UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MESTRADO ACADÊMICO EM COMPUTAÇÃO

GABRIEL BRITO DOS SANTOS

UMA PROPOSTA DE SOLUÇÃO PARA OFFLOADING DE MÉTODOS ENTRE DISPOSITIVOS MÓVEIS

FORTALEZA

2017

GABRIEL BRITO DOS SANTOS

UMA PROPOSTA DE SOLUÇÃO PARA OFFLOADING DE MÉTODOS ENTRE DISPOSITIVOS MÓVEIS

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

Orientador: Prof. Dr. Fernando Antonio Mota Trinta

Co-Orientador: Prof. Dr. Paulo Antonio Leal Rego

FORTALEZA

2017

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)

S235p Santos, Gabriel Brito dos.
Uma Proposta de Solução para Offloading de Métodos entre Dispositivos Móveis / Gabriel Brito dos Santos. – 2017.
81 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2017.
Orientação: Prof. Dr. Fernando Antonio Mota Trinta.
Coorientação: Prof. Dr. Paulo Antonio Leal Rego.

1. Mobile Cloud Computing. 2. Offloading. I. Título.

CDD 005

GABRIEL BRITO DOS SANTOS

UMA PROPOSTA DE SOLUÇÃO PARA OFFLOADING DE MÉTODOS ENTRE DISPOSITIVOS MÓVEIS

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

Aprovada em: 29 de novembro de 2017

BANCA EXAMINADORA

Prof. Dr. Fernando Antonio Mota Trinta (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Paulo Antonio Leal Rego (Co-Orientador)

Prof. Dr. José Neuman de Souza
Universidade Federal do Ceará (UFC)

Prof. Dr. Francisco Airton Pereira da Silva
Universidade Federal do Piauí (UFPI)

AGRADECIMENTOS

Aos meus pais, Vilma e Francisco, que por incontáveis vezes sacrificaram o que possuíam e o que não possuíam, sempre se colocando em segundo plano para que os seus dois filhos pudessem ter as oportunidades que eles não tiveram na vida.

Aos meus orientadores, os professores Fernando Trinta e Paulo Rego, por terem me apresentado à oportunidade de trabalhar com o tema tratado nessa dissertação; pela excelente orientação e excepcional paciência na condução desta pesquisa.

Aos professores Neuman Souza e Airton Silva, pela disponibilidade em participar da banca examinadora dessa dissertação e pelas valiosas colaborações e sugestões.

Aos professores Windson Viana e Lincoln Rocha, pelas valiosas indicações nas diferentes oportunidades em que nos encontramos, que me ajudaram no desenvolvimento desse trabalho.

Aos meus colegas de pesquisa Almada e Joari, dos quais sem a ajuda e conhecimento não seria possível a conclusão dessa pesquisa em tempo hábil.

À minha esposa Aline, que tem me aturado durante quase 14 anos, sendo uma amiga, cúmplice e acima de tudo, um exemplo para mim; acreditando em mim mesmo quando eu não acreditei.

"Nada é permanente, exceto a mudança."

(Heráclito)

RESUMO

Apesar da capacidade de processamento de dispositivos móveis estar em ritmo de crescimento, o desempenho desses está sempre muito aquém de microcomputadores ou outros computadores estacionários para realização de certas tarefas. Além disso, dispositivos móveis possuem recursos energéticos limitados, de forma que o projeto de aplicações para tais dispositivos deve sempre levar em consideração a economia de energia. Uma alternativa para lidar com estes problemas é o uso da técnica de *offloading*, que consiste em oferecer ao dispositivo móvel a possibilidade de delegar uma tarefa para outro dispositivo com objetivo de melhoria de tempos de execução de processamento ou economia de energia. Este dispositivo pode ser um servidor remoto em uma nuvem pública ou na mesma rede do dispositivo. Diante desse problema, esta dissertação apresenta o CAOS D2D, uma proposta de *framework* para possibilitar a dispositivos móveis realizar *offloading* de computação para outros dispositivos, e também possibilita a dispositivos móveis atuarem como servidores de *offloading*. O protótipo implementado usa como base o *framework* CAOS, desenvolvido no GREat, e busca estendê-lo, de forma a possibilitar a implantação de todos os seus elementos em um dispositivo móvel. Para avaliação da solução, foram verificados aspectos como redução de tempo de execução de aplicações ao realizar *offloading* e melhorias de consumo de energia, utilizando-se diferentes dispositivos Android, relatórios de tempo de execução de aplicativos, e um equipamento para medição in loco de consumo de energia. Nos cenários analisados, houve casos em que, tanto no aspecto de tempos de execução, quanto no do consumo de energia, a execução de tarefas em *offloading* foi mais vantajosa do que a execução local. Mas também, devido a fatores como a complexidade computacional da tarefa e o volume de dados a serem processados, houve também situações em que o contrário se tornou verdade.

Palavras-chave: Mobile Cloud Computing. Offloading.

ABSTRACT

In spite of the increasing processing power of handheld smart devices, their capacity to perform some tasks is always a few steps behind their contemporary desktop counterparts. Besides, mobile devices have limited power supplies, which leads software designers to always keep energy consumption in mind when dealing with such devices. An alternative to help overcome this issue is using the offloading technique, which allows a mobile device to offload an expensive task to another device, for the sake of performance or energy saving. This second device may be a remote server hosted in a public cloud, or in the same Wi-fi network as the first mobile device. Facing this problem, this dissertation presents CAOS D2D, a proposal for a framework which allows for a mobile device to offload tasks to other mobile devices, as well as acting as an offloading server too. The prototype implementation is based on the CAOS framework, aiming to extend it in a way that its elements were embedded in a mobile device.In order to evaluate the solution, aspects such as the reduction in execution times of applications when performing offloading and improvements of energy consumption were verified, using different Android devices, application runtime reports, and an equipment for in loco measurement of consumption power. In the analyzed scenarios, there were cases in which, in terms of execution times and energy consumption, the execution of tasks in offloading was more advantageous than the local execution. But also, due to factors such as the computational complexity of the task and the volume of data to be processed, there were also situations where the opposite became true.

Keywords: Mobile Cloud Computing. Offloading.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACK	Acknowledge
APK	Android Package Kit
B	Byte
CoAP	Constrained Application Protocol
IC	Intervalo de Confiança
JAR	Java Archive
MB	Megabyte
Mbps	Megabits por segundo
MiB	Mebibyte
MP	Megapixel
OCR	Optical Character Recognition
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

SUMÁRIO

1 INTRODUÇÃO

O uso de tecnologias digitais está cada vez mais presente na sociedade mundial, e o uso da computação móvel tem aumentado consideravelmente nos últimos anos. Em 2014, o número de usuários de dispositivos móveis ultrapassou o número de usuários de microcomputadores, indicando também que usuários tendem a passar mais tempo conectados à Internet através de dispositivos móveis do que de *desktops* (CHAFFEY, 2016). Essa tendência é fortemente representada pelo aumento considerável no uso de aplicativos móveis com acesso à Internet por praticamente todas as camadas sociais. Aplicativos como *WhatsApp Messenger*[1] tornaram-se quase imprescindíveis para comunicação em nosso cotidiano. Essa massificação foi conquistada a partir da popularização dos *smartphones*, um modelo simplificado de aquisição de aplicações facilitado a partir das lojas virtuais, e da diversidade de aplicações como jogos em tempo real, redes sociais, dentre outras. Porém, diferente de microcomputadores pessoais, o grande diferencial da computação móvel é a possibilidade do usuário ainda acessar suas aplicações, mesmo enquanto o mesmo estiver em deslocamento. Além disso, dispositivos móveis carregam diferentes sensores que permitem obter informações contextuais relevantes, que podem ser usadas para melhorar a experiência do usuário ao usar um determinado aplicativo. A mobilidade e uso de informações contextuais são características chaves para a concretização do cenário de computação ubíqua, onde o usuário usa suas aplicações e dados, em uma interação natural.

Porém, a mobilidade tem seu preço. Ao mesmo tempo que se tornam populares, as aplicações tornam-se mais complexas, quer seja por utilizar um número cada vez maior de sensores do dispositivo, ou por manipular mais dados heterogêneos e complexos. Além disso, dispositivos móveis precisam ser leves para serem carregados pelos usuários, e com isso, possuem limitações em relação aos recursos de processamento e armazenamento. Por mais que haja atualmente uma tendência de crescimento no poder computacional de dispositivos móveis, características como peso, tamanho, e limitações de bateria tornam intrínseco aos dispositivos móveis o fato de não serem tão computacionalmente capazes quanto dispositivos estáticos contemporâneos a eles (SATYANARAYANAN, 1993).

A natureza compacta e o requisito da mobilidade de dispositivos móveis impõem restrições de uso de CPU, memória, armazenamento e bateria desses, o que torna pouco recomendável a esses dispositivos realizar certos tipos de tarefas, como processamento de imagens ou reconhecimento de padrões (ABOLFAZLI *et al.*, 2016), principalmente se for levado em

[1] http://www.whatsapp.com

consideração a necessidade de poupar o consumo de energia, de modo a aumentar a autonomia de funcionamento do aparelho, que normalmente é de poucas horas. Todas estas restrições são ainda mais problemáticas quando as aplicações apresentam uma tendência forte de uso intensivo de dados e sensores embutidos nos dispositivos. Exemplos dessas aplicações incluem jogos e realidade aumentada (e.g., Pokémon Go[1]), processamento de imagens e linguagem natural (e.g., Google Translator[2]) e computação vestível (e.g., Nike+ Run Club[3]).

Uma das possíveis soluções para contornar esse problema é a Computação Móvel em Nuvem (do inglês, *Mobile Cloud Computing* - MCC). Segundo Dinh *et al.* (2013), MCC tem por objetivo provisionar um conjunto de serviços equivalentes ao da nuvem, adaptados à capacidade de dispositivos com recursos restritos, de modo a trazer melhorias de desempenho das aplicações ou mesmo economia de energia nos dispositivos. Em geral isso é alcançado por meio de uma técnica conhecida como *offloading*, em que processos ou dados são transferidos de um dispositivo mais fraco (e.g., *smartphone*) para um dispositivo mais potente (e.g., máquina virtual na nuvem). Segundo Satyanarayanan *et al.* (2009), a técnica de *offloading* tem resultados ainda melhores quando se dá entre dispositivos próximos, uma vez que a latência de comunicação é um dos principais empecilhos para que a migração de tarefas seja realmente eficiente (CUERVO *et al.*, 2010; FERNANDO *et al.*, 2013). Com isso, propôs-se o conceito de *cloudlet*, uma infraestrutura para processar tarefas ou armazenar dados em nome de dispositivos mais fracos, mas que encontra-se na mesma rede local do dispositivo móvel. Um *cloudlet* pode ser tanto um servidor dedicado, um notebook pessoal de um usuário, ou mesmo outro dispositivo móvel que deseje ofertar seus recursos para ajudar dispositivos mais fracos.

O suporte a *offloading* já foi proposto por vários estudos (CHUN *et al.*, 2011; CUERVO *et al.*, 2010; GORDON *et al.*, 2012; KOSTA *et al.*, 2012) [4]. Um deles é o CAOS (*Context Acquisition and Offloading System*) (GOMES *et al.*, 2017), uma plataforma para *offloading* de métodos em aplicações Android, e que também permite o compartilhamento de dados contextuais de diversos usuários em *cloudlets*. O CAOS permite que o desenvolvedor marque quais métodos se deseja migrar para um *cloudlet*, e de acordo com regras sobre o tipo de processamento, parâmetros utilizados e a condição da rede, o sistema decide ou não migrar o processo. Vários experimentos realizados com o CAOS mostraram a eficiência da solução (GOMES *et al.*, 2016; GOMES *et al.*, 2017), porém, **a atual versão da plataforma**

[1] http://www.pokemongo.com
[2] http://translate.google.com
[3] http://www.nike.com.br/running/nrc-app
[4] Ver Capítulo 3 (Trabalhos Relacionados), na página 32

impõe restrições no tipo de máquina utilizada como *cloudlet*, o que impede o *offloading* de tarefas entre dois dispositivos móveis.

O cenário de Internet das Coisas (IoT) hoje é progressivamente emergente. Dispositivos microprocessados e conectados estão cada vez mais presentes no coditiano de pessoas comuns, com tecnologias para dispositivos compactos, como o Android Things[1], dispositivos vestíveis, como o Android Wear[2], aparelhos de TV inteligentes, como o Android TV[3] ou para veículos inteligentes, como o Android Auto[4].

Essa realidade com residências ou ambientes de trabalho repletos de dispositivos inteligentes, e que executam sistemas operacionais dentro do mesmo ecossistema, proporciona um cenário propício para que dispositivos vestíveis — que possuem recursos computacionais e energéticos escassos — possam se beneficiar da técnica de *offloading* para poupar seus recursos, prolongar a vida útil de suas baterias, e ganhar agilidade na execução de tarefas.

Desenvolver uma solução que facilite o projeto e desenvolvimento de aplicações que usem da técnica de *offloading* sem a necessidade de retrabalho para codificar tarefas relacionadas a esta representa um passo importante em direção da ubiquidade de recursos idealizada em (WEISER, 1991), pois pode aumentar a agilidade no desenvolvimento de tais aplicações, e com isso, aumentar a oferta delas.

1.1 Objetivos e contribuições

Este trabalho tem como objetivo principal contribuir para a área de MCC, apresentando **um *framework* que abstraia para desenvolvedores as tarefas de baixo nível relacionadas ao *offloading* de processamento para dispositivos móveis**. Como objetivos específicos, esse *framework* deve contar com recursos que facilitem seu uso e manutenção, como: descoberta de serviços e suporte a múltiplos servidores, e API's internas que viabilizam sua extensibilidade.

Este trabalho apresenta o CAOS D2D (*CAOS Device to Device*), uma extensão do CAOS com suporte ao *offloading* de tarefas entre dispositivos móveis baseados na plataforma Android. O CAOS D2D se apresenta para o desenvolvedor como um *framework*, mas que internamente oculta um *middleware* capaz de tratar do *offloading* de maneira transparente para desenvolvedores e usuários finais. Ele parte da refatoração de componentes existentes no CAOS,

[1] https://developer.android.com/things
[2] https://developer.android.com/wear
[3] https://developer.android.com/tv
[4] https://developer.android.com/auto

trazendo algumas das responsabilidades dos demais componentes para estes, de forma a tornar o *offloading* entre dispositivos móveis possível sem a infraestrutura de *cloudlets* necessária no CAOS. Dessa forma, o CAOS D2D nasce como uma versão complementar do CAOS, que possibilita que uma grande variedade de dispositivos possam contribuir para reduziro consumo de energia e o tempo de execução de métodos de uma aplicação móvel, o que representa mais um passo dentro do objetivo das aplicações distribuídas e ubíquas.

O escopo deste trabalho está na realização de *offloading* de processamento de um dispositivo móvel para outro dispositivo móvel que esteja ao alcance da sua rede local. O CAOS D2D possui um *middleware* subjacente ao *framework*, que trata do *offloading* de computação com relativamente poucas configurações a serem feitas pelo desenvolvedor de aplicativos móveis; tratando de questões como recuperação de falhas e injeção de dependências de forma transparente. Ele possui uma série de funcionalidades de apoio ao aspecto de *offloading*, dentre as quais se destacam: mecanismo de descoberta de serviço, que permite que aplicativos localizem dinamicamente dispositivos servidores dentro da rede, funcionalidade de implantação de dependências a partir do cliente, e carregamento de dependências por parte do servidor, que permite que um aplicativo cliente possa executar uma variedade maior de métodos, com maior variedade de argumentos possíveis. Essas funcionalidades são testadas e avaliadas nesse trabalho. Uma outra contribuição desse trabalho é a sua solução de comunicação de rede, implementada por cima de tecnologias nativas do Java e do Android.

1.2 Metodologia da pesquisa

A metodologia empregada no desenvolvimento desse trabalho é caracterizada de maneira abreviada nas seguintes fases:

1. **Revisão inicial de literatura:** A princípio, foi realizada através da ferramenta Google Acadêmico[1] uma revisão em artigos publicados, utilizando nas buscas as expressões “mobile cloud computing”, “mobile computing offloading”, “computation offloading” e “mobile cloud”, e filtrando resultados para artigos a partir de 2013 em diante. Dentre os resultados dessas buscas foram considerados relevantes artigos que tratavam da técnica de *offloading* de processamento dentro do contexto de *Mobile Cloud Computing*. Artigos que tinham como objeto principal outros aspectos do *Mobile Cloud Computing* (e.g., políticas

[1] http://scholar.google.com

de incentivos, métodos de tomada de decisão, ou particionamento de aplicações[1]) foram deixados de lado a princípio, mas reservados para eventuais consultas posteriores. Nessa fase inicial, dentre as publicações selecionadas estavam quatro *surveys*: (SHIRAZ *et al.*, 2013), (FERNANDO *et al.*, 2013), (KUMAR *et al.*, 2013) e (KHAN, 2015), que ajudaram a nortear a seleção dos principais artigos relacionados a *Mobile Cloud Computing*, e dos artigos que constituem trabalhos relacionados ao desenvolvimento do CAOS D2D.

2. **Seleção de trabalhos relacionados:** Após a revisão inicial da literatura sobre o tema de *Mobile Cloud Computing*, foram selecionados trabalhos que tratavam do desenvolvimento de alguma solução que possibilitasse o *offloading* de computação de dispositivos móveis, que descrevessem protótipos implementados e testados, e que apresentassem detalhes de arquitetura que permitissem a comparação com outros estudos e com o próprio CAOS D2D.
3. **Definição do CAOS D2D:** A partir da revisão de literatura, principalmente dos trabalhos relacionados, foram definidos os princípios de projeto do CAOS D2D: Um *framework* que abstrai ao desenvolvedor tarefas de baixo nível relacionados ao *offloading*, com suporte que este seja realizado de um dispositivo móvel para outro. Características dos trabalhos relacionados foram avaliadas de forma a auxiliar na definição das características do CAOS D2D.
4. **Implementação de protótipo do CAOS D2D:** Após a definição da arquitetura do CAOS D2D, foi definido como seria conduzida a implementação do protótipo da ferramenta, que foi executada de acordo com o plano previsto.
5. **Avaliação do protótipo:** Com um protótipo preparado, foram definidos testes para avaliar diferentes aspectos do CAOS D2D: O impacto do *offloading* no tempo de execução de métodos e na eficiência energética de aplicações, o tempo decorrido e o consumo energético da injeção de dependências, e o tempo de execução de diferentes estratégias de carregamento de dependências. Foram especificados experimentos para medições desses aspectos, e implementados aplicativos com o objetivo de serem utilizados nesses testes, avaliando-se os resultados obtidos.

[1] Ver seção 2.4 (Desafios em MCC e *Offloading*), na página 26

1.3 Organização da dissertação

Esta dissertação está organizada em 6 capítulos, organizados da seguinte forma: O atual capítulo apresenta a contextualização do problema enfrentado nessa pesquisa, e descreve uma introdução breve ao tema do *offloading*, diretamente ligado a essa dissertação. Esse capítulo também apresenta os objetivos e contribuições que se esperam ser alcançados por essa pesquisa, e a metodologia empregada na condução desta.

O Capítulo 2 apresenta a fundamentação teórica sobre os temas de *Mobile Cloud Computing* e *offloading*; temas fundamentais para compreensão dessa dissertação. São apresentadas conceitos como a definição dos temas, diferentes arquiteturas e abordagens possíveis para os paradigmas apresentados. São expostos também desafios enfrentados na área de pesquisa, apresentando entre outros, o desafio principal a ser enfrentado por esse trabalho. E ao final do capítulo, é apresentado um trabalho de fundamental importância para o desenvolvimento desse trabalho, servindo como ponto de partida para o CAOS D2D.

No Capítulo 3 são apresentados trabalhos publicados no meio acadêmico, relacionados aos temas de *Mobile Cloud Computing* e *offloading*. Esse capítulo expõe as principais características e limitações são discutidas desses trabalhos, e discute comparativamente essas características, inclusive comparando com o CAOS D2D, a proposta dessa dissertação.

O Capítulo 4 apresenta a proposta do CAOS D2D: as características da solução, como é efetuada a configuração do *framework*, como se processa o fluxo de execução da solução, a arquitetura definida para o CAOS D2D, e finalmente, como foi conduzida a implementação do protótipo do CAOS D2D.

O Capítulo 5 apresenta os testes realizados para a avaliação do protótipo desenvolvido, com foco no tempo de execução e consumo de energia dos dispositivos móveis. O capítulo caracteriza os experimentos que foram definidos, como eles foram conduzidos, e discute os resultados obtidos por esses experimentos.

Por fim, o Capítulo 6 descreve os resultados alcançados por esse trabalho, apresenta publicações realizadas durante o tempo de pesquisa para essa dissertação, e demonstra conclusões desse trabalho, além de discutir possíveis melhorias a serem realizadas em trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

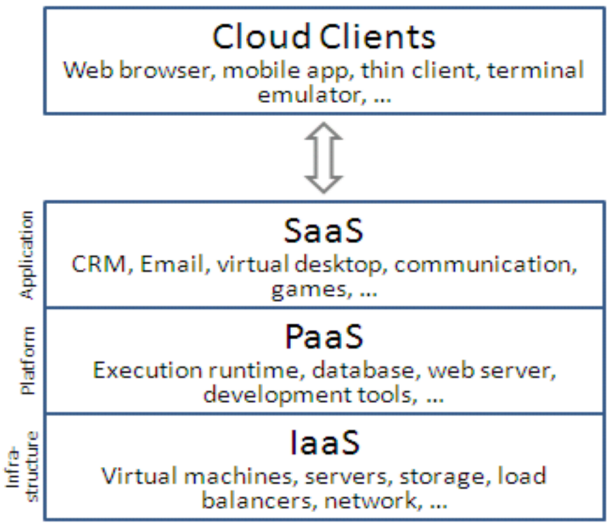
Os temas diretamente relacionados ao trabalho proposto são o paradigma de *Mobile Cloud Computing*, e o conceito de *offloading*. Esta seção apresenta os principais conceitos relacionados ao tema, e o CAOS (GOMES *et al.*, 2017) — uma solução para dar suporte ao *offloading* de processamento e dados a aplicações móveis da plataforma Android.

2.1 *Cloud Computing*

Não há uma definição absoluta do que seja *Cloud Computing*. Vários autores, acadêmicos ou da indústria de de tecnologia da informação, usam definições diferentes para o termo. Porém, normalmente *Cloud Computing* é descrita como um conjunto de recursos ou serviços fornecidos por servidores em *data centers* ou *clusters*, através da *Internet* (ASRANI, 2013), normalmente em modelos de tarifação *pay-as-you-use* e virtualizados.

