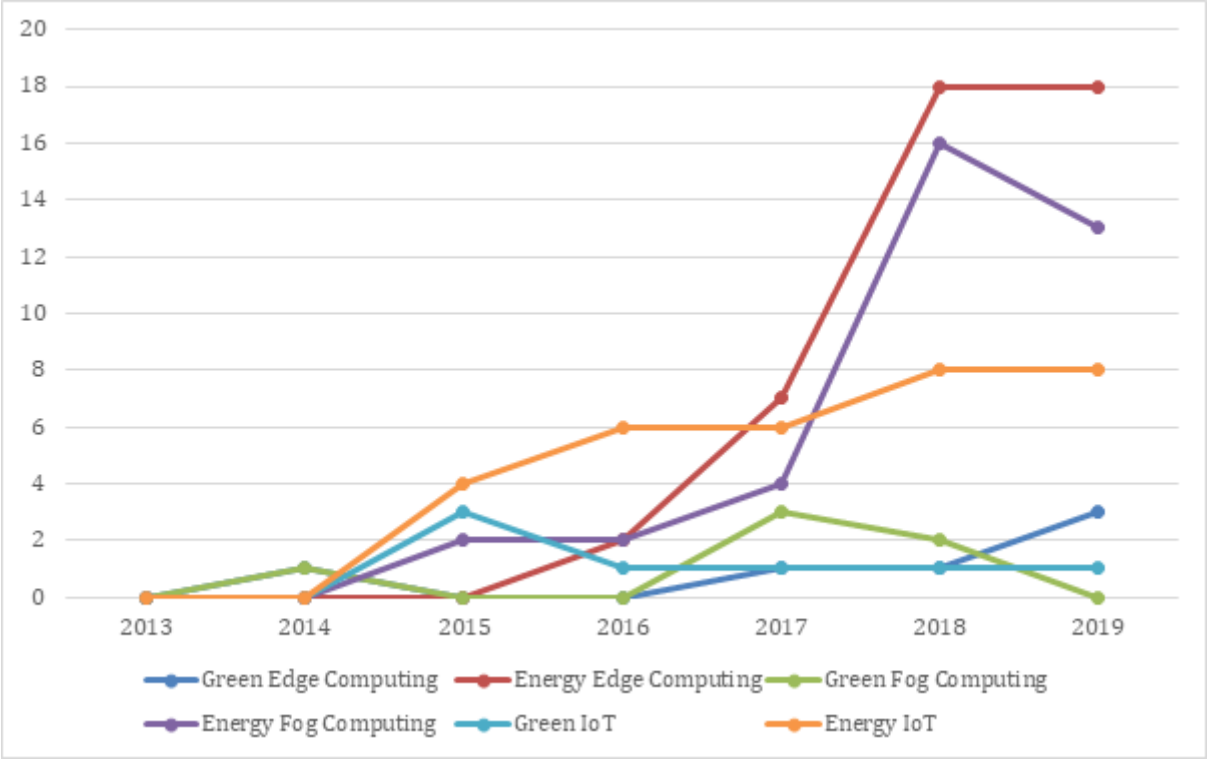
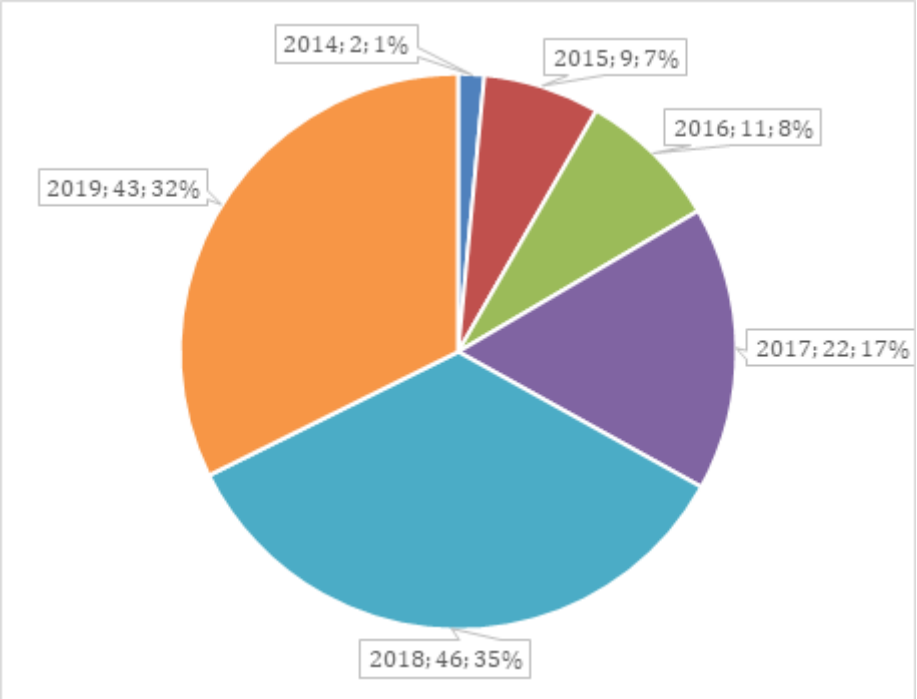


Figura 3.11: Participação percentual do ano de publicação dos artigos selecionados.



nos primeiros resultados, dando prioridade a artigos mais acessados, que naturalmente também são mais antigos. Logo, os artigos encontrados foram os mais relevantes no momento das buscas, e os tópicos pesquisados têm apresentado cada vez mais produções acadêmicas.

Os artigos selecionados nesta etapa apresentaram modelos de custo para minimizar o consumo de energia e latência em sistemas FC e MEC, totalizando 7 artigos. Nas seções seguintes são apresentados em detalhes cada um desses artigos.

3.2 Trabalho 1: *Fog Computing for Energy-Aware Load Balancing and Scheduling in Smart Factory*

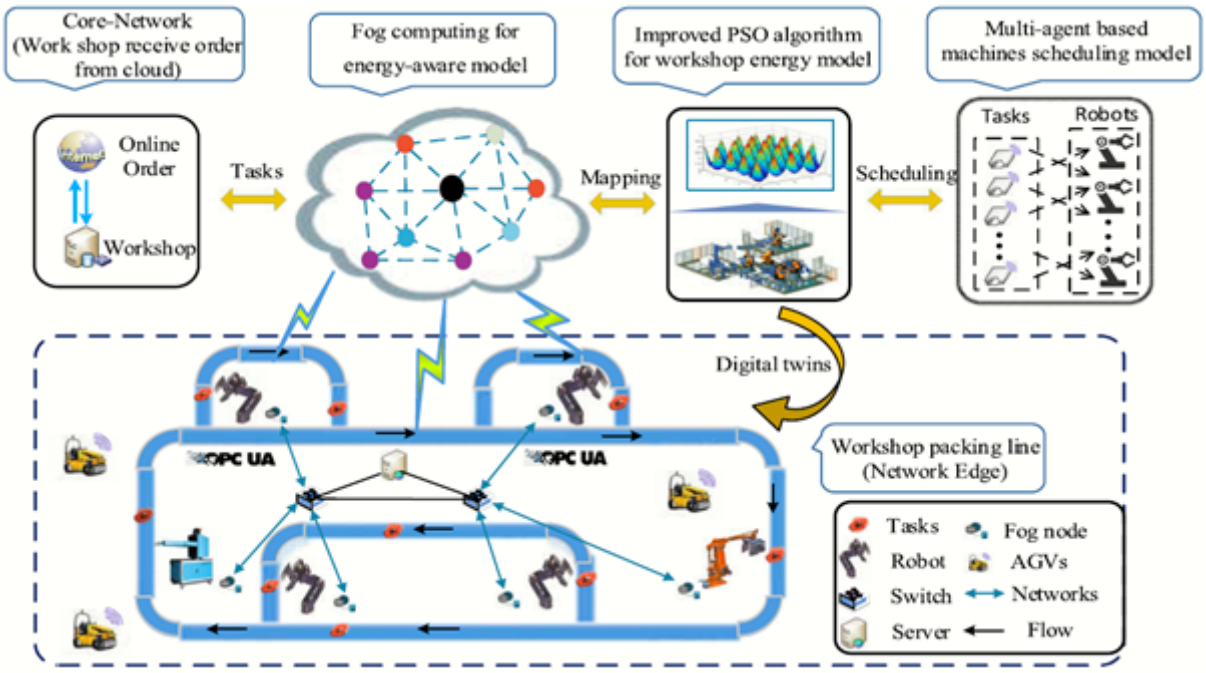
O trabalho (WAN et al., 2018) possui como proposta um método de escalonamento e balanceamento de carga baseado em energia em um ambiente de FC. Os autores comentam que o problema de escalonamento trabalhado é da classe de problemas *NP-hard* (SELS; GHEYSEN; VANHOUCKE, 2012), com multi-tarefas e multi-agentes atuando na rede.

Unidades de processamento, chamadas de *fog nodes*, são instaladas próximas aos equipamentos da rede, coletando dados atualizados dos dispositivos IoT da rede e mantendo essas informações sempre atualizadas. Uma rede de informações sobre os *fog nodes* é criada e compartilhada entre eles, e as informações são depois utilizadas para as tomadas de decisão. O modelo de consumo de energia é montado utilizando todos os *fog nodes*. Aliado aos *fog nodes*, é implantado um sistema de multi-agentes, que consultam as informações fornecidas pelos *fog nodes* e negociam entre si quais tarefas devem ser executadas em cada equipamento da rede, de modo a solucionar o problema de escalonamento e balanceamento de carga de tarefas em paralelo, sempre com foco no consumo de energia.

A aplicação do modelo é realizada em uma fábrica inteligente (*smart factory*) que recebe ordens da Internet e executa tarefas na rede local, conforme arquitetura da Figura 3.12. Os agentes, após realizarem as tomadas de decisão, enviam as tarefas aos *fog nodes*. O processamento das tarefas é realizado pelos *fog nodes*, e, por fim, os resultados são migrados aos dispositivos corretos para serem consumidos.

O modelo é separado em duas etapas. A primeira é do modelo de energia propriamente dito, no qual os *fog nodes* coletam dados sobre consumo de energia dos dispositivos da rede e de suas cargas de trabalho e compartilham entre si. Na segunda etapa os *fog no-*

Figura 3.12: Arquitetura de *Fog Computing* em uma fábrica inteligente (WAN et al., 2018).



des se comunicam via protocolos de rede e realizam escalonamento dinâmico das tarefas entre si, e em conjunto solucionam os problemas em paralelo. O escalonamento é realizado através de uma função de otimização de balanceamento de carga baseado em uma otimização do algoritmo de enxame de partículas e que considera o consumo de energia dos equipamentos para distribuir as cargas de trabalho.

Foram realizados experimentos com dispositivos reais. Os *fog nodes* foram configurados em equipamentos *Raspberry Pi*, visto que possuem características típicas de *fog nodes*, podendo ser instalados próximos dos dispositivos da rede. Os resultados mostraram que o modelo de balanceamento de carga e redução do consumo energético é mais preciso quando executado nos *fog nodes* ao invés da plataforma de *Cloud* centralizada. Os robôs puderam responder mais rapidamente à medida que informações atualizadas eram coletadas, melhorando a inteligência terminal da rede e o tempo de resposta das solicitações.

3.2.1 Críticas ao trabalho 1

- Embora o modelo de consumo de energia seja executado em cada *fog node*, o autor não discorre sobre o consumo de energia que a própria execução do modelo geraria na rede, visto que é um algoritmo que está em constante execução e em todos os

fog nodes, ou seja, consome ciclos de máquina e energia de todos os equipamentos da rede;

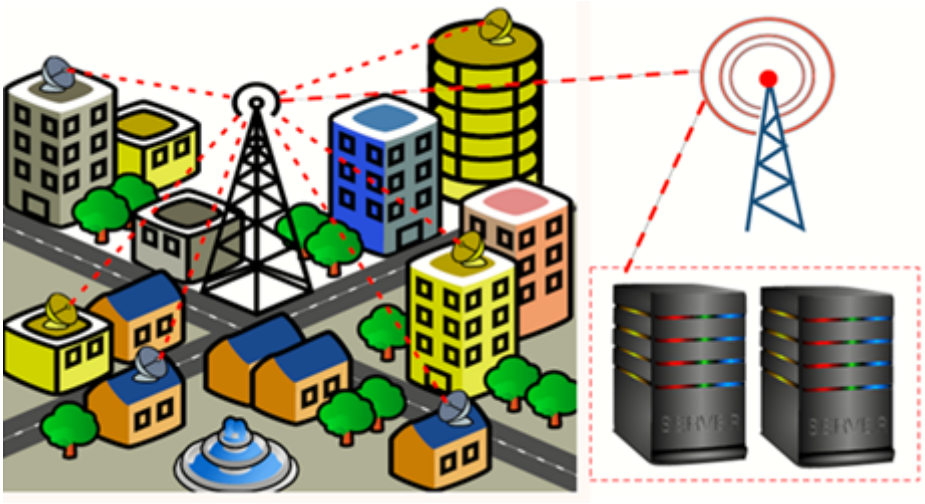
- *Fog nodes* utilizam comunicação *machine-to-machine*, considerada rápida, porém o impacto dessas comunicações em termos da latência das comunicações não foi mensurado pelo autor;
- Embora o autor fale que sistemas multi-agente são importantes para solução de problemas que envolvam latência, nem o modelo e nem o algoritmo desenvolvido consideram latência das comunicações. Logo, é correta a afirmação que FC reduz a latência das transmissões de dados, devido à proximidade dos equipamentos e se comparado às plataformas de *Cloud*, porém essa redução não foi mensurada ou mesmo experimentada nos experimentos;
- A experimentação do trabalho foi bastante focada na distribuição de carga, deixando de lado o consumo de energia. Não são apresentados gráficos ou dados que mostrem que o algoritmo desenvolvido de fato trouxe redução do consumo de energia à rede. O escalonamento ocorre, mas os dados ilustram, basicamente, a distribuição da carga de trabalho dos diferentes *fog nodes*, esta sim, com melhoras de distribuição de *workload* entre os *fog nodes* se comparada a outra estratégia de escalonamento e distribuição de carga apresentada no artigo.

3.3 Trabalho 2: *Energy efficient scheduling in IoT networks*

Em (SARANGI; GOEL; SINGH, 2018) os autores apresentam uma arquitetura de 3 camadas para dispositivos IoT, na qual a camada da base (*bottom layer*) possui toda sorte de redes de sensores sem fio (*Wireless Sensor Networks*, ou WSNs), como veículos inteligentes, casas inteligentes, prédios, semáforos, lâmpadas de iluminação pública, bem como dispositivos do tipo *wearables*, *smartphones*, calçados e *smartcards*. A camada intermediária (*middle layer*) possui os nodos de *gateway* que coletam e armazenam em *buffers* os dados lidos dos sensores, realizam diversos tipos de computações, tipicamente operações de *Data Analytics*, e encaminham os dados através de múltiplos níveis na rede (hops) até o *Data Center*. A última camada, camada de topo (*top layer*), consiste em servidores tipicamente localizados em Data Centers. Esses servidores processam os dados recebidos dos *gateways*, armazenam os dados e enviam ordens aos atuadores da rede para alterarem seu status. As mensagens enviadas dos servidores aos dispositivos da rede

trafegam em sentido inverso pelos *gateways* e *hubs*.

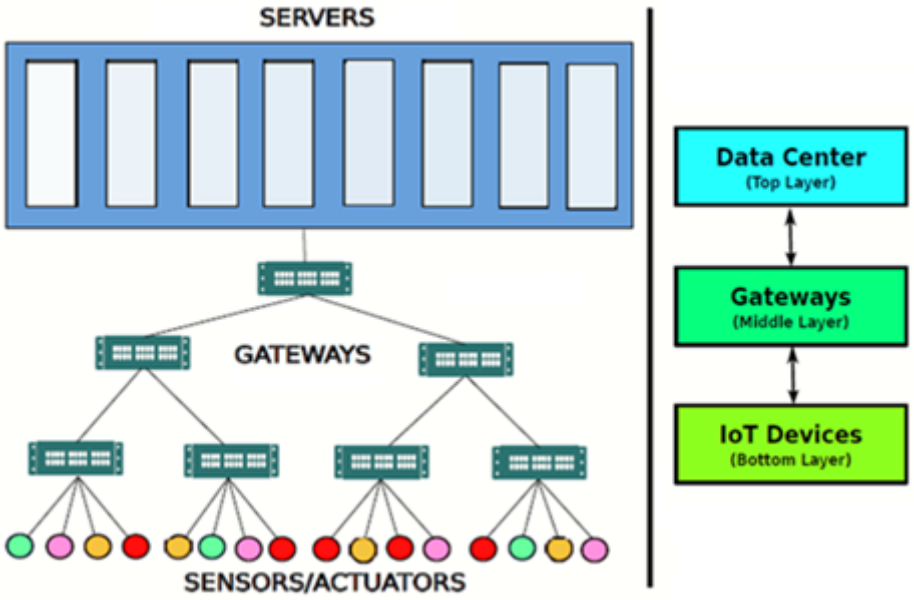
Figura 3.13: Arquitetura IoT.



Fonte: (SARANGI; GOEL; SINGH, 2018).

São apresentados dois algoritmos de escalonamento, um local e um global. No algoritmo local os dispositivos IoT trocam informações entre os nodos vizinhos, enquanto no algoritmo global é utilizado um servidor global (conectado com *links* de transmissão de alta velocidade) que possui visão recente do estado global de toda a rede (*snapshot*). Em ambas estratégias o foco é a redução do consumo de energia.

Figura 3.14: Arquitetura IoT de três camadas detalhada.



Fonte: (SARANGI; GOEL; SINGH, 2018).

O modelo de consumo de energia é baseado na técnica DVFS que determina que o consumo de energia de um dispositivo pode ser calculado pelo produto da quantidade

de ciclos de processamento necessários para o execução da tarefa e a frequência de operação do núcleo. Assim, alterando a frequência de operação do núcleo é possível reduzir o consumo de energia do dispositivo à custa de maior tempo necessário para o processamento ser concluído. Na estratégia local o próprio dispositivo determina sua frequência de operação para executar a tarefa a ele alocada, enquanto na estratégia global o servidor remoto calcula a configuração de cada nodo da rede. Também é utilizada a técnica de mudança de estado dos dispositivos para *idle*, enquanto não estiverem realizando nenhuma computação, de modo a reduzir o consumo de energia quando neste estado.

Vale notar que o trabalho não aborda a arquitetura FC, visto que não prevê a utilização de um servidor local para atender às requisições ou necessidades de recursos dos dispositivos da rede. A abordagem utilizada prevê uma estratégia local na qual a rede de dispositivos IoT troca mensagens entre si e uma estratégia global com o auxílio de *Data Centers* (*Cloud Computing*). Porém, embora os autores não tenham comentado, a arquitetura utilizada possui características de MEC, pois há dispositivos IoT que se movimentam na rede, a exemplo dos veículos inteligentes e *smartphones*, logo, é previsto que haja mobilidade dos dispositivos inteligentes dentro da rede e que estes sigam mantendo contato entre si e com o servidor central.

A avaliação foi realizada via simulação. Para a rede de sensores foi utilizado o simulador *NS3* e para o *Data Center* foi utilizado o simulador *CloudSim*. Os autores comentam que utilizaram dois simuladores, pois desconhecem a existência de um simulador completo para IoT, que englobe os sensores (dispositivos IoT) e o servidor remoto (*Data Center*). Os algoritmos propostos com DVFS foram confrontados com uma abordagem *No-DVFS*, na qual a frequência dos núcleos é fixa, e com uma abordagem *DeadLineShare-DVFS*, na qual o deadline da tarefa é dividido igualmente entre todos os nodos de processamento e cada nodo escala a sua frequência aplicando DVFS de acordo com o tempo alocado para executar a tarefa.

Os resultados mostraram que a abordagem com DVFS ganha do No-DVFS em todos os casos e em alguns casos do *DeadLineShare-DVFS*, com relação à redução do consumo de energia. Além disso, a abordagem DVFS viola poucos deadlines, se comparado ao *DeadLineShare-DVFS*. A redução no consumo de energia com as estratégias apresentadas foi da ordem de 40%.

3.3.1 Críticas ao trabalho 2

- Não foram considerados os custos de consumo de energia dos *gateways* e *hubs* da rede;
- O consumo de energia de processadores em *idle* é desconsiderado. A função que determina a frequência de operação de cada núcleo do dispositivo analisa se o processador está em *idle* e caso esteja determina a frequência de operação e executa a nova tarefa;
- O tempo de execução das tarefas é considerado para evitar ou minimizar que o deadline de tarefas seja violado, porém não há preocupação em reduzir a latência das tarefas em geral.

3.4 Trabalho 3: *Energy Efficient Scheduling for Heterogeneous Fog Computing Architectures*

No trabalho (WU; LEE, 2018), primeiramente os autores desenvolveram um modelo de programação linear de inteiros (*Integer Linear Programming*) com foco na redução do consumo de energia dos dispositivos IoT em uma arquitetura FC heterogênea. Baseado nas observações do modelo, foi derivado um algoritmo de escalonamento para minimização de energia (*Energy Minimization Scheduling* - EMS) que combina diferentes políticas para se aproximar do escalonamento ótimo.

O modelo considera que há apenas dois tipos de *edge nodes*, ou nodos de borda, nodos rápidos e nodos lentos. Os nodos rápidos são de alta performance, com poderosos recursos computacionais, mas consomem mais energia, enquanto os nodos lentos são eficientes energeticamente, mas com recursos computacionais limitados. Além disso, o consumo de energia e o tempo de execução dos programas em cada tipo de nodo pode ser determinado com antecedência, o que é razoável, visto que a maioria das tarefas dos dispositivos IoT são executadas repetidas vezes e periodicamente. Não são permitidas migrações de tarefas entre nodos diferentes.

Figura 3.15: Equação de consumo de energia total minimizada pelo modelo.

$$E = W + I = W_F + W_S + I_F + I_S,$$

Fonte: (WU; LEE, 2018).

O modelo é bastante simples e considera o consumo de energia de núcleos em atividade e núcleos em modo de baixo consumo (*idle*). Na equação, *W* representa a energia de trabalho, ou seja, consumo de energia para núcleos de processamento com carga em execução e *I* representa o consumo de energia dos núcleos em *idle*. *F* refere-se aos nodos rápidos e *S* aos nodos lentos.

O tempo de execução das tarefas é utilizado como uma restrição à minimização, de modo que as tarefas não podem exceder o tempo de execução máxima previsto, porém não há preocupação em reduzir o tempo de execução. O tempo de execução da tarefa é um dado importante para calcular o quanto de energia é gasta pelo núcleo de processamento ao longo do tempo, enquanto estiver no modo de execução.

Os autores não deixam claro, mas não é comentado se os dispositivos da rede se comunicam entre si, dando a entender que o algoritmo de escalonamento é executado por um nodo *master*, com visão global da rede. O que reforça essa conclusão é que o algoritmo desenvolvido retorna um mapeamento das tarefas para os nodos rápidos e nodos lentos, ou seja, em algum momento as informações sobre todos os nodos e todas as tarefas são compiladas para que o algoritmo possa chegar ao mapeamento final.

A experimentação foi realizada via simulação. Foram utilizados três outros algoritmos em comparação ao algoritmo proposto. Os resultados mostraram que o algoritmo proposto pode atingir redução do consumo de energia se comparado à solução ótima para diferentes configurações de nodos e cargas de trabalho.

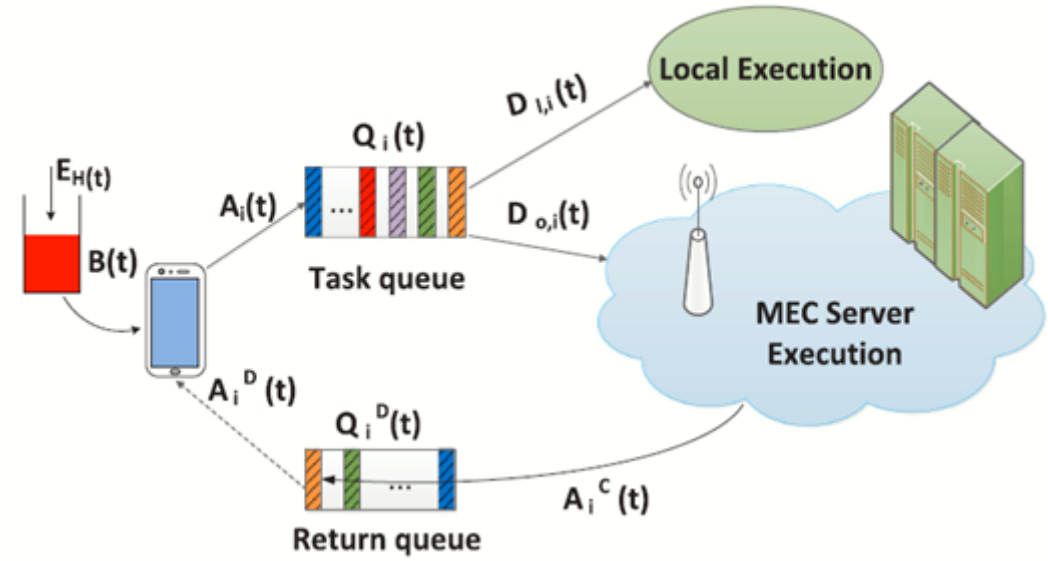
3.4.1 Críticas ao trabalho 3

- Os autores não deixam claro se há comunicação entre os dispositivos IoT ou como ela funciona.

3.5 Trabalho 4: *Energy-Delay Tradeoff for Dynamic Offloading in Mobile-Edge Computing System With Energy Harvesting Devices*

A proposta do trabalho (ZHANG et al., 2018) é desenvolvida para um ambiente MEC focado em redução do consumo de energia e latência de execução, utilizando de modo concomitante a capacidade de coleta de energia de fontes renováveis para abastecimento das baterias dos dispositivos móveis.

Figura 3.16: Modelo do sistema.



Fonte: (ZHANG et al., 2018).

Porém, vale uma ressalva referente às fontes de energia renováveis. Do modo como é apresentado o modelo, é indiferente ao escalonamento a origem da fonte de energia. Sendo esta renovável ou não renovável nada muda no mapeamento de alocação de tarefas. Assim, o apelo ao uso de fontes renováveis de energia fica prejudicado, pois não influi em nada nos resultados de mapeamento ou mesmo nos resultados de consumo de energia ou latência.

O modelo é elaborado como um problema de minimização da soma ponderada entre o consumo de energia e o tempo de execução das tarefas, utilizando como restrições as filas de tarefas e o nível de bateria dos dispositivos. As tarefas são alocadas para execução em um núcleo local dos dispositivos ou em um servidor MEC, também local com uma infraestrutura de processamento rica em recursos, ou também podem ser descartadas.

Considera-se que cada dispositivo executa N aplicações e cada aplicação possui uma fila de tarefas e uma fila de retorno de resultados. As novas tarefas das aplicações ficam nas filas de tarefas aguardando serem escalonadas para execução local ou no servidor MEC. A fila de retorno de resultados é abastecida pelo servidor MEC, quando este executa uma tarefa e gera resultados, deixando estes resultados disponíveis para *download* (utilizando transmissão sem fio) pela aplicação do dispositivo móvel. A transmissão de dados entre os dispositivos da rede e o servidor MEC é realizada via canais sem fio e conexões *device-to-device* (D2D), método de alta eficiência e baixa latência segundo os autores.

A execução das tarefas no servidor MEC leva a transmissão de dados. As transmissões de dados incorrem em consumo de energia e latência adicionais, não podendo, segundo os autores, ser desconsideradas. Desse modo, quando dados são transmitidos ao servidor MEC é considerado o custo energético das transmissões de dados, bem como o tempo agregado pela transmissão da tarefa e dados e pela transmissão dos resultados de volta à aplicação do dispositivo móvel. O tempo total de processamento pelo servidor MEC é igual à soma do tempo de transmissão da tarefa, tempo de execução no servidor MEC e tempo de *download* dos resultados.

O algoritmo de escalonamento é desenvolvido com base no método de otimização de *Lyapunov*, permitindo o cálculo de um escalonamento ótimo para as tarefas nos diferentes dispositivos da rede. A redução da latência de execução é realizada pelo ajuste dinâmico da frequência de operação dos núcleos locais e a escolha de bons canais de transmissão para a transmissão de dados auxilia na redução do consumo de energia. O algoritmo decide se a alocação das tarefas novas será para execução local, no servidor MEC ou descartadas.

A execução do algoritmo é realizada em cada dispositivo. As novas tarefas das aplicações são avaliadas quanto a execução local, no próprio dispositivo ou no servidor MEC, conforme as restrições de bateria do dispositivo, latência e consumo de energia.

A experimentação foi realizada em ambiente simulado e o algoritmo proposto não foi confrontado com outras propostas, mas posto em execução para confirmar sua corretude. O resultado da simulação mostrou um sistema com alta eficiência energética e baixa latência, se comparada a uma abordagem com apenas execução local (sem servidor MEC).

3.5.1 Críticas ao trabalho 4

- Não ficou claro o que o autor considera ser GC, pois em momentos ele fala de GC se referindo à redução do consumo de energia e em outros momentos fala em *green energy* para se referir a fontes de energia limpa e sustentável. Porém, a dúvida que fica é se GC seria já por si só a redução do consumo de energia do sistema ou se, necessariamente, deve haver uso de fontes de energia renovável e limpa para ser definido dessa forma, para o autor;
- Não ficou claro como é realizada a busca por fontes de energia renovável para carregar a bateria dos dispositivos e qual o impacto de utilizar energia renovável. Do

modo como é apresentado, é possível concluir que as baterias são abastecidas por energia renovável quando há necessidade de carregá-las. Desse modo, carregando as baterias com energia renovável ou não renovável não faz diferença no modelo, pois o que é contabilizado é a necessidade de energia para as baterias, sem haver distinção entre energia de fontes renováveis e não renováveis. Portanto, nada muda no modelo, pois os resultados finais são indiferentes ao tipo de energia que abastece as baterias.

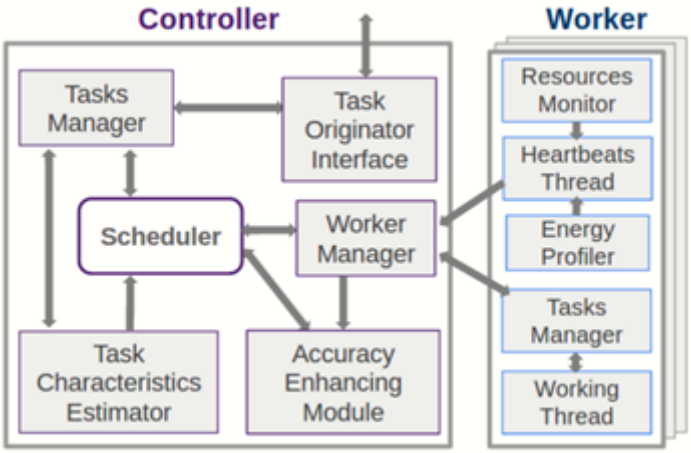
3.6 Trabalho 5: *Awakening the Cloud Within Energy-Aware Task Scheduling on Edge IoT Devices*

O objetivo principal do trabalho (GEDAWY et al., 2018) é, nas palavras dos autores, desenvolver um escalonador otimizado que minimize o tempo total de completude do processamento de tarefas *batch*, ou seja, minimizar a latência do sistema. Critérios de energia são considerados, mas secundários. Reduzir o consumo de energia do sistema não é um dos objetivos principais do trabalho. Os critérios de energia são definidos pelo usuário para que o consumo não ultrapasse um determinado *threshold*, não havendo, necessariamente, redução do consumo de energia, se comparado a outras estratégias que objetivem isso. Difere da proposta de trabalho deste autor, que é reduzir o consumo de energia e também a latência, porém o trabalho analisado aborda os aspectos de escalonamento, importantes para estudo e comparação.

No trabalho em análise é utilizado o poder computacional dos dispositivos IoT que operam em *idle*, agrupando-os em *clusters* locais de dispositivos tratados como pequenas estruturas de *Edge Micro-Cloud*, ou seja, não há uma infraestrutura auxiliar dedicada ao processamento de tarefas. Esses *clusters* de dispositivos recebem tarefas de um controlador e o dispositivo ocioso dentro do *cluster* processa a tarefa e devolve os resultados para a aplicação destino.

A arquitetura do sistema é formada por dois componentes principais, o *Controller* e o conjunto de *Workers*. O *Controller* gerencia o grupo de dispositivos e seus recursos. É ele quem executa o algoritmo de escalonamento e alocação de tarefas, também cabendo a ele estimar características das tarefas e distribuí-las aos *Workers* disponíveis. Os *Workers* são dispositivos IoT ou dispositivos que executam o sistema operacional *Android* e são responsáveis por executar as tarefas recebidas pelo *Controller*, além de compartilhar o seu estado.

Figura 3.17: Controlador e *Workers* do sistema.



Fonte: (GEDAWY et al., 2018).

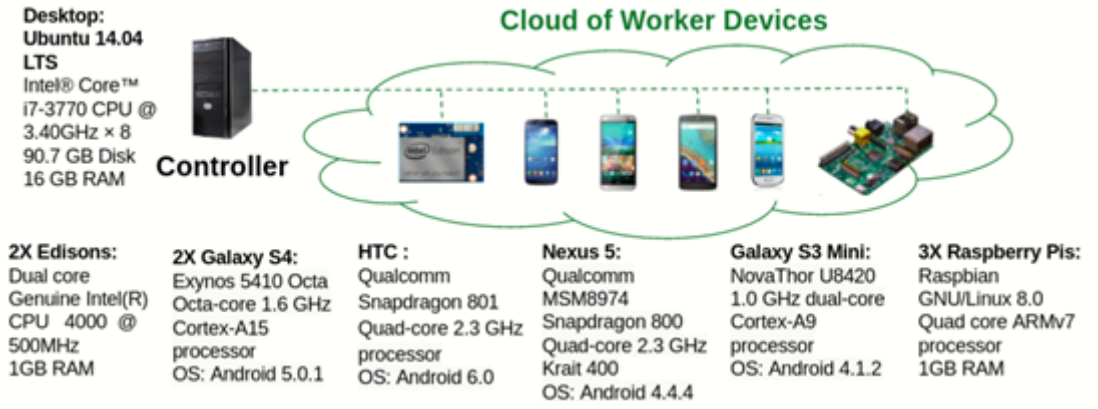
O modelo apresentado maximiza o uso de recursos, sem drenar muito a bateria dos dispositivos e também há a possibilidade de o usuário definir um limite de consumo total de energia do sistema. São consideradas as bandas dos *links* de *download* e *upload* ao se comunicar com o *Controller* e seus respectivos consumos para transmitir unidades de dados, o que não é comum em outros modelos. Além disso, considera o tamanho da entrada de dados e dos resultados da tarefa no cálculo de tempo e consumo de energia.

Cabe ressaltar que mesmo com mais de uma tarefa alocada para um *Worker*, não é realizada execução de modo concorrente. Também é dimensionado tempo suficiente para a completa execução da tarefa no *Worker*, além de tempo suficiente para enviar os resultados de volta ao *Controller*. O tempo total de execução de uma tarefa considera o tempo de *offload* da entrada, tempo de execução da tarefa no dispositivo selecionado e tempo para enviar os resultados ao controlador.

O escalonamento é modelado como um problema de programação de inteiros (0s e 1s) que pode ser provado como um problema NP-Completo usando redução de tempo polinomial do *Bin Packing Problem*. Um algoritmo heurístico é desenvolvido para solucionar o modelo de modo eficaz. A heurística, entretanto, não é apresentada pelo autor. A complexidade da heurística de escalonamento é $O((nlogn) + nm)$, *n* representa as tarefas e *m* representa os *Workers*. A complexidade é dividida em $O(nlogn)$ para a etapa de inicialização, na qual tarefas são ordenadas de acordo com suas prioridades, e $O(nm)$ para a etapa de alocação, na qual cada tarefa é alocada para um *Worker*.

É interessante observar que o autor desenvolveu uma função de priorização de tarefas, com base em diversos critérios. O resultado dessa função permite ao escalonador definir qual tarefa que será primeiramente alocada aos *Workers* disponíveis.

Figura 3.18: Conjunto de dispositivos e *setup* utilizados para os experimentos.



Fonte: (GEDAWY et al., 2018).

O protótipo desenvolvido pelos autores conta com 10 dispositivos, sendo um grupo de dispositivos IoT e um grupo de dispositivos *Android* (*smartphones*). Os resultados mostraram um ganho de 40% em *speedup* se comparado a outros escalonadores, porém não são apresentados ganhos com relação ao consumo de energia.

