**UNIVERSIDADE FEDERAL DO CEARÁ**

**CENTRO DE CIÊNCIAS**

**DEPARTAMENTO DE COMPUTAÇÃO**

**PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO**

**PEDRO PAIVA ALVES**

# UMA ABORDAGEM SENSÍVEL A CONTEXTO PARA *OFFLOADING* DE PROCESSAMENTO

**FORTALEZA**

**2019**

PEDRO PAIVA ALVES

# UMA ABORDAGEM SENSÍVEL A CONTEXTO PARA *OFFLOADING* DE PROCESSAMENTO

Dissertação apresentada ao Curso de do Programa de Pós-Graduação em Ciências da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciências da Computação. Área de Concentração: Engenharia de Software

Orientador: Fernando Antonio Mota Trinta, DSc

Coorientador: Paulo Antonio Leal Rego, DSc

FORTALEZA

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

---

A481a Alves, Pedro Paiva.
Uma abordagem sensível a contexto para offloading de processamento / Pedro Paiva Alves. – 2019.
80 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2019.
Orientação: Prof. Dr. Fernando Antonio Mota Trinta.
Coorientação: Prof. Dr. Paulo Antonio Leal Rego.

1. Offloading. 2. Mobile Cloudlet. 3. Fog Computing. 4. Multicritérios para análise de decisão. 5. Dispositivo Móvel. I. Título.

CDD 005

---

PEDRO PAIVA ALVES

UMA ABORDAGEM SENSÍVEL A CONTEXTO PARA *OFFLOADING* DE PROCESSAMENTO

Dissertação apresentada ao Curso de  do Programa de Pós-Graduação em Ciências da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciências da Computação.  Área de Concentração: Engenharia de Software

Aprovada em:

BANCA EXAMINADORA

---

Fernando Antonio Mota Trinta, DSc  (Orientador)
Universidade Federal do Ceará (UFC)

---

Paulo Antonio Leal Rego, DSc  (Coorientador)
Universidade Federal do Ceará (UFC)

---

Emanuel Ferreira Coutinho, DSc
Universidade Federal do Ceará – Quixadá (UFC)

Dedico esse trabalho a minha mãe Maria Linete Paiva, que moveu montanhas para garantir a educação de seus dois filhos, a Francisco Alexandre de Paiva Forte, que representa em minha vida um norte a ser seguido e a Davi Paiva Alves, com quem posso contar a qualquer momento.

## AGRADECIMENTOS

Ao Prof. Dr. Fernando Antonio Mota Trinta, por me acolher como seu aluno e me orientar em minha dissertação de mestrado.

Ao Prof. Dr. Paulo Antonio Leal Rego, por me coorientar no mestrado e auxiliar em assuntos técnicos.

Ao Doutorando em Ciências da Computação, Francisco Anderson de Almada Gomes, por me auxiliar diversas vezes no trabalho de mestrado.

Aos amigos mestrandos Pedro Teixeira de Araújo, Adriano Lima Cândido e Máquison Felipe Ribeiro de Castro, que durante toda o período do mestrado estiveram disponíveis para troca de ideias e ajuda na realização de experimentos.

A todos os meus professores que foram responsáveis por minha educação e que me mostraram o poder do conhecimento.

A minha namorada Nathália Gonçalves de Oliveira, que sempre me apoiou e me estimulou aos estudos, mesmo em finais de semanas.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

“Não basta saber, é preferível saber aplicar. Não é o bastante querer, é preciso saber querer.”

(Johann Goethe)

## RESUMO

A evolução das tecnologias de processamento e da capacidade energética de dispositivos móveis foi bastante acelerada nos últimos anos. Porém, as aplicações para tais dispositivos estão seguindo essa evolução, exigindo cada vez mais poder de processamento e consumo energético. Nessas circunstâncias, abordagens que auxiliam no processamento dos aplicativos tornaram-se necessárias. Uma delas é a abordagem da utilização da infraestrutura de *Fog Computing*, que auxilia dispositivos com baixa capacidade de processamento na execução de tarefas que seriam computacionalmente custosas e/ou que necessitariam de um longo período de tempo para serem concluídas, evitando a alta latência da troca de informações na rede. Para esse fim, um dos processos mais utilizados é o conhecido como *offloading* computacional, onde o processamento dos métodos utilizados pelos aplicativos é enviado para outros dispositivos com poder superior de processamento ou capacidade energética elevada. A delegação dessas tarefas pode ser feita a uma variedade de ambientes remotos de execução (do inglês, *remote execution environment* - REE), que possuem múltiplas características. Nesse contexto, a escolha do REE é um processo complexo, pois a heterogeneidade dos dispositivos interfere no tempo de conclusão das tarefas migradas e a dinamicidade dos mesmos pode inviabilizar a execução da atividade de *offloading*. Neste trabalho, propomos um método para selecionar um REE que atenda a uma solicitação de *offloading* de métodos, considerando o contexto de cada dispositivo e os critérios que impactam diretamente na velocidade de conclusão da tarefa delegada de modo a selecionar o REE mais adequado. Por fim, apresentamos uma infraestrutura funcional para permitir o *offloading* de métodos de dispositivos móveis para outros dispositivos usando a abordagem proposta e mostramos sua eficácia através de experimentos.



**Palavras-chave:** Offloading. Dispositivo Móvel. Multicritérios para análise de decisão. Mobile Cloudlet. Fog Computing.

## ABSTRACT

The evolution of the processing and energy technologies of mobile devices has been quite accelerated in recent years. However, applications are following such evolution, requiring more processing power and energy capacity. Under these circumstances, approaches that aid in the processing of the applications become necessary. One of those is the approach of using the Fog Computing infrastructure, to speed up the execution of tasks on devices with low processing capacity and to avoid high latency of information exchange in the network. For this purpose, a process knew as offloading is used, where the processing of methods used by the applications is sent to other devices with the superior power of processing or energetic capacity. The delegation of these tasks could be done to a variety of remote execution environments (REE)s that have multiples characteristics that interfer with task completion speed. In this context, a decision method is needed to select where to perform offloading, once these REEs are heterogeneous and are dinamiccally connecting and desconnecting of the network. In this work, we propose a method to decide the best device to perform the offloading method, considering multiple criteria of these devices and their variance according to time, and present a functional infrastructure to enable the offloading of tasks by devices to others devices using the method proposed.



**Keywords:** Offloading. Mobile Devices. Multiple-criteria decision analysis. Mobile Cloudlet. Fog Computing

## LISTA DE FIGURAS

## LISTA DE ALGORITMOS

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| REE | *Remote Execution Environment* |
| MCC | *Mobile Cloud Computing* |
| UDP | *User Datagram Protocol* |
| WLAN | *Wireless Local Network* |
| SHA-1 | *Secure Hash Algorithm 1* |
| API | *Application Programming Interface* |
| ASAP | *As Soon As Possible* |
| RSSI | *Received Signal Strength Indicator* |
| RTT | *Round Time Trip* |
| IoT | *Internet of Things* |
| AHP | *Analytic Hierarchy Process* |
| TOPSIS | *Technique for Order Preference by Similarity to Ideal Solution* |
| IP | *Internet Protocol* |
| GPU | *Graphics Processing Unit* |

**SUMÁRIO**

# 1 INTRODUÇÃO

Em 2018, as tecnologias e serviços móveis geraram 4,6% do PIB mundial, totalizando 3,9 trilhões de dólares em valor econômico agregado, com a existência de mais de 5,13 bilhões de pessoas com dispositivos móveis em todo o mundo, significando 66,53% da população mundial (INTELLIGENCE, 2019). Esse crescente número de dispositivos móveis trouxe consigo um novo mundo de comércio eletrônico nas quais as mais diversas aplicações são vendidas ou são distribuídas sem custo ao usuário.

Até o segundo trimestre de 2019, o número de aplicativos para Android disponíveis na *Play Store* estava em torno de 2,46 milhões de aplicativos (STATISTA, 2019). De forma a atrair o usuário, tais aplicações procuram prover serviços que executem em tempo hábil e que contemplem o maior número de dispositivos o possível, conquistando maior público por consequência.

Apesar da evolução dos dispositivos aos longo das últimas décadas, o surgimento de novas aplicações como as que utilizam realidade aumentada, processamento de dados contextuais, dentre outras, tem demandado maiores capacidades de processamento e armazenamento dos *smartphones*. Com isso, certos dispositivos enfrentam dificuldades para executar tais aplicações, gerando uma baixa qualidade na experiência do usuário. Em paralelo, a execução de certas tarefas pode levar ao consumo exacerbado de energia, diminuindo a autonomia de funcionamento do dispositivo móvel. Para atenuar esta questão, uma abordagem que tem sido bastante investigada é a de integração da computação móvel com serviços de *Cloud Computing*.

A *Cloud Computing*, ou Computação em Nuvem, representa um modelo de entrega de serviços sob demanda a um conjunto de recursos computacionais configuráveis que podem ser rapidamente provisionados e liberados de forma automatizada (MELL *et al.*, 2011). Na nuvem, o uso extensivo de virtualização permite a elasticidade de serviços e aplicações, passando a "ilusão" de recursos infinitos para atender a demanda crescente de clientes. Mais especificamente, quando aplicado ao contexto de dispositivos móveis para oferecer um poder computacional atrativo para aplicações móveis com capacidade limitada, este cenário é comumente referenciado na literatura como *Mobile Cloud Computing*. Nesse contexto, uma técnica bastante utilizada é o *offloading* computacional, na qual a estratégia para auxiliar os dispositivos se baseia em demandar atividades que seriam computacionalmente custosas (exigindo um longo período de processamento ou consumo energético) para outros dispositivos com maior poder computacional, maior capacidade de armazenamento ou maior autonomia energética (AKHERFI *et al.*, 2018).

Uma problemática na realização de *offloading* em um cenário de *Cloud Computing* voltado para dispositivos móveis para algumas aplicações, especialmente aquelas sensíveis a latência, é o atraso decorrente da comunicação entre dispositivos móveis e recursos virtualizados em provedores de nuvem pública. Nestas aplicações incluem-se aplicações de tempo-real restrito como os jogos multiusuário, processamento de linguagem natural e realidade aumentada. Para contornar tais limitações, uma solução é trazer os serviços de *offloading* de dados e processamento para mais próximo dos dispositivos cliente. Esta abordagem criou vários novos termos como *Fog Computing*, *Edge Computing*, dentre outros (YOUSEFPOUR *et al.*, 2019). Estas visões possuem peculiaridades entre si, mas todas se assemelham em propor a aproximação entre serviços e dispositivos na borda da rede.

Em um possível cenário de *Edge Computing*, pode haver mais de um possível destino para migração de uma tarefa de *offloading*. Com isso, uma decisão não trivial é a escolha de qual o melhor *Remote Execution Environment* (REE) para atender à solicitação de *offloading* de um ou vários clientes. A dificuldade por trás desta escolha está relacionada ao objetivo da realização do *offloading*. Enquanto para uns o melhor poderia ser o local mais eficiente, para outros o mais adequado seria o local mais confiável (sendo a confiabilidade relacionada a garantia de que a execução seria realizada por completo, sem falhas ou abandono da atividade).

Portanto, em um cenário com múltiplas possibilidades de *offloading*, é relevante a questão de "como escolher qual melhor local para migrar o processamento de uma tarefa?".

Esta pergunta deriva outros questionamentos secundários, como:

1. Como realizar a comparação entre vários dispositivos heterogêneos de forma a acertar na escolha?
2. Como monitorar o contexto de tais dispositivos de modo a adaptar o processo de escolha para atender aos objetivos da realização de *offloading*?

A partir destas perguntas, este trabalho propõe **um algoritmo de seleção para escolha do melhor local de *offloading* de acordo com objetivos previamente definidos, e uma arquitetura de software que dá suporte a este algoritmo.**

## 1.1 Metodologia

A metodologia utilizada para a criação desse trabalho de dissertação pode ser apresentada de acordo com os seguintes passos:

1. Primeiramente foi realizado um estudo sobre a literatura disponível a respeito da temática

de *Cloud Computing*, *Fog Computing* e *Mobile Cloud Computing* com a utilização da abordagem de *offloading* nas ferramentas de pesquisa Google Acadêmico[1] e Scopus[2]. Dentre os encontrados, focou-se nos que abordavam a melhoria de velocidade de processamento e economia de energia dos dispositivos móveis. Durante os estudos, foi criado um interesse pelo tema de *Fog Computing* com foco na seleção de dispositivos móveis para requisição e atendimento de processos de *offloading*. Desta forma, foi verificado o estado da arte no qual se encontram as soluções e desafios relacionados a temática.

2. O segundo passo foi a busca por trabalhos relacionados a temática de interesse que propunham soluções para as problemáticas de heterogeneidade e dinamicidade na seleção de dispositivos móveis para o atendimento a requisições de *offloading*. Também foram pesquisados trabalhos que viabilizassem a implementação funcional de uma infraestrutura para realizar *offloading* entre dispositivos móveis.
3. Tendo realizado os dois passos anteriores e adquirido conhecimento crítico sobre o tema, foi elaborado um algoritmo de seleção otimizada de dispositivos para atender a requisições de *offloading* em um ambiente com dispositivos dinâmicos e heterogêneos, além de considerar critérios de decisão em relação ao auxílio na velocidade de processamento e no balanceamento de consumo energético dos dispositivos.
4. Com base nos projetos CAOS e CAOS D2D que serão apresentados neste trabalho, foi iniciada a implementação da abordagem que será sugerida nesta dissertação para criar uma infraestrutura funcional que permita o *offloading* entre dispositivos móveis e desta forma testar o algoritmo criado.
5. Por fim, foram realizados experimentos com a infraestrutura criada e a coleta dos dados para análise e verificação de se a abordagem proposta é eficaz no que condiz com a seleção de dispositivos móveis para atendimento de requisições de *offloading*.

## 1.2 Organização da Dissertação

Esta dissertação está organizada da seguinte maneira: o atual capítulo realiza uma contextualização da temática abordada, relacionando os assuntos que serão apresentados e resumindo a problemática existente e a solução proposta por este trabalho.

No segundo capítulo é realizada uma fundamentação teórica sobre os temas relacio-

[1] https://scholar.google.com.br/
[2] https://www.scopus.com/

nados com o trabalho, demonstrando a utilização do processo de *offloading* com infraestruturas de *Cloud Computing*, *Fog Computing* e *Mobile Cloud Computing*. Logo após, são abordadas os desafios presentes para a seleção de dispositivos móveis para atendimento de requisições de *offloading*. E por fim, as infraestruturas dos projetos CAOS e CAOS D2D, que foram o alicerce para a construção deste trabalho.

No terceiro capítulo são apresentados os trabalhos relacionados. São demonstrados como tais trabalhos procuram resolver as problemáticas de seleção de dispositivos móveis para atendimento de requisições de *offloading* considerando a heterogeneidade e a dinamicidade de tais dispositivos.

O quarto capítulo apresenta a abordagem proposta por este trabalho para solucionar os desafios elencados no capítulo dois, descrevendo as ideias que foram aplicadas para a criação de um algoritmo de seleção e demonstrando o funcionamento da infraestrutura criada neste trabalho.

O quinto capítulo demonstra os experimentos realizados com a infraestrutura criada para verificar a eficácia do algoritmo proposto.

Por fim, o sexto capítulo realiza as considerações finais sobre o trabalho e elenca as limitações enfrentadas e os trabalhos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são abordados tópicos que se relacionam com o presente estudo de mestrado, apresentando definições de temáticas importantes para seu entendimento, e os desafios ainda existentes.

## 2.1 *Cloud Computing*

*Cloud Computing* é um modelo ubíquo para provisionar armazenamento, processamento e serviços de forma dinâmica a outros dispositivos (MELL *et al.*, 2011). Esse paradigma se concretizou em torno da primeira década do novo milênio, com a ideia de que os serviços poderiam ser realizados de forma mais eficiente se providos por grandes infraestruturas acessadas via internet (MARINESCU, 2017). Dessa forma, tornou-se possível disponibilizar serviços a dispositivos com baixo poder de processamento/armazenamento no qual o interesse desses dispositivos é o resultado final para o usuário.

Para a implantação de uma infraestrutura que atenda aos serviços dos usuários são necessários diversos investimentos. Por exemplo, é preciso um ambiente para a implementação da infraestrutura com salas climatizadas, conexões de rede, aquisição de servidores, modens, etc., além de serem necessários investimentos na manutenção de tal infraestrutura. Já o paradigma de *Cloud Computing* realiza os processamentos e armazenamentos de dados na "nuvem", uma metáfora que significa a abstração do local onde estes dados serão processados. Desta forma, a implantação da infraestrutura fica a cargo de outra empresa livrando o usuário de tais custos para atender aos seus serviços, podendo-se utilizar tal serviço sob demanda em qualquer tempo e em qualquer lugar (MELL *et al.*, 2011).

A infraestrutura básica de comunicação da *Cloud Computing* funciona através da comunicação via Internet entre o servidor armazenado remotamente e o dispositivo do usuário. Devido ao desconhecimento de onde está presente tal servidor, algumas aplicações podem ter sua qualidade de serviço prejudicada devido a distância da aplicação ao servidor em termos de topologia de rede (MACH; BECVAR, 2017). Aplicações como jogos em tempo real, aplicações que utilizam realidade aumentada e aplicações com transmissão em tempo real são exemplos de aplicações sensíveis a latência de transmissão de dados para a utilização de *Cloud Computing* (YI *et al.*, 2015).

Devido a essas limitações, o provisionamento de suporte ao processamento e ar-

mazenamento para as aplicações foi movido para mais próximo das mesmas, sendo este novo paradigma chamado de *Fog Computing*.

### *2.1.1 Fog Computing*

A *Fog Computing* aproxima a execução do processamento e armazenamento de dados ao usuário, com o objetivo de prover às aplicações serviços com baixa latência, alta largura de banda e a informação para o usuário onde seus dados estão sendo executados/armazenados. Dessa forma, essa abordagem foi batizada de *Fog Computing*, pois faz analogia a uma nuvem que está mais próxima ao usuário (YI *et al.*, 2015).

Apesar da similaridade com a *Cloud Computing*, Bonomi *et al.* (2012) esclarece que a *Fog Computing* não é uma extensão trivial da *Cloud Computing* através dos seguintes tópicos que demonstram características próprias da *Fog Computing*:

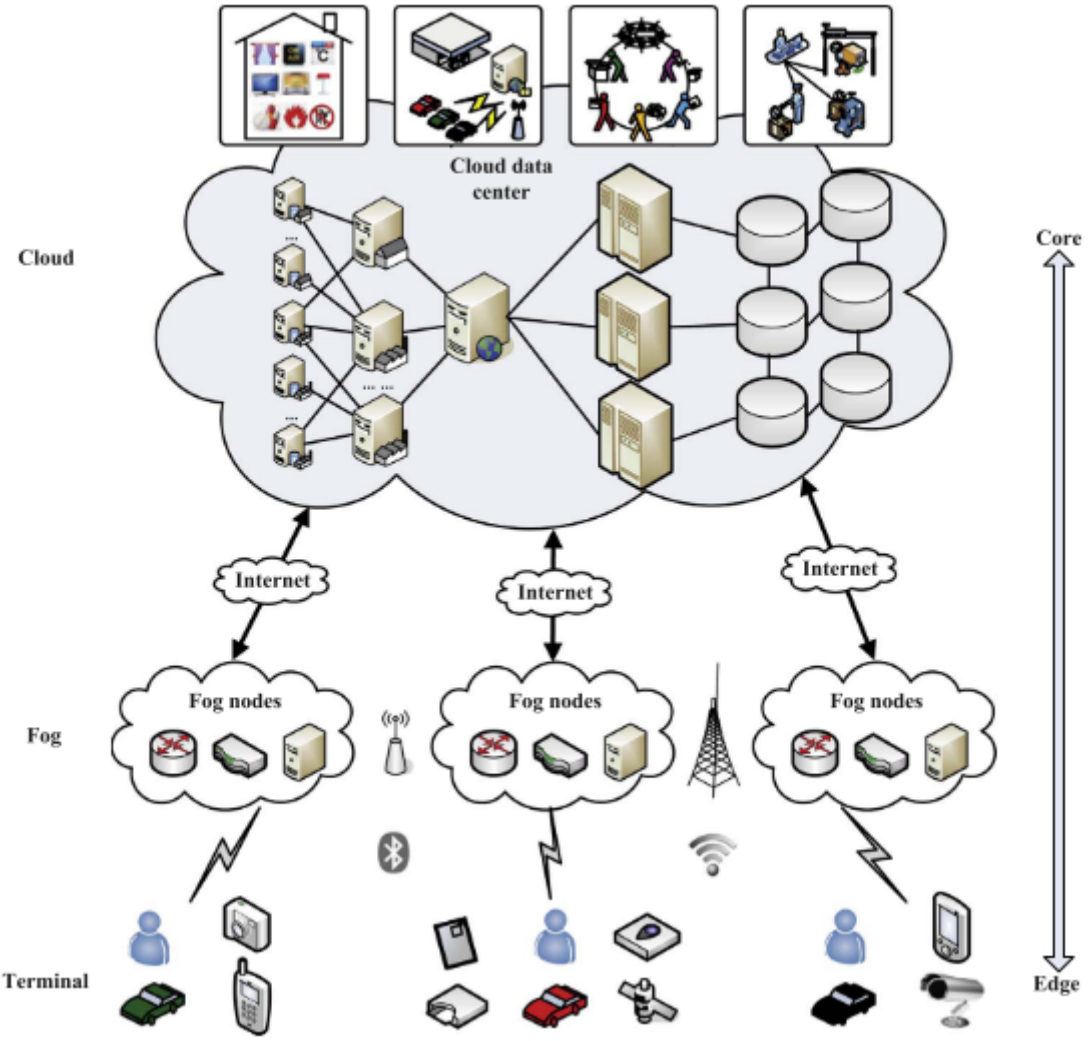
- Localização na "borda da rede", significando a proximidade ao usuário. A *Fog Computing* é distribuída geograficamente ao contrário da *Cloud Computing* que é mais centralizada. Desta forma, a *Fog Computing* provê alta qualidade de serviço através de pontos de acessos distribuídos;
- Suporte a mobilidade. É essencial a *Fog Computing* se comunicar diretamente com os dispositivos móveis e portanto dar suporte a técnicas de mobilidade;
- Interações em tempo real, ao invés de processamento em lote;
- Predominância do acesso sem fio;
- Heterogeneidade, uma vez que os nós da *Fog Computing* podem possuir diferentes hardwares e softwares, além de ser possível serem implantados em diferentes ambientes.

A utilização do paradigma de *Fog Computing* foi fortemente incentivada pelo surgimento da Internet das Coisas, paradigma no qual diversos dispositivos físicos são habilitados para ver, ouvir, pensar e executar tarefas através do compartilhamento de informações e tomada de decisões coordenadas (AL-FUQAHA *et al.*, 2015). O volume e variedade de dados gerados pela Internet das Coisas e o tempo decorrido para processar os dados pode tornar o modelo de *Cloud Computing* não eficiente para algumas aplicações (CISCO, 2015).