Essencialmente, os serviços de *Cloud Computing* seguem uma arquitetura em três níveis, como ilustrado na Figura 1, sendo elas:

Figura 1 – Níveis de serviços em *Cloud Computing*



Fonte – (ASRANI, 2013)

- Infraestrutura – nível mais baixo e que compreende dispositivos físicos e *hardware* virtualizados, disponibilizados com finalidades diversas para os usuários. Nesse nível é permitido ao usuário instalar sistema operacional e *software*. Esse nível de serviço é chamado *Infrastructure as a Service* (IaaS). Um exemplo conhecido de IaaS é o *Amazon*

EC2[1].

- Plataforma – nível que fornece ao usuário diversos serviços de suporte a desenvolvimento, manutenção, testes, bancos de dados, *deploy*, e do que mais ele precisar para desenvolver seus produtos de *software* sobre o serviço de nuvem. Considerado o nível mais importante dessa estrutura (ASRANI, 2013), serviços ofertados nesse nível são classificados como *Platform as a Service* (PaaS). Exemplos desse nível de serviço são o *Google App Engine*[2] e o *Microsoft Azure*[3].
- Aplicação – esse nível de serviço fornece ao usuário produtos de *software* acessíveis através da *Internet*. Serviços fornecidos nesse nível são classificados como *Software as a Service* (SaaS). Qualquer aplicação distribuída através da *Internet* e hospedada remotamente pode ser considerada um SaaS (ASRANI, 2013). Exemplos desses serviços são o *Google Docs*[4] e *Salesforce*[5].

O aspecto central da *Cloud Computing* é a virtualização dos recursos. Esses são alocados de maneira transparente, sem que haja a necessidade de saber onde o recurso está fisicamente localizado. Isso permite a elasticidade e escalabilidade dos recursos que podem ser alocados dinamicamente sob demanda (ASRANI, 2013). Serviços em *Cloud Computing* são autônomos, i.e., recursos são alocados automaticamente e permitem ao usuário realizar tarefas como *build*, *deploy*, criação e remoção de máquinas virtuais sem interrupção do serviço, e sem interação humana. Além disso, o consumo e manutenção desses serviços pode ser realizado de qualquer lugar, através de navegadores *web*.

2.2 Mobile Cloud Computing

MCC é um paradigma que une três tecnologias heterogêneas: computação móvel e computação em nuvem através de redes sem fio (FERNANDO *et al.*, 2013), podendo ser apresentado como a disponibilidade de recursos de *Cloud Computing* em ambientes móveis. Esse paradigma destina-se a mitigar as limitações dos dispositivos móveis em relação ao consumo de energia e poder computacional, utilizando conectividade de redes sem fio para recursos de nuvens locais ou públicas. Esses recursos de nuvem podem ser usados para aumentar o poder computacional de dispositivos móveis, preservar recursos destes, aumentar a capacidade de

[1] http://aws.amazon.com/ec2
[2] http://cloud.google.com/appengine
[3] http://www.microsoft.com/azure
[4] http://docs.google.com
[5] http://www.salesforce.com

armazenamento, e enriquecer a experiência de usuários.

Na literatura, vários estudos cunharam suas definições para MCC:

> MCC é o mais recente paradigma computacional prático, que estende a visão da computação utilitária de computação em nuvem para aumentar os recursos limitados dos dispositivos móveis. (SHIRAZ *et al.*, 2013)

> MCC é uma rica tecnologia de computação móvel que utiliza recursos elásticos unificados de nuvens variadas e tecnologias de rede com funcionalidade irrestrita, armazenamento e mobilidade para servir uma multidão de dispositivos móveis em qualquer lugar, a qualquer momento através do canal de *Ethernet* ou Internet independentemente de ambientes heterogêneos e plataformas baseadas no princípio *pay-as-you-use*. (SANAEI *et al.*, 2014)

De acordo com essas definições, *Mobile Cloud Computing* pode ser apresentada como a disponibilização de serviços de *Cloud Computing* em ambientes móveis, integrados através de redes sem fio. Nesse modelo, aspectos como o processamento de dados e armazenamento são transferidos de dispositivos móveis para plataformas com recursos abundantes, alocados na nuvem (ASRANI, 2013).

2.2.1 Tipos de aplicações móveis

De acordo com Kovachev *et al.* (2011), aplicações móveis podem ser classificadas como: *offline*, *online* ou híbridas. Aplicações *offline* processam tanto a lógica de negócios quanto a apresentação localmente nos dispositivos móveis, normalmente com dados gerados pelo usuário ou descarregados de serviços remotos. Em alguns casos eles podem sincronizar dados com servidores remotos. Essas aplicações também são conhecidas como aplicações nativas. Aplicações *offline* possuem vantagens como: a integração com recursos dos dispositivos, desempenho otimizado para o hardware específico do dispositivo, e os recursos sempre disponíveis, mesmo sem conectividade de rede. Por outro lado, aplicações nativas apresentam desvantagens como: a falta de portabilidade para outras plataformas, o código-fonte mais complexo, e o fato dessas aplicações serem muito dependentes das capacidades de processamento e armazenamento do dispositivo, o que pode não ser suficiente para uma experiência satisfatória.

As aplicações *online* pressupõem que há conexão entre o dispositivo móvel e o servidor remoto na maior parte do tempo. Elas normalmente são baseadas em tecnologias *web* e apresentam algumas vantagens quando comparadas com aplicações *offline*, como por exemplo: Elas são multiplataforma, acessíveis a partir de qualquer lugar (com conectividade à *web*). Porém, elas apresentam também desvantagens como: a latência de conexões à *Internet* pode ser um

problema para algumas aplicações (e.g.: aplicações em tempo real, jogos), dificuldades em lidar com cenários que requerem que uma sessão de comunicação aberta por muito tempo, e a falta de acesso a sensores do dispositivo, como câmera ou localização.

Aplicações híbridas são o objeto de estudo para pesquisas em Mobile Cloud Computing e *Offloading*. Elas podem ser executadas localmente quando não há conexão disponível a serviços remotos, como aplicações *offline*. Entretanto, quando há disponibilidade de conexão com serviços remotos, essas aplicações podem migrar parte de suas tarefas para serem computadas fora do dispositivo móvel, em serviços na *web*, como aplicações online fazem. Aplicações híbridas buscam unir as vantagens de aplicações *online* e *offline*.

No decorrer desta dissertação consideraremos apenas as aplicações híbridas, que são aquelas que podem se beneficiar da técnica do *offloading*, que é o tema diretamente ligado à dissertação.

2.3 *Offloading*

Offloading de computação é uma técnica que visa melhorar o desempenho de aplicações móveis e reduzir o consumo de energia de dispositivos móveis através da migração de processamento ou dados de dispositivos móveis para outra infraestrutura; com maior poder computacional e de *storage*. O conceito de *offloading*, também conhecido como *cyber foraging*, surgiu em 2001 (SATYANARAYANAN, 2001), e foi aperfeiçoado em 2002 (BALAN *et al.*, 2002), a fim de permitir que dispositivos móveis se utilizem de maneira oportunista de recursos computacionais disponíveis.

Dada a limitação de recursos de dispositivos móveis, pesquisadores abordam o *offloading* como uma forma de elevar o desempenho de aplicações, reduzir o consumo de bateria, e executar aplicações que não seriam possíveis devido a insuficiência de recursos. Portanto, as principais razões para a aplicação do *offloading* são (REGO, 2016):

- **Melhora de tempos de processamento:** Quando o objetivo principal é a redução do tempo de execução de processamento em aplicações.
- **Economia de energia:** Quando a eficiência energética (i.e. a redução do consumo de energia) é o objetivo principal.
- **Outro:** Quando o tempo de processamento ou a eficiência energética não são o principal motivador do *offloading*. Em vez disso, o motivador pode ser o aumento de colaboração, a extensão de capacidade de armazenamento, ou a redução de custos financeiros.

Vale frisar que a *Mobile Cloud Computing* através do *offloading* é diferente do modelo cliente-servidor. No tradicional modelo cliente-servidor, o servidor sempre tem a responsabilidade de executar certas tarefas. A grande desvantagem dessa abordagem é a possibilidade de o dispositivo móvel não estar conectado à *Internet*; nesse caso, o usuário fica impossibilitado de usufruir de todas as funcionalidades da aplicação. O conceito de *offloading* busca eliminar essa desvantagem, oferecendo ao dispositivo móvel a possibilidade de executar uma tarefa computacionalmente cara local ou remotamente, de acordo com fatores como disponibilidade de rede e processos de tomada de decisão.

A literatura apresenta aplicações concebidas dentro do paradigma de *Mobile Cloud Computing* e auxiliadas com *offloading* como soluções para estender capacidades de sistemas móveis e possíveis soluções para estender o tempo de vida de bateria de dispositivos móveis (KUMAR; LU, 2010; AHUJA; ZAMBRANO, 2016); também apresenta cenários em que *Mobile Cloud Computing* com a capacidade de *offloading* possibilita o funcionamento de aplicações móveis complexas, como processamento de imagens, OCR[1], processamento de linguagem natural, *crowd computing* e *crowd sensing*.

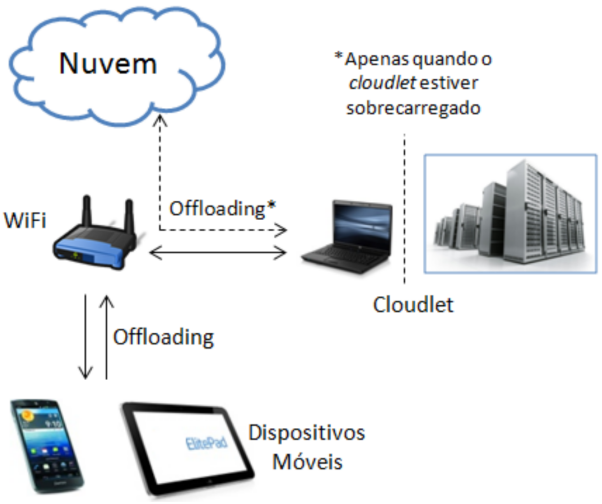
Tipicamente, existem dois tipos principais de *offloading*: processamento e dados (FERNANDO *et al.*, 2013). O *offloading* de processamento é a entrega de um processamento computacional do dispositivo móvel para outro ambiente de execução (e.g. *laptop*, máquina virtual na nuvem ou outro dispositivo móvel), a fim de prolongar a vida útil da bateria e aumentar a capacidade computacional. O *offloading* de dados tem por objetivo estender a capacidade de armazenamento do dispositivo móvel. Assim, retira-se o armazenamento do dispositivo e enviam-se os dados para um ambiente com maior capacidade de armazenamento, com a possibilidade também de enviar dados processados de volta para o dispositivo móvel (GOMES *et al.*, 2016).

Dispositivos móveis usam recursos remotos para melhorar o desempenho de aplicações móveis através de técnicas de *offloading*. Esse *Offloading* pode ser executado em uma nuvem pública remota, ou em um outro dispositivo dentro da mesma rede local sem fio, conectado à nuvem através de rede cabeada. Nesse último caso, o dispositivo remoto é chamado de *cloudlet* (SATYANARAYANAN *et al.*, 2009). A abordagem de uso de *cloudlets* é vantajosa por oferecer uma melhor qualidade de serviço e menor tempo percebido de execução, pela maior velocidade de transmissão de dados dentro de uma rede *wi-fi* de um único salto, em comparação

[1] OCR: Sigla em inglês para Reconhecimento Óptico de Caracteres

com a execução em uma nuvem pública remota. A Figura 2 ilustra um cenário de *offloading*. Nele, dispositivos móveis realizam *offloading* para recursos remotos através da rede *Wi-Fi*. Esses recursos podem estar hospedados em computadores próximos dedicados (*cloudlets*), ou em uma nuvem pública.

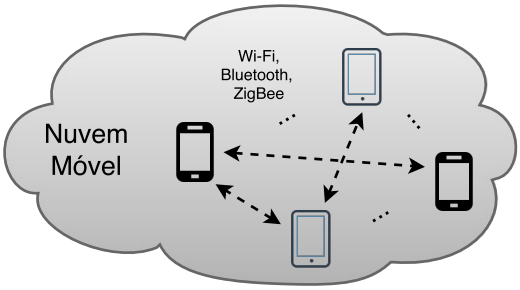
Figura 2 – Cenário geral de *offloading*



Fonte – (COSTA, 2014)

A Figura 3 ilustra uma outra técnica, comumente referenciada como *Mobile Cloudlet* ou *Mobile Cloud*, que consiste em formar uma nuvem com diversos dispositivos móveis próximos, conectados em uma rede *ad-hoc*, para que juntos esses dispositivos móveis realizem tarefas de maneira colaborativa, ou para que um dos dispositivos móveis se beneficie das vantagens do *offloading* para essa nuvem. As vantagens dessa abordagem são o potencial de disponibilidade de uma nuvem móvel, tendo em vista a crescente popularização de dispositivos móveis; e o potencial de escalabilidade da nuvem móvel à medida que mais dispositivos se conectam à rede (e.g. em locais com grande concentração de pessoas).

Figura 3 – Ambiente de execução de *offloading* formado por uma nuvem de dispositivos móveis

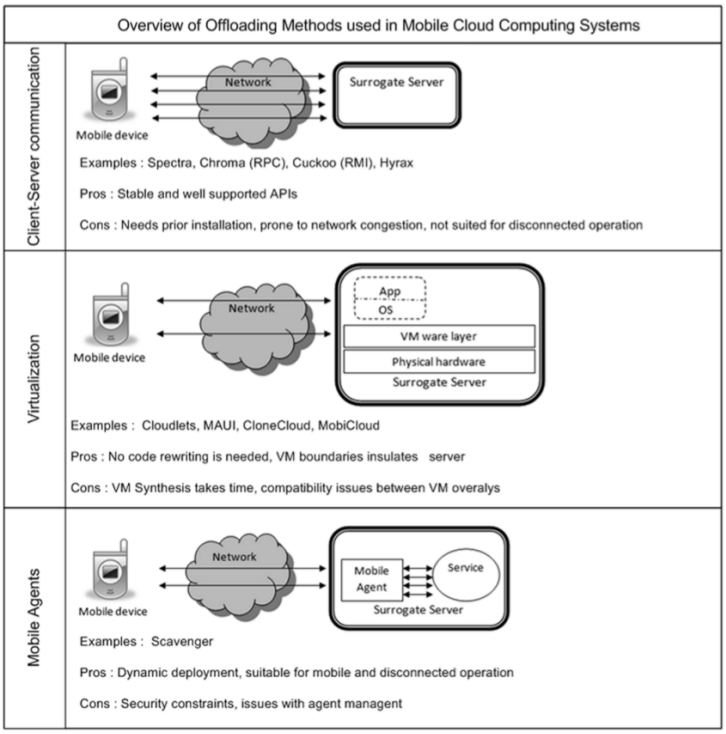


Fonte – (REGO, 2016)

2.3.1 Métodos de offloading

Devido a fatores como a distância entre dispositivo móvel e a nuvem, e a heterogeneidade de sistemas de suporte, diferentes pesquisas abordam a questão da escolha do método de *offloading* de diferentes maneiras. Atualmente, essas pesquisas apontam três abordagens principais: modelo cliente-servidor, virtualização, e agentes móveis (FERNANDO *et al.*, 2013). A Figura 4 mostra um resumo sobre essas abordagens.

Figura 4 – Principais abordagens para métodos de *offloading*



Fonte – (FERNANDO *et al.*, 2013)

No modelo de *offloading* cliente-servidor, a comunicação entre dispositivo móvel e *surrogate* é realizada através de chamada de procedimento remoto (RPC), Java RMI[1], ou outro *framework* que permita a chamada remota de procedimentos através de *sockets*. Protocolos como RPC ou RMI possuem API's[2] bem documentadas e estáveis, mas têm a desvantagem de que ambas as partes – cliente e servidor – necessitam ter os serviços pré-instalados. Isso vai de encontro à natureza *ad-hoc* de uma nuvem móvel, e restringe mobilidade de dispositivos (FERNANDO *et al.*, 2013).

O método de *offloading* através de migração de máquinas virtuais (VM's) consiste em transferir o estado da memória de uma VM, de um dispositivo para outro, sem interromper

[1] Java *Remote Method Invocation*, uma implementação nativa na plataforma Java para o RPC.
[2] API: Application Program Interface

sua execução. Esse método teoricamente passa uma sensação de migração transparente, e garante que não há mudança de código quando é feito *offloading* de uma tarefa, e a executa de maneira relativamente segura. Porém, essa migração é um tanto demorada, e relativamente pesada para dispositivos móveis (FERNANDO *et al.*, 2013).

Agentes móveis são itens de *software* com mobilidade, que podem ser enviados através de um dispositivo em uma rede e trafegar entre os nós desta rede. Nesse modelo, um processo pode ser suspenso e em seguida convertido em uma mensagem incluindo todas as suas informações de estado (pilha de execução, referências externas, estado de memória), para ser enviado a um outro dispositivo dentro da rede e continuar sua execução neste. Após o término do processo, o agente é migrado novamente para o dispositivo original (CHESS *et al.*, 1997)(ANGIN; BHARGAVA, 2013). No método de *offloading* com agentes móveis, o *deploy* de agentes é realizado dinamicamente dentro da rede *ad-hoc*, mas não há garantia de um ambiente de execução seguro, e há questões de dificuldade técnica com o gerenciamento de agentes (FERNANDO *et al.*, 2013).

No desenvolvimento do CAOS D2D, a abordagem escolhida para método de *offloading* foi o modelo cliente/servidor, pelas vantagens apresentadas; adotando-se medidas de contingência para permitir a mobilidade de dispositivos.

2.4 Desafios em MCC e *Offloading*

Existem várias questões de pesquisa relacionadas a *Mobile Cloud Computing* e *offloading*. Dentre as mais importantes, pode-se citar (FERNANDO *et al.*, 2013; KUMAR *et al.*, 2013):

- **Análise de custo-benefício (tomada de decisão):** Uma série de fatores podem determinar quando é vantajoso fazer ou não *offloading* de alguma operação: Taxa de transmissão de rede, latência, nível de bateria, tarifas de uso de redes celulares. Até mesmo a definição do que seria "vantajoso" é passível de ponderação.
- **Políticas de oferta de incentivos:** Para o ingresso em uma nuvem móvel, o usuário deve ser convencido a "ceder" recursos computacionais e de energia de seu dispositivo para uma nuvem colaborativa. Para lidar com essa questão uma variedade de abordagens foram propostas. (ARTAIL *et al.*, 2015), por exemplo, propõe o uso de recompensas financeiras, em forma de brindes ou descontos na compra de produtos ou serviços para usuários que permitem que seus dispositivos móveis sejam incorporados à nuvem móvel. (HUERTA-

CANEPA; LEE, 2010) argumenta que uma forma de incentivo é encontrar usuários que tenham uma tarefa em comum para executar, e que esta seja dividida entre eles; assim o ganho de desempenho seria um incentivo para esses usuários.

- **Particionamento de aplicações:** Para que haja o *offloading* de processamento ou dados, uma parte do aplicativo móvel precisa ser transferida para a nuvem. A escolha do nível de granularidade do particionamento das aplicações deve ser feita com cautela, e pode variar entre diferentes abordagens: Desde granularidade mais fina, como em nível de métodos, até níveis mais altos, em que máquinas virtuais inteiras podem ser migradas.
- **Privacidade e Segurança:** Devido à natureza limitada de dispositivos móveis, usuários e aplicações estão cada vez mais armazenando dados em serviços de nuvem (e.g.: Google Photos[1]). Porém, algumas vezes tais serviços podem sofrer invasões ou problemas que podem variar da àrea judicial à financeira. Caso algum desses problemas aconteça, usuários podem ter seus dados perdidos ou sua privacidade violada.
- **Onde realizar o *offloading*:** Em geral, a execução dessa técnica pode acontecer em três locais possíveis: (*i*) uma nuvem pública; (*ii*) um computador próximo dedicado para esse fim (um *cloudlet*); e (*iii*) outro dispositivo móvel.

2.4.1 Onde executar o offloading

Na execução em nuvem pública, os desenvolvedores de aplicativos móveis podem utilizar os serviços da nuvem para obter melhores tempos de processamento, assim como elasticidade e conectividade a redes sociais para melhorar os serviços dos aplicativos. A execução em *cloudlet* é proposta em (SATYANARAYANAN *et al.*, 2009) e tem como principal objetivo aproximar os serviços de *offloading* dos dispositivos móveis para serem executados em máquinas locais, ao invés de utilizar uma nuvem pública. Essas máquinas podem ser *desktops* ou *laptops*, que estão disponíveis em uma mesma rede local sem fio que os usuários das aplicações móveis.

Executar o *offloading* em outro dispositivo móvel com maior poder computacional também é possível. Nessa abordagem, os dispositivos móveis são normalmente conectados usando uma rede *peer-to-peer*, criando uma nuvem de dispositivos (nuvem móvel), em que dispositivos mais robustos cedem parte dos seus recursos computacionais, para resolver uma tarefa em comum a todos os outros dispositivos (FERNANDO *et al.*, 2013) mais fracos. Existem diversas tecnologias que podem ser usadas para a conexão entre os dispositivos (e.g., *Wi-Fi*,

[1] https://photos.google.com

Bluetooth e ZigBee), como ilustrado na Figura 3.

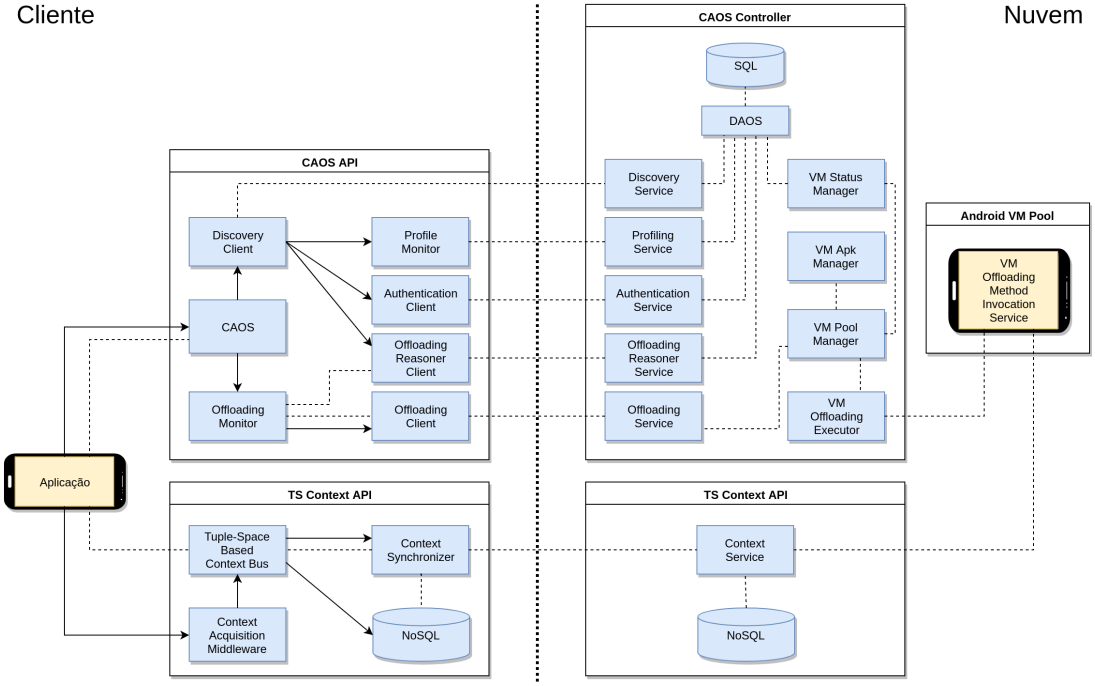
A seguir, é apresentada uma solução para dar suporte ao gerenciamento de contexto, *offloading* de processamento e dados de aplicações móveis da plataforma Android, intitulada de CAOS (GOMES *et al.*, 2017), que foi o ponto de partida da proposta deste trabalho.