3.6.1 Críticas ao trabalho 5

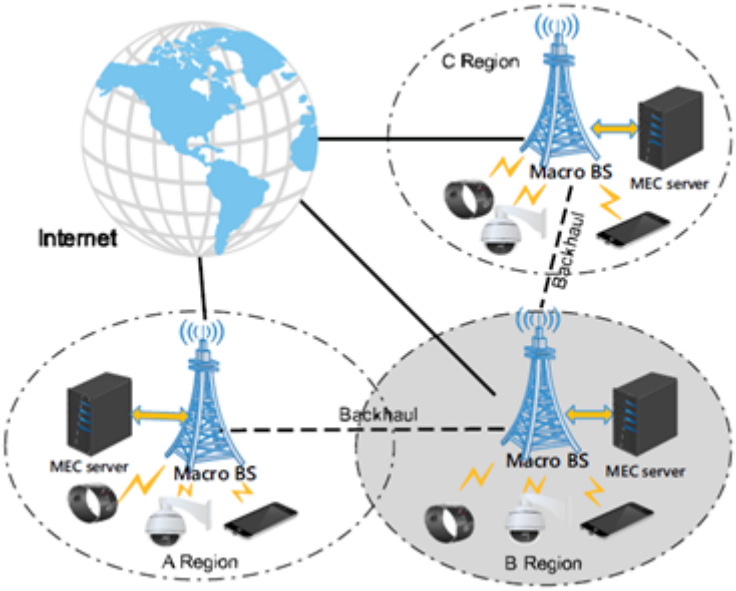
- O autor prioriza tarefas com requisitos de maiores quantidades de recursos de modo a, em suas palavras, aumentar a eficiência e maximizar a execução paralela de tarefas. Porém o autor não fala como chegou a esta conclusão, apenas assume isso como sendo um fato conhecido. Não discute, por exemplo, se seria melhor priorizar tarefas que demandam muitos recursos ou tarefas que demandam poucos recursos;
- O autor estabelece uma restrição de energia total consumida pelo sistema, chamada de *energy threshold*. O usuário deve conhecer muito bem o consumo energético do sistema, podendo haver problemas caso seja estabelecido um limite baixo. Em sistemas genéricos, sem usuários especialistas em consumo energético, o principal objetivo é manter o sistema funcionando, com a execução total das tarefas em tempo razoável e, na medida do possível, reduzir o consumo energético. Assim, é mais assertiva uma abordagem que vise minimizar o consumo de energia, sem valores absolutos impostos, o que pode comprometer a execução total das tarefas do sistema, pois ao alcançar o limite de consumo o sistema para de funcionar.

3.7 Trabalho 6: *Energy-efficient Offloading Policy for Resource Allocation in Distributed Mobile Edge Computing*

O trabalho (WANG et al., 2018) foi desenvolvido para o ambiente MEC, tendo foco em dispositivos móveis que circulam por vastas áreas geográficas e se comunicam com a Internet por redes móveis, basicamente sinais 3G e 4G. Assim, a abordagem utilizada pelos autores é diferente de outros trabalhos também desenvolvidos para dispositivos IoT, mas que são fixos em determinados locais ou possuem mobilidade restrita a, por exemplo, um ambiente industrial ou uma residência.

Para melhorar a qualidade da experiência dos usuários da rede os autores propuseram uma arquitetura na qual são executadas três políticas de descarregamento de tarefas, (1) execução local das tarefas nos próprios dispositivos, (2) descarregamento para servidores na região local (servidores MEC) ou (3) descarregamento para servidores em regiões vizinhas. As políticas (1) e (2) são suficientes para a maioria dos casos, porém os servidores MEC possuem recursos limitados e havendo insuficiência de recursos em uma dada região os servidores de uma região vizinha podem suprir esta necessidade, utilizando, desse modo, a política (3). Não está previsto escalonamento de tarefas para a *Cloud*, sendo a "Internet" descrita na Figura 3.19 mero canal de comunicação entre as diversas regiões.

Figura 3.19: Arquitetura do sistema distribuído com dispositivos móveis.



Fonte: (WANG et al., 2018).

O modelo desenvolvido contempla um problema de otimização no qual o objetivo é minimizar os custos totais do sistema, dados pelo somatório dos custos de todos os

dispositivos móveis quanto às decisões de descarregamento (*offloading*). O custo de um dispositivo móvel é composto pelo tempo de atraso e pelo consumo de energia, ambos com diferentes pesos atribuídos pela equação de custo, gastos para completar a execução das suas tarefas.

Cada servidor MEC do sistema é instalado em uma estação base macro (*Macro Base Station*, ou MBS), servindo de infraestrutura auxiliar para processamento das cargas de trabalho geradas pelos dispositivos móveis daquela região. Já cada região MBS possui N dispositivos móveis e M canais de comunicação sem fio. Cada dispositivo móvel se comunica com o servidor MEC via redes móveis de conexão sem fio e não são utilizadas redes WiFi, pois pontos de acesso WiFi não são partes inerentes de uma rede móvel, além de redes WiFi apresentarem uma cobertura limitada.

É importante ressaltar que o modelo proposto considera para os custos de tempo o tempo de transmissão da entrada de dados e código do dispositivo ao servidor MEC, o tempo de execução e tempo de transmissão dos resultados do servidor MEC de volta até o dispositivo. Além disso, são considerados no modelo os gastos energéticos das transmissões de dados (quando ocorre comunicação entre dispositivo e servidor) e da execução da tarefa.

Encontrar o custo mínimo do sistema é um problema *NP-Hard*. Assim, foi desenvolvido um algoritmo distribuído para a política de descarregamento de tarefas baseado no *algoritmo de Jacob*, capaz de aproximar o custo mínimo total do sistema utilizando um conjunto de políticas sub-ótima para cada dispositivo móvel em um pequeno período de tempo.

Os experimentos ocorreram via simulação, havendo a comparação com outras duas estratégias: (i) apenas execução local e (ii) descarregamento de tarefas sem considerar insuficiência de recursos. Segundo os autores o algoritmo desenvolvido consegue diminuir os custos totais do sistema em até 50%, se comparado às estratégias (i) e (ii).

Os resultados também mostraram que o equilíbrio da equação de custo, que considera o tempo e o consumo de energia, não deve priorizar muito o consumo de energia na equação (peso muito alto para consumo de energia), senão todos os dispositivos decidem enviar as tarefas para o servidor MEC. Isso ocorre, pois o consumo de energia de um ciclo de processamento no servidor MEC é inferior ao do dispositivo local, e o dispositivo móvel consegue satisfazer a latência de execução das tarefas no servidor MEC com menor peso ponderado. Logo, os pesos atribuídos ao consumo de energia e à latência na equação de custo devem ser bem dimensionados para que esse comportamento não ocorra.

3.7.1 Críticas ao trabalho 6

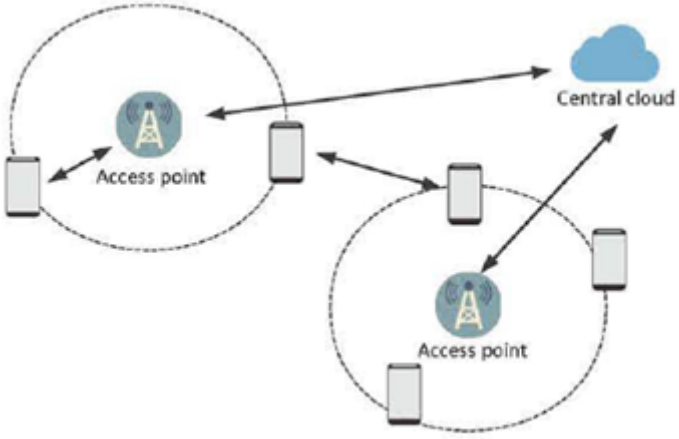
- Os cuidados com gastos menores de energia ocorrem apenas quando o nível de bateria do dispositivo móvel está baixo, não atentando devidamente à minimização do consumo de energia. Essa abordagem pode trazer problemas, pois com a bateria do dispositivo muito baixa ele pode ficar indisponível.

3.8 Trabalho 7: *Energy-Efficient Task Offloading and Resource Scheduling for Mobile Edge Computing*

O trabalho (YU; WANG; GUO, 2018) é desenvolvido para o ambiente MEC e prevê o escalonamento de tarefas para infraestruturas auxiliares de borda da rede, ou *Edge Clouds*, estendendo a capacidade de processamento dos dispositivos móveis do sistema. Há como objetivos o uso racional e eficiente de energia e a preservação da experiência do usuário, ao considerar o tempo de finalização das tarefas.

Os autores formularam o problema de descarregamento de tarefas (*offloading problem*) como um problema de minimização, levando em consideração o tempo de conclusão das tarefas e o consumo de energia. Para o problema de otimização foi proposto um algoritmo distribuído constituído de seleção de estratégias de descarregamento de tarefas, configuração da frequência de *clock* dos núcleos de processamento, alocação da potência de transmissão e escalonamento do canal de comunicação.

Figura 3.20: Arquitetura do ambiente MEC.



Fonte: (YU; WANG; GUO, 2018).

São definidos três modelos para o sistema. O modelo de computação local calcula

o tempo de processamento da tarefa e o consumo energético, ambos com base no número de ciclos de processamento necessários para a conclusão da tarefa e na frequência de operação do núcleo. A equação de custo é semelhante àquela utilizada no Trabalho 6, pois realiza uma soma ponderada do tempo de processamento e do consumo energético da tarefa.

O modelo de computação na *Edge Cloud*, além dos cálculos realizados no modelo local, também considera os tempos de transmissão dos dados e do código à *Edge Cloud* e o custo energético dessa transmissão. O tempo de espera na fila de tarefas prontas para execução também é considerado para o custo total de tempo, porém não agrega em custos energéticos. O tempo de *download* dos resultados, porém, não é considerado, nem seu custo energético.

O modelo de computação na *Cloud* é muito semelhante ao modelo de computação na *Edge Cloud*. A única diferença reside no custo energético total que não considera a parcela de consumo de execução da tarefa, mas considera o consumo das transmissões de dados. Os autores explicam que essa parcela de custo energético não é devida ao modelo de computação em *Cloud*, pois ela possui recursos virtualmente ilimitados, ou seja, por não terem recursos limitados não seria necessária sua racionalização.

A equação de custo dos modelos de computação local e *Edge Cloud* são as mesmas para a soma ponderada do tempo e consumo energético. Já a equação de custo do modelo de *Cloud* é dada pela soma ponderada das equações de custo do modelo local e do modelo de *Edge Cloud*, porém, os autores não explicam por qual motivo a equação de custo do modelo de *Cloud* é dada desta forma. E, por fim, a equação de custo total do sistema é dada pela soma ponderada do custo do modelo de *Cloud* e do custo do modelo local.

O problema de otimização definido consiste em minimizar a equação de custo total. São otimizados, dessa forma, a frequência de operação dos núcleos de processamento, para definir a melhor relação entre tempo de processamento e consumo energético durante a execução da tarefa, bem como a escolha dos canais de comunicação para cada transmissão de dados, garantindo a melhor relação entre tempo de transmissão dos dados e o seu consumo energético, além da minimização do tempo de espera na fila do canal.

O algoritmo desenvolvido pelos autores não é apresentado. São apresentados alguns detalhes de cálculo da frequência de operação dos núcleos, potência da transmissão, tempo de espera na fila do canal e custo total, bem como a sua complexidade de operação. E, embora se diga que o algoritmo é distribuído, a forma como os cálculos de minimização ocorrem leva o leitor a crer que há um nodo central com conhecimento sobre todo

o sistema, de posse da frequência de operação dos núcleos, tamanho e quantidade das tarefas e características dos canais de comunicação e suas filas de tarefas.

Ele é comparado com outras três estratégias de escalonamento e descarregamento de tarefas, algoritmo guloso (RA et al., 2011), *método de Mao* (MAO et al., 2017) e *método de Dihn* (DINH et al., 2017). Os resultados mostraram que o algoritmo proposto possui melhores resultados quando exposto a uma maior quantidade de tarefas, enquanto os métodos de Mao e Dihn são melhores para pequenas quantidades de tarefas. Os dados apresentados nas simulações mostram apenas o resultado dos algoritmos em termos dos custos totais do sistema, quanto ao tempo gasto e ao consumo de energia elétrica, ficando difícil saber o modo de trabalho do algoritmo e o que ele prioriza. Os próprios autores apresentam suas conclusões de performance do algoritmo proposto apenas com base nos custos totais do sistema, não se referindo ao tempo ou ao consumo energético.

3.8.1 Críticas ao trabalho 7

- A equação de custo para o modelo de *Cloud* não foi explicada adequadamente, pois considera o custo do modelo de *Edge Cloud* e do modelo local;
- Os resultados são dados em termos dos custos totais do sistema, não sendo possível saber sobre a otimização do tempo gasto ou sobre a redução no consumo de energia com relação aos demais algoritmos utilizados para o comparativo. Os autores também realizam todas as suas observações sobre os resultados em cima dos custos totais, não se referindo ao tempo ou ao consumo energético.

3.9 Considerações Sobre o Capítulo

Neste capítulo foram apresentados em detalhes os artigos do estado da arte que trabalham com modelos de minimização de custos de ambientes FC e MEC. A partir dos detalhes apresentados será possível extrair características e compará-las no capítulo seguinte.

4 ANÁLISE COMPARATIVA

Este capítulo apresenta uma análise comparativa dos artigos apresentados no Capítulo 3 discutindo a forma com que lidam com problemas de consumo de energia, latência das comunicações, tempo de processamento e deadline de tarefas e experimentação em ambientes *Fog Computing* e *Mobile Edge Computing*. Essas informações são utilizadas para a definição da proposta de modelo do sistema no Capítulo 5.

4.1 Tópicos de Análise

A partir da análise dos trabalhos relacionados foi elaborada uma série de tópicos pertinentes para análise nesta pesquisa e que servirão de base para identificar características importantes para o desenvolvimento de um modelo com foco em redução do consumo de energia e latência. Cada tópico possui uma tabela permitindo visualizar se cada um dos trabalhos atende ou não aos requisitos apresentados.

Cada trabalho relacionado é referenciado por um número, conforme a lista a seguir:

- [1] : (WAN et al., 2018);
- [2] : (SARANGI; GOEL; SINGH, 2018);
- [3] : (WU; LEE, 2018);
- [4] : (ZHANG et al., 2018);
- [5] : (GEDAWY et al., 2018);
- [6] : (WANG et al., 2018);
- [7] : (YU; WANG; GUO, 2018).

4.1.1 Tópico 1: Quanto ao tipo de arquitetura utilizado

Todos os trabalhados desenvolveram propostas para dispositivos inteligentes com conexão à Internet. Houve cuidado em selecionar para análise trabalhos que fossem desenvolvidos com esta tecnologia em mente, justamente por este trabalho também utilizar tais tecnologias.

A maioria deles possuía algum tipo de infraestrutura auxiliar local para o proces-

Tabela 4.1: Tipos de arquiteturas utilizadas.

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Fog Computing	X			X	X	X	X
Edge Computing	X	X	X	X	X	X	X
Mobile Edge Computing		X		X		X	X
Cloud Computing		X					X
Internet of Things (IoT)	X	X	X	X	X	X	X

samento de tarefas, realizando o descarregamento de tarefas a esses centros de processamento sempre que o algoritmo assim determinasse. Entretanto em [2] e [3] essa característica não foi observada, visto que todo o processamento ocorreu apenas nos próprios dispositivos ou também na *Cloud* centralizada como em [2].

Dentre as arquiteturas apresentadas, todas possuíam características de EC, ou seja, trouxeram inteligência à periferia da rede, tendo que lidar com questões como consumo de energia e latência, seja para estender a vida das baterias dos dispositivos ou melhorar a experiência do usuário.

Os trabalhos [2], [4], [6] e [7] tiveram preocupação com a mobilidade dos dispositivos inseridos no sistema. Para esses autores as redes atuais são dinâmicas, com dispositivos que se movem pela rede e os cenários explorados consideraram essa mobilidade. Já os trabalhos [1], [3] e [5] consideraram cenários de dispositivos inteligentes, que embora tenham conexão com a Internet e possuam aplicações que geram tarefas de modo recorrente, são estáticos, não havendo necessidade de lidar com problemas como a maior dificuldade em estabelecer conexões estáveis quando em movimento.

Dada a formatação da sociedade atual, dispositivos móveis estão se proliferando com a evolução tecnológica e cada vez mais aplicações com demandas computacionais intensivas vão surgindo nesses dispositivos (WANG et al., 2018). Mesmo com o aumento da capacidade de processamento desses dispositivos, eles não conseguem lidar com tarefas que necessitam de muitos recursos computacionais e grandes quantidades de energia das baterias em períodos pequenos (DINH et al., 2013). MEC é um paradigma que pode lidar com tarefas de uso intensivo de energia por prover computação e recursos de armazenamento para os dispositivos móveis conseguirem completar suas tarefas (HECK et al., 2018).

Tabela 4.2: Locais de processamento das tarefas no sistema

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Processamento local, no próprio dispositivo	X	X	X	X	X	X	X
Processamento local, no servidor local	X			X		X	X
Processamento remoto, na *Cloud* centralizada		X					X

4.1.2 Tópico 2: Quanto aos locais de processamento das tarefas

Segundo (YOUSEFPOUR et al., 2019), FC preenche a lacuna entre a nuvem e os dispositivos finais, como os dispositivos IoT, permitindo a computação, armazenamento, rede e gerenciamento de dados em nós da rede próximos de dispositivos IoT. Consequentemente, computação, armazenamento, rede, tomada de decisão e gerenciamento de dados não ocorrem apenas na *Cloud*, mas também ao longo do caminho *IoT-to-Cloud* à medida que os dados passam para a nuvem, e, preferencialmente, isso ocorre próximo dos dispositivos IoT.

Essa definição trazida por (YOUSEFPOUR et al., 2019) vai ao encontro do que foi observado no preenchimento da Tabela 4.1 e da Tabela 4.2, pois todos os trabalhos desenvolvidos em arquiteturas FC possuem algum *hardware* dedicado para o processamento das tarefas das aplicações da rede, e o mesmo é válido para arquiteturas MEC. A exceção ocorre com [5] que não possui servidor MEC dedicado, mas a partir de um *cluster* de dispositivos IoT explora a capacidade de processamento ociosa deles para o processamento das tarefas da rede. Logo, mesmo não possuindo uma infraestrutura dedicada de processamento, também pode-se dizer que possui uma espécie de servidor local (*fog node*) para o ambiente FC.

Poucos foram os trabalhos nos quais observou-se o uso da plataforma de *Cloud* centralizada. Os trabalhos se dividiram entre aqueles desenvolvidos para arquiteturas de FC e aquelas para arquiteturas de MEC, permitindo que a computação pudesse ser realizada ou no dispositivo ou em uma unidade dedicada de processamento e armazenamento próxima aos dispositivos da rede (*fog nodes* ou servidores MEC), gerando menores latência de comunicação e custo energético com transmissão de dados.

A redução da latência certamente é uma das principais vantagens do uso dessas arquiteturas, porém, em casos extremos, redes com milhares ou dezenas de milhares de dispositivos podem ficar saturadas. Os servidores locais podem não ser capazes de atender

a todas as solicitações por recursos, já que essas infraestruturas também possuem limitação de recursos. Em cenários como esses a utilização da *Cloud* é importante, visto que possuem recursos virtualmente infinitos e podem desafogar a rede local em caso de necessidade. Desse modo, uma arquitetura de FC ou MEC operando em conjunto com serviços de *Cloud Computing* é fundamental para implementar uma política de escalonamento de sucesso que atenda às necessidades de recursos independente de sua monta.

4.1.3 Tópico 3: Quanto às transmissões de dados

Tabela 4.3: Variáveis consideradas para transmissões de dados.

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Considera a taxa de envio dos dados (*input*) e código para o servidor local quando ocorre offloading?				X	X	X	X
Considera a taxa de *download* dos resultados do servidor local para os dispositivos da rede?				X	X	X	

Não há um padrão, porém a maioria dos trabalhos desenvolvidos para ambiente MEC considera a taxa de envio de dados dos dispositivos locais ao servidor local nos seus modelos do sistema. Em alguns casos também é considerada a taxa de *download* dos resultados, mas em menor quantidade.

O envio de tarefas (código mais dados) é o que mais consome banda dos canais de comunicação da rede, visto que o volume de dados é muito maior do que os resultados pós-processamento. A taxa de *download* dos resultados é ignorada em alguns trabalhos, pois ela representa muito pouco do uso dos recursos dos canais de comunicação, seja em termos do uso de banda ou mesmo do consumo energético associado ao uso do canal. Porém, com o crescimento da quantidade de dispositivos, e, consequentemente, tarefas, a tendência é que o uso dos canais de comunicação para *download* de resultados não seja desprezível. Assim, é um dado importante a ser considerado no modelo do sistema.

4.1.4 Tópico 4: Quanto ao consumo de energia

Referente às transmissões de dados observa-se que a maioria dos trabalhos desenvolvidos para ambientes MEC considera o consumo de energia das transmissões de dados de dispositivos para servidor local e do servidor local para dispositivos. São va-

Tabela 4.4: Variáveis de consumo energético utilizadas pelo modelo do sistema.

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Considera o consumo de energia para enviar dados ao servidor local?				X	X	X	X
Considera o consumo de energia para realizar *download* dos dados do servidor local para os dispositivos da rede?				X	X	X	
Considera o consumo de energia para transmissões de dados entre dispositivos da rede?	X			X			
Realiza ajuste dinâmico da frequência do processador dos dispositivos da rede para reduzir o consumo de energia?		X		X			X
Realiza ajuste dinâmico da frequência do processador do servidor local para reduzir o consumo de energia?				X			X
Realiza ajuste dinâmico da frequência do processador do servidor remoto (*Cloud*)?		X					
Considera o consumo de energia dos processadores em execução?	X	X	X	X	X	X	X
Considera o consumo de energia dos processadores em *idle*?	X		X				
Realiza predição (estimada) do consumo energético da execução das tarefas?	X		X	X	X	X	X
Considera o custo energético dos dispositivos da rede (*access points*, roteadores, *gateways*, *switches*, etc)?							
Considera o nível de bateria do dispositivo móvel para a tomada de decisão?				X	X	X	

riáveis importantes, pois impactam diretamente no consumo global do sistema, visto que a comunicação entre os dispositivos e servidores locais é toda feita via rede sem fio e o consumo energético associado a esse meio de comunicação não é desprezível. Foram poucos os trabalhos a considerar a energia consumida em transmissões entre dispositivos da rede. Um dos principais motivos é que a ocorrência desse evento é muito pequena, pois, ou o processamento ocorre localmente ou é escalonado e descarregado para executar no servidor local ou mesmo na *Cloud*.

A técnica de ajuste dinâmico de frequência nos processadores dos dispositivos da rede e nos processadores dos servidores locais também está presente. Porém, é uma técnica que não foi utilizada em todos os trabalhados avaliados. Os ganhos em termos de consumo de energia ao diminuir a frequência de operação dos processadores são consideráveis, mas ao mesmo tempo impacta negativamente no tempo de conclusão das tarefas,

aumentando o tempo de entrega dos resultados. Mesmo havendo uma relação inversa entre consumo energético e tempo de conclusão das tarefas, o ajuste da frequência dos processadores é uma técnica poderosa que pode trazer mais flexibilidade ao modelo de otimização do sistema e maiores possibilidades de consumo energético ou redução do tempo de conclusão da tarefa, dependendo da prioridade que for estabelecida.

O ajuste dinâmico de frequência nos processadores da *Cloud* ocorreu em apenas um trabalho. Embora seja uma possibilidade, a expectativa é que a *Cloud* seja pouco utilizada e que as variações na frequência de operação dos processadores da *Cloud* não tragam grandes ganhos de performance ou redução do consumo energético. Portanto, para o modelo que será proposto neste trabalho, essa possibilidade não é considerada.

Conforme esperado, o consumo energético dos processadores em execução é uma variável amplamente utilizada. Já o consumo dos processadores quando em *idle* ocorre em apenas alguns casos. Porém o consumo dos processadores em estado *idle* não é desprezível, ainda mais se forem considerados dezenas de milhares de dispositivos na rede. À medida que a rede for aumentando em quantidade de dispositivos inteligentes, a tendência é que o consumo associado a esse estado (*idle*) impacte cada vez mais nos custos energéticos do sistema, logo é uma variável importante a ser considerada.

Praticamente todos os trabalhos, com exceção de um, realizam predição do consumo energético das tarefas, baseado na expectativa de quantidade de ciclos de processamento necessários para a conclusão da tarefa. De posse dessa informação, é possível predizer o consumo energético de cada tarefa criada para os diferentes núcleos de processamento disponíveis, sejam eles núcleos de dispositivos IoT, servidores locais ou da *Cloud* centralizada.

Uma pergunta foi inserida no questionário de consumo energético referente aos dispositivos de rede utilizados pelo sistema, como, por exemplo, *access points*, roteadores, repetidores, *gateways* e *switches*. Nenhum dos trabalhos avaliados se preocupou com esse consumo. Neles não foram realizados comentários sobre o motivo de não inserir os custos energéticos de operação desses dispositivos, porém este autor acredita que isso tenha ocorrido devido à inserção de mais complexidade no modelo. Logo, no presente trabalho, tais custos energéticos também não serão considerados.

Por fim, um parâmetro importante é o nível de bateria dos dispositivos móveis com acesso à rede, visto que tarefas que demandam muitos recursos energéticos podem não ser concluídas caso o dispositivo não possua energia suficiente para concluir a operação.

4.1.5 Tópico 5: Quanto ao tempo

Tabela 4.5: Variáveis de consumo de tempo utilizadas pelo modelo do sistema.

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
O tempo de execução das tarefas é conhecido (estimado)?		X	X	X	X	X	X
Considera o tempo das transmissões de dados?		X		X	X	X	X
Considera o tempo de transmissão do código e dados (*input*) do dispositivo até o servidor local ou *Cloud*?		X		X	X	X	X
Considera o tempo que a tarefa espera na fila do canal até iniciar a transmissão dos dados para o servidor local?		X		X			X
Considera o tempo de execução da tarefa no servidor local?		X		X	X	X	X
Considera o tempo de *download* dos resultados do servidor local até o dispositivo?				X	X	X	
Considera o tempo que os resultados da tarefa esperam na fila do canal até iniciar a transmissão para o dispositivo?				X			

Assim como para as variáveis energéticas há a predição da quantidade de energia gasta pela tarefa, há também a predição do tempo de conclusão na maioria dos trabalhos. Conforme Tabela 4.5, a predição do tempo é baseada na frequência de operação dos núcleos de processamento e na quantidade de ciclos que a tarefa exige até ser finalizado seu processamento. Isso aplica-se tanto para os dispositivos como também para os servidores locais e para a *Cloud*.

Não só o custo energético, mas também os tempos necessários para transmitir dados e código dos dispositivos aos servidores locais, bem como transmitir os resultados de volta aos dispositivos são importantes informações que o modelo deve agregar. O tempo que a tarefa fica em estado de espera na fila do canal de comunicação até ser enviada ao seu destino também é outra informação a ser considerada. Geralmente é um valor pequeno, porém em redes grandes, com muitas tarefas sendo escalonadas, pode representar uma fatia grande do tempo total. Logo que a tarefa é escalonada, ela fica aguardando o canal de comunicação terminar as transmissões em andamento para então iniciar a sua transmissão.

Vale notar que na maioria dos trabalhos avaliados não foi considerado o tempo que os resultados aguardam na fila do canal ao retornarem aos dispositivos. Isso ocorre

principalmente porque os resultados constituem volumes de dados pequenos, sendo transmitidos e já liberando quase que imediatamente o canal, sem gerar filas. Portanto, para este trabalho essa variável também não será utilizada.

4.1.6 Tópico 6: Quanto às características gerais do modelo

Tabela 4.6: Características observadas nos modelos estudados.

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Possui nodo *master* para executar o algoritmo de escalonamento?			X		X		X
Há preocupação com energias renováveis (*Green Computing*)?				X			
O algoritmo de escalonamento é executado em cada device?	X	X		X		X	
Possui foco em reduzir latência?				X	X	X	X
Possui foco em reduzir consumo de energia?	X	X	X	X		X	X
Possui foco em uso de energias renováveis?				X			
É implementado como problema de minimização / otimização?	X	X	X	X	X	X	X
Faz diferenciação de nodos? Nodos rápidos, nodos lentos? Diferenciação por frequência de operação nos núcleos?			X				
Trabalha com fragmentação de tarefas (para agilizar o processamento)?							
Faz *duty cycling* (desliga dispositivos ociosos)?		X	X				

Dentre os trabalhos avaliados apenas três expressam claramente que o algoritmo de escalonamento desenvolvido é centralizado na rede. Os demais não falam explicitamente que não possuem controlador centralizado, mas assume-se que são aplicados de modo distribuído, executando de modo independente nos dispositivos da rede. Não há uma resposta conclusiva sobre qual o modelo mais adequado, se centralizado ou distribuído, pois embora o modelo distribuído seja preferível devido à escalabilidade da rede, o modelo centralizado é mais adequado pela gestão facilitada da quantidade de tarefas da rede, seus tamanhos e previsões de conclusão e custos energéticos, além da visão ampla sobre quais os recursos estão disponíveis, sejam núcleos de processamento, canais de comunicação, níveis de bateria dos dispositivos, dentre outros. Desse modo, visando reduzir o grau de complexidade do modelo, o algoritmo será executado por um nodo master, com

controle sobre as tarefas e recursos da rede.

No âmbito de redução geral de custos, sejam custos energéticos ou em tempo de processamento, há uma correlação direta com um menor impacto ambiental, pois menos uso de energia e arquiteturas mais eficientes geram menos emissões atmosféricas. Houve, portanto, um cuidado em busca de trabalhos com soluções que abordassem aspectos de GC. Porém, dentre os trabalhos avaliados o termo GC praticamente não foi mencionado. Apenas durante a etapa de buscas por artigos é que os motores de busca das editoras associaram o termo aos trabalhos.

Todos os trabalhos, conforme esperado, desenvolveram modelos de minimização para reduzir os custos do sistema, com variáveis para latência e gasto energético. O uso de energias renováveis foi inserido em apenas um dos modelos de dados, mas não como variável para minimização de custos, apenas como alternativa e preferência ao uso de uma fonte de energia limpa, na indisponibilidade do uso de energias geradas por fontes tradicionais térmica e hídrica. No presente trabalho não serão consideradas alternativas de fontes de energia elétrica, visto que isso não impacta no modelo proposto. O foco aqui será em minimizar o custo energético e o tempo total, podendo a fonte de energia ser qualquer uma, proveniente de fontes renováveis ou não renováveis.

Em um dos trabalhos avaliados for introduzido o conceito de nodos de processamento lentos e rápidos. Essa distinção facilitou a escolha entre um nodo ou outro na hora de realizar o processamento das tarefas. Porém nos demais trabalhos avaliados essa distinção não foi percebida, pois o foco passou a ser alterar a frequência de operação dos núcleos de processamento, permitindo torná-los lentos ou rápidos, conforme a necessidade do momento.

Outro questionamento refere-se à fragmentação de tarefas na rede para acelerar o processamento e conclusão. Mas isso revelou-se uma tarefa bastante complexa e não abordada em nenhum trabalho, pois há forte dependência dos dados e também porque muitas tarefas são desenvolvidas com lógicas sequenciais, necessitando que houvesse uma reestruturação das tarefas para que a possibilidade de processamento paralelo fosse explorada. Essa não é, portanto, uma característica que o presente trabalho irá explorar.

Também foi observado em alguns trabalhos o uso da estratégia de *duty cycling*, que desliga os dispositivos ociosos da rede para economizar energia. Porém não foi explicado com muita clareza como se daria a retomada desses núcleos em caso de necessidade e também se haveria algum impacto na geração de novas tarefas, visto que alguns núcleos desligados eram de dispositivos móveis. Assim, foi dada preferência à estratégia de con-

tabilização do custo energético de processadores em *idle*, ao invés de desligá-los, o que este autor acredita ser mais viável por apresentar menos impactos negativos à rede.

4.1.7 Tópico 7: Quanto à forma de experimentação

Tabela 4.7: Formas de experimentação nos trabalhos estudados.

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Utiliza ambiente real para execução dos experimentos?	X				X		
Utiliza ambiente simulado para execução dos experimentos?		X	X	X		X	X
Faz comparação da estratégia de escalonamento apresentada com outras estratégias (outros trabalhos)?	X	X	X		X	X	X

De acordo com a Tabela 4.7, a maioria dos trabalhos realizou experimentação em ambiente simulado. Essa é uma alternativa bastante viável, permitindo simular mais cenários com mais dispositivos móveis, em contrapartida a um ambiente real. Portanto, neste trabalho a experimentação será realizada com simulação, devido à praticidade e à versatilidade dessa alternativa.

Por fim, observa-se que as estratégias de escalonamento desenvolvidas foram comparadas com propostas de outros trabalhos, para levantar as vantagens e desvantagens de cada uma. O trabalho [4] atuou de modo diferente, implementando e experimentando a sua proposta com algoritmos de escalonamento diversos, elaborados pelo próprio autor.

4.2 Considerações Sobre o Capítulo

O presente capítulo teve como objetivo comparar os trabalhos selecionados e discutir as características que serão utilizadas no Capítulo 5 para a definição da proposta de modelo do sistema e construção de um modelo de custo com diversas variáveis de consumo de energia e tempo decorrido, incluindo a utilização da Cloud como alternativa de descarregamento de tarefas.

5 PROPOSTA DE MODELO

Neste capítulo são apresentadas as características do sistema e o modelo proposto, a partir da discussão realizada no Capítulo 4. Em seguida é apresentado o modelo de custo do sistema, as premissas escolhidas para o seu desenvolvimento e suas três opções de processamento, processamento local no dispositivo IoT, processamento local no servidor MEC e processamento remoto na Cloud. No final é apresentada uma seção dedicada a como é tratado o nível de bateria dos dispositivos IoT pelo modelo de custo e a equação de custo global do sistema.

5.1 Características do Sistema e Modelo Proposto

Com base na análise comparativa dos trabalhos avaliados, há uma janela de pesquisa que considera os critérios apresentados na Figura 5.1.

A Figura 5.1 resume as características gerais e as variáveis que serão utilizadas no modelo de minimização de custo proposto por este trabalho. As características que o diferenciam em relação ao estado da arte são a interação com a Cloud, a inserção dos custos associados às transmissões de dados entre dispositivos IoT e servidores MEC e entre servidores MEC e a *Cloud*, além dos custos advindos da fila de prontos, bem como o uso de dois perfis de tarefas, críticas com *deadline* e não críticas sem *deadline*.

Para a arquitetura do sistema foi escolhida uma com características de MEC, permitindo que os dispositivos IoT tenham liberdade de mobilidade pela rede, com servidores locais estrategicamente localizados, que dão assistência ao processamento local sempre que os dispositivos necessitarem. Havendo saturação dos recursos da rede local, sejam eles dos dispositivos IoT ou dos servidores locais, é possível descarregar tarefas à *Cloud*. O uso da *Cloud* não foi muito difundido nos trabalhos avaliados, porém é um recurso importante para aliviar gargalos da rede local.

A Figura 5.2 apresenta a arquitetura do sistema em três camadas, composta por dispositivos móveis (dispositivos IoT) que realizam processamento local, no próprio dispositivo, servidores MEC que realizam o processamento próximo à borda da rede e por *Data Centers* que realizam processamento remoto. A Figura 5.2 não inova em arquitetura, tendo como únicos objetivos esclarecer qual a arquitetura utilizada e evidenciar a interação entre os elementos do sistema.

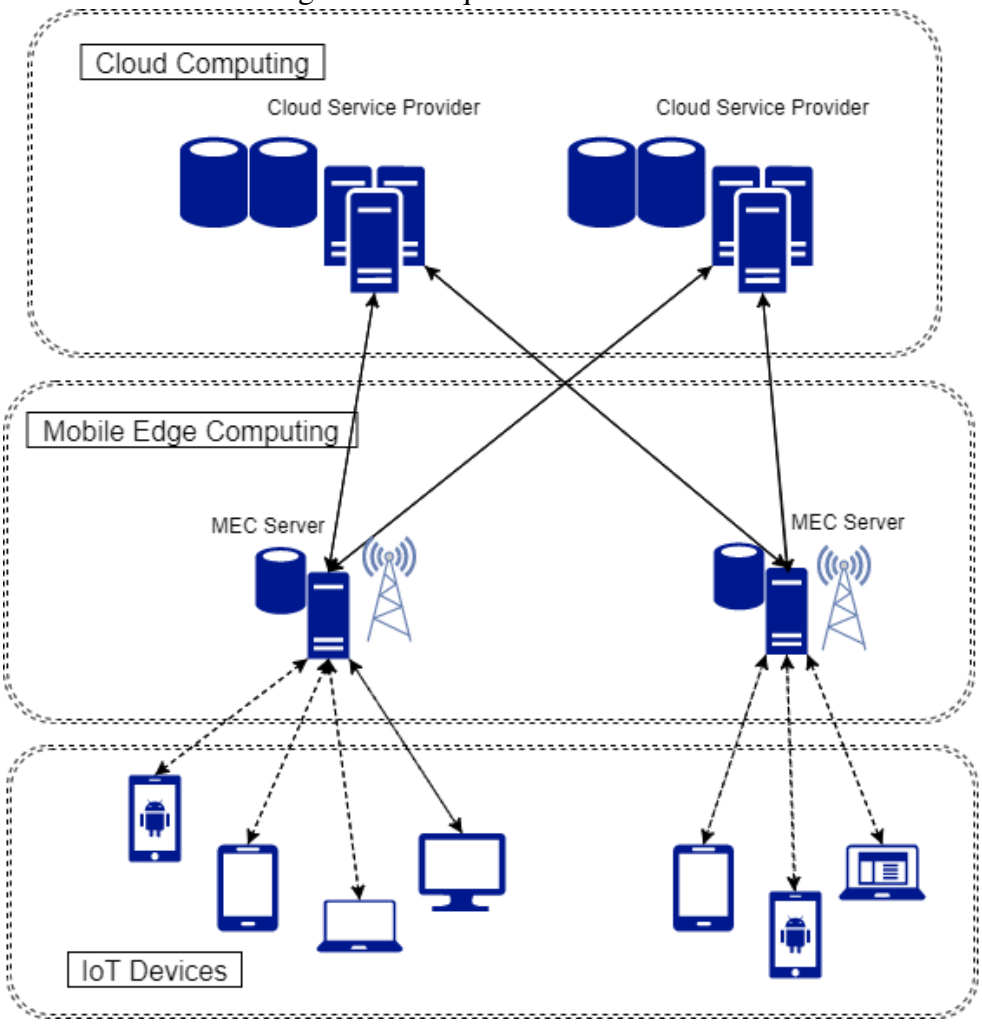
A alocação de tarefas ocorre prioritariamente no próprio dispositivo, não havendo

Figura 5.1: Resumo comparativo de características dos trabalhos analisados e da proposta deste trabalho.