Apesar da *Fog Computing* poder substituir a *Cloud Computing* para algumas aplicações, isso não significa que as duas abordagens não possam ser utilizadas em conjunto, pelo contrário, muitas arquiteturas utilizam as duas abordagens como demonstrado na Figura 1. Nessa figura podemos observar que a *Fog* faz um intermédio entre os dispositivos e a *Cloud*. Geral-

mente uma arquitetura que utiliza as duas infraestruturas executa os processos que necessitam de respostas rápidas na *Fog* e deixam para a *Cloud* aqueles processos que são mais demorados ou que precisam de mais dados para serem executados, assim como processos de *Data Mining* (BONOMI *et al.*, 2012).

Figura 1 – Visão geral da arquitetura de *Cloud Computing* e *Fog Computing*




Fonte: Hu et al., 2017

Um ramo em que as infraestruturas de *Cloud* e *Fog Computing* são bastante utilizadas é o de aplicações móveis. Sendo tão grande as possibilidades que surgiu uma área de estudo chamada de *Mobile Cloud Computing*.

### *2.1.2 Mobile Cloud Computing*

O crescente número de aplicações que requerem grande poder de processamento em curtos períodos de tempo, aliado com a capacidade limitada de processamento dos dispositivos móveis, e o consumo energético elevado dessas aplicações, motivaram a utilização da *Cloud Computing* para dispositivos móveis (MACH; BECVAR, 2017).

O paradigma de *Mobile Cloud Computing* (MCC) combina os benefícios da computação móvel com os de *Cloud Computing*, principalmente no quesito de auxiliar nas limitações dos dispositivos móveis por razão de seus poucos recursos para realizar processamentos complexos, permitindo que tais processamentos sejam entregues em forma de serviço (NOOR *et al.*, 2018).

Com o foco em auxiliar os dispositivos móveis em seus processamentos/armazenamentos de dados, foram criadas diversas abordagens para prover tais serviços, utilizando-se tanto das abordagens de *Cloud Computing* quanto de *Fog Computing*. Entretanto, muitas vezes se torna custosa a criação de infraestruturas computacionais para dar suporte as aplicações. Por conseguinte, uma outra abordagem utilizada que possui o mesmo objetivo de auxiliar os dispositivos móveis é a de utilizar REEs que podem ser inclusive outros dispositivos móveis.

Através do uso oportunístico de dispositivos móveis para o auxílio na execução de uma atividade, é possível a implementação de uma abordagem colaborativa para obter maior poder de computação para os dispositivos através da utilização dos demais dispositivos que estão presentes no ambiente, sendo esse ambiente de suporte às aplicações extremamente mutável, uma vez que é formado pelos dispositivos que estão dinamicamente se conectando e desconectando da rede, estando entre estes dispositivos os *smartphones* que possuem capacidade de processamento subutilizada, com uma porcentagem de somente 25% de uso por hora (VARGHESE; BUYYA, 2018).

Para a realização do auxílio de um dispositivo por outro, uma abordagem bastante utilizada é a de transferir a execução de atividades de um dispositivo com capacidade inferior para outro de capacidade superior, sendo essa abordagem conhecida como *Offloading*.

### *2.1.3 Offloading*

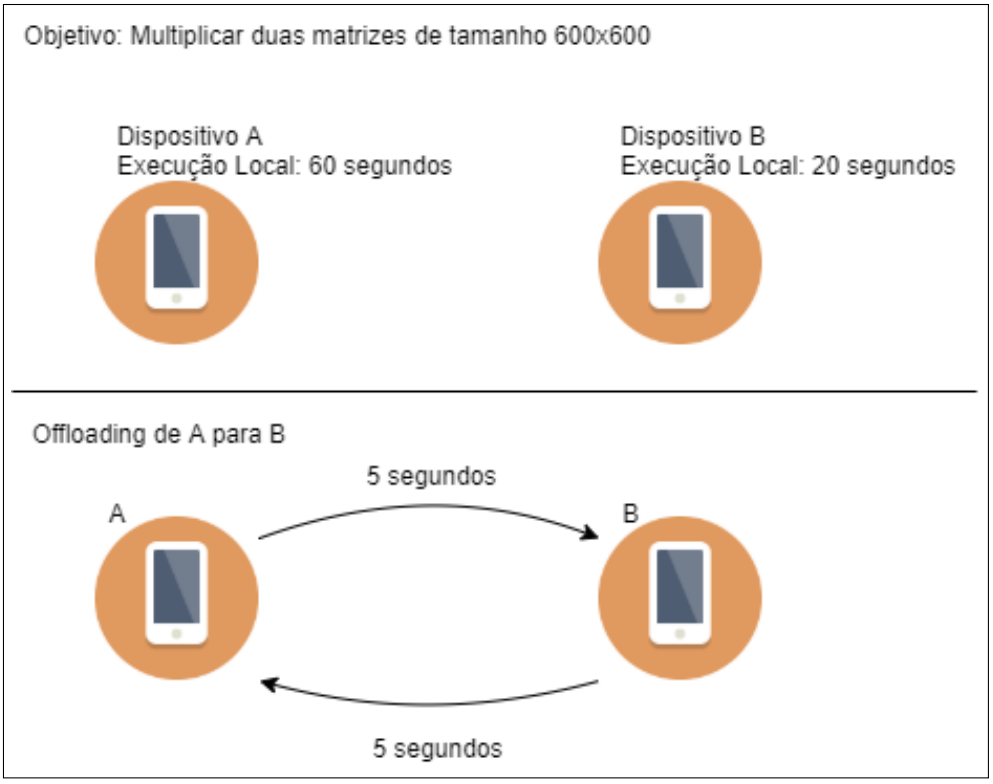
O *offloading* computacional auxilia dispositivos móveis em suas limitações de processamento de informações, armazenamento de dados, e economia de energia através através da delegação dessas atividades para outro dispositivo de maior poder computacional (AKHERFI *et*

*al.*, 2018).

Portanto, a estratégia adotada quando se utiliza uma abordagem de *offloading* para a economia de energia é a realização de processamentos que demandam do dispositivo um gasto energético elevado em outro dispositivo que não possua restrições energéticas ou que possua um valor energético superior. A utilização dessa abordagem pode, por exemplo, aumentar o tempo de duração de bateria de um dispositivo móvel ao executar certa aplicação.

Já com relação a melhoria de qualidade de serviço de uma aplicação, a estratégia de *offloading* auxilia ao transferir execuções de processamentos que levariam muito tempo para serem finalizados pelo dispositivo (ou que não seriam possíveis de serem realizados) para outro dispositivo de poder computacional superior. Por exemplo, observe a Figura 2 em que dois dispositivos A e B executam localmente a tarefa de multiplicação de duas matrizes de tamanho 600x600 em 60 segundos e 20 segundos, respectivamente. Com a execução do método de *offloading* para auxiliar o dispositivo A nesta tarefa utilizando o dispositivo B, que possui capacidade de processamento superior, é possível entregar ao dispositivo A o resultado da resposta em 30 segundos (20 segundos do processamento de B mais, por exemplo, 10 segundos do tempo de transferência), desta forma A obteria o resultado da atividade em metade do tempo que levaria para executar localmente.

Figura 2 – Exemplo de *offloading* para obter melhoria na velocidade de processamento




Fonte: o autor.

Contudo, a execução de *offloading* pode não ser beneficial ao dispositivo se não forem corretamente observados uma série de fatores como qualidade de conexão com a rede, taxa de transferência, e volume de dados para realização do processo.

### *2.1.4 Cenários futuros e Desafios*

Devido a grande popularidade dos *smartphones* e a crescente melhoria na qualidade das conexões, está disponível uma quantidade enorme de recursos computacionais para serem utilizados para melhoria do poder de processamento de aplicações com baixa latência em espaços públicos e sistemas de transporte (VARGHESE; BUYYA, 2018). Entretanto, para usufruir desse poder ainda é preciso solucionar uma série de desafios presentes.

O estudo realizado por Mach e Becvar (2017) elenca três principais desafios que precisam ser cuidadosamente analisados de forma a poder realizar uma operação de *offloading* que traga benefícios ao usuário. O primeiro deles é referente a quando se deve realizar o *offloading* em termos de consumo energético e tempo de execução, uma vez que o envio de uma solicitação de *offloading* pode gera desperdícios, tanto de tempo, como energético de todos os dispositivos envolvidos se o processo de *offloading* levar mais tempo para ser finalizado do que a execução local. O segundo desafio elencado é o de alocação eficiente do processo de *offloading* para um local adequado que consiga atender a requisição de modo a minimizar o tempo de execução, caso a solicitação tenha foco em velocidade de processamento, ou que consiga realizar o balanceamento de carga das requisições para os dispositivos. E, por fim, um terceiro desafio diz respeito a garantia da continuidade do serviço através da preocupação com a mobilidade dos dispositivos.

Este trabalho propõe uma solução para abordar o segundo e o terceiro desafio elencados no parágrafo anterior, através da construção de um método de decisão que selecione o melhor dispositivo para atender a uma requisição de *offloading*, considerando a mobilidade dos mesmos. Com relação ao primeiro desafio o estudo realizado por Rego *et al.* (2017) apresenta uma estratégia bastante interessante para a realização da tomada de decisão de quando se deve ou não ser executado um processo de *offloading* através da construção de uma árvore de decisão com métricas monitoradas do dispositivo.

A seguir são apresentadas duas infraestruturas que realizam o processamento de *offloading* para um *cloudlet* fixo e entre dispositivos, respectivamente. Tais infraestruturas foram utilizadas como passo inicial para o desenvolvimento do presente trabalho.

### *2.1.5 CAOS*

O *framework* CAOS é uma solução para o desenvolvimento de aplicações sensíveis ao contexto utilizando a plataforma Android, provendo mecanismos de *offloading* de dados contextuais e de processamento para um ambiente em nuvem (GOMES *et al.*, 2017). O CAOS, acrônimo de *Context Acquisition and Offloading System*, teve sua origem a partir de duas outras soluções: o MPoS (*Multiplatform Offloading System*) (COSTA *et al.*, 2015), e o LoCCAM (*Loosely Coupled Context Acquisition Midlleware*) (MAIA *et al.*, 2013).

O CAOS tem por objetivo abordar problemas comuns ao mundo da computação para dispositivos móveis, como o consumo de energia e as tarefas de processamento, delegando execuções de atividades para um nó na rede, de modo a auxiliar dispositivos móveis com pouca capacidade de processamento através da utilização do método de *offloading* de processamento atrelado ao potencial da *Fog* e *Cloud Computing*.

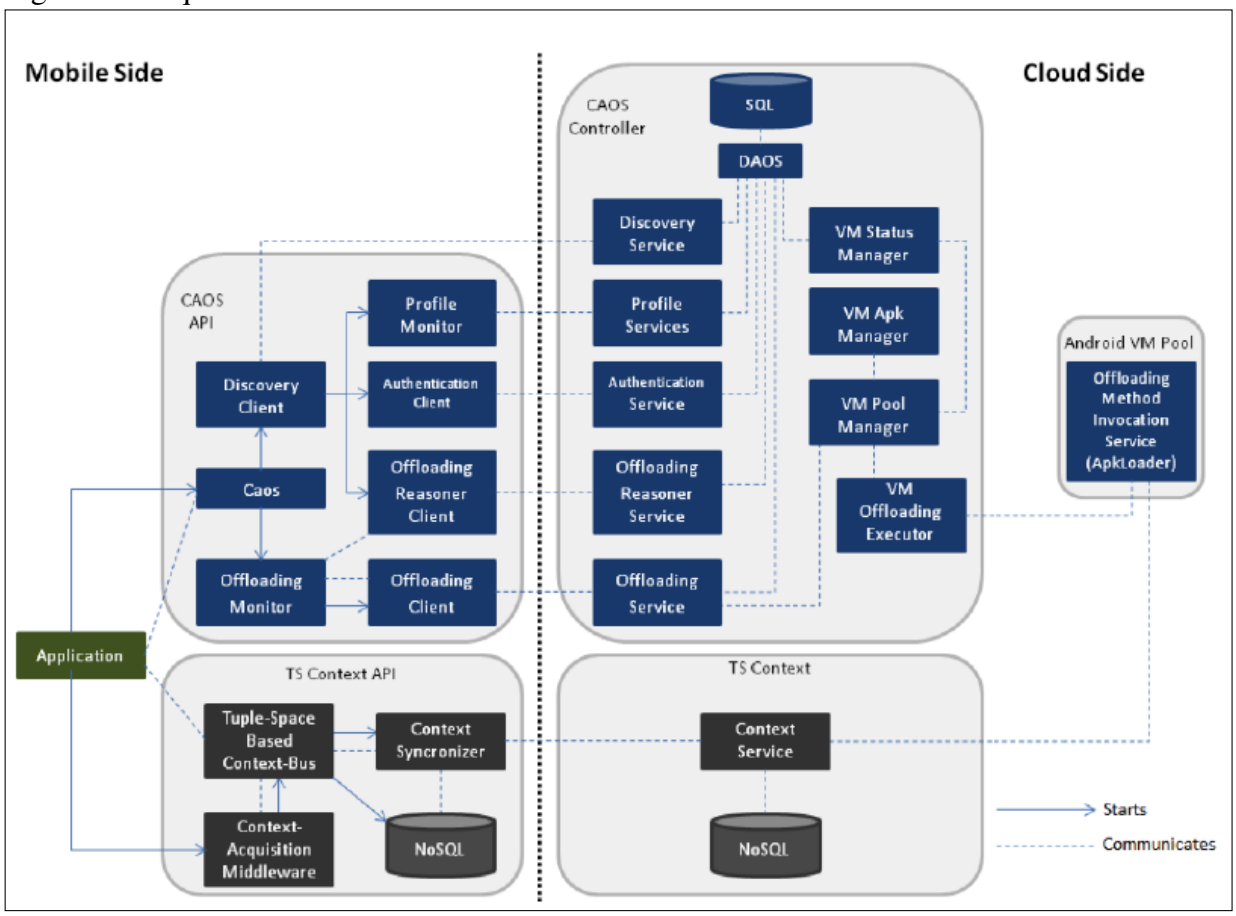
O CAOS utiliza o modelo Cliente-Servidor, no qual um computador faz o papel do servidor e os dispositivos móveis fazem o papel do cliente. No servidor é criada uma máquina virtual no virtualizador Virtual Box com a imagem do sistema operacional Android. Esta máquina virtual se comunica com o servidor via ADB (*Android Debug Bridge*) e é a responsável por realizar os processamentos advindos das requisições de *offloading* através de uma aplicação denominada ApkLoader.

No lado cliente, as aplicações que utilizam o CAOS necessitam implementar uma API criada para o CAOS. Tal API permite que o desenvolvedor da aplicação possa habilitar a comunicação de sua aplicação com o servidor CAOS e marcar, através da utilização de *tags*, os métodos os quais este deseja que sejam passíveis de *offloading*.

A arquitetura do CAOS é dividida basicamente em duas partes, uma parte referente ao lado servidor que executa no *cloudlet*, e uma parte referente ao lado cliente que é executado nos dispositivos móveis, assim como ilustrado na Figura 3.

O lado cliente, referente aos dispositivos móveis, é composto de 10 módulos. O módulo chamado CAOS é responsável por orquestrar a inicialização dos outros módulos do lado cliente. O *Discovery Client* é responsável por verificar se existe na rede alguma aplicação CAOS *Controller*, que é o servidor das aplicações. O *Authentication Client* envia dados do dispositivo móvel para que o lado servidor tenha um controle de quais dispositivos estão solicitando seus serviços. O *Offloading Monitor* supervisiona a execução da aplicação móvel verificando a existência de uma anotação denominada @*Offloadable*, que indica que o método pode ser

Figura 3 – Arquitetura do CAOS




Fonte: GOMES et al., 2017.

executado remotamente. Ao encontrá-la, o módulo *Offloading Reasoner Client* consulta o módulo *Offloading Reasoner Server* no lado servidor, recebendo uma estrutura de decisão para que o *Offloading Monitor* possa percorre-la e determinar se a realização do *offloading* é viável. Caso a resposta seja negativa a aplicação segue seu fluxo normal executando localmente, porém, caso seja positiva, o *Offloading Client* inicia o processo de *offloading* que transfere o método e seus parâmetros para ser executado no lado servidor (*Offloading Service*).

O módulo *Profile Monitor* é responsável por monitorar informações como latência da rede, estado da memória, e velocidade de transmissão da rede, para que esses dados possam ser enviados ao *Offloading Reasoner Service* no lado servidor, que criará uma estrutura de decisão a partir destes dados.

Para o gerenciamento do contexto, o CAOS criou os módulos *Context-Acquisition Middleware* e *Tuple-Space Based Context-Bus* com base no projeto LoCAMM (MAIA *et al.*, 2013) e adicionou o módulo *Context Syncronizer*, responsável por compartilhar dados contextuais entre o lado cliente e o lado servidor.

Similar ao lado cliente, o lado servidor divide-se em módulos em que cada um tem

sua especificidade. São 11 módulos: *Discovery Service*, que provê pontos de conexão para acesso as aplicações CAOS; *Authentication Service*, responsável por controlar quais dispositivos estão conectados ao serviços do CAOS, além de salvar informações sobre os dispositivos; *Profile Service*, que mantém uma avaliação histórica do tempo de execução dos métodos que fizeram *offloading*, sendo estas informações utilizadas para decidir se o método deve realizar o *offloading* ou não;*Offloading Reasoner Service*, responsável por gerar a estrutura de decisão para realização do *offloading* enviada ao *Offloading Reasoner Client*; *Offloading Service*, que recebe as requisições de *offloading* e redireciona para o VM *Pool Manager*, que ao fim da execução retorna o resultado para o *Offloading Client* e persiste os dados; VM *Pool Manager*, provê o ambiente virtual de execução Android para as execução do *offloading*; *VM Apk Manager*, responsável por enviar os arquivos compilados da aplicação para a máquina virtual Android; *VM Status Manager*, monitora e mantém todas as informações sobre as máquinas virtuais na *VM Pool Manager*; *VM Offloading Executor*, requisita a execução do *offloading* na máquina virtual fazendo a chamada ao *Offloading Method Invocation*, este que por fim executa na máquina virtual o *offloading* do método solicitado pela aplicação.

Na abordagem do CAOS, a necessidade de utilização de uma máquina virtual torna a infraestrutura necessária para o processo de *offloading* um custo adicional, uma vez que é necessário um computador que faça o papel de servidor com a máquina virtual. além de ser complicado o processo de instanciação do servidor. Outra dificuldade encontrada no CAOS é que a atribuição de IP's das máquinas e a alocação dos APK's das aplicações que utilizarão o processo de *offloading* tem que ser de forma manual.

### *2.1.6 CAOS D2D*

O CAOS D2D (*Context Acquisition and Offloading System Device to Device*) é um *framework* baseado no CAOS, onde é possível efetuar *offloading* diretamente de um dispositivo móvel para outro. Nessa abordagem, cada dispositivo pode atuar como cliente ou servidor sem a figura de um servidor central (SANTOS *et al.*, 2017). Deste modo, é eliminada a dependência de uma máquina virtual e é habilitada a execução de *offloading* em diversos dispositivos que estejam conectados à rede e ofereçam o papel de servidor. Outra vantagem em relação ao CAOS é que no CAOS D2D a descoberta dos IP's dos dispositivos se faz de forma automática, assim como o *deploy* dos APK's das aplicações que utilizarão o processo de *offloading*.

Para realizar a descoberta dos servidores a aplicação do CASO D2D que deseja

realizar o *offloading* faz periodicamente um *broadcast* na rede, enviando mensagens *User Datagram Protocol* (UDP) para encontrar algum dispositivo que faça o papel de servidor. Quando um dispositivo recebe tal mensagem, esse responde com uma mensagem TCP com informações sobre seu endereço IP e as portas onde os serviços do CAOS estão sendo executados.

Com a utilização do CAOS D2D é possível o uso de múltiplos ambientes de *offloading*, porém os processos de decisão e descoberta dos ambientes fica a cargo dos dispositivos móveis, o que gera mais processos a serem executados e um aumento do consumo de energia. No CAOS D2D o autor reconhece a necessidade de implementação de um módulo de decisão mais robusto para auxiliar na escolha do ambiente de *offloading*.

## 2.2 Considerações Finais

Esse capítulo apresentou as abordagens de *Cloud Computing*, *Fog Computing* e *Mobile Cloud Computing* que são utilizadas para o auxílio nos processamentos dos dispositivos móveis através da abordagem conhecida como *offloading*. Esses conhecimentos são importantes para contextualização do presente trabalho e compreensão das próximas seções. Nesse capítulo também foram apresentados os desafios ainda existentes para a implementação de uma infraestrutura que utilize o poder latente dos dispositivos móveis, desafios estes que serão alvo dessa dissertação.

Por fim, foram apresentadas duas infraestruturas de realização de *offloading*, CAOS e CAOS D2D. A apresentação de tais infraestruturas é importante para o entendimento da arquitetura que será introduzida ao leitor no capítulo 4.

# 3 TRABALHOS RELACIONADOS

Neste capítulo abordaremos alguns artigos que têm propostas semelhantes com a desta dissertação. Tais artigos apresentam abordagens em relação ao tema de decisão otimizada sobre escolha dos melhores dispositivos para processamento de *offloading*.

## 3.1 *Offloading* para dispositivos móveis

Dentre os vários estudos com relação a *offloading* computacional, foram selecionados aqueles que abordam a seleção do melhor dispositivo para a realização desse processo, levando em consideração diversos parâmetros escolhidos pelos autores para sua tomada de decisão. Dentre esses parâmetros, o cálculo da capacidade computacional dos dispositivos, a heterogeneidade dos dispositivos, e a preocupação com a mobilidade dos dispositivos foram o foco principal.

Para compreensão dos seguintes estudos faz-se necessário definir que em MCC são empregados outros dispositivos comumente denominados de *cloudlets* que junto com os servidores de *Cloud* e *Fog Computing* auxiliam na melhoria das aplicações móveis (MAHMUD *et al.*, 2018). Um *cloudlet* é definido como um dispositivo confiável, rico em processamento e que está próximo ao usuário, como na mesma rede *Wireless Local Network* (WLAN) (VERBELEN *et al.*, 2012). Portanto, sendo estes também compreendidos como REEs, como abordado neste trabalho.