2.5 CAOS

CAOS *Context Acquisition and Offloading System* é uma infraestrutura de software projetada dentro do Grupo de Redes de Computadores, Engenharia de Software e Sistemas — GREat-UFC, para auxiliar no desenvolvimento de aplicações móveis e sensíveis ao contexto na plataforma Android. Ele conta com com suporte ao mecanismo de *offloading* para delegar a migração e o processamento de dados contextuais de dispositivos móveis para plataformas de nuvem (GOMES *et al.*, 2017). O CAOS tem como base duas soluções: *(i) Loosely Coupled Context Acquisition Middleware* (LoCCAM) (DUARTE *et al.*, 2015; MAIA *et al.*, 2013), que é um *middleware* que auxilia os desenvolvedores de aplicações móveis e sensíveis ao contexto a obter informações dos sensores, bem como a separar questões relacionadas à aquisição de informações contextuais dos aplicativos da lógica de negócio; e *(ii) Multiplatform Offloading System* (MpOS) (REGO *et al.*, 2016), que é um *framework* que permite aos desenvolvedores marcarem métodos nos aplicativos usando anotações Java, a fim de que esses métodos sejam transferidos e executados em servidores da nuvem, ao invés de executados localmente no dispositivo móvel.

Com respeito ao suporte de *offloading* do CAOS, a granularidade adotada depende do método da classe, que é marcado com uma anotação Java. O CAOS suporta ambiente de execução remoto com base em nuvem pública e *cloudlets* e o módulo de decisão de *offloading* levam em consideração métricas do dispositivo móvel, do servidor, da rede (e.g., latência, configurações de *hardware*), e dados históricos para a execução dinâmica do processamento. No CAOS, a localização do módulo de decisão é compartilhada, onde parte dele é executado no dispositivo móvel e a outra parte é executada no servidor. O CAOS também possui um mecanismo de descoberta de serviço para procurar nuvens/*cloudlets* disponíveis no ambiente do usuário. O principal objetivo do CAOS é fornecer *offloading* de processamento e dados contextuais.

Figura 5 – Visão geral da arquitetura do CAOS



Fonte – Autoria própria

2.5.1 Arquitetura do CAOS

O CAOS possui componentes que são executados no dispositivo móvel, no servidor de nuvem, e em máquinas virtuais hospedadas nesses servidores. A Figura 5 ilustra esses componentes. Os principais componentes do lado móvel são: *Offloading Monitor*, *Offloading Reasoner Client*, *Offloading Client*, *Discovery Client*, *Profile Monitor* e os módulos de aquisição, gerenciamento e armazenamento de contexto do LoCCAM.

O *Offloading Monitor* é o componente responsável por monitorar a execução das aplicações e interceptar o fluxo de execução do método marcado. No CAOS, os métodos são marcados com anotação Java (*@Offloadable*), o qual define que o método deve executar preferencialmente fora do dispositivo móvel. Depois de interceptar o método, esse componente pergunta ao *Offloading Reasoner Client* se é possível iniciar o processo de *offloading* ou não. Esse último componente auxilia a decisão de *offloading* através de uma estrutura de dados de decisão que é recebida do servidor (REGO *et al.*, 2017). Ao usar a estrutura de dados recebida, o *Offloading Monitor* pode decidir localmente quando vale a pena realizar o *offloading*. Se a resposta for negativa, o fluxo de execução do método é retomado e sua execução continua localmente. Caso contrário, quando a resposta for positiva, o *Offloading Monitor* solicita ao *Offloading Client* que inicie o processo de *offloading* do método, que transfere o método e seus parâmetros ao servidor da nuvem/*cloudlet*. O componente *Discovery Client* usa um mecanismo

baseado em UDP/Broadcast para descobrir a parte servidora do CAOS, que executa em *cloudlets*. O módulo *Profile Monitor* é responsável por monitorar o ambiente do dispositivo móvel (e.g., largura de banda, latência) e enviar essas informações periodicamente para o servidor da nuvem, que gera a estrutura de dados de decisão.

Os principais componentes do lado remoto são: *Discovery Service*, *Profile Services*, *Offloading Reasoner Service*, *Offloading Service*, *VM Pool Manager*, *VM Apk Manager*, *VM Status Manager*, *VM Offloading Executor* e *Offloading Method Invocation Service*.

O componente *Discovery Service* fornece o endereço da parte servidora do CAOS para os usuários. O *Profile Services* é responsável por receber os dados dos dispositivos móveis (e.g., largura de banda, latência) e o tempo de execução local do método, para manter uma avaliação histórica do tempo decorrido para esses métodos. Esses registros podem ser usados para decidir se um método deve ser descarregado ou não. O *Offloading Reasoner Service* processa os dados do dispositivo móvel e gera uma estrutura de dados de decisão, que pode ser usada pelo módulo *Offloading Reasoner Client* no lado móvel, para decidir sobre a execução do processo de *offloading*. O *Offloading Service* recebe pedidos de *offloading* diretamente do *Offloading Client* e redireciona para *VM Pool Manager*. Quando o processo de *offloading* terminar, o *Offloading Service* retorna o resultado ao *Offloading Client* e persiste a informação de *offloading*.

No CAOS, os métodos descarregados são executados em máquinas virtuais Android que funcionam em computadores x86 tradicionais. O *VM Pool Manager* é responsável por fornecer um ambiente que redireciona as solicitações de *offloading* para uma máquina virtual Android apropriada, onde ocorre a execução de *offloading*. Para executar um método de uma aplicação específica, o CAOS exige que os pacotes de implantação correspondentes (i.e., APK) de aplicações compatíveis com o CAOS sejam armazenados em uma pasta especial na plataforma CAOS. O *VM Apk Manager* envia todos os arquivos APK para todas as máquinas virtuais Android disponíveis. O *VM Status Manager* é responsável por monitorar e manter informações (e.g., número atual de execuções de descarregamento) sobre cada máquina virtual gerenciada pelo *VM Pool Manager*. O *VM Offloading Executor* é responsável por solicitar a execução de *offloading* em máquinas virtuais Android, chamando o serviço *Offloading Method Invocation*, que é executado na máquina virtual e realiza a execução do método.

2.6 Considerações finais do capítulo

Este capítulo apresentou os conceitos de Mobile Cloud Computing (MCC) e *Offloading*, que são os temas diretamente ligados a este trabalho. O capítulo classificou diferentes tipos de aplicações móveis, apresentou questões relacionadas a *offloading*, e desafios de pesquisa para a área; definindo dentro desses fatores o escopo de pesquisa para este trabalho.

O capítulo também apresenta o CAOS (GOMES *et al.*, 2017), um *framework* que permite o *offloading* de processamento e dados contextuais para *cloudlets*, que foi o ponto de partida para o desenvolvimento do CAOS D2D.

3 TRABALHOS RELACIONADOS

O suporte a *offloading* de processamento para nuvem e *cloudlet* é proposto por uma série de estudos (CHUN *et al.*, 2011; CUERVO *et al.*, 2010; GORDON *et al.*, 2012; GOMES *et al.*, 2017); e nos últimos anos, uma variedade de estudos propôs também o suporte a *offloading* de processamento para outros dispositivos móveis (ARTAIL *et al.*, 2015; LIAO *et al.*, 2015; TEO, 2012; FERRARI *et al.*, 2016). Esses estudos apresentam diferentes requisitos e abordagens para alcançar os objetivos a que se propõem. A seguir, são apresentados tais trabalhos e suas principais características, e ao final do capítulo, é feita uma comparação entre esses trabalhos e o CAOS D2D.

O *middleware* CloneCloud (CHUN *et al.*, 2011) propõe uma solução para *offloading* de processamento para a plataforma Android. Nela, os autores usam uma versão modificada da Dalvik VM (máquina virtual Java utilizada pelo Android) para promover o *offloading* de processamento através da migração de *threads*. Na solução, uma máquina virtual clone da que está em execução no dispositivo móvel pode ser criada e executada em uma nuvem pública, ou um *cloudlet*, e *threads* de aplicações móveis em execução no dispositivo móvel podem ser migradas e executadas dinamicamente nos clones. O particionamento da aplicação é feito pelo próprio *middleware*, no momento da implantação da máquina virtual modificada, e o *middleware* decide de antemão quais porções do aplicativo serão migrados em *offloading*. No momento do *offloading*, o estado da máquina virtual cliente é transferido para a nuvem, e a *thread* selecionada para execução remota é executada nesta. Um ponto desfavorável do CloneCloud é o fato de ele se basear em uma versão modificada da máquina virtual utilizada pelo Android; o que implica em alterar o sistema operacional dos dispositivos móveis para torná-los aptos a seu uso, e que para cada nova versão do Android deve ser lançado também uma nova versão do CloneCloud.

O MAUI (CUERVO *et al.*, 2010) é um *framework* para *offloading* de processamento para nuvem, desenvolvido em .NET para a plataforma Windows Phone, com foco principal no aspecto da economia de energia. O MAUI usa a implementação existente de *Remote Procedure Call* (RPC) na plataforma .NET para invocar a execução de métodos em nuvens. Nesse *middleware*, o desenvolvedor deve indicar através de anotações C# no código-fonte das aplicações quais métodos ele deseja que sejam propensos a *offloading*. O MAUI usa uma abordagem baseada em um *proxy* para interceptar métodos e, de acordo com um processo de decisão, redirecioná-los para execução em um servidor remoto, de uma maneira similar ao CAOS D2D. Um ponto de limitação do MAUI é a abordagem de serialização baseada em XML, utilizada para transferência de dados

entre cliente e nuvem, o que torna a representação dos dados serializados bem maior do que sua representação na memória dos dispositivos. À medida que a complexidade e o tamanho dos dados a serem transferidos aumenta, a diferença para o tamanho de sua representação serializada também aumenta.

O COMET (GORDON *et al.*, 2012) é um *middleware* para o sistema Android, que busca o suporte a *offloading* de aplicações *multi-thread* entre dispositivos móveis e nuvem, com foco em melhoria de tempo de processamento em aplicações e economia de energia. Assim como o CloneCloud, o COMET também se utiliza de uma versão modificada da Dalvik VM para alcançar esse objetivo. Porém, diferente do CloneCloud, a migração de tarefas é realizada através de uma implementação do conceito de *Distributed Shared Memory* (DSM), que torna a memória da máquina virtual Java em execução compartilhada entre o dispositivo móvel e o *cloudlet*. Em outras palavras, o COMET transforma a máquina virtual Java em execução no dispositivo móvel em uma parte de um cluster formado por duas máquinas virtuais Java: uma hospedada no dispositivo móvel, e outra no *cloudlet*. As duas máquinas virtuais têm seus estados constantemente sincronizados, e a execução de *threads* é dividida entre as duas máquinas virtuais, podendo-se considerar que nesse cenário há apenas uma máquina virtual, que usa dois processadores distribuídos — O do dispositivo móvel e o do *cloudlet*. O cerne de funcionamento do COMET é a sua base implementada em uma versão modificada da Dalvik VM, o que também representa uma grande desvantagem: para se usar o COMET, deve-se alterar a "originalidade" do sistema operacional do dispositivo móvel, tanto que seu protótipo foi implementado a partir do código de uma versão do CyanogenMod[1].

O *framework* definido em (ARTAIL *et al.*, 2015) oferece suporte a *offloading device-to-device*, em um modelo de nuvem federada. Nesse *framework*, dispositivos móveis dispostos a agir como servidores oferecem serviços para dispositivos próximos dentro da mesma rede, nos moldes de uma nuvem tradicional — SaaS, DaaS, etc, independente de sistema operacional. Para que um aplicativo possa usufruir dos recursos do *framework*, ele deve ser projetado levando em consideração a modularização entre serviço e cliente de nuvem, o que adiciona uma certa complexidade ao projeto. Por um lado, a solução conta com mecanismos de privacidade e incentivos; e por outro lado, o *framework* pressupõe que haja um diretório central de *cloudlets* móveis dentro da rede, o que pode reduzir a disponibilidade do serviço caso, por exemplo, esse dispositivo esteja sobrecarregado ou fora da rede. Além disso, o consumo de energia do

[1] CyanogenMod: Um sistema operacional de código aberto derivado do Android. Atualmente está descontinuado

dispositivo que assume o papel de nó raiz aumenta com o passar do tempo.

A abordagem proposta em (LIAO *et al.*, 2015) propõe um *framework* para *offloading device-to-device*, voltado para melhoria tempo de execução em aplicações, em que o particionamento delas é feito em duas grandes camadas — *front end* e *back end*. Nesse trabalho, a porção *front end* da aplicação sempre é executada no lado cliente, e a porção *back end* da aplicação apresenta a possibilidade de ser executada em um dispositivo remoto. Esse *framework* usa o protocolo *OpenFlow*[1] implementado em cima de *Software Defined Networks* (SDN[2]) para abstrair a comunicação de rede entre os dispositivos. Entretanto, as implementações existentes do *OpenFlow* para dispositivos móveis são relativamente recentes, e ainda enfrentam alguns problemas em dispositivos móveis (KOLIAS *et al.*, 2013). Além disso, a divisão da aplicação entre duas grandes camadas (*front end* e *back end*) é um pressuposto para utilização da solução, de forma que para que um aplicativo possa se utilizar da solução proposta, ele deve passar por alterações de projeto que o encaixem nessa abordagem.

O *framework* Hyrax (TEO, 2012) é uma solução de *offloading device-to-device* voltado para *crowdsourcing*, que busca a melhoria tempo de execução em aplicações utilizando uma implementação de *Map Reduce* derivada do *Hadoop*[3]. O Hyrax busca dar suporte à formação de *clusters* compostos por dispositivos Android. Apesar da proposta de *crowdsourcing* com dispositivos móveis alcançar resultados satisfatórios, a arquitetura proposta conta com uma série de pressupostos que nem sempre podem ser cumpridos, tais como:

- A necessidade de um computador que age como nó central, que não pode falhar.
- Todos os dispositivos envolvidos no processamento já devem possuir os dados a serem processados de antemão.
- Os dados não podem sofrer alterações durante o processamento.

Além dos pressupostos de projeto, os aplicativos propensos a utilizarem o Hyrax devem implementar as funções *Map* e *Reduce* — cerne do paradigma do MapReduce — para as funcionalidades que devem ter as melhorias de tempo de execução.

O *framework ThinkAir* (KOSTA *et al.*, 2012) é uma solução desenvolvida para a plataforma Android, prezando o desenvolvedor, exigindo pouca mudança em código-fonte de aplicativos. A granularidade de particionamento de aplicação adotado é em nível de métodos, que devem ser anotados para que sejam considerados passíveis de *offloading*. No *ThinkAir*, o

[1] http://archive.openflow.org/
[2] https://www.opennetworking.org/sdn-definition/
[3] *http://hadoop.apache.org/*

código-fonte de aplicativos é pré-processado, para que os métodos anotados sejam alterados por um gerador de código que adiciona o suporte a *offloading*. O *ThinkAir* provê *offloading* para nuvens públicas ou *cloudlets*, e pode ser configurado para atuar de acordo com quatro políticas de decisão sobre a realização de *offloading*: Redução de tempos de execução, redução de consumo de energia, uma combinação das duas primeiras, e uma combinação das duas primeiras com o custo de uso de uma nuvem pública. O *framework* aloca múltiplos clones da máquina virtual da aplicação em nuvem pública, com o objetivo de melhorar os tempos de execução de 1 a 2 ordens de grandeza através do paralelismo de execução de tarefas em vários servidores. Como pontos negativos da solução, o *ThinkAir* exige que máquinas virtuais de diferentes versões do Android estejam disponíveis no servidor remoto. Além disso, a tomada de decisão complexa e as dependências de plataforma acabam por deteriorar parte da transparência no *offloading* em aplicações. (AHMED *et al.*, 2015)

A arquitetura *Anyrun Computing* (ARC), apresentada em (FERRARI *et al.*, 2016) propõe um modelo de *offloading* dinâmico e oportunista entre dispositivos Android, com foco em melhoria tempo de execução e de economia de energia, com particionamento de aplicação em nível de métodos. Para utilizar o ARC em uma aplicação, o desenvolvedor deve refatorar os códigos-fonte destas, de maneira que as classes do ARC sejam usadas para realizar as chamadas dos métodos dos quais deve ser feito o *offloading*. O ARC usa o *framework* SCAMPI(PITKÄNEN *et al.*, 2012) para abstrair a comunicação entre os dispositivos envolvidos no processo de *offloading*.

O *middleware* CAOS, apresentado em (GOMES *et al.*, 2017), é implementado para a plataforma Android, e promove o *offloading* de processamento para nuvem[1]. A granularidade de particionamento de aplicações adotada é a nível de métodos, que devem ser marcados pelo desenvolvedor com uma anotação Java, fornecida pelo *framework*. Métodos marcados são interceptados pelo middleware, que decide encaminhar ou não sua execução para a nuvem. Além de possibilitar o *offloading* de processamento para a nuvem, o CAOS também provê uma infraestrutura para o *offloading* de dados contextuais, com possibilidade de *crowdsensing*. O CAOS se mostra uma solução bastante aproximada do que é desejado para este trabalho, com a diferença de que o CAOS suporta *offloading* apenas para uma estrutura em nuvem, e não para dispositivos móveis. O CAOS D2D surge como uma versão estendida do CAOS, com suporte a *offloading device-to-device*[2].

[1] Ver seção 2.5 (CAOS), na página 28

[2] Ver seção 4.1 (Características do CAOS D2D), na página 39

3.1 Comparação entre os trabalhos relacionados

Os trabalhos CloneCloud (CHUN *et al.*, 2011) e COMET (GORDON *et al.*, 2012), apesar de terem focos de melhorias um pouco diferentes — o CloneCloud preza pelos ganhos em tempo de execução, enquanto o COMET busca além disso a economia de energia — apresentam semelhanças em suas abordagens de como implementar o processo de *offloading*. Ambas as soluções se baseiam em versões modificadas da Dalvik VM, equipadas de forma que elas mantenham em um *cloudlet* uma máquina virtual Android clone da máquina virtual em execução no dispositivo móvel. A principal diferença entre os trabalhos é que o CloneCloud transmite *threads* e seus resultados entre dispositivo móvel e *cloudlet*, enquanto o COMET sincroniza o estado da própria máquina virtual entre *cloudlet* e dispositivo móvel. O particionamento de aplicações é feito pela própria implementação da máquina virtual, sem interferência do desenvolvedor do aplicativo. Porém, o fato de necessitar de uma versão modificada de um componente fundamental do sistema operacional representa uma grande desvantagem, pois modificar o sistema operacional do dispositivo móvel é menos intuitivo para a maioria dos usuários do que a simples instalação de um aplicativo através dos repositórios de cada plataforma. Nesse aspecto, para utilizar o CAOS D2D, o escopo das alterações necessárias é a nível de aplicativo, de maneira independente do restante do sistema operacional, e isolado de outros aplicativos.

Os trabalhos (ARTAIL *et al.*, 2015), (LIAO *et al.*, 2015) e Hyrax (TEO, 2012) buscam possibilitar o *offloading* para dispositivos móveis, e têm em comum o requisito de que, apesar de dois dos trabalhos — (ARTAIL *et al.*, 2015) e (LIAO *et al.*, 2015) — serem multiplataforma, e os três trabalhos mencionados obterem bons resultados em seus *benchmarks*, eles necessitam que os aplicativos que estejam utilizando essas soluções estejam encaixados em um paradigma de aplicação exigido pelo *framework*, o que pode dificultar a adaptação de aplicativos para essas soluções. No que diz respeito ao particionamento de aplicações, o CAOS D2D adota o particionamento por métodos, de forma que as modificações que o desenvolvedor deve aplicar no código-fonte de aplicativos se resumem à adição de anotações nos métodos propensos a *offloading* — e da adição da biblioteca que contém tais anotações, representando portanto, modificações bem menos intrusivas. Os três trabalhos buscam melhorias em tempos de execução, sendo que o trabalho de Artail *et al.* (2015) também busca estender a capacidade de armazenamento do dispositivo móvel através de *offloading*.

O *ThinkAir* e o CAOS proveem *offloading* de computação a nível de métodos, e

ambos se baseiam em virtualização em nuvem pública ou *cloudlet*. Enquanto o *ThinkAir* fornece uma estrutura para paralelismo de tarefas em *offloading* para múltiplas máquinas virtuais remotas, o CAOS permite o *crowdsourcing* através do *offloading* de dados contextuais. No *ThinkAir* a maior parte da tomada de decisão é feita pelo dispositivo móvel cliente, enquanto no CAOS a maior parte da tomada de decisão é feita pelo servidor, através de dados de execução de outros clientes.

Os trabalhos ARC (FERRARI *et al.*, 2016), MAUI (CUERVO *et al.*, 2010) e CAOS (GOMES *et al.*, 2017) são *frameworks* que proporcionam o *offloading* através de uma camada de abstração para aplicativos móveis. O particionamento das aplicações é em nível de métodos, que são interceptados pelo *framework*, que realiza o devido tratamento nestes, de forma que o *middleware* associado se encarrega de invocar a execução dos métodos local ou remotamente. Apesar dessa semelhança, os três trabalhos apresentam grandes diferenças no que diz respeito a outros aspectos.

No MAUI e no CAOS, os métodos propensos a *offloading* devem ser marcados pelo desenvolvedor através de anotações no código-fonte; no ARC, as invocações de métodos a serem interceptadas pelo *framework* devem ser refatoradas, de forma a serem feitas pelas classes fornecidas. O ARC e o CAOS são desenvolvidos para Android, enquanto o MAUI é implementado para Windows Mobile. O foco do MAUI é a economia de energia, enquanto o foco do CAOS e ARC é, além disso, a melhoria tempo de execução nas aplicações. E finalmente, apesar do ARC porporcionar *offloading* de processamento para dispositivos móveis, MAUI e CAOS são projetados para o *offloading* para *cloudlets*.