	[1]	[2]	[3]	[4]	[5]	[6]	[7]	Proposta
Arquiteturas								
Mobile Edge Computing + IoT	Não	Sim	Não	Sim	Não	Sim	Sim	Sim
Cloud Computing	Não	Sim	Não	Não	Não	Não	Sim	Sim
Local de processamento								
Processamento local, no próprio dispositivo	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Processamento local, no servidor local	Sim	Não	Não	Sim	Não	Sim	Sim	Sim
Processamento remoto, na Cloud	Não	Sim	Não	Não	Não	Não	Sim	Sim
Variáveis de consumo energético								
Energia gasta nas transmissões de para envio de tarefas (código + dados) ao servidor local ou cloud	Não	Não	Não	Sim	Sim	Sim	Sim	Sim
Energia gasta nas transmissões de download dos resultados do servidor local ou cloud	Não	Não	Não	Sim	Sim	Sim	Não	Sim
Energia dos processadores em execução	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Energia dos processadores em idle	Sim	Não	Sim	Não	Não	Não	Não	Sim
Nível de bateria dos dispositivos inteligentes	Não	Não	Não	Sim	Sim	Sim	Não	Sim
Variáveis de tempo decorrido								
Tempo gasto nas transmissões de envio de tarefas (código + dados) ao servidor local ou cloud	Não	Sim	Não	Sim	Sim	Sim	Sim	Sim
Tempo gasto na fila de prontos do canal de transmissão	Não	Sim	Não	Sim	Não	Não	Sim	Sim
Tempo gasto durante a execução da tarefa	Não	Sim	Não	Sim	Sim	Sim	Sim	Sim
Tempo gasto nas transmissões de download dos resultados do servidor local ou cloud	Não	Não	Não	Sim	Sim	Sim	Não	Sim
Características gerais								
Execução do algoritmo de escalonamento em nodo master	Não	Não	Sim	Não	Sim	Não	Sim	Sim
Foco em reduzir consumo de energia	Sim	Sim	Sim	Sim	Não	Sim	Sim	Sim
Foco em reduzir latência	Não	Não	Não	Sim	Sim	Sim	Sim	Sim
Modelo como problema de minimização/otimização	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Experimentação								
Ambiente simulado para execução dos experimentos	Não	Sim	Sim	Sim	Não	Sim	Sim	Sim
Teste da estratégia de escalonamento em diferentes cenários	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim

latência de comunicação, porém, é preciso considerar o nível de bateria do dispositivo. Havendo necessidade de aumentar o tempo de vida da bateria, ou caso o algoritmo de escalonamento decida alocar a tarefa para executar em outro local de menor custo, o servidor MEC é uma alternativa. Os servidores MEC são compostos por um *hardware multicore* e memória de alta eficiência energética e ficam localizados fisicamente próximos dos dispositivos da rede, diminuindo a latência. As tarefas são descarregadas através de comunicação celular (5G) dos dispositivos aos servidores locais.

Figura 5.2: Arquitetura do sistema.

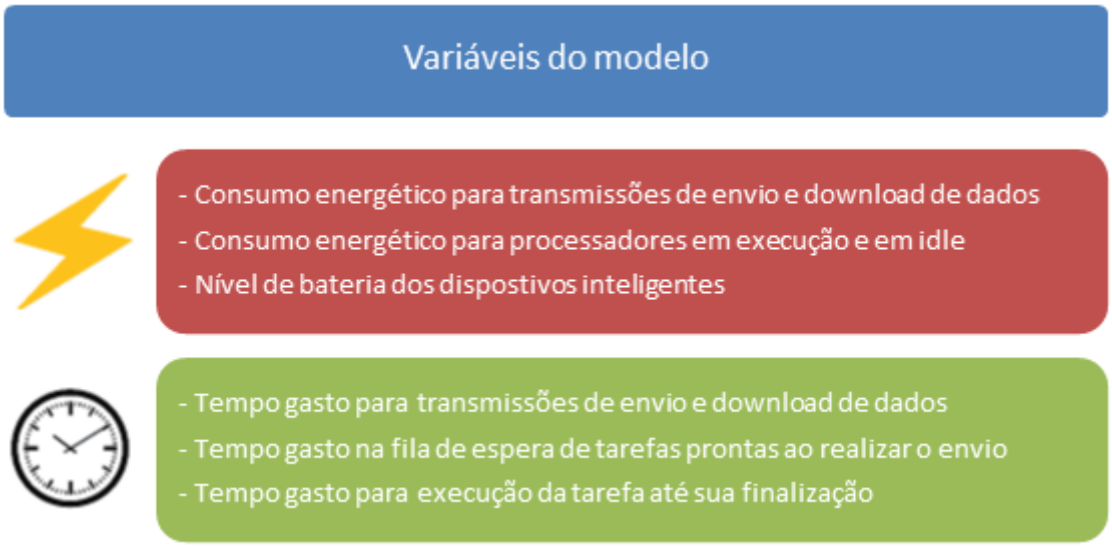


Por fim, na camada de *Cloud Computing* ocorre o processamento remoto de tarefas. Toda vez que o sistema necessita de capacidade de recursos de *hardware*, não satisfeitos pelos dispositivos locais ou pelos servidores locais, a *Cloud* centralizada é utilizada para fornecer os recursos desejados. Conforme a Figura 5.2, é necessário realizar duas transmissões de dados, uma do dispositivo IoT ao servidor MEC (via conexão sem fio) e outra do servidor MEC à *Cloud* (via cabeada). Há mais latência envolvida, pois os *Data Centers* ficam distantes fisicamente das redes de dispositivos móveis, porém a continuidade de entrega de resultados às aplicações segue mantida.

O modelo do sistema calcula o custo total considerando as variáveis descritas na Figura 5.3. O valor dessas variáveis é calculado a partir de variáveis primárias, conforme as equações apresentadas no Capítulo 2:

$$T = \frac{CT}{f} \tag{5.1}$$

Figura 5.3: Variáveis consideradas para o modelo de custo.



$$E = (C * V^2 * f) * T \tag{5.2}$$

A Equação 5.1 calcula o tempo de execução da tarefa no núcleo de processamento, enquanto a Equação 5.2 calcula a energia consumida para a execução da tarefa no núcleo durante o período de tempo *T*. Segundo (YU; WANG; GUO, 2018), *CT* é número de ciclos de CPU necessários para completar a tarefa, f é a frequência de operação do núcleo, também denominada de capacidade de computação, C denota a capacitância da arquitetura da CPU na qual o núcleo está inserido e V é a tensão de alimentação elétrica do núcleo da CPU.

No modelo do sistema a componente de frequência está sujeita a limites inferiores e superiores, visto que cada núcleo possui limites de operação. Esses dados são fornecidos pelo próprio fabricante e podem ser consultados via requisições de software. A técnica de DVFS é utilizada nas equações, configurando a frequência de operação dos núcleos em tempo real, conforme decisão do escalonador do sistema.

O algoritmo do modelo é executado por um nodo *master*, com visão global da rede. Essa abordagem foi escolhida por já ter sido utilizada em outros trabalhos e ter apresentado bons resultados. Além disso, o fato de haver apenas um nodo que executa o algoritmo de escalonamento e realiza as tomadas de decisão facilita a gestão da rede, já que pode fazer a coleta das informações necessárias para alimentar o modelo de custo e a partir daí executar o algoritmo. Esse nodo *master* fica localizado na camada de MEC, podendo ser qualquer núcleo dos servidores MEC, visto que no modelo não há segregação de regiões, ou seja, todos os dispositivos IoT possuem acesso a todos os servidores MEC.

O modelo é desenvolvido como um problema de minimização e otimização de variáveis, em busca do menor custo total do sistema, em termos do consumo de energia e do tempo de execução das tarefas. A experimentação é realizada em ambiente simulado, visto a praticidade e versatilidade dessa alternativa.

5.2 O Modelo de Custo do Sistema

Nessa seção é apresentado o modelo de custo do sistema. Inicialmente são apresentadas as premissas escolhidas para a construção do modelo e, após, o modelo da rede. Por fim, é formulado o modelo de computação local no dispositivo IoT, no servidor local (servidor MEC) e na *Cloud*.

5.2.1 Escolhas e premissas para o modelo

Para o desenvolvimento do modelo, além das características definidas após análise comparativa, também foram estabelecidas algumas premissas e definições que facilitam o entendimento sobre o modo de funcionamento do modelo e as considerações que ele realiza ao longo do processo de minimização de custos e otimização de variáveis.

É sabido que muitas tarefas possuem *deadline* de conclusão, ou seja, possuem um tempo máximo para que haja uma resposta com os resultados do processamento da tarefa. Logo, algumas das tarefas analisadas pelo modelo possuem essa componente de *deadline* com um tempo máximo definido para que o dispositivo receba a resposta. Essas tarefas são chamadas de **tarefas críticas** e as demais são **tarefas comuns**. Desse modo o tempo de execução e os tempos de comunicação são considerados para decidir se uma tarefa executa ou não no dispositivo local.

Uma situação considerada a respeito do *deadline* é a impossibilidade de execução da tarefa em tempo hábil fora do dispositivo, ao mesmo tempo em que o nível de bateria do dispositivo está baixo. Nesse caso, estando a bateria em um nível crítico, é priorizada a execução da tarefa, ou seja, a tarefa não é encerrada automaticamente, mesmo que isso leve o dispositivo a consumir toda a sua bateria e ser desligado.

Os dispositivos locais e os servidores locais com mais de um núcleo podem realizar execução simultânea de tarefas, cada tarefa em um núcleo. A escolha do núcleo utilizado é feita pelo algoritmo de escalonamento, com base nas características de cada

um, como frequência de operação, tempo de execução e consumo de energia. Para essas duas políticas de escalonamento, se todos os recursos de processamento estiverem sendo utilizados, a tarefa deverá aguardar até que um núcleo esteja livre. Para execuções na *Cloud* assume-se que os recursos são ilimitados e não é contabilizado tempo de espera para liberação de recursos de processamento.

Para os núcleos do sistema, sejam eles de dispositivos IoT ou de servidores da camada MEC, é feita a contabilização da **energia *idle*** e **energia dinâmica**, que somadas à energia gasta nas transmissões de dados entre as camadas da arquitetura, compõem a energia total consumida no sistema. A energia *idle* refere-se ao consumo gerado pelos núcleo que estão parados, sem carga de trabalho. Já a energia dinâmica ocorre quando um núcleo está realizando processamento.

A segurança em sistemas de borda de rede também é uma questão muito discutida atualmente. Em sistemas consolidados e centralizados, como os *Data Centers*, há uma imensidão de tecnologias de segurança, como autenticação de acesso, criptografia de dados, implementação de *firewalls*, uso de VPNs e *Sistemas de Prevenção de Invasão* (*Intrusion Prevention System*, ou IPS). Embora seja importante trabalhar no aprimoramento da segurança das redes de borda, este trabalho não lida com aspectos de segurança, pois o tópico é suficiente para justificar um trabalho à parte.

Outro ponto importante é o modo como é encarado o nível das baterias nos dispositivos móveis. O algoritmo de escalonamento periodicamente realiza o monitoramento do nível de bateria dos dispositivos e isso permite decidir entre execução da tarefa no dispositivo, no servidor local ou na *Cloud*. O modelo verifica se o nível de bateria do dispositivo IoT está em uma zona segura, que não permita que o dispositivo morra por falta de energia.

As variáveis de custo inseridas no modelo possuem forte dependência da frequência de operação dos núcleos, mas também da tensão de alimentação dos diferentes *hardwares*. Desse modo, o algoritmo de escalonamento busca essas informações nas CPUs presentes na rede, via requisições de software. Previamente, o algoritmo de escalonamento também é abastecido com os valores de consumo energético e latência das comunicações de dados realizadas entre as camadas (sem fio e cabeada).

O tamanho das tarefas, em termos de ciclos de processamento, é outra variável crítica expressa no modelo de custo. É possível definir o tamanho das tarefas, pois geralmente as tarefas em dispositivos IoT são periódicas, e com uma amostragem prévia é possível estimar seu tempo de execução. Assim assume-se que a amostragem prévia já

foi realizada e é sabido o perfil de criticidade de cada tipo de tarefa da rede, informação que é passada ao algoritmo de escalonamento no momento de criação da tarefa.

O consumo global de energia de uma rede, em um cenário real, agrega os consumos de todos os dispositivos de rede, tais como roteadores, *gateways*, *switches*, repetidores e *access points*. Porém, a fim de simplificar o cálculo de custo energético da rede, os gastos energéticos dos dispositivos de rede foram abstraídos, para que o foco fosse tão somente nos dispositivos móveis, servidores locais e servidores da *Cloud*.

5.2.2 Modelo da rede

A rede do sistema possui um conjunto finito $D = \{1, 2, 3, ..., D\}$ de dispositivos inteligentes móveis, $S = \{1, 2, 3, ..., S\}$ de servidores locais e $W = \{1, 2, 3, ...W\}$ canais de comunicação sem fio. Cada dispositivo ou servidor local pode ter zero ou mais tarefas, sem que haja um limite expresso de tarefas por dispositivo ou servidor local, sendo o limite implícito de tarefas em execução igual ao número de núcleos. O sistema como um todo possui $A = \{1, 2, 3, ..., \infty\}$ tarefas. Cada tarefa A_i é representada por uma tupla $A_i = (C_i, s_i, d_i, t_i)$, para $i \in A$.

Para a tarefa A_i, C_i representa a quantidade de ciclos de processamento necessários para a execução completa da tarefa, s_i e d_i representam, respectivamente o tamanho do código e entrada de dados e t_i representa o *deadline* da tarefa, ou seja, o tempo máximo permitido para a completa execução da tarefa. Para A_i o *deadline* da tarefa pode ou não estar definido. Se $t_i = \infty$, então não há *deadline* para a tarefa, já se $t_i = t'$ e $t_i > 0$, quer dizer que a tarefa A_i possui tempo t', a partir do momento de sua criação, para ser concluída.

A escolha de um canal sem fio para uma tarefa $i \in A$ é determinada por h_i, na qual h_i é um elemento de *H*. Um mesmo canal w pode estar associado a n tarefas. O valor atribuído a h_i depende da política de escalonamento selecionada. Se a tarefa *i* for executada localmente no próprio dispositivo, então $h_i = 0$, indicando que não há canal de comunicação associado à tarefa. Se a alocação for no servidor local ou na *Cloud*, então $h_i \in W \cup 0$.

5.2.3 Computação local no dispositivo IoT

Cada dispositivo móvel $j \in D$ da rede possui um ou mais núcleos. Assim, os núcleos de processamento disponíveis na rede do dispositivo *j* são dadas por $PL_j = \{pl_{j,1}, pl_{j,2}, pl_{j,3}, ..., pl_{j,n}\}$. Todo núcleo $pl_{j,k} \in PL_j$ possui valor 0 ou 1, 0 se estiver vago e 1 se estiver ocupado. Para núcleo vago, é contabilizada a energia *idle* e para núcleo ocupado é contabilizado a energia dinâmica ($E_{i,local,dinamico}$). Cada núcleo possui uma frequência de operação ($f_{local,j,k}$), ou capacidade de processamento, uma capacitância comutativa efetiva ($C_{local,j,k}$), dependendo da arquitetura da CPU (YU; WANG; GUO, 2018) e uma tensão de alimentação ($V_{local,j,k}$). Além disso, cada tarefa *i* possui uma quantidade total de ciclos de processamento (CT_i) para sua execução. Desse modo é possível calcular o tempo total de execução da tarefa $i \in A$, pelo núcleo $j \in PL$, bem como a energia dinâmica consumida durante a execução. Esses cálculos são expressos pelas equações (LIU et al., 2007):

$$T_{i,local,dinamico} = \frac{CT_i}{f_{local,j,k}} \tag{5.3}$$

$$P_{i,local,dinamico} = C_{local,j,k} * V^2_{local,j,k} * f_{local,j,k} \tag{5.4}$$

$$E_{i,local,dinamico} = P_{i,local,dinamico} * T_{i,local,dinamico} \tag{5.5}$$

Se todos os núcleos do dispositivo estiverem ocupados é adicionado a $T_{i,local,dinamico}$ um custo de espera definido como $T_{i,espera}$ que será igual ao menor $T_{i,local,dinamico}$ das demais tarefas em execução, visto que assim que esta tarefa finalizar, o núcleo ficará disponível. O algoritmo de escalonamento se encarrega de identificar quais núcleos pertencem a quais dispositivos móveis. Assim, a equação de $T_{i,local,total}$ no caso de todos os núcleos do dispositivo estarem ocupados é dada por:

$$T_{i,local,total} = \frac{CT_i}{f_{local,j,k}} + T_{i,espera} \tag{5.6}$$

Considerando o nível de bateria e a latência como restrições do modelo, um dispositivo D_j deve decidir se é mais adequado executar a tarefa local ou remotamente. Sendo o nível de bateria um fator crítico na decisão o sistema irá prezar por uma política que reduza o consumo de energia. Já se a tarefa precisa ser concluída rapidamente, a ten-

dência é pela escolha de uma política que economize tempo de execução. São utilizados u_{localT} e $u_{localE} \in [0, 1]$ para representar o peso dessas duas limitantes, tempo e energia, respectivamente. O custo local de **uma tarefa *i*** pode ser expresso por:

$$Custo_{i,local,dinamico} = u_{localT} * T_{i,local,total} + u_{localE} * E_{i,local,dinamico} \tag{5.7}$$

Na Equação 5.7 $0 \leq u_{localT} \leq 1$, $0 \leq u_{localE} \leq 1$ e $u_{localT} + u_{localE} = 1$, ou seja, $u_{localT} = 1 - u_{localE}$. Conforme (WANG et al., 2018), o parâmetro u_{localE} funciona como um *trade-off* entre o tempo de execução e o consumo energético. A decisão de descarregar a tarefa, dando preferência a minimizar o tempo de execução ou o consumo energético, é determinada pelo parâmetro ponderado u_{localE}.

Para o núcleo que for escolhido para a execução da tarefa é atribuído o valor 1, ou seja, $pl_{j,k} = 1$. Quando a execução finaliza é atribuído o valor 0 à variável correspondente, ou seja, $pl_{j,k} = 0$, sinalizando que o núcleo está vago.

Tanto a quantidade de ciclos de processamento para execução da tarefa, quanto a capacitância da CPU são variáveis imutáveis. Porém, a frequência e tensão dos núcleos pode ser ajustada para minimizar o custo local.

5.2.4 Computação local no servidor local (servidor MEC)

Do mesmo modo como ocorre para os dispositivos móveis da rede, os servidores locais possuem múltiplos núcleos de processamento. Assim, os núcleos disponíveis em um servidor local S_j são dadas por $PS_j = \{ps_{j,1}, ps_{j,2}, ps_{j,3}, ..., ps_{j,n}\}$. Cada núcleo $ps_{j,k}$ possui uma frequência de operação ($f_{mec,j,k}$), coeficiente energético de capacitância da arquitetura do chip ($C_{mec,j,k}$) e tensão de alimentação ($V_{mec,j,k}$).

As comunicações que ocorrem entre dispositivo e servidor local utilizam canais de comunicação sem fio. Toda vez que uma tarefa *i* de um dispositivo móvel *m* é escalonada e descarregada para o servidor local ou para a *Cloud*, um canal de comunicação sem fio é escolhido e armazenado em h_i. A taxa de transferência de dados para descarregar a tarefa *i* pode ser calculada conforme a *Fórmula de Shannon* (YU; WANG; GUO, 2018):

$$r_i(h) = W * \log_2 \left(1 + \frac{p_m * g_{j,m}}{N + I_i}\right) \tag{5.8}$$

Para a Equação 5.8, W é a largura de banda do canal em *Hz*, $g_{j,m}$ representa o

ganho do canal entre o dispositivo móvel *m* e um servidor local, representado por *j*. A variável p_m representa a potência de transmissão do dispositivo móvel *m* ao descarregar a tarefa *i*, seja para os servidores locais ou para a *Cloud*. *N* representa a potência do ruído térmico do canal sem fio alocado em h_i e I_i representa a potência de interferência mútua entre os usuários daquele canal. A interferência I_i do canal sem fio armazenado em h_i pode ser calculada por (YU; WANG; GUO, 2018):

$$I_i = \sum_{n \in A, n \neq i | \{i\}: h_n = h_i} p_{m'} * g_{j',m'} \tag{5.9}$$

Na equação de I_i, todos os dispositivos móveis m', com suas respectivas tarefas *n*, que também tenham escolhido o mesmo canal sem fio h_i da tarefa *i* no dispositivo *m*, têm seus ruídos contabilizados, pois as diferentes transmissões realizadas sobre o mesmo canal causam interferência umas nas outras. Assim, se h_n igual a h_i, então diferentes tarefas estão sendo transmitidas no mesmo canal, agregando ruído à potência de interferência mútua.

Diferentemente do processamento no dispositivo, no processamento no servidor local é necessário que os dados (d_i) e código (s_i) da aplicação sejam transmitidos para o servidor. Assim, o tempo necessário para que um dispositivo móvel $j \in D$ transmita os dados de uma tarefa *i* para um núcleo $ps_{j,k} \in PS_j$ é dado pela equação (YU; WANG; GUO, 2018):

$$T_{i,mec-up}(h_i) = \frac{s_i + d_i}{r_i(h_i)} \tag{5.10}$$

Do mesmo modo, assim que a execução for finalizada, os resultados gerados, representados por d'_i, devem ser transmitidos de volta à origem, ou seja, ao dispositivo que criou a tarefa. O tempo de *download* dos resultados do servidor local ao dispositivo móvel é dado por:

$$T_{i,mec-down}(h_i) = \frac{d'_i}{r_i(h_i)} \tag{5.11}$$

O tempo de execução da tarefa *i* no servidor local *k* é dado por:

$$T_{i,mec,dinamico} = \frac{CT_i}{f_{mec,j,k}} \tag{5.12}$$

Além disso, há a componente $T_{i,espera}$ que possui valor igual ao menor $T_{i,mec,dinamico}$ no servidor local PS_j, visto que é necessário aguardar a tarefa mais rápida finalizar e li-

berar o núcleo para dar sequência à execução. Caso haja um núcleo disponível em PS_j, então $T_{i,espera}$ será igual a zero. Portanto, o tempo exigido para completar a execução da tarefa no servidor local é dado por:

$$T_{i,mec,total} = T_{i,mec-up}(h_i) + T_{i,mec,dinamico} + T_{i,mec-down}(h_i) + T_{i,espera} \tag{5.13}$$

A energia consumida para a transmissão dos dados do dispositivo móvel para o servidor local é dada pelo tempo decorrido durante a transmissão de código e dados (s_i e d_i) da tarefa *i* multiplicado pela potência de transmissão para a tecnologia sem fio escolhida no sistema ($p_{wireless}$), conforme segue:

$$E_{i,mec-up}(h_i) = p_{wireless}(s_i, d_i) * T_{i,mec-up}(h_i) \tag{5.14}$$

Esse custo de consumo energético pode ser pequeno ou grande, a depender da quantidade de dados transmitida (s_i e d_i). Além disso, é importante notar, que assim como $T_{i,mec-up}(h_i)$, $E_{i,mec-up}(h_i)$ é calculado em função do canal de comunicação utilizado, pertencente ao conjunto W.

De modo semelhante, a transmissão dos resultados do processamento de volta ao dispositivo IoT também demanda recursos energéticos. A única diferença está no volume de dados, pois nesta etapa apenas os resultados (d_i') são transmitidos. Esse consumo energético é expresso por:

$$E_{i,mec-down}(h_i) = p_{wireless}(d_i') * T_{i,mec-down}(h_i) \tag{5.15}$$

Por fim, a potência dinâmica e o consumo energético do processamento da tarefa *i* no núcleo $ps_{j,k} \in PS_j$ é dada por:

$$P_{i,mec,dinamico} = C_{mec,j,k} * V_{mec,j,k}^2 * f_{mec,j,k} \tag{5.16}$$

$$E_{i,mec,dinamico} = P_{i,mec,dinamico} * T_{i,mec,dinamico} \tag{5.17}$$

O consumo dinâmico total é uma combinação dos consumos parciais e expresso pela equação:

$$E_{i,mec,total} = E_{i,mec-up}(h_i) + E_{i,mec,dinamico} + E_{i,mec-down}(h_i) \tag{5.18}$$

Do mesmo modo como descrito para a formatação da equação de custo local, a equação de custo para o servidor local é expressa da seguinte forma:

$$Custo_{i,mec,dinamico} = u_{mecT} * T_{i,mec,total} + u_{mecE} * E_{i,mec,total} \tag{5.19}$$

Os coeficientes $u_{mecT} + u_{mecE} \in [0, 1]$, onde $0 \leq u_{mecT} \leq 1$, $0 \leq u_{mecE} \leq 1$ e $u_{mecT} + u_{mecE} = 1$. Esses coeficientes podem ser dimensionados para priorizar um ou outro custo, dependendo da necessidade das aplicações do sistema ou do objetivo da rede. Caso a prioridade seja a redução do consumo de energia o coeficiente u_{mecE} deve ter maior peso, pois por representar um maior valor no custo final há melhores condições de ser minimizado. O mesmo se aplica para a latência, pois caso as aplicações do sistema possuam *deadlines* muito curtos, u_{mecT} pode ter um peso maior do que u_{mecE}, tendo mais possibilidades de ser minimizado.

5.2.5 Custo em *idle* para dispositivos móveis e servidores locais

A energia *idle* multiplica a potência específica de cada núcleo quando em *idle* pelo tempo em que ela está nessa condição, dado por t'. O cálculo de consumo de energia para núcleos livres é dado por:

$$E_{local,idle} = \sum_{j=1}^{m} \sum_{i=1}^{n} P_{idle,pl_{j,i}} * t', \text{ se } pl_{j,i} = 0 \tag{5.20}$$

$$E_{mec,idle} = \sum_{j=1}^{m} \sum_{i=1}^{n} P_{idle,ps_{j,i}} * t', \text{ se } ps_{j,i} = 0 \tag{5.21}$$

5.2.6 Computação remota na *Cloud* centralizada

As equações para latência e consumo energético na *Cloud* centralizada são muito semelhantes às do servidor local. A equação para o cálculo total da latência não possui a componente $T_{i,espera}$, pois assume-se que a *Cloud* possui recursos ilimitados, não havendo necessidade de esperar por um núcleo livre. Além disso, há o tempo gasto para transmissão celular de dados entre o dispositivo móvel e o servidor MEC e entre o servidor MEC e a *Cloud*, ou seja, duas transmissões de dados para *upload* e duas para *download*. Já a equação para o consumo de energia considera a energia gasta nas transmissões de dados,

conforme segue:

$$T_{i,cloud,total} = T_{i,mec-up}(h_i) + T_{i,mec-cloud-up} + T_{i,mec-cloud-up} + T_{i,cloud,dinamico} + T_{i,mec-down}(h_i) \quad (5.22)$$

$$E_{i,mec-cloud-up} = p_{wired}(s_i, d_i) * T_{i,mec-cloud-up} \quad (5.23)$$

$$E_{i,mec-cloud-down} = p_{wired}(r_i) * T_{i,mec-cloud-down} \quad (5.24)$$

$$E_{i,cloud,dinamico} = p_{cloud,dinamico} * T_{i,cloud,dinamico} \quad (5.25)$$

$$E_{i,cloud,total} = E_{i,servidor-up}(h_i) + E_{i,mec-cloud-up} + E_{i,cloud,dinamico} + E_{i,mec-cloud-down} + E_{i,mec-down}(h_i) \quad (5.26)$$

As Equações 5.23 e 5.24 representam o custo energético para as transmissões de dados em rede cabeada, via fibra óptica, e a Equação 5.25 representa o consumo dinâmico da *Cloud*. Não há utilização da técnica de DVFS pelo escalonador do sistema na *Cloud*, pois esse recurso fica restrito ao provedor do serviço. Por fim, não se contabiliza o custo em *idle* da *Cloud*, visto que a oferta de núcleos de processamento é virtualmente infinita, não fazendo sentido, portanto, contabilizar esse custo.

$$Custo_{i,cloud,dinamico} = u_{cloudT} * T_{i,cloud,total} + u_{cloudE} * E_{i,cloud,total} \quad (5.27)$$

Os coeficientes seguem a expressão u_{cloudT}+$u_{cloudE} \in [0, 1]$, na qual $0 \leq u_{cloudT} \leq 1$, $0 \leq u_{cloudE} \leq 1$ e $u_{cloudT} + u_{cloudE} = 1$. Para reduzir a latência, a escolha mais acertada é que o coeficiente de latência seja muito maior do que o coeficiente para consumo energético, ou seja, $u_{cloudT} \gg u_{cloudE}$. Desse modo a minimização de custo se dará principalmente sobre a latência, visto que os custos de consumo energético para o sistema, quando há comunicação com a *Cloud*, seriam mínimos. A escolha correta de coeficientes da computação em *Cloud* também evita que o algoritmo de escalonamento priorize a alocação de tarefas para a *Cloud*, pois dependendo do balanceamento realizado o algoritmo

escolheria a execução na *Cloud*.

5.2.7 Equação de custo do sistema - problema de *programação linear inteira* (ILP)

Para cada uma das políticas de escalonamento do sistema há uma equação de custo específica. O custo de cada tarefa *i* é dado pela minimização da equação a seguir.

$$\begin{aligned} Custo_{i,dinamico} = min(&\alpha * Custo_{i,local,dinamico}, \\ &\beta * Custo_{i,mec,dinamico}, \\ &\gamma * Custo_{i,cloud,dinamico}) \end{aligned} \tag{5.28}$$

Na Equação 5.28 $\alpha, \beta, \gamma \in [0, 1]$ e $\alpha + \beta + \gamma = 1$. Os três coeficientes podem ser dimensionados de modo a priorizar a computação em uma ou outra política de escalonamento. Porém eles funcionam de modo diferente daqueles utilizados nas equações de custo específicas de cada política, na qual um coeficiente de valor alto determina se a latência será minimizada, se aplicado à latência, ou se o consumo de energia elétrica será minimizado, se aplicado ao consumo elétrico. Aqui, um coeficiente de valor elevado, indica que o custo da política em questão é elevado, havendo menores possibilidades daquela política de escalonamento ser escolhida. A escolha pelo menor valor de custos, dentre os três calculados, também define qual a política de escalonamento que será escolhida para aquela tarefa, se execução local no dispositivo, local no servidor local, ou remota na *Cloud*.

Porém, cabem alguns cuidados para o correto dimensionamento dos coeficientes. Digamos que o coeficiente α seja elevado, com valor superior aos coeficientes β e γ. Para a equação de custo do sistema o custo específico da política de escalonamento local, no dispositivo, seria elevado. Assim, as políticas de escalonamento no servidor local e na *Cloud* seriam priorizadas em detrimento da política de escalonamento local no próprio dispositivo. Por isso, é de suma importância que tais coeficientes sejam dimensionados de modo adequado para que não haja desbalanceamento na escolha das políticas.

Conclui-se, portanto, que o custo percebido pela equação de custo do sistema pode não ser, necessariamente, o custo real do sistema, pois os coeficientes podem ser dimensionados para que uma determinada política seja mais utilizada do que as demais, mesmo sendo mais custosa em termos de recursos energéticos e em latência. Com os três coeficientes ajustados com o mesmo peso, não há favorecimento entre uma política ou

outra, pois competem em condições iguais de importância.

Visando a minimização da latência nas conclusões de tarefas e da energia consumida pelo sistema, é preferível que γ seja mais elevado e α e β semelhantes, de modo a priorizar a execução das tarefas localmente, seja no próprio dispositivo ou no servidor local. Essa preferência se dará, pois qualquer execução na *Cloud* contribuirá com um custo elevado para a equação de custo final do sistema.

A equação de custo do sistema considera os custos minimizados para a execução de cada tarefa, ou seja, custo dinâmico. Além disso ela considera os custos energéticos em *idle* para dispositivos móveis e servidores locais. Portanto, para $i \in A$ e $A = \{1, 2, 3, ..., A\}$, o **problema de minimização do custo total do sistema pode ser formulado pela equação**:

$$Custo_{sistema} = \sum_{i=1}^{A} Custo_{i,dinamico} + \alpha * E_{local,idle} + \beta * E_{mec,idle} \tag{5.29}$$

5.2.7.1 Coeficientes da equação de custo do sistema

Os coeficientes aplicados à energia *idle* são os mesmos definidos na fase do custo minimizado por tarefa, de modo a manter a mesma magnitude de peso no custo do sistema. Essa equação está sujeita a alguns coeficientes definidos pelo usuário que objetivam estabelecer graus de importância mínimos quanto à preferência em realizar a alocação da tarefa localmente, no servidor local ou na *Cloud*, ou mesmo em dar preferência para minimizar o tempo de execução da tarefa ou o seu consumo de energia. Cabe ao administrador do sistema conhecer as suas necessidades e ajustar os coeficientes do modo mais adequado. Os coeficientes utilizados pela equação de custo do sistema são listados, conforme segue:

- $u_{localT}, u_{localE} \in [0, 1]$ e $u_{localT} + u_{localE} = 1$
- $u_{mecT}, u_{mecE} \in [0, 1]$ e $u_{mecT} + u_{mecE} = 1$
- $u_{cloudT}, u_{cloudE} \in [0, 1]$ e $u_{cloudT} + u_{cloudE} = 1$
- $\alpha, \beta, \gamma \in [0, 1]$ e $\alpha + \beta + \gamma = 1$
- $T_{i,local,total}, T_{i,mec,total}, T_{i,cloud,total} < t_i$, para $t_i \in A_i$

De modo geral, todos os coeficientes utilizados, seja nas equações específicas de custo ou na equação total de cada tarefa compreendem o intervalo real de 0 a 1. Além

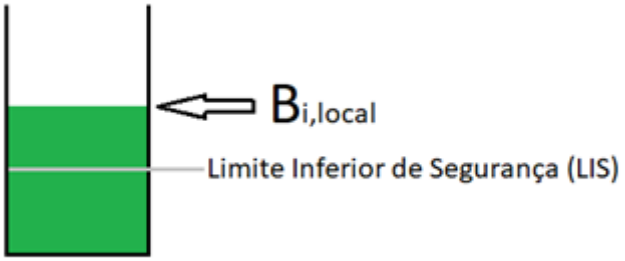
disso, o tempo de conclusão da tarefa deve ser respeitado para pelo menos uma das políticas de escalonamento. Aquelas políticas que violarem essa restrição estarão automaticamente descartadas do rol de escolha de políticas de escalonamento viáveis para a execução da tarefa.

5.2.7.2 Nível de bateria dos sispositivos IoT

Outra restrição muito importante refere-se ao nível de bateria dos dispositivos móveis. Foi comentado na Seção 5.2.3 que a bateria é um fator crucial para o sucesso da execução local nessa modalidade de processamento, visto que o dispositivo deve possuir energia suficiente para a execução e conclusão da tarefa.

No algoritmo de escalonamento a bateria é analisada previamente e, estando acima de um *Limite Inferior de Segurança* (LIS), dá-se sequência à alocação de recursos e posterior execução da tarefa. A execução é realizada até seu término, mesmo que isso leve o nível da bateria a zero e à inoperabilidade do dispositivo. Porém, se ao término da execução o dispositivo permanecer ligado, mas com nível de bateria abaixo do LIS, ele não poderá executar novas tarefas até que o limite mínimo seja reestabelecido, sendo necessário, portanto, conectá-lo a uma fonte de energia externa. Logo, é possível que o dispositivo possua um nível de bateria saudável, acima de LIS, porém, após a execução de uma tarefa, fique com o nível abaixo do LIS. A Figura 5.4 ilustra os limites da bateria do dispositivo móvel.

Figura 5.4: Nível de bateria do dispositivo e limite inferior de segurança.



Referente às políticas de escalonamento, há restrição de utilização da política de alocação local caso a bateria esteja muito baixa e da política de alocação no servidor local caso a bateria não consiga suprir a energia para a transmissão de dados. O cálculo de custo do sistema estará sujeito às seguintes restrições quanto ao nível de bateria do dispositivo:

- $B_{i,local} > E_{i,local}$

- $B_{i,local} > E_{i,servidor-up}(h)$
- $B_{i,local} > LIS$

$B_{i,local}$ representa o nível de bateria do dispositivo móvel gerador da tarefa *i*. Para a execução local no dispositivo todo o consumo de energia deve ser suprido pela bateria, já para as execuções fora do dispositivo, a bateria deve suprir apenas o consumo para a transmissão de dados e código da tarefa ao destino. Assim, se a execução da tarefa for no servidor local ou na *Cloud* e ao enviar os resultados de volta ao dispositivo ele esteja desligado, as tarefas são canceladas. Já na execução local, a tarefa é cancelada automaticamente, pois o dispositivo desliga ao atingir nível de bateria zero e não pode dar continuidade à execução.