### *3.1.1 FemToClouds: Leveraging Mobile Devices to Provide Cloud Service at the Edge*

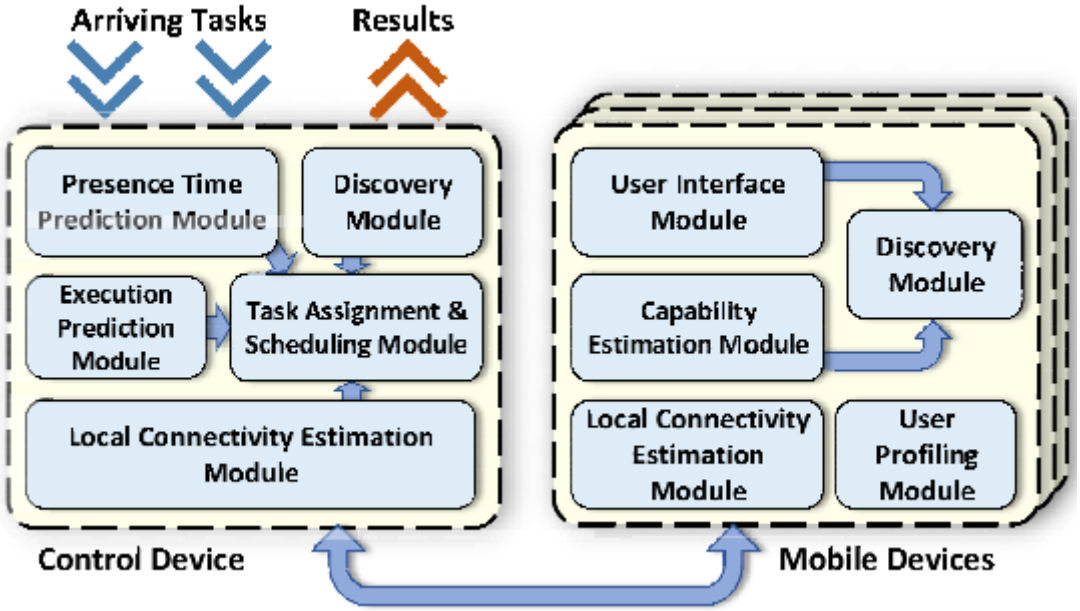
FemToClouds (HABAK *et al.*, 2015) é um protótipo de sistema com o objetivo de prover uma nuvem móvel dinâmica, autoconfigurável e multidimensional de um conjunto de dispositivos móveis, levando em consideração a rotatividade na participação dos dispositivos.

A arquitetura do sistema pressupõe que os dispositivos móveis possuem o serviço *FemToClouds client* instalado. Tal serviço estima a capacidade computacional do dispositivo móvel e permite ao usuário a realização de configurações, para que junto com um histórico de utilização do dispositivo possa ser determinada a sua capacidade disponível. Após este processo, a informação coletada é compartilhada com o dispositivo de controle na rede, o qual será responsável por estimar o tempo de presença do usuário e configurar a participação deste dispositivo como um oferecedor de serviço.

A Figura 4 apresenta a arquitetura projetada para o sistema FemToClouds. O modulo

do lado direito, responsável pelos dispositivos móveis que oferecem serviços, tem as funções de: obter as configurações do dispositivo, incluindo as definições do usuário; estimar a capacidade computacional de tal dispositivo; obter dados sobre as preferencias e comportamento do usuário em diferentes cenários, para determinar seu período conectado ao *FemToClouds Server*.

Figura 4 – Arquitetura FemToClouds




Fonte: HABAK et al., 2015.

O módulo do lado esquerdo, referente ao dispositivo de controle, é responsável por: estimar a carga de execução dos processamentos demandados; predizer o tempo de presença dos usuários na rede FemToClouds; atribuir tarefas para os dispositivos; estimar o nível de conexão entre o dispositivo de controle e os dispositivos móveis; descobrir novos dispositivos que entraram na rede.

O sistema permite ao usuário a habilitação ou não do dispositivo móvel para ser utilizado pelo FemToClouds e a definição do nível mínimo de bateria que o dispositivo deve possuir para estar conectado. Dentre os meios utilizados para estimar a capacidade dos dispositivos encontra-se a contagem do número de núcleos do processador dos dispositivos. Para a previsão da presença do usuário na rede é utilizada a coleta de dados históricos de comportamento do usuários, além da possibilidade do usuário definir seu tempo de permanência.

Para avaliar a performance do sistema FemToClouds, os autores realizaram previamente a avaliação dos dispositivos móveis através da execução de uma aplicação de multiplicação de matrizes e realizaram um estudo das necessidades das aplicações reais. Os experimentos focaram em entender o efeito de diferentes parâmetros de ambientes na performance do FemTo-

Clouds e avaliar a performance do protótipo desenvolvido. Em seus experimentos, os autores demonstraram que a predição de presença impactou positivamente na escolha dos dispositivos se comparado com a mesma heurística de seleção sem a predição de presença.

### *3.1.2 Optimal mobile device selection for mobile cloud service providing*

Zhou *et al.* (2016) propõem um método de seleção do melhor dispositivo móvel considerando seu status e estabilidade na rede para o provimento de serviços. Seu objetivo é melhorar a qualidade de serviço provido pelos dispositivos móveis, uma vez que a saída deste dispositivo da rede antes do termino do processamento que lhe foi designado impacta negativamente no serviço, pois é necessário selecionar outro dispositivo e reiniciar o processamento.

Os autores partem do pressuposto de que todos os dispositivos clientes que estão provendo serviço possuem a aplicação necessária para coleta de informações e que todos os clientes estão dispostos a compartilhar suas informações. Tais informações são enviadas para um dispositivo móvel que tem o papel de controlador e possui as atividades de descoberta de novos dispositivos, distribuição dos processos, monitoramento do perfil dos dispositivos, estimação do tempo de execução dos processos, e predição do tempo de presença dos dispositivos.

Dentre as informações dos dispositivos móveis levadas em consideração pelos autores estão: o status de carregamento do dispositivo (carregando ou não), o tempo declarado de partida do usuário, o real tempo de partida do usuário, a causa da partida do usuário, e o horário atual. O estudo considera um dispositivo estável aquele em que o tempo de partida do usuário é igual ou superior ao declarado pelo mesmo. Para resolver o problema de otimização na escolha do dispositivo os autores constroem um modelo de nuvem baseado em pesos, onde os dispositivos são classificados de acordo com sua distância ao dispositivo central, sua cobertura de rede, e sua dispersão na rede.

O algorítimo proposto pelos autores realiza uma distribuição das tarefas entre os dispositivos móveis presentes calculando a diferença entre o tempo de presença do dispositivo e o tamanho da atividade (que é medida em tempo que esta leva pra executar), selecionando os que tão aptos e buscando o de melhor pontuação através dos pesos que foram criados.

Para a realização dos experimentos de performance, os autores utilizaram a seleção randômica de dispositivos para a execução dos processamentos como comparação. Como resultado dos experimentos foi demonstrado que o tempo de realocação dos dispositivos por causa de falhas por abandono de rede é menor, e que é reduzida a escolha de dispositivos que

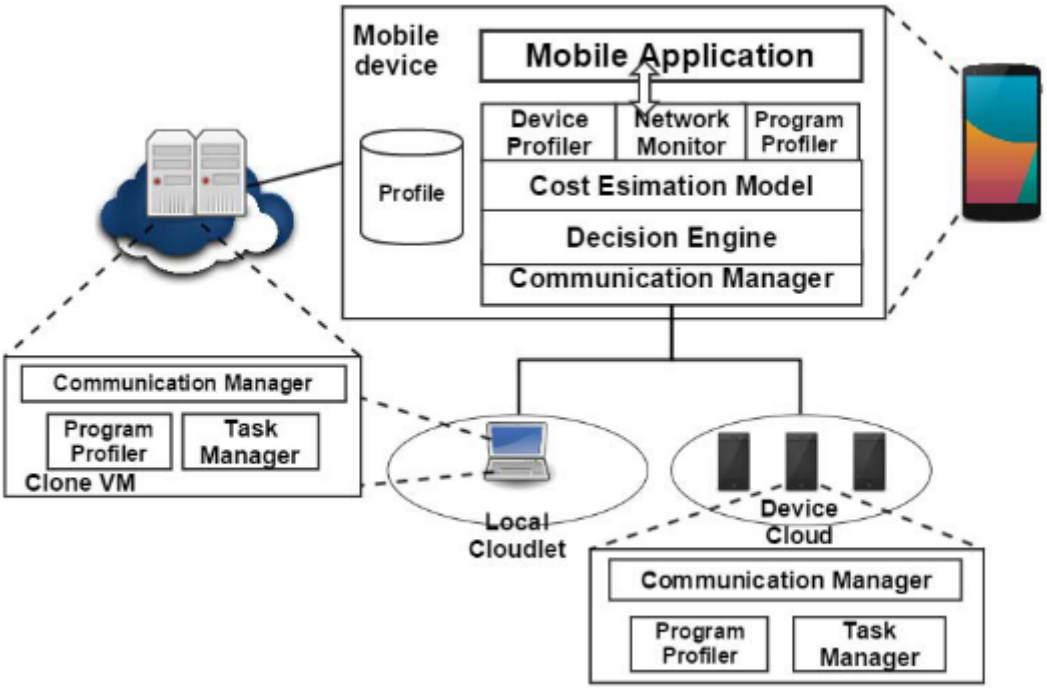
não cumpram com seu tempo declarado de permanência.

### *3.1.3 A Context Sensitive Offloading Scheme for Mobile Cloud Computing Service*

Zhou *et al.* (2015) apresentam um sistema para a decisão otimizada de onde, quando e para quem enviar o *offloading* de métodos, levando em consideração o contexto dos dispositivos móveis e seus recursos para prover uma melhor performance e economia de energia.

Os autores propõem a arquitetura apresentada na Figura 5, que se baseia no modelo de comunicação cliente-servidor. O lado cliente possui as atividades de monitoramento do contexto, gerenciador de comunicação e é responsável pelo método de decisão. O lado servidor é responsável pelo gerenciamento de comunicação, a gerencia dos perfis dos dispositivos e a gerencia de atribuição das tarefas.

Figura 5 – Arquitetura do sistema




Fonte: ZHOU et al., 2015.

Para a predição de custo de *offloading*, são armazenados no banco de dados do dispositivo algumas informações, tais como o tempo de execução do método, o número de instruções executadas, a memória alocada, o número de chamadas ao método, e o local de execução do método (local ou na nuvem). Atualizando-se tais informações a cada chamada do método.

Dentre as características coletadas para a construção do perfil dos dispositivos estão a média da frequência da CPU, a média de uso da CPU, a memória utilizada e o nível de bateria. Tais informações são utilizadas pelo modelo de estimativa de custo utilizado pelo módulo de decisão no dispositivo.

Os autores partem do pressuposto de que as operações executadas pelos métodos que realizam *offloading* são independentes e podem ser particionadas em sub-tarefas para execução paralela. Os autores consideram as máquinas virtuais utilizadas como homogêneas e não fazem sua classificação.

A problemática solucionada pelos autores é a de distribuir entre dispositivos heterogêneos o processamento de tarefas. Tais tarefas são solicitadas de modo uniforme e imprevisível, portanto o problema de distribuição destas tarefas é mostrado ser NP-completo.

No algoritmo de decisão baseado em contexto apresentado pelos autores, são considerados dois fatores. O primeiro é o melhor meio de comunicação (Wi-fi, 3g, 4g, Bluetooth) levando em consideração a velocidade de transmissão dos dados, custo energético e monetário. O segundo é o melhor dispositivo para execução através da comparação do tempo de execução armazenado nos dados históricos.

Para a realização da avaliação de performance, os autores utilizam dois tipos de aplicações, uma com uma carga reduzida de transferência de dados e outra com carga elevada. A primeira é uma aplicação que realiza operações matemáticas de acordo com o entrada de dados, e a segunda é uma aplicação de detecção facial. Como resultado de suas avaliações os autores demonstraram uma média de 50% de redução do consumo energético e a redução do tempo de processamento.

### *3.1.4 RainCloud - Cloudlet Selection for Effective Cyber Foraging*

Chilukuri *et al.* (2017) propõem uma heurística de seleção do melhor *cloudlet* para o processo de *offloading*. Os *cloudlets* são classificados de acordo com seus recursos disponíveis, estabilidade na rede e necessidades da aplicação que solicita o *offloading*.

Quando o dispositivo cliente necessita realizar o processo de *offloading*, ele envia uma mensagem via *broadcast* tentando estabelecer conexão com todos os *cloudlets* que estiverem disponíveis na rede. Caso existam *cloudlets* disponíveis, estes enviam uma mensagem de resposta com suas configurações. Uma pontuação é calculada para cada *cloudlet* de acordo com suas configurações. Por fim, o dispositivo cliente seleciona o de melhor pontuação e realiza uma

conexão para a execução do processo de *offloading*.

Os autores levam em consideração os seguinte parâmetros para o cálculo da pontuação dos *cloudlets*:

1. Velocidade de processamento;
2. Memória disponível;
3. Latência da rede;
4. Velocidade de banda;
5. Estabilidade do dispositivo;

O cálculo da pontuação do *cloudlet* ocorre utilizando-se pesos em seus parâmetros de configurações. Tais pesos são definidos pelo desenvolvedor da aplicação, que tem a liberdade de priorizar o que é mais importante para a execução de sua aplicação.

Para o cálculo da estabilidade os autores consideram a intensidade do sinal de rede recebida pelo dispositivo, avaliando como os de melhor estabilidade aqueles em que o sinal é forte e não sofre variações.

Para a avaliação de seu trabalho os autores utilizam 3 aplicações, uma de tradução de livros digitais, uma de reconhecimento facial, e uma de transmissão de áudio. Tais aplicações servem para demonstrar os diferentes tempos de respostas de acordo com as necessidades de cada tipo de aplicação. Os autores demonstram que sua heurística reduz o tempo das atividades de *offloading* se comparado com a seleção básica da utilização do primeiro dispositivo que responde a solicitação.

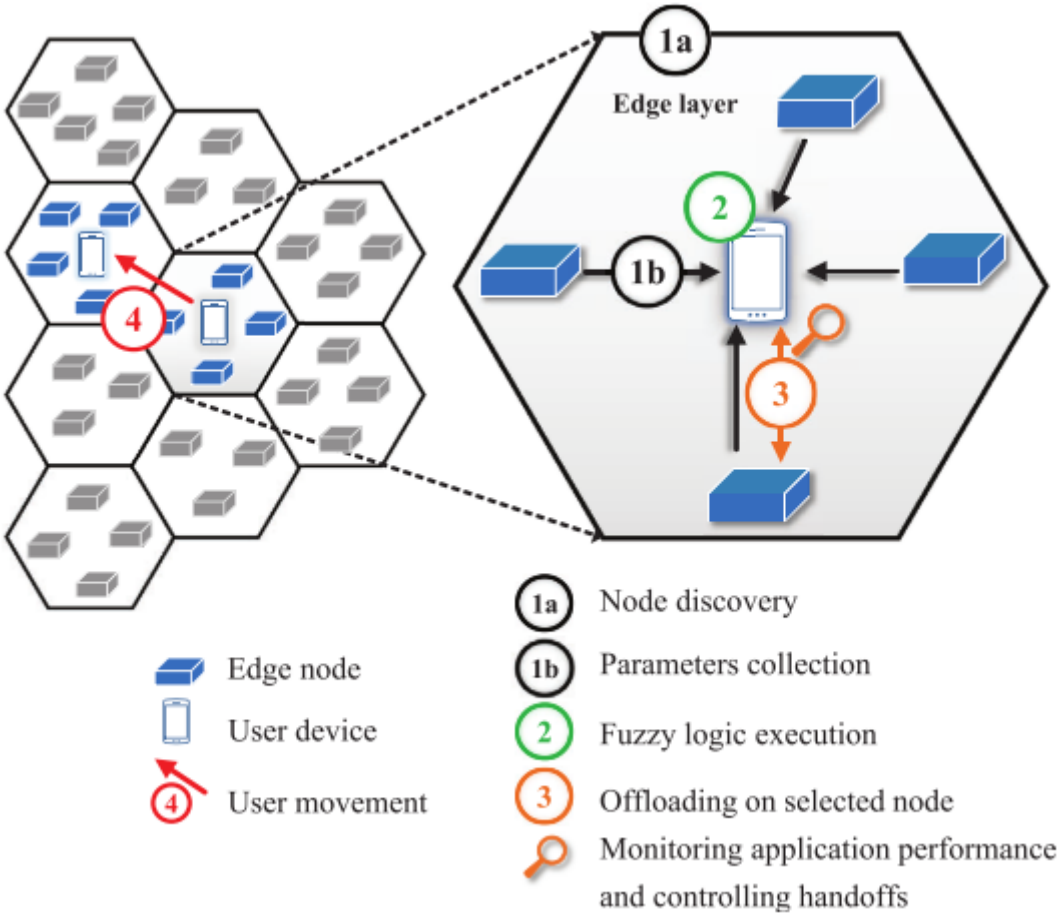
### *3.1.5 Fuzzy Handoff Control in Edge Offloading*

Basic *et al.* (2019) propõem um algoritmo baseado em lógica difusa (ou lógica Fuzzy) para selecionar um nó de destino para a realização do processo de *offloading* computacional com base nos parâmetros de largura de banda, velocidade do processador e latência. É proposto também um controlador de transferência do processo de *offloading*, que leva em consideração o tempo de resposta da aplicação como indicador para decidir se deve mover a execução da tarefa para outro nó.

A Figura 6 demonstra como funciona o fluxo de *offloading* proposto pelos autores. Inicialmente o dispositivo que deseja realizar o processo de *offloading* procura por outros dispositivos a "um salto de distância", ou seja, na mesma WLAN. Os dispositivos que estão presentes na mesma rede se identificam, informando seus dados sobre capacidades de largura de

banda, processador e latência. De posse de tais parâmetros, o dispositivo executa o algoritmo baseado em lógica difusa para a decisão de para qual dispositivo enviar a requisição de *offloading*. Durante o processo de *offloading*, é monitorado o tempo de execução da atividade e após diversas execuções o dispositivo mapeia para quais dispositivos é mais vantajoso demandar execuções de processamentos.

Figura 6 – Modelo de *offloading* em *Edge Computing*.




Fonte: BASIC et al., 2019.

Para a realização da seleção do dispositivo a atender uma requisição de *offloading*, o algoritmo de decisão funciona da seguinte maneira: após receber os parâmetros dos dispositivos (largura de banda, poder de processamento e latência), tais valores são mapeados para conjuntos difusos relevantes com um valor de associação entre 0 e 1, descrevendo a adequação de cada nó a tarefa de *offloading*, em que valores maiores significam maior adequação. Dentre os dispositivos disponíveis é então selecionado o que obteve mais alta adequação.

Os autores demonstram a redução do tempo de resposta da operação de *offloading* quando utilizada sua seleção de nós através da lógica difusa com a utilização de duas aplicações.

A primeira realizava o reconhecimento facial e a segunda auxiliava na navegação em mapas. Para a realização da comparação de sua abordagem, os autores utilizaram dois métodos de seleção do dispositivo ideal: o primeiro considera o dispositivo mais próximo como o ideal e o segundo considera o dispositivo de maior largura de banda. Seus resultados demonstraram que sua abordagem reduz o tempo de resposta em 86.14%.

## 3.2 Comparativo Entre os Trabalhos

Para a comparação entre os trabalhos apresentados na seção anterior e o proposto nessa dissertação, foram elencados alguns pontos de importância para a escolha otimizada do dispositivo a atender o processo de *offloading*.

I. Consumo energético: verifica se o trabalho considera o consumo energético dos dispositivos ou possui formas de realizar a economia de energia dos dispositivos utilizados no processo de *offloading*.
II. Capacidade computacional: verifica se o trabalho leva em consideração métricas de comparação de poder de processamento dos dispositivos para auxiliar a determinar os dispositivos que estão aptos a executar os processamentos em menos tempo.
III. Decisão do usuário: refere-se a possibilidade de o usuário influenciar na tomada de decisão através da configuração de suas preferências de habilitação do serviço em seu dispositivo.
IV. Decisão dos desenvolvedores: corresponde a possibilidade de os desenvolvedores da aplicação influenciarem na tomada de decisão através da informação das prioridades da sua aplicação (E.g. solicitar maior velocidade de processamento o possível)
V. Dinamicidade dos dispositivos: considera se o trabalho leva em conta que os dispositivos podem entrar e sair da rede de forma dinâmica e afetar a tomada de decisão.
VI. Otimização da distribuição: esse parâmetro é relacionado a escolha dos melhores dispositivos para dado instante de tempo e distribuição de carga se diversas solicitações estiverem sendo feitas no mesmo momento.
VII. Heterogeneidade dos dispositivos: verifica se o trabalho leva em consideração diferentes dispositivos (Tablets, máquinas virtuais, celulares, etc.)

Na Tabela 1 os campos que foram marcados em vermelho possuem algumas ressalvas em relação a forma como foram utilizados pelos autores. Primeiramente, no estudo realizado por Habak *et al.* (2015), em relação a capacidade computacional, os autores em seus experimentos realizam uma medição prévia de três dispositivos que serão utilizados, porém o interessante seria

Tabela 1 – Comparativo entre os trabalhos relacionados e a proposta

| | I | II | III | IV | V | VI | VII |
|---|---|---|---|---|---|---|---|
| HABAK et al., 2015 | | X | X | | X | X | X |
| ZHOU et al., 2015 | X | X | | | | X | X |
| ZHOU et al., 2016 | | | X | | X | X | X |
| CHILUKURI et al., 2017 | | X | | X | X | X | X |
| BASIC et al., 2019 | | X | | | X | X | X |
| ESTE TRABALHO | X | X | X | X | X | X | X |

Fonte: Elaborado pelo autor.

a realização da medição de forma automática e sob demanda que sirva para qualquer dispositivo.

No estudo de Zhou *et al.* (2015), a respeito da heterogeneidade dos dispositivos, os autores pressupõem que as máquinas virtuais sejam homogêneas, entretanto é interessante tratar as diferentes configurações de cada máquina virtual.

Em seu estudo, os autores Zhou *et al.* (2016) consideram o estabelecimento do horário de partida do usuário a única possibilidade de decisão. Com essa informação e a utilização de dados históricos, é realizada a decisão de se é possível confiar no usuário com relação a sua permanência na rede. Contudo, seria interessante atribuir mais poder de decisão aos usuários, como por exemplo a possibilidade de desativação do provimento de serviços pelo seu dispositivo e a procura de uma forma de se precaver da saída do usuário da rede de forma automática, sem necessitar que o usuário determinasse seu momento de partida.

Por fim, o estudo dos autores Basic *et al.* (2019) não especifica uma abordagem para evitar falhas na comunicação devido a dinamicidade do dispositivo. O dispositivo cliente só tem conhecimento da indisponibilidade de outro dispositivo que tenha saído da rede após uma falha ou o não recebimento de resposta por um longo período de tempo.

## 3.3 Considerações Finais

Esse capítulo apresentou os trabalhos relacionados que mais se aproximam a proposta desta dissertação, descrevendo as contribuições dos autores e seus experimentos.

Cada estudo aborda o problema de escolha do melhor dispositivo para realização do *offloading* de acordo com suas próprias estratégias e priorizações. Dentre os parâmetros considerados pelos autores, foram elencados 7 que possuem grande relevância para a tomada de decisão. Por fim, estes parâmetros foram utilizados para a realização da comparação entre os trabalhos com a construção da Tabela 1.