A Tabela 1 apresenta de forma resumida as principais características de cada trabalho com relação a: restrições de sistema operacional, tipo de servidor, foco de melhoria da solução, abordagem de particionamento de aplicações e nível de esforço para adaptação de aplicativos.

O esforço para adaptar aplicativos de forma a usarem as funcionalidades de cada solução foi classificado como:

- Nenhum: Quando a aplicação não necessita de qualquer alteração em código-fonte para se tornar apta a fazer o *offloading*.
- Baixo: Quando as alterações em código-fonte se resumem apenas a adicionar bibliotecas e marcar os métodos passíveis de *offloading* com anotações, sem grandes refatorações.
- Médio: Quando o código interno de métodos deve ser refatorado, a fim de que ele use classes do *framework*, mas sem necessariamente alterar seu comportamento externo.

- Alto: Quando para um aplicativo usar a solução proposta, ele deve se encaixar em um paradigma de aplicação específico (*front end* e *back end*, cliente/servidor ou *MapReduce*), o que pode exigir um extenso trabalho de refatoração de código-fonte.

Tabela 1 – Comparação das principais características dos trabalhos citados

Trabalho	Sistema operacional	Tipo de servidor	Foco de melhoria	Abordagem de particionamento	Nível de esforço para adaptação
CloneCloud	Android	Nuvem	Tempo de execução	*Threads** (estático)	Nenhum
COMET	Android	Nuvem	Tempo de execução e economia energética	*Threads*** (estático)	Nenhum
Artail et. Al (2015)	Multiplataforma	Dispositivo móvel	Tempo de execução e armazenamento	Serviços	Alto
Liao, Qiu e Leung (2015)	Multiplataforma	Dispositivo móvel	Tempo de execução	Camadas Front end e back end	Alto
Hyrax	Android	Dispositivo móvel	Tempo de execução	Map reduce	Alto
MAUI	Windows Mobile	Nuvem	Economia energética	Métodos (dinâmico)	Baixo
ARC	Android	Dispositivo móvel	Tempo de execução e economia energética	Métodos (dinâmico)	Médio
ThinkAir	Android	Nuvem	Tempo de execução e economia energética	Métodos (estático e dinâmico)	Baixo
CAOS	Android	Nuvem	Tempo de execução e economia energética	Métodos (estático e dinâmico)	Baixo
CAOS D2D	Android	Dispositivo móvel	Tempo de execução e economia energética	Métodos (estático e dinâmico)	Baixo

* Através da migração de *threads* entre máquinas virtuais
** Através da sincronização de estado da máquina virtual Android
Fonte – Autoria própria

Após essa análise notamos que os trabalhos MAUI, ARC e CAOS apresentam requisitos mais próximos da proposta do CAOS D2D, que combina características desses três trabalhos. Ele é um *middleware* que busca melhorias na questão de tempos de execução e de economia de energia, através do *offloading* para dispositivos móveis próximos, dentro da mesma rede local. Ele é desenvolvido para a plataforma Android, e internamente intercepta métodos previamente marcados pelo desenvolvedor, que decide por transferir ou não sua execução para um outro dispositivo. Diferente do ARC, o CAOS D2D usa soluções presentes em bibliotecas nativas do Java para promover a comunicação entre os dispositivos móveis.

4 CAOS D2D

A iniciativa de desenvolver um *framework* para dar suporte a *offloading* entre dispositivos móveis surge como uma resposta aos problemas apresentados até aqui nessa dissertação. Este capítulo apresenta a solução proposta – CAOS D2D, suas características principais, arquitetura, metodologia e abordagens utilizadas para o seu desenvolvimento, bem como exemplos de utilização com código-fonte.

4.1 Características do CAOS D2D

O principal objetivo deste trabalho é apresentar um *framework* que possibilite o *offloading* de processamento para dispositivos móveis. Para alcançar esse objetivo, a abordagem escolhida para o desenvolvimento da solução foi a adaptação de componentes existentes no *framework* CAOS. Tal estratégia foi adotada levando em consideração as características levantadas de diferentes soluções disponíveis, apresentadas no Capítulo 3, e o fato de o CAOS — uma solução consolidada, com vários dos requisitos necessários para o CAOS D2D — ter sido desenvolvido dentro do GREat, o que favorece em questões como o acesso rápido a código-fonte e o contato mais próximo com pesquisadores envolvidos.

Assim como o CAOS, o CAOS D2D foi desenvolvido como um *framework* para o sistema operacional Android, e escrito usando a linguagem e bibliotecas Java. Pelo fato do CAOS D2D nascer como uma versão complementar do CAOS, ele foi projetado de maneira a manter a compatibilidade com a API do CAOS. Isso significa que um aplicativo que foi previamente desenvolvido ou adaptado para a utilização do CAOS não necessita de alterações em seu código-fonte para se tornar apto a se valer do CAOS D2D; apenas a atualização das bibliotecas é suficiente para isso.

No que diz respeito ao suporte de *offloading* do CAOS, a granularidade adotada pela solução é a nível de método, que deve ser marcado com a anotação *@Offloadable*[1], para que o *framework* possa identificar esse método como passível de ser feito *offloading* para outro dispositivo. O CAOS D2D suporta execução local ou remota de métodos anotados, dependendo da disponibilidade de servidores na mesma rede *Wi-Fi* do cliente.

A arquitetura do CAOS D2D está dividida basicamente em duas grandes partes[2]:

- **CAOS D2D API:** Essa é a parte cliente, que é integrada ao aplicativo. Ela tem a forma

[1] Ver seção 4.3 (Configuração do CAOS D2D), na página 49

[2] Ver seção 4.2 (Arquitetura do CAOS D2D), na página 41

de uma biblioteca Android que deve ser adicionada ao projeto do aplicativo, e deve ser configurada em tempo de projeto.

- **CAOS D2D Server:** Essa é a parte servidora, que se encontra em execução no dispositivo móvel que receberá as requisições de *offloading*. Ela tem a forma de um serviço Android a ser instalado no dispositivo servidor. Esse serviço é independente do aplicativo móvel que realiza o *offloading*, ou seja, o dispositivo móvel servidor não precisa ter instalado a mesma aplicação do dispositivo cliente.

A solução possui um mecanismo de descoberta de serviço, que é necessário para que um dispositivo móvel que esteja executando um aplicativo com o CAOS D2D API possa localizar, na mesma rede, dispositivos dispostos a fazer o papel de servidor. Diferente do que se pode imaginar a princípio para uma solução Java que se propõe a executar métodos remotamente, o CAOS D2D não utiliza o Java RMI, ou outra implementação existente de RPC. Em vez disso, ele realiza as trocas de mensagens entre o lado cliente e o servidor através de Java *Socket* para mensagens TCP, e do protocolo CoAP para mensagens UDP, e a execução remota dos métodos é realizada através de Java *Reflection*.

A oportunidade de realização de *offloading* para um dispositivo móvel presente na rede introduz a possibilidade de que haja mais de um servidor disponível para o qual um aplicativo possa fazer o *offloading* de processamento. Uma característica do CAOS D2D que o torna diferente do CAOS é o fato de ele ter sido projetado levando tal possibilidade em consideração. O CAOS D2D API mantém um registro de servidores disponíveis na rede, bem como suas características. Isso abre a possibilidade de extensão do CAOS D2D para a implementação de módulos mais robustos de tomada de decisão.

Para atender melhor à maior variedade de dispositivos servidores, o CAOS D2D permite que um dispositivo móvel fazendo papel de servidor receba dinamicamente as dependências necessárias para possibilitar o recebimento das requisições de execução de método do aplicativo cliente, caso o servidor não possua tais dependências. Essa injeção (ou *deploy*) é feita do cliente para o servidor, de maneira dinâmica. Vale frisar que em um cenário dinâmico proporcionado por inúmeros dispositivos que potencialmente podem assumir o papel de servidores, o *deploy* dinâmico de dependências é um requisito importante do CAOS D2D, que não existia no CAOS, portanto os componentes relacionados à *deploy* de dependências foram desenvolvidos desde o princípio para essa solução. O componente *Deploy Client* age de maneira transparente para o usuário, que não precisa adicionar as dependências em um diretório de recursos no projeto do

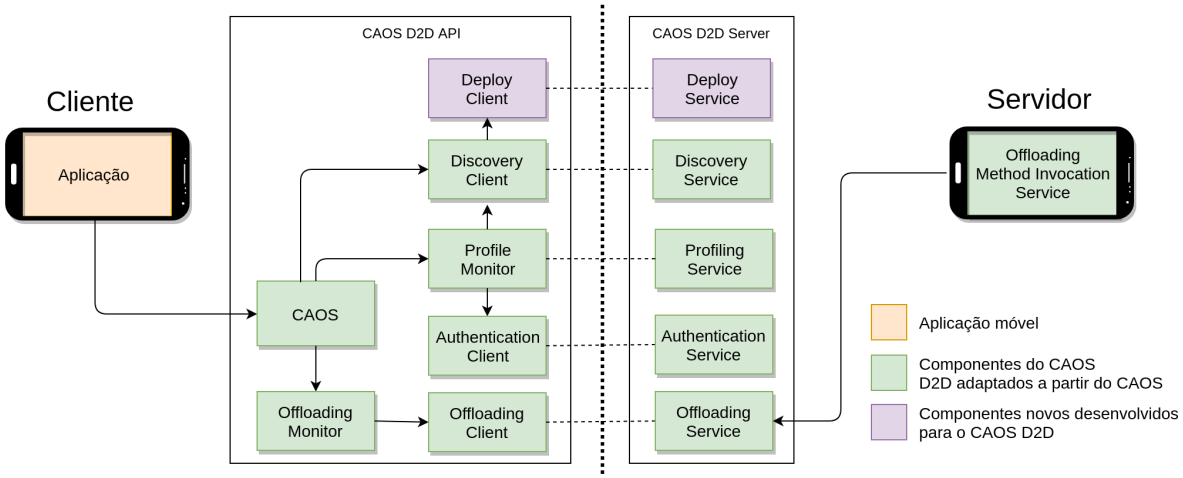
aplicativo.

O escopo definido para o desenvolvimento desse trabalho foi limitado ao aspecto do *offloading* para dispositivos móveis, portanto, não foram implementados mecanismos de segurança e privacidade no protótipo atual, e foi implementado um mecanismo simplificado de tomada de decisão.

4.2 Arquitetura do CAOS D2D

O CAOS D2D possui componentes que são executados no dispositivo móvel cliente e no dispositivo móvel que fará o papel de servidor. A Figura 6 ilustra a arquitetura geral do CAOS D2D, e como os seus componentes se comunicam entre si e com o aplicativo móvel. Esses componentes estão divididos em duas grandes partes: O CAOS D2D API é a parte cliente da solução; a porção que é associada com o aplicativo do qual se deseja realizar o *offloading* de métodos. Ele compreende uma biblioteca Java com classes e anotações que abstraem para o desenvolvedor as tarefas de baixo nível relacionadas com o *offloading*. A parte servidora da solução é o CAOS D2D Server, que encapsula serviços que possibilitam a um dispositivo móvel atuar como servidor de *offloading* de processamento.

Figura 6 – Visão geral da arquitetura do CAOS D2D



Fonte – Autoria própria

A arquitetura do CAOS D2D é composta de componentes que, em sua maioria, são formados por módulos correspondentes em ambas as partes da solução, ou seja, componentes que são formados por módulos presentes tanto no lado cliente quanto no lado servidor, de forma que esses módulos se comunicam entre si, através da abordagem cliente/servidor. Internamente, os módulos presentes no CAOS D2D API são os que tem a iniciativa para iniciar todos os fluxos

de comunicação da solução. É o CAOS D2D API que tem a responsabilidade de:

- Anunciar que está presente na rede;
- Se autenticar junto ao CAOS D2D Server;
- Injetar dependências no CAOS D2D Server;
- Disparar pacotes de rede para tarefas de *profiling*;
- Iniciar o processo de *offloading*.

Já o CAOS D2D Server responde aos estímulos do cliente. Ele tem a responsabilidade de:

- Oferecer os serviços na rede, e mantê-los disponíveis;
- Se manter visível para dispositivos clientes;
- Receber requisições do cliente, tratá-las de acordo, e responder adequadamente;
- Receber dependências de aplicações de clientes, e armazená-las no sistema de arquivos do dispositivo servidor, para se tornar apto a realizar o *offloading* para essas aplicações.
- Realizar o *offloading* de processamento de aplicativos de clientes sob demanda, e retornar o seu resultado.

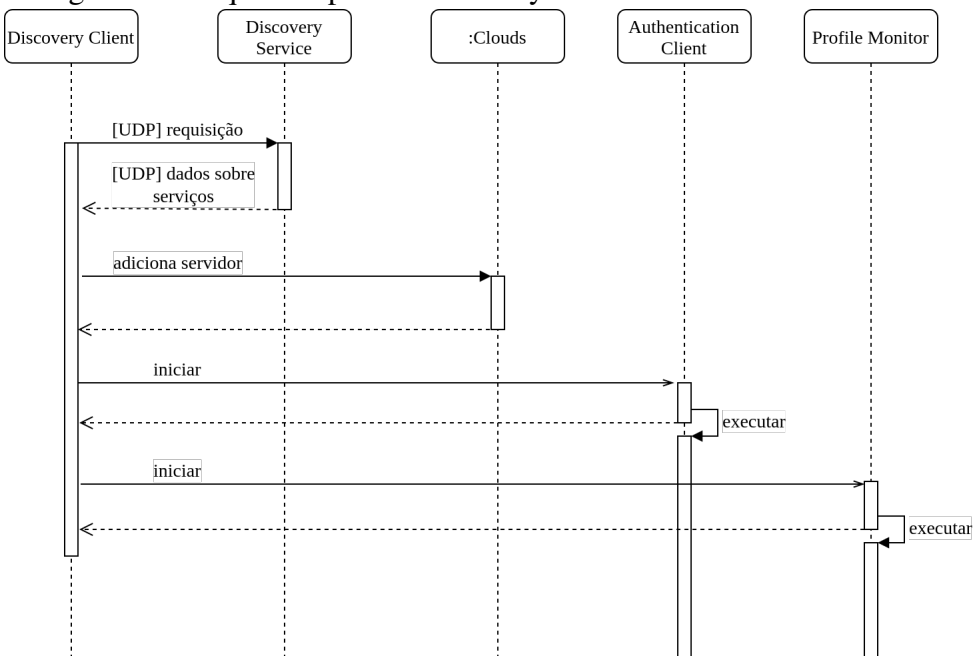
Os principais componentes da solução são: *Discovery Service*, *Authentication Service*, *Profile Monitor*, *Offloading Monitor*, *Offloading Service* e *Deploy Service*.No dispositivo móvel servidor, após a inicialização do aplicativo CAOS D2D Server, todos os módulos que hospedam serviços permanecem de prontidão, no aguardo de uma mensagem enviada pelos módulos clientes que estiverem em execução no CAOS D2D API, em algum outro dispositivo móvel. A seguir descrevemos os componentes do CAOS D2D, e apresentamos seus fluxos de funcionamento, ilustrados com diagramas de atividades que mostram a comunicação entre os módulos.

4.2.1 Discovery Service

O Serviço e o Cliente de Descoberta — *Discovery Service* no servidor e *Discovery Client* no cliente — são os componentes do CAOS D2D que tornam possível que dispositivos cliente e servidor possam se identificar dentro de uma rede local.

A Figura 7 ilustra o fluxo de funcionamento do componente. O módulo *Discovery Client* envia uma mensagem UDP em *Broadcast* na rede local, de forma que todos os dispositivos na rede recebam essa mensagem, com o objetivo de descobrir servidores CAOS D2D (ou CAOS) disponíveis na rede. https://www.sharelatex.com/project/59426b9027f68c2e215380f8

Figura 7 – Diagrama de sequência para o Discovery Service



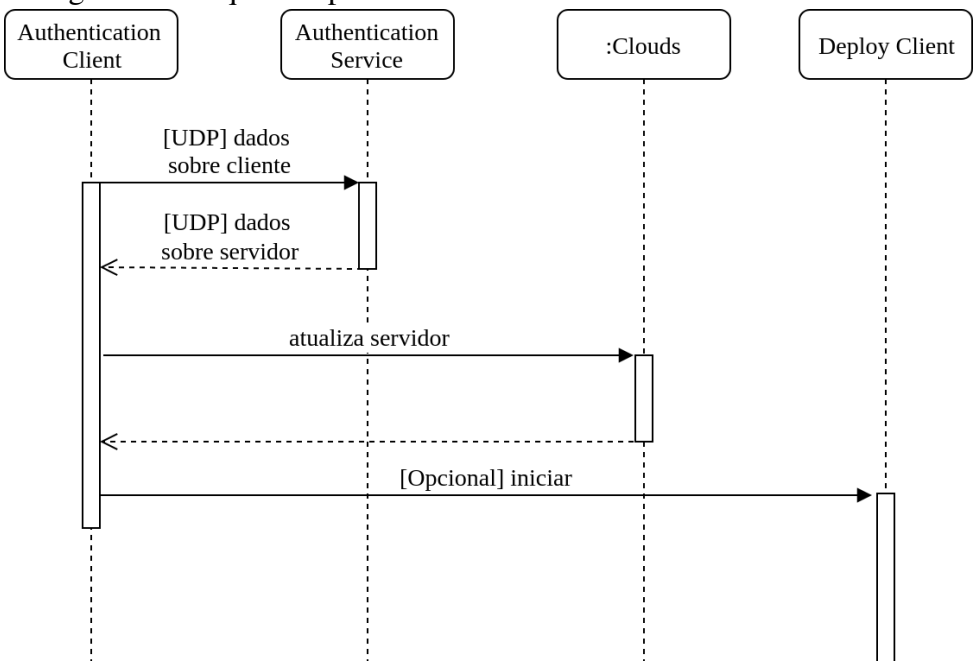
Fonte – Autoria própria

Ao receber a mensagem *Broadcast*, o *Discovery Service* considera que descobriu um *Discovery Client* dentro da rede, e envia de volta para ele uma mensagem TCP com informações sobre o seu endereço IP, e as portas onde os demais serviços do CAOS D2D Server estão sendo executados.

O *Discovery Client* recebe essa mensagem, e com essas informações o lado cliente do CAOS D2D pode então iniciar a comunicação com os demais serviços do dispositivo servidor localizado. Tendo os dados sobre os serviços em execução no CAOS D2D Servidor, o *Discovery Client* inicia os módulos *Authentication Client* e *Profile Monitor*. O *Discovery Service* repete esse fluxo periodicamente, para que o CAOS D2D API esteja sempre atualizando seus registros sobre dispositivos móveis servidores.

O suporte a mais de um servidor é uma característica do CAOS D2D que não havia no CAOS original. Esse suporte foi adicionado devido ao aspecto da mobilidade de dispositivos. No CAOS D2D, foi implementada uma classe especial, denominada *Clouds*, que trata de armazenar de maneira estática informações sobre os diferentes dispositivos servidores disponíveis na rede. Na atual implementação do CAOS D2D, em caso de mais de um servidor disponível na rede, o *framework* opta por realizar o *offloading* no primeiro dispositivo que esteja armazenado na estrutura de dados interna da classe *Clouds*.

Figura 8 – Diagrama de sequência para o Authentication Service



Fonte – Autoria própria

4.2.2 Authentication Service

O Serviço de Autenticação — *Authentication Service* e *Authentication Client* — tem como função permitir que os dispositivos cliente e servidor se identifiquem um para o outro. No lado cliente, o *Authentication Client* envia uma mensagem para o *Authentication Service* no lado servidor, com informações sobre o *hardware* do dispositivo cliente — modelo e marca do aparelho, informações sobre o processador e memória RAM, aplicativo e versão sendo executados no cleinte. Ao receber essa mensagem, o *Authentication Service* envia de volta para o *Authentication Client* uma mensagem com dados sobre o servidor — O tipo de servidor (*cloudlet* ou *smartphone*), o nível de carga da bateria, se o aparelho está ligado a uma fonte de energia, e se o servidor possui as dependências do aplicativo executado pelo cliente. Quando recebe essa mensagem, o *Authentication Client* armazena os dados sobre o na classe *Clouds*. Caso o servidor encontrado não possua as dependências do aplicativo, o *Authentication Client* faz uma chamada ao módulo *Deploy Service*, para iniciar o *deploy* de dependências no dispositivo servidor. A Figura 8 ilustra o fluxo descrito aqui.

4.2.3 Deploy Service

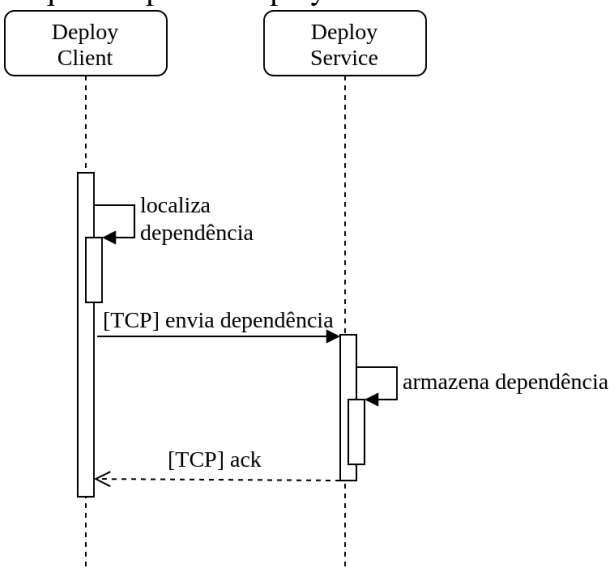
O Serviço de *Deploy* de Dependências — *Deploy Service* e *Deploy Client* — é responsável por realizar a injeção de dependências em formato APK dinamicamente do dispositivo

cliente para o servidor, de forma que este se torne apto a receber chamadas de *offloading* feitas pelo cliente.

Quando o *Authentication Client* recebe a infomação de que o servidor encontrado não possui as dependências para receber o *offloading* de métodos da aplicação em execução no cliente, o *Deploy Client* envia para o *Deploy Service* em execução no CAOS D2D Server uma mensagem contendo: nome, pacote e versão do aplicativo, bem como o arquivo APK deste. O arquivo APK enviado é extraído diretamente do diretório de APK's instalados no sistema Android. Portanto, não é necessário que seja adicionado nenhum arquivo APK ou JAR redundante ao projeto do aplicativo com essa finalidade.

O *Deploy Service* recebe a mensagem enviada pelo *Deploy Client* e armazena o arquivo APK em um diretório no sistema de arquivos, para que o servidor esteja apto a executar adequadamente as chamadas de execução remota feitas pelo cliente. O fluxo de funcionamento do *Deploy Service* está ilustrado na Figura 9.