$B_{i,local}$ irá sofrer oscilações, seja pelo consumo da bateria ou carregamento. O consumo da bateria é natural, já o carregamento ocorre caso o dispositivo IoT seja conectado a uma fonte de energia. Esse comportamento é esperado pelo sistema.

Além disso, mesmo que as restrições sejam atendidas, o dispositivo pode ficar sem energia, pois mesmo com bateria saudável no início da execução ela pode acabar durante o processamento. Em caso de indisponibilidade do dispositivo IoT, as tarefas do dispositivo são canceladas.

5.3 Considerações Sobre o Capítulo

Este capítulo apresentou as características da proposta do modelo do sistema e introduziu premissas para definir o modelo de custo, com todas as variáveis utilizadas nas equações de custo para computação local no dispositivo IoT, computação local no servidor MEC e computação remota na *Cloud*. As equações de custo parciais foram utilizadas para compor a equação de custo do sistema.

6 ALGORITMO DE ESCALONAMENTO

Neste capítulo é apresentado o algoritmo de escalonamento TEMS (*Time and Energy Minimization Scheduler*) que provê uma solução alternativa à resolução integral do modelo ILP do sistema que obtém uma solução ótima. O algoritmo proposto obtém uma solução próxima da solução ótima, porém em tempo reduzido e com menor demanda computacional. A complexidade do algoritmo proposto é de $O(n^2)$.

6.1 Descrição do Problema

Conforme descrito no Capítulo 5, o problema proposto foi desenvolvido como um problema de otimização de *Programação Linear Inteira* (*Integer Linear Programming*, ou ILP), o que garante ao sistema uma solução ótima quanto à minimização dos custos (CAPUCHO; RESENDO, 2013). Porém, a resolução do problema de programação linear inteira apresenta um custo computacional elevado, agregando latência à rede. Considerando um cenário dinâmico com o surgimento de novas tarefas a todo instante e a necessidade de novas rodadas de cálculo do modelo ILP, a situação se agrava com mais latência na rede. Novas rodadas de cálculo do modelo ILP para obtenção da solução ótimo são, portanto, indesejáveis, conflitando com a necessidade de alocar as tarefas rapidamente.

Desse modo, foi desenvolvido um algoritmo heurístico para o escalonamento das tarefas no sistema proposto, para obter uma boa solução, próxima da solução ideal, mas com custo computacional reduzido. Baseado em observações do problema de programação linear inteira, para pequenas instâncias a alocação de tarefas com maior demanda por recursos computacionais deve ser priorizada para aumentar a eficiência do sistema e maximizar a execução paralela em nível de tarefas. Idealmente ambas variáveis de consumo de energia e tempo de processamento das tarefas devem ser minimizadas, mas no conflito de decisão entre minimizar uma ou outra, a redução do consumo energético é priorizada. Porém, para os casos específicos de tarefas críticas, nas quais o tempo de processamento deve ser respeitado, a minimização ocorrerá, naturalmente, sobre o tempo de processamento. O algoritmo heurístico detalhado é apresentado na seção seguinte.

6.2 Time and Energy Minimization Scheduler (TEMS)

O algoritmo heurístico *Time and Energy Minimization Scheduler* (TEMS) foi desenvolvido objetivando solucionar as demandas do sistema de consumo energético e tempo de processamento das tarefas. A estrutura geral do algoritmo é apresentada na imagem do Algoritmo 1.

Algoritmo 1: Time and Energy Minimization Scheduler (TEMS)

```
Result: Mapeamento de tarefas às unidades de processamento
execute Etapa 1 - Coleta de informações do sistema e inicialização
repeat
    execute Etapa 2 - Escolha do local de processamento
    execute Etapa 3 - Monitor de conclusão de tarefas
    execute Etapa 4 - Novas tarefas e nível de bateria dos dispositivos
until usuário decidir manter execução;
```

O algoritmo desenvolvido é dividido em quatro etapas. Na Etapa 1 ocorre a coleta de informações que compõem os conjuntos de dados utilizados, tais como, nível de bateria dos dispositivos IoT, capacidade de processamento, frequências mínimas e máximas de operação e coeficiente de eficiência energética do *hardware* dos dispositivos locais e servidores locais, bem como o conjunto de tarefas criadas e suas características. São definidos os conjuntos dos dispositivos móveis, servidores locais, conjunto de canais de comunicação sem fio e conjunto de decisões de escolha dos canais sem fio associados a cada tarefa. Também são definidos os coeficientes de balanceamento utilizados na equação de minimização.

Na Etapa 2 é realizada a escolha do local de processamento de cada tarefa. Primeiramente as tarefas são classificadas entre tarefas críticas e tarefas não críticas, sendo que as tarefas críticas possuem uma componente de tempo futuro limite para sua execução, um *deadline*, que deve ser respeitado visto a sensibilidade dos resultados do processamento. As tarefas críticas são ordenadas de modo crescente por *deadline* de finalização, enquanto as tarefas não críticas são enfileiradas em uma lista para consulta. São calculados os custos de tempo de processamento e consumo energético de processamento pelos núcleos, bem como o tempo e consumo energético das transmissões de dados, caso a alocação seja feita no servidor MEC ou na *Cloud*. Verifica-se a disponibilidade de núcleos nos dispositivos locais e servidor local, além do nível de bateria dos dispositivos locais. O nível de bateria deve estar acima do LIS e também deve comportar a estimativa de consumo energético da tarefa. Após, as tarefas críticas são alocadas em ordem, do menor *deadline*

Algoritmo 2: Etapa 1 – Coleta de informações do sistema e inicialização

```
1  D = Conjunto de dispositivos móveis
2  foreach D_j ∈ D do
3  |  coletar nível de bateria B_{j,local}
4  |  definir LIS para nível de bateria
5  |  PL_j = Conjunto núcleos pertencentes ao dispositivo móvel D_j
6  |  foreach Núcleo pl_{j,k} ∈ PL_j do
7  |  |  pl_{j,k} = 0 // Indica núcleo livre
8  |  end
9  end
10 S = Conjunto de servidores locais
11 foreach Sj ∈ S do
12 |  PS_j = Conjunto núcleos pertencentes ao servidor local S_j
13 |  foreach Núcleo ps_{j,k} ∈ PS_j do
14 |  |  ps_{j,k} = 0 // Indica núcleo livre
15 |  end
16 end
17 W = Conjunto de canais de comunicação sem fio
18 A = Conjunto de tarefas
19 foreach tarefa A_i ∈ A do
20 |  definir tupla de variáveis (C_i, s_i, d_i, t_i)
21 end
22 h = Conjunto de decisões de escolha dos canais sem fio associados a cada
     tarefa
23 foreach decisão h_{i,p} ∈ h do
24 |  h_{i,0} = void, h_{i,1} = void, h_{i,2} = void // Nenhum canal de
   |      comunicação definido para tarefa i
25 end
   // Definição de coeficientes para equações de custo
26 u_{localT}, u_{localE} // Execução local no dispositivo
27 u_{mecT}, u_{mecE} // Execução local no servidor local
28 u_{cloudT}, u_{cloudE} // Execução remota na Cloud
29 α, β, γ // Equação de minimização
```

ao maior, no núcleo que oferece o menor tempo total, incluindo tempo de processamento e tempo decorrido nas transmissões de dados, mesmo que o custo final seja maior. Assim é dada prioridade à conclusão da tarefa, mesmo que outro núcleo forneça um custo calculado menor. Em seguida são alocadas as tarefas não críticas, cada uma em um núcleo que lhe confira o menor custo, derivado da relação consumo energético e tempos decorridos.

Cabe ressaltar que as alocações de tarefas em *hardwares* que proporcionam o menor consumo energético não necessariamente refletem em um menor tempo de processamento; o mesmo é válido para o consumo energético nas alocações por *deadline*. As alocações no dispositivo local, no servidor MEC ou na *Cloud* sempre buscam o menor custo final para as tarefas não críticas e o menor tempo total para as tarefas críticas.

Foi dada preferência à alocação das tarefas críticas ao invés das tarefas não críticas, primeiro devido à sensibilidade do tempo disponível até a conclusão e também porque é esperado que o sistema tenha muitas mais tarefas não críticas do que tarefas críticas. Desse modo, a alocação primeira das tarefas críticas não deve impactar a eficiência do sistema de modo significativo. É uma decisão de projeto que prima por evitar o cancelamento das tarefas, embora o objetivo principal seja reduzir o custo energético total do sistema, reduzindo também os tempos de execução na medida do possível.

Na Etapa 3 as tarefas são monitoradas quanto ao seu status, se em execução ou concluídas, e quando concluídas liberam os recursos de *hardware* tornando-os novamente disponíveis para outras alocações na Etapa 2.

Na Etapa 3, caso a tarefa esteja sendo executada na *Cloud*, não há necessidade de liberar recursos assim que a tarefa seja finalizada, pois a premissa utilizada para a *Cloud* é que ela possui recursos de *hardware* ilimitados, e, portanto, pode absorver qualquer quantidade de tarefas para execução sem precisar liberar outros recursos. Além disso, o teste de nível de bateria é realizado somente para os dispositivos móveis, visto que os servidores locais não são móveis e possuem alimentação externa que lhes permite operação ininterrupta. O cancelamento de tarefas não é um resultado desejável, porém é uma métrica importante que deve ser monitorada para identificar o nível de eficiência do algoritmo proposto. Porém o cancelamento de tarefas ocorrerá ao natural caso a carga de trabalho da tarefa seja muito grande e o *deadline* pequeno, sem que nenhuma alocação possibilite a conclusão da tarefa em tempo hábil, por mais rápido que seja o núcleo.

Por fim, na Etapa 4 são coletados dados atualizados de nível de bateria dos dispositivos móveis, bem como sobre as novas tarefas recém criadas. As Etapas 2, 3 e 4 seguem em execução contínua e paralela enquanto houver tarefas em execução, ou até que o admi-

Algoritmo 3: Etapa 2 – Escolha do local de processamento

```
// Tarefas críticas
ordenar de modo crescente as tarefas críticas por deadline de finalização
foreach tarefa A_i da lista ordenada do
    foreach Núcleo pl_{j,k} ∈ PL_j do
        if (B_{j,local} > LIS) e (B_{j,local} > E_{estimada}) e (pl_{j,k} = 0) then
            calcular tempo de execução
        end
    end
    foreach Núcleo ps_{j,k} ∈ PS_j do
        if ps_{j,k} = 0 then
            calcular tempo de execução + tempos de transmissão de dados
        end
    end
    calcular tempo de execução na Cloud + tempos de transmissão de dados
    alocar tarefa para núcleo com o menor tempo de execução calculado
    if alocação para um núcleo pl_{j,k} ou ps_{j,k} then
        setar o pl_{j,k} ou ps_{j,k} correspondente com 1, indicando núcleo ocupado
    end
end
// Tarefas não críticas
foreach tarefa A_i da lista de tarefas não críticas do
    foreach Núcleo pl_{j,k} ∈ PL_j do
        if (B_{j,local} > LIS) e (B_{j,local} > E_{estimada}) e (pl_{j,k} = 0) then
            calcular consumo de energia, tempo de execução e custo
        end
    end
    foreach Núcleo ps_{j,k} ∈ PS_j do
        if ps_{j,k} = 0 then
            calcular consumo de energia e tempo de execução da tarefa no
              núcleo e das transmissões de dados e definir o custo final
        end
    end
    calcular consumo de energia e tempo de execução na Cloud e das
      transmissão de dados e definir o custo final
    alocar tarefa para núcleo com o menor custo calculado (menor relação de
      tempo e energia consumida)
    if alocação para um núcleo pl_{j,k} ou ps_{j,k} then
        setar o pl_{j,k} ou ps_{j,k} correspondente com 1, indicando núcleo ocupado
    end
end
```

Algoritmo 4: Etapa 3 – Monitoramento de tarefas, nível de bateria dos dispositivos móveis e atualização de recursos disponíveis

```
1  repeat
2      foreach tarefa crítica A_i em execução do
3          if Tempo de execução ≥ deadline then
4              cancela tarefa
5          end
           // Teste para núcleo de dispositivo local
6          if (execução em um núcleo pl_{j,k}) e (B_{j,local} = 0) then
7              cancela tarefa
8          end
9          if tarefa finalizada then
10             setar o pl_{j,k} ou ps_{j,k} correspondente com 0, caso aplicável,
                indicando núcleo livre
11         end
12     end
13     foreach tarefa não crítica A_i em execução do
14         if (execução em um núcleo pl_{j,k}) e (B_{j,local} = 0) then
15             cancela tarefa
16         end
17         if tarefa finalizada then
18             setar o pl_{j,k} ou ps_{j,k} correspondente com 0, caso aplicável,
                indicando núcleo livre
19         end
20     end
21 until enquanto houver tarefas críticas e não críticas em execução;
```

Algoritmo 5: Etapa 4 – Monitor de novas tarefas e nível de bateria dos dispositivos móveis

```
1  repeat
2      coletar conjunto de novas tarefas
3      coletar nível de bateria dos dispositivos móveis atualizado
4  until enquanto houver tarefas críticas e não críticas em execução;
```

nistrador da rede decida encerrar os processos. Em um sistema real essas etapas seguiriam executando de modo indefinido, permitindo que o escalonador pudesse mapear as tarefas aos núcleos indefinidamente.

6.3 Complexidade

O algoritmo heurístico TEMS é composto de 4 etapas. A Etapa 1 é realizada quando ocorre a inicialização do sistema e é executada apenas uma vez. Logo, não é previsto que novos dispositivos móveis sejam agregados à rede. Nessa etapa são identificados *n* dispositivos móveis com *m* núcleos cada, *n* servidores locais com *m* núcleos cada, *n* canais de comunicação sem fio, *n* tarefas A_i com uma tupla de 4 variáveis cada, *n* decisões de escolha de canais sem fio com 3 variáveis cada e 9 coeficientes para a equação de custo. Para essa etapa temos:

$$nm + nm + n + 4n + 3n + 9 = 2nm + 8n + 9 = 2nm = O(n * m)$$

A princípio a complexidade deste algoritmo seria $O(n * m)$, porém para dispositivos móveis top de linha, haja vista o *smartphone Samsung Galaxy S10*, modelo mais atual da empresa *Samsung* no momento da pesquisa, a quantidade de núcleos se limita a 8, pois são construídos com *chips octa-core* (SAMSUNG, 2019). Dispositivos IoT mais simples como placas *Arduino Mega 2560* possuem apenas um núcleo[1]. Já servidores locais construídos com 5 placas *Raspberry Pi 4* de 4 núcleos cada (FOUNDATION, 2019) possuem 20 núcleos. Também é importante ressaltar que a quantidade de dispositivos móveis do sistema sempre será muito maior do que a quantidade de servidores locais, então mesmo que um servidor local possa ter diversos núcleos, haverá poucas unidades deste equipamento no sistema. Assim, é possível afirmar que para o contexto do sistema proposto a Etapa 1 possui complexidade $O(n)$, pois a quantidade de núcleos é previsível e pequena em cada equipamento.

A Etapa 2 possui ordenação de valores. Em *Python* a função $sort()$ executa o algoritmo *Timsort*, derivado dos algoritmos *Merge sort* e *Insertion sort*, que possui complexidade $O(n \log(n))$ nos casos pior e médio e complexidade $O(n)$ no melhor caso (PYTHON, 2019). Já a função $sort()$ do *Java 6* implementa uma versão modificada

[1] <https://store.arduino.cc/usa/mega-2560-r3>

do *Merge sort*, garantindo performance $O(n\log(n))$ (ORACLE, 2019). Esta é a complexidade da ordenação realizada na Etapa 2. É realizada uma varredura entre todas as n tarefas com todos os *n* dispositivos locais e servidores locais, em busca do núcleo que ofereça o menor tempo de execução ou menor consumo energético, conforme o tipo de tarefa, se crítica ou não. Essa etapa possui complexidade $O(n^2)$. A alocação da tarefa para um núcleo local ou para a *Cloud* ocorre em tempo $O(1)$. A ordenação é utilizada três vezes, para ordenar as tarefas críticas por *deadline*, os tempos para as tarefas críticas e os custos totais para as tarefas não críticas. A busca de núcleo e alocação ocorre duas vezes, uma para tarefas críticas e outra para tarefas não críticas. Portanto, a complexidade do algoritmo da Etapa 2 pode ser descrita como:

$$3 * O(n\log(n)) + 2 * [O(n^2) + O(1)] = 3 * O(n\log(n)) + 2 * O(n^2) + 2 * O(1) = O(n^2)$$

A Etapa 3 realiza uma série de testes simples para cada tarefa, gerando *n* iterações. O teste é realizado duas vezes, uma vez para cada conjunto de tarefas. Portanto, a complexidade da Etapa 3 é $O(n)$. A Etapa 4 é semelhante à Etapa 3, pois realiza uma varredura para coleta de novas tarefas e atualização do nível de bateria dos dispositivos IoT, tendo também complexidade $O(n)$.

Compilando os custos de complexidade, o algoritmo heurístico TEMS tem a complexidade descrita a seguir:

$$O(TEMS) = O(Etapa1) + O(Etapa2) + O(Etapa3) + O(Etapa4)$$
$$O(TEMS) = O(n) + O(n^2) + O(n) + O(n) = O(n^2)$$

Cabe uma ressalva, pois à medida que as alocações são realizadas não haverá mais necessidade de analisar todos os núcleos do sistema, mas somente aqueles disponíveis. Isso auxilia o algoritmo a executar mais rapidamente, pois diminui o número de comandos executados.

6.4 Considerações Sobre o Capítulo

Este capítulo descreveu as etapas de funcionamento do escalonador TEMS, um algoritmo heurístico que implementa uma solução para o problema de linear inteiro descrito no Capítulo 5, ou seja, uma solução ao modelo de custo do sistema. Ademais, foi calculada a complexidade do algoritmo TEMS, que pode ser solucionado em tempo polinomial.

7 IMPLEMENTAÇÃO E AVALIAÇÃO

Neste capítulo são apresentadas as aplicações utilizadas nas simulações de validação do protótipo, bem como o simulador desenvolvido para a realização dos experimentos. São também apresentadas as características de consumo energético e tempo decorrido nas comunicações de dados utilizadas na arquitetura do sistema, ou seja, para redes de fibra óptica e 5G. Além disso, são definidas as características dos dispositivos IoT, servidores MEC e Cloud quanto ao seu consumo energético, poder computacional, tempo de processamento de tarefas e faixas de operação de DVFS utilizadas nas simulações. A capacidade energética da bateria também é definida. E, por fim, é descrito o conjunto de experimentos realizados e os resultados obtidos.

7.1 Características Gerais da Implementação

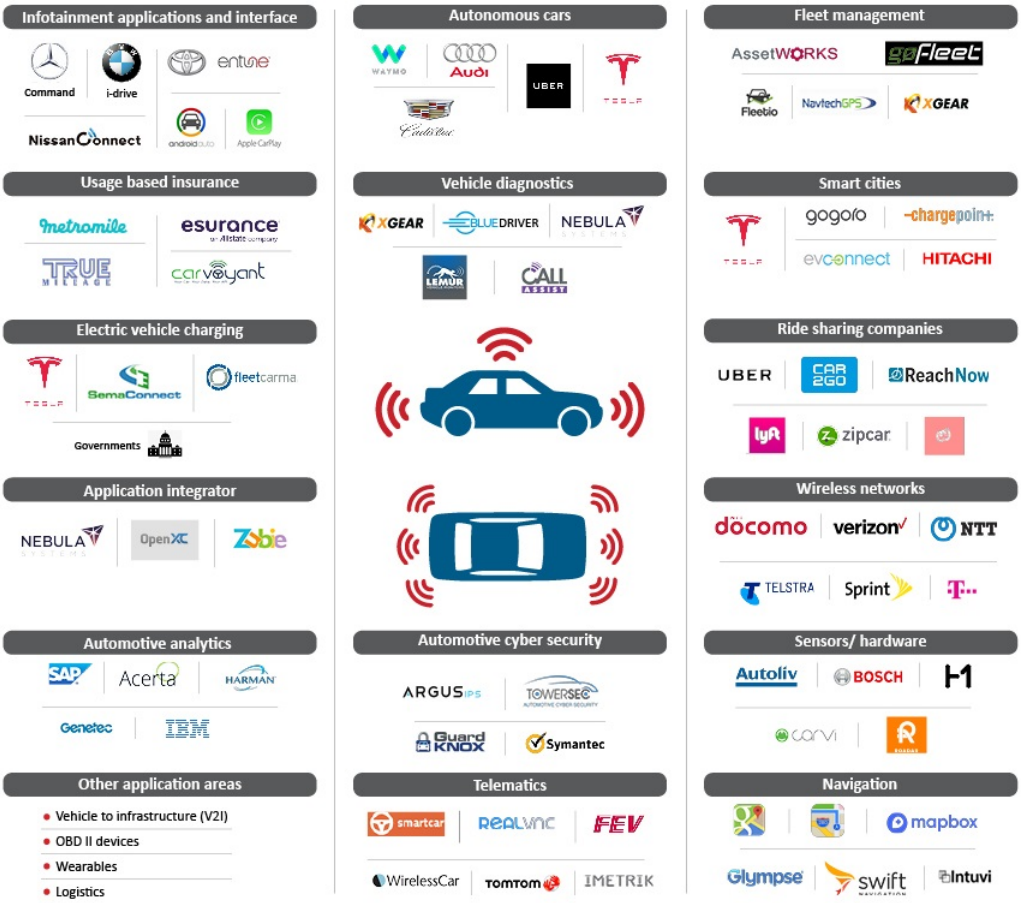
Nesta seção são apresentadas as aplicações utilizadas nas simulações e suas características e os tipos de conexão de dados com suas respectivas velocidades de transmissão, latência de comunicação e potência de transmissão. Além disso são apresentadas as características dos dispositivos IoT, dos servidores MEC e da *Cloud*, com a descrição das CPUs utilizadas em cada caso, suas faixas de frequência de operação, tensões de alimentação para cada perfil de frequência e consumo energético. Também é definida a carga de bateria para os dispositivos IoT e apresentados os detalhes sobre o simulador desenvolvido para a realização dos experimentos.

7.1.1 Aplicações veiculares

Um dos principais casos de uso para MEC com utilização da tecnologia 5G são as aplicações veiculares. Na Figura 7.1 pode-se observar o quão vasto é esse ecossistema, com dezenas de empresas participantes nos mais variados segmentos de mercado, a exemplo de aplicações para carros autônomos, gerenciamento de frota de veículos, diagnóstico, cidades inteligentes, segurança em sistemas veiculares, sensoriamento e *hardware*, navegação, dentre outros.

Para aplicações veiculares há três principais cenários para comunicação. Para veículos equipados com dispositivos de *Comunicação Dedicados de Curto Alcance* (*Dedi-*

Figura 7.1: Ecossistema veicular de aplicações inteligentes.

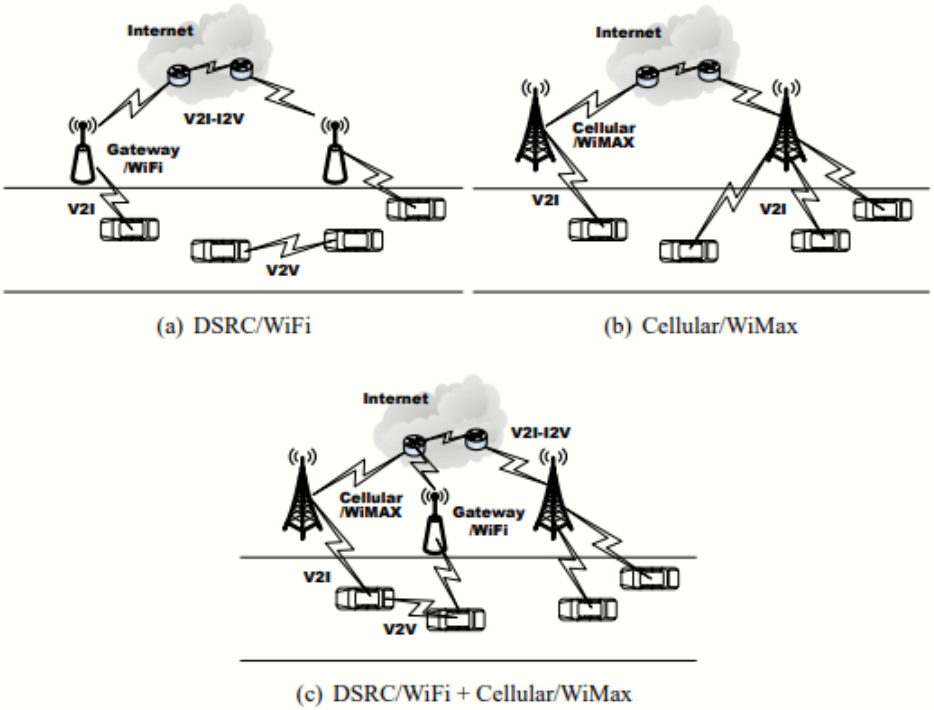


Fonte: (NETSCRIBES, 2018).

cated Short-Range Communications, ou DSRC) é possível haver comunicação *Veículo para Veículo* (*Vehicle-To-Vehicle*, ou V2V) e *Veículo para Infraestrutura* (*Vehicle-To-Infrastrtucture*, ou V2I), porém com as limitações de distância de comunicação permitidas pelas redes DSRC e WiFi, mostrado na Figura 7.2(a). Caso os veículos estejam habilitados apenas para acesso sem fio de banda larga (i.e., 5G, WiMAX), eles podem se comunicar pela Internet, conforme Figura 7.2(b). O terceiro cenário envolve os dois primeiros, ou seja, na qual os veículos podem tanto se comunicar com DSRC e conexões celulares.

Comunicações do tipo V2V, V2I ou mesmo *Veículo para Pedestre* (*Vehicle-to-Pedestrian*, ou V2P) podem ser utilizadas para aplicações inovadoras como prevenção de colisão, temporização adequada de sinais de trânsito e alertas de segurança para pedestres e ciclistas, ou seja, nas quais a comunicação ocorre próxima entre os dispositivos IoT. Já comunicações de *Veículo para Rede* (*Vehicle-To-Network*, ou V2N), através de redes celulares, permitem que serviços de *Cloud* sejam incluídos nas aplicações, oferecendo serviços de *feedback* e monitoramento sobre o tráfego em tempo real e roteamento

Figura 7.2: Cenários para comunicação veicular.



Fonte: (LEE; CHEUNG; GERLA, 2009).

dinâmico de caminhos.

As aplicações exploradas na avaliação do modelo serão para comunicações V2N, visto que não é prevista a comunicação entre os dispositivos. Dentre elas há duas classes de aplicações, separadas em geração de dados e consumo de dados (LEE; CHEUNG; GERLA, 2009).

Na primeira classe, de geração de dados, estão as aplicações veiculares que coletam informações do ambiente urbano, como imagens das estradas e veículos, imagens das placas dos carros, informações sobre o clima e outros eventos de trânsito. Essas aplicações requerem protocolos de comunicação de permitam o compartilhamento adequado das informações para que depois possam ser acessadas por outros veículos ou sistemas.

Na segunda classe estão as aplicações consumidoras de dados, a exemplo do *streaming* de vídeo para passageiros e mapas atualizados para os motoristas. Essas aplicações requerem grandes quantidades de dados, portanto conexões de alta velocidade são exigidas para o *download* das informações desejadas.

Há também uma terceira classe de aplicações, de geração e consumo de dados, na qual os veículos são tanto produtores como consumidores de conteúdo. Os principais exemplos são os informativos sobre condições das estradas e acidentes, monitoramento de congestionamento e alertas emergenciais, e.g. freios com defeito. Essas aplicações re-

querem comunicação em tempo real de alta velocidade entre os veículos ou pela Internet.

7.1.2 Aplicações escolhidas

Foram escolhidas duas aplicações. As duas são aplicações veiculares, sendo a primeira pertencente à primeira classe de aplicações, conforme descrito na Subseção 7.1.1, portanto, envolve geração de grandes volumes de dados pelos dispositivos IoT. A aplicação escolhida é de processamento de placas de veículos para auxílio às forças policiais na busca por carros roubados. Além do grande volume de informações, há também uma forte carga de processamento para analisar as imagens, sendo uma boa aplicação para verificar o comportamento do algoritmo de escalonamento, considerando os custos com transmissão de dados e processamento.

Uma imagem de resolução *1080p*, possui dimensões de 1920 colunas por 1080 linhas, o que equivale a 2.073.600 *pixels*. Uma imagem com estas dimensões e com 14 *bits* por *pixel* para a representação de cor, possui tamanho de 3,6288 *MB* (10^6 *bytes*) no formato RAW (NGUYEN; BROWN, 2016). Assim, a aplicação irá simular capturas múltiplas de lotes de imagem de 10 placas, totalizando 36,3 *MB*, para processamento por cada tarefa. Algumas dessas tarefas serão definidas como prioritárias, ou seja, terão *deadline* de conclusão, simulando situações na qual a urgência de processamento se dê por alguma ocorrência policial. O percentual estabelecido para tarefas críticas nessa aplicação é de 10% do total de tarefas criadas. Além disso, o resultado do processamento será de 1250 *bytes*, suficiente para registrar informações sobre as placas identificadas, localização e horário.

A segunda aplicação pertence à terceira classe de aplicações veiculares. Foi escolhida a aplicação de prevenção de colisões pela atuação automática dos freios ou atuação no volante. Conforme (JANSSON, 2005), cerca de 85% dos motoristas conseguem reagir a alertas dentro de 1,18 segundos, porém podem não conseguir realizar a melhor manobra para prevenir a colisão. Para velocidade baixas, de até 37 *km/h*, a eficácia de atuação dos freios é muito superior à atuação no volante, visto que a distância necessária para a total frenagem do carro nessa velocidade é de aproximadamente 10 *m*. Já para velocidades de 80 *km/h* a atuação dos freios necessita de quase 40 *m* para a total frenagem, enquanto a atuação no volante para desvio de curso é de pouco mais de 20 *m*. Portanto, ambas as formas de atuação são importantes, a depender da situação.

Na segunda aplicação o veículo possui uma série de sensores frontais e laterais

que permitem identificar objetos próximos. Dentre eles destacam-se o uso de radar e de câmeras para captura de imagens. A atuação nos freios e no volante possui latência de 0,3 *s* e 0,2 *s*, respectivamente, para surtir efeito, tão logo o sistema acione essas formas de prevenção. Assim, é importante que a aplicação seja executada o mais rapidamente possível, minimizando os tempos envolvidos desde a atuação até seus efeitos na prevenção da colisões. Para essa aplicação é definida uma taxa de geração de tarefas de 100 milésimos de segundo (*ms*) e 50% de tarefas críticas, devido à criticidade da aplicação. A entrada para a Aplicação 2 é de 4MB (10^6 *bytes*), o que equivale à análise de uma imagem RAW somada a demais dados sensoriais. As tarefas analisam as imagens e informações sensoriais capturadas ao redor do veículo e o resultado do processamento é definido em 625 *bytes*.

Tabela 7.1: Características das aplicações escolhidas.

Características	**Aplicação 1**	**Aplicação 2**
Taxa de geração (segundos)	10	0,1
Entrada (mega *bytes* - MB)	36,3	4
Resultados (bytes)	1250	625
Carga computacional (milhões de ciclos de processamento)	2000	2
Tarefas críticas (%)	10	50
Deadline para tarefas críticas (ms)	500	100

A Tabela 7.1 resume as características das aplicações que serão simuladas. A Aplicação 1, por apresentar uma maior quantidade de dados para processamento também possui carga computacional maior e para as tarefas críticas é requerido um *deadline* de 500 *ms*. A Aplicação 2 possui carga computacional menor, pois analisa menos dados, mas tem uma frequência de geração de tarefas muito superior, gerando uma tarefa a cada 100 *ms*, visto a criticidade do cenário de colisões.

7.1.3 Latência nas transmissões de dados

A tecnologia 5G é um dos tipos de conexão de dados utilizado na arquitetura do sistema. Esse tipo de conexão é utilizado pelos dispositivos IoT para comunicação com os servidores MEC. As taxas de transmissão de dados para *download* e *upload* e latência do 5G estão descritas na Tabela 7.2. As tecnologias 3G e 4G são apresentadas apenas para efeito de comparação.

Segundo (GUPTA; JHA, 2015), é esperado que as conexões 5G tenham latências

Tabela 7.2: Velocidades das conexões 3G DC-HSPA+, 4G LTE-A Cat16, 5G.

Tipo	Latência	*Download*	*Upload*
3G DC-HSPA+	∼100ms	8Mbps	3Mbps
4G LTE-A Cat16	∼50ms	200Mbps	100Mbps
5G	< 5ms	1Gbps	200Mbps
Fibra Óptica	∼5ms	1Gbps	1Gbps

Fonte: (GUPTA; JHA, 2015) (BROGI; FORTI; IBRAHIM, 2018).

de comunicação inferiores a 5 *ms*, podendo chegar a 1 *ms*, mas para chegar nesse patamar é necessário que todas as etapas da entrega de dados sejam otimizadas (5G-AMERICAS, 2018). Nas simulações é utilizada a latência de 5 *ms* nas conexões 5G.

A latência especifica quanto tempo leva para que os dados viajem pela rede de um computador a outro (COMER, 2015). Para dispositivos IoT, por exemplo, a latência contempla o tempo necessário para os sensores realizarem uma comunicação com o servidor MEC e esta comunicação ser conhecida no destino. À parte da latência também há o tempo de transferência de dados, dependente do volume a ser transferido, ambos compondo o tempo total necessário para uma transmissão de dados. Pode-se dizer que a latência é soma dos seguintes atrasos (COMER, 2015):

- Atraso de propagação do sinal no meio;
- Atraso de acesso ao meio;
- Atraso de encaminhamento de pacotes;
- Atraso no qual um pacote espera na memória de um *switch* ou roteador até ser selecionado e transmitido; e
- Atraso para que um servidor responda à requisição e envie a resposta.

O tempo de transmissão, por sua vez, é dependente do tamanho do arquivo e da taxa de transferência (largura de banda em *bits* por segundo). Assim, a latência indicada para comunicações 5G corresponde aos atrasos envolvidos na transmissão de dados, sem considerar o tempo de transmissão do volume de dados.

A latência para as transmissões via fibra óptica é de cerca de 5 $\mu s/km$ para fibra óptica de modo de operação único (JAY, 2012). É sabido que a latência efetiva do sistema é afetada pela distância entre os *Data Centers* e os servidores MEC, devido ao tempo de propagação. Considerando a região metropolitana de Porto Alegre, Rio Grande do Sul, os *Data Centers* para serviços de *Cloud* da *Google Cloud*, *Amazon AWS* e *Microsoft Azure* mais próximos, estão localizados, respectivamente, em Quilicura, Chile[1] e São Paulo,

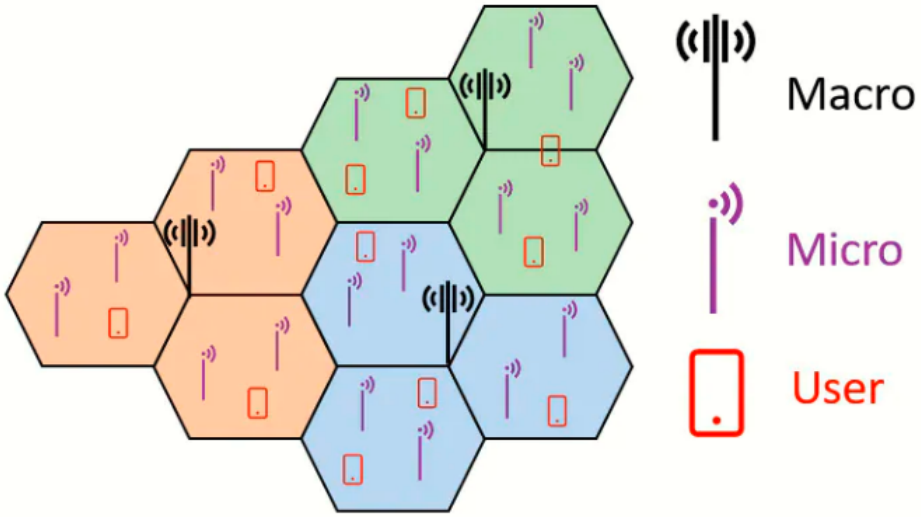
[1]<https://www.google.com/about/datacenters/locations/>

Brasil[2][3]. Para a região de São Paulo a distância é de cerca de 1000 km, ou seja, via fibra óptica a latência é de aproximadamente 5 *ms*. Esta é a latência utilizada para conexões feitas com fibra óptica nas simulações.

7.1.4 Consumo energético nas transmissões de dados

Segundo (KITANOV; JANEVSKI, 2017), 5G possui eficiência energética 100 vezes maior do que as conexões 4G-LTE. A eficiência energética é calculada pela divisão entre a energia gasta para a transmissão de determinada quantidade de dados, ou seja, 5G consegue transferir mais dados a um menor custo energético. Porém, a infraestrutura necessária para manter a rede 5G requer a utilização de mais torres de transmissão e antenas próximas aos usuários, visto que o 5G é uma tecnologia que opera em pequenos *clusters* de antenas, ao contrário dos serviços 3G e 4G. A Figura 7.3 apresenta uma visão geral de implementação da infraestrutura 5G. Estudos indicam que o custo energético total dos equipamentos 5G operando em 3,5GHz pode chegar de 300% a 350% dos custos incorridos nas estações base do 4G-LTE (CARLINI, 2019) (CLARK, 2019).