Após a realização da comparação dos trabalhos elencados, pôde-se perceber que ainda existem melhorias a serem realizadas para a realização do processo de seleção otimizada de um dispositivo que atenda a requisições de *offloading*, como por exemplo uma abordagem que englobe os 7 parâmetros elencados anteriormente. O próximo capitulo apresenta a abordagem proposta por este trabalho de mestrado que contempla todos estes parâmetros.

# 4 ESTRATÉGIA PARA SELEÇÃO DO REE EM UM AMBIENTE DINÂMICO E HETEROGÊNEO

Esse capítulo apresenta a abordagem proposta para contornar as problemáticas de (*i*) heterogeneidade e (*ii*) dinamicidade dos dispositivos, além de considerar o (*iii*) poder de decisão de usuários e (*iv*) do desenvolvedor da aplicação ao se realizar processos de *offloading* entre dispositivos móveis.

Para a apresentação da abordagem desenvolvida, a estrutura desse capítulo foi dividida em 4 seções principais. Na primeira seção, é apresentada uma visão geral do fluxo de comunicação entre os dispositivos para a realização de processos de *offloading*. Na segunda seção, é demonstrado como foi possível capturar o contexto dos dispositivos e utilizá-lo para criar um algoritmo de seleção do melhor dispositivo para atender requisições de *offloading* baseado em métodos matemáticos de seleção multicritério. Na terceira seção, é apresentada a arquitetura de software, criada a partir dos projetos CAOS e CASO D2D, que habilita o processo de requisição de *offloading* de métodos em um ambiente com múltiplos dispositivos móveis. Esta infraestrutura foi utilizada para execução dos experimentos do capítulo 5, em que são demonstradas a corretude e a viabilidade da abordagem de seleção proposta. Por fim, na última seção são realizadas as considerações finais sobre o capítulo.

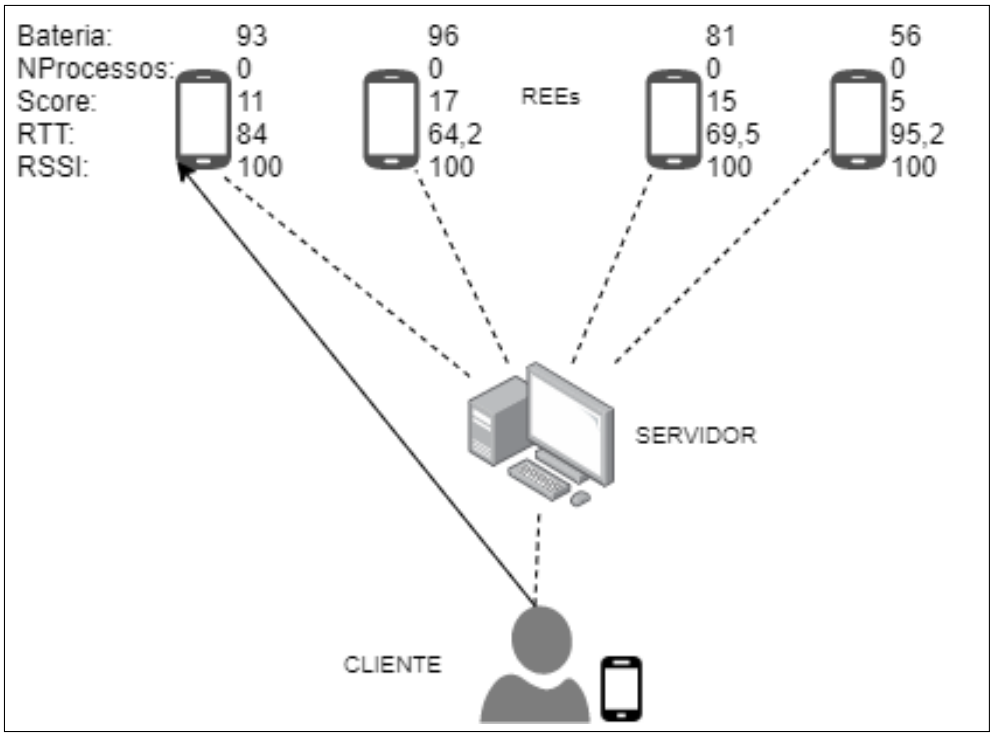
## 4.1 Visão Geral

Esta dissertação buscar dar suporte à escolha do melhor local para realização de *offloading* em um ambiente com a presença de diversos dispositivos com capacidade de executar tarefas em nome de dispositivos mais fracos.

Para uma visualização mais prática, a Figura 7 apresenta um exemplo do processo de *offloading* com múltiplas opções de REEs. A ideia geral é a de que quando um dispositivo cliente deseja realizar uma operação de *offloading*, exista um componente que possa indicar para onde o processo/método deve ser migrado, sendo esse componente um intermediário no processo, tendo o papel de realizar a decisão de para qual dos REEs o cliente deve solicitar o *offloading*, se comportando como um servidor de distribuição de *offloading*. Este servidor precisa conhecer e ser conhecido pelo cliente e por todos os REEs disponíveis na rede. Na figura, as setas tracejadas representam essa comunicação entre o servidor e todos os dispositivos. Para que o servidor possa decidir o melhor dispositivo para atender as solicitações dos cliente, é necessário que esse saiba algumas informações de contexto que sejam relevantes para a tomada de decisão de cada

REE. Na figura, cada REE tem seu valor de bateria, número de processos sendo executados (NProcessos), uma pontuação de performance do dispositivo (Score), velocidade de comunicação do dispositivo na rede (RTT), e o nível da qualidade de sinal wi-fi (RSSI). Através dessas informações, o servidor pode então realizar a escolha do melhor dispositivo para atender a uma requisição de *offloading*. Após a tomada dessa decisão o dispositivo cliente pode se comunicar diretamente com o dispositivo REE escolhido, representado na figura pela seta direcional, e realizar o processo de *offloading*.

Figura 7 – Exemplo de seleção para *offloading* com múltiplos REEs.




Fonte: Elaborado pelo autor

Para viabilizar este cenário, existem uma série de informações e decisões a serem tomadas. Estas incluem quais são a informações que são interessantes para se determinar que um REE é mais adequado para atender uma solicitação de *offloading* que outro. Por exemplo, pode-se tomar apenas como critério de decisão aquele REE com melhor desempenho, de modo a tornar o processo mais rápido.

Sendo assim, este capítulo busca apresentar uma solução para viabilizar o cenário aqui proposto a partir de (a) quais são os critérios para seleção de um REE para execução de uma tarefa no processo de *offloading*, (b) um algoritmo que utilize métricas para viabilizar a escolha do REE de acordo com os critérios definidos previamente e (c) uma arquitetura de software que dê suporte ao cenário aqui descrito.

### *4.1.1 Critérios*

Definir qual o melhor REE é uma tarefa subjetiva. O melhor pode ser o mais rápido, considerando o tempo que o dispositivo finaliza uma atividade que lhe foi delegada, mais o tempo de transferência do resultado. Ou o melhor pode ser o com maior autonomia de funcionamento (mais bateria). Ou ainda, o com maior disponibilidade (sem chances de sair da rede).

Nesta dissertação foram definidos 4 critérios de forma a balancear a seleção de acordo com os objetivos de velocidade de processamento e economia de energia dos REEs:

- Melhor dispositivo em termos de velocidade de processamento.
- Melhor dispositivo em termos de autonomia de energia
- Melhor dispositivo em termos de velocidade de processamento dentre os que tem autonomia de energia.
- Melhor dispositivo realizando o balanceamento entre capacidade energética e velocidade de processamento.

Além dos critérios definidos, filtros foram aplicados para que em todas as formas de seleção elencadas fossem levadas em consideração a dinamicidade dos dispositivos na rede e as decisões atribuídas aos proprietários do dispositivo. A próxima seção demonstra a estratégia adotada para que fosse possível realizar tão seleção.

## 4.2 Algoritmo de Seleção Baseado em Contexto dos Dispositivos

O meio encontrado para realização do monitoramento dos dispositivos foi a definição de diversos parâmetros que refletissem o estado atual do dispositivo. Tais parâmetros foram então utilizados para realizar a seleção do melhor dispositivo através da criação de um algoritmo que aplica dois métodos matemáticos de seleção multicritério.

Para a compreensão do algoritmo proposto, as próximas subseções abordam o significado de cada parâmetro coletado e como esses auxiliam no monitoramento do contexto para tomada de decisão. Em seguida é apresentado como tais parâmetros foram utilizados para a construção de filtros que permitissem remover do processo de seleção os REEs não propícios para atender requisições de *offloading* e a construção do algoritmo para selecionar o melhor dispositivo dentre os REEs restantes.

### *4.2.1 Parâmetros utilizados para escolha de um REE*

A comparação entre diferentes dispositivos móveis não é uma tarefa trivial. Diversos são os fatores que podem influenciar no poder de processamento de um dispositivo como CPU, memória RAM, ROM, e consumo energético. Além disto, combinações de recursos cada vez mais sofisticados para criar produtos para diferentes segmentos de usuários finais complicam tal comparação (HAN; CHO, 2016).

De forma a simplificar a comparação da capacidade dos dispositivos móveis, neste estudo foram utilizados diversos parâmetros que nos permitiram identificar qual o melhor dispositivo para atender a uma requisição de *offloading*, além de terem sido utilizados para realizar a verificação do contexto dos dispositivos no que concerne à bateria, dinamicidade e limites impostos pelos usuários.

#### *4.2.1.1 Poder de processamento*

Como abordado anteriormente, o poder de processamento de um dispositivo móvel é derivado de diversos componentes o que torna difícil sua definição. Além disso, de acordo com o número de processos sendo executados no dispositivo, este pode apresentar variações na velocidade de processamento em momentos distintos.

Uma abordagem bastante adotada quando se necessita realizar comparações entre diversos tipos de objetos é a utilização de métodos de *Benchmarking*. Na área de dispositivos móveis, estes métodos são bastante utilizados para verificar o poder de processamento de um dispositivo (GUO *et al.*, 2017). Tais métodos funcionam basicamente através do cálculo do tempo de execução de uma tarefa que é atribuída aos dispositivos. Após o fim da execução, é realizada uma classificação dos dispositivos de acordo com o tempo de finalização de tal tarefa. Os dispositivos que finalizaram primeiro são então considerados os de maior poder de processamento.

Baseado na ideia de *Benchmarking*, neste trabalho foi utilizado um método que proporciona a classificação dos dispositivos de acordo com a velocidade de processamento a partir da execução da função de criptografia *Secure Hash Algorithm 1* (SHA-1). O SHA-1 funciona de forma a transformar uma mensagem de qualquer tamanho em 40 dígitos hexadecimais, por meio da aplicação de diversos procedimentos nesta mensagem. O valor *Hash* gerado é um valor único e que torna difícil a obtenção do conteúdo da mensagem através da simples posse do código

*Hash* (KALE; DHAMDHERE, 2018).

A escolha do *Benchmarking* utilizando o SHA-1 foi baseada na necessidade de um método de rápida execução, porém que fosse eficaz para determinar a capacidade de processamento de um dispositivo. No algoritmo proposto nesta dissertação, o valor de processamento de cada dispositivo é calculado pela execução contínua da transformação da mensagem *"MDCC UFC CAOS Project"* em um valor *Hash* por 30 mil vezes. O tempo para execução desse processamento é retornado na forma de nanosegundos, que é então transformado em um valor de dois dígitos e atribuído ao dispositivo como seu Score. Através deste Score é possível criar uma escala de poder de processamento dos dispositivos, sendo os dispositivos com menor valor de Score os mais rápidos no momento.

A Tabela 2 apresenta um exemplo de como ocorre a criação do Score para três dispositivos denominados de A,B e C. Os dois valores associados para cada dispositivo representam o tempo em nanosegundos da realização do processo descrito no parágrafo anterior e o seu Score atribuído de acordo com esse tempo. Neste exemplo, o dispositivo de maior poder de processamento seria o dispositivo B, pois esse dispositivo executa em menos tempo a mesma atividade que foi executada por todos. Porém, o tempo de execução da atividade pode aumentar conforme o dispositivo esteja sobre grande carga de processamento.

Tabela 2 – Exemplo de Scores de dispositivos.

| **Dispositivo** | **Tempo (ns)** | **Score** |
|---|---|---|
| A | 1594060211 | 15 |
| B | 495365982 | 4 |
| C | 1105736491 | 11 |

Fonte: Elaborado pelo autor

#### *4.2.1.2 Autonomia de Bateria*

A capacidade energética dos dispositivos móveis ainda é um empecilho na vida dos usuários. Apesar dos dispositivos mais modernos possuírem recursos cada vez sofisticados para economizar energia, esta ainda é um recurso finito que precisa ser utilizada com moderação (LI *et al.*, 2018).

Como qualquer processo realizado por dispositivos móveis demanda consumo energético, é necessário que se monitore a porcentagem de bateria dos dispositivos de modo a considerar este valor no processo de decisão de qual o dispositivo mais adequado para o envio de solicitações de *offloading*. Portanto, dentre os parâmetros a serem monitorados inclui-se

também o nível de bateria atual do dispositivo. Em nossa abordagem, quando o dispositivo está conectado a uma fonte de energia é atribuída a porcentagem de bateria igual a 101%. Dessa forma, posteriormente quando o algoritmo de seleção analisar o nível de bateria dos dispositivos disponíveis, é possível filtrar aqueles que estão conectados à um fonte de energia e considerá-los mais adequados para receber tarefas de *offloading*. Na subseção 4.2.2 será demonstrado que tal parâmetro influencia diretamente no método de seleção, tanto quando o objetivo for velocidade de processamento como quando for a economia de energia dos REEs.

#### *4.2.1.3 Decisão do usuário*

Ao envolver dispositivos de outros usuários para a execução do *offloading*, é interessante atribuir poder de decisão a estes usuários, uma vez que seus dispositivos podem ser os REEs para executar processos em nome de outros dispositivos, ocasionando um consumo energético. Para tanto, duas decisões básicas foram implementadas para dar o direito de participação ou não destes colaboradores na rede.

Primeiramente, a possibilidade de ativação e desativação do serviço foi implementada, permitindo que o usuário decida quando quer que seu dispositivo esteja disponível ou não como REE. Desta forma, pode-se realizar programaticamente o envio de um alerta para ter ciência da desconexão do dispositivo.

Uma segunda possibilidade é a atribuição, através da interface do usuário, de um limite mínimo de bateria no qual o usuário estaria disposto a oferecer seu dispositivo para processamento de métodos advindos via *offloading*. Deste modo, o usuário não precisa se preocupar que o serviço consuma sua bateria quando esta já estiver num percentual considerado baixo pelo mesmo. Esse limite é utilizado no algoritmo de decisão para remoção dos dispositivos que já atingiram o percentual de bateria indicado, respeitando a decisão do colaborador.

#### *4.2.1.4 Decisão do desenvolvedor*

Segundo a literatura sobre o tema, dentre as razões para a realização de *offloading* estão: economia de energia do dispositivo e a melhoria no tempo de execução de um determinado procedimento ou tarefa (REGO, 2016). Em geral estes dois objetivos são conflitantes. Neste trabalho, o foco é a diminuição no tempo de execução de um método por meio do *offloading*. Porém, como os dispositivos que atendem às solicitações de *offloading* podem ser dispositivos móveis, estes também possuem limitações energéticas. Com isso foi criada uma forma de

balanceamento energético de tais dispositivos de acordo com a intenção do desenvolvedor em solicitar o *offloading*.

Desta forma, foi dada a opção para o desenvolvedor decidir se o método marcado via a *Application Programming Interface* (API) para *offloading* tem foco em economia de energia ou diminuição no tempo de processamento. Nesse trabalho foi assumido que todo processo de *offloading* gera uma economia de energia para o cliente. Portanto, ao selecionar que o foco é encomia de energia, será tomado como objetivo balancear o consumo energético dos REEs disponíveis, selecionando aquele com maior capacidade energética no momento ou que esteja conectado à uma fonte de energia para atender à requisição de *offloading*.

Para isso, um novo parâmetro foi adicionado na API do CAOS na marcação *@Offloadable*. Este parâmetro foi batizado de *As Soon As Possible* (ASAP), e que pode assumir um valor lógico, funcionando da seguinte forma:

- Caso ASAP seja igual a "*false*", indica que para aquele método não é necessária uma resposta imediata, sendo feita a execução do *offloading* com o propósito de economia de energia dos REEs;
- Caso ASAP seja igual a "*true*", indica que para aquele método o mais importante é uma resposta o mais rápido o possível, sendo feito o *offloading* com o propósito de melhoria de desempenho da aplicação. Com isso, será selecionado o dispositivo com capacidade de atender de forma mais eficiente o pedido do cliente.

Na subseção 4.2.2 será explicado como tal decisão do desenvolvedor é considerada no algoritmo de seleção do melhor REE para atender a requisição.

#### *4.2.1.5 Intensidade de sinal Wi-Fi*

Como o objetivo de abordar a dinamicidade dos dispositivos na rede, foi inserido como parâmetro o valor de *Received Signal Strength Indicator* (RSSI) do dispositivo, comumente referenciado como intensidade de sinal Wi-Fi.

O monitoramento da intensidade do sinal *Wi-Fi* não é realizado somente com o propósito de ajudar no método de seleção para escolha de um dispositivo com boa qualidade de conexão. Este valor também permite criar uma taxa de confiança com a qual possamos verificar que um dispositivo não está na “borda” da rede, e com isso possa subitamente se tornar inalcançável por clientes de *offloading*.

Dessa forma, o valor em *dBm* da conexão *Wi-Fi* é capturado e normalizado em uma

escala de 0 a 100 para identificação da qualidade do sinal. O valor de 30% foi definido como limite de aceitação do dispositivo na rede, que será utilizado no algoritmo de seleção, pois este valor é próximo de -90 dBm o que torna extremamente falha qualquer comunicação na rede (METAGEEK, 2017).

#### *4.2.1.6 Métricas de conexão*

Para monitoramento do dispositivo com relação a sua qualidade de conexão na rede, a métrica escolhida é o monitoramento do *Round Time Trip* (RTT) entre o dispositivo e o servidor. Esse parâmetro tem forte influência no tempo de execução, pois quando solicitado o processo de *offloading*, é necessário que o REE receba do cliente os parâmetros do método que está sendo demandado para processamento, e, ao fim da execução, transfira o resultado de volta para o cliente.

O valor do RTT é medido entre o servidor e o REE, porém o processo de *offloading* é realizado diretamente entre o cliente e o REE. Nesse trabalho, considerou-se que o servidor é instalado junto do ponto de acesso à rede, tendo em vista que todo tráfego de rede passa pelo ponto de acesso e que a tendência é a utilização de novos modelos de ponto de acesso, como os desenvolvidos para *Internet of Things* (IoT) que estão em ascendência permitindo uma rápida conexão entre os dispositivos, assim como o já comercial da Samsung[1]. Portanto, a taxa de RTT entre o cliente e o REE é praticamente a mesma entre o REE e o servidor. Assume-se este fato como verdade, pois seria muito mais custoso monitorar a taxa de RTT entre cada dispositivo cliente a medida que aumentasse o número de clientes e REEs.

Para aferição do valor de RTT é estabelecida uma comunicação com o servidor, na qual são enviados 10 objetos para o servidor que os devolve para o REE, armazenando-se seu tempo de envio e recebimento da resposta do REE. Logo após, para cada objeto é calculada a taxa de transferência através da subtração do tempo final pelo tempo inicial. Por fim, é calculada a média entre o tempo decorrido do envio dos objetos, considerando-se esse tempo como a taxa de RTT do momento.

#### *4.2.1.7 Número de Processos*

Um dispositivo quanto atuando como REE pode atender a mais de uma requisição de *offloading*. Para avaliar o número de requisições que um dispositivo está processando, um

[1] https://samsung-networks.com/products/iot-access-points

contador (chamado de NProcessos) mantém o mapeamento do número de requisições enviadas para cada REE, de modo que o algoritmo de seleção possa levar em consideração a carga de trabalho de cada um no processo de seleção.

### *4.2.2 Algoritmo de Decisão*

Essa subseção tem por objetivo apresentar como os parâmetros previamente apresentados foram utilizados para selecionar o melhor dispositivo para atender requisições de *offloading* de acordo com o contexto dos dispositivos e a intenção do desenvolvedor da aplicação ao habilitar o método para *offloading*, resultando na criação do algoritmo de decisão.

#### *4.2.2.1 Filtros*

Um primeiro passo ao realizar a seleção do melhor REE a atender uma requisição de *offloading* é a remoção daqueles que no momento não se adéquam ao contexto do *offloading* da lista dos REEs disponíveis, como por exemplo aqueles que estão em desacordo com as as regras estabelecidas de participação na rede de *offloading* estipulada pelo proprietário do dispositivo ou aqueles que possam apresentar alto risco de não completar a tarefa estipulada por eventuais desconexões na rede.

Portanto, de forma a auxiliar no processo de decisão, filtros foram criados para a remoção dos REEs da lista dos candidatos a atender a uma requisição de *offloading*. Para isso, foram utilizados dois parâmetros previamente explanados.

Através do parâmetro de limite de bateria estipulado pelo proprietário do REE, apresentado na seção 4.2.1.3, um dos primeiros passos do algoritmo de decisão é verificar todos os REEs que estão abaixo deste limite para que sejam removidos da lista de dispositivos a serem escolhidos, respeitando assim a decisão dos usuários sobre sua participação.

Um segundo processo de filtragem utiliza o parâmetro de intensidade do sinal *Wi-Fi*, apresentado na seção 4.2.1.5, para filtrar todos os REEs que estejam próximos a "borda da rede", ou seja, que estejam próximos do limite de cobertura da rede *Wi-Fi* com o risco de se desconectar do servidor. Nesta proposta, esta é a forma de abordar a dinamicidade dos dispositivos, uma vez que através do monitoramento constante da intensidade do sinal *Wi-Fi* é possível verificar se o dispositivo está se distanciando do ponto de acesso a rede e com o valor de RSSI do dispositivo atribuir um nível de confiança ao dispositivo que impactará na sua seleção para atender a requisições de *offloading*.

Após a filtragem dos dispositivos pelos critérios previamente mencionados, um outro passo realizado pelo algoritmo de seleção é a verificação de qual a intenção do desenvolvedor da aplicação ao habilitar o *offloading*: economia de energia ou melhoria na velocidade de execução. Esta verificação é realizada consultando o valor do atributo ASAP incluído na assinatura do método. Dependendo de tal valor, um vetor de pesos específico é criado para cada situação e utilizado no método de tomada de decisão multicritério. Tais processos serão explicados a seguir.