Figura 9 – Diagrama de sequência para o Deploy Service

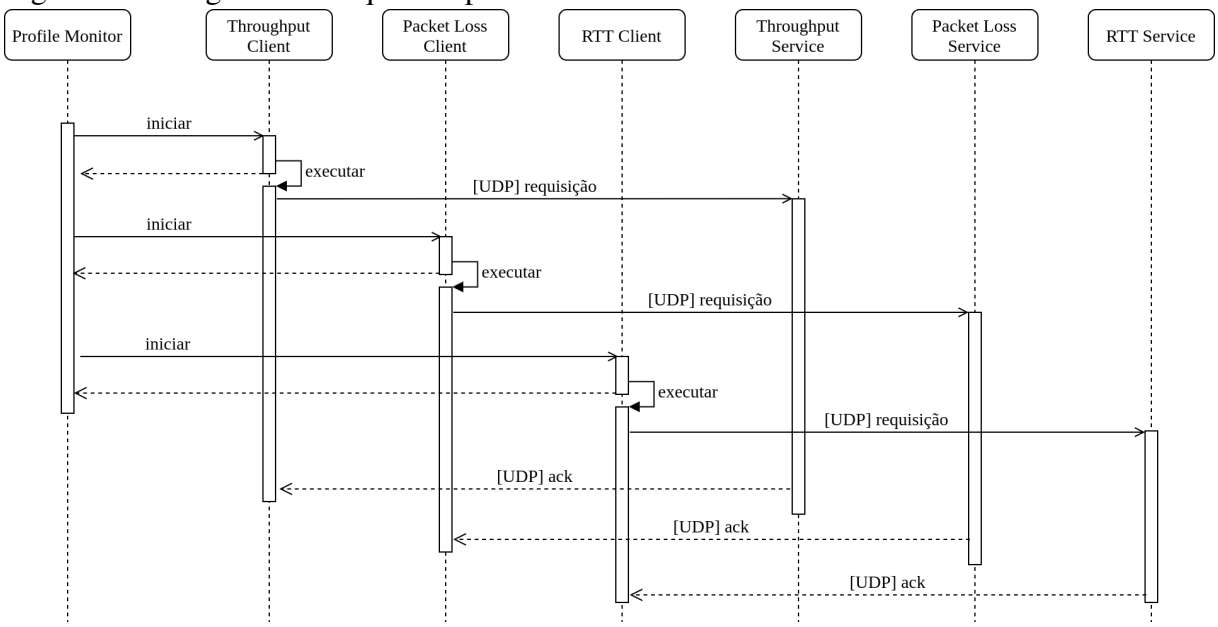


Fonte – Autoria própria

4.2.4 Profile Service

O Serviço de Monitoramento — *Profile Service* e *Profile Monitor* — é o módulo responsável por realizar uma série de tarefas relacionadas ao monitoramento da qualidade e velocidade de comunicação entre os dispositivos móveis cliente e servidor. A Figura 10 ilustra o fluxo de funcionamento do componente.

Figura 10 – Diagrama de sequência para o Profile Service



Fonte – Autoria própria

Ao ser invocado pelo *Discovery Client*, o *Profile Monitor* inicia os submódulos *Throughput Client*, *Packet Loss Client*, e *RTT Client*. As responsabilidades desses submódulos são, respectivamente, mensurar a velocidade de conexão, a taxa de perda de pacotes, e a latência de comunicação entre cliente e servidor.

Os três submódulos *Throughput Client*, *Packet Loss Client*, e *RTT Client* também possuem módulos correspondentes no CAOS D2D Server, respectivamente: *Throughput Service*, *Packet Loss Service*, e *RTT Service*. Internamente, esses submódulos apresentam comportamentos similares, com objetivos distintos. O submódulo cliente envia mensagens para o submódulo servidor, que envia uma mensagem de resposta para o submódulo cliente. O conteúdo, a quantidade, e o tamanho dessas mensagens são definidos especificamente para a finalidade de cada submódulo, seja para mensurar a velocidade de conexão, a taxa de perda de pacotes, ou a latência de comunicação entre dispositivos cliente e servidor. Os submódulos do Profile Service executam suas rotinas periodicamente, e o fluxo de execução do componente está ilustrado na Figura 10.

4.2.5 Offloading Monitor

O *Offloading Monitor* é o componente responsável por monitorar a execução das aplicações e interceptar o fluxo de execução de métodos marcados. No CAOS D2D (e no CAOS), os métodos são marcados com anotação Java (*@Offloadable*), o qual define que o método é apto

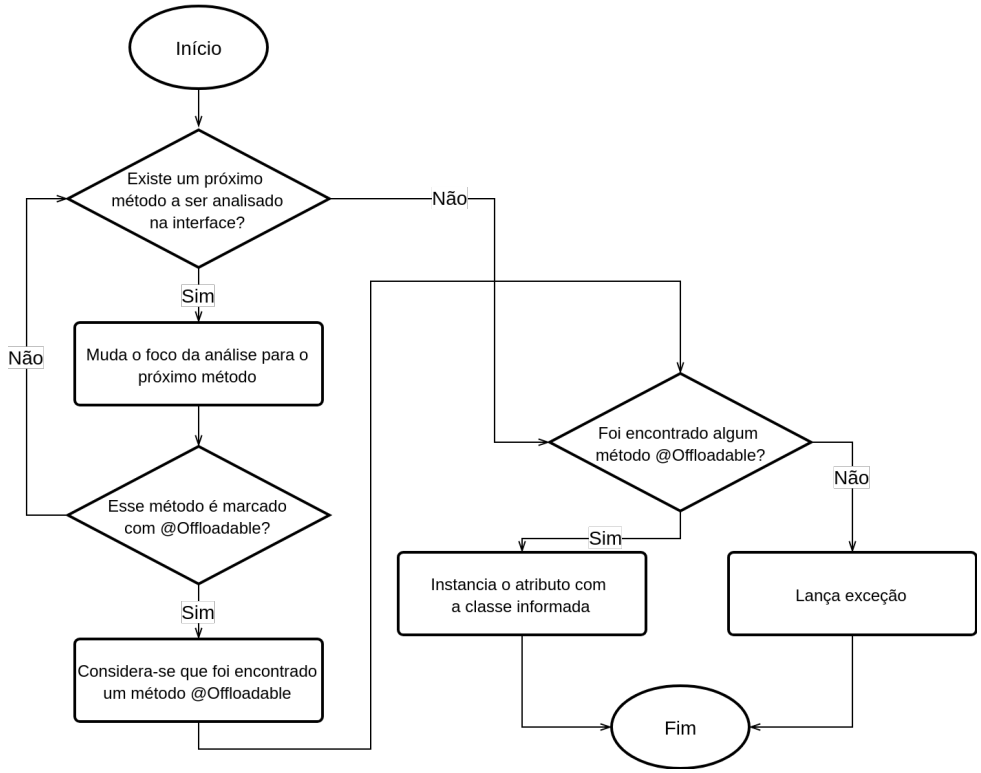
para execução preferencialmente fora do dispositivo móvel. O *Offloading Monitor* entra em ação em dois momentos específicos: na inicialização do *framework*, para instanciar classes com métodos passíveis de *offloading*, e no momento em que um método marcado com @*Offloadable* é invocado, para que essa a execução do método seja interceptada, e o *Offloading Monitor* possa verificar se deve ou não realizar o *offloading*.

Para gerenciar o *Offloading Monitor*, o CAOS D2D API conta com uma classe denominada *Caos*, projetada segundo o padrão de projeto *Singleton*(GAMMA, 1995). Esse padrão foi adotado para que haja controle sobre os serviços em execução e instâncias geradas pelo *framework*. Essa classe possui dentre outros, dois métodos principais: O método *start* é responsável por instanciar os atributos marcados com a anotação @*Inject*, e por iniciar o *Discovery Service*; e o método *stop* interrompe os serviços em execução, para poupar recursos do dispositivo ou para encerrar o aplicativo em execução.

Na inicialização do *framework* — a chamada ao método *start* da classe CAOS, o *framework* primeiramente verifica quais são os atributos da *Activity* que estão marcados com a anotação @*Inject*. A Figura 11 ilustra o fluxo de funcionamento do módulo. Para cada um dos atributos encontrados, é feita uma verificação quanto ao tipo do atributo, para averiguar se eles correspondem a interfaces com métodos marcados com a anotação @*Offloadable*. Em caso positivo, o atributo analisado é então instanciado usando a classe concreta utilizada como parâmetro para a anotação @*Inject*. Caso o atributo não seja de uma interface com métodos marcados com @*Offloadable*, é lançada uma exceção que informa ao desenvolvedor que é necessário marcar os métodos passíveis de *offloading* com a anotação @*Offloadable*.

No momento em que um método marcado com @*Offloadable* é invocado, o *Offloading Monitor* intercepta a execução do método e analisa se há algum servidor CAOS D2D disponível e com as dependências necessárias na mesma rede. Se a resposta for negativa, o fluxo de execução do método é retomado e sua execução continua localmente. Caso contrário, o *Offloading Monitor* instancia um objeto que encapsula o método invocado, e os argumentos repassados; e envia esse objeto ao *Offloading Client*, para que ele inicie o processo de *offloading* do método, enquanto o *Offloading Monitor* aguarda o seu resultado. Quando o *Offloading Client* envia o resultado do *offloading*, o *Offloading Monitor* retoma a sua execução, e retorna esse resultado para a entidade que invocou o método originalmente. Caso haja alguma exceção disparada pelo *Offloading Client* durante o *offloading* (por exemplo, um problema de tempo limite de conexão), o *Offloading Monitor* parte para a execução local do método invocado, e

Figura 11 – Fluxograma simplificado de análise de interfaces no Offloading Monitor



Fonte – Autoria própria

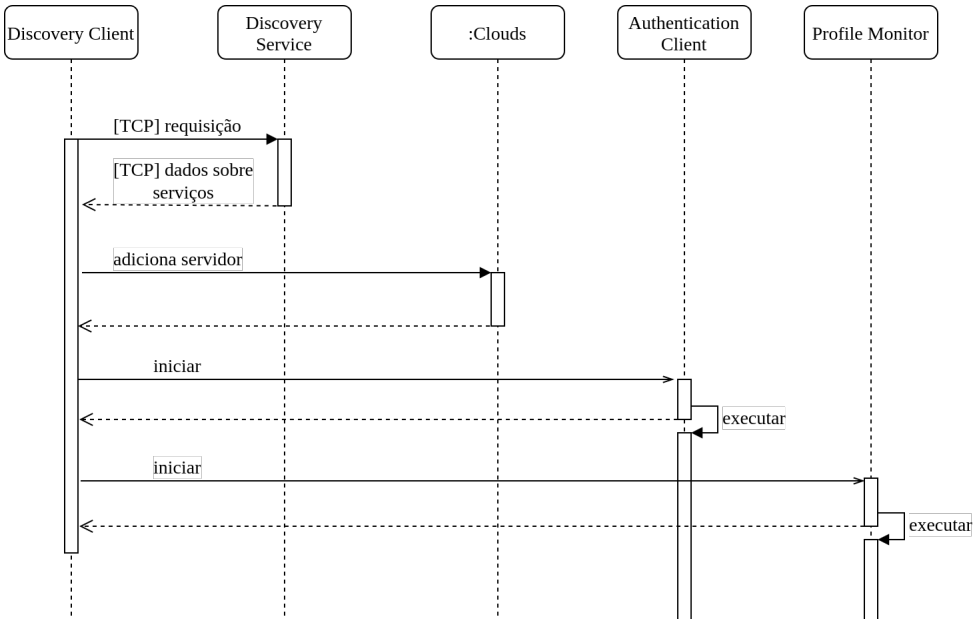
retorna o seu resultado.

4.2.6 Offloading Service

O Serviço de *Offloading* — *Offloading Client*, *Offloading Service* e *Offloading Method Invocation Service* — trata efetivamente de encaminhar a execução de um método de um dispositivo cliente para um servidor. A Figura 12 ilustra o fluxo de execução do componente, e sua comunicação com o *Offloading Monitor*. No lado cliente, o *Offloading Client* recebe do *Offloading Monitor*, uma mensagem contendo o método a ser executado e os argumentos a serem repassados. O *Offloading Client* então envia para o *Offloading Service* no lado servidor uma mensagem contendo dois objetos: um que encapsula nome e versão do aplicativo em execução no lado cliente, e o objeto recebido do *Offloading Monitor*, que encapsula método e argumentos.

No lado servidor, ao receber a mensagem enviada pelo *Offloading Client*, o *Offloading Service* utiliza o primeiro objeto para, de posse do nome do aplicativo e da versão deste, construir um objeto *ClassLoader* que tem como dependência raiz o arquivo APK referente a esse aplicativo, previamente armazenado no sistema de arquivos. Tendo esse *ClassLoader* preparado, o *Offloading Service* envia o *ClassLoader* e o método encapsulado para o *Offloading Method*

Figura 12 – Diagrama de sequência para o Offloading Monitor, Offloading Service e Remote Method Invocation Service



Fonte – Autoria própria

Invocation Service que, através de técnicas de reflexão computacional, trata efetivamente da execução do método e de retornar o seu resultado para o *Offloading Service*, que envia esse resultado de volta para o *Offloading Client*, no lado cliente.

O *Offloading Client*, ao receber a mensagem contendo o resultado do processamento, "desempacota" essa mensagem e envia o objeto descompactado para o *Offloading Monitor*, que retoma o seu fluxo de execução.

4.3 Configuração do CAOS D2D

Para integrar as funcionalidades do CAOS D2D em um aplicativo, o desenvolvedor deste deve definir um conjunto mínimo de configurações. Primeiramente, para ter acesso às anotações e classes do *framework*, deve-se importar para o projeto do aplicativo a biblioteca do CAOS D2D API. Em seguida, o desenvolvedor deve criar ou adaptar interfaces com métodos que podem ser propensos a *offloading*. Se um método da interface for propício a execução em *offloading*, o desenvolvedor deve aplicar neste a anotação @*Offloadable*, de forma que o CAOS D2D possa localizar e interceptar esse método. O Código-fonte 4.3.1 ilustra um exemplo de interface com um método marcado para *offloading* com a interface @*Offloadable*.

Código-fonte 4.3.1 – Exemplo de interface com método anotado com @*Offloadable*

```
import br.ufc.great.caos.api.offload.Offloadable;

public interface Calculator  {
    @Offloadable
    public int sum(int a, int b);
}
```

Após definir as interfaces que contém métodos passíveis de *offloading*, o desenvolvedor deve implementar essas interfaces, caso elas não já estejam implementadas. Essa implementação pode ser codificada normalmente, sem alterações de código-fonte, o que significa que tais classes não precisam de ser refatoradas, caso já existam.

Para preparar (configurar) o mecanismo de varredura de métodos marcados com @*Offloadable*, o próximo passo que o desenvolvedor deve seguir é aplicar a anotação @*CaosConfig* na classe *Activity* principal do seu aplicativo. Isso é necessário para que a inicialização do *framework* ocorra juntamente com a inicialização da *Activity*.

Para instanciar os atributos da *Activity* que implementam as interfaces com métodos marcados, na *Activity* principal, deve-se utilizar a anotação @*Inject* nas declarações dos atributos que implementem essas interfaces. Em vez de o próprio desenvolvedor instanciar esses atributos, o *framework* CAOS D2D deve instanciá-los, para que eles possam posteriormente ser interceptados pelo *Offloading Service*. A anotação @*Inject* recebe como parâmetro o objeto *Class* referente a uma classe que instancie a interface anotada.

Por fim, para iniciar o mecanismos de varredura de métodos, descoberta de serviço e monitoramento, deve-se invocar o método *start* da classe Caos, de preferência em algum momento da inicialização da *Activity* principal, como por exemplo, no método *onCreate* ou *onResume*. O método *Caos.start* recebe dois argumentos: um objeto que contém atributos instanciados com a anotação @*Inject*, e um objeto *Context*. Na maioria dos casos, a *Activity* é o objeto com os atributos a serem injetados, e ela é seu próprio *Context*. Nesses casos é seguro repassar o próprio objeto *Activity* para os dois argumentos (através da palavra reservada “this”).

Para encerrar os serviços iniciados com método *start*, e com isso liberar recursos do dispositivo e poupar energia, o desenvolvedor deve invocar o método *stop*, preferencialmente na fase de pausa ou destruição da *Activity*. Por exemplo, no método *onPause*, *onStop* ou *onDestroy*. A Código-fonte 4.3.2 ilustra o uso dessas chamadas de método em uma classe *Activity* simples.

Código-fonte 4.3.2 – Exemplo de *Activity* com *@CaosConfig*

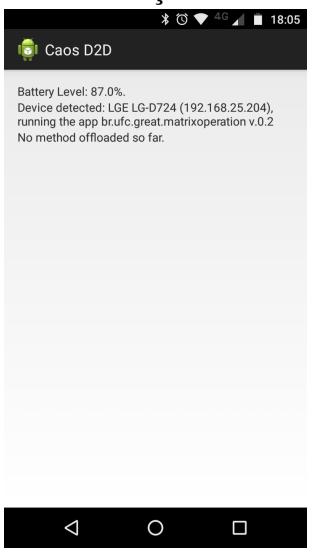
```
import br.ufc.great.caos.api.Caos;
import br.ufc.great.caos.api.config.CaosConfig;
import br.ufc.great.caos.api.config.Inject;

@CaosConfig
public class MainActivity extends Activity {
    //Classe que implementa Calculator
    @Inject(CalculatorImp.class)
    //Interface marcada com a anotação @Offloadable
    Calculator calc;
    //Demais atributos e métodos
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //Demais operações de inicialização
        //Argumentos: um objeto das quais os atributos
        //serão avaliados pelo CAOS, e um Context
        Caos.getInstance().start(this, this);
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        Caos.getInstance().stop();
    }
}
```

Fazendo os passos descritos acima, as tarefas de monitoramento, decisão e o *off-loading* dos métodos anotados estão aptos a serem realizadas de maneira transparente para o usuário. Os métodos marcados com *@Offloadable* podem ou não ser executados remotamente, dependendo da disponibilidade de servidores descobertos na mesma rede do cliente.

No lado servidor, tudo que o usuário deve fazer é instalar um aplicativo Android, que encapsula todos os módulos do *CAOS D2D Server*, e o seu dispositivo móvel estará apto a

Figura 13 – Captura de tela do protótipo do CAOS D2D Server, após localizar um potencial dispositivo cliente em sua vizinhança



Fonte – Autoria própria

receber dependências e invocações de métodos de clientes que estejam próximos na mesma rede sem fio. A Figura 13 ilustra uma captura de tela do protótipo do CAOS D2D Server.

4.4 Processo de execução do CAOS D2D

Nessa seção será descrito brevemente o funcionamento de uma aplicação móvel configurada para uso do CAOS D2D, conforme os passos descritos na seção anterior. Para esse cenário estamos considerando as seguintes condições:

- O aplicativo cliente está configurado conforme descrição na seção 4.3;
- O aplicativo servidor está instalado em um dispositivo móvel diferente do dispositivo móvel cliente;
- O dispositivo móvel que está agindo como servidor não possui o aplicativo cliente;
- Os dois dispositivos móveis utilizados estão dentro da mesma rede sem fio.

Na inicialização do *framework* ele instancia os atributos da *Activity* que estão marcados com a anotação *@Inject*. Ainda durante a inicialização do *framework*, é inicializado o módulo de descoberta. Esse serviço envia uma mensagem em *broadcast* para todos os dispositivos na rede, informando que há um cliente CAOS D2D na rede.

No dispositivo móvel servidor, ao receber essa mensagem de descoberta de serviço, o *CAOS D2D Server* responde ao dispositivo cliente com uma mensagem contendo informações sobre como se comunicar com os demais serviços do CAOS D2D hospedados naquele dispositivo móvel.

No dispositivo móvel cliente, se o módulo de descoberta receber uma resposta de algum outro dispositivo, ele armazena as informações contidas nessa mensagem em uma estrutura de dados que armazena informações sobre todos os dispositivos servidores encontrados, e inicia a comunicação com os serviços de autenticação e *profiling* do dispositivo móvel servidor localizado.

O módulo de *profiling* do *CAOS D2D API* inicia uma série de testes de comunicação entre os dispositivos cliente e servidor, para investigar aspectos referentes à qualidade de comunicação entre cliente e servidor, como velocidade da rede e taxa de perda de pacotes.

Ao ser inicializado no dispositivo móvel cliente, o módulo de autenticação envia para o serviço de autenticação hospedado no dispositivo servidor uma mensagem contendo informações sobre, entre outros, a marca e modelo do dispositivo, e nome, pacote e versão do aplicativo executando o CAOS D2D. No lado servidor, no momento em que o serviço de autenticação recebe essa mensagem, este serviço envia de volta uma mensagem contendo dados sobre: O tipo do dispositivo servidor (nuvem CAOS ou dispositivo móvel); o estado da energia desse dispositivo móvel – se está conectado à corrente elétrica e recarregando, ou se está descarregando, e qual o nível de bateria restante no momento; e se o servidor tem ou não a dependência para o nome e versão de aplicativo informado pelo cliente.

No dispositivo cliente, no momento em que o módulo de autenticação do *CAOS D2D API* recebe a mensagem resposta do serviço de autenticação, ele armazena os dados sobre o tipo de servidor e o status de bateria na estrutura de dados de servidores. Caso o servidor não conte com a dependência para o aplicativo que está sendo executado no servidor, o módulo de *deploy* de dependências é inicializado. Caso o servidor já possua tal dependência, o CAOS D2D API está pronto para realizar o *offloading* de métodos para o CAOS D2D Server.

Caso seja iniciado, o módulo de injeção de dependências do *CAOS D2D API* envia para o serviço de injeção de dependências do *CAOS D2D Server* uma mensagem contendo o nome e a versão do aplicativo em execução, e o arquivo APK referente a esse aplicativo, extraído diretamente do diretório de instalação do sistema Android. Ao receber essa mensagem, o serviço de injeção de dependências do CAOS D2D Server armazena o arquivo APK recebido no sistema de arquivos, para ser utilizado posteriormente.

Os passos descritos acima se referem a procedimentos preparatórios para o processo de *offloading*. Após a execução desse plano, tanto o CAOS D2D API quanto o *CAOS D2D Server* estão prontos para o *offloading* de métodos entre si. No momento da chamada de um

método propenso a *offloading* no aplicativo cliente, o módulo de monitoramento de *offloading* intercepta a execução desse método e verifica se há algum servidor disponível em rede que possua a dependência desse aplicativo. Em caso positivo, o módulo de monitoramento instancia um objeto que encapsula o nome e versão do aplicativo, o método invocado, e os argumentos repassados; e envia esse objeto em uma mensagem através da rede para o serviço de *offloading* no CAOS D2D Server.