Figura 7.3: Exemplo de um cenário com estações base micro e macro para redes 5G.



Fonte: (FRENGER; TANO, 2019).

Logo, para redes 5G assume-se que a eficiência energética é 100 vezes maior do que conexões 4G, porém, por ter custo energético estimado de cerca de 3 a 3,5 vezes maior para a infraestrutura, a energia consumida quando o sistema fica em *idle* é assumida que

[2]<https://aws.amazon.com/pt/about-aws/global-infrastructure/>
[3]<https://azure.microsoft.com/pt-br/global-infrastructure/locations/>

seja 3 vezes maior do que em sistemas 4G. A Tabela 7.3 apresenta o custo energético para as transferências de dados, indicado pela letra α, e o custo energético base quando a taxa de transferência é zero, indicado pela letra β.

Tabela 7.3: Consumo energético dos tipos de transmissões de dados.

Tecnologia	α [*mW/Mbps*]	β [*mW*]
3G	122,12	817,88
4G	51,97	1288,04
5G	∼0,52	3864,12

Fonte: (KITANOV; JANEVSKI, 2017).

A potência de transmissão de dados de uma comunicação 5G (P_{5G}), para uma taxa de transferência de arquivos T, é dada pela Equação 7.1. Nela é feita um ajuste para transformar a potência de *mili-watts* para *watts*, multiplicando a equação por 10^{-3}. A taxa de transferência T é dada em *Mbps*. A energia consumida (E_{5G}) leva em consideração o tempo decorrido do envio do primeiro ao último pacote de dados na origem.

$$P_{5G} = (\alpha * T + \beta) * 10^{-3} \; [Watt] \tag{7.1}$$

$$E_{5G} = P_{5G} * T_{transferencia} \; [Watt * segundo] \tag{7.2}$$

Já quanto a fibra óptica, segundo (SKUBIC et al., 2012), para redes que utilizam esta tecnologia, as *Unidades de Rede Óptica* (*Optical Network Unit*, ou ONU) são os equipamentos responsáveis pela maior parte do consumo durante a transmissão de dados, e, portanto, uma boa referência para estimar o consumo energético da rede. Esses equipamentos são instalados junto aos nodos da rede e transmitem sinais ópticos pela fibra ou os convertem para sinais elétricos, sua função é transmitir e receber dados. A potência de operação desse tipo de equipamento é, na média, de 3,65W. Outros equipamento como repetidores e agregadores também se fazem presentes, porém serão desconsiderados para simplificação do modelo. Assim, o consumo energético para transmissões com fibra óptica é dado por:

$$E_{fibra} = 3,65 * T_{transferencia} \; [Watt * segundo] \tag{7.3}$$

7.1.5 Frequências de operação e consumo energético nos dispositivos IoT

O hardware escolhido para representar os dispositivos IoT é o *kit* de desenvolvimento *Arduino Mega 2560*. O *Arduino* é um *kit* de desenvolvimento muito popular na indústria para pequenas implementações e entre *hobbystas* para automação residencial. Nele podem ser inseridos diversos tipos de sensores para as mais variadas aplicações, como, por exemplo, sensores de umidade, de luz, de ruído, proximidade, temperatura, pressão, dentre outros. É um *hardware* barato, de fácil acesso e configuração, pequeno, com conectividade e baixo consumo energético, portanto, uma boa opção para representar dispositivos IoT.

O *Arduino Mega 2560* possui o microcontrolador *ATmega2560*. Ele possui uma CPU com um núcleo de 16 *MHz*, com 5 frequências configuráveis, de 16 *MHz*, 8 *MHz*, 4 *MHz*, 2 *MHz* e 1 *MHz*. A alimentação elétrica é reduzida pelo microcontrolador, à medida que a frequência de operação diminui, de modo a propiciar menor consumo energético (ATMEL, 2014). Além disso, a capacitância do microcontrolador é aproximadamente 2,2 *nF* (*nanofarads*), ou $2,2 * 10^{-9}$ *F*. A Tabela 7.4 apresenta dados de potência e tempo de execução, para uma tarefa com carga computacional de 200.000 ciclos de CPU, para todos os degraus de frequência de operação do microcontrolador. As tensões correspondentes às frequências de operação também são apresentadas.

Tabela 7.4: Potência dinâmica e tempo de execução para uma carga de trabalho no *Arduino Mega 2560*.

C (*nF*)	Tensão (*V*)	f (*MHz*)	P (*mW*)	Ciclos CPU	t (*ms*)	Energia (*mW* * *ms*)
2,2	5,0	16	880,00	200.000	12,5	11.000,00
2,2	4,0	8	281,60	200.000	25	7.040,00
2,2	2,7	4	64,152	200.000	50	3.207,60
2,2	2,3	2	23,276	200.000	100	2.327,60
2,2	1,8	1	7,128	200.000	200	1.425,60

A folha de dados do microcontrolador *ATmega2560* ainda prevê um estado de baixíssimo consumo, com frequência de operação em 1 *MHz*, 1,8 *V* de alimentação e corrente de operação de 500 μA, o equivalente a uma potência de 900 μW. Esta é a potência utilizada para quando a CPU estiver em *idle*. Portanto:

$$P_{iot-dinamica} = 2,2 * 10^{-9} * 5^2 * f \ [Watt] \tag{7.4}$$

$$P_{iot-idle} = 900 * 10^{-6} \ [Watt] \tag{7.5}$$

A energia dinâmica consumida para a execução completa de uma tarefa, é dada por:

$$E_{iot-dinamica} = P_{iot-dinamica} * T_{execucao} \, [Watt * segundo] \tag{7.6}$$

7.1.6 Frequências de operação e consumo energético nos servidores MEC

Para a camada MEC foram escolhidos servidores compostos por 5 placas de *Raspberry Pi 4 Model B*. Esse é o modelo mais atual de placas *Raspberry Pi*, até o momento. Cada placa é construída com um *chipset Broadcom BCM2711* equipada com um processador *Quad-core Cortex-A72 (ARM v8) 64-bit SoC* (*System-on-a-Chip*) de 1,5 *GHz* por núcleo (FOUNDATION, 2019), totalizando 20 núcleos por servidor.

O *firmware* do *Raspberry Pi 4 Model B* implementa a técnica de DVFS. As frequências configuráveis são 1500 *MHz*, 1000 *MHz*, 750 *MHz* e 600 *MHz*, segundo o FAQ da página oficial do *Raspberry Pi*[4]. Quando o processador não está executando na velocidade máxima, a tensão para a área específica do *chip* é reduzida de acordo com a redução do *clock*. Como resultado, apenas a tensão necessária é fornecida ao núcleo para que o processamento ocorra corretamente. Isso resulta em menor potência consumida pelo SoC. Ou seja, há uma correlação entre frequências mais altas e a necessidade de tensões também mais elevadas. Essa relação frequência *vs.* tensão é descrita por (PEREIRA; ILIC; SOUSA, 2017). Segundo o autor, para 1200 *MHz*, 1000 *MHz* e 600 *MHz* as tensões de operação, são, respectivamente, 1 *V*, 0,9 *V* e 0,8 *V*. Com base nessas frequências e tensões, é possível inferir as tensões de operação aproximadas para as frequências de 1500 *MHz* e 750 *MHz*, através de uma linha de tendência polinomial de ordem 2, resultando, respectivamente, nas tensões de 1,2 *V* e 0,825 *V*.

Segundo (FOUNDATION, 2019), o consumo de corrente típica para manter a placa em operação é de 600 *mA*, o que equivale a 2,7 *W* para uma alimentação de 5 *V*. Este é o consumo da placa no estado *idle*, com a frequência de operação reduzida para 600 *MHz*, equivalente ao consumo dos 4 núcleos em *idle*. Assim, o consumo em *idle* por núcleo é de 0,675W. A capacitância, por sua vez, é de aproximadamente 1,8 *nF*, ou $1,8 * 10^{-9}$ *F*. Portanto, para uma carga com 200.000 ciclos de CPU temos os seguintes valores:

As equações que definem a potência dinâmica e em *idle* dos núcleos do servidor

[4]<https://www.raspberrypi.org/documentation/faqs/>

Tabela 7.5: Potência dinâmica e tempo de execução para um núcleo com uma carga de trabalho no *Raspberry Pi 4 Model B*.

C (*n*F)	Tensão (*V*)	f (*MHz*)	P (*mW*)	Ciclos CPU	t (*ms*)	Energia (*mW * ms*)
1,8	1,2	1500	3.888,00	200.000	0,133	518,40
1,8	1	1000	1.800,00	200.000	0,200	360,00
1,8	0,825	750	918,84	200.000	0,267	245,03
1,8	0,8	600	691,20	200.000	0,333	230,40

MEC são apresentadas a seguir.

$$P_{mec-dinamica} = 1,8 * 10^{-9} * 5^2 * f \ [Watt] \tag{7.7}$$

$$P_{mec-idle} = 675 * 10^{-3} \ [Watt] \tag{7.8}$$

A energia dinâmica consumida para execução completa de uma tarefa, é dada por:

$$E_{mec-dinamica} = P_{mec-dinamica} * T_{execucao} \ [Watt * segundo] \tag{7.9}$$

Comparando as Tabelas 7.4 e 7.5 podemos observar que o tempo de finalização das tarefas reduz em uma ordem de cerca de 100 vezes no servidor local, porém a potência do sistema aumenta. Mesmo considerando o aumento de potência, há que se considerar também o tempo de processamento. Por exemplo, para um dispositivo IoT com frequência de operação de 16 *MHz* e carga de trabalho de 200.000 CT temos energia consumida de 11.000 *mW*ms* (*mili-Watt * mili-segundo*), enquanto para o servidor MEC operando com frequência de 1500 *MHz* e com a mesma carga de trabalho temos energia consumida de 518,4 *mW*ms*, ou seja, uma diferença de pouco mais de 20 vezes. Importante ressaltar que a execução no servidor MEC também incorre em custos adicionais com transmissões de dados, agregando em tempo e consumo energético, o que também deve ser considerado. Cabe ao administrador do sistema definir adequadamente os graus de importância das variáveis de energia consumida e tempo de execução, priorizando um ou outro.

7.1.7 Frequência de operação e consumo energético na *Cloud*

A *Google Cloud* possui uma linha de processadores escalonáveis *Intel Xeon Cascade Lake* de 2,8 *GHz* por núcleo, podendo chegar a 3,9 *GHz* na opção *Turbo Boost*[5].

[5]<https://www.intel.com.br/content/www/br/pt/architecture-and-technology/turbo-boost/turbo-boost-technology.html>

Em testes de desempenho e consumo energético[6] o *Intel Xeon 8280* de 28 núcleos chega a consumir 680W para uma condição de máxima carga e máximas frequências de operação. Para operação com frequência padrão, o consumo é de 388 *W*. Assim, para um núcleo da *Cloud* em frequência de operação padrão a 2,8 *GHz* o consumo por núcleo é de 13,85 *W* e a 3,9 *GHZ*, com a opção *Turbo Boost* ativada, o consumo é de aproximadamente 24,28 *W*. É um consumo energético elevado, porém não agrega aos custos da rede local, havendo, entretanto, a necessidade de considerar os custos envolvidos com as transmissões de dados. Além disso, a *Cloud* oferece diversos outros serviços e recursos suplementares, como grandes quantidades de memória RAM e armazenamento de mídia em SSD, contribuindo com ganhos de performance. Para o modelo do sistema, são utilizadas as frequências de 2,8 e 3,9 *GHz* para a *Cloud*.

7.1.8 Bateria dos dispositivos IoT

Para a bateria dos dispositivos IoT, assume-se que a bateria possui tensão de 5 *V* e forneça corrente de até 2.000 *mAh*, o equivalente a 10 *Wh* ou 36.000 *Ws*. Assim, conforme Tabela 7.4, uma aplicação executando na frequência de 16 *MHz*, com potência dinâmica de 880 *mW*, poderia seguir ininterruptamente por pouco mais de 11 horas. Já uma aplicação executando a 1 *MHz*, com potência dinâmica de 7,128 *mW*, teria uma autonomia de mais de 58 dias.

$$Autonomia = 36.000Ws/0,007128W = 5.050.505,05s = 1.402,92h = 58,45d$$

7.2 Simulador *Ad-hoc*

Inicialmente foi aventada a possibilidade de utilizar o simulador *iFogSim* para realizar a bateria de experimentos, porém alguns problemas foram encontrados. A instalação do simulador e a preparação do ambiente de programação estão muito bem descritos em um tutorial publicado recentemente (MAHMUD; BUYYA, 2018). Porém este autor encontrou algumas dificuldades, não superadas para a avaliação que se propunha. Dentre os principais problemas encontrados no *iFogSim* destacam-se:

1. Geração de fluxo de tarefas: Para cada tarefa é requerido que haja a geração de um fluxo de tarefas. Em aplicações veiculares, por exemplo, cada atuador da pe-

[6]<https://www.tomshardware.com/>

riferia da rede ou dispositivo IoT cria tarefas periodicamente, seja para divulgação de informações ou para cálculos de rotas e análise das informações sensoriais do veículo. Não foi encontrado no *iFogSim* um módulo que pudesse simular esse fluxo de criação de tarefas para cada dispositivo IoT.

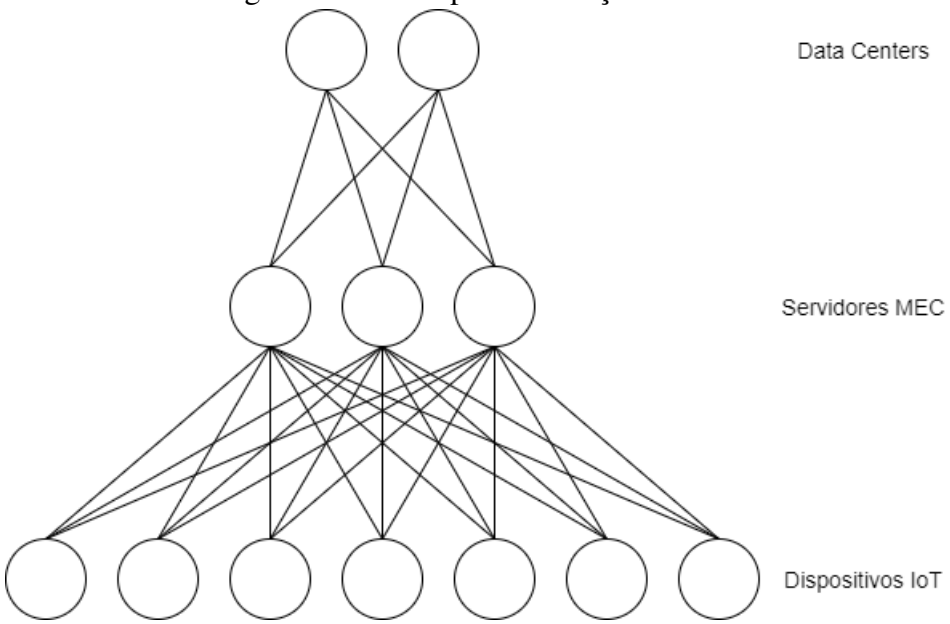
2. Algoritmo de escalonamento: Dada uma topologia de dispositivos IoT, servidores MEC e data centers, o autor não encontrou uma forma de permitir que o escalonador pudesse pegar cada tarefa, alocar para determinado dispositivo IoT ou servidor MEC e calcular os custos de latência e energia no processo.
3. Tarefas críticas: Não foi encontrada uma forma de realizar gestão sobre as tarefas que possuem *deadline* de conclusão, ou seja, monitorar o tempo envolvido desde a criação da tarefa, sua transmissão, processamento e retorno de resultados, para estabelecer se os resultados chegaram à origem antes da finalização do *deadline*.
4. Nível de bateria dos dispositivos IoT: Não foi encontrado nenhum módulo que pudesse emular o comportamento da bateria dos dispositivos à medida que a computação fosse realizada. Este é um ponto incluso no algoritmo de escalonamento e é importante para saber quais dispositivos seguem disponíveis na rede.

É possível que tais dificuldades fossem superadas com estudos suplementares sobre a ferramenta, mas o tempo investido não compensaria, na visão deste autor. Assim, optou-se pela implementação de um simulador *ad-hoc*, focado em uma metodologia que considera a taxa de geração de tarefas e a devida alocação das mesmas pelo escalonador após análise dos custos dos elementos da rede (nodos de processamento e transmissões de dados). O simulador funciona especificamente para a arquitetura utilizada neste trabalho, explorando as relações existentes entre os elementos da rede, focado no tempo de execução das tarefas e no consumo de energia para processamento, nível de bateria e transmissões de dados.

A rede foi implementada como um grafo em três camadas, tal qual descrito na Figura 5.2, que apresenta a arquitetura do sistema. Todos os nodos da camada de dispositivos IoT comunicam-se com todos os servidores MEC, assim como todos os servidores MEC comunicam-se com os *Data Centers*, conforme Figura 7.4.

Cada aresta representa um canal de comunicação com seu peso correspondente. Cada servidor MEC possui um único canal de comunicação com os dispositivos móveis, ou seja, se os dispositivos *A* e *B* quiserem se comunicar com o servidores MEC *C*, então *A* e *B* irão utilizar o mesmo canal de comunicação para transmissão de dados. Essa comunicação sobreposta no mesmo canal gera ruído e modifica a taxa de transmissão, conforme

Figura 7.4: Grafo para simulação da rede.



a *Equação de Shannon*, expressa na Equação 5.8. Para fins de simplificação, toda vez que dois dispositivos IoT tentarem realizar alguma transmissão de dados simultaneamente será aplicado um fator de atenuação de 10% na taxa de transferência.

7.3 Detalhamento da Implementação

O simulador *ad-hoc* foi desenvolvido na *IDE Eclipse versão 2019-12* (*4.14.0*) em linguagem *Java versão 1.8.0_241*. Ele é composto de uma série de classes que simulam o comportamento de cada elemento da rede incluído no modelo do sistema. Nas Figuras 7.5 e 7.6 são apresentadas as classes utilizadas pelo simulador.

A Figura 7.5 demonstra o core do sistema, com o escalonador, com a visão macro de todos os equipamentos da rede, com seus respectivos custos energéticos e tempos de execução. Ele também tem acesso aos custos energéticos e tempos necessários para realizar as transmissões de dados entre diferentes camadas da arquitetura. As classes dessa etapa são:

- *Scheduler*: Monitora os recursos da rede, implementa uma versão heurística do problema de minimização descrito no modelo do sistema e realiza a alocação das tarefas, conforme o tipo de tarefa e os custos dos nodos. Esse comportamento é descrito no Capítulo 6.
- *IoTDevice*: Representa um dispositivo IoT. Possui os custos de tempo e energé-

Figura 7.5: Diagrama UML de classes do simulador *ad-hoc*. Classes *IoTDevice*, *MECServer*, *Task*, *Scheduler*, *RAN-5G*, *FiberOptics* e *CloudDataCenter*.

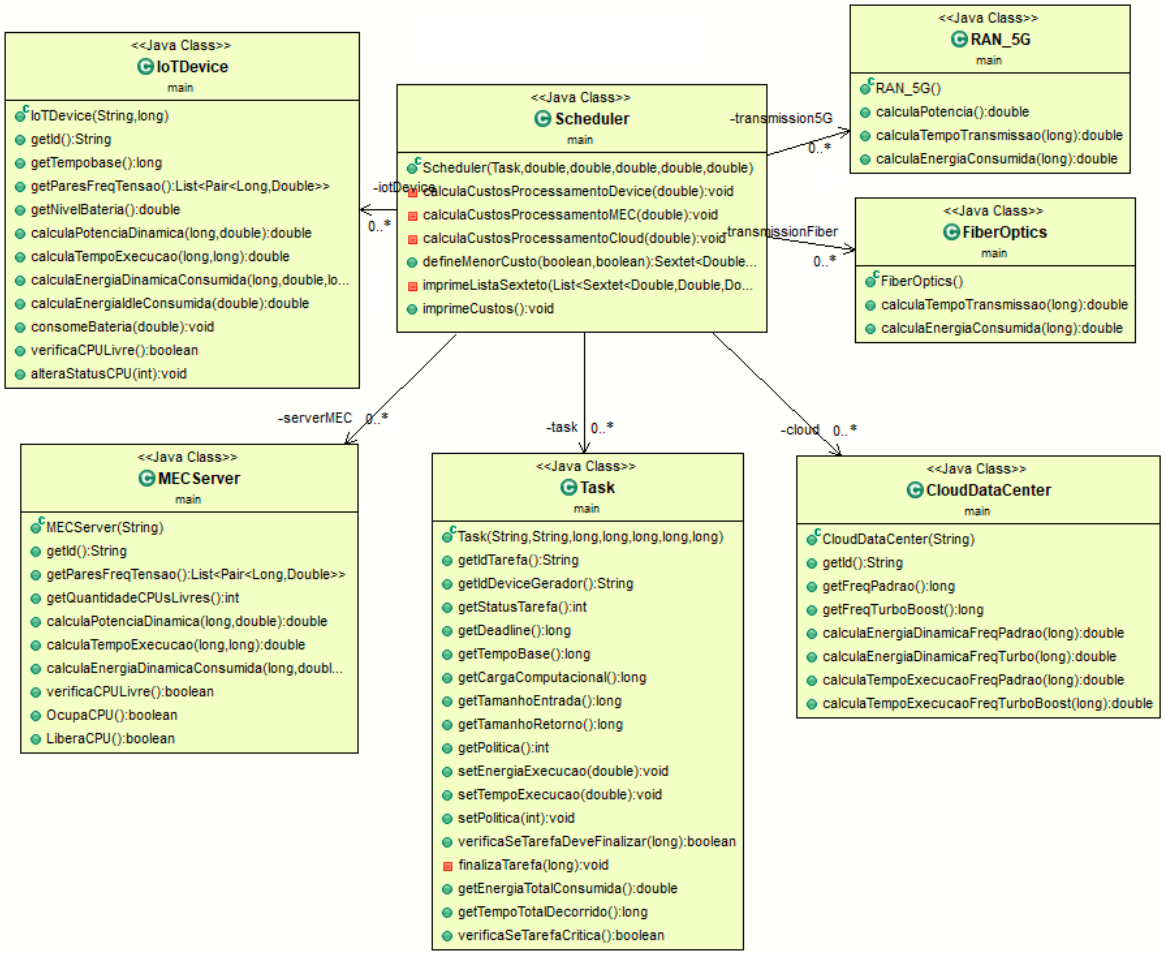
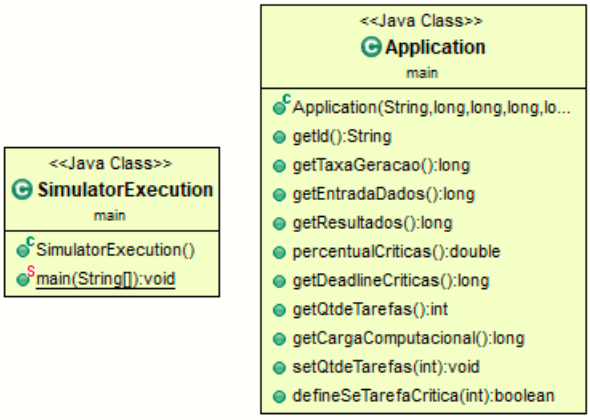


Figura 7.6: Diagrama UML de classes do simulador *ad-hoc*. Classes *SimulatorExecution* e *Application*.



tico de um *Arduino Mega 2560* com uma CPU de um núcleo descritos na Subseção 7.1.5.

- *MECServer*: Representa um servidor MEC. Possui os custos de tempo e energético de um servidor composto por 5 placas *Raspberry Pi 4 Model B*, totalizando 20

núcleos de processamento, descritos na Subseção 7.1.6.

- *CloudDataCenter*: Representa um data center. Possui os custos de tempo e energético descritos na Subseção 7.1.7. Não prevê limitante para a alocação de recursos.
- *RAN_5G*: Representa uma transmissão de dados 5G. Possui os custos de tempo e consumo energético descritos nas Subseções 7.1.3 e 7.1.4.
- *FiberOptics*: Representa uma transmissão de dados com fibra óptica. Possui os custos de tempo e consumo energético descritos nas Subseções 7.1.3 e 7.1.4.
- *Task*: Representa uma tarefa. A cada tarefa estão associadas informações sobre a aplicação, como perfil de carga, tipo de tarefa, se crítica ou não, *deadline*, se aplicável, política de alocação utilizada, dentro outros.

O escalonador coleta as informações sobre os elementos da rede e define a política de alocação mais adequada para as tarefas críticas e não críticas. A Figura 7.6 apresenta duas classes acessórias, do simulador da rede e da aplicação:

- *SimulatorExecution*: Responsável por controlar o andamento da simulação, com as quantidades de equipamentos que estarão presentes na rede e a quantidade de tarefas. Também é nela que são geradas as estatísticas e dados sobre cada rodada de execução.
- *Application*: Representa a aplicação que será executada na simulação, estabelece os parâmetros de taxa de geração de tarefas, percentual de tarefas críticas, carga computacional, tamanho da entrada e dos resultados e *deadline* de finalização de tarefas.

7.4 Descrição dos Experimentos

As rodadas de simulação são realizadas com quantidades variadas de tarefas, dispositivos IoT e servidores, a depender de cada caso. Sempre há a presença do *Data Center* da *Cloud* à disposição.

Os valores de potência para núcleos e frequências de operação são os mesmos descritos nas seções anteriores deste capítulo. O mesmo se aplica para a potência e tempo das transmissões de dados. Já os parâmetros utilizados pela aplicação e tarefas, são os mesmos descritos pela Tabela 7.1.

Diversos cenários de teste foram elaborados, e envolvem a variação de parâmetros como carga de processamento das tarefas, nível máximo de bateria dos dispositivos IoT,

tamanho da entrada de dados da aplicação, taxa de geração de tarefas, *deadline* das tarefas críticas e alteração dos coeficientes de custo de energia e tempo para compor o custo final.

Nas subseções seguintes os cenários de teste são descritos com mais detalhes, esclarecendo o objetivo de cada teste.

7.4.1 Cenários de teste com variação de carga de processamento das tarefas

As aplicações escolhidas na Tabela 7.1 possuem carga de processamento fixa. Ela impacta no cálculo de energia consumida e tempo de processamento nos núcleos da rede, e, portanto, variar essa carga de processamento pode oferecer resultados importantes sobre o comportamento da alocação de tarefas para as diferentes políticas existentes e também sobre o nível de tarefas canceladas, caso não seja possível processar as tarefas em tempo hábil, ou seja, antes do *deadline*.

7.4.2 Cenários de teste com variação do coeficiente de custo (relação tempo vs. energia)

No início da simulação são estabelecidos os coeficientes de energia consumida e tempo total decorrido até a finalização da tarefa, utilizados para dimensionar o custo das diferentes opções de alocação que o sistema oferece. Somados os dois coeficientes possuem valor igual a 1. O objetivo desse teste é variar essa relação de coeficientes e observar como o custo de cada política de alocação varia, e, consequentemente, influi nas decisões de alocação do escalonador de tarefas do sistema.

7.4.3 Cenários de teste com variação do tamanho da entrada de dados da aplicação

Utilizando a Aplicação 1 para esta simulação, o tamanho da entrada de dados é variado, mantendo os demais parâmetros constantes com objetivo de observar como os custos do sistema se alteram e como isso impacta a alocação de tarefas pelo escalonador do sistema. É esperado que haja variação significativa no consumo energético e no tempo decorrido nas transmissões de dados, o que é levado em consideração ao escolher a política de alocação mais adequada.

7.4.4 Cenários de teste com variação da taxa de geração de tarefas

A taxa de geração de tarefas tem um impacto grande na alocação de tarefas, pois com uma taxa de geração muito rápida, os recursos de processamento disponíveis nos dispositivos IoT e no servidor MEC são rapidamente ocupados, restando apenas alocação de tarefas nos núcleos da *Cloud*, visto que a *Cloud* possui recursos de processamento ilimitados na simulação. Logo, o objetivo desse experimento é observar o comportamento das alocações e a saturação dos recursos de *hardware* na rede local, conforme a mudança na taxa de geração de tarefas.

7.4.5 Cenários de teste com variação do tamanho do *deadline*

O *deadline* é utilizado pelas aplicações em tarefas críticas para definir o tempo máximo de resposta exigido pela aplicação até que os resultados sejam entregues. Aplicações veiculares e na área de *healthcare* possuem casos com restrições de tempo, nos quais o tempo de resposta é uma variável importante, seja, por exemplo, para evitar uma colisão entre dois veículos ou emitir um alerta para um familiar quando um paciente sofre uma queda. O objetivo desse teste é observar o impacto do *deadline* na quantidade de tarefas canceladas pela aplicação. Se o *deadline* for muito pequeno, é provável que não seja possível entregar os resultados em tempo hábil, e, portanto, a tarefa é cancelada. Um tempo de *deadline* adequado evita esse cenário, mas também é necessário que o *deadline* seja compatível com o QoS da aplicação.

7.4.6 Cenários de teste com variação do nível de bateria dos dispositivos IoT

O nível de bateria dos dispositivos IoT é um parâmetro importante, continuamente monitorado pelo escalonador de tarefas, que prioriza que o dispositivo siga ligado, gerando as tarefas da aplicação. Quando a bateria do dispositivo atinge níveis críticos, o escalonador do sistema impede que a alocação de novas tarefas seja feita nesse dispositivo, priorizando a continuidade de sua operação. O objetivo desse experimento é observar a progressão do nível de bateria dos dispositivos IoT confrontado com as decisões de alocação realizadas pelo escalonador de tarefas.

7.4.7 Cenários de teste com e sem DVFS

O modelo do sistema utiliza a equação de consumo de potência dinâmica para calcular a energia consumida, com a tensão aplicada no núcleo e a frequência de operação. O ajuste dinâmico dessas duas variáveis constituem a estratégia de DVFS e tem por objetivo oferecer menor potência consumida quando possível, reduzindo o consumo de energia. Logo, o objetivo desse experimento é observar qual o consumo energético e o tempo decorrido em dois cenários, um com DVFS e outro sem DVFS.

7.5 Análise dos Resultados

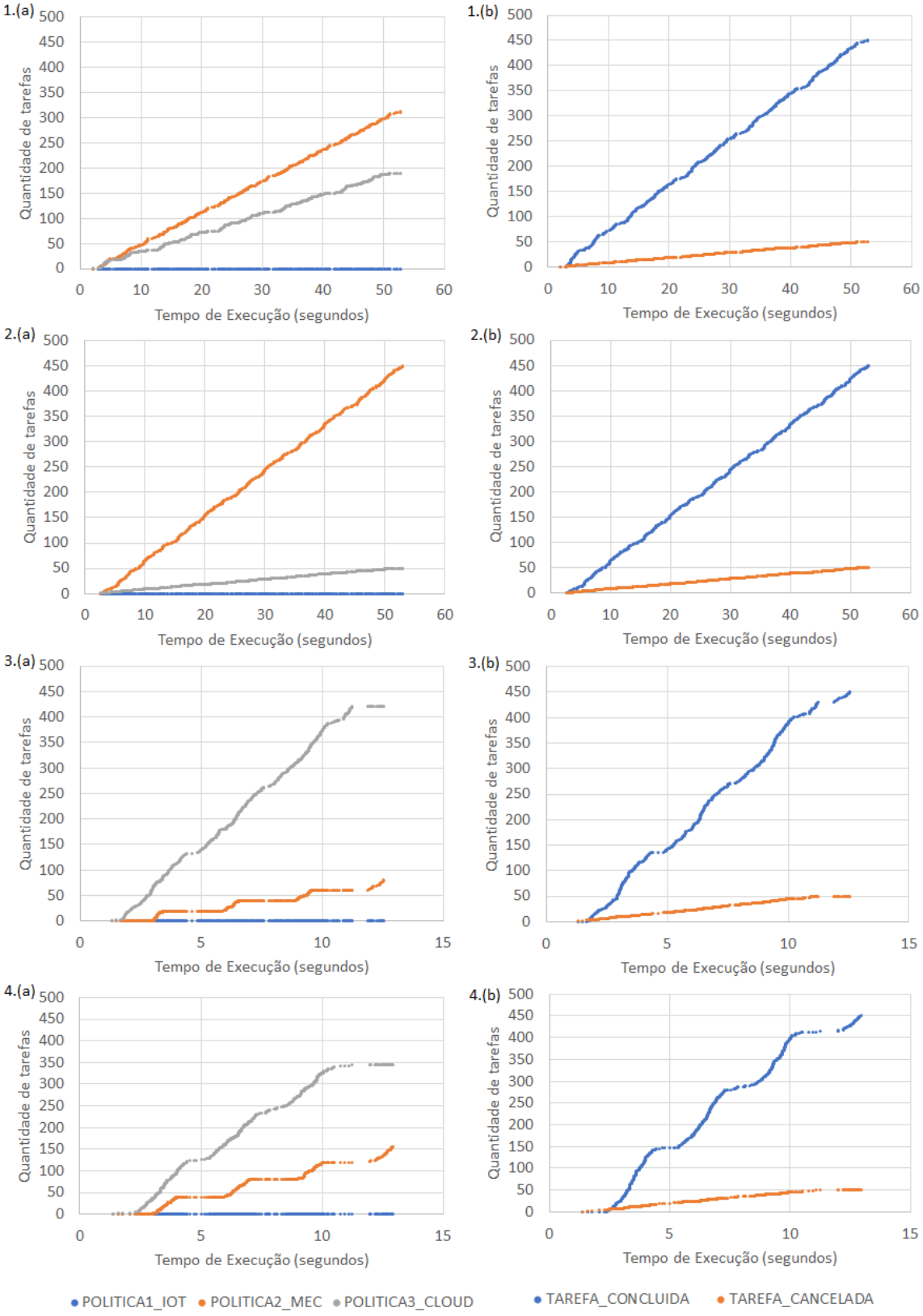
A execução dos experimentos foi realizada em uma máquina com *Windows 10 Home 64-bit*, processador *Intel Quad-Core i7-7700HQ 2,80 GHz* com 16 *GB* de memória RAM. Os experimentos são analisados nas subseções a seguir.

7.5.1 Variação de carga de processamento das tarefas

Na Figura 7.7 é apresentado o comportamento da alocação de tarefas ao longo do tempo, bem como o comportamento da finalização das tarefas, para quatro casos distintos. Todos possuem carga de processamento de $2.000 * 10^6$ ciclos de CPU e as características de taxa de geração de tarefas e *deadline* de finalização são as mesmas da Aplicação 1. Os quatro casos são: (1) 500 tarefas, 100 dispositivos IoT e 1 servidor MEC, (2) 500 tarefas, 100 dispositivos IoT e 2 servidores MEC, (3) 500 tarefas, 500 dispositivos IoT e 1 servidor MEC e, por fim, (4) 500 tarefas, 500 dispositivos IoT e 2 servidores MEC.

Para esses experimentos os coeficientes para energia e tempo utilizados foram de, respectivamente, $4/5$ e $1/5$, ou seja, foi dado um peso alto para a energia consumida, de modo que ela fosse minimizada. A Figura 7.7-1.(a) não realiza nenhuma alocação para a política 1, ou seja, para os dispositivos IoT, visto que a carga de processamento utilizada eleva em muito os custos energéticos e o tempo de execução local, e ao invés dela, as outras duas políticas são utilizadas. A alocação é feita preferencialmente no servidor MEC, porém quando este fica com todas os núcleos ocupados, a alocação é realizada na *Cloud*. O mesmo ocorre como a Figura 7.7-2.(a), porém são realizadas menos alocações na *Cloud*, já que há dois servidores MEC à disposição, cada um com 20 núcleos. Quando

Figura 7.7: Variação de carga de processamento das tarefas utilizando a Aplicação 1.



estes dois servidores esgotam seus recursos, a alocação na *Cloud* é realizada.

Na Figura 7.7-3.(a) a alocação muda um pouco de perfil. Tanto na Figura 7.7-3.(a)

quanto na Figura 7.7-4.(a) a maioria das alocações ocorre na *Cloud*. Esse fenômeno é explicado, pois nesses dois cenários há 500 tarefas e 500 dispositivos IoT, e cada um dos dispositivos gera uma tarefa. Como todas as tarefas são geradas muito próximas umas das outras, mesmo havendo preferência em alocar as tarefas no servidor local, ele fica saturado rapidamente. Isso pode ser observado pelo tempo de simulação no gráfico, pois enquanto 7.7-1.(a) e 7.7-2.(a) executam por mais de 50 segundos para gerar e finalizar 500 tarefas, em 7.7-3.(a) e 7.7-4.(a) as 500 tarefas finalizam em pouco menos de 13 segundos. Assim, a maioria das tarefas é destinada à *Cloud*. A influência do servidor MEC é percebida na Figura 7.7-4.(a), pois há mais alocações para a política 2, o que só é possível por haver mais servidores MEC na rede.

Tabela 7.6: Somatório de custos de processamento e transmissões de dados para consumo energético e tempo decorrido.