#### *4.2.2.2 AHP*

O método *Analytic Hierarchy Process* (AHP) é uma abordagem utilizada para tomada de decisão baseada em multicritérios em que os atributos utilizados possuem uma relação de hierarquia entre si (SAATY, 1990). Tal método é bastante utilizado em diversas áreas como administração, políticas, economia, esportes e inclusive na área de tecnologia. Basicamente, a utilização do *Analytic Hierarchy Process* (AHP) ocorre em 4 passos:

I. Primeiramente escolhe-se os atributos que são relevantes para a tomada de decisão e realiza-se uma hierarquia de relevância entre eles.
II. Em seguida, prioriza-se um atributo sobre outro, fazendo-se uma comparação em pares sobre o valor no qual um atributo é mais importante que o outro. De acordo com a escala criada por Saaty (1990), o valor de relevância de um atributo sobre outro deve variar de 1 a 9;
III. Após a priorização, verifica-se sua consistência através da soma dos valores de cada coluna, criação de um auto vetor para cada atributo, normalização de auto vetor e soma dos valores cujo resultado não deve ser diferente de 1 para indicar consistência na decisão;
IV. Por fim, a média ponderada obtida no passo anterior determina a relevância de cada atributo.

Nesse trabalho, o método AHP foi utilizado para ajudar no balanceamento da comparação entre os atributos dos dispositivos e justificar matematicamente a escolha dos mesmos. O AHP nos fornece pesos que são utilizados em outro método de decisão multicritério, *Technique for Order Preference by Similarity to Ideal Solution* (TOPSIS). Portanto, os atributos coletados na seção 4.2.1 foram utilizados como os atributos de interesse no AHP. Em nossa abordagem a priorização dos atributos foi definida de acordo com a intenção da seleção, que varia entre a economia de energia ou tempo de processamento. No caso de economia de energia considerou-se a seguinte ordem entre os critérios: Bateria > NProcessos > Score > RTT > *Wi-Fi*, pois desta

forma a priorização dos dispositivos considera primordialmente o nível de bateria do dispositivo e, somente em seguida, os parâmetros que são responsáveis pela velocidade de processamento. Já no caso de velocidade de processamento a ordem considerada foi: NProcessos > Score > RTT > *Wi-Fi* > Bateria, onde desta forma os parâmetros responsáveis pela velocidade de processamento vêm em primeiro lugar em ordem de importância. Os valores números na escala de Saaty (1990) foram atribuídos através de diversas execuções do processo de seleção e observação de sua adequação.

Nas Tabelas 3 e 4 pode-se verificar as planilhas construídas com os valores para a priorização do tempo de processamento e do balanceamento energético, respectivamente. As planilhas foram elaboradas de acordo com os passos anteriores para implementação do método AHP utilizando os atributos de nossa solução, e demonstrando as relevâncias dos atributos entre si, além do vetor de pesos gerado.

Tabela 3 – Parametrização que prioriza o tempo de processamento.

| | **NProcessos** | **Score** | **RTT** | **RSSI** | **Bateria** | | **Autovetor** | | **Autovetor Normalizado** |
|---|---|---|---|---|---|---|---|---|---|
| **NProcessos** | 1 | 2 | 5 | 7 | 9 | | 3,629678 | | 0,445885377 |
| **Score** | 1/2 | 1 | 5 | 7 | 9 | | 2,750782 | | 0,337917926 |
| **RTT** | 1/5 | 1/5 | 1 | 3 | 9 | | 1,015511 | | 0,124749804 |
| **RSSI** | 1/7 | 1/7 | 1/3 | 1 | 9 | | 0,571986 | | 0,070265179 |
| **Bateria** | 1/9 | 1/9 | 1/9 | 1/9 | 1 | | 0,172427 | | 0,021181715 |
| **Soma** | 1,9540 | 3,4540 | 11,4444 | 18,1111 | 37,0000 | | 8,1404 | | 1 |

Fonte: Elaborado pelo autor

Tabela 4 – Parametrização que prioriza o balanceamento energético.

| | **Bateria** | **NProcessos** | **Score** | **RTT** | **RSSI** | | **Autovetor** | | **Autovetor Normalizado** |
|---|---|---|---|---|---|---|---|---|---|
| **Bateria** | 1 | 5 | 5 | 7 | 9 | | 4,359695 | | 0,54356335 |
| **NProcessos** | 1/5 | 1 | 2 | 5 | 7 | | 1,695218 | | 0,211358473 |
| **Score** | 1/5 | 1/2 | 1 | 5 | 7 | | 1,284735 | | 0,160179769 |
| **RTT** | 1/7 | 1/5 | 1/5 | 1 | 3 | | 0,443421 | | 0,055285431 |
| **RSSI** | 1/9 | 1/7 | 1/7 | 1/3 | 1 | | 0,237513 | | 0,029612978 |
| **Soma** | 1,6540 | 6,8429 | 8,3429 | 18,3333 | 27,0000 | | 8,0206 | | 1 |

Fonte: Elaborado pelo autor

#### *4.2.2.3 TOPSIS*

O método TOPSIS (*Technique for Order Preference by Similarity to Ideal Solution*) é um método que busca resolver problemas de decisão através da classificação das alternativas e da seleção que mais chega próxima de uma solução ideal (HWANG *et al.*, 1993). Este método é

bastante utilizado na literatura para resolver problemas de tomada de decisão nas mais diversas áreas, inclusive utilizando o método AHP para melhorar o processo de decisão (BANGUI *et al.*, 2017).

Em nosso trabalho, o TOPSIS foi utilizado com o auxílio do método AHP para a seleção do melhor dispositivo de acordo com os parâmetros que foram apresentados na seção 4.2.1. A realização do cálculo do melhor dispositivo ocorre de acordo com os seguintes passos:

I. Primeiramente, é criada uma matriz com os dispositivos e suas características em que cada coluna da matriz representa um atributo do dispositivo (Score, Bateria, *Wi-Fi*, Número de Processos, RTT);
II. Para cada coluna atribui-se um peso que foi calculado de antemão pelo método AHP, e previamente apresentadas nas Tabelas 3 e 4;
III. Define-se para cada atributo da matriz se este é benéfico ou não para a solução. Por exemplo, em nosso estudo, quanto maior o valor de bateria, melhor para a escolha daquele dispositivo. Portanto este é um valor considerado benéfico. Já para o valor de RTT, quanto maior o seu valor, pior será para o dispositivo, sendo então um valor não-benéfico;
IV. Realiza-se a normalização da matriz;
V. Cada valor normalizado da matriz é então multiplicado pelos pesos estabelecidos;
VI. Calcula-se o melhor e o pior valor ideal de cada atributo selecionando-se o valor mínimo ou máximo de cada atributo de acordo se este é benéfico ou não para solução;
VII. Para cada valor é então calculado a distância Euclidiana do pior e do melhor valor ideal obtidos no passo anterior;
VIII. Por fim, calcula-se a pontuação final de cada dispositivo de acordo com sua distância a solução ideal obtida no passo anterior. Com esta pontuação, pode-se criar uma classificação para saber qual o melhor dispositivo.

De forma a visualizar melhor o processo de utilização do TOPSIS com o AHP foi adaptada uma planilha apresentada no Apêndice A que exemplifica o processo de classificação de 5 dispositivos com seus atributos.

#### *4.2.2.4 Algoritmo de decisão*

De posse de todos os parâmetros necessários para a tomada de decisão, foi criado um algoritmo que utiliza os métodos e os filtros apresentados nas seções anteriores, conforme o fluxo descrito no Algoritmo 1.

**Algoritmo 1:** Seleção de REE para *offloading*

```
Function selecionarMelhorREE(REEs_Disponíveis, ASAP):
    REEs_Conectados = new List();
    for each dispositivo ∈ REEs_Disponíveis do
        if dispositivo.bateria < LIMITE_BATERIA then
            REEs_Disponíveis.remove(dispositivo);
        end
        if dispositivo.rssi < LIMITE_RSSI then
            REEs_Disponíveis.remove(dispositivo);
        end
        if dispositivo.conectado then
            REEs_Conectados.add(dispositivo);
        end
    end
    pesos = vetor_Pesos_Processamento;
    lista_REE = REEs_Disponíveis;
    if not ASAP then
        if listaDM_Conectados.size() > 0 then
            lista_REE = REEs_Conectados;
        end
        else
            pesos = vetor_Pesos_Bateria;
        end
    end
    return REE_escolhido = TOPSIS(pesos, lista_REE);
EndFunction
```

Para cada REE que se conecta com o servidor é criado um objeto para armazenar seus dados. Quando o cliente envia uma solicitação de *offloading* para o servidor, esse executa o algoritmo de seleção do melhor REE para atender a requisição, passando como parâmetros de entrada, a lista de dispositivos conectados naquele momento e o valor de ASAP advindo da solicitação de *offloading*. O algoritmo filtra a lista recebida, removendo os dispositivos que já atingiram o limite de bateria estabelecido pelo usuário da aplicação e removendo os dispositivos que estão com a intensidade de sinal *Wi-Fi* abaixo de 30, por considerar estes muito próximos a uma desconexão. Cada REE que estiver com nível de bateria igual a 101 é adicionado a uma nova lista para armazenar todos os que estão conectados a uma fonte de energia.

Após o processo de filtragem, o algoritmo verifica o valor de ASAP (verdadeiro ou falso) para decisão de qual vetor de pesos criados pelo método AHP este vai carregar no método TOPSIS para seleção do melhor dispositivo. Caso o valor seja verdadeiro, o desenvolvedor da aplicação solicitou velocidade de processamento, portanto os pesos relacionados são atribuídos ao TOPSIS que utiliza tais pesos e a lista filtrada de REEs disponíveis para a seleção do melhor dispositivo. Caso o valor seja falso, o algoritmo verifica se existe algum dispositivo conectado a energia, se existir somente um, este é o escolhido, se não existir nenhum, o método TOPSIS é executado com os pesos atribuídos a economia de energia e a lista filtrada de REEs, se existir

mais de um, o método TOPSIS utiliza a lista de pesos de velocidade de processamento na lista de REEs que estão conectados a energia.

## 4.3 Infraestrutura para realização de *offloading* entre múltiplos dispositivos móveis

Para habilitar a utilização do algoritmo apresentado na seção anterior e viabilizar a execução de *offloading* entre múltiplos dispositivos móveis, foi criada uma arquitetura com base nos projetos CAOS e CAOS D2D. Nessa seção é apresentada tal arquitetura através da explicação de seus módulos e as adaptações realizadas de seus componentes, além da apresentação dos novos componentes que foram necessários para dar suporte ao *offloading* entre dispositivos móveis. Em seguida, é demonstrado como ocorre a comunicação no processo de *offloading* através de um Diagrama de Sequência UML e é apresentado um processo essencial para viabilizar o *offloading* entre dispositivos, chamado de *Deploy* de dependências. Por fim, são apresentados como foram enfrentados desafios em tempo de desenvolvimento a respeito de limitações impostas pelo sistema Android.

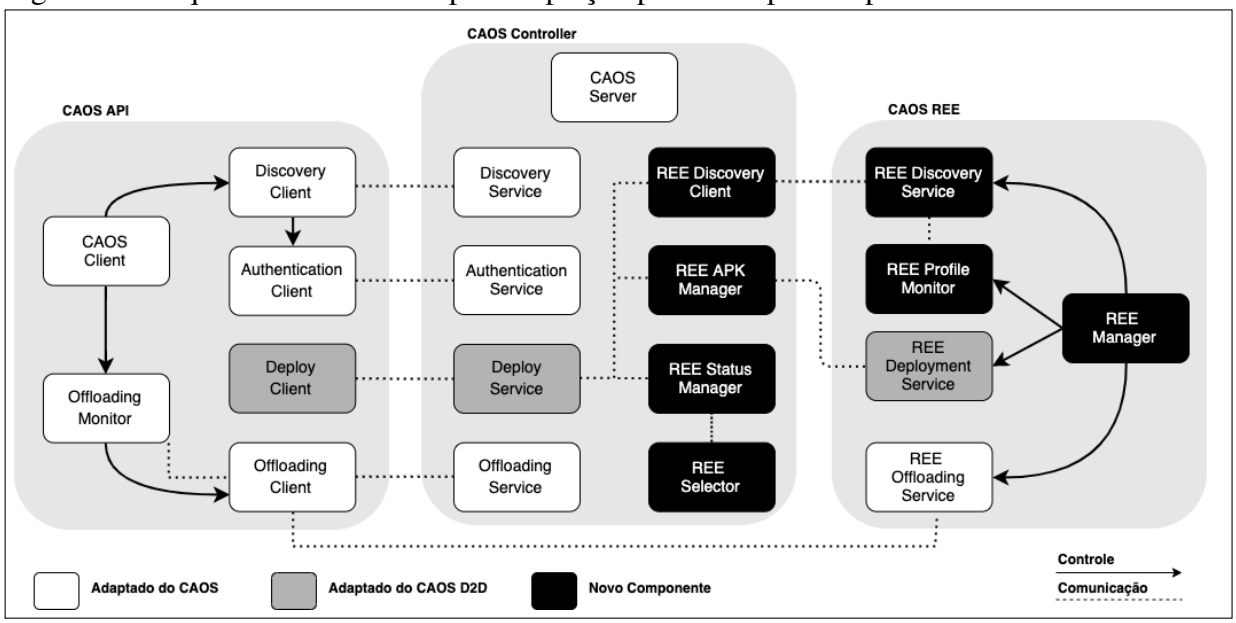
### *4.3.1 Módulos do Sistema*

A arquitetura do sistema proposto após a adaptação para o *offloading* com múltiplos dispositivos passou a ser composta por três componentes que se comunicam entre si para o funcionamento do sistema. A Figura 8 apresenta tais componentes, sendo estes o *CAOS API*, *CAOS Controller* e o CAOS REE.

O *CAOS API*, como apresentado na seção 2.1.5, é o responsável pela customização da aplicação do usuário para a comunicação com o servidor. Neste componente foi adicionado o módulo *Deploy Client* que é responsável pelo envio das dependências da aplicação para o servidor. Tal módulo foi adaptado do CAOS D2D para também permitir o *deploy* de dependências dos aplicativos para múltiplos REEs. Os demais módulos do CAOS API, em cinza, sofreram pequenas modificações de modo a adaptar ao novo ambiente de execução, com destaque para o módulo *Offloading Client* que agora, além de se comunicar com o seu par no componente Servidor, envia diretamente requisições de *offloading* para o componente *REE Offloading Service*.

O componente Servidor agora possui um módulo chamado *Deploy Service*, responsável por receber as dependências de uma aplicação cliente. Este módulo se comunica com um novo módulo chamado *REE APK Manager*, que faz o trabalho de detecção das dependências

Figura 8 – Arquitetura do CAOS após adaptação para múltiplos dispositivos




Fonte: Elaborado pelo autor

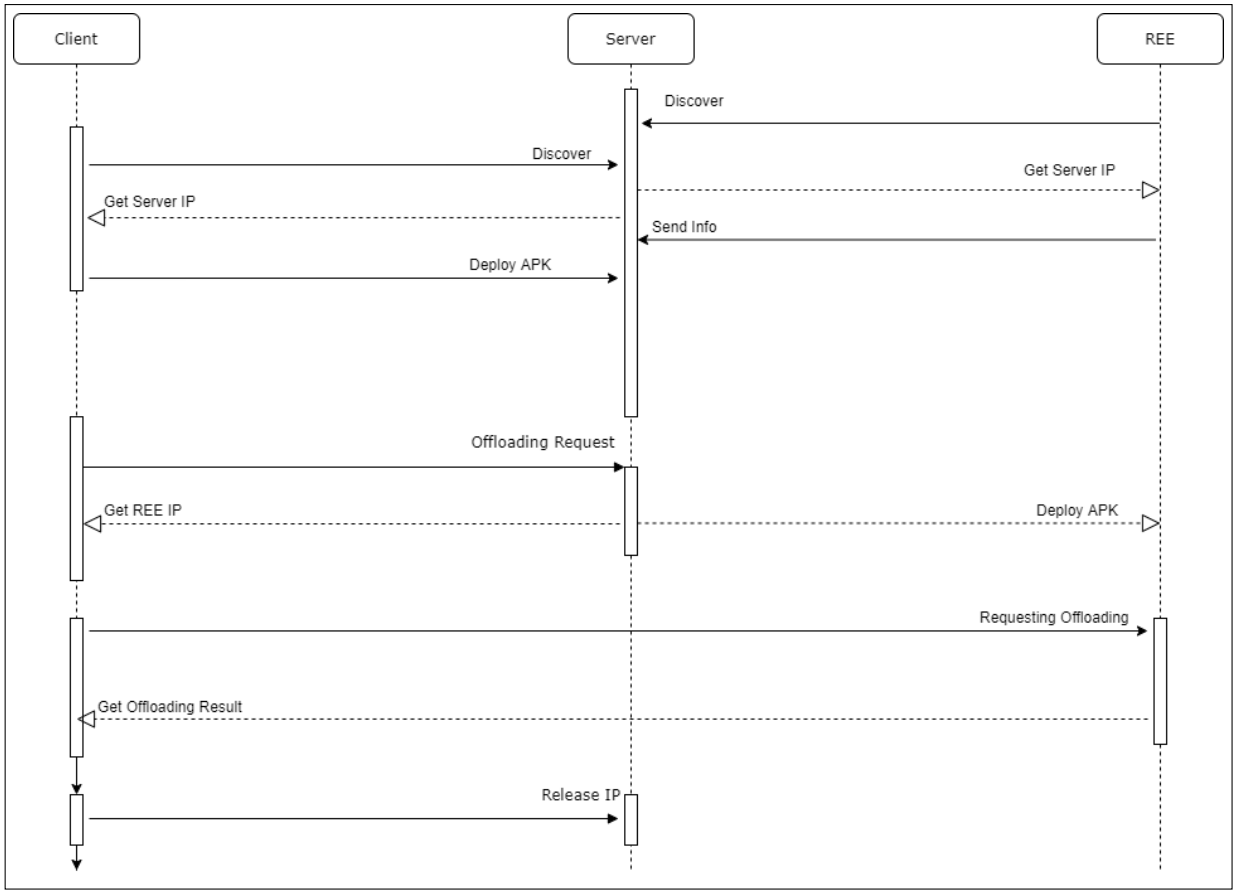
e seu envio para o REE correto, além de mapear as dependências que os REE possuem para evitar envios desnecessários, esse processo é muito importante para a execução do processo de *offloading* e será mais detalhado na seção 4.3.2. Para a habilitar as descobertas dos REEs foi criado o módulo *REE Discovery Service*, baseado no *Discovery Service*, porém dedicado somente aos REEs. Este módulo, após realizar a detecção dos REEs, invoca o módulo *REE Status Manager* que é responsável por monitorar e atualizar as métricas de cada REE. Por fim, foi adicionado um novo módulo para ser o responsável pela seleção do melhor dispositivo para atender requisições de *offloading*, chamado de *REE Selection Module*.

O componente CAOS REE é uma aplicação a ser executada nos dispositivos dos usuários que forem se candidatar como REEs, ie, ambientes de execução das operações de *offloading*. Para isto, este componente implementa o módulo *REE Manager*, responsável pela orquestração dos outros módulos e interação da interface do usuário com o sistema. O módulo *REE Discovery Service* possui função similar ao *Discovery Client* do componente *CAOS API*, gerenciando a comunicação com o servidor. O módulo *REE Profile Monitor* é o responsável pela obtenção, atualização, e envio a cada 60 segundos dos atributos, apresentados na seção 4.2.1, a respeito do REE para o *CAOS Controller*. O módulo *REE Deployment Service* tem a função de receber as dependências necessárias das aplicações que solicitarão o processo de *offloading*. Por fim, o módulo *REE Offloading Service* atende a requisições de *offloading* e retorna seu resultado aos clientes.

Para visualização do processo de *offloading* através da infraestrutura criada neste

trabalho apresentamos o diagrama de sequência em UML na Figura 9.

Figura 9 – Diagrama de sequência do processo de *offloading* entre os componentes da arquitetura.




Fonte: Elaborado pelo autor

Inicialmente tanto a aplicação cliente como o REE buscam na rede um servidor *CAOS Controller* ativo. Ao detectar a solicitação de comunicação, o *CAOS Controller* responde a ambos com seu *Internet Protocol* (IP) para futuras comunicações. Em seguida, a aplicação cliente envia sua dependência, caso o servidor tenha comunicado sua necessidade, e o CAOS REE inicia o envio de seus parâmetros para conhecimento do servidor sobre o contexto do dispositivo. Quando a aplicação cliente solicita o processo de *offloading*, o servidor através do algoritmo proposto neste trabalho (subseção 4.2.2.4) decide o melhor REE para atender a tal requisição, retornando o IP do REE para aplicação cliente. Ao mesmo tempo em que esse processo ocorre, o servidor envia ao REE selecionado a dependência da aplicação, se necessária, para que este já esteja preparado quando a solicitação chegar. De posse do IP do REE que irá atender a sua requisição, o cliente solicita diretamente o processo de *offloading* para o IP do REE selecionado. Após finalizado a execução remota do processamento, o REE envia a resposta diretamente ao cliente. Tendo finalizado o *offloading*, o cliente avisa ao servidor que o REE

utilizado está liberado para novas conexões.

### *4.3.2 Implantação de Dependências*

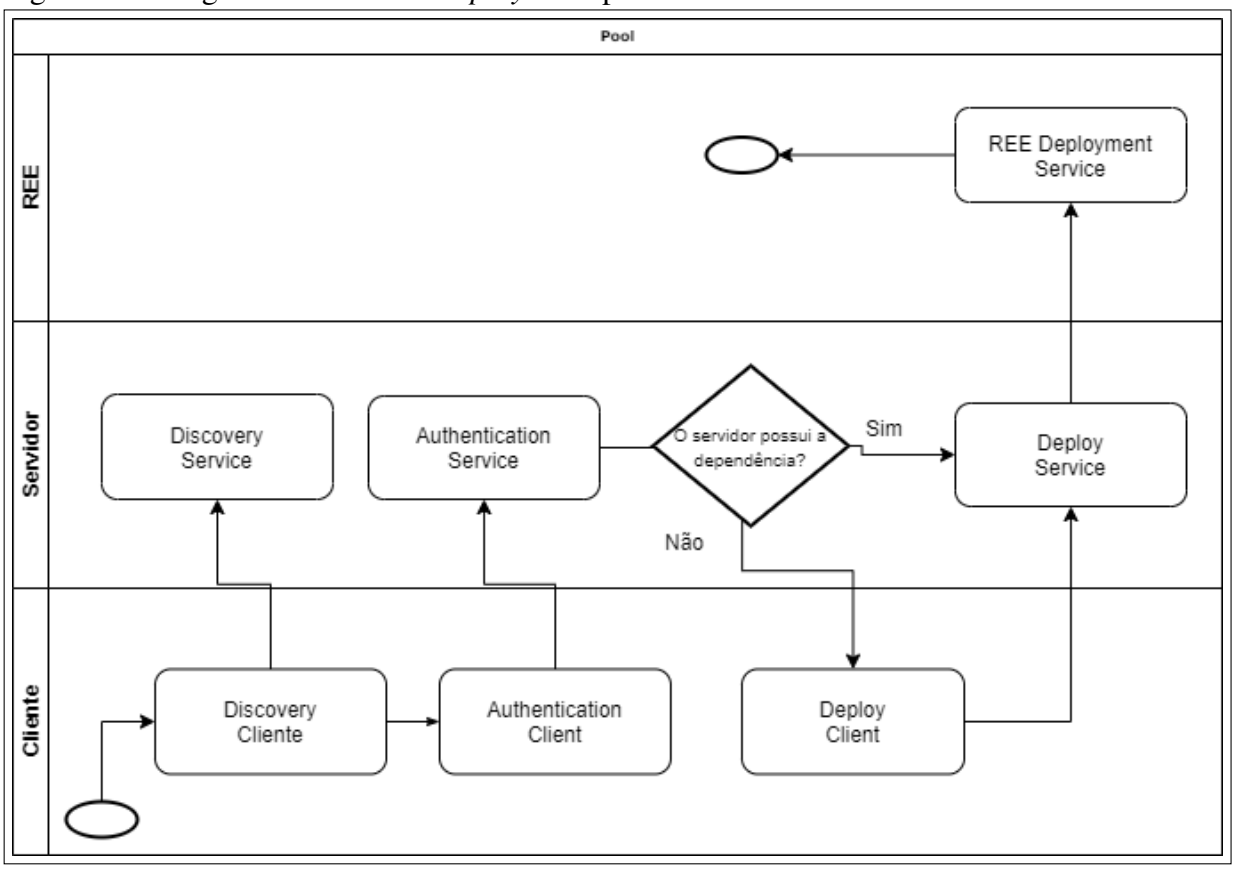
Para possibilitar o processo de *offloading*, é necessário que o REE possua previamente a aplicação do cliente que realiza a requisição de *offloading*. Desta forma, uma estratégia de *deploy* de tais aplicações para os REEs é necessária, de modo a entregar tais dependências o mais rápido o possível para que ao ocorrer uma requisição de *offloading*, o REE esteja pronto para atendê-la. Tal implementação é crucial, pois a transferência da aplicação para o REE impacta a velocidade na qual o processo *offloading* é concluído.