Ao receber a mensagem mencionada, o serviço de *offloading* do CAOS D2D Server utiliza a informação de nome e versão do aplicativo cliente para localizar o arquivo APK referente à dependência do aplicativo no sistema de arquivos, e utiliza esse arquivo, juntamente com técnicas de reflexão de código para invocar o método chamado pelo cliente, com os argumentos que ele repassou. Ao fim da execução do método, o serviço de *offloading* envia para o módulo de *offloading* no CAOS D2D API um objeto que encapsula o retorno do método.

O módulo de *offloading* do CAOS D2D API, após receber a mensagem de resposta do serviço de *offloading*, extrai dessa mensagem o resultado da execução do método e o retorna, como se o método tivesse sido executado localmente.

Caso o CAOS D2D API não tenha detectado nenhum dispositivo móvel servidor, ou caso nenhum dispositivo móvel servidor localizado possua a dependência necessária para execução dos métodos, o CAOS D2D API procede para a execução local do método invocado, e retorna o seu resultado normalmente.

4.5 Processo de desenvolvimento do protótipo

Pelas razões apresentados na seção 4.1, a estratégia adotada para a implementação do CAOS D2D foi a adaptação de componentes existentes no CAOS. Para implementação do lado servidor do CAOS D2D, a abordagem adotada a princípio foi partir da implementação existente do *Offloading Method Invocation Service* — componente do CAOS que possui a forma de um aplicativo Android, residente na máquina virtual que executa os métodos remotos — e através de adaptações em seu código-fonte, adicionar serviços a ele, com responsabilidades similares aos serviços desempenhados pelo *cloudlet* CAOS. Essa extensão do *Offloading Method Invocation Service* teve como finalidade estender esse componente e torná-lo um aplicativo autônomo, capaz de se comunicar com todos os módulos do CAOS D2D API.

Para a implementação do lado cliente, o ponto de partida foi o *CAOS API*, o lado cliente do CAOS. A partir dele, foram feitas adaptações em todos os seus módulos, a fim de que

eles pudessem se comunicar com as versões adaptadas dos módulos servidores, presentes no CAOS D2D Server.

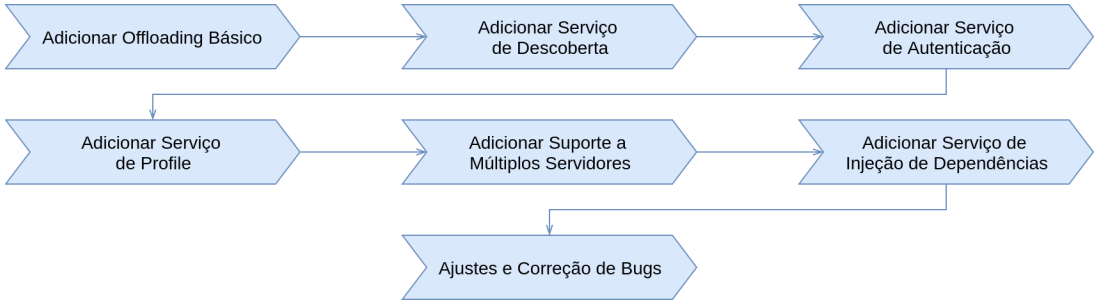
As implementações dos componentes que são formados por módulos cliente e servidor foram realizadas um a um, de maneira incremental, e de forma que o desenvolvimento dos módulos cliente e servidor de cada componente fosse realizado em conjunto. Por exemplo: Durante o desenvolvimento do componente *Discovery Service*, o esforço de desenvolvimento foi focado apenas nos módulos cliente e servidor desse componente. Os módulos cliente e servidor foram trabalhados e testados até que estivessem se comunicando normalmente e realizando os seus fluxos de execução. Apenas após essa homologação e verificação do *Discovery Service*, o desenvolvimento desse módulo foi considerado encerrado, e seguiu para outro módulo.

A seguir são apresentadas todas as etapas definidas para a implementação do protótipo do CAOS D2D. Será exposto como essas etapas foram seguidas, e qual o resultado obtido de cada uma delas.

4.5.1 Etapas de desenvolvimento

O desenvolvimento do protótipo do CAOS D2D foi realizado de forma incremental, dividido em marcos, cada um com um objetivo específico, conforme ilustrado na Figura 14. Aqui apresentamos esses marcos, como eles foram conduzidos, e qual o resultado final de cada um deles.

Figura 14 – Etapas de implementação do protótipo do CAOS D2D



Fonte – Autoria própria

Primeiramente, com o objetivo de realizar o *offloading* de métodos diretamente de um dispositivo móvel para outro, foi realizado um trabalho no *CAOS D2D Server* para que fosse incorporado nele o módulo *Offloading Service*, de forma a torná-lo capaz de receber as requisições de *offloading* e executá-las, atuando como um servidor de *offloading* relativamente independente. No CAOS D2D API, foram feitas alterações no componente *Offloading Client*,

para que ele pudesse se comunicar com o módulo *Offloading Service* recém-implementado. Ao final desse marco, o CAOS D2D era capaz de realizar *offloading device-to-device* de métodos, porém apenas em um ambiente controlado; os endereços IP dos dispositivos móveis cliente e servidor precisavam estar codificados diretamente nos módulos implementados, e não havia tratamento de erros em caso de problemas.

Com o propósito de eliminar a limitação dos IP's pré-determinados, o próximo passo do desenvolvimento no *CAOS D2D Server* foi implementar nele o *Discovery Service*, para permitir a comunicação com o *Discovery Client*, que por sua vez, foi adaptado no *CAOS D2D API* para que a comunicação entre os dois módulos pudesse transcorrer sem problemas. Ao final desse marco, o *CAOS D2D* tinha a capacidade de descobrir servidores dentro da rede, independente de endereço IP, bem como as portas onde os demais serviços viriam a ser implantados. Porém, os dois dispositivos móveis envolvidos ainda não tinham condições de conhecer detalhes importantes um sobre o outro, como por exemplo: O aplicativo que o cliente está executando, ou se o servidor possui a dependência para executá-lo.

A fim de que as partes cliente e servidor do *CAOS D2D* possam identificar seus recursos entre si, a etapa seguinte no desenvolvimento foi embarcar o componente *Authentication Service* no protótipo, a partir de adaptações realizadas no código-fonte do *Authentication Service* do CAOS. Ao fim desse marco, o CAOS D2D Server tinha condições de conhecer detalhes sobre o *hardware* do dispositivo cliente, o aplicativo em execução nele e sua versão; e o *CAOS D2D API* tinha a condições de saber detalhes sobre o dispositivo servidor: O tipo de dispositivo, o nível de bateria, se ele está ligado a uma fonte externa de energia, e se ele possui a dependência para executar o aplicativo em execução no cliente.

Com a finalidade de incorporar no CAOS D2D as funcionalidades de *profiling* de rede do CAOS, o próximo passo da implementação foi a incorporação dos submódulos do *Profile Service* no CAOS D2D Server, e a adaptação dos submódulos do *Profile Monitor* no CAOS D2D API. Os seis submódulos adicionados foram implementados aos pares — cliente e servidor — de forma que o desenvolvimento dos próximos submódulos apenas se iniciasse após o fim do desenvolvimento do par em curso. Ao fim desse estágio de desenvolvimento, o CAOS D2D tinha agregado as funcionalidades principais relacionadas a descoberta de serviço, autenticação entre cliente e servidor, e *profiling* de rede, o que amplia a possibilidade de interoperabilidade básica entre CAOS e CAOS D2D[1].

[1] Ver Capítulo 6 (Conclusões), na página 73

Atendendo à necessidade de que o CAOS D2D fosse capaz de lidar com múltiplos servidores disponíveis em rede, o próximo passo do desenvolvimento foi uma grande refatoração nos módulos implementados até então. Como uma boa parte do código-fonte do CAOS D2D servidor foi adaptado de módulos do CAOS, a lógica de negócios desses códigos contava apenas com um servidor disponível (uma das premissas do CAOS). A refatoração afetou todos os módulos presentes no CAOS D2D API. Após o fim desse marco, o CAOS D2D estava pronto para um cenário em que mais de um servidor se apresentem na rede.

E finalmente, para tornar possível que um dispositivo servidor seja capaz de receber requisições de *offloading* de um aplicativo do qual esse dispositivo ainda não dispõe, o próximo passo foi o desenvolvimento do componente *Deploy Service*. Esse componente foi desenvolvido do princípio, como algo novo no CAOS D2D. Em sua implementação foram utilizadas as mesmas tecnologias empregadas em outros módulos: comunicação cliente/servidor através de *sockets*, e as mensagens trocadas entre os módulos são objetos serializados. Para obter o arquivo APK do aplicativo em execução, o *Deploy Client* se utiliza de um recurso da própria API do Android. Ao fim desse marco, o CAOS D2D Server tinha agregado a possibilidade de receber dependências de aplicativos, enviados pelo CAOS D2D API, caso o servidor não as possuísse.

Após os marcos de desenvolvimento descritos acima, todos os componentes do CAOS D2D estavam implementados. Posteriormente, durante uma fase de experimentações e testes, foram encontrados alguns *bugs*, que foram pontualmente corrigidos, e foram efetuados ajustes em código-fonte, para correção de comportamentos dos componentes implementados.

4.6 Considerações finais

Este capítulo apresentou o processo de implementação do protótipo do CAOS D2D; como foi iniciada a implementação do CAOS D2D API e Server, a partir de módulos existentes no CAOS, e quais foram os marcos de desenvolvimento nos quais o projeto foi dividido. Foi descrito o processo incremental no qual foram implementados os módulos, e foi apresentado o resultado final de cada marco de projeto.

5 AVALIAÇÃO DO PROTÓTIPO

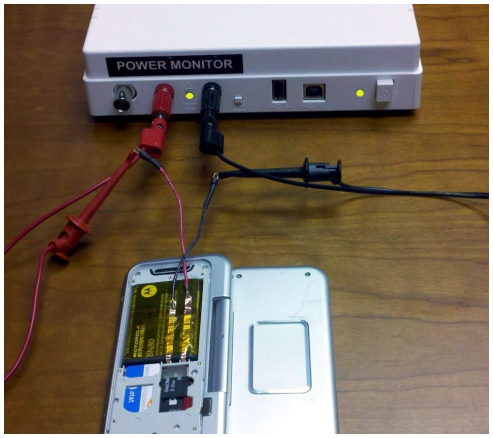
Com o objetivo de validar as funcionalidades e módulos do CAOS D2D, foram elaborados experimentos para verificar o consumo de energia dos dispositivos e os ganhos nos tempos de execução de aplicações que foram configuradas para utilizar o *framework*. Esses experimentos foram realizados com os seguintes objetivos:

- Obter resultados sobre melhorias de tempo de execução e economia de energia durante o *offloading* de processamento.
- Mensurar o consumo de energia e o tempo decorrido durante o *deploy* dinâmico de dependências.
- Aferir o tempo de execução de diferentes estratégias de *class loading*, para decidir qual estratégia é a mais apropriada para a solução.

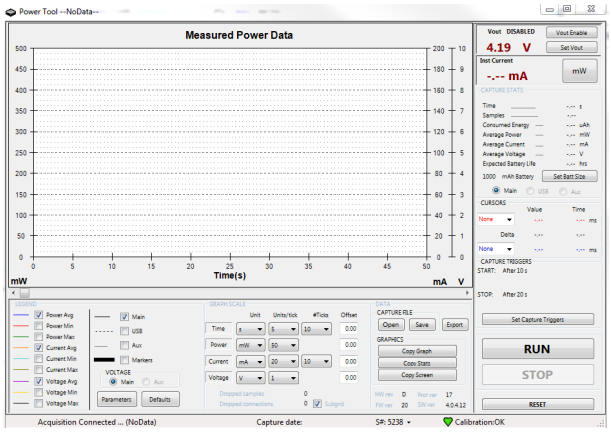
Os valores de métrica usados para medir o tempo empregado para a execução foram coletados através de artifícios da própria linguagem Java, com o auxílio de logs de aplicações fornecidos pela plataforma Android.

Figura 15 – O Monsoon Power Monitor

(a) O aparelho ligado a um *smartphone*



(b) Captura de tela do *software* de monitoramento



Fonte – Manual do Monsoon Power Monitor, disponível em http://msoon.github.io/powermonitor/LVPM.html

A métrica usada para avaliação do consumo de energia pelos dispositivos móveis foi calculada através de medições feitas no aparelho *Monsoon Power Monitor*[1], ilustrado na figura Figura 15 Esse aparelho é instalado nos contatos da bateria do dispositivo móvel, agindo entre a bateria e o dispositivo móvel. Através de um *software* fornecido pelo fabricante, o Power Monitor age como uma "bateria configurável", no qual fatores como voltagem e capacidade podem ser

[1] https://www.msoon.com

definidos pelo usuário, e também fornece dados em tempo real sobre o consumo de energia, potência, voltagem, corrente, e estimativa de tempo de duração da "bateria". O *software* de monitoramento do aparelho fornece medições sobre a potência média consumida pelo dispositivo móvel em miliwatts (mW). A partir dos dados coletados durante os experimentos, de tempo de execução dos métodos em milissegundos (ms) e potência média demandada pelo dispositivo móvel em miliwatts (mW) obtemos a energia consumida pelo dispositivo móvel em joules (J):

$$W = \frac{J}{s}$$

$$J = W * s$$

$$J = \frac{mW}{1000} * \frac{ms}{1000}$$

$$J = \frac{mW * ms}{1000000}$$

Como o Power Monitor executa suas medições no dispositivo como um todo, sem isolar aplicativos, durante os testes realizados foi tomada a medida de se finalizar os aplicativos em segundo plano nos dispositivos utilizados nos testes, deixando apenas os aplicativos diretamente envolvidos nos experimentos.

Os próximos tópicos relatam a definição, planejamento, e como foram conduzidos os experimentos, e apresentam discussões sobre os resultados obtidos.

5.1 Avaliação de *offloading*

Esta seção apresenta os experimentos realizados para avaliar os tempos de execução de tarefas de duas aplicações Android que foram configuradas para permitir o *offloading* de métodos através do CAOS D2D. O tempo de execução dos métodos escolhidos e o consumo de energia do dispositivo foram calculados em diferentes cenários.

Os dispositivos Android utilizados nessa fase foram: um *smartphone* de marca LG, modelo G3 Beat, aqui referenciado como "*Handset* A"; um *smartphone* Motorola modelo Moto

G4 Play, chamado aqui de “*Handset* B”, e um aparelho do tipo “Android TV Box”, aqui chamado de “TV Box”. A Tabela 2 resume a configuração dos equipamentos utilizados.

Tabela 2 – Descrição dos equipamentos utilizados

Dispositivo	**Configuração**
Handset A	Android 4.4.2, 1 GB de RAM, chipset Qualcomm Snapdragon 400 (CPU Cortex A7, 1.2 GHz quad-core).
Handset B	Android 6.0.1, 2 GB de RAM, chipset Qualcomm Snapdragon 410 (CPU Cortex A53, 1.2 GHz quad-core).
TV Box	Android 6.0, 1 GB RAM, chipset Amlogic S905X (CPU Cortex-A53 2GHz quad-core).

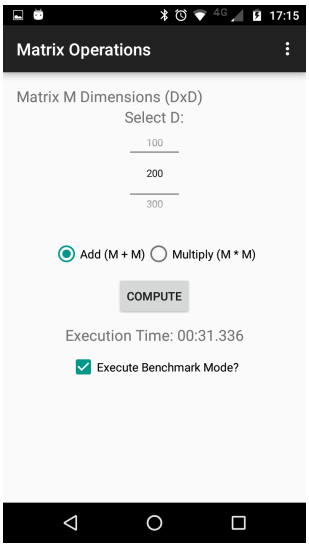
Fonte – Autoria própria

Os dispositivos foram conectados à mesma rede sem fio 802.11n, através de um ponto de acesso da marca TP-Link[1], modelo TL-WA901ND. A rede estava isolada, de forma que os dispositivos móveis Android não estavam conectados a outros dispositivos externos aos testes.

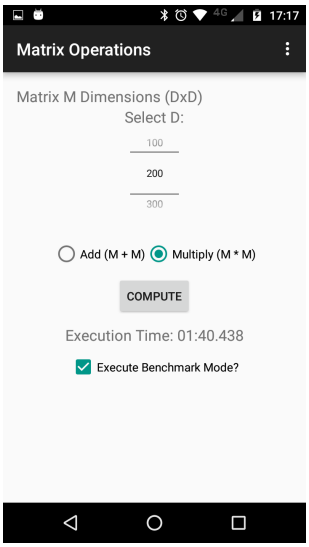
Um dos aplicativos escolhidos para os testes é o Matrix Operations, ilustrado na Figura 16. Esse aplicativo desenvolvido dentro do GREat, gera matrizes quadradas com valores aleatórios, e mede o tempo decorrido em operações de soma e multiplicação.

Figura 16 – Capturas de tela do aplicativo Matrix Operations, tirados no *Handset* B

(a) Após realizar 30 operações de adição em pares de matrizes 200x200



(b) Após realizar 30 operações de multiplicação em matrizes 200x200



Fonte – Autoria própria

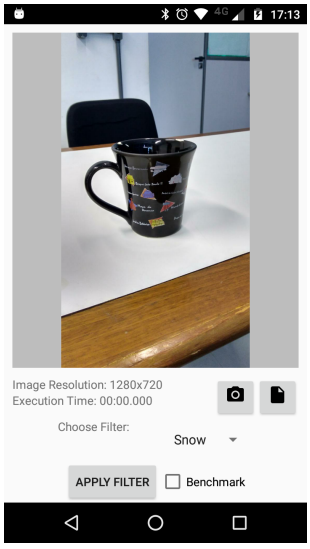
[1] http://www.tp-link.com

Esse aplicativo foi escolhido para os testes por ilustrar 2 cenários distintos quanto ao comportamento do custo computacional de métodos: A soma de matrizes é uma operação de custo computacional que cresce a uma taxa relativamente baixa – considerando-se N o tamanho da matriz quadrada, a complexidade da adição é calculada como $O(N^2)$; enquanto a multiplicação de matrizes é uma operação de custo mais alto, com complexidade calculada como $O(N^3)$. O uso desses dois métodos, com diferentes tamanhos de argumentos podem representar casos em que o *offloading* se mostre uma opção favorável, e também pode representar casos em que o custo de transferir grandes argumentos pela rede não compensem a economia de energia ou de tempo proporcionados pelo *offloading*. Durante os testes, foram feitas operações de soma e multiplicação de matrizes quadradas de tamanho 200x200 e 800x800.

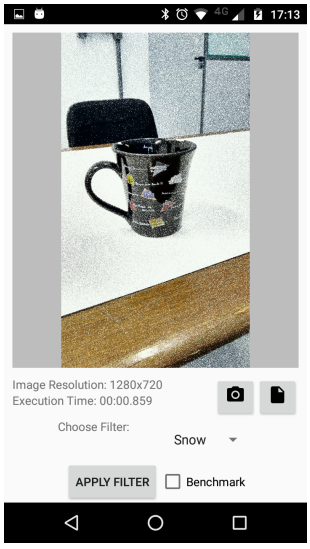
O outro aplicativo escolhido para os testes é o Camera Offloading, ilustrado na Figura 17. Um aplicativo de processamento de imagens, também desenvolvido no GREat, que permite ao usuário tirar uma foto com a câmera e aplicar diferentes efeitos.

Figura 17 – Capturas de tela do aplicativo Camera Offloading, tirados no *Handset* B

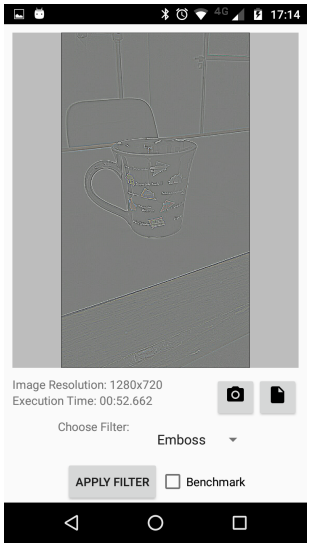
(a) Imagem sem efeito aplicado



(b) Imagem com efeito *Snow* aplicado



(c) Imagem com efeito *Emboss* aplicado



Fonte – Autoria própria

Para demonstrar que o CAOS D2D também está apto para ser usado com bibliotecas externas, foi incorporado ao Camera Offloading a biblioteca PhotoFilter[1] de processamento de imagens, e foram usados dois filtros dessa biblioteca nos testes: *Snow* e *Emboss*. Esses filtros foram escolhidos por representarem dois exemplos distintos: O filtro *Snow* é de custo baixo, com

[1] https://github.com/mukeshsolanki/photofilter

execução rápida, mesmo localmente; e o filtro *Emboss* é de custo mais elevado, devido ao fato de ele ser uma combinação de outros filtros. Durante os experimentos com o Camera Offloading foram aplicados os dois efeitos mencionados, em fotos de 1,6MP.

As operações dos aplicativos citados têm diferentes requisitos de computação e, por conseguinte, tempos de execução diferentes, e demandas de energia diferentes. Esses aplicativos foram originalmente desenvolvidos com suporte ao CAOS, e como o CAOS D2D não alterou a API original do CAOS, esses aplicativos estão automaticamente preparados para o CAOS D2D.

Para esses experimentos, foram considerados três grandes cenários de testes: no cenário C1, o *Handset* B atua como cliente interessado no *offloading*, e o *Handset* A atua como servidor. No cenário C2, o *Handset* B age como cliente, e o TV Box faz o papel de servidor. E no cenário C3, o *Handset* B executa os métodos localmente. Esse é o nosso cenário de controle, que serve de base de comparação com os outros dois cenários.

Em cada um dos três cenários definidos foram realizados os mesmos procedimentos: No aplicativo Matrix Operations, foi realizada a soma e a multiplicação de matrizes, com tamanhos 200x200 e 800x800, resultando em quatro diferentes casos. No aplicativo Camera Offloading, foram aplicados os dois efeitos escolhidos (*Snow* e *Emboss*) em imagens de 1,6MP, resultando em três casos. No total, foram realizados sete casos, repetidos trinta vezes, em três diferentes cenários; o que totaliza 630 execuções de métodos avaliadas. Durante os três experimentos, houve uma pausa programada de 1 segundo entre todas as execuções de métodos, e o consumo de energia do *Handset* B foi medido com o *Power Monitor*. As medições foram feitas para, além de termos métricas sobre tempos de execução de métodos, medir o consumo de energia do *Handset* B quando ele atua como cliente, e quando executa os processamentos localmente. Os experimentos realizados para a avaliação do *offloading* estão resumidos na Tabela 3

Os dados coletados durante a realização desses experimentos estão dispostos no Apêndice A. A seguir, são analisados os dados relacionados aos tempos de execução de métodos e consumo de energia por parte do *Handset* B, nos três casos possíveis no que diz respeito a *offloading*: quando ele atua como cliente, servidor, e quando ele executa as tarefas localmente.