Descrição	Caso 1	Caso 2	Caso 3	Caso 4	Caso 5	Caso 6
E_{CORE} (*W*s*)	2.752,26	1.725,18	4.478,94	3.913,30	5.074,35	5.074,35
E_{Trans} (*W*s*)	835,60	1.725,18	1.081,44	1.000,90	1.166,21	1.166,21
E_{Total} (*W*s*)	3.587,86	3.450,35	5.560,38	4.914,21	6.240,56	6.240,56
T_{CORE} (*s*)	956,21	1.225,64	503,26	651,64	347,07	347,07
T_{Trans} (*s*)	199,75	159,69	267,09	245,03	290,31	316,44
T_{Total} (*s*)	1.155,96	1.385,33	770,35	896,67	637,38	663,51

Na Tabela 7.6 podemos perceber o consumo dinâmico de energia e o tempo dinâmico consumido na rede como um todo. Os valores apresentados são o somatório dos consumos e tempos individuais de cada tarefa. A ideia dessa tabela é demonstrar como o consumo energético da rede e o tempo médio das tarefas até a finalização pode alterar conforme as políticas de alocação escolhidas.

Para os Casos 1 e 2 (Figura 7.7-1.(a) e 2.(a)), as 500 tarefas tem alocação equilibrada entre servidor MEC e *Cloud*, gerando consumos energéticos semelhantes, com destaque para o Caso 2 que consome um pouco menos, mesmo utilizando dois servidores MEC, mas em contrapartida, faz menos alocações à *Cloud*. Nos Casos 3 e 4 (Figura 7.7-3.(a) e 4.(a)) o consumo energético aumenta, mas em compensação os tempos totais são reduzidos.

Os Casos 5 e 6 são cenários sem servidor MEC, mas mantendo a *Cloud*, com 500 tarefas em ambos os casos e 100 dispositivos IoT para o Caso 5 e 500 dispositivos IoT para o Caso 6. Em ambos a alocação ocorreu apenas na *Cloud*, visto que o custo final para processar no dispositivo IoT, para a carga de $2.000 * 10^6$ ciclos de CPU, foi muito elevado. Nos dois casos o consumo energético total aumentou em relação aos casos anteriores, porém houve uma redução no tempo de processamento das tarefas. Os núcleos

da *Cloud* processam mais rapidamente, mas possuem maiores custos energéticos com processamento e transmissões de dados. No Caso 6 o tempo total é um pouco superior devido ao maior volume simultâneo de alocações para a *Cloud* do que no Caso 5. A taxa de transferência é reduzida em cerca de 10%, devido às interferências das diferentes transmissões de dados na rede sem fio uma sobre as outras, impactando o tempo total.

Os Casos 1 e 2, comparado ao Caso 5, possuem as mesmas quantidades de tarefas e dispositivos IoT, com a diferença de que nos dois primeiros casos há existência de servidores MEC. Para o Caso 1 a redução na energia consumida pelo sistema foi de 42,51%, enquanto para o Caso 2 a redução foi de 44,71%. O tempo de processamento foi maior, porém se o processamento ocorrer dentro de uma janela segura, antes de atingir o *deadline*, não haverá problemas.

Tabela 7.7: Custos calculados pelo escalonador do sistema para a Aplicação 1.

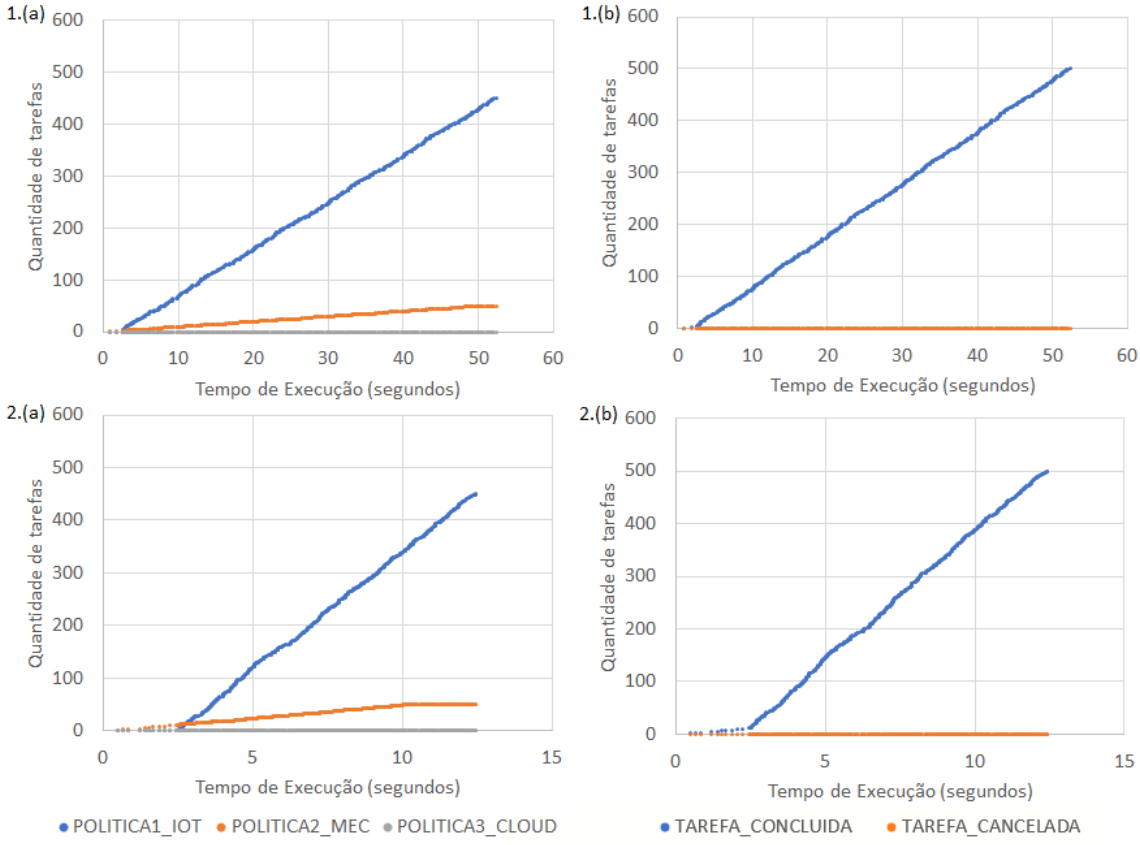
Custo	E_{CORE} (*W*s*)	E_{tran} (*W*s*)	T_{CORE} (*s*)	T_{tran} (*s*)	f (*MHz*)	T (*V*)	Pol.
1,19	2,45	1,27	2,67	0,29	750	0,825	2
1,20	2,30	1,27	3,33	0,29	600	0,8	2
1,45	3,60	1,27	2,00	0,29	1.000	1	2
1,83	5,18	1,27	1,33	0,29	1.500	1,2	2
3,35	9,89	2,33	0,71	0,58	2.800	-	3
4,02	12,45	2,33	0,51	0,58	3.900	-	3
35,44	70,40	-	250,00	-	8	4	1
37,67	110,00	-	125,00	-	16	5	1
41,89	32,08	-	500,00	-	4	2,7	1
72,87	23,28	-	1.000,00	-	2	2,3	1
137,13	14,26	-	2.000,00	-	1	1,8	1

Para entender melhor como o escalonador do sistema percebe o custo de alocação, a Tabela 7.7 é apresentada. Os resultados estão ordenados por custo, do menor ao maior; lembrando que estes custos são obtidos após aplicar os coeficientes de energia e tempo sobre a energia consumida e o tempo decorrido. Para a carga de processamento de $2.000 * 10^6$ ciclos de CPU e coeficientes de energia e tempo em $4/5$ e $1/5$, respectivamente, o consumo energético e tempo de processamento nos dispositivos IoT são extremamente elevados, por isso ficam no final da lista. Como primeira opção de alocação está o servidor MEC, operando a 750 *MHz* com 0,825 *V* de tensão no núcleo (DVFS ativado). Mesmo considerando o consumo energético e tempo de transmissão de dados extras, a alocação no servidor MEC com essas características ainda é mais vantajosa do que a execução no dispositivo IoT. Estando o servidor MEC com todos os núcleos ocupados, a política de alocação que fornece o menor custo na sequência é a alocação na *Cloud*, com núcleo operando a 2,8 *GHz*.

Ainda sobre a Figura 7.7, das 500 tarefas em 1.(b), 2.(b), 3.(b) e 4.(b) apenas 450 são finalizadas com sucesso, enquanto as outras 50 são canceladas. As 50 tarefas canceladas referem-se às tarefas críticas. Conforme mostrado na tabela de custos calculados pelo escalonador do sistema, nenhuma política de alocação oferece tempo total inferior ao *deadline* da tarefa, definido em 500 *ms*. Desse modo todas elas são canceladas. Esse é um caso no qual a carga de processamento é incompatível com o *deadline*, pois não há nenhuma alternativa viável para a execução da tarefa em tempo.

Na Figura 7.8 são apresentados novos casos de execução, mantendo os mesmos parâmetros utilizados nos casos da Figura 7.7, porém agora com carga de processamento de $20 * 10^6$ ciclos de CPU, ou seja, 100 vezes inferior à anterior. As Figuras 7.8-1.(a) e 1.(b) consideram um caso com 100 dispositivos IoT e um servidor MEC, enquanto 2.(a) e 2.(b) consideram um caso com 500 dispositivos IoT e um servidor MEC.

Figura 7.8: Variação de carga de processamento das tarefas utilizando a Aplicação 2.



Com carga de processamento reduzida para $20 * 10^6$ ciclos de CPU o escalonador do sistema manteve a alocação de praticamente todas as tarefas no próprio dispositivo. Apenas as 50 tarefas críticas foram alocadas para o servidor MEC, pois a restrição de *deadline* impedia que o processamento fosse realizado localmente, conforme mostrado nos custos da Tabela 7.8.

Tabela 7.8: Custos calculados pelo escalonador do sistema para a Aplicação 2.

Custo	E_{CORE} (*W*s*)	E_{tran} (*W*s*)	T_{CORE} (*s*)	T_{tran} (*s*)	f (*MHz*)	T (*V*)	Pol.
0,3544	0,7040	-	2,5000	-	8	4	1
0,3671	0,0245	1,2728	0,0267	0,2903	750	0,825	2
0,3671	0,0230	1,2728	0,0333	0,2903	600	0,8	2
0,3697	0,0360	1,2728	0,0200	0,2903	1.000	1	2
0,3735	0,0518	1,2728	0,0133	0,2903	1.500	1,2	2
0,3767	1,1000	-	1,2500	-	16	5	1
0,4189	0,3208	-	5,0000	-	4	2,7	1
0,6875	0,0989	2,3324	0,0071	0,5806	2.800	0	3
0,6942	0,1245	2,3324	0,0051	0,5806	3.900	0	3
0,7287	0,2328	-	10,0000	-	2	2,3	1
1,3713	0,1426	-	20,0000	-	1	1,8	1

É interessante observar na Tabela 7.8 que agora a primeira opção de alocação é a política 1. Na Tabela 7.8 os coeficientes de energia e tempo foram mantidos em $4/5$ e $1/5$. Mesmo com DVFS definido em 8 *MHz* e 4 *V*, foi dada preferência a esta alocação, ao invés de executar no servidor local com frequência de operação em 750 *MHz*. O custo final para a primeira linha da tabela de custos foi decisivo para o escalonador manter as tarefas no dispositivo. Porém, as tarefas críticas com *deadline* de 500 *ms* não poderiam executar no próprio dispositivo, pois mesmo com o menor custo o tempo de execução ainda seria 2,5 *s*. Para tarefas críticas, a alocação ocorreu no núcleo com menor tempo total, também considerando os tempos de transmissão de dados. A alocação para essas tarefas foi realizada no servidor MEC, com configuração de DVFS de 1.500 *MHz* e 1,2 *V*, perfazendo um tempo total de $0,0133 + 0,2903 = 0,3037s$. Conforme Figuras 7.8-1.(b) e 2.(b), todas as tarefas foram finalizadas com status de concluídas, pois as tarefas críticas conseguiram terminar a execução antes do *deadline*.

7.5.2 Variação dos coeficientes de custo do sistema

Para a realização desses experimentos foram utilizadas as Aplicações 1 e 2. A Aplicação 1 foi configurada com carga computacional de $200 * 10^6$ ciclos de CPU, ao invés de $2.000 * 10^6$, pois para $2.000 * 10^6$ ciclos de CPU foi observado nos experimentos da Subseção 7.5.1 que essa carga de trabalho não permitiu a finalização das tarefas críticas com status de concluídas. Para a simulação, a Aplicação 1 faz alocação de 500 tarefas, com 100 dispositivos IoT e dois servidores MEC. Já a Aplicação 2 faz alocação de 500 tarefas, com 100 dispositivos IoT, mas utilizando apenas um servidor MEC.

Tabela 7.9: Variação dos coeficientes de custo do sistema utilizando a Aplicação 1.

$C_{E1.0}$	$C_{E2.0}$	$C_{E3.0}$	$C_{E4.0}$	E_{Total}	T_{Total}	f (*MHz*)	T (*V*)	Pol.
0,250	**0,314**	**0,378**	**0,442**	1,518	0,557	750	0,825	2
0,240	0,316	0,384	0,442	1,633	0,490	1.000	1,000	2
0,232	0,324	0,392	0,468	1,791	0,424	1.500	1,200	2
0,267	0,325	0,415	0,506	1,503	0,624	600	0,800	2
0,395	0,573	0,751	0,929	3,322	0,652	2.800	-	3
0,407	0,603	0,800	0,996	3,578	0,632	3.900	-	3
4,067	3,967	3,867	3,544	11,000	12,500	16	5,000	1
7,136	5,939	4,741	3,767	7,040	25,000	8	4,000	1
13,547	10,428	7,308	4,189	3,208	50,000	4	2,700	1
26,822	20,310	13,799	7,287	2,328	100,000	2	2,300	1
53,428	40,190	26,952	13,713	1,426	200,000	1	1,800	1

A Tabela 7.9 apresenta a relação de custos da Aplicação 1 para carga de $200 * 10^6$ ciclos de CPU com coeficientes de custos de energia configurados para $1/5$, $2/5$, $3/5$, $4/5$ e os coeficientes de tempo configurados, respectivamente, em $4/5$, $3/5$, $2/5$ e $1/5$. Os valores em negrito representam o menor custo calculado pelo escalonador do sistema para cada par de coeficientes configurados. Percebe-se que para a Aplicação 1 a variação dos coeficientes de custo não apresentou grandes mudanças na alocação de tarefas. Em todos os 4 casos simulados foi utilizada a política de alocação 2. Pelos custos percebidos, a alocação prioritária foi nos servidores MEC, e na falta de núcleos disponíveis a alocação ocorreu na *Cloud*. Nos 4 casos simulados todas as 500 tarefas foram finalizadas com status de concluídas, visto que as 50 tarefas críticas geradas pela Aplicação 1 tinham como alternativa de alocação o servidor MEC, com frequência de operação em 1.500 *MHz*, gerando um tempo total de processamento no núcleo mais as transmissões de dados de 0,424 *s*, ou seja, inferior ao *deadline* de 0,5 *s*.

Houve apenas uma pequena mudança de alocação entre o Caso 1 ($C_{E1.0}$) e os outros três, pois no Caso 1 o escalonador definiu os valores de DVFS para o núcleo do servidor MEC em 1.500 *MHz* e 1,2 *V*, enquanto nos demais a alocação utilizou 750 *MHz* e 0,825 *V*. A alocação utilizada no Caso 1 apresentou um consumo energético 17,98% maior do que nos outros três casos, porém o tempo total envolvido nas transmissões de dados e processamento nos núcleos reduziu 23,87%. Conforme a necessidade da aplicação os coeficientes podem ser ajustados de acordo para oferecer minimização de energia consumida ou do tempo de execução das tarefas, cabe ao administrador do sistema definir essa relação.

Importante ressaltar que os custos apresentados na Tabela 7.9 são custos por tarefa, logo, o consumo do sistema deve levar em consideração a quantidade de tarefas. Porém

não basta multiplicar a quantidade de tarefas pelo gastos de tempo e energia apresentados na menor opção de custo, pois para tarefas críticas o custo percebido é diferenciado. Há diferença no custo final devido à alocação das tarefas críticas, que são sempre alocadas pelo menor tempo total, independentemente do custo calculado. Um menor tempo de execução pode estar associado a maiores consumos energéticos, dependendo do caso. Os tempos totais e energia total consumida pelo sistema considera os gastos das tarefas normais multiplicado pela sua quantidade mais os gastos das tarefas críticas multiplicado pela sua quantidade.

Tabela 7.10: Variação dos coeficientes de custo do sistema utilizando a Aplicação 2.

$C_{E1.0}$	$C_{E2.0}$	$C_{E3.0}$	$C_{E4.0}$	E_{Total}	T_{Total}	f (*MHz*)	T (*V*)	Pol.
0,01859	0,02607	0,03355	0,04102	0,1455	0,0334	1.500	1,200	2
0,01867	0,02599	0,03332	0,04065	0,1439	0,0340	1.000	1,000	2
0,01877	**0,02597**	**0,03318**	0,04038	0,1428	0,0347	750	0,825	2
0,01894	0,02609	0,03324	0,04039	0,1426	0,0354	600	0,800	2
0,03506	0,04855	0,06203	0,07552	0,2670	0,0647	2.800	-	3
0,03518	0,04855	0,06203	0,07552	0,2696	0,0645	3.900	-	3
0,04067	0,03967	0,03867	0,03767	0,1100	0,1250	16	5,000	1
0,07136	0,05939	0,04741	**0,03544**	0,0704	0,2500	8	4,000	1
0,13547	0,10428	0,07308	0,04189	0,0321	0,5000	4	2,700	1
0,26822	0,20310	0,13799	0,07287	0,0233	1,0000	2	2,300	1
0,53428	0,40190	0,26952	0,13713	0,0143	2,0000	1	1,800	1

Na Tabela 7.10 são apresentados os custos calculados pelo escalonador do sistema para a Aplicação 2 com carga de $2 * 10^6$ ciclos de CPU variando os coeficientes de custo para energia e tempo. Os custos mais baixos estão destacados em negrito. Para os Casos 2 e 3 não houve mudança no consumo energético ou no tempo total, visto que o menor custo foi encontrado na mesma faixa de frequência, tensão e política de alocação. No Caso 1 o menor custo, e a correspondente alocação, ocorreu para o servidor MEC, com DVFS configurado em 1.500 *MHz* e 1,2 *V*. Já para o Caso 4, com coeficiente de energia em $4/5$ e o coeficiente de tempo em $1/5$ a alocação ocorreu no próprio dispositivo, com DVFS configurado em 8 *MHz* e 4 *V*. A Tabela 7.11 apresenta um resumo dos menores custos em cada caso para facilitar a comparação dos tempos e energias consumidas.

Tabela 7.11: Menores custos calculados para cada caso da Tabela 7.10.

Caso	Custo	E_{Total}	T_{Total}	f (*MHz*)	T (*V*)	Política
$C_{E1.0}$	0,01859	0,14550	0,03336	1.500	1,200	2
$C_{E2.0}$	0,02597	0,14276	0,03469	750	0,825	2
$C_{E3.0}$	0,03318	0,14276	0,03469	750	0,825	2
$C_{E4.0}$	0,03544	0,07040	0,25000	8	4,000	1

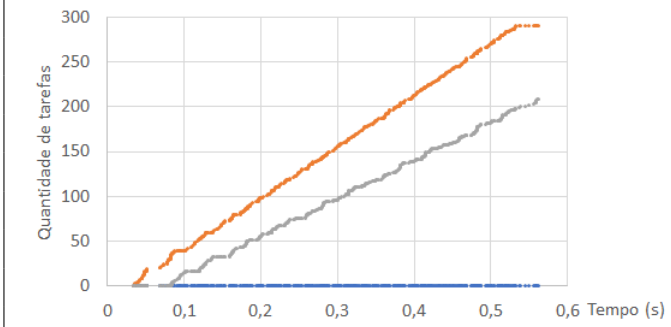
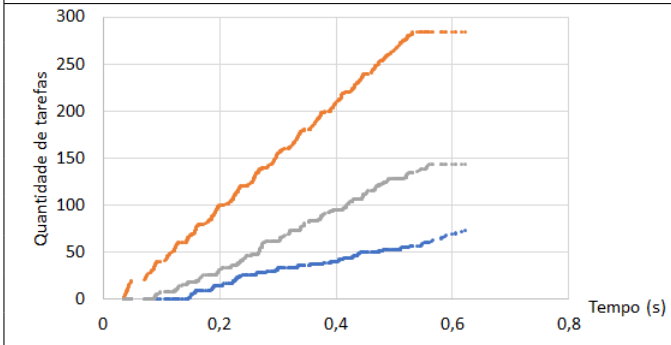
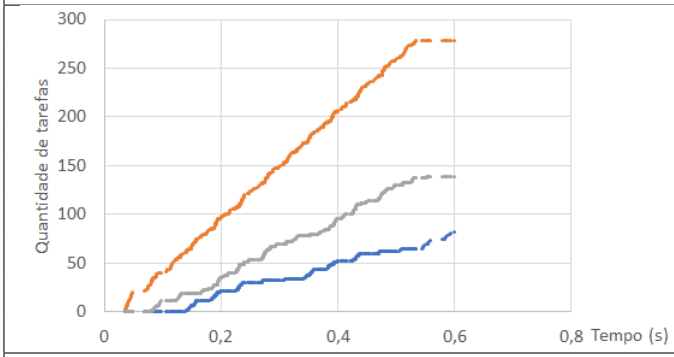
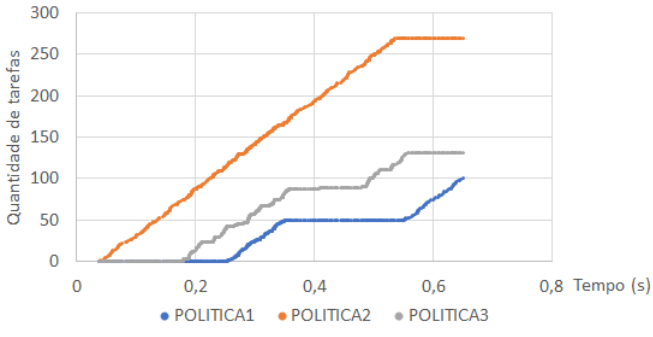
Na Tabela 7.11 podemos observar que os gastos com tempo e energia para os Casos 1, 2 e 3 são praticamente os mesmos, mesmo com o custo calculado bastante diverso. Nesses três casos a política de alocação foi a mesma, ou seja, a alocação prioritária foi realizada no servidor MEC. Já no Caso 4 a política de alocação com menor custo calculado é a política 1, com o processamento destinado ao próprio dispositivo IoT. Com foco em reduzir o consumo energético, a escolha dos coeficientes $4/5$ e $1/5$ para, respectivamente, energia e tempo, oferece uma redução de 51,61% no consumo energético das tarefas normais. Se o foco for na redução do tempo de finalização da tarefa, coeficientes para energia de $1/5$, $2/5$ e $3/5$, com seus respectivos complementos para coeficiente de tempo, é possível obter uma redução de 86,65% nos tempos de finalização das tarefas normais.

Quanto à finalização de tarefas, não houve nenhum cancelamento devido a estouro de *deadline*, pois com *deadline* de 100 *ms* havia opções de alocação que permitiram o processamento da tarefa em tempo hábil. Com os tempos totais, abaixo de 100 *ms* a alocação pode ser feita no servidor local ou na *Cloud*. Para as tarefas críticas o consumo energético e o tempo total decorrido são os mesmos nos 4 casos, 0,1455 *W*s* e 0,0334 *s*, com a política de alocação 2 e parâmetros de DVFS configurados para 1.500 *MHz* e 1,2 *V*. Se o servidor MEC não tiver nenhum núcleo disponível, então a política de alocação 3 é escolhida, com consumo energético de 0,26960 *W*s*, 0,0645 *s* de tempo total e DVFS configurado para 3,9 *GHz*. Esses consumos são por tarefa.

Na Tabela 7.12 é apresentado o comportamento da alocação de tarefas pelo escalonador do sistema, mantendo 500 tarefas, 100 dispositivos IoT e um servidor MEC, conforme os custos percebidos e os recursos disponíveis, tanto para alocação na CPU do dispositivo IoT como nos núcleos dos servidores MEC. Para simplificar as tabelas, as duas últimas linhas de custo para cada caso foram suprimidas. Para o Caso 1 a alocação ficou concentrada no servidor MEC e na sequência na *Cloud*, sem nenhuma alocação para os dispositivos IoT. Isso ocorreu, pois conforme os custos calculados, a política 2 foi a primeira opção e na ausência de núcleos livres no servidor MEC, a política 3 é escolhida em sequência. Já para as tarefas críticas, conforme Tabela 7.10, o menor tempo total ocorre no servidor MEC e na ausência de núcleos livres a *Cloud* oferece o segundo menor tempo total. Assim, nenhuma alocação foi realizada no próprio dispositivo.

Os Casos 2, 3 e 4 são semelhantes. Dentre os menores custos está a alocação da tarefa no próprio dispositivo. À medida que as alocações são feitas no servidor MEC e seus núcleos ficam ocupados, naturalmente é escolhida a opção de alocação no próprio

Tabela 7.12: Comportamento da alocação de tarefas para a Aplicação 2 ao variar os coeficientes de energia e tempo.

Comportamento da alocação	$C_{E1.0}$	**Política**
	0,01859	2
	0,01867	2
	0,01877	2
	0,01894	2
	0,03506	3
	0,03518	3
	0,04067	1
	0,07136	1
	0,13547	1
Comportamento da alocação	$C_{E2.0}$	**Política**
	0,02597	2
	0,02599	2
	0,02607	2
	0,02609	2
	0,03967	1
	0,04855	3
	0,04855	3
	0,05939	1
	0,10428	1
Comportamento da alocação	$C_{E3.0}$	**Política**
	0,03318	2
	0,03324	2
	0,03332	2
	0,03355	2
	0,03867	1
	0,04741	1
	0,06203	3
	0,06203	3
	0,07308	1
Comportamento da alocação	$C_{E4.0}$	**Política**
	0,03544	1
	0,03767	1
	0,04038	2
	0,04039	2
	0,04065	2
	0,04102	2
	0,04189	1
	0,07287	1
	0,07552	3

dispositivo. Porém, quando novas tarefas são geradas, sem que as anteriores tenham sido concluídas, tanto o servidor MEC quanto os dispositivos IoT ficam com seus núcleos ocupados, dando espaço para alocação na *Cloud*. Assim, há um equilíbrio na alocação

entre as três camadas da arquitetura, de acordo com a disponibilidade de recursos em cada camada e os custos calculados para cada política de alocação pelo escalonador do sistema.

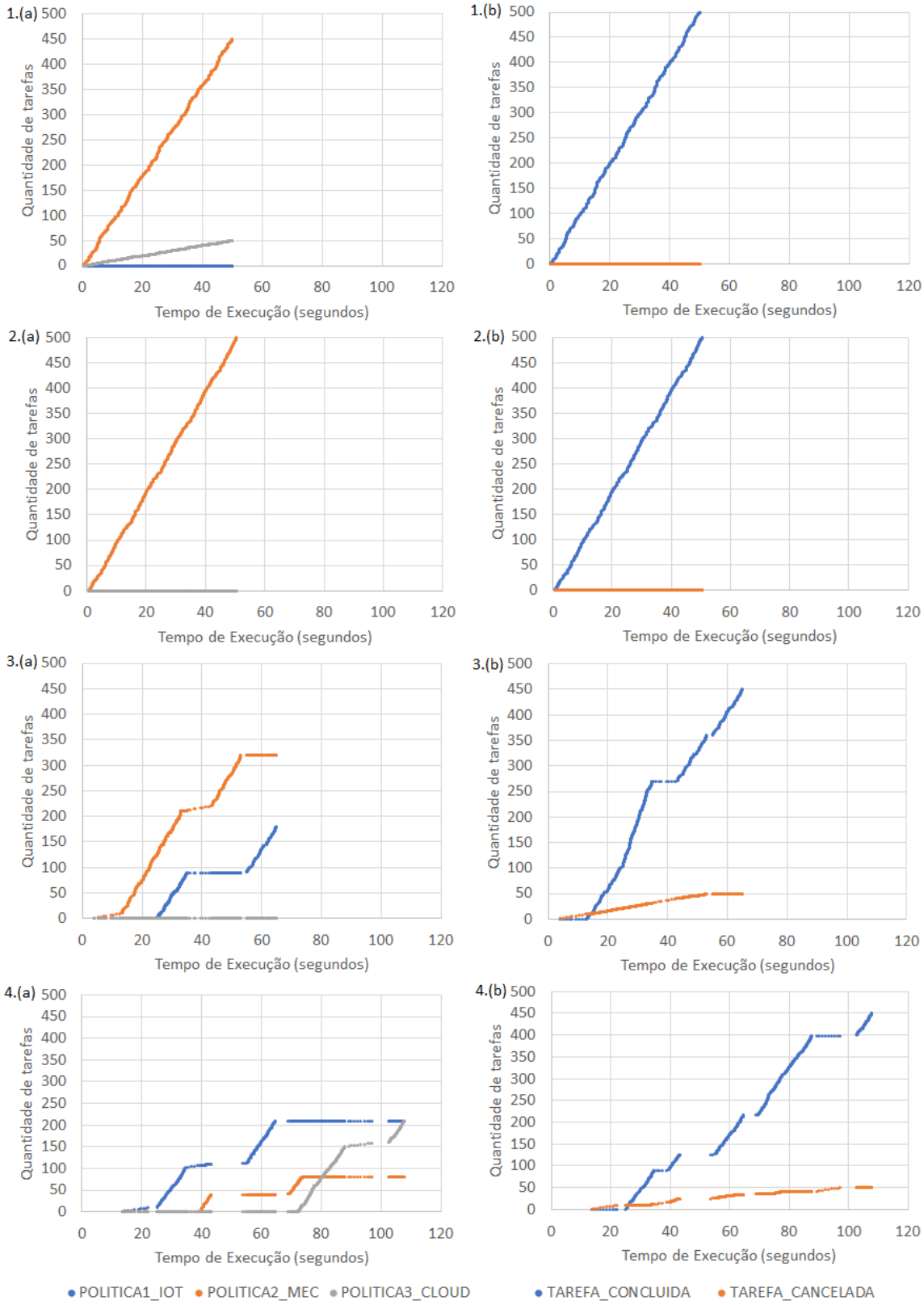
7.5.3 Variação do tamanho da entrada de dados

Para essa simulação é utilizada a Aplicação 1, com carga de trabalho configurada em $200 * 10^6$ ciclos de CPU, coeficiente de custo energético em $4/5$, 500 tarefas, 100 dispositivos IoT e 2 servidores MEC. Os demais parâmetros seguem ajustados conforme concepção original da Aplicação 1. Foram utilizadas entradas de 4 tamanhos distintos, variando em ordens de 10 vezes. Os tamanhos de entradas de dados utilizados foram $36,288 * 8 * 10^5$ *bits* ($\propto$ 3,6 *MB*), $36,288 * 8 * 10^6$ *bits* ($\propto$ 36 *MB*), $36,288 * 8 * 10^7$ *bits* ($\propto$ 362 *MB*) e $36,288 * 8 * 10^8$ *bits* ($\propto$ 3,6 *GB*).

Na Figura 7.9 é apresentada a progressão na alocação de tarefas para cada caso analisado; as entradas de dados utilizadas são (1) 3,6 *MB*, (2) 36 *MB*, (3) 362 *MB* e (4) 3,6 *GB*. Para o Caso 1 a alocação prioritária ocorre no servidor MEC e na sequência o menor custo percebido faz com que as alocações sejam na *Cloud*. Não há alocações nos dispositivos IoT, pois com entrada de dados pequena, de cerca de 3,6 *MB*, o consumo de energia e tempo para enviar as informações ao servidor MEC ou à *Cloud* são mais vantajosos do que executando a tarefa localmente. Essa diferenciação de custo por política de alocação pode ser observada na Figura 7.10. O custo de executar no dispositivo é muito elevado, e, portanto, a alocação é feita no servidor MEC. Na ausência de núcleos livres a alocação migra para a *Cloud*.

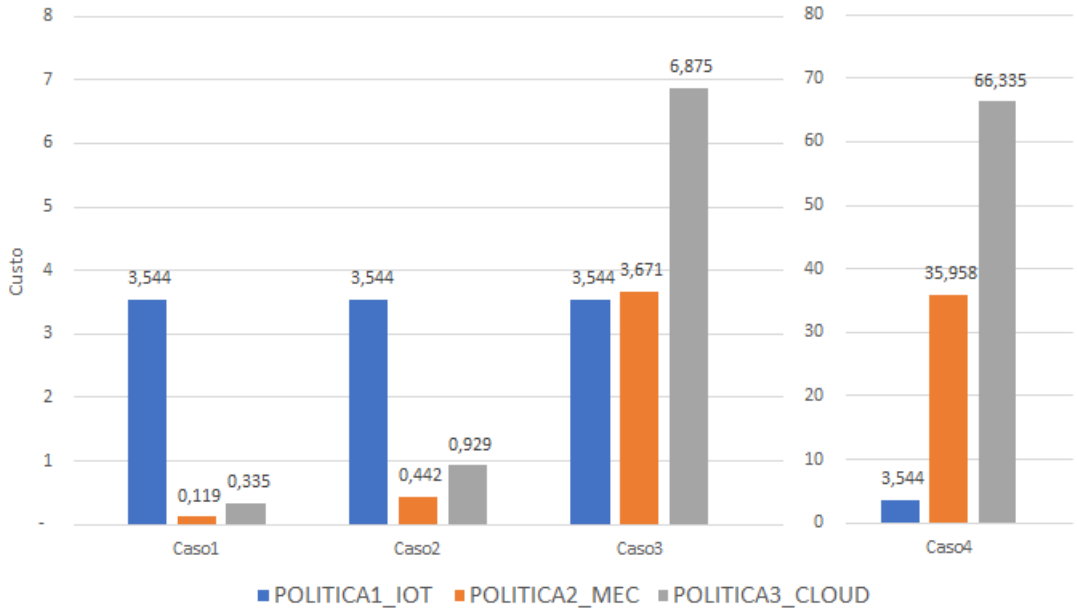
À medida que o tamanho da entrada de dados aumenta, os custos calculados para cada caso progridem nas políticas 2 e 3. O custo da política 1 segue sempre o mesmo, pois não são realizadas transmissões de dados, ou seja, não há adição de consumo de energia ou de tempo decorrido para compor o custo da política. Quando a entrada de dados escala, as políticas de alocação que necessitam de transmissões de dados ficam muito custosas. Conforme o tamanho dos dados aumenta, a alocação no próprio dispositivo torna-se cada vez mais vantajosa. Em situações reais, entretanto, deve se considerar o *hardware* disponível para o armazenamento local de grandes volumes de informações. Caso fossem inseridos outros gatilhos de custo ao modelo do sistema, o custo de armazenamento de dados poderia ser levado em consideração, o que levaria os custos das três políticas a uma outra relação de pesos, também aumentando para a política 1 no Caso 4, por exemplo. Em

Figura 7.9: Variação do tamanho da entrada de dados usando a Aplicação 1 com carga definida em $200 * 10^6$ ciclos de CPU.



Data Centers, por exemplo, o uso de discos rígidos com capacidade de armazenamento elevada possui custo muito mais baixo do que memórias de estado sólido para armaze-

Figura 7.10: Custos totais percebidos pelo escalonador do sistema para a Aplicação 1 com carga definida em $200 * 10^6$ ciclos de CPU com diferentes tamanhos de entrada de dados. Entrada de dados em (1) 3,6 *MB*, (2) 36 *MB*, (3) 362 *MB* e (4) 3,6 *GB*.



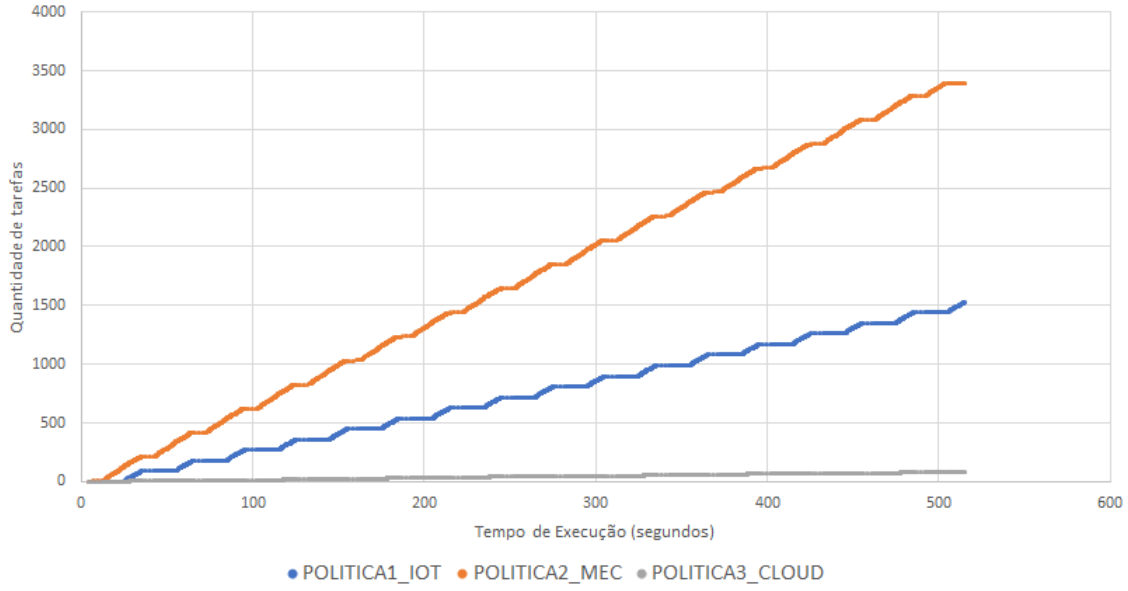
namento de dados em dispositivos móveis. Portanto, um modelo de custo que englobe também essa informação, associando o volume de armazenamento ao custo financeiro, poderia trazer resultados interessantes.