A estratégia implementada em nosso estudo utilizou como passo inicial o processo de *deploy* de dependências presente no CAOS D2D, apresentado na subseção 2.1.6, adaptando a implementação para o modelo cliente servidor e solucionando problemáticas para sua utilização em um ambiente com múltiplos dispositivos. O diagrama de fluxo da Figura 10 exemplifica a sequência de funcionamento do *deploy* que ocorre através dos seguintes passos:

- Ao inicializar a aplicação, o módulo *DiscoveryClient* verifica se existe um servidor de tomada de decisão para realização de *offloading* na mesma WLAN que o cliente;
- Após a conexão com o servidor, o módulo *AuthenticantionClient* inicializa o envio de informações do cliente ao servidor;
- O servidor através do módulo *AuthenticationServer* do cliente verifica se a dependência do aplicativo com o qual está se comunicando já está presente em seu repositório de aplicações;
- Caso o servidor não possua a dependência, é enviada a solicitação ao módulo *DeployClient*. O cliente então responde a requisição ao módulo *DeployService*. Este envio ocorre apenas uma vez, pois o servidor armazena a dependência que servirá para caso outros clientes com a mesma aplicação se conectarem;
- A partir do recebimento da dependência, o servidor dispara envios para os múltiplos REEs que possam estar presentes, para que no futuro se esses REEs forem os escolhidos para atender a solicitação de *offloading* do cliente, esses já estejam de posse das dependências necessárias.

O processo descrito para deploy ocorre na primeira conexão do cliente com o servidor e o envio da dependência ocorre somente para os REEs que estejam presentes naquele momento. Porém, para adaptar o envio de dependências a dinamicidade dos dispositivos na rede, foi

Figura 10 – Diagrama de fluxo do *deploy* de dependências.




Fonte: o autor.

necessário criar um mapeamento dos REEs de acordo com suas dependências, pois deste modo evita-se que o servidor fique enviando dependências para dispositivos que já as possuam.

O mapeamento é realizado no primeiro contato da aplicação cliente com o servidor, onde após armazenar e enviar as dependências para os REEs disponíveis, o servidor cria uma estrutura de *HashMap* que possui como chave o IP do REE e como atributos as dependências que este possui. Desta forma, para cada novo REE que se conecta com o servidor, é criada pelo servidor uma lista de dependências já enviadas a aquele REE, de modo a enviá-las somente uma única vez. Com isso, é possível que o servidor possa ter ciência se será necessário ou não o envio da dependência ao REE que irá atender a uma requisição, evitando assim atrasos no tempo de execução.

### *4.3.3 Limitações do sistema Android*

Na criação da aplicação CAOS REE foram enfrentados desafios para seu funcionamento de acordo com a proposta definida nesta dissertação de mestrado.

Um desses desafios foi a variação do valor de *score* dos dispositivos ao estarem

sendo manipulados pelo usuários, que mostrou-se bastante ampla. Essa variação indicou que os dispositivos estavam tendo dificuldade para executar processos na aplicação CAOS REE, ao lidar com as aplicações dos usuários. Isso acarretou em uma demora excessiva na resposta dos dispositivos se comparado quando os mesmos estavam em repouso, além de gerar um grande número de exceções provocadas por falha de comunicação e *Timeout Exceptions*. Esse problema inviabilizou completamente a execução de processos de *offloading*, uma vez que exceções ocasionadas no processo fazem com que a execução tenha que ser reiniciada pelo dispositivo cliente, ocasionando maior tempo para finalização da atividade e desperdício de energia.

Outro grande desafio foi o novo modo de economia de energia introduzido pelo Android 6.0 chamado de *Doze Mode*, e que faz com que após um período de tempo em que o usuário esteja sem interagir com o dispositivo, esse adie atividades de CPU e conexões de rede (DEVELOPERS, 2015). Além disso, no Android 8.0 foram inseridas algumas limitações para processos em *background* de modo a melhorar a experiência do usuário (DEVELOPERS, 2017). Tais imposições tiveram grande impacto no serviço de monitoramento de contexto da aplicação, que executava em *background*, pois ao invés de se comunicar com o servidor a cada 60 segundos, se o dispositivo entrasse em modo *Doze* a comunicação com servidor era interrompida. Além disso, neste modo o dispositivo passava a não atender requisições de *offloading*, devido a interrupção do *socket* responsável por esperar por essas solicitações.

Para contornar essas questões, uma nova implementação do CAOS REE foi desenvolvida, de forma a garantir que o dispositivo pudesse atender requisições de *offloading*, independente de estar sendo manipulado pelo usuário ou estar em *Doze Mode*, além de garantir que a resposta fosse enviada em tempo hábil. Para isso, o meio utilizado para prover tal funcionalidade foi através da implementação do CAOS REE utilizando *Foreground Service* e *Partial Wake Lock*.

Um *Foreground Service* permite que um serviço seja executado, mesmo que o usuário não esteja interagindo com a aplicação e não o finaliza quando o sistema está com pouca memória. Porém, esta abordagem tem como regra a implementação de uma notificação para o usuário de que o serviço está ativo (SERVICES. . . , 2019). Já o *Partial Wake Lock* faz com que a CPU continue a executar os processos da aplicação, mesmo quando o dispositivo está com a tela desligada (POWERMANAGER. . . , 2009). Dessa forma, com a utilização desses dois recursos do Android, foi possível implementar na aplicação CAOS REE, um serviço de monitoramento de contexto constante e manter um *socket* sempre aberto para o atendimento de requisições de

*offloading*, resolvendo as problemáticas elencadas anteriormente.

## 4.4 Considerações Finais

Nesse capítulo foi apresentada a solução para para escolha otimizada de um dispositivo para atender a requisições de *offloading*, com objetivo de se escolher o melhor dispositivo de acordo com seu contexto de execução, e tomando em consideração decisões tanto do usuário final, quanto do desenvolvedor da aplicação móvel.

A arquitetura apresentada e o algoritmo construído foram implementados para a criação de um sistema que possibilite tornar real o cenário de *offloading* para múltiplos dispositivos, habilitando assim os testes que serão apresentados na próxima seção para demonstrar a avaliabilidade do método proposto.

# 5 EXPERIMENTOS REALIZADOS

Este capítulo apresenta os experimentos realizados para verificar a viabilidade da abordagem proposta por meio de execuções em dispositivos reais utilizando-se da arquitetura apresentada no capítulo anterior. Os experimentos realizados tiveram como objetivos:

- Avaliar a eficácia do algoritmo de seleção quando o objetivo geral do *offloading* é a seleção de um dispositivo que realize determinada tarefa no menor tempo o possível;
- Avaliar a adaptação do algoritmo de seleção em relação às mudanças de contexto dos dispositivos envolvidos na realização de operações de *offloading*;
- Avaliar a adaptação do algoritmo de seleção em relação à dinamicidade dos dispositivos;
- Avaliar desempenho do algoritmo de seleção quando o objetivo é o balanceamento de consumo energético.
- Avaliar o consumo energético da aplicação CAOS REE ao ser utilizada durante o atendimento de requisições de *offloading*.

Os experimentos foram realizados com um conjunto de dispositivos apresentados na Tabela 5. Entretanto, por questões de disponibilidade, os dispositivos foram divididos em dois conjuntos utilizados em ambientes diferentes. No primeiro conjunto estão os dispositivos {A,B,C,D}, e no segundo conjunto estão os dispositivos {C,D,E,F}. Para cada conjunto de dispositivos foi realizada a mesma bateria de testes. Isto ajudou a obter métricas mais fidedignas com relação aos experimentos.

Tabela 5 – Dispositivos utilizados nos experimentos.

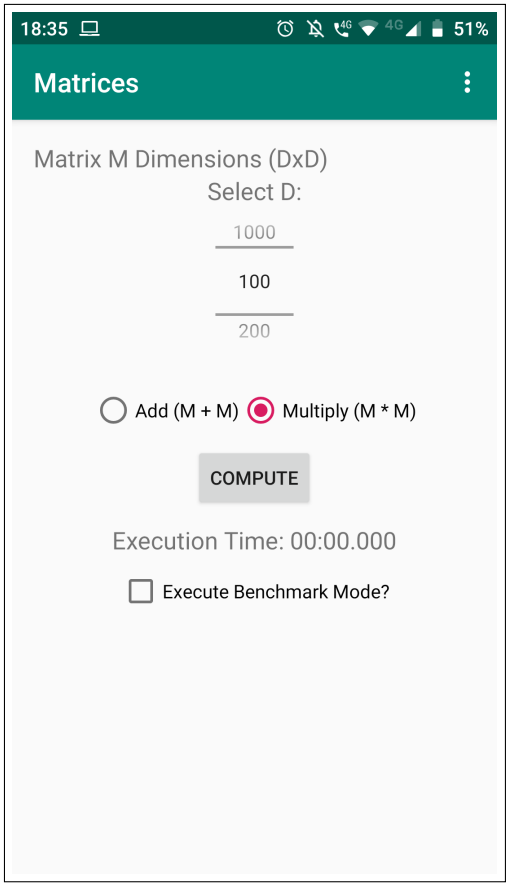
| **Dispositivo** | **Modelo** | **Android** | **Processador** | **Memória** | **Disco** |
|---|---|---|---|---|---|
| Servidor | Dell Inspiron 5557 | - | Intel Core I7 2.5GHZ | 8 GB | 1 TB |
| Cliente | Galaxy S3 Slim Duos | 4.2.2 | Quad-Core 1.2GHZ | 1 GB | 64 GB |
| Dispositivo A | Xiaomi Mi 8 Lite | 9.0 | Octa-Core 2.2GHZ | 6 GB | 128 GB |
| Dispositivo B | Galaxy J7 Neo | 9.0 | Octa-Core 1.3GHZ | 2 GB | 16 GB |
| Dispositivo C | Moto Z2 Play | 9.0 | Octa-Core 2.2GHZ | 4 GB | 64 GB |
| Dispositivo D | Tablet Samsung SM-T560 | 4.4.4 | Octa-Core 1.3GHZ | 1,5 GB | 8 GB |
| Dispositivo E | One Plus 5T | 9.0 | Octa-Core 2.2GHZ | 8 GB | 128 GB |
| Dispositivo F | Xiaomi Pocophone F1 | 9.0 | Octa-Core 2.3GHZ | 6 GB | 128 GB |

Fonte: Elaborado pelo autor

Para a realização dos experimentos foi utilizada uma aplicação móvel que permite a realizações de operações de adição ou multiplicação de matrizes aleatórias com dimensões entre 100x100 e 1000x1000, de acordo com a seleção do usuário ( Figura 11). Esta aplicação mostra-se interessante para a execução dos testes de *offloading*, pois conforme o tamanho da matriz selecionada e a operação escolhida, maior o poder de processamento exigido para encontrar a

solução. Além disso, quanto maior a matriz, maior é o seu tempo de transferência pela rede, impactando no tempo de resposta.

Figura 11 – Aplicação de operações com matrizes




Fonte: o autor.

Nos experimentos realizados, foi utilizada a operação de multiplicação entre matrizes, pois este tipo de operação demanda uma maior processamento do dispositivo cliente. Após execuções prévias, foi decidido utilizar as dimensões 600x600 para a multiplicação das matrizes pois constatou-se que, para o cliente utilizado nos testes (Tabela 5), a média do tempo de execução local para matrizes dessa dimensão é de aproximadamente 38 segundos e espera-se uma redução desse tempo ao realizar *offloading* para outros dispositivos.

De modo a obter confiabilidade estatística em relação aos resultados dos testes, para cada teste, foram realizadas 30 execuções.

Nas seções seguintes, cada um dos experimentos é melhor detalhado e seus resultados são discutidos.

## 5.1 Experimentos de eficiência do algoritmo de seleção quando o objetivo é diminuir o tempo de processamento

O objetivo desse experimento é verificar se a abordagem proposta neste trabalho realiza corretamente a escolha do dispositivo que melhor atende a uma requisição de *offloading*, de acordo com o tempo de resposta de cada dispositivo. Espera-se verificar se a solução consegue contornar o problema de heterogeneidade e acertar qual o dispositivo, dentre os disponíveis, terá o maior poder de processamento naquele momento, e, por tanto, responderá mais rápido a uma solicitação de *offloading*.

Para isso, foi criada uma variação da API do CAOS que possui a capacidade de enviar solicitações de *offloading* para múltiplos dispositivos de forma simultânea. O servidor envia para o cliente o endereço IP de todos os dispositivos que se oferecem como REEs no ambiente. A aplicação cliente dispara requisições de *offloading* para todos os IPS recebidos. Ao obter os resultados de cada um dos pedidos de *offloading*, o tempo contabilizado do envio do pedido ao recebimento do resultado é armazenado. Com estes dados foi possível verificar se o dispositivo escolhido pelo algoritmo de seleção foi realmente aquele com maior capacidade de processamento, e consequentemente, com capacidade de processar a solicitação do cliente mais rapidamente.

Para verificar a taxa de acerto do algoritmo ao escolher o dispositivo ideal para atender uma requisição de *offloading*, foi elaborado um teste de predição que funciona de acordo com os seguintes passos:

I. O *CAOS Controller* que faz o papel de servidor (apresentado na Tabela 5) é inicializado e fica em aguardo de aplicações e REEs;
II. A aplicação CAOS REE é ativada em todos os dispositivos que serão utilizados como REEs para que se identifiquem com o servidor;
III. A aplicação de multiplicação de matrizes é inicializada no dispositivo cliente para que se identifique com o servidor e envie sua dependência;
IV. As dependências necessárias para a execução dos métodos da aplicação de multiplicação de matrizes é enviada para todos os REEs disponíveis;
V. O cliente realiza uma operação de multiplicação de duas matrizes aleatórias de dimensão 600x600. Esta ação dispara uma solicitação de *offloading* para o servidor;
VI. O servidor recebe o pedido de execução e realiza então a predição baseada no algoritmo criado para identificar quem será o melhor dispositivo a atender a requisição;

VII. A aplicação cliente recebe a lista de todos os REEs disponíveis e dispara requisições de *offloading* simultaneamente para estes dispositivos;

VIII. A aplicação cliente aguarda até todos os dispositivos REEs responderem, e armazena seu tempo de resposta;

IX. Por fim, se o dispositivo escolhido pelo servidor foi o mais rápido a responder, é pontuado um acerto e se o servidor acertou a ordem de resposta entre todos os dispositivos, pontua-se um acerto perfeito.

Para verificar a correta execução do algoritmo durante os experimentos, foram criados *logs* de acompanhamento para as diversas execuções realizadas. A Figura 12 representa o *log* do *CAOS Controller* onde são apresentados os valores dos atributos de cada dispositivo que atua como REE. Destas informações, destaca-se o valor calculado de desempenho para cada dispositivo, por meio do algoritmo de seleção.

Figura 12 – Log no *CAOS Controller* ao receber uma requisição de *offloading*

```
1  TopsisCAOS class ---> Score: 3 Batery: 33 Wifi 100 RTT: 63.3
2  TopsisCAOS class ---> Ip: 192.168.1.110 Performance: 0.9450664667473306
3
4  TopsisCAOS class ---> Score: 17 Batery: 70 Wifi 82 RTT: 57.5
5  TopsisCAOS class ---> Ip: 192.168.1.114 Performance: 0.10382496500844467
6
7  TopsisCAOS class ---> Score: 5 Batery: 35 Wifi 100 RTT: 63.4
8  TopsisCAOS class ---> Ip: 192.168.1.113 Performance: 0.849123905404376
9
10 TopsisCAOS class ---> Score: 11 Batery: 96 Wifi 33 RTT: 56.5
11 TopsisCAOS class ---> Ip: 192.168.1.112 Performance: 0.4232376804637394
12
13 The best cloudlet will be: 192.168.1.110
```

Fonte: o autor.

De acordo com os valores obtidos, aquele de maior valor absoluto é predito como o melhor dispositivo a atender a requisição de *offloading*. No exemplo da 12 a linha 13 demonstra que o dispositivo predito como o melhor será o de IP 192.168.1.110, devido ao seu valor de desempenho (linha 2) ser o maior entre os REEs disponíveis.

Já a Figura 13 representa o *log* definido para o aplicativo cliente. Nele demonstra-se o envio da requisição de *offloading* para cada dispositivo REE, marcando seu tempo de envio inicial, e, logo após, o tempo decorrido até a chegada da resposta.

De modo a garantir uma “corrida” justa foram implementados métodos para realizar o envio das requisições de *offloading* com a mínima diferença de tempo de inicialização o

Figura 13 – Log na aplicação cliente ao solicitar uma requisição *offloading*

```
1  Sending Offloading Request to Cloudlet: 192.168.1.110 Initial Time: 1573075277453
2  Sending Offloading Request to Cloudlet: 192.168.1.114 Initial Time: 1573075277454
3  Sending Offloading Request to Cloudlet: 192.168.1.113 Initial Time: 1573075277455
4  Sending Offloading Request to Cloudlet: 192.168.1.112 Initial Time: 1573075277455
5
6  Ip: 192.168.1.110 Response Time: 17024
7  Ip: 192.168.1.113 Response Time: 19689
8  Ip: 192.168.1.112 Response Time: 30961
9  Ip: 192.168.1.114 Response Time: 36788
```

Fonte: o autor.

possível. Nas linhas 1 a 4 da Figura 13, é possível observar que a diferença de tempo de início da execução entre cada dispositivo é inferior a 12 casas decimais, desse modo nenhum dispositivo foi favorecido.

Percebe-se pela Figura 13 que a previsão do algoritmo foi adequada ao selecionador o IP 192.168.1.110, pois ele foi o mais rápido a responder a requisição de *offloading*. Portanto nos testes contaríamos um acerto. Além disso, os valores do cálculo do desempenho para cada dispositivo estão na mesma ordem em que os dispositivos responderam a requisição, portanto nesse caso contaríamos um acerto perfeito.

Para demonstrar a adaptação ao contexto, os testes foram divididos em duas categorias. Na primeira os dispositivos REEs foram configurados de modo a executar apenas a aplicação CAOS REE, e sem manuseio de qualquer usuário. Já na segunda categoria, usuários ficaram livres para utilizar os dispositivos REEs, podendo disparar a execução concorrente de outras aplicações. Desta forma, foi possível verificar se haveria impacto na corretude de seleção de acordo com a utilização ou não do dispositivo.

### *5.1.1 Experimentos com dispositivos dedicados*

Para este experimento, todos os dispositivos estavam lado a lado, sem interferência do usuário, sem executar aplicações, e a mesma distância do ponto de acesso a rede. Sobre esse último requisito é interessante comentar que apesar de estarem lado a lado, a uma mesma distância do ponto de acesso, os dispositivos apresentaram valores de intensidade de sinal Wi-Fi diferentes em algumas ocasiões, demonstrando assim a importância do parâmetro monitorado.

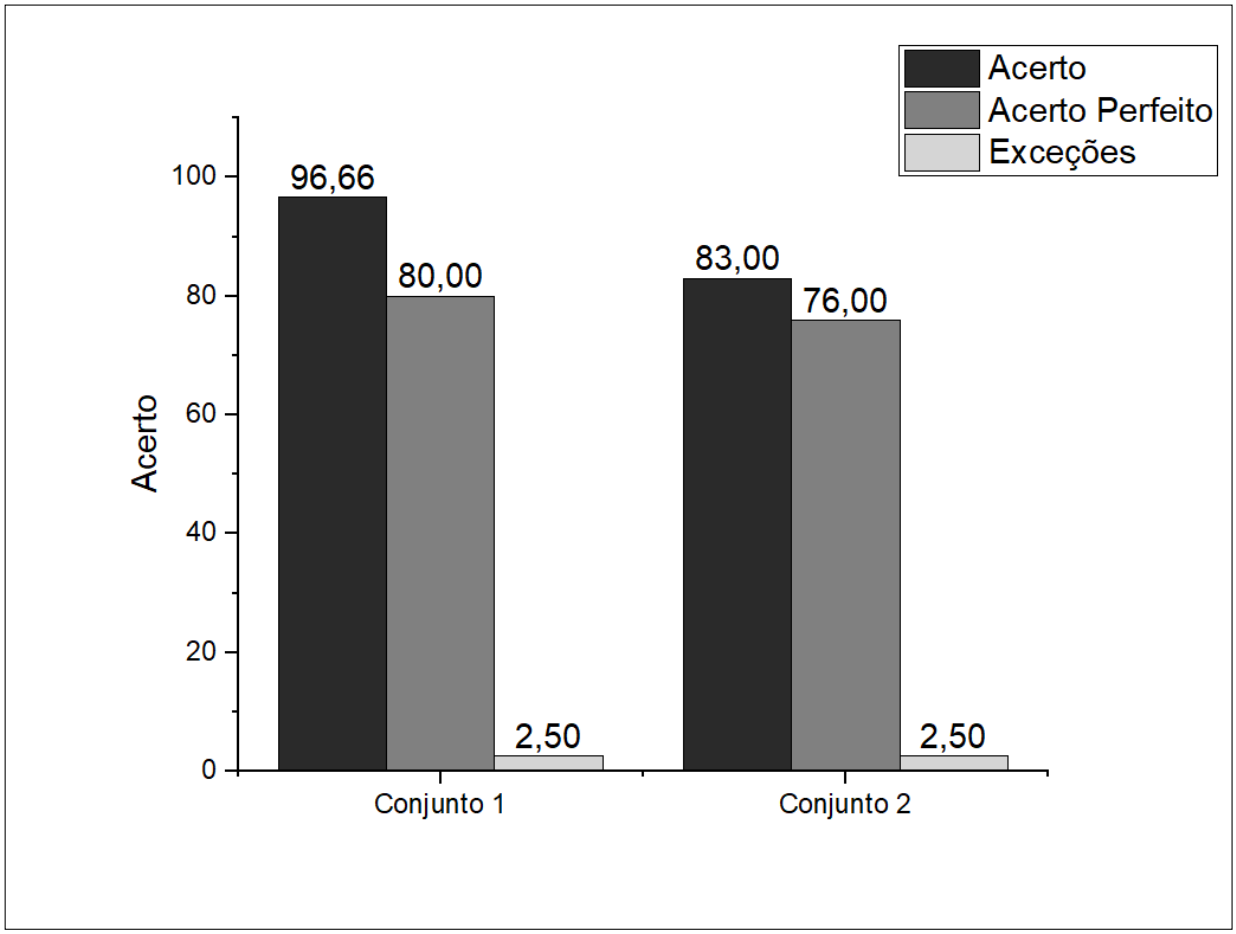
Na primeira bateria de testes, os dispositivos utilizados foram os dispositivos do conjunto {A,B,C,D} da Tabela 5. Como resultado obteve-se 96,66% de acerto no melhor dispositivo a atender a requisição de *offloading*; 80% de acerto perfeito, em que se acerta a

sequencia de tempo de resposta dos dispositivos, e; 2,5% de exceções ocasionadas por falha na comunicação entre os dispositivos.