5.1.1 Avaliação de tempo de execução

Os dados das médias de medições de tempo de execução dos métodos dos casos analisados estão organizados na Figura 18. A partir desses dados, podemos chegar a algumas

Tabela 3 – Sumário dos experimentos

Fatores	Método, tamanho do argumento, e onde executar o método.
Níveis	2 métodos para o Camera Offloading (*Snow* e *Emboss*), 2 métodos para o Matrix Operations (*add* e *multiply*), 1 tamanho de argumento para o Camera Offloading (1,6MP), 2 tamanhos de argumentos para o Matrix Operations (200x200 e 800x800), e 3 locais para executar os métodos (*Handset* A, TV Box e localmente).
Projeto	O experimento foi repetido 30 vezes para as combinações de níveis dos fatores, considerando-se o *Handset* B como cliente, pois as medições foram feitas de acordo com o ponto de vista do *Handset* B.
Resposta	Tempo decorrido para a execução dos métodos e o consumo de energia por parte do dispositivo cliente.

Fonte – Autoria própria

conclusões.

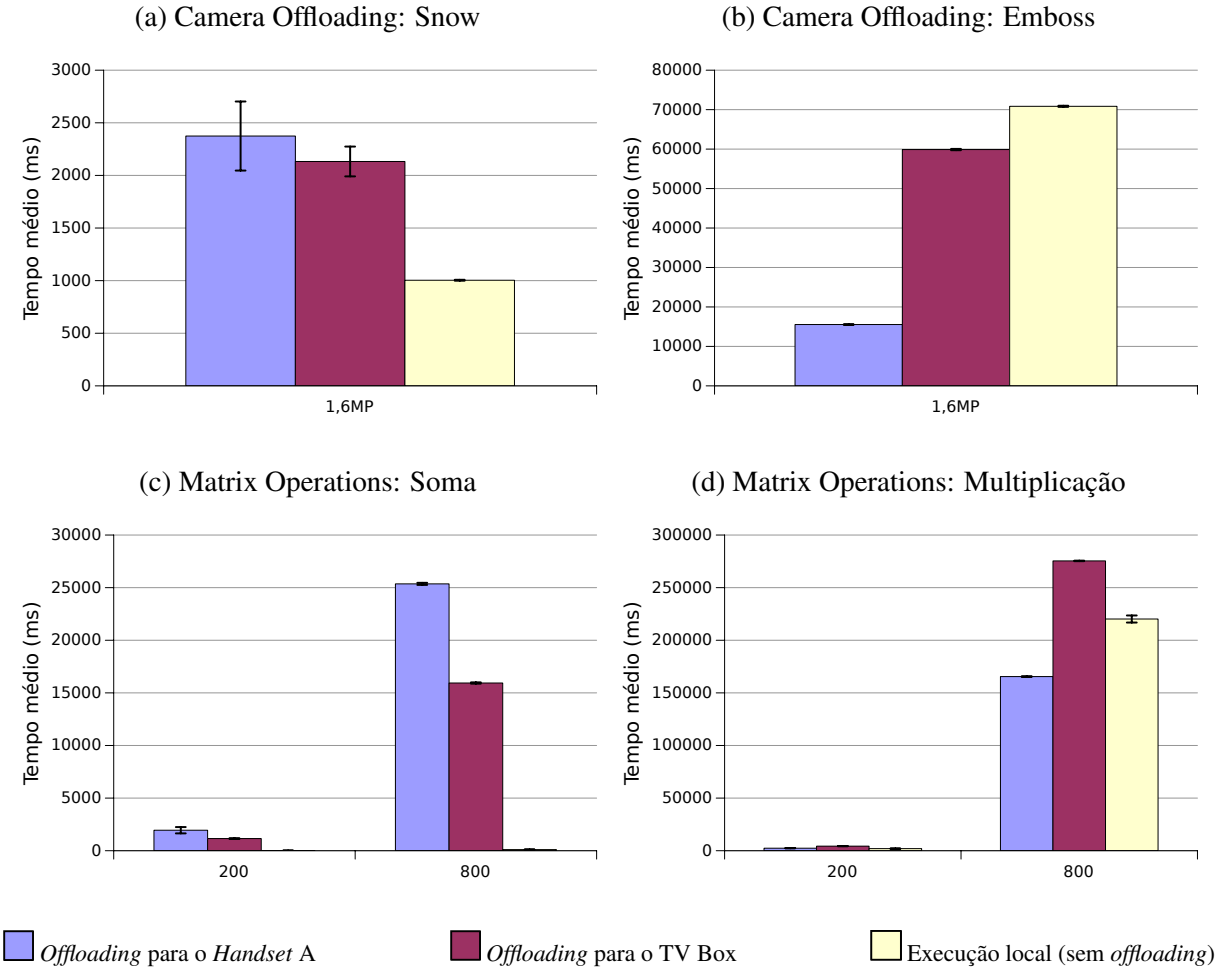
No momento que o *framework* decide pelo *offloading* de um método, ele encapsula o método e seus argumentos em um objeto a ser transmitido pela rede[1]. Justamente pelo fato de a transmissão desse objeto também ter o seu custo de tempo, notamos que operações de baixo custo computacional não se beneficiaram com o *offloading* em termos de tempo de execução. No caso do aplicativo Matrix Operations, a execução local da soma de matrizes de tamanho 800, por exemplo, foi 99,6% mais rápida do que quando foi executada no *Handset* A, e 99,3% mais rápida do que quando foi executada no TV Box. Isso se deve ao fato de o objeto transmitido pelo CAOS D2D para realização do *offloading* ser em média, nesse caso em especial, 5,12MB; e o objeto representando o resultado dessa soma ter tamanho médio de 2,6MB. O tempo necessário para transmissão desses dois objetos fez grande diferença frente ao tempo empregado na operação de soma propriamente dita, que é de custo baixo.

De maneira análoga, no caso do aplicativo Camera Offloading, percebemos que o *Handset* B teve melhores tempos de execução quando aplicou o efeito menos custoso – *Snow* – localmente. A execução local do efeito *Snow* foi em média 57,7% mais rápida do que no *Handset* A, e 52,9% mais rápida do que no TV Box. O ganho de tempo proporcionado pelo *offloading* não compensou o custo de enviar e receber dados pela rede *Wi-Fi* para a realização de tarefas de baixo custo.

Em termos de tempos de execução, o *offloading* se mostrou uma opção mais proveitosa para casos que demandam maior poder computacional. Para esses casos, o custo de execução maior apresentam um peso mais significativo no processo de *offloading* do que o custo de transmissão dos argumentos através da rede. No aplicativo Camera Offloading, A aplicação

[1] Ver seção 4.2 (Arquitetura do CAOS D2D), na página 41

Figura 18 – Tempos médios de operações avaliadas nos testes de *offloading*



Intervalo de confiança de 95%. Cada gráfico representa um caso de aplicativo e método, ilustrando os três cenários avaliados, com variações no tamanho dos argumentos a serem processados (no caso do aplicativo Matrix Operations).
Fonte – Autoria própria

do efeito *Emboss* com *offloading* para o *Handset* A foi 78,1% mais rápida, e para o TV Box foi 15,5% mais rápida, ambos os casos em comprarção com a a execução local.

Uma observação interessante desse experimento é o caso da multiplicação de matrizes, no aplicativo Matrix Operations. Apesar de a soma de matrizes ter apresentado tempos mais baixos quando o *offloading* era realizado no TV Box do que no *Handset* A, a multiplicação de matrizes de tamanho 800 no TV Box foi em média 25,1% mais lenta do que quando foi executada localmente; bem diferente da multiplicação no *Handset* A, que foi 24,8% mais rápida do que na execução local. Essa tendência continua também com a multiplicação de matrizes de tamanho 200x200.

Essa divergência também pode ser observada no aplicativo Camera Offloading: a aplicação do efeito *Snow* em *offloading* para o TV Box foi mais rápida do que para o *Handset* A, porém a aplicação do efeito *Emboss* para o *Handset* A foi mais rápida para o *Handset* A do que

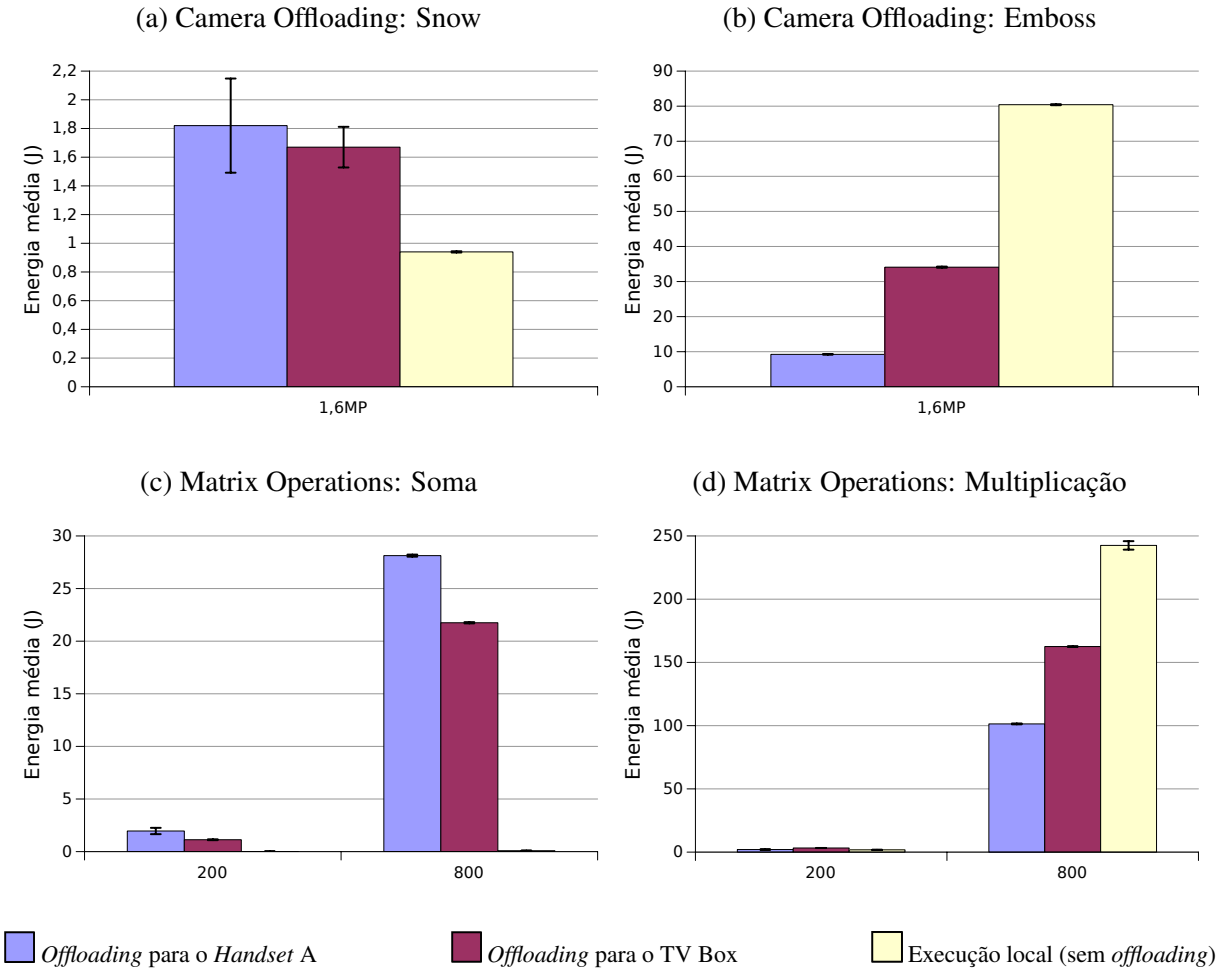
para o TV Box.

Esses casos apontam um indício de que um dispositivo que executa uma determinada tarefa mais rápido do que os outros não necessariamente será sempre a melhor opção para o *offloading*.

5.1.2 Avaliação de consumo de energia

Os dados calculados sobre o consumo de energia nos experimentos estão organizados na Figura 19. Partindo da análise desses dados, podemos observar que o consumo de energia por parte do *Handset* B foi em geral proporcional ao tempo de execução das tarefas.

Figura 19 – Cálculos da energia média consumida em operações avaliadas nos testes de *offloading*



Intervalo de confiança de 95%. Cada gráfico representa um caso de aplicativo e método, ilustrando os três cenários avaliados, com variações no tamanho dos argumentos a serem processados (no caso do aplicativo Matrix Operations).
Fonte – Autoria própria

Nos casos em que o *offloading* proporcionou melhoria no tempo de execução de tarefas, ele também proporcionou melhoria no consumo de energia. No aplicativo Camera

Offloading, a aplicação do efeito *Emboss* apresentou economia de 88,5% quando feito em *offloading* para o *Handset* A, e economia de 57,6% quando realizado em *offloading* para o TV Box. No caso do aplicativo Matrix Operations, a multiplicação de matrizes de tamanho 800, por exemplo, se mostrou 58,2% mais econômica em termos de consumo de energia quando realizada em *offloading* para o *Handset* A. E, diferente do comportamento observado quanto ao aspecto de tempos de execução, a mesma tarefa também se mostrou vantajosa quando realizada em *offloading* para o TV Box. Nesse caso a economia foi em média de 33%. Essa divergência entre o tempo decorrido e a energia consumida se deve ao fato de que em boa parte do tempo empregado pelo *Handset* B durante a operação de *offloading* o dispositivo móvel se encontra "aguardando" o resultado do processamento, em vez de efetivamente realizando a tarefa, e portanto, poupando energia. Por esse motivo podemos considerar que, no aspecto de melhora no consumo de energia, o *offloading* é mais proveitoso à medida que a tarefa é mais custosa.

Por outro lado, nos casos em que a tarefa que é feita em *offloading* é de custo computacional baixo, percebemos que a execução local das tarefas representa uma opção melhor em termos de consumo de energia. No aplicativo Camera Offloading, a aplicação do efeito *Snow* em *offloading* para o *Handset* A apresentou em média consumo 93,7% maior do que na execução local, e consumo 77,1% maior quando realizada em *offloading* para o TV Box. Para o Matrix Operations, a soma de matrizes de tamanho 800, quando executado localmente, apresentou consumo 99,7% menor do que quando em *offloading* para o *Handset* A, e 99,6% menor do que realizando em *offloading* para o TV Box.

5.2 Avaliação de *deploy* de dependências

Esta seção apresenta os experimentos realizados com o fim de avaliar o consumo energético e o tempo gasto pelo módulo de *deploy* de dependências do CAOS D2D. Durante esses experimentos, foram utilizados os mesmos dispositivos *Handset* A e *Handset* B, utilizados para os experimentos de avaliação de *offloading*[1]. Esses dispositivos foram conectados à uma rede sem fio 802.11n. Porém, em vez de um ponto de acesso sem fio convencional, para esses experimentos um computador foi configurado para atuar como um ponto de acesso. Esse computador foi usado para que se pudesse ter controle sobre a velocidade máxima da rede ao qual os dois dispositivos móveis estavam conectados.

Para a realização desses experimentos, foram desenvolvidos dois aplicativos configu-

[1] Ver seção 5.1 (Avaliação de *offloading*), na página 59

rados com o CAOS D2D, a partir de variações do aplicativo Camera Offloading, mencionado anteriormente. Um deles – chamado aqui de "*App* A"– tem no seu arquivo APK 1.830.413B, ou 1,83MB (ou 1,74MiB). O outro aplicativo – referenciado aqui como "*App* B" – tem em seu APK 54.251.284B, ou 54,25MB.

Os experimentos para avaliação do módulo de *deploy* de dependências seguiram o seguinte roteiro: Para cada um dos dois aplicativos – *App* A e *App* B – foram realizadas operações de *deploy* de dependências do *Handset* B para o *Handset* A, 30 vezes com a velocidade máxima da rede configurada para 8mbps, e mais 30 vezes com a velocidade de rede em 4mbps. Foi adicionado também um intervalo de 1 segundo entre as execuções individuais. Durante os experimentos, foram coletados dados sobre o tempo decorrido para as operações de *deploy* de dependências através de *logs* extraídos do *Handset* B, e foram coletados dados sobre o consumo de energia do *Handset* B, através do *Power Monitor*. A Tabela 4 resume os experimentos realizados.

Tabela 4 – Sumário dos experimentos para avaliação de *deploy* de dependências

Fatores	Tamanho do arquivo APK, e velocidade de rede.
Níveis	2 diferentes tamanhos de arquivo (1,8MB e 54,3MB), e 2 diferentes velocidades de rede (4Mbps e 8Mbps).
Projeto	O experimento foi repetido 30 vezes para as combinações de níveis dos fatores, considerando-se o *Handset* B como cliente, pois as medições foram feitas de acordo com o ponto de vista do *Handset* B.
Resposta	Tempo decorrido para a execução dos métodos e o consumo de energia por parte do dispositivo cliente.

Fonte – Autoria própria

Para realizar o *deploy* de dependências no dispositivo servidor, o CAOS D2D envia o próprio arquivo APK do aplicativo no qual está sendo usado, em vez de um arquivo dentro do diretório de recursos do aplicativo[1]. Por isso, espera-se que o tempo necessário para *deploy* de dependências dependa, além da velocidade da rede na qual o dispositivo móvel cliente está, também do tamanho do arquivo APK do aplicativo que está usando o CAOS D2D. No caso do *App* A e do *App* B, os tempos ideais de transmissão podem ser calculados da seguinte forma:

$$\frac{1.830.413B}{8mbps} = 1,83s \tag{5.1}$$

[1] Ver seção 4.2 (Arquitetura do CAOS D2D), na página 41

$$\frac{54.251.284B}{8mbps} = 54,25s \tag{5.2}$$

$$\frac{1.830.413B}{4mbps} = 3,66s \tag{5.3}$$

$$\frac{54.251.284B}{4mbps} = 108,5s \tag{5.4}$$

Os resultados obtidos nos experimentos descritos estão organizados na Tabela 5. Partindo da observação desses dados, podemos perceber que, primeiramente, os tempos decorridos para transmitir os arquivos APK pela rede são próximos dos tempos ideais demonstrados nas equações 5.1 a 5.4; as perdas médias se situaram entre 5,3% e 10,2% acima do tempo ideal, sendo que essas diferenças se tornaram menos significativas à medida que o tamanho do arquivo APK transmitido era maior.

Tabela 5 – Dados coletados durante os experimentos para avaliação do módulo de *deploy* de dependências

Velocidade de Rede (mbps)	Tamanho do APK (B)	Potência Média (mW)	Tempo Ideal (ms)	Média tempo (ms)	IC Tempo (95%)	Diferença Tempo Ideal (%)	Média Energia (J)	IC Energia (95%)
4	1830413	677.79	3660.83	3943.77	58.17	7.17	2.67	0.04
	54251284	667.44	108502.57	114538.63	373.67	5.27	76.45	0.25
8	1830413	671.81	1830.41	2039.03	53.71	10.23	1.37	0.04
	54251284	739.43	54251.28	57530.03	168.14	5.70	42.54	0.12

Fonte – Autoria própria

Tabela 6 – Análise de variância do consumo de energia durante o *deploy* de dependências

	Grau de Lib.	Soma Quad.	Quad. Médio	Estatística F	Valor P
Tamanho APK	1	$9.91x10^{10}$	$9.91x10^{10}$	$1.94x10^{32}$	$<2x10^{-16}$
Tempo (s)	1	$1.73x10^{10}$	$1.73x10^{10}$	$3.38x10^{31}$	$<2x10^{-16}$
Largura banda	1	$4.13x10^{5}$	$4.13x10^{5}$	$8.07x10^{26}$	$<2x10^{-16}$
Resíduos	112	0.00	0.00		

$\alpha = 0.05$
Fonte – Autoria própria

Um detalhe que chama a atenção é o fato de a potência elétrica média demandada pelo *Handset* B durante o processo de *deploy* de dependência se manter próximo em 3 dos 4

casos analisados. Isso sugere que, na princípio, o consumo energético do dispositivo móvel durante o *deploy* de dependências depende do tempo decorrido no processo. A exceção está no caso em que o arquivo a ser transmitido pela rede é grande, e a velocidade de rede é alta. Isso nos sugere que a transmissão continuada em alta velocidade aumenta o consumo médio de energia por parte do dispositivo móvel. Observando a análise de variância na Tabela 6 percebemos que os fatores tamanho do arquivo a ser transmitido, a velocidade de rede tempo decorrido para transmissão têm impacto na energia consumida pelo dispositivo móvel cliente. A diferença na potência média aferida para o quarto caso analisado (*App* B a 8mbps) pode ser explicado pela transmissão de rede realizada em alta velocidade por um tempo mais longo; o que eleva o consumo médio de energia (BARBERA *et al.*, 2013).

5.3 Avaliação do carregamento de dependências

Essa seção apresenta experimentos que foram realizados com o CAOS D2D, a fim de mensurar o peso que fatores como a complexidade de métodos, o tamanho das dependências a serem carregadas pelo servidor, ou a escolha de manter ou não essas dependências em cache exercem no tempo de execução em operações.

Para a realização desses experimentos, foram utilizados os mesmos aplicativos utilizados no experimento da seção 5.2: *App* A e *App* B, que são duas variações do aplicativo Camera Offloading. Porém, para efeito de avaliação, nesse experimento o código-fonte do CAOS D2D sofreu alterações de forma a implementar duas diferentes estratégias de carregamento de dependências. Em uma delas, o objeto *ClassLoader* com o arquivo APK referente à dependência demandada pelo dispositivo cliente é pré-instanciada pelo dispositivo servidor, e mantido em uma estrutura de dados em memória, de prontidão para ser utilizado no momento do *offloading*. A outra estratégia consiste em instanciar o objeto *ClassLoader* apenas no momento em que ele é necessário, ou seja, no momento do *offloading*, e em seguida, descartado.

Nessa bateria de experimentos foram utilizados os mesmos dispositivos móveis *Handset* A e *Handset* B, utilizados nos experimentos anteriores. Para os dois aplicativos, foram efetuadas medições de tempo de execução dos filtros *Snow* e *Emboss*, com variações na abordagem de carregamento de dependências. Cada caso foi executado 30 vezes, empregando-se o *Handset* A como servidor e o *Handset* B como cliente. A Tabela 7 resume os experimentos realizados.