Para os Casos 3 e 4, conforme Figura 7.9, as tarefas críticas foram todas canceladas. Nos dois casos, como o tamanho da entrada de dados aumentou, o tempo total de processamento e transmissão de dados aumentou para o servidor MEC e para a *Cloud*, assim, as políticas 2 e 3 deixaram de ser opções viáveis para processar as tarefas antes da conclusão do *deadline*. A política 1 não sofreu alteração de tempo, mas o tempo de processamento já era superior a 0,5 *s* para a configuração de DVFS mais rápida disponível, e nunca foi uma opção viável. Assim, nos Casos 3 e 4 as tarefas críticas acabaram estourando o *deadline* e foram canceladas.

No Caso 4 ainda houve uma quantidade considerável de alocações na *Cloud*, mesmo com um custo extremamente elevado. Isso ocorre, pois em um primeiro momento as primeiras tarefas recém geradas são destinadas ao processamento no próprio dispositivo, ocupando os núcleos. Tarefas críticas optam pelo menor custo de tempo, que nesse caso é o próprio dispositivo IoT, porém como os núcleos estão ocupados, a alocação das tarefas críticas migra ao servidor MEC. À medida que novas tarefas são geradas, com as CPUs dos dispositivos IoT ocupadas e os núcleos do servidor MEC ocupados aguardando a conclusão das transferências de dados (3,6 *GB* de dados), a opção de alocação que resta às tarefas normais é a *Cloud*. Mesmo que o processamento não seja muito de-

morado no servidor MEC, recursos de processamento ficam indisponíveis enquanto há transferência de arquivos na rede, e sem CPUs livres nos dispositivos IoT, a *Cloud* surge como alternativa.

Figura 7.11: Novo caso para experimentação de tamanho da entrada de dados usando a Aplicação 1 com carga definida em $200 * 10^6$ ciclos de CPU.



Assim, é importante dimensionar as aplicações de modo que as transferências de dados pela rede não sejam muito grandes por tarefa, evitando o congestionamento na rede ou a alocação de recursos que ficam em *stand-by* aguardando a conclusão da transferência de arquivos. É preferível que a entrada seja dividida e processada por mais tarefas, diminuindo a ociosidade dos núcleos. Na Figura 7.11 é apresentado um novo caso de experimentação (Caso 5) com as mesmas características do Caso 3, porém com 10 vezes mais tarefas e cada tarefa com $1/10$ dos dados com objetivo de demonstrar essa argumentação. O Caso 5, portanto, possui 5.000 tarefas com entrada de dados de 362 *MB* cada, enquanto o Caso 4 possui 500 tarefas com entrada de dados de 3,6 *GB* cada.

O Caso 5 segue o mesmo comportamento de alocação de tarefas demonstrado no Caso 3, e diferentemente do Caso 4, que acaba tendo muitas alocações na *Cloud*, tanto o Caso 3 (500 tarefas) como o Caso 5 (5.000 tarefas), utilizam em larga escala os recursos disponíveis nos dispositivos IoT e nos servidores MEC. Não há quase nenhuma alocação na *Cloud*, pois a transferência dos lotes de 362 *MB* de dados para os servidores MEC não segura os recursos por muito tempo e a finalização das tarefas ocorre antes do *deadline*. Assim que novas tarefas são criadas os recursos de processamento já estão novamente disponíveis. Além disso, a pequena quantidade de alocações na *Cloud* reduziu a energia consumida e o tempo dispendido decorrentes das transmissões de dados. No Caso 4, a

energia e tempo consumidos apenas com transmissões de dados para as 500 tarefas foi de 59.160,92 *W*s* e 14.515,21 *s*. Já para o Caso 5 esse custo foi de, 45.011,49 *W*s* e 10.305,94 *s*, ou seja, uma redução de, respectivamente, 23,92% e 29%. Logo, conforme o exposto, em um ambiente com geração contínua de tarefas com baixo *deadline* e necessidade de baixas latências, as tarefas não devem ser muito intensivas em dados, pois as transmissões de dados agregam gastos adicionais de energia e tempo.

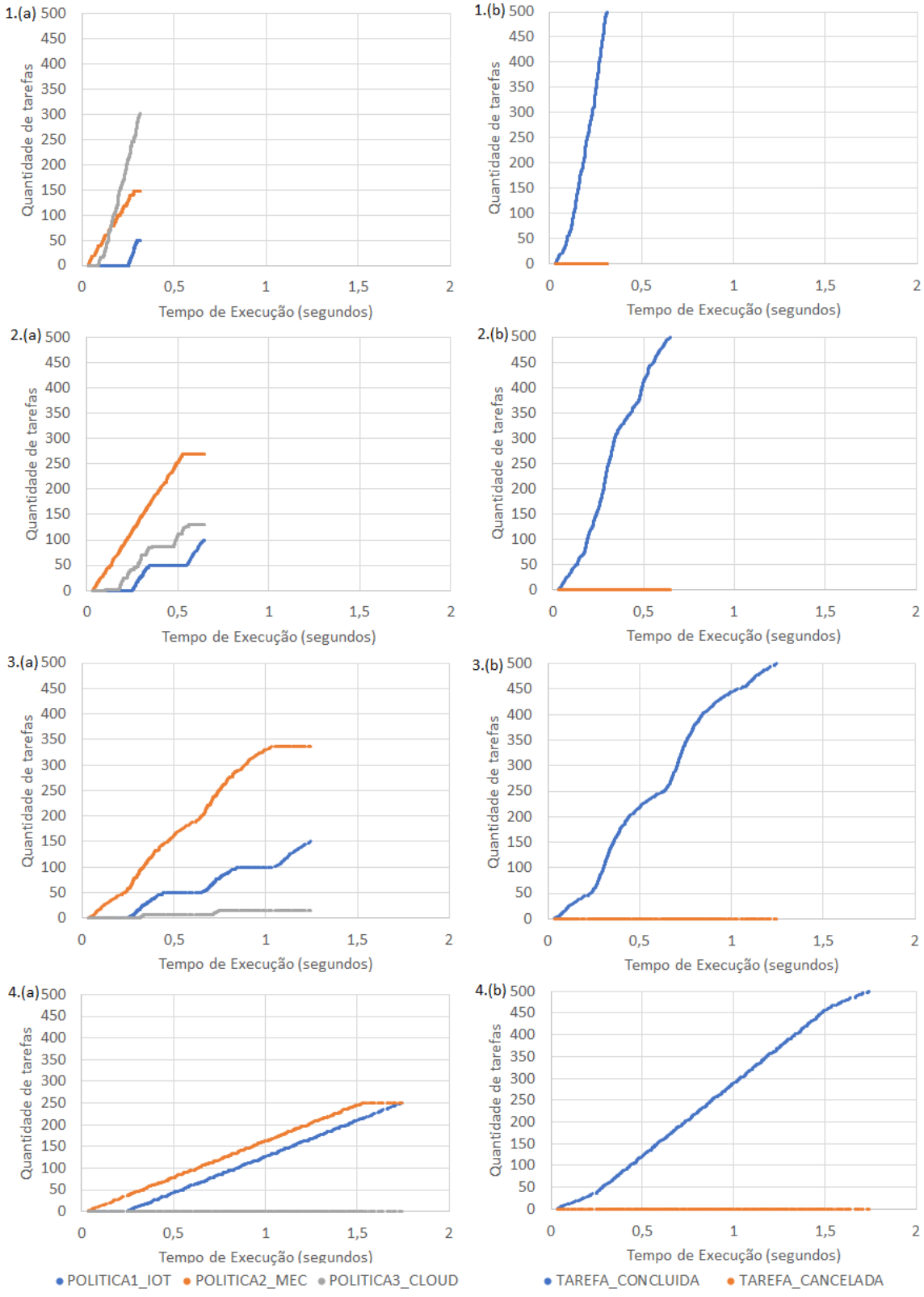
7.5.4 Variação da taxa de geração de tarefas

A variação da taxa de geração de tarefas ocorreu para a Aplicação 2, com 500 tarefas, 100 dispositivos IoT e 1 servidor MEC. Na Figura 7.12 é apresentado o comportamento das alocações realizadas pelo escalonador do sistema para cada caso com taxas de geração de (1) 0,05 *s*, (2) 0,1 *s*, (3) 0,2 *s* e (4) 0,3 *s*.

No Caso 1 é percebida uma grande quantidade de tarefas destinadas à *Cloud*. Embora a *Cloud* possua um custo elevado, de acordo com a Tabela 7.13, isso ocorre, pois inicialmente são criadas 50 tarefas normais e na sequência 50 tarefas críticas. A quantidade de tarefas críticas da Aplicação 2 é de 50%, logo metade dos dispositivos IoT fica dedicado à geração de tarefas normais, enquanto a outra metade fica dedicada à geração de tarefas críticas. Dos 50 dispositivos IoT que geram tarefas normais, a alocação ocorre no próprio dispositivo. Já os 50 dispositivos que geram tarefas críticas, têm a tarefa inicialmente destinada ao servidor MEC. Porém, por haver apenas um servidor MEC com 20 núcleos, assim que ele atinge sua lotação máxima, 30 tarefas críticas são alocadas à *Cloud*, pois conforme a Tabela 7.13, que apresenta os custos para cada local de processamento, a *Cloud* é a segunda política que oferece o menor tempo total para a finalização da tarefa. No Caso 1 a taxa de geração é de 0,05 *s*. Com essa taxa de geração, as primeiras 50 tarefas normais alocadas os dispositivos IoT concluem a execução próximo ao tempo de sistema de 0,3s, pois o processamento no dispositivo é mais lento. Enquanto esse processamento ocorre, as novas tarefas normais e críticas são alocadas, respectivamente, para o servidor MEC e para a *Cloud*. Nesse caso, devido à taxa de geração extremamente rápida, considerando também o perfil de carga de processamento da tarefa, os recursos da rede local ficam saturados, tanto nos dispositivos IoT como no servidor MEC, restando a *Cloud* como última alternativa.

No Caso 2 a taxa de geração de tarefas é de 0,1 *s*, ou seja, em cada dispositivo e a cada 0,1 *s* é gerada uma nova tarefa. Como há um intervalo maior entre um lote de tarefas

Figura 7.12: Variação da taxa de geração de tarefas utilizando a Aplicação 2.



e outro, há mais tempo disponível para que as tarefas no servidor MEC e nos dispositivos IoT executem e finalizem, liberando recursos de processamento para novas tarefas. A utilização da *Cloud* é reduzida, se comparada ao Caso 1. O tempo total do sistema no

Tabela 7.13: Custos calculados pelo escalonador para a Aplicação 2, com coeficiente de energia em $4/5$. Valores ordenados por custo.

Custo	E_{Total} (*W*s*)	T_{Total} (*s*)	f (*MHz*)	T (*V*)	Política
0,0354	0,0704	0,2500	8	4	1
0,0377	0,1100	0,1250	16	5	1
0,0404	0,1428	0,0347	750	0,825	2
0,0404	0,1426	0,0354	600	0,8	2
0,0406	0,1439	0,0340	1.000	1	2
0,0410	0,1455	0,0334	1.500	1,2	2
0,0419	0,0321	0,5000	4	2,7	1
0,0729	0,0233	1,0000	2	2,3	1
0,0755	0,2670	0,0647	2.800	0	3
0,0762	0,2696	0,0645	3.900	0	3
0,1371	0,0143	2,0000	1	1,8	1

Caso 2 é superior ao do Caso 1, mas esse é um comportamento esperado, pois no Caso 1 todas as tarefas são criadas até o tempo de sistema de 0,25 *s*. Após a geração da última tarefa, sua alocação e conclusão, o tempo final do sistema fica em torno de 0,3 *s*. Já no Caso 2 o tempo de sistema ultrapassa 0,6 *s*. Deve ser levado em consideração que são criados 5 lotes de tarefas, cada lote a cada 0,1 *s* (taxa de geração). Após a última tarefa ser criada, alocada e finalizada, o tempo de sistema fica próximo de 0,65 *s*.

O Caso 3 já possui taxa de geração de tarefas de 0,2 *s*. Essa mudança proporciona ao sistema maior utilização dos recursos locais. O escalonador busca sempre o menor custo, mas também depende da disponibilidade de recursos de processamento. Uma taxa de geração de tarefas adequada à aplicação possibilita maior utilização dos recursos locais. Porém, para aplicações que a taxa de geração baixa seja uma exigência, uma alternativa para evitar alocações na *Cloud* seria disponibilizar mais servidores MEC na rede local.

Por fim, no Caso 4, nenhuma alocação foi realizada na *Cloud*. Houve um equilíbrio entre as alocações realizadas entre servidor MEC e dispositivos IoT, visto que as tarefas tiveram tempo suficiente para sua conclusão entre um lote de tarefas e outro, sempre havendo recursos de processamento disponíveis na rede local. Mais uma vez observa-se a importância de uma taxa de geração de tarefas adequada, pois foi possível maximizar a utilização dos recursos da rede local, com todas as tarefas finalizadas com status de concluídas.

A Tabela 7.14 permite a visualização dos consumos de energia e tempos decorridos para processamento no núcleo e transmissões de dados. Os Casos 1, 2, 3 e 4 correspondem, respectivamente, às taxas de geração (1) 0,05 *s*, (2) 0,1 *s*, (3) 0,2 *s* e (4) 0,3 *s* apresentados na Figura 7.12. Com a taxa de geração de tarefas configurada para 0,05 *s* o

Tabela 7.14: Somatórios dos custos para diferentes taxas de geração de tarefas utilizando a Aplicação 2.

Casos	E_{CORE}	E_{trans}	E_{total}	T_{CORE}	T_{trans}	T_{total}	$Custo_{total}$
Caso 1	7,519	98,303	105,822	12,954	24,038	36,992	92,056
Caso 2	6,519	82,416	88,935	13,138	19,688	32,826	77,714
Caso 3	5,924	72,954	78,878	13,248	17,097	30,344	69,171
Caso 4	5,306	63,141	68,447	13,366	14,409	27,776	60,313

consumo energético aumentou, seja a energia consumida para processamento no núcleo ou para transmissões de dados, enquanto o tempo decorrido nas transmissões de dados também aumentou. O tempo de processamento teve uma pequena queda, se comparada aos demais casos, visto que o processamento na *Cloud* é mais rápido, com núcleos de 2,8 *GHz* e 3,9 *GHZ*, contra 1,5 *GHz* no servidor local.

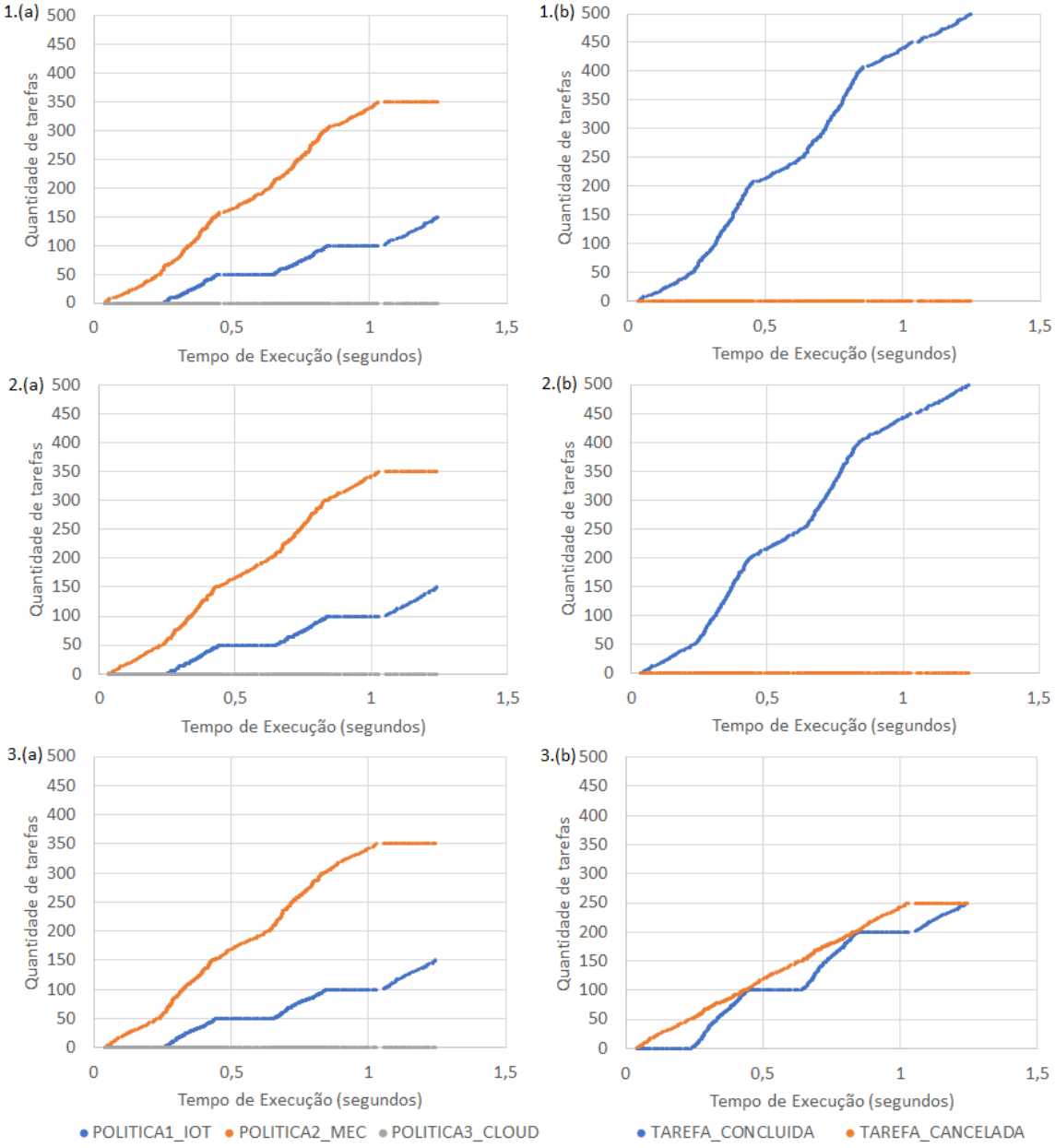
Comparando os 4 casos, é possível verificar que tanto o consumo energético total como o tempo total foram reduzindo progressivamente, visto que o escalonador pôde aumentar a quantidade de tarefas alocadas na rede local conforme o aumento dos períodos da taxa de geração de tarefas. Os Casos 2, 3 e 4, comparados ao Caso 1, apresentaram uma redução no consumo energético total de 15,96%, 25,46% e 35,32%. Já os tempos totais apresentaram redução de 11,26%, 17,97% e 24,91%. Novamente, os resultados reforçam a importância de dimensionar a taxa de geração de tarefas adequadamente, trazendo ganhos para o consumo de energia e para o tempo total decorrido.

7.5.5 Variação do tamanho do *deadline*

Para os experimentos realizados nesta etapa foi utilizada a Aplicação 2, com taxa de geração de tarefas de 0,2 *s* e coeficientes de custo para consumo de energia e tempo em $4/5$ e $1/5$, mantendo as demais características conforme concepção original da aplicação. Na Figura 7.13 são apresentados três casos de *deadlines* distintos (1) 0,1 *s*, (2) 0,05 *s* e (3) 0,01 *s*. Metade das tarefas da Aplicação 2 são críticas. Os experimentos consideraram uma rede com 500 tarefas, 100 dispositivos IoT e 2 servidores MEC.

Os Casos 1 e 2 possuem o mesmo comportamento, pois a mudança no *deadline* da aplicação de 0,1 *s* para 0,05 *s* não é suficiente para causar qualquer impacto no status de finalização das tarefas. Em ambos os casos o escalonador do sistema busca a política de alocação que oferece o menor tempo de execução, sendo, neste caso, o servidor MEC, para os parâmetros de DVFS em 1.500 *MHz* e 1,2 *V*. Todas as tarefas, sejam elas normais ou críticas, são finalizadas com status de concluídas. Há mais execuções de tarefas no

Figura 7.13: Variação do tamanho do *deadline* utilizando a Aplicação 2.



servidor MEC do que nos dispositivos IoT, pois as tarefas críticas são sempre alocadas para o servidor MEC para execução. As tarefas normais tem prioridade de execução no próprio dispositivo, porém, conforme novos lotes são gerados, alguns dispositivos IoT já possuem uma tarefa em andamento, logo, a nova tarefa é alocada para o servidor MEC.

Quanto ao perfil de alocação de tarefas no sistema, o Caso 3 é idêntico aos Casos 1 e 2. A grande diferença está nas tarefas críticas, pois todas são canceladas. Isso ocorre devido à criticidade ser muito elevada. A carga computacional exigida pela aplicação, de $2 * 10^6$ ciclos de CPU, não consegue ser processada em tempo hábil por nenhuma política de alocação. No próprio dispositivo com frequência de operação em 16 *MHz* o menor

tempo possível para processar essa carga computacional é de 125 *ms*. No servidor MEC com 1.500 *MHz* o tempo de processamento é de 1,3 *ms*, e somado ao tempo necessário para realizar a transmissão da entrada de dados e dos resultados o tempo total sobe para 33,4 *ms*. Por fim, a *Cloud* com 3,9 *GHz* na opção *Turbo Boost* oferece tempo de processamento de 0,51 *ms*, mas as transmissões de dados de entrada da tarefa e dos resultados eleva o tempo total para 64,5 *ms*. Com *deadline* de 0,01 *s* sempre haverá estouro, independentemente do local de alocação e variáveis de DVFS definidas.

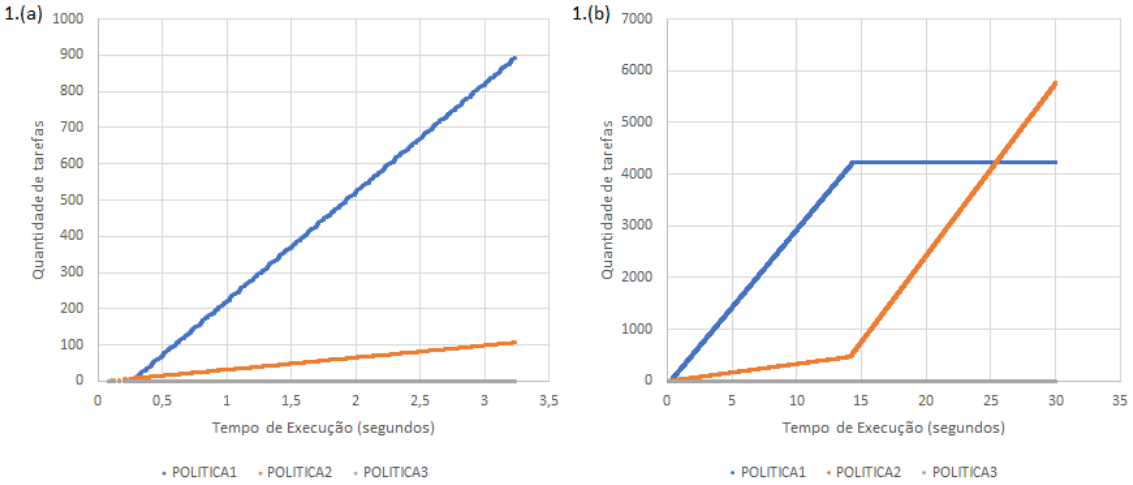
Assim, é importante que o *deadline* da aplicação seja dimensionado adequadamente. Do mesmo modo como foi mencionado para os experimentos realizados com variações da taxa de geração de tarefas, um *deadline* muito pequeno traz problemas à rede. Para a taxa de geração de tarefas o problema principal foi a progressão no consumo energético, devido ao aumento expressivo de alocações na *Cloud*. Quanto ao *deadline*, se dimensionado com valor muito pequeno, as tarefas críticas não serão concluídas. Com poder de processamento alto no próprio dispositivo a tarefa poderia ser concluída em 10 *ms* caso tivesse um núcleo de 2 *GHZ* ou mais, porém a bateria iria exaurir rapidamente. A alternativa, portanto, é conhecer a aplicação e definir um *deadline* suficiente para que seja garantida a Qualidade de Serviço.

7.5.6 Variação do nível de bateria dos dispositivos IoT

Até agora o nível de bateria dos dispositivos IoT não foi apresentado nos experimentos, pois conforme definido inicialmente, um dispositivo IoT possui uma bateria de 2.000 *mAh*, o equivalente a 36.000 *Ws*, e nenhum experimento foi capaz de consumir a bateria até que fosse atingido o LIS. Assim, nesse experimento é utilizada a Aplicação 2, com taxa de geração de tarefas de 0,3 *s*, *deadline* de 0,1 *s*, percentual de tarefas críticas em 10% e nível de bateria em 3,6 *W*s*, com objetivo de forçar a exaustão da bateria. A rede possui um total de 10.000 tarefas, 100 dispositivos IoT e 2 servidores MEC.

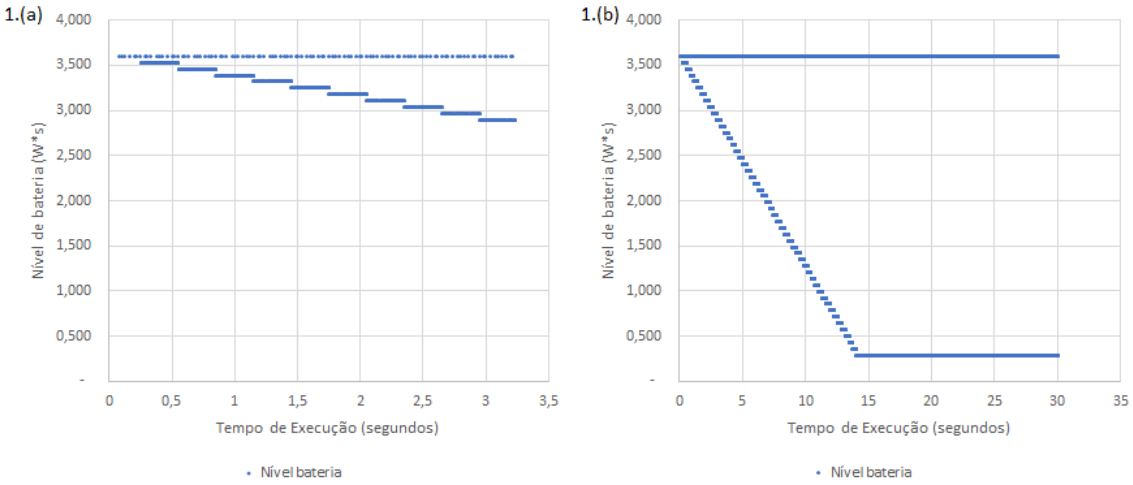
A Figura 7.14 apresenta o comportamento da alocação para as 10.000 tarefas geradas no sistema. A Figura 7.14-1.(a) apresenta o comportamento inicial da alocação, das primeiras 1.000 tarefas, enquanto os dispositivos mantinham um nível de bateria saudável. A alocação ocorreu preferencialmente nos dispositivos IoT para as tarefas normais, já as tarefas críticas tiveram alocação prioritária no servidor MEC, correspondendo a 10% das tarefas geradas pelos dispositivos. Na Figura 7.14-1.(b) está expresso o comportamento completo do sistema para a alocação das 10.000 tarefas. Em dado momento as alocações

Figura 7.14: Alocação de tarefas conforme o nível de bateria dos dispositivos IoT varia. Utilização da Aplicação 2.



nos dispositivos IoT cessam e ocorrem apenas nos servidores MEC.

Figura 7.15: Nível de bateria dos dispositivos IoT ao alocar tarefas utilizando a Aplicação 2.

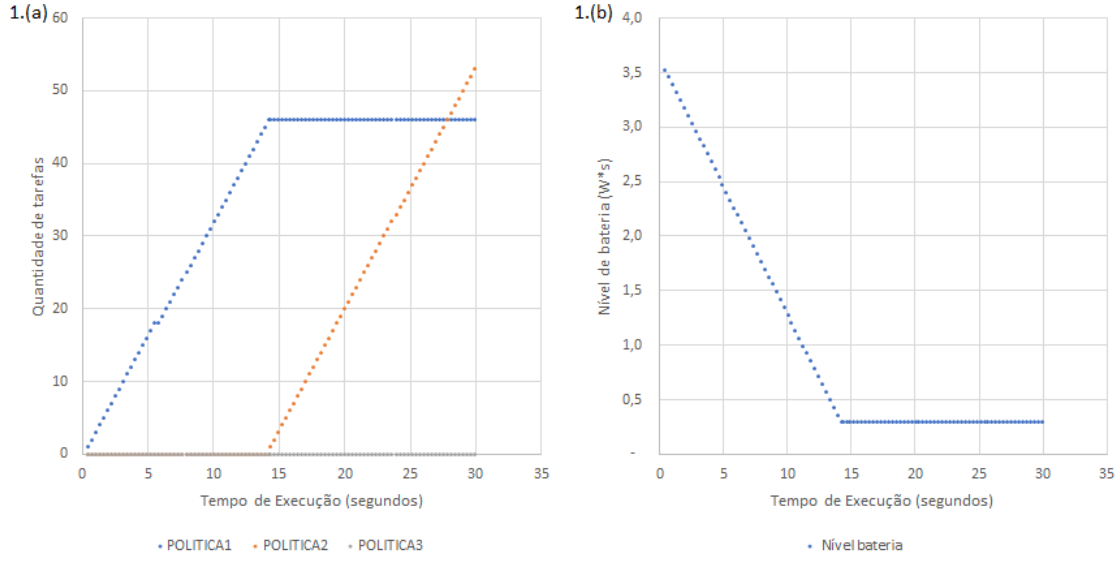


A mudança de perfil na alocação de tarefas observada na Figura 7.14 ocorre porque o nível de bateria dos dispositivos IoT atinge um patamar crítico, igual ou inferior ao LIS, e o escalonador opta em preservar a bateria remanescente dos dispositivos escolhendo outras políticas para alocar as novas tarefas. Na Figura 7.15 é apresentada a progressão do nível da bateria dos dispositivos IoT destinados à geração de tarefas normais e daqueles destinados à geração de tarefas críticas. Na Figura 7.15-1.(a) é apresentado um corte da execução total da simulação com apenas o resultado para as primeiras 1.000 tarefas alocadas e seus efeitos no nível de bateria dos dispositivos IoT. Os degraus observáveis correspondem ao consumo de energia realizado por cada lote de tarefas geradas. A linha pontilhada na horizontal representa o nível de bateria dos dispositivos dedicados à geração

de tarefas críticas. Como elas são sempre alocadas no servidor MEC devido ao tempo de conclusão, o nível de bateria desses dispositivos segue o mesmo. Já na Figura 7.15-1.(b), próximo ao tempo de 15 segundos de execução da simulação, os dispositivos IoT que geram tarefas normais atingem um nível de bateria crítico, e as alocações deixam de ser feitas nos dispositivos, passando a ser somente no servidor MEC, conforme Figura 7.14.

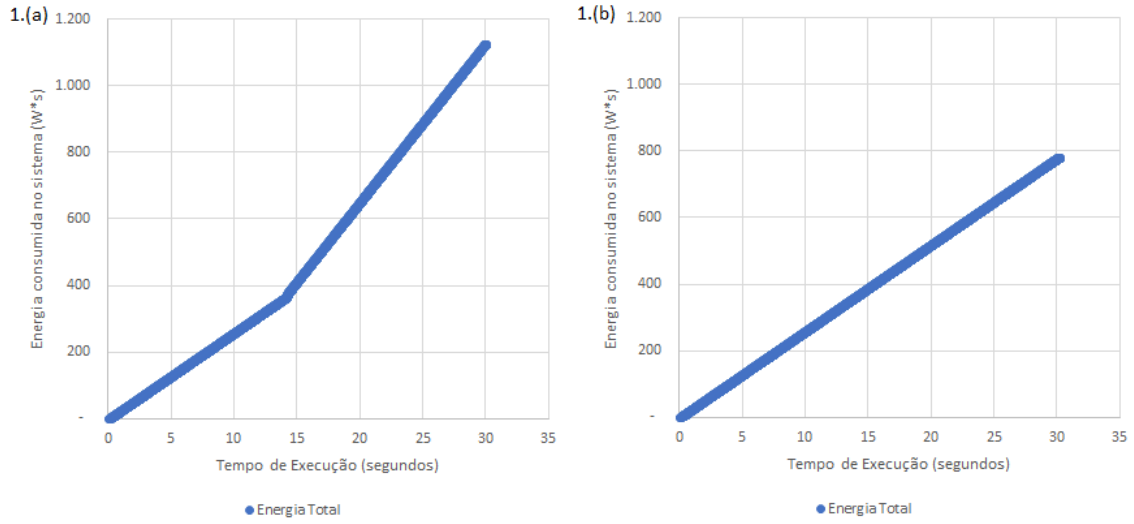
Quanto à finalização de tarefas, não houve nenhuma tarefa cancelada, visto que o processamento das tarefas críticas pôde ser suprido pelo servidor MEC, entregando tempo total inferior ao *deadline*. As tarefas normais foram preferencialmente executadas no próprio dispositivo, que ofereceu o menor custo calculado, considerando os coeficientes de $4/5$ para consumo energético e $1/5$ para tempo decorrido, e a partir do momento em que o nível de bateria dos dispositivos IoT atingiu um patamar crítico, a alocação foi redirecionada ao servidor MEC, também concluindo essas tarefas corretamente.

Figura 7.16: Alocação de tarefas e nível de bateria de apenas um dispositivo IoT utilizando a Aplicação 2.



Na Figura 7.16 estão separados, para maior clareza, o comportamento de alocação de tarefas e o consumo de bateria de um único dispositivo IoT. Inicialmente a alocação ocorre com a política 1, com processamento no próprio dispositivo, até próximo ao tempo de simulação de 15 *s*. Enquanto isso o nível de bateria é consumido progressivamente até atingir o LIS. Após, o perfil de alocação muda, estagnando as alocações no próprio dispositivo e novas alocações ocorrem somente no servidor MEC. Durante as alocações no servidor MEC o nível de bateria do dispositivo segue inalterado. Já para os dispositivos de tarefas críticas, o nível de bateria segue no máximo, com alocações somente no servidor MEC.

Figura 7.17: Progressão da energia total para dispositivos IoT com nível de bateria em (1.(a)) 3.6 *W*s* e (1.(b)) 36.000 *W*s*.



O consumo de energia total do sistema teve um comportamento semelhante à alocação de tarefas pela política 2, visto que com maior alocação de tarefas para o servidor MEC, há maior consumo energético por tarefa. Isso faz o consumo do sistema apresentar uma nova curva de crescimento quando é atingido o LIS nas baterias dos dispositivos IoT, conforme exposto na Figura 7.17. Porém, comparado a outro caso de simulação, com as mesmas características do primeiro, mas no qual os dispositivos IoT não atingem o LIS, o consumo total da rede é reduzido, de acordo com a Figura 7.17-1.(b). Isso ocorre porque o escalonador do sistema escolhe os menores custos, resultando em um consumo energético total menor. No caso 1.(a) o consumo é de 1.124,27 *W*s*, contra 779,10 *W*s* do caso 1.(b) que não atinge LIS. Há um aumento no consumo energético de 44,30% entre os dois casos. Assim, é importante que o nível de bateria do dispositivo siga sempre saudável, evitando a indisponibilidade do equipamento ou mesmo restringindo as opções de alocação por baixo nível de bateria, podendo elevar o consumo energético do sistema.

7.5.7 Cenário com e sem uso de DVFS

Para esse experimento foi utilizada a Aplicação 1, com suas características originais, mas com carga de processamento de $200 * 10^6$ ciclos de CPU. Os coeficientes de energia e tempo são configurados em $4/5$ e $1/5$. O cenário de simulação com uso de DVFS, não possui nenhuma alteração no algoritmo de escalonamento e nas premissas básicas de pares de frequência e tensão para dispositivos móveis, servidores MEC e para a *Cloud*. No cenário que não utiliza DVFS foram utilizados pares de frequência e tensão

padrão de cada equipamento. Nos dispositivos IoT foi utilizado o par 16 *MHz* e 5 *V* e nos servidores MEC foi utilizado o par 1.500 *MHz* e 1,2 *V*. Já na *Cloud* foi utilizada a frequência de 2,8 *GHz* e potência dinâmica de 13,85 *W*. Ambos os experimentos foram executados com 500 tarefas, 100 dispositivos IoT e 1 servidor MEC.

Tabela 7.15: Cenário sem uso de DVFS para a Aplicação 1 com carga de $200 * 10^6$ ciclos de CPU.