Na segunda bateria de testes, os dispositivos presentes eram os do conjunto {C,D,E,F} descritos na Tabela 5. Este teste foi mais desafiador, pois os dois primeiros dispositivos da tabela possuíam poder de processamento bem similar, de modo a deixar a disputa bem acirrada. Como resultado obteve-se 83% de acerto no melhor dispositivo a atender a requisição de *offloading*; 76,6% de acerto perfeito, em que se acerta a sequencia de tempo de resposta dos dispositivos, e; 2,5% de exceções levantadas por algum gatilho nas conexões.

A Figura 14 ilustra os valores adquiridos de cada teste. A diminuição da porcentagem de acertos neste segundo conjunto de dispositivos é atribuída a similaridade entre os dispositivos E e F da Tabela 5. Entretanto, em todos os casos em que o algoritmo não acertou o melhor dispositivo, 5 vezes das 30 execuções (17%), o dispositivo selecionado foi aquele com o segundo melhor desempenho. A diferença média de tempo de execução entre o primeiro dispositivo a responder e o dispositivo selecionado pelo algoritmo foi 0,7 segundos.

Figura 14 – Taxa de acertos, acertos perfeitos e exceções quando utilizados dispositivos dedicados.




Fonte: o autor.

### *5.1.2 Experimentos com dispositivos não-dedicados*

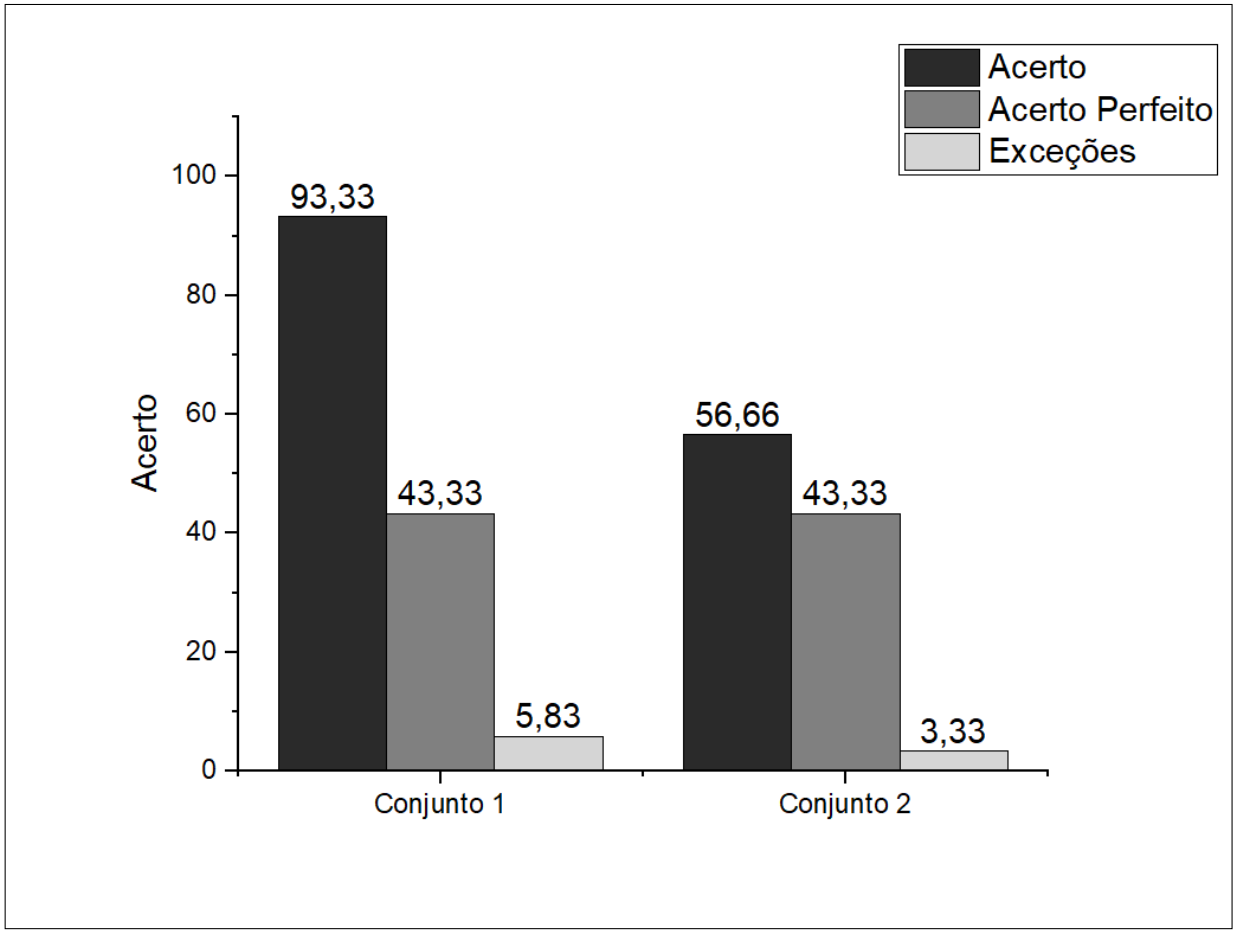
Para esse experimento os usuários ficaram livres para utilização dos dispositivos de modo a verificar o funcionamento do sistema em um ambiente real, no qual a utilização do dispositivo pelo usuário é imprevisível. As aplicações reportadas que foram utilizadas durante a execução das requisições de *offloading* foram Instagram, Facebook, Whatsapp, Twitter, Deezer e Youtube. Todas essas aplicações possuem bastante impacto no processamento do dispositivo e necessitam de tráfego de dados.

Na primeira bateria de testes com os dispositivos do conjunto {A,B,C,D} da Tabela 5 obteve-se como resultado 93,33% de acerto na seleção do melhor dispositivo para atender a requisição de *offloading*, 43,33% de acertos perfeitos e 5,83% de exceções levantadas por algum gatilho nas conexões. Já na segunda bateria de testes com os dispositivos do conjunto {C,D,E,F} da Tabela 5 obteve-se 56,66% de acerto no melhor dispositivo a atender a requisição de *offloading*, 43,33% de acerto perfeito e 3,33% de exceções levantadas por algum gatilho nas conexões.

A Figura 15 apresenta um gráfico que demonstra os valores obtidos nesse experimento com os dois conjuntos de dispositivos. A diminuição do percentual de acerto na segunda bateria de testes é mais uma vez atribuída a grande similaridade de poder de processamento dos dispositivos E e F da Tabela 5. Como o intervalo de aquisição do contexto do dispositivo é de 60 segundos, dentro desse tempo devido a atividade do usuário o contexto do dispositivo pode não ser mais o mesmo do adquirido ao chegar uma solicitação de *offloading*, portanto os cálculos para decisão do melhor dispositivo são realizados com valores desatualizados, provocando uma decisão errônea. Ainda assim, o dispositivo selecionado pelo algoritmo quando errônea a seleção foi sempre o segundo lugar em tempo de resposta, sendo a diferença do tempo médio de resposta entre o primeiro e o segundo dispositivo de somente 2.295 segundos.

As Tabelas 6 e 7 demonstram o tempo médio de execução de cada dispositivo nos testes sem manipulação do dispositivo pelo usuário e com manipulação do dispositivo pelo usuário. É interessante observar que a variação do tempo de execução não foi brusca, isso devido ao método de implementação da aplicação como explicado na seção 4.3.3, no qual se dedica CPU a aplicação. Outro ponto a ser observado é que os dispositivos Moto Z2 Play e TABLET SAMSUNG SM-T560 que estavam presentes nos dois ambientes de testes, possuem valores diferentes de média de tempo de execução nos dois ambientes. Isso é interessante por demonstrar como o tempo de resposta de um dispositivo varia de acordo com o contexto.

Figura 15 – Taxa de acertos, acertos perfeitos e exceções quando utilizados dispositivos não dedicados.




Fonte: o autor.

Tabela 6 – Comparação entre tempo médio de respostas em milissegundos dos dispositivos nos dois experimentos.

| Modelo | Tempo médio sem manipulação | Tempo médio com manipulação |
|---|---|---|
| Xiaomi Mi 8 Lite | 20177 | 20125 |
| Moto Z2 Play | 31288 | 33857 |
| Galaxy J7 Neo | 35736 | 36131 |
| Tablet Samsung SM-T560 | 38789 | 37082 |

Fonte: Elaborado pelo autor

Tabela 7 – Comparação entre tempo médio de respostas em milissegundos dos dispositivos

| Modelo | Tempo médio sem manipulação | Tempo médio com manipulação |
|---|---|---|
| Xiaomi Pocophone F1 | 13869 | 16716 |
| One Plus 5T | 14188 | 18477 |
| Moto Z2 Play | 21120 | 31625 |
| Tablet Samsung SM-T560 | 24792 | 36640 |

Fonte: Elaborado pelo autor

Nesse experimento foi dada liberdade para os usuário utilizarem os dispositivos de acordo com suas vontades, de modo a verificar o comportamento do algoritmo de decisão em um ambiente real. Os usuários intercalaram entre aplicações, utilizaram intensamente o

dispositivo, ou manipularam os dispositivos somente para verificação de mensagens e redes sociais esporadicamente, deixando-os em repouso em parte do tempo. Nesses testes, foi possível observar que apesar da variação constante do contexto dos dispositivos o tempo de resposta das solicitações de *offloading* não aumentou consideravelmente e que o algoritmo de seleção do melhor dispositivo sofreu um impacto negativo em sua taxa de acerto. Porém, no caso mais desafiador, com os dispositivos do conjunto {C,D,E,F} da Tabela 5, ao errar a seleção, o dispositivo selecionado pelo algoritmo foi sempre o segundo lugar em tempo de resposta, sendo a diferença média de tempo entre o primeiro e o segundo dispositivo de apenas 2,295 segundos.

Ainda assim, se for necessário aplicar a solução com mais precisão com relação ao contexto, é possível diminuir o tempo de aquisição de contexto para menos de 60 segundos. Desta forma o algoritmo de seleção poderia realizar os cálculos de decisão com parâmetros mais atualizados. Porém, há de se verificar o impacto que isso geraria no fluxo de comunicações e no consumo energético.

## 5.2 Experimentos de adaptação à mobilidade do REE

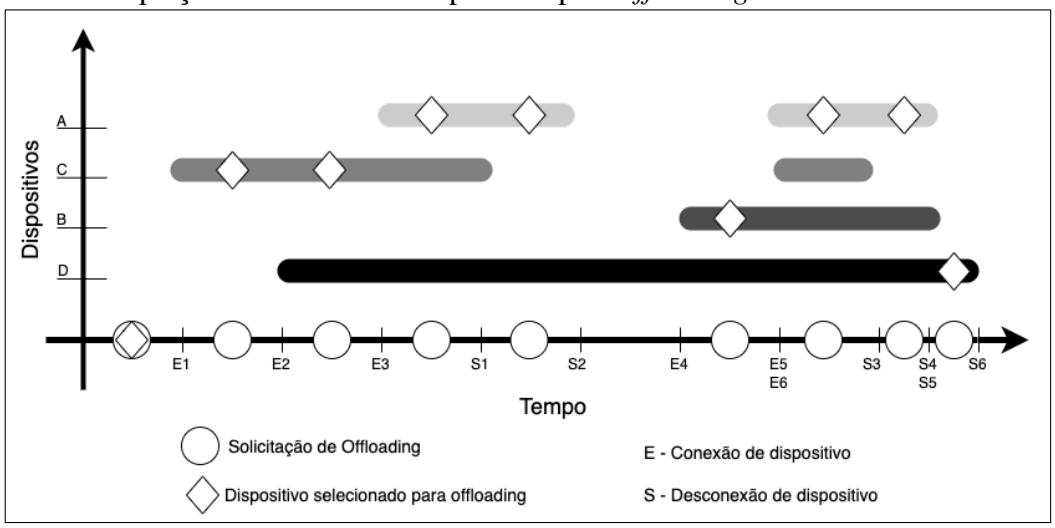
O objetivo deste experimento foi demonstrar que a solução se adapta a presença dos dispositivos de acordo com suas conexões e desconexões com o servidor, de modo a sempre indicar o melhor dispositivo disponível no momento para realização do *offloading*.

Para isto consideramos o resultado do experimento anterior utilizando-se do conjunto de dispositivos {A,B,C,D} em que pôde-se observar uma classificação de acordo com o poder de processamento dos 4 dispositivos, na seguinte ordem em que Px é o poder de processamento do dispositivo x: Pa > Pc > Pb > Pd. Desta forma, intercalamos a presença dos dispositivos no servidor de forma aleatória para verificar se sempre que se realiza uma solicitação de *offloading*, há uma adaptação a escolha do melhor dispositivo de acordo com os que estão presentes no exato momento.

A Figura 16 representa um gráfico temporal, onde pode-se observar a conexão e desconexão dos dispositivos com o servidor e a adaptação do REE escolhido de acordo com as requisições de *offloading* a cada momento.

De modo a facilitar a visualização, os dispositivos estão representados de acordo com seu poder de processamento no eixo y, sendo o mais alto o de maior poder. Cada dispositivo representa seu tempo de conexão com o servidor por uma linha de uma cor e o hexágono presente em sua linha de vida significa que este foi escolhido para atender a uma solicitação de *offloading*,

Figura 16 – Adaptação da escolha do dispositivo para *offloading* de acordo com o contexto.




Fonte: o autor.

representada pela esfera na dimensão tempo. O hexágono presente no início do eixo x, dentro da esfera de solicitação de *offloading*, representa que na primeira solicitação o processamento foi realizado localmente, pois como pode ser observado no gráfico não haviam dispositivos presentes naquele momento para atender a requisição.

## 5.3 Experimentos de balanceamento de consumo energético

O foco dessa seção é demonstrar o funcionamento do algoritmo na escolha do melhor dispositivo para *offloading* ao se priorizar o balanceamento do consumo energético dentre os dispositivos habilitados como REEs, ou seja, quando o desenvolvedor da aplicação seleciona o valor de ASAP como "*false*".

Para esse teste utilizamos o conjunto de dispositivos {A,B,C,D} demonstrados na Tabela 5 e verificamos a corretude do algoritmo ao selecionar o dispositivo de maior reserva de bateria dentre os disponíveis ou o que esteja conectado a uma fonte de energia. Os seguintes casos foram considerados para a avaliação do algoritmo:

I. Somente um dispositivo conectado a uma fonte de energia;
II. Mais de um dispositivo conectado a uma fonte de energia;
III. Nenhum dispositivo conectado a uma fonte de energia.

O primeiro caso de testes é o mais trivial, pois o algoritmo não necessita fazer cálculos de qual seria o melhor dispositivo naquele momento, e sim selecionar aquele que esteja "carregando". Durante os testes, o algoritmo sempre escolheu o único dispositivo que estava conectado a uma fonte de energia para atender as requisições, pois o foco é não gastar a energia

dos outros dispositivos que estão dependendo de suas baterias.

Para o segundo caso, é necessário definir uma estratégia quando dois ou mais dispositivos estão conectados à uma fonte de energia. Nesse trabalho, a decisão realizada foi a de selecionar o dispositivo com maior capacidade de processamento dentro do conjunto de dispositivos REEs "em carregamento". Dessa forma, para tal conjunto foi executado o algoritmo de seleção utilizando-se dos pesos obtidos do método AHP com foco em velocidade de processamento. A Figura 17 demonstra parte do *log* de execução, num cenário onde dos quatro dispositivos do conjunto {A,B,C,D} da Tabela 5, somente dois estavam conectados a energia. Nesse caso o algoritmo filtrou por somente os dispositivos que estavam conectados e realizou o processo de decisão utilizando o TOPSIS com os pesos do AHP que priorizam o poder de processamento para decidir qual era o REE de maior poder de processamento para atender a requisição de *offloading*.

Figura 17 – Exemplo de seleção do REE dentre os dispositivos “em carregamento”.

```
1  (CloudletStatusManager) Energy saving was selected
2  (CloudletStatusManager) More then one device charging!
3
4  TopsisCAOS class ---> Score: 15 Batery: 101 Wifi 100 RTT: 115.4
5  TopsisCAOS class ---> Ip: 192.168.0.13 Performance: 0.19542979148826625
6  TopsisCAOS class ---> Score: 6 Batery: 101 Wifi 100 RTT: 209.0
7  TopsisCAOS class ---> Ip: 192.168.0.33 Performance: 0.8045702085117338
8  Cloudlet Designeted: 192.168.0.33
```

Fonte: o autor.

No terceiro e último caso, o algoritmo utiliza o vetor de pesos sobre economia de energia em todo o conjunto de dispositivos disponíveis, buscando aquele de maior valor de bateria. Através dos testes, percebeu-se que nem sempre aquele de maior percentual de bateria era o escolhido. No caso em que dois ou mais dispositivos possuem valores próximos de bateria, o escolhido é o de melhor poder de processamento. Esse comportamento era esperado devido a seleção multicritério que está sendo aplicada, uma vez que cada parâmetro tem seu poder de decisão no algoritmo. Portanto, para valores próximos de bateria, outros parâmetros irão fazer a diferença na escolha. Neste trabalho, este comportamento é considerado válido uma vez que une o balanceamento de consumo energético e a melhoria da velocidade de processamento.

A Figura 18 demonstra o caso descrito no parágrafo anterior. Nessa imagem, que é parte do *log* de execução dos testes, podemos observar que em valores absolutos de bateria, o REE que deveria ter sido escolhido era o de IP 192.168.0.20, por possuir 96% de bateria, valor

superior a todos os demais. Porém, o escolhido foi o de IP 192.168.0.32 que estava com 93% de bateria, pois além de possuir valor de bateria similar, possuía uma capacidade de processamento superior aos demais dispositivos.

Figura 18 – Exemplo de seleção do REE considerando-se o balanceamento energético.

```
(CloudletStatusManager) Energy saving was selected

TopsisCAOS class ---> Score: 15 Batery: 81 Wifi 100 RTT: 69.5
TopsisCAOS class ---> Ip: 192.168.0.13 Performance: 0.47883145299416074

TopsisCAOS class ---> Score: 17 Batery: 96 Wifi 100 RTT: 64.2
TopsisCAOS class ---> Ip: 192.168.0.20 Performance: 0.584618582050341

TopsisCAOS class ---> Score: 11 Batery: 93 Wifi 100 RTT: 84.0
TopsisCAOS class ---> Ip: 192.168.0.32 Performance: 0.722487051121185

TopsisCAOS class ---> Score: 5 Batery: 56 Wifi 100 RTT: 95.2
TopsisCAOS class ---> Ip: 192.168.0.33 Performance: 0.415381417949659

Cloudlet Designeted: 192.168.0.32
```

Fonte: o autor.

## 5.4 Experimentos de consumo energético da solução

Por fim, essa seção tem por objetivo demonstrar o consumo energético da aplicação CAOS REE quando utilizada pelos dispositivos para atender solicitações de *offloading*. Essa verificação tem o propósito de avaliar o impacto do consumo de energia para dispositivos que ofertem o serviço e se esse valor é viável para que se possa utiliza a aplicação.

Quando foram implementadas as abordagens apresentadas na seção 4.3.3, para evitar que a aplicação deixasse de responder as requisições de *offloading* quando em *Doze Mode* e que o poder computacional da aplicação permanecesse constante para atendimento de requisições, surgiu uma preocupação com o consumo energético dessas abordagens. Portanto, para a verificação do real consumo de bateria pela aplicação, utilizamos um recurso do próprio sistema Android que é a demonstração do valor utilizado de bateria por cada aplicativo nas configurações do sistema.

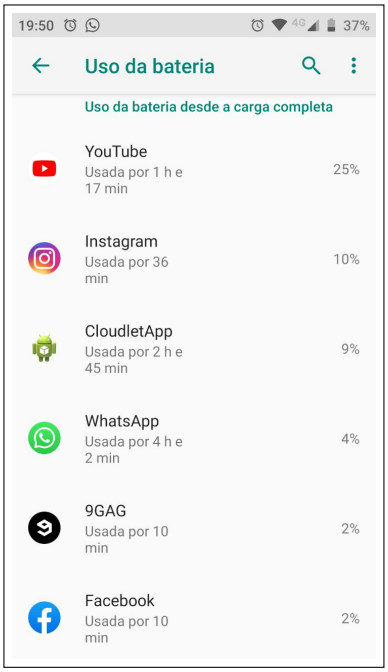
Durante a realização dos testes anteriores, em que a aplicação ficava constantemente se comunicando com o servidor e atendendo requisições de *offloading*, capturamos o percentual de bateria utilizada pelo aplicativo de acordo com sua utilização, que é disponibilizado pelo sistema Android. De modo a sumarizar a apresentação do consumo de energia pelos dispositivos,

coletamos dados de três dispositivos de fabricantes diferentes em tempos diferentes nos testes.

A Figura 19 demonstra o consumo da aplicação no dispositivo Moto Z2 Play, após a utilização do serviço por 2 horas e 45 minutos, contabilizando 9% da bateria, o que pode ser calculado como 1% de bateria a cada 18,33 minutos. Já na Figura 21 o dispositivo One Plus 5T utilizou a aplicação por 39 minutos, o que representou somente 1% de seu consumo energético. Já na Figura 20 apresenta-se o consumo energético do dispositivo Galaxy J7 Neo, a aplicação consumiu 5% da bateria em um período de 1 hora e 10 minutos de utilização, contabilizando 1% a cada 14 minutos.