Custo	E_{CORE} (*W*s*)	E_{tran} (*W*s*)	T_{CORE} (*s*)	T_{tran} (*s*)	f (*MHz*)	T (*V*)	Pol.
0,506	1,791	1,273	0,424	0,290	1.500	1,2	2
0,929	3,322	2,332	0,652	0,581	2.800	-	3
3,767	11,000	-	12,500	-	16	5,0	1

Os custos percebidos pelo escalonador do sistema para a Aplicação 1 com DVFS são os mesmos apresentados na Tabela 7.9 para coeficiente de energia em $4/5$ e coeficiente de tempo em $1/5$. Já os custos para a Aplicação 1 sem a utilização da técnica de DVFS estão na Tabela 7.15. Para a abordagem sem DVFS há bem menos opções de alocação disponíveis para seleção pelo escalonador do sistema. As opções com frequências mais baixas e tensões mais baixas não estão listadas, pois sem a técnica de DVFS apenas a frequência e tensão padrão é utilizada em cada equipamento.

Tabela 7.16: Cenário com e sem uso de DVFS para a Aplicação 1 com carga de $200 * 10^6$ ciclos de CPU.

Cenário	E_{CORE} (*W*s*)	E_{tran} (*W*s*)	T_{CORE} (*s*)	T_{tran} (*s*)	Custo
Com DVFS	136,18	636,39	126,67	145,17	672,42
Sem DVFS	259,20	636,39	66,67	145,17	758,84

A Tabela 7.16 apresenta os custos da Aplicação 1 para alocação das 500 tarefas, com e sem a técnica de DVFS. Conforme mencionado anteriormente, na abordagem sem DVFS há menos opções disponíveis para seleção. Embora as opções restantes ofereçam tempo de execução inferior nos núcleos do servidor MEC (política 2), o consumo energético é superior. Nos dois casos, todas as tarefas conseguiram ser finalizadas corretamente, e todas as alocações ocorreram no servidor MEC. Como todas as alocações ocorrem no servidor MEC, para todas as tarefas, o tempo de transmissão de dados de entrada e resultado, bem como o consumo energético dessas transmissões de dados, foi o mesmo nos dois casos.

A principal diferença entre as duas abordagens está no consumo energético total e no tempo decorrido total. Para o caso com DVFS ativado, o consumo energético total apresentou uma redução de 13,736%, enquanto o tempo total teve um acréscimo de 28,32%. Isso demonstra a efetividade do modelo do sistema, bem como do algoritmo

de escalonamento, em minimizar o consumo energético total. O tempo total foi superior na abordagem com DVFS, mas não é problemático porque as tarefas da aplicação foram concluídas dentro do limite de tempo imposto pelo *deadline*.

7.6 Considerações Sobre o Capítulo

Este capítulo descreveu as premissas utilizadas para a elaboração dos experimentos, estabelecendo as aplicações utilizadas nas simulações e os custos associados às tecnologias de comunicação e equipamentos da rede, como consumo energético e tempo de processamento. Também foram descritos os detalhes de implementação do simulador *ad-hoc* com as classes desenvolvidas, suas funções e variáveis e o funcionamento de cada uma.

Quanto à experimentação, foram definidos 7 experimentos distintos, com o objetivo de verificar a efetividade do escalonador do sistema. Foi observado o comportamento da alocação de tarefas para as políticas de alocação possíveis, a redução do consumo energético e do tempo total do sistema e o status de finalização das tarefas. Os diferentes cenários simulados utilizaram variáveis de carga de processamento de cada tarefa, relação de coeficientes para energia e tempo do modelo, tamanho da entrada de dados e resultados das aplicações, taxa de geração de tarefas, *deadline*, nível de bateria dos dispositivos IoT e utilização da técnica de DVFS. A seguir encontra-se um resumo dos resultados obtidos na experimentação:

- **Carga de processamento das tarefas:** A variação da carga de processamento das tarefas permitiu observar o perfil de alocação de tarefas na rede. Tarefas com carga computacional elevada tiveram custos elevados para o processamento local no dispositivo e foram alocadas majoritariamente no servidor MEC. À medida que os recursos locais foram saturando, a *Cloud* foi utilizada para prover os recursos necessários. Também foi possível observar que a quantidade de servidores MEC na rede local influiu na economia energética. Para cargas de trabalho pesadas, se comparadas a um sistema sem servidores MEC, o uso de um servidor MEC permitiu redução de energia em 42,51%, enquanto utilizar dois servidores MEC trouxe redução de 44,71% na energia consumida.
- **Coeficientes de custo do sistema:** Com a variação dos coeficientes de custo para energia e tempo, mantendo as demais características iguais, foi possível observar

a mudança no perfil de alocação das tarefas. Coeficientes de energia elevados permitiram maiores reduções de energia consumida, enquanto coeficientes de tempo elevado permitiram reduções nos tempos totais.

- **Tamanho da entrada de dados:** Entrada de dados muito elevadas consumiram grande quantidade de energia em transmissões de dados pela rede. Foi observado que uma abordagem com mais tarefas e menos dados por tarefa, mantendo a quantidade de dados total inalterada, pôde trazer economias de energia e tempo de 23,92% e 29%, respectivamente.
- **Taxa de geração de tarefas:** A taxa de geração de tarefas é um fator crítico para o sistema, visto que taxas de geração muito pequenas geram uma quantidade elevada de tarefas em um período pequeno, inundando a rede com tarefas. Isso faz com que os recursos locais fiquem rapidamente saturados, havendo necessidade de recorrer à *Cloud*, que agrega mais custos energéticos e tempos totais.
- **Tamanho do *deadline*:** *Deadlines* muito pequenos geraram grande quantidade de tarefas canceladas, visto que não havia tempo hábil para que a alocação e processamento ocorressem antes do *deadline* ser atingido. Os experimentos reforçaram a importância de dimensionar o *deadline* adequadamente, de modo a permitir alocação e execução em tempo hábil, antes de atingir o *deadline*, concluindo as tarefas corretamente.
- **Nível de bateria dos dispositivos IoT:** Nos experimentos com nível de bateria, foi observada a importância de manter os dispositivos IoT com um nível de bateria saudável, visto que ao atingir o LIS as tarefas não podem mais ser alocadas no próprio dispositivo. A alocação passa a ser no servidor MEC ou na *Cloud*, aumentando os custos energéticos e os tempos totais. Logo, manter o nível de bateria dos dispositivos IoT saudáveis permite que os custos totais do sistema sejam reduzidos.
- **DVFS ativado e desativado:** O uso da técnica de DVFS permitiu uma redução de 13,73% na energia consumida, com um aumento de 28,32% nos tempos totais, se comparada a uma abordagem que não utiliza a técnica de DVFS. O consumo energético foi otimizado e o aumento dos tempos totais não foram suficientes para atingir o *deadline* das tarefas críticas, ou seja, a conclusão de tarefas ocorreu normalmente.

8 CONCLUSÃO

A geração acelerada de grandes volumes de dados, aliada a aplicações com requisitos restritos de QoS, como baixa latência, e quantidades cada vez maiores de dispositivos conectados à Internet com limitações de bateria, levanta dúvidas sobre a utilização da Cloud como única alternativa para o descarregamento de tarefas (SATYANARAYANAN et al., 2009). Uma camada intermediária, entre dispositivos finais e a Cloud, composta de equipamentos que aproximam do usuário final serviços semelhantes àqueles oferecidos pela Cloud, são uma alternativa viável para a redução da latência das comunicações, bem como da preservação da bateria dos dispositivos móveis da rede (MAHMUD; BUYYA, 2016).

Este trabalho, portanto, desenvolveu, no Capítulo 5, um modelo de custo composto de um conjunto de variáveis sobre os equipamentos da rede e tecnologias de comunicação utilizadas entre seus elementos, para que as tarefas geradas no sistema fossem alocadas, tendo em vista a minimização do consumo energético dos elementos da rede e dos tempos totais para as transmissões de dados e tempo de processamento associados a cada tarefa. Porém, devido à complexidade do modelo proposto e sua elevada carga de processamento, no Capítulo 6 um algoritmo heurístico foi desenvolvido para implementar o modelo, com custo computacional reduzido, objetivando obter uma solução ao modelo com custo computacional reduzido. A complexidade do novo algoritmo foi demonstrada e provou-se viável computacionalmente, obtendo soluções em tempo polinomial (n^2).

A heurística foi incorporada a um escalonador, responsável por utilizar as definições do modelo do sistema na tomada de decisão para a alocação das tarefas. A experimentação do escalonador ocorreu em um simulador desenvolvido pelo próprio autor, constituindo um ambiente bem controlado, que reproduziu o modelo de três camadas do sistema, composto pelos dispositivos móveis, servidores locais, *Data Centers* da *Cloud* e tecnologias de comunicação entre esses equipamentos. O simulador ofereceu experimentação ágil e controle sobre as variáveis observadas, utilizadas para verificar a eficácia do modelo proposto, bem como do algoritmo de escalonamento implementado.

Os experimentos realizados no Capítulo 7 foram importantes para observar o comportamento do sistema quanto a alocação de tarefas por política, quantidade de tarefas concluídas e canceladas, modificação dos custos percebidos pelo escalonador do sistema e seus impactos no consumo energético e nos tempos demandados para a finalização das tarefas. Dentre os experimentos realizados, concluiu-se que o adequado ajuste dos coefi-

cientes de custo para consumo de energia e tempo decorrido foram imprescindíveis para que o custo final percebido pelo algoritmo de escalonamento minimizasse tanto energia consumida como tempo de processamento das tarefas. Coeficientes adequados permitiram que a energia do sistema fosse reduzida em até 51,61% ou que os tempos totais diminuíssem em até 86,65%, finalizando as tarefas antes do *deadline*. O sistema tornou-se mais sustentável e a experiência do usuário não foi afetada. A utilização de servidores MEC permitiu prolongar a vida da bateria dos dispositivos IoT e auxiliou na execução mais ágil de tarefas com perfil de carga de processamento elevado.

Além disso, constatou-se que o dimensionamento do *deadline* da aplicação deve ser compatível com sua carga de trabalho. Conforme discutido na Subseção 7.5.1, havendo um desequilíbrio entre a carga de processamento em ciclos de CPU e o *deadline*, não é possível finalizar a tarefa em tempo hábil, independentemente da política de alocação escolhida. Uma carga de trabalho muito grande não pode vir associada a um *deadline* ínfimo, porque mesmo com *hardware* potente as tarefas críticas serão sempre canceladas.

O tamanho da entrada, por sua vez, fez progredir os custos calculados nas políticas de alocação 2 e 3, tornando-os mais custosos à medida que o tamanho dos dados aumentou. Logo, com entradas maiores, ficou mais vantajoso processar os dados localmente, no próprio dispositivo IoT, oferecendo redução de até 23,92% no consumo energético do sistema para os cenários simulados.

Também foi evidenciado que taxas de geração de tarefas muito pequenas geram tarefas próximos demais umas das outras, que inundam a rede e ocupam todos os recursos disponíveis na rede local, sejam eles dos dispositivos IoT ou dos servidores MEC. Nessa situação a *Cloud* foi uma opção complementar à alocação, mas que agregou custos ao sistema. Taxas de geração adequadas permitiram maximização de uso dos recursos de processamento da rede local, minimizando o consumo energético do sistema em até 35,32%, com a manutenção dos requisitos de QoS da aplicação e sem tarefas canceladas.

Para aplicações que tem como primeira opção de alocação os dispositivos locais, constatou-se que o nível de bateria precisa ser bem dimensionado. Ao atingir o LIS os dispositivos IoT ficaram indisponíveis para novas alocações, restando apenas os servidores MEC e a Cloud como opções possíveis. Nos cenários simulados, o consumo energético aumentou em 44,30%, pois com a bateria praticamente exaurida e os dispositivos IoT indisponíveis, houve maior gasto total com energia consumida ao utilizar as políticas de alocação 2 e 3. Portanto, o nível de bateria dos dispositivos IoT deve ser monitorado e mantido em patamares saudáveis, permitindo que a política 1 siga sendo considerada pelo

escalonador de tarefas.

A técnica de DVFS mostrou-se eficaz para a alocação de tarefas no sistema observando os menores custos disponíveis. Foi possível, apenas com a utilização dessa técnica e mantendo todos os outros parâmetros inalterados, reduzir o consumo energético em 13,76%. É um ganho muito positivo, considerando que os recursos da rede, tipo de aplicação e características das tarefas foram os mesmos para o cenário sem o uso de DVFS.

Em um cenário ideal, a carga de processamento das tarefas deveria ser grande o suficiente para que a alocação fosse vantajosa no próprio dispositivo IoT ou no servidor MEC. Os coeficientes de energia e tempo deveriam ser dimensionados com o melhor equilíbrio possível para que as tarefas tivessem a conclusão no limite dos *deadlines*, minimizando a energia consumida. A técnica de DVFS deveria estar habilitada. O tamanho da entrada de dados não deveria ser muito elevado, evitando transmissões de dados custosas, seja em consumo energético ou em tempo decorrido durante as transferências de arquivos. As taxas de geração de tarefas deveriam ser grandes o suficiente para não haver congestionamento na rede, dando tempo suficiente para a finalização de um lote de tarefas na rede local antes de um novo ser gerado, evitando alocações na *Cloud*, e consequentemente diminuindo o consumo energético da rede e os tempos totais percebidos por cada tarefa. Além disso a rede local deveria oferecer recursos de processamento suficientes para que a alocação de tarefas fosse alternada entre dispositivos IoT e servidores MEC, permitindo consumo energético reduzido e tempos totais menores, evitando a *Cloud*. Tal cenário permitiria o ápice em termos de minimização do consumo energético e dos tempos de processamento, porém dificilmente todas essas características serão observadas no mesmo cenário.

Frente ao exposto, a questão de pesquisa **"É possível desenvolver um modelo de custo com múltiplas variáveis de energia e tempo para processamento de tarefas que permita a redução do consumo de energia e do tempo de processamento em um sistema *Mobile Edge Computing* para dispositivos móveis, com a *Cloud* como alternativa ao descarregamento de tarefas?"**, que fundamenta este trabalho, foi respondida com os resultados obtidos nos experimentos realizados. A eficácia do modelo de custo para o sistema MEC foi comprovada nas diversas simulações realizadas. A implementação do modelo na forma de um escalonador de tarefas com utilização do algoritmo heurístico permitiu a redução do consumo energético e dos tempos totais, de acordo com a ponderação de custos dos coeficientes para energia e tempo, o ajuste dos parâmetros

utilizados nas aplicações e os recursos de processamento oferecidos na rede, com suas respectivas características de frequência de operação e tensão de alimentação para uso da técnica de DVFS.

8.1 Contribuições

Conforme o exposto, as principais contribuições realizadas neste trabalho quanto a resposta à questão de pesquisa e os objetivos estabelecidos, são listadas a seguir:

- Elaboração de um modelo de custo para um sistema *Mobile Edge Computing*, composto de dispositivos móveis, geradores de tarefas, interagindo com servidores locais e com a *Cloud*. O modelo proposto permitiu ganhos energéticos e tempos totais reduzidos conforme os parâmetros estabelecidos no modelo, as características das aplicações e as caraterísticas dos dispositivos da rede para utilização da técnica de DVFS. **A principal contribuição do modelo de custo proposto em relação aos seus pares do estado da arte refere-se à utilização de custos não contabilizados por outros modelos, como os custos com transmissões de dados, tempos gastos em fila de espera, controle do nível de bateria dos dispositivos IoT, definição de dois perfil de tarefas, críticas com *deadline* e não críticas sem *deadline*, atenuação da taxa de transmissões de dados devido ao congestionamento da rede e a interação com a *Cloud***.
- Desenvolvimento de um algoritmo heurístico que implementa o modelo de custo, oferecendo complexidade reduzida. O algoritmo é utilizado para o escalonamento das tarefas do sistema e proveu soluções que permitiram reduzir o gasto energético e os tempos de processamento das tarefas.
- Mapeamento do estado da arte de modelos de custo utilizados em ambientes *Mobile Edge Computing* e *Fog Computing* para o descarregamento de tarefas e minimização do consumo energético do sistema.

8.1.1 Contribuições adicionais

Como contribuições adicionais à proposição principal da questão de pesquisa deste trabalho, pode-se citar:

- O esclarecimento de definições existentes na literatura referentes às arquiteturas de *Edge Computing*, *Fog Computing*, *Mobile Edge Computing* e *Cloudlet Computing* que recorrentemente são citadas de modo intercalado para se referir à mesma arquitetura, causando imensa confusão. A definição conceitual de cada arquitetura permitiu maior clareza sobre qual arquitetura seria utilizada neste trabalho e quais os principais problemas que teriam de ser estudados e considerados durante a elaboração do modelo de custo do sistema.
- Realização de busca e esclarecimento sobre a definição conceitual de *Green Computing* que, assim como para as arquiteturas de sistema, possui diferentes usos na literatura consultada. A definição utilizada aqui foi concernente a arquiteturas que realizam esforços para reduzir consumo energético, diminuindo os impactos ambientais decorrentes da produção de energia elétrica, como a emissão de CO_2 na atmosfera (HARMON; AUSEKLIS, 2009) (ROSSI et al., 2014).
- Realizado estudo detalhado sobre o consumo de energia elétrica em CPUs, complementado com noções básicas de eletricidade e descrição sobre as componentes de potência que compõem a potência total dissipada por uma CPU quando realiza processamento.
- Definição de uma revisão sistemática para pesquisa de artigos em duas etapas com descrição passo-a-passo das ações e decisões que moveram as escolhas dos artigos até encontrar um conjunto de artigos pertinentes para análise comparativa. Dessa análise surgiu a base para a definição das caraterísticas do modelo do sistema.
- Desenvolvimento de um simulador focado para a arquitetura em três camadas de *Mobile Edge Computing* utilizada neste trabalho. A implementação engloba as tecnologias de comunicação 5G, para comunicação dos dispositivos móveis com os servidores locais, e fibra óptica, para comunicação entre os servidores locais e a *Cloud*. O simulador possui suporte para o escalonador de tarefas que implementa o modelo de custo do sistema e a execução dos experimentos tomou como base as descrições das aplicações veiculares definidas no Capítulo 7.

8.2 Trabalhos Futuros

Embora os resultados apresentados tenham cumprido os objetivos previamente estabelecidos e respondido a questão de pesquisa, percebe-se a oportunidade de realizar

melhorias adicionais na pesquisa realizada. Como trabalhos futuros pode-se citar:

- O aperfeiçoamento do modelo de custo do sistema, com a inserção de novas variáveis, além do consumo energético e do tempo decorrido, tais como custo financeiro para utilização de serviços de *Cloud* ou de serviços dos servidores MEC. A alocação de recursos nesses equipamentos, comercialmente falando, incorre em custos financeiros, visto que o uso de serviços de processamento nessas plataformas ocorre através de um modelo *pay-per-use*. Além disso, poderia ser adicionado ao modelo o custo financeiro associado à utilização de armazenamento nos nodos da rede. Para armazenamento no próprio dispositivo o custo financeiro seria maior, visto que nesses equipamentos o custo por *GB* é alto, por utilizarem memórias de estado sólido, enquanto nos servidores MEC e na *Cloud* há a possibilidade de usar discos rígidos com custo por *GB* muitíssimo inferior. As possibilidades não se limitam a essas, podendo haver muitas mais variáveis, tornando o modelo cada vez mais completo.
- O aperfeiçoamento do simulador do sistema, de modo a permitir maior flexibilidade para a inserção de outros algoritmos de escalonamento e outras arquiteturas. Por ora, o simulador é limitado à proposta deste trabalho, com a arquitetura aqui definida e para o descarregamento de tarefas utilizando o algoritmo de escalonamento desenvolvido.
- A realização de novos experimentos utilizando o modelo de custo aprimorado, com mais variáveis de custo e com um escalonador para o sistema adaptado ao novo modelo. Mais aplicações podem ser exploradas, em cenários da indústria, *healthcare*, aviação, mineração, dentre outros.

REFERÊNCIAS

5G-AMERICAS. *New Services & Applications With 5G Ultra-reliable Low Latency Communications*. [S.l.], 2018.

AL-FUQAHA, A. et al. Internet of things: A survey on enabling technologies, protocols, and applications. **IEEE Communications Surveys Tutorials**, v. 17, n. 4, p. 2347–2376, Fourthquarter 2015. ISSN 2373-745X.

ARMY, U. S. D. of the. **Army ADP Review**. [S.l.]: Department of Defense, Department of the Army., 1977. (DA pam, Nº 9).

ARXIV.ORG, C. U. L. **Cornel University Library - arXiv.org**. 2019. Disponível em: <https://arxiv.org/>. Acessado em: 08/11/2019.

ATMEL. **ATmega/V-2560 Datasheet**. [S.l.], 2014.

BACCARELLI, E. et al. Energy-efficient dynamic traffic offloading and reconfiguration of networked data centers for big data stream mobile computing: review, challenges, and a case study. **IEEE Network**, v. 30, n. 2, p. 54–61, March 2016. ISSN 1558-156X.

BANGUI, H. et al. Moving to the edge-cloud-of-things: Recent advances and future research directions. In: . [S.l.: s.n.], 2018.

BASSOLI, R.; RENZO, M. D.; GRANELLI, F. Analytical energy-efficient planning of 5g cloud radio access network. In: **2017 IEEE International Conference on Communications (ICC)**. [S.l.: s.n.], 2017. p. 1–4. ISSN 1938-1883.

BONOMI, F. et al. Fog computing: A platform for internet of things and analytics. In: **Big Data and Internet of Things**. [S.l.: s.n.], 2014.

BROGI, A.; FORTI, S.; IBRAHIM, A. Deploying fog applications: How much does it cost, by the way? In: **CLOSER**. [S.l.: s.n.], 2018.

BURD, T.; BRODERSEN, R. Processor design for portable systems. **Journal of VLSI Signal Processing**, v. 13, 11 1996.

CAPUCHO, J. H. L.; RESENDO, L. C. Ilp model and effective genetic algorithm for routing and spectrum allocation in elastic optical networks. In: **2013 SBMO/IEEE MTT-S International Microwave Optoelectronics Conference (IMOC)**. [S.l.: s.n.], 2013. p. 1–5.

CARLINI, S. **Massive 5G Electricity Costs are in Focus Ahead of the Global Build-out at the Edge**. 2019. Disponível em: <https://blog.se.com/co-location/2019/11/11/massive-5g-electricity-costs-are-in-focus-ahead-of-the-global-build-out-at-the-edge/>. Acessado em: 07/01/2020.

CARROLL, A.; HEISER, G. An analysis of power consumption in a smartphone. In: . [S.l.: s.n.], 2010.

CHEN, Y.-L. et al. Learning-directed dynamic voltage and frequency scaling scheme with adjustable performance for single-core and multi-core embedded and mobile systems. **Sensors**, MDPI AG, v. 18, n. 9, p. 3068, Sep 2018. ISSN 1424-8220.

CLARK, R. **Operators Starting to Face Up to 5G Power Cost**. 2019. Disponível em: <https://www.lightreading.com/asia-pacific/operators-starting-to-face-up-to-5g-power-cost-/d/d-id/755255>. Acessado em: 07/01/2020.

COMER, D. E. **Computer Networks and Internets**. 6. ed. [S.l.]: Pearson Education Limited, 2015. ISBN 978- 0-13-358793-7.

CONCEPTDRAW. **How Cloud Computing Works**. 2019. Disponível em: <https://www.conceptdraw.com/How-To-Guide/cloud-computing-architecture>. Acessado em: 10/11/2019.

DINH, H. et al. A survey of mobile cloud computing: Architecture, applications, and approaches. **Wireless Communications and Mobile Computing**, v. 13, 12 2013.

DINH, T. Q. et al. Offloading in mobile edge computing: Task allocation and computational frequency scaling. **IEEE Transactions on Communications**, v. 65, n. 8, p. 3571–3584, Aug 2017. ISSN 1558-0857.

DOLUI, K.; DATTA, S. K. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In: **2017 Global Internet of Things Summit (GIoTS)**. [S.l.: s.n.], 2017. p. 1–6.

EECS. **Featured Research Article - Green Computing: Higher Energy Efficiency from Silicon to the Cloud**. [S.l.]: EECS, 2012. Disponível em: <http://www.eecs.umich.edu/eecs/about/articles/2010/green-computing.pdf>. Acessado em: 10/11/2019.

FOUNDATION, R. P. **Raspberry Pi 4 Model B**. 2019. Disponível em: <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/>. Acessado em: 10/08/2019.

FRENGER, P.; TANO, R. **A technical look at 5G energy consumption and performance**. [S.l.], 2019. Disponível em: <https://www.ericsson.com/en/blog/2019/9/energy-consumption-5g-nr>. Acessado em: 07/01/2020.

GEDAWY, H. et al. Awakening the cloud within: Energy-aware task scheduling on edge iot devices. In: **2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)**. [S.l.: s.n.], 2018. p. 191–196. ISSN null.

GHANBARI, S.; OTHMAN, M. A priority based job scheduling algorithm in cloud computing. **Procedia Engineering**, v. 50, p. 778–785, 01 2012.

GUPTA, A.; JHA, R. K. A survey of 5g network: Architecture and emerging technologies. **IEEE Access**, v. 3, p. 1206–1232, 2015. ISSN 2169-3536.

HAOUARI, F.; FARAJ, R.; ALJA'AM, J. M. Fog computing potentials, applications, and challenges. In: **2018 International Conference on Computer and Applications (ICCA)**. [S.l.: s.n.], 2018. p. 399–406.

HARMON, R. R.; AUSEKLIS, N. Sustainable it services: Assessing the impact of green computing practices. In: **PICMET '09 - 2009 Portland International Conference on Management of Engineering Technology**. [S.l.: s.n.], 2009. p. 1707–1717. ISSN 2159-5119.

HASSAN, N. et al. The role of edge computing in internet of things. **IEEE Communications Magazine**, v. 56, n. 11, p. 110–115, November 2018. ISSN 1558-1896.

HECK, M. et al. Iot applications in fog and edge computing: Where are we and where are we going? In: **2018 27th International Conference on Computer Communication and Networks (ICCCN)**. [S.l.: s.n.], 2018. p. 1–6. ISSN 1095-2055.

INTEL. **Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor (White Paper)**. [S.l.], 2004.

IORGA, M.; FELDMAN, L.; BARTON, R. **SP 800-191. The NIST Definition of Fog Computing**. Gaithersburg, MD, USA, 2011.

JALALI, F. et al. Greening iot with fog: A survey. In: **2017 IEEE International Conference on Edge Computing (EDGE)**. [S.l.: s.n.], 2017. p. 25–31. ISSN null.

JANSSON, J. **Collision Avoidance Theory with Application to Automotive Collision Mitigation**. Thesis (PhD), 06 2005.

JAY, J. A. **Low Signal Latency in Optical Fiber Networks**. [S.l.], 2012.

JENNINGS, B.; STADLER, R. Resource management in clouds: Survey and research challenges. **Journal of Network and Systems Management**, v. 23, 03 2014.

JIN, X.; GOTO, S. Hilbert transform-based workload prediction and dynamic frequency scaling for power-efficient video encoding. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 31, p. 649–661, 05 2012.

JIN, X.; GOTO, S. Hilbert transform-based workload prediction and dynamic frequency scaling for power-efficient video encoding. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 31, n. 5, p. 649–661, May 2012. ISSN 1937-4151.

KAUR, T.; CHANA, I. Energy efficiency techniques in cloud computing: A survey and taxonomy. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 48, n. 2, oct. 2015. ISSN 0360-0300.

KESHAV, S. How to read a paper. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, v. 37, n. 3, p. 83–84, jul 2007. ISSN 0146-4833.

KITANOV, S.; JANEVSKI, T. Energy efficiency of fog computing and networking services in 5g networks. In: **IEEE EUROCON 2017 -17th International Conference on Smart Technologies**. [S.l.: s.n.], 2017. p. 491–494. ISSN null.

KITCHENHAM, B.; CHARTERS, S. **Guidelines for performing Systematic Literature Reviews in Software Engineering**. 2007.

LEE, U.; CHEUNG, R.; GERLA, M. Emerging vehicular applications. **Boca Raton**, 03 2009.

LIU, Y. et al. Thermal vs energy optimization for dvfs-enabled processors in embedded systems. In: **8th International Symposium on Quality Electronic Design (ISQED'07)**. [S.l.: s.n.], 2007. p. 204–209. ISSN 1948-3295.

LOPEZ, P. G. et al. Edge-centric computing: Vision and challenges. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 45, n. 5, p. 37–42, sep. 2015. ISSN 0146-4833. Available from Internet: <https://doi.org/10.1145/2831347.2831354>.

MAHMUD, M.; BUYYA, R. Fog computing: A taxonomy, survey and future directions. In: ____. [S.l.: s.n.], 2016. ISBN 978-981-10-5861-5.

MAHMUD, R.; BUYYA, R. Modelling and simulation of fog and edge computing environments using ifogsim toolkit. **ArXiv**, abs/1812.00994, 2018.

MAO, Y. et al. Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems. **IEEE Transactions on Wireless Communications**, v. 16, n. 9, p. 5994–6009, Sep. 2017. ISSN 1558-2248.

MARVIN, R. **The Big Data Market Is Set to Skyrocket by 2022**. 2020. Disponível em: <https://www.pcmag.com/news/368958/the-big-data-market-is-set-to-skyrocket-by-2022>. Acessado em: 02/01/2020.

MELL, P. M.; GRANCE, T. **SP 800-145. The NIST Definition of Cloud Computing**. Gaithersburg, MD, USA, 2011.

NETSCRIBES. **The role of IoT in the future of the automotive industry**. 2018. Disponível em: <https://www.netscribes.com/the-present-and-future-role-of-automotive-iot/>. Acessado em: 05/01/2020.

NGUYEN, R. M. H.; BROWN, M. S. Raw image reconstruction using a self-contained srgb-jpeg image with only 64 kb overhead. In: **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2016. p. 1655–1663. ISSN 1063-6919.

ORACLE. **Java 6 - sort()**. 2019. Disponível em: <https://docs.oracle.com/javase/6/docs/api/index.html>. Acessado em: 11/08/2019.

PATEL, M.; HU, Y.; HéDé, P. **Mobile-Edge Computing**. [S.l.], 2014. Available from Internet: <https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf>.

PEREIRA, D.; ILIC, A.; SOUSA, L. On boosting energy-efficiency of heterogeneous embedded systems via game theory. In: . [S.l.: s.n.], 2017. p. 19–24.

PHAM, X.-Q.; HUH, E.-N. Towards task scheduling in a cloud-fog computing system. In: **2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)**. [S.l.: s.n.], 2016. p. 1–4.

PRIYA, B.; PILLI, S. E.; JOSHI, R. C. A survey on energy and power consumption models for greener cloud. In: **2013 3rd IEEE International Advance Computing Conference (IACC)**. [S.l.: s.n.], 2013. p. 76–82.

PU, L. et al. Content retrieval at the edge: A social-aware and named data cooperative framework. **IEEE Transactions on Emerging Topics in Computing**, v. 7, n. 1, p. 135–148, Jan 2019. ISSN 2376-4562.

PYTHON. **Python Manual - Timsort**. 2019. Disponível em: <https://docs.python.org/>. Acessado em: 11/08/2019.

RA, M.-R. et al. Odessa: enabling interactive perception applications on mobile devices. In: **MobiSys '11**. [S.l.: s.n.], 2011.

RODRIGUEZ, J. **Fundamentals of 5G Mobile Networks**. [S.l.]: Wiley, 2015. ISBN 1118867521,9781118867525.

ROMERO, J. et al. Gsm, gprs and edge performance: Evolution towards 3g/umts, second edition. In: ____. [S.l.: s.n.], 2004. p. 235 – 305. ISBN 9780470866962.

ROSSI, C.; GAETANI, M.; DEFINA, A. Aurora: an energy efficient public lighting iot system for smart cities. **ACM SIGMETRICS Performance Evaluation Review**, v. 44, p. 76–81, 09 2016.

ROSSI, F. D. et al. Green software development for multi-core architectures. In: **2014 IEEE Symposium on Computers and Communications (ISCC)**. [S.l.: s.n.], 2014. p. 1–6. ISSN 1530-1346.

SAMSUNG. **Smartphone Galaxy S10**. 2019. Disponível em: <https://www.samsung.com/>. Acessado em: 10/08/2019.

SARANGI, S. R.; GOEL, S.; SINGH, B. Energy efficient scheduling in iot networks. In: **Proceedings of the 33rd Annual ACM Symposium on Applied Computing**. New York, NY, USA: Association for Computing Machinery, 2018. p. 733–740. ISBN 9781450351911.

SATYANARAYANAN, M. et al. The case for vm-based cloudlets in mobile computing. **IEEE Pervasive Computing**, v. 8, n. 4, p. 14–23, Oct 2009. ISSN 1558-2590.

SATYANARAYANAN, M. et al. Edge analytics in the internet of things. **IEEE Pervasive Computing**, v. 14, n. 2, p. 24–31, Apr 2015. ISSN 1558-2590.

SEARS, F. W. **Física III - Eletromagnetismo**. 12nd ed.. ed. [S.l.]: Addison-Wesley, 2009. (PRINCIPLES OF PHYSICS II). ISBN 9788588639348.

SELS, V.; GHEYSEN, N.; VANHOUCKE, M. A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions. **International Journal of Production Research**, v. 50, p. 4255–4270, 08 2012.

SELVARANI, S.; SADHASIVAM, G. S. Improved cost-based algorithm for task scheduling in cloud computing. In: **2010 IEEE International Conference on Computational Intelligence and Computing Research**. [S.l.: s.n.], 2010. p. 1–5.

SHI, W. et al. Edge computing: Vision and challenges. **IEEE Internet of Things Journal**, v. 3, n. 5, p. 637–646, Oct 2016. ISSN 2372-2541.

SINGH JAGBEER, P. B.; SINGH, S. P. An algorithm to reduce the time complexity of earliest deadline first scheduling algorithm in real-time system. v. 2, 12 2010.

SKUBIC, B. et al. Energy-efficient next-generation optical access networks. **IEEE Communications Magazine**, v. 50, n. 1, p. 122–127, January 2012. ISSN 1558-1896.

STOJMENOVIC, I. Fog computing: A cloud to the ground support for smart things and machine-to-machine networks. In: **2014 Australasian Telecommunication Networks and Applications Conference (ATNAC)**. [S.l.: s.n.], 2014. p. 117–122.

TALEB, T. et al. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. **IEEE Communications Surveys Tutorials**, v. 19, n. 3, p. 1657–1681, thirdquarter 2017. ISSN 2373-745X.

TANENBAUM, A. S.; AUSTIN, T. **Structured Computer Organization**. 6th. ed. [S.l.]: Prentice Hall, 2012. ISBN 0132916525,9780132916523.

THAKUR, A.; GORAYA, M. S. A taxonomic survey on load balancing in cloud. **Journal of Network and Computer Applications**, v. 98, p. 43 – 57, 2017. ISSN 1084-8045.

VAQUERO, L. M.; RODERO-MERINO, L. Finding your way in the fog: Towards a comprehensive definition of fog computing. **SIGCOMM Comput. Commun. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 44, n. 5, p. 27–32, oct. 2014. ISSN 0146-4833.

VICTOR, B. Emergence of micro datacenter (cloudlets/edges) for mobile computing. In: . [S.l.: s.n.], 2015.

VISUALCAPITALIST. **IIoT - Industrial Internet of Things**. 2019. Disponível em: <visualcapitalist.com>. Acessado em: 13/11/2019.

VOGELEER, K. D. et al. The energy/frequency convexity rule: Modeling and experimental validation on mobile devices. In: . [S.l.: s.n.], 2014.

WADHONKAR, A.; THENG, D. A survey on different scheduling algorithms in cloud computing. In: **2016 2nd International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB)**. [S.l.: s.n.], 2016. p. 665–669.

WAN, J. et al. Fog computing for energy-aware load balancing and scheduling in smart factory. **IEEE Transactions on Industrial Informatics**, v. 14, n. 10, p. 4548–4556, Oct 2018. ISSN 1941-0050.

WANG, C. et al. Energy-efficient offloading policy for resource allocation in distributed mobile edge computing. In: **2018 IEEE Symposium on Computers and Communications (ISCC)**. [S.l.: s.n.], 2018. p. 00366–00372. ISSN 1530-1346.

WINSYSTEMS. **Cloud, Fog And Edge Computing – What's The Difference?** 2019. Disponível em: <https://www.winsystems.com/cloud-fog-and-edge-computing-whats-the-difference/>. Acessado em: 28/04/2019.

WU, H.; LEE, C. Energy efficient scheduling for heterogeneous fog computing architectures. In: **2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)**. [S.l.: s.n.], 2018. v. 01, p. 555–560. ISSN 0730-3157.

YOUSEFPOUR, A. et al. All one needs to know about fog computing and related edge computing paradigms: A complete survey. **Journal of Systems Architecture**, v. 98, p. 289 – 330, 2019. ISSN 1383-7621.

YU, B. et al. Energy efficient scheduling for iot applications with offloading, user association and bs sleeping in ultra dense networks. In: **2018 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)**. [S.l.: s.n.], 2018. p. 1–6.

YU, H.; WANG, Q.; GUO, S. Energy-efficient task offloading and resource scheduling for mobile edge computing. In: **2018 IEEE International Conference on Networking, Architecture and Storage (NAS)**. [S.l.: s.n.], 2018. p. 1–4.

YU, Y. Mobile edge computing towards 5g: Vision, recent progress, and open challenges. **China Communications**, v. 13, n. Supplement2, p. 89–99, N 2016. ISSN 1673-5447.

ZHANG, G. et al. Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices. **IEEE Transactions on Industrial Informatics**, v. 14, n. 10, p. 4642–4655, Oct 2018. ISSN 1941-0050.

ZHANG, Y. et al. **Accurate CPU Power Modeling for Multicore Smartphones**. [S.l.], 2015.