É interessante observar que cada dispositivo tem sua maneira de gerenciar o consumo energético e no caso desses três dispositivos utilizados como exemplo, o valor consumido pela aplicação teve menos impacto nos dispositivos mais modernos.

Figura 19 – Consumo de bateria apresentado pelo dispositivo Moto Z2 Play.



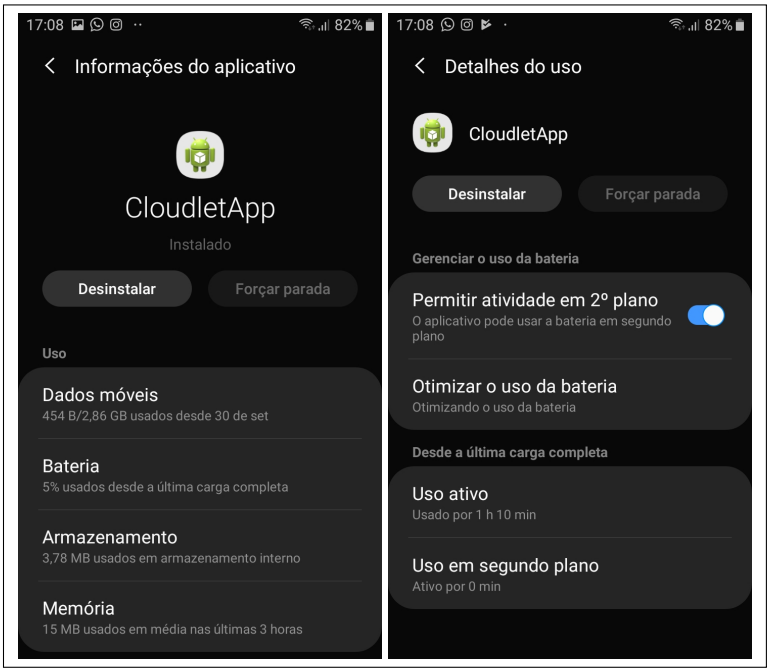

Fonte: o autor.

Na Figura 19 podemos visualizar a utilização do aplicativo CAOS REE e relacionar o seu consumo com o dos demais aplicativos utilizados pelo usuário. Como pode ser observado, a nossa aplicação apresenta um consumo moderado, se comparado com o das demais.

## 5.5 Considerações Finais

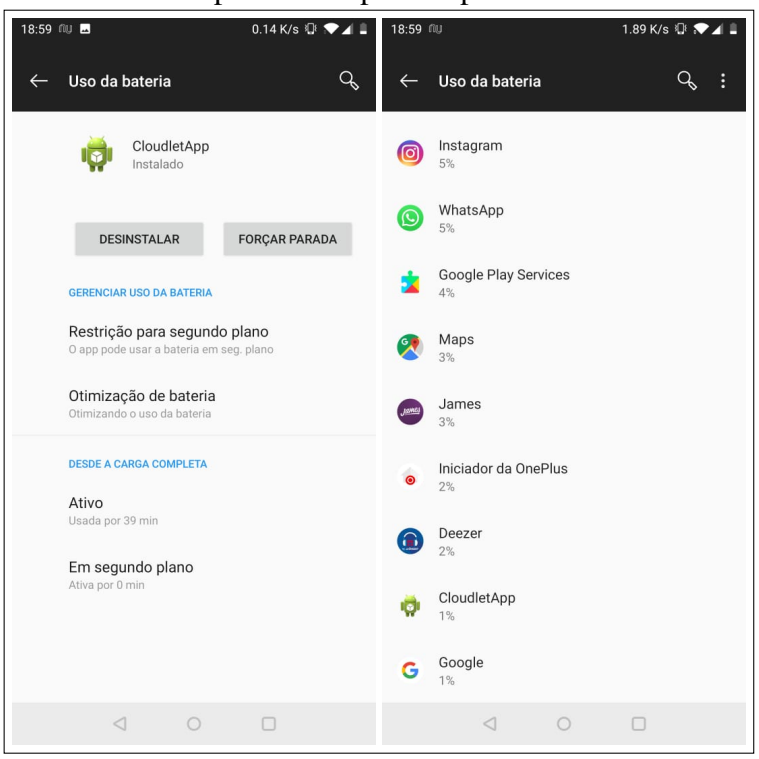
Esse capítulo apresentou os testes realizados para verificar a corretude da abordagem proposta ao selecionar o melhor dispositivo para atender requisições de *offloading* em um

Figura 20 – Consumo de bateria apresentado pelo dispositivo Galaxy J7 Neo.




Fonte: o autor.

Figura 21 – Consumo de bateria apresentado pelo dispositivo One Plus 5T.




Fonte: o autor.

ambiente com múltiplos dispositivos heterogêneos, de forma a adaptar a escolha de acordo com o contexto dos dispositivos e de acordo com a intenção da requisição de *offloading*. Foi demonstrado o comportamento do sistema implantado ao tratar a dinamicidade dos dispositivos no ambiente, e, por fim, o consumo energético da aplicação responsável por monitorar o contexto

e atender a requisições de *offloading*.

Em relação ao processo de escolha do dispositivo para atender as requisições, os resultados demonstraram que o sistema consegue selecionar adequadamente o melhor dispositivo, tanto estando estes dispositivos em repouso ou em utilização. Quando errônea a decisão, por similaridade dos dispositivos ou mudança de contexto, a diferença entre o dispositivo escolhido pela solução e o dispositivo que seria o ideal é mínima, sendo portanto qualquer uma das duas seleções adequadas.

O experimentos também demonstraram que o sistema consegue se adaptar a mobilidade dos dispositivos, verificando as conexões e desconexões dos dispositivos e selecionando o dispositivo mais adequado de acordo com os presentes para atendimento de requisições de *offloading*.

Quanto ao aspecto de consumo energético, os testes demonstraram a eficácia do sistema ao considerar o balanceamento de consumo energético dos dispositivos REEs, quando a requisição de *offloading* não tem por objetivo velocidade de processamento, e selecionando sempre para atender a tais requisições aqueles que estavam a carregar sua bateria ou aqueles de maior percentual de bateria no momento, poupando os demais.

Os testes realizados nesse capítulo também verificaram o consumo de energia pelo aplicativo proposto para monitoramento de contexto e atendimento de solicitações de *offloading*, CAOS REE. Através da análise do percentual de consumo energético de acordo com o tempo de execução, foi possível comprovar que a aplicação possui uso adequado de energia.

Com os testes realizados conseguimos demonstrar a eficácia da abordagem construída de seleção multicritério de acordo com o contexto dinâmico de dispositivos heterogêneos para atender requisições de *offloading* de dispositivos móveis.

# 6 CONCLUSÕES

Apesar das capacidades computacionais e de armazenamento energético dos dispositivos móveis terem evoluído bastante, tais fatores ainda necessitam de auxílio para a melhoria na execução de aplicações móveis. Tal auxílio pode vir de outros dispositivos móveis, uma vez que é vasta a heterogeneidade de tais dispositivos com poderes de processamento diferentes. Entretanto, a utilização de dispositivos móveis apresenta uma séria de desafios que precisam ser considerados para a criação de uma infraestrutura que habilite a realização de processos de *offloading* com vista na melhoria de velocidade de processamento das aplicações.

Este trabalho apresentou a abordagem criada para auxiliar a construção dessa infraestrutura onde possa ser possível a realização de *offloading* entre dispositivos móveis, apresentando um algoritmo de seleção para atender de forma otimizada as requisições de *offloading*, considerando-se a heterogeneidade e dinamicidade dos dispositivos.

## 6.1 Limitações

Como limitação do trabalho realizado podemos citar a restrição de utilização de redes Wi-Fi, uma vez que existem outras abordagens a serem utilizadas que permitem a conexão de dispositivos para a realização de atividades de *offloading*, como Bluetooth e 4G.

Outra limitação que pode ser elencada é a falta de mais métricas para monitoramento da qualidade de comunicação do dispositivo, além do valor de RTT. Uma abordagem que calculava as taxa de transmissão de *upload* e *download* dos dispositivos chegou a ser implementada, porém devido ao seu alto impacto na comunicação dos dispositivos durante esta medição, esta métrica não foi considerada.

Por fim, outra necessidade que foi observada foi a de uma métrica para verificação de poder de processamento de *Graphics Processing Unit* (GPU), uma vez que esta tem bastante impacto em atividades relacionadas a processamentos de vídeos e imagens do dispositivo.

## 6.2 Trabalhos Futuros

Como trabalhos futuros podemos elencar a adaptação da abordagem proposta para a utilização com dispositivos de IoT (AL-FUQAHA *et al.*, 2015), uma vez que tais dispositivos também possuem restrições de capacidade computacional e dependem de fontes de energia limitadas. A utilização de *smartphones* para processamentos de dispositivos de IoT poderia

auxiliar bastante tais dispositivos e a infraestrutura apresentada neste trabalho se adequaria bem para ser utilizada por tais dispositivos.

Outra abordagem que poderia ser melhor explorada futuramente é a realização do particionamento de atividades de *offloading* para execução por múltiplos dispositivos, de modo a acelerar o tempo de processamento. Para utilização de tal abordagem, um passo inicial já foi dado uma vez que apresentamos uma forma dos dispositivos realizarem *broadcast* de requisições de *offloading* para a verificação do tempo de resposta de cada dispositivo, que foi utilizado nos experimentos para verificação da corretude da seleção.

Adicionalmente, melhorias poderiam ser realizadas para sanar as limitações apresentadas na seção anterior, de modo a deixar mais robusto o algoritmo de decisão e obter maior precisão na seleção dos dispositivos para atendimento de requisições de *offloading*.

# REFERÊNCIAS

AKHERFI, K.; GERNDT, M.; HARROUD, H. Mobile cloud computing for computation offloading: Issues and challenges. **Applied computing and informatics**, Elsevier, v. 14, n. 1, p. 1–16, 2018.

AL-FUQAHA, A.; GUIZANI, M.; MOHAMMADI, M.; ALEDHARI, M.; AYYASH, M. Internet of things: A survey on enabling technologies, protocols, and applications. **IEEE communications surveys & tutorials**, IEEE, v. 17, n. 4, p. 2347–2376, 2015.

BANGUI, H.; GE, M.; BUHNOVA, B.; RAKRAK, S.; RAGHAY, S.; PITNER, T. Multi-criteria decision analysis methods in the mobile cloud offloading paradigm. **Journal of Sensor and Actuator Networks**, Multidisciplinary Digital Publishing Institute, v. 6, n. 4, p. 25, 2017.

BASIC, F.; ARAL, A.; BRANDIC, I. Fuzzy handoff control in edge offloading. In: IEEE. **2019 IEEE International Conference on Fog Computing (ICFC)**. *[S. I.]*, 2019. p. 87–96.

BONOMI, F.; MILITO, R.; ZHU, J.; ADDEPALLI, S. Fog computing and its role in the internet of things. In: ACM. **Proceedings of the first edition of the MCC workshop on Mobile cloud computing**. *[S. I.]*, 2012. p. 13–16.

CHILUKURI, S.; BOLLAPRAGADA, S.; KOMMINENI, S.; CHAKRAVARTHY, K. Raincloud-cloudlet selection for effective cyber foraging. In: IEEE. **Wireless Communications and Networking Conference (WCNC), 2017 IEEE**. *[S. I.]*, 2017. p. 1–6.

CISCO. the internet of things: Extend the cloud to where the things are. **Cisco White Paper**, 2015.

COSTA, P. B.; REGO, P. A.; ROCHA, L. S.; TRINTA, F. A.; SOUZA, J. N. de. Mpos: a multiplatform offloading system. In: ACM. **Proceedings of the 30th Annual ACM Symposium on Applied Computing**. *[S. I.]*, 2015. p. 577–584.

DEVELOPERS, A. **Optimize for Doze and App Standby | Android Developers**. 2015. Disponível em: https://developer.android.com/training/monitoring-device-state/doze-standby. Acesso em: 13/11/2019.

DEVELOPERS, A. **Background Execution Limits | Android Developers**. 2017. Disponível em: https://developer.android.com/about/versions/oreo/background.html#services. Acesso em: 13/11/2019.

GOMES, F. A.; REGO, P. A.; ROCHA, L.; SOUZA, J. N. de; TRINTA, F. Caos: A context acquisition and offloading system. In: IEEE. **2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)**. *[S. I.]*, 2017. p. 957–966.

GUO, Y.; XU, Y.; CHEN, X. Freeze it if you can: Challenges and future directions in benchmarking smartphone performance. In: ACM. **Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications**. *[S. I.]*, 2017. p. 25–30.

HABAK, K.; AMMAR, M.; HARRAS, K. A.; ZEGURA, E. Femto clouds: Leveraging mobile devices to provide cloud service at the edge. In: IEEE. **2015 IEEE 8th International Conference on Cloud Computing (CLOUD)**. *[S. I.]*, 2015. p. 9–16.

HAN, Q.; CHO, D. Characterizing the technological evolution of smartphones: insights from performance benchmarks. In: ACM. **Proceedings of the 18th Annual International Conference on Electronic Commerce: e-Commerce in Smart connected World**. *[S. I.]*, 2016. p. 32.

HWANG, C.-L.; LAI, Y.-J.; LIU, T.-Y. A new approach for multiple objective decision making. **Computers & operations research**, Elsevier, v. 20, n. 8, p. 889–899, 1993.

INTELLIGENCE, G. **Mobile Economyn2019**. 2019. Disponível em: https://atronocom.io/wp-content/uploads/2019/07/MobileEconomy_2019.pdf. Acesso em: 05/10/2019.

KALE, M. A.; DHAMDHERE, S. Survey paper on different type of hashing algorithm. **INTERNATIONAL JOURNAL**, v. 3, n. 2, 2018.

LI, H.; LIU, X.; MEI, Q. Predicting smartphone battery life based on comprehensive and real-time usage data. **arXiv preprint arXiv:1801.04069**, 2018.

MACH, P.; BECVAR, Z. Mobile edge computing: A survey on architecture and computation offloading. **IEEE Communications Surveys & Tutorials**, IEEE, v. 19, n. 3, p. 1628–1656, 2017.

MAHMUD, R.; KOTAGIRI, R.; BUYYA, R. Fog computing: A taxonomy, survey and future directions. In: **Internet of everything**. *[S. I.]*: Springer, 2018. p. 103–130.

MAIA, M. E.; FONTELES, A.; NETO, B.; GADELHA, R.; VIANA, W.; ANDRADE, R. Loccam-loosely coupled context acquisition middleware. In: ACM. **Proceedings of the 28th Annual ACM Symposium on Applied Computing**. *[S. I.]*, 2013. p. 534–541.

MARINESCU, D. C. **Cloud computing: theory and practice**. *[S. I.]*: Morgan Kaufmann, 2017.

MATHEW, M. **Multi Criteria Decision Making – Manoj Mathew**. 2019. Disponível em: https://mathewmanoj.wordpress.com/multi-criteria-decision-making/. Acesso em: 02/11/2019.

MELL, P.; GRANCE, T. *et al.* The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National . . . , 2011.

METAGEEK. **WiFi Signal Strength Basics | MetaGeek**. 2017. Disponível em: https://www.metageek.com/training/resources/wifi-signal-strength-basics.html. Acesso em: 05/11/2019.

NOOR, T. H.; ZEADALLY, S.; ALFAZI, A.; SHENG, Q. Z. Mobile cloud computing: Challenges and future research directions. **Journal of Network and Computer Applications**, Elsevier, v. 115, p. 70–85, 2018.

POWERMANAGER | Android Developers. 2009. Disponível em: https://developer.android.com/reference/android/os/PowerManager.html#PARTIAL_WAKE_LOCK. Acesso em: 17/11/2019.

REGO, P. A.; CHEONG, E.; COUTINHO, E. F.; TRINTA, F. A.; HASAN, M. Z.; SOUZA, J. N. de. Decision tree-based approaches for handling offloading decisions and performing adaptive monitoring in mcc systems. In: IEEE. **2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)**. *[S. I.]*, 2017. p. 74–81.

REGO, P. A. L. **Seamless Mobile Cloud Computing: Providing Adaptive Offloading Decision and Mobility Support**. 22–24 p. Tese (Doutorado) – Universidade Federal do Ceará, Fortaleza, 2016.

SAATY, T. L. How to make a decision: the analytic hierarchy process. **European journal of operational research**, Elsevier, v. 48, n. 1, p. 9–26, 1990.

SANTOS, G. B. dos; REGO, P. A.; TRINTA, F. Uma proposta de solução para offloading de métodos entre dispositivos móveis. In: SBC. **Anais Estendidos do XXIII Simpósio Brasileiro de Sistemas Multimídia e Web**. *[S. I.]*, 2017. p. 76–81.

SERVICES overview | Android Developers. 2019. Disponível em: https://developer.android.com/guide/components/services. Acesso em: 17/11/2019.

STATISTA. **Number of apps available in leading app stores as of 2nd quarter 2019**. 2019. Disponível em:https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/. Acesso em: 05/10/2019.

VARGHESE, B.; BUYYA, R. Next generation cloud computing: New trends and research directions. **Future Generation Computer Systems**, Elsevier, v. 79, p. 849–861, 2018.

VERBELEN, T.; SIMOENS, P.; TURCK, F. D.; DHOEDT, B. Cloudlets: Bringing the cloud to the mobile user. In: ACM. **Proceedings of the third ACM workshop on Mobile cloud computing and services**. *[S. I.]*, 2012. p. 29–36.

YI, S.; HAO, Z.; QIN, Z.; LI, Q. Fog computing: Platform and applications. In: IEEE. **2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)**. *[S. I.]*, 2015. p. 73–78.

YOUSEFPOUR, A.; FUNG, C.; NGUYEN, T.; KADIYALA, K.; JALALI, F.; NIAKANLAHIJI, A.; KONG, J.; JUE, J. P. All one needs to know about fog computing and related edge computing paradigms: A complete survey. **Journal of Systems Architecture**, Elsevier, 2019.

ZHOU, A.; WANG, S.; LI, J.; SUN, Q.; YANG, F. Optimal mobile device selection for mobile cloud service providing. **The Journal of Supercomputing**, Springer, v. 72, n. 8, p. 3222–3235, 2016.

ZHOU, B.; DASTJERDI, A. V.; CALHEIROS, R. N.; SRIRAMA, S. N.; BUYYA, R. A context sensitive offloading scheme for mobile cloud computing service. In: IEEE. **Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on**. *[S. I.]*, 2015. p. 869–876.

# APÊNDICE A – EXEMPLIFICAÇÃO DO MÉTODO TOPSIS

*Planilha adaptada de (MATHEW, 2019) para visualização do método TOPSIS com utilização dos pesos adquiridos via método AHP para exemplificação do processo de classificação dos dispositivos móveis de acordo com os parâmetros de interesse selecionados neste trabalho.*

## Solving MCDM problem using TOPSIS Method

**Step-0** **Create the table withe and input values**

| | Non Benf. | Benf. | Benf. | Non Benf. | Non Benf. |
|---|---|---|---|---|---|
| **weightage** | 0,337917926 | 0,021181715 | 0,070265179 | 0,445885377 | 0,124749804 |
| | **Score** | **Bateria** | **Wifi** | **Proc.** | **RTT** |
| **Mobile 1** | 3 | 31 | 100 | 0 | 81,7 |
| **Mobile 2** | 16 | 69 | 82 | 0 | 58,6 |
| **Mobile 3** | 4 | 33 | 100 | 0 | 62,1 |
| **Mobile 4** | 11 | 96 | 33 | 0 | 58,8 |
| **Mobile 5** | 25 | 10 | 50 | 0 | 400 |

**Step-1** **Calculate Normalised Matrix**

$$\overline{X}_{ij} = \frac{X_{ij}}{\sqrt{\sum_{i=1}^{n} X_{ij}^{2}}}$$

| | **Score** | **Bateria** | **Wifi** | **Proc.** | **RTT** |
|---|---|---|---|---|---|
| **Mobile 1** | 0,093612972 | 0,244109623 | 0,57436179 | 0,00 | 0,193962362 |
| **Mobile 2** | 0,499269184 | 0,543340775 | 0,470976668 | 0,00 | 0,139121107 |
| **Mobile 3** | 0,124817296 | 0,259858631 | 0,57436179 | 0,00 | 0,147430388 |
| **Mobile 4** | 0,343247564 | 0,755952382 | 0,189539391 | 0,00 | 0,139595923 |
| **Mobile 5** | 0,780108099 | 0,07874504 | 0,287180895 | 0,00 | 0,949632128 |

**Step-2** **Calculate weighted Normalised Matrix**

$$V_{ij} = \bar{X}_{ij} \times W_j$$

| | **Score** | **Bateria** | **Wifi** | **Proc.** | **RTT** |
|---|---|---|---|---|---|
| **Mobile 1** | 0,031633501 | 0,00517066 | 0,040357634 | 0 | 0,024196767 |
| **Mobile 2** | 0,168712007 | 0,011508889 | 0,03309326 | 0 | 0,017355331 |
| **Mobile 3** | 0,042178002 | 0,005504251 | 0,040357634 | 0 | 0,018391912 |
| **Mobile 4** | 0,115989505 | 0,016012368 | 0,013318019 | 0 | 0,017414564 |
| **Mobile 5** | 0,263612511 | 0,001667955 | 0,020178817 | 0 | 0,118466422 |

**Step-3** **Calculate the ideal best and ideal worst value**

| | | | | | |
|---|---|---|---|---|---|
| V+ | 0,031633501 | 0,016012368 | 0,040357634 | 0 | 0,017355331 |

| | | | | | |
|---|---|---|---|---|---|
| V- | 0,263612511 | 0,001667955 | 0,013318019 | 0 | 0,118466422 |

**Step-4 Calculate the Euclidean distance from the ideal best**

$$S_i^+ = \left[ \sum_{j=1}^{m} \left( V_{ij} - V_j^+ \right)^2 \right]^{0.5}$$

**Step-5 Calculate the Euclidean distance from the ideal worst**

$$S_i^- = \left[ \sum_{j=1}^{m} \left( V_{ij} - V_j^- \right)^2 \right]^{0.5}$$

**Step-6 Calculate Performance Score**

$$P_i = \frac{S_i^-}{S_i^+ + S_i^-}$$

| | Si+ | Si- | Pi | RANK |
|---|---|---|---|---|
| **Mobile 1** | 0,012819823 | 0,251881795 | 0,951568777 | 1 |
| **Mobile 2** | 0,137344709 | 0,140418882 | 0,505533794 | 4 |
| **Mobile 3** | 0,014922517 | 0,244528132 | 0,942484181 | 2 |
| **Mobile 4** | 0,088583743 | 0,179470867 | 0,669530984 | 3 |
| **Mobile 5** | 0,254264942 | 0,006860798 | 0,026273924 | 5 |