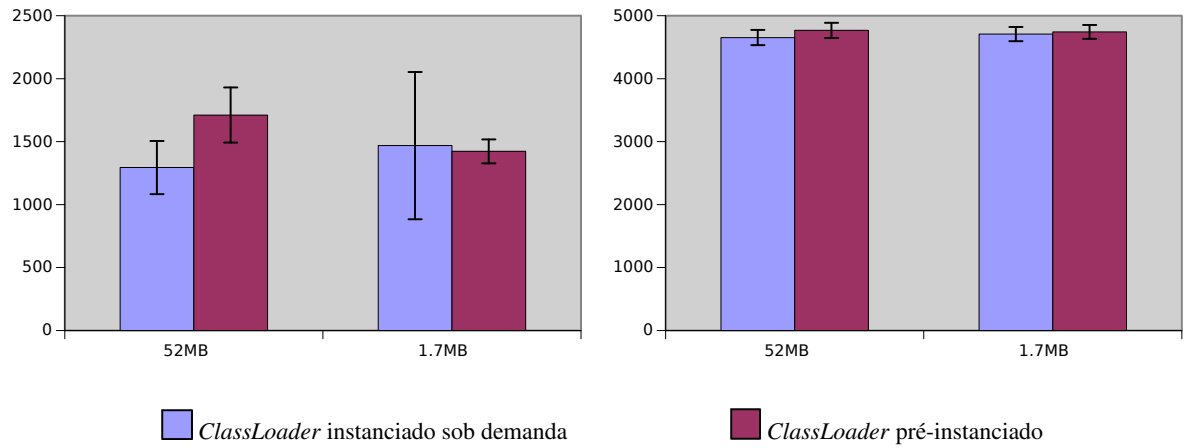
As médias de tempos de execução dos métodos estão organizados na Figura 20.

Tabela 7 – Sumário dos experimentos para avaliação de carregamento de dependências

Fatores	Tamanho do arquivo APK, método a ser executado e estratégia de carregamento.
Níveis	2 diferentes tamanhos de arquivo (1,8MB e 54,3MB), e 2 métodos (snow e emboss) e manter ou não o *ClassLoader* da dependência em *cache*.
Projeto	O experimento foi repetido 30 vezes para as combinações de níveis dos fatores, considerando-se o *Handset* B como cliente, pois as medições foram feitas de acordo com o ponto de vista do *Handset* B.
Resposta	Tempo decorrido para a execução dos métodos.

Fonte – Autoria própria

Figura 20 – Tempos médios de operações realizadas nos testes de carregamento de dependências



Intervalo de confiança de 95%. Cada gráfico representa um caso de aplicativo e método, com variações na estratégia de carregamento de dependências

Fonte – Autoria própria

A partir desses dados, podemos perceber que via de regra, a estratégia de instanciar o objeto *ClassLoader* apenas no momento da execução dos métodos em *offloading* teve resultados médios melhores do que manter esse objeto pré-instanciado, armazenado em uma estrutura de dados em memória. Isso sugere que a instanciação de um novo objeto *ClassLoader* foi realizada mais rápido do que a busca de um *ClassLoader* já instanciado e armazenado em uma estrutura de dados. Percebemos também que no caso do filtro *Emboss*, a diferença nas médias de tempo de execução foi pouco significativa.

Tabela 8 – Análise de valor P de tempos de aplicação dos filtros

Cenário	Valor P
APK de 52MB, filtro Snow	0.1837
APK de 52MB, filtro Emboss	0.5008
APK de 1.7MB, filtro Snow	0.0226
APK de 1.7MB, filtro Emboss	0.7660

$\alpha = 0.05$

Fonte – Autoria própria

O comportamento na média dos tempos não se repetiu no caso do filtro *Snow*, com a versão do aplicativo de 1,7MB. Porém, o alto intervalo de confiança sugere uma margem de erro elevada. Realizando um cálculo de valor-P a partir das duas amostras de cada caso (*ClassLoader* com e sem cache), chegamos aos valores na Tabela 8. Podemos testar a hipótese nula de que a diferença média entre essas duas amostras é igual a zero, no caso do efeito *Snow* para o APK de 1,7MB. Nesse caso, chegamos a um valor P de 2,25%; abaixo do α adotado de 5%, o que nos leva a rejeitar a hipótese nula para esse caso.

5.4 Considerações finais

Este capítulo descreveu e discutiu três baterias de experimentos que foram realizados com o *framework* CAOS D2D, que foram concebidos com o objetivo avaliar diferentes aspectos da solução.

No que diz respeito ao tempo de execução de tarefas em *offloading*, os resultados demonstrados neste capítulo apontam que existem casos em que o *offloading* representa uma opção muito vantajosa em termos de ganhos de tempo — com ganhos médios de até 78,1% com relação à execução local — mas também existem casos em que a execução local é a melhor opção. Essa variação depende da complexidade da tarefa a ser delegada, e do tamanho de seus argumentos. Houve casos em que o *offloading* para um dispositivo representava a opção com melhores tempos, mas também houve casos em que o melhor tempo de execução era verificado com o *offloading* para outro dispositivo.

Quanto ao aspecto da economia de energia proporcionada pelo *offloading*, os resultados apresentados aqui demonstram casos em que o *offloading* permitiu economia de até 88,5%; mas também casos em que a execução local foi a melhor opção, devido ao consumo adicional de energia necessário para transferir o objeto que encapsula método e argumento para o dispositivo servidor. Alguns resultados dos experimentos demonstram um caso em que o *offloading* para um dispositivo representava aumento no tempo de execução, porém com melhoria de consumo de energia.

Através dos experimentos percebemos que o tempo decorrido na injeção de dependências é geralmente próximo do tempo ideal, e que o consumo energético do dispositivo móvel durante a injeção de dependências cresce em função do tempo empregado na transmissão, o que é consequência de tamanho do arquivo APK da aplicação e da velocidade da rede sem fio.

Os resultados apresentados no capítulo demonstram também que na maioria dos

casos analisados, a estratégia de instanciar o *ClassLoader* com as dependências da aplicação cliente no momento da execução dos métodos em *offloading* oferece melhor tempo de execução, em comparação à abordagem de se manter um *ClassLoader* à disposição em uma estrutura de dados em memória.

Adicionalmente, Apesar do ambiente controlado onde foram realizados os testes, o estudo feito em (FUNAI *et al.*, 2016) mostra que, por mais que o cenário ideal para o *offloading* seja dentro de uma rede com rotas de apenas um salto, realizar o *offloading* para um dispositivo que esteja a alguns saltos de rede de distância ainda pode ser vantajoso, o que endossa os benefícios do *offloading* entre dispositivos móveis em situações do mundo real.

6 CONCLUSÕES

Neste dissertação foi apresentado o CAOS D2D, um *framework* para dar suporte ao *offloading* entre dispositivos móveis, sem a necessidade de uma estrutura de nuvem ou *cloudlet*. Foram descritas a concepção e a implementação do *framework*, a partir de uma outra solução existente, desenvolvida dentro do nosso grupo de pesquisa.

O trabalho apresenta contribuições para assuntos pertinentes ao desenvolvimento de *middlewares* e *frameworks* com finalidades similares, detalhando o funcionamento de componentes e limitações da abordagem e possibilidades de trabalhos futuros (adiante), o que é de contribuição para futuros pesquisadores que pretendam desenvolver soluções similares, ou versões ampliadas do próprio CAOS D2D.

Essa dissertação também apresenta experimentos que foram realizados para mensurar o impacto nos custos de tempo e consumo de energia proporcionados pelo CAOS D2D, em diferentes casos de uso de diferentes aplicativos, comparando os custos entre os cenários em que o dispositivo móvel realiza o *offloading* para diferentes dispositivos com os cenários em que ele executava os processamentos localmente.

Através dos resultados dos testes, foi verificado que normalmente o *offloading* representa uma opção vantajosa para melhorar o desempenho e economizar energia, seja quando a tarefa de processamento é custosa, ou quando a quantidade de dados a serem processados é grande. Mas em casos em que o processamento dos dados é computacionalmente barato, e a quantidade de dados é pequena, é menos custoso em termos de tempo e energia realizar o processamento localmente.

A seguir serão apresentadas a produção bibliográfica feita durante o tempo de curso desse mestrado, algumas limitações conhecidas da solução, e propostas de trabalhos futuros.

6.1 Produção bibliográfica

Durante o tempo de curso do mestrado foram publicados dois trabalhos em âmbito nacional, ambos relacionados com a proposta dessa dissertação e publicados no Simpósio Brasileiro de Sistemas Multimídia e Web (*WebMedia*):

- SANTOS, G.; TRINTA, F.; REGO, P.; GOMES, F. Caos d2d: Uma solução para offloading de métodos entre dispositivos móveis. In: **XVI WFA**. [S.l.: s.n.], 2017. p. 164–168.
- SANTOS, G.; REGO, P.; TRINTA, F. Uma proposta de solução para offloading de métodos

entre dispositivos móveis. In: **XVII WTD - WebMedia 2017**. [S.l.: s.n.], 2017. p. 76–81.

A primeira publicação, de Qualis B5, apresenta a ferramenta CAOS D2D, como uma possibilidade prática para o *offloading* no *Workshop* de Ferramentas e Aplicações (WFA). A segunda publicação apresenta uma proposta para esse projeto de pesquisa no *Workshop* de Teses e Dissertações (WTD).

Futuramente pretende-se publicar artigos explorando aspectos como a arquitetura e o desempenho do CAOS D2D em outros eventos de âmbito nacional, como o Simpósio Brasileiro de Redes de Computadores (SBRC).

6.2 Limitações

Uma limitação conhecida na arquitetura do CAOS D2D é que a atual implementação do protótipo não conta com módulos de tomada de decisão capazes de verificar se é interessante realizar o *offloading*, e para qual dispositivo realizar esse *offloading*, além de distinguir as diferentes situações verificadas durante os testes. Essa limitação, no entanto, foi prevista pois, apesar da extensibilidade do CAOS D2D para a implementação de tais mecanismos, esta foge ao escopo dessa pesquisa. Outra limitação de arquitetura do CAOS D2D é o pré-requisito de que métodos em *offloading* precisam necessariamente de um valor de retorno, o que pode exigir alterações em assinaturas de classes dos aplicativos, apesar da improbabilidade da necessidade de se realizar o *offloading* de um método que não retorne resultado.

Outra limitação da implementação do CAOS D2D é a necessidade de que a comunicação entre os dispositivos móveis seja via redes padrão 802.11, o que exige que haja um ponto de acesso *Wi-Fi* para a comunicação entre os dispositivos móveis envolvidos no processo de *offloading*, impossibilitando a comunicação ad hoc entre eles.

Uma característica do CAOS D2D que pode ser considerada uma limitação é o fato de um cliente CAOS D2D não poder se comunicar com um servidor CAOS, nem vice-versa. Isso pode ser considerado uma limitação de arquitetura do CAOS D2D, quando se considera que o CAOS D2D é uma versão complementar do CAOS.

6.3 Trabalhos futuros

Levando-se em consideração as limitações expostas, pretende-se em trabalhos futuros usar da estrutura que o CAOS D2D já disponibiliza para a extensão desta, através da implantação

de módulos de tomada de decisão capazes de avaliar se é vantajoso realizar o *offloading* em comparação com a execução local de métodos, e em qual dispositivo executar o *offloading*, dado que mais de um servidor podem estar disponíveis.

Outra limitação a ser trabalhada futuramente é a conectividade entre dispositivos, atualmente suportando apenas redes padrão 802.11. A introdução de padrões como *Wi-Fi Direct* ou *Bluetooth* pode aumentar as possibilidades de casos que se beneficiem do *offloading*.

Apesar de realizarem uma tarefa fundamental para garantir o desempenho das operações de *offloading*, otimizações podem ser feitas aos módulos responsáveis pelo *profiling* de rede, pois apesar de necessário para o monitoramento de conectividade, o envio de pacotes de rede aumentam o consumo de bateria pelo dispositivo móvel cliente. Essas otimizações podem tornar esse consumo de bateria o mínimo possível.

Um trabalho futuro a ser considerado é a avaliação de diferentes estratégias de injeção de dependência, como por exemplo, o uso de um repositório central de dependências, em vez da injeção direta a partir do dispositivo móvel cliente. Testes e características de ambas as abordagens podem ser analisados de forma a verificar se uma estratégia é melhor do que a outra; ou ainda, se uma abordagem híbrida é uma boa opção.

Um outro trabalho futuro que pode ser realizado é a avaliação do *framework* do CAOS D2D do ponto de vista de programadores, em aspectos com adaptabilidade ou nível de conforto dos desenvolvedores com as alterações de código.

Adicionalmente, pretende-se tornar o CAOS e o CAOS D2D interoperáveis entre si, ou seja, convergir as duas soluções de forma que um cliente CAOS D2D possa usufruir de um *cloudlet* CAOS ou de um dispositivo móvel com CAOS D2D, dependendo da disponibilidade de um ou outro dentro da rede. Essas melhorias apresentam potencial para expansão das possibilidades do CAOS D2D, mas podem demandar uma alteração na arquitetura deste.

REFERÊNCIAS

ABOLFAZLI, S.; SANAEI, Z.; SANAEI, M. H.; SHOJAFAR, M.; GANI, A. Mobile cloud computing. **Encyclopedia of Cloud Computing**, John Wiley & Sons, p. 29, 2016.

AHMED, E.; GANI, A.; Khurram Khan, M.; BUYYA, R.; KHAN, S. U. Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges. **Journal of Network and Computer Applications**, Elsevier, v. 52, p. 154–172, 2015. ISSN 10958592. Disponível em: <http://dx.doi.org/10.1016/j.jnca.2015.03.001>.

AHUJA, S. P.; ZAMBRANO, J. Mobile cloud computing: Offloading mobile processing to the cloud. **Computer and Information Science**, v. 9, n. 1, p. 90, 2016.

ANGIN, P.; BHARGAVA, B. K. An agent-based optimization framework for mobile-cloud computing. **JoWUA**, v. 4, n. 2, p. 1–17, 2013.

ARTAIL, A.; FRENN, K.; SAFA, H.; ARTAIL, H. A framework of mobile cloudlet centers based on the use of mobile devices as cloudlets. In: IEEE. **Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference on**. [S.l.], 2015. p. 777–784.

ASRANI, P. Mobile cloud computing. **International Journal of Engineering and Advanced Technology (IJEAT)**, Citeseer, v. 2, n. 4, p. 606–609, 2013.

BALAN, R.; FLINN, J.; SATYANARAYANAN, M.; SINNAMOHIDEEN, S.; YANG, H.-I. The case for cyber foraging. In: ACM. **Proceedings of the 10th workshop on ACM SIGOPS European workshop**. [S.l.], 2002. p. 87–92.

BARBERA, M. V.; KOSTA, S.; MEI, A.; STEFA, J. To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In: IEEE. **INFOCOM, 2013 Proceedings IEEE**. [S.l.], 2013. p. 1285–1293.

CHAFFEY, D. **Mobile Marketing Statistics Compilation**. 2016. Http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/. Acessado em 09/05/2016.

CHESS, D.; HARRISON, C.; KERSHENBAUM, A. Mobile agents: Are they a good idea? In: **Mobile Object Systems Towards the Programmable Internet**. [S.l.]: Springer, 1997. p. 25–45.

CHUN, B.-G.; IHM, S.; MANIATIS, P.; NAIK, M.; PATTI, A. Clonecloud: Elastic execution between mobile device and cloud. In: **Proceedings of the Sixth Conference on Computer Systems**. New York, NY, USA: ACM, 2011. (EuroSys '11), p. 301–314. ISBN 978-1-4503-0634-8. Disponível em: <http://doi.acm.org/10.1145/1966445.1966473>.

COSTA, P. B. **Uma Abordagem para Offloading em Múltiplas Plataformas Móveis**. Dissertação (Mestrado) — Universidade Federal do Ceará, Fortaleza, 2014.

CUERVO, E.; BALASUBRAMANIAN, A.; CHO, D.-k.; WOLMAN, A.; SAROIU, S.; CHANDRA, R.; BAHL, P. Maui: Making smartphones last longer with code offload. In: **MobiSys 2010, Proceedings ACM**. New York, NY, USA: ACM, 2010. p. 49–62. ISBN 978-1-60558-985-5. Disponível em: <http://doi.acm.org/10.1145/1814433.1814441>.

DINH, H. T.; LEE, C.; NIYATO, D.; WANG, P. A survey of mobile cloud computing: architecture, applications, and approaches. **Wireless Communications and Mobile Computing**, v. 13, n. 18, p. 1587–1611, 2013. ISSN 1530-8677. Disponível em: <http://dx.doi.org/10.1002/wcm.1203>.

DUARTE, P. A.; SILVA, L. F. M.; GOMES, F. A.; VIANA, W.; TRINTA, F. M. Dynamic deployment for context-aware multimedia environments. In: **Proceedings of the 21st Brazilian Symposium on Multimedia and the Web**. New York, NY, USA: ACM, 2015. (WebMedia '15), p. 197–204. ISBN 978-1-4503-3959-9. Disponível em: <http://doi.acm.org/10.1145/2820426.2820443>.

FERNANDO, N.; LOKE, S. W.; RAHAYU, W. Mobile cloud computing. **Future Gener. Comput. Syst.**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 29, n. 1, p. 84–106, jan. 2013. ISSN 0167-739X. Disponível em: <http://dx.doi.org/10.1016/j.future.2012.05.023>.

FERRARI, A.; GIORDANO, S.; PUCCINELLI, D. Reducing your local footprint with anyrun computing. **Computer Communications**, Elsevier, v. 81, p. 1–11, 2016.

FUNAI, C.; TAPPARELLO, C.; HEINZELMAN, W. Mobile to mobile computational offloading in multi-hop cooperative networks. In: IEEE. **Global Communications Conference (GLOBECOM), 2016 IEEE**. [S.l.], 2016. p. 1–7.

GAMMA, E. **Design patterns: elements of reusable object-oriented software**. [S.l.]: Pearson Education India, 1995.

GOMES, F. A.; VIANA, W.; ROCHA, L. S.; TRINTA, F. A contextual data offloading service with privacy support. In: **Proceedings of the 22Nd Brazilian Symposium on Multimedia and the Web**. New York, NY, USA: ACM, 2016. (Webmedia '16), p. 23–30. ISBN 978-1-4503-4512-5. Disponível em: <http://doi.acm.org/10.1145/2976796.2976860>.

GOMES, F. A. A.; REGO, P. A. L.; ROCHA, L.; SOUZA, J. N. de; TRINTA, F. Caos: A context acquisition and offloading system. **COMPSAC**, 2017.

GORDON, M. S.; JAMSHIDI, D. A.; MAHLKE, S.; MAO, Z. M.; CHEN, X. Comet: Code offload by migrating execution transparently. In: **Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation**. Berkeley, CA, USA: USENIX Association, 2012. (OSDI'12), p. 93–106. ISBN 978-1-931971-96-6. Disponível em: <http://dl.acm.org/citation.cfm?id=2387880.2387890>.

HUERTA-CANEPA, G.; LEE, D. A virtual cloud computing provider for mobile devices. In: ACM. **Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond**. [S.l.], 2010. p. 6.

KHAN, M. A. A survey of computation offloading strategies for performance improvement of applications running on mobile devices. **Journal of Network and Computer Applications**, Elsevier, v. 56, p. 28–40, 2015.

KOLIAS, C.; AHLAWAT, S.; ASHTON, C. *et al.* Openflow-enabled mobile and wireless networks. **White Paper**, 2013.

KOSTA, S.; AUCINAS, A.; HUI, P.; MORTIER, R.; ZHANG, X. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In: **INFOCOM, 2012 Proceedings IEEE**. [S.l.: s.n.], 2012. p. 945–953. ISSN 0743-166X.

KOVACHEV, D.; CAO, Y.; KLAMMA, R. Mobile cloud computing: a comparison of application models. **arXiv preprint arXiv:1107.4940**, 2011.

KUMAR, K.; LIU, J.; LU, Y.-H.; BHARGAVA, B. A survey of computation offloading for mobile systems. **Mobile Networks and Applications**, Springer, v. 18, n. 1, p. 129–140, 2013.

KUMAR, K.; LU, Y.-H. Cloud computing for mobile users: Can offloading computation save energy? **Computer**, IEEE, n. 4, p. 51–56, 2010.

LIAO, L.; QIU, M.; LEUNG, V. C. Software defined mobile cloudlet. **Mobile Networks and Applications**, Springer, v. 20, n. 3, p. 337–347, 2015.

MAIA, M. E. F.; FONTELES, A.; NETO, B.; GADELHA, R.; VIANA, W.; ANDRADE, R. M. C. Loccam - loosely coupled context acquisition middleware. In: **Proceedings of the 28th Annual ACM Symposium on Applied Computing**. New York, NY, USA: ACM, 2013. (SAC '13), p. 534–541. ISBN 978-1-4503-1656-9. Disponível em: <http://doi.acm.org/10.1145/2480362.2480465>.

PITKÄNEN, M.; KÄRKKÄINEN, T.; OTT, J.; CONTI, M.; PASSARELLA, A.; GIORDANO, S.; PUCCINELLI, D.; LEGENDRE, F.; TRIFUNOVIC, S.; HUMMEL, K. *et al.* Scampi: service platform for social aware mobile and pervasive computing. **ACM SIGCOMM Computer Communication Review**, ACM, v. 42, n. 4, p. 503–508, 2012.

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

REGO, P. A. L.; 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: **2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)**. [S.l.: s.n.], 2017.

REGO, P. A. L.; COSTA, P. B.; COUTINHO, E. F.; ROCHA, L. S.; TRINTA, F. A.; SOUZA, J. N. de. Performing computation offloading on multiple platforms. **Computer Communications**, Elsevier, 2016. ISSN 0140-3664.

SANAEI, Z.; ABOLFAZLI, S.; GANI, A.; BUYYA, R. Heterogeneity in mobile cloud computing: Taxonomy and open challenges. **IEEE Communications Surveys Tutorials**, v. 16, n. 1, p. 369–392, First 2014. ISSN 1553-877X.

SANTOS, G.; REGO, P.; TRINTA, F. Uma proposta de solução para offloading de métodos entre dispositivos móveis. In: **XVII WTD - WebMedia 2017**. [S.l.: s.n.], 2017. p. 76–81.

SANTOS, G.; TRINTA, F.; REGO, P.; GOMES, F. Caos d2d: Uma solução para offloading de métodos entre dispositivos móveis. In: **XVI WFA**. [S.l.: s.n.], 2017. p. 164–168.

SATYANARAYANAN, M. Mobile computing. **Computer**, IEEE, v. 26, n. 9, p. 81–82, 1993.

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

SATYANARAYANAN, M.; BAHL, P.; CACERES, R.; DAVIES, N. The case for vm-based cloudlets in mobile computing. **IEEE Pervasive Computing**, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 8, n. 4, p. 14–23, out. 2009. ISSN 1536-1268. Disponível em: <http://dx.doi.org/10.1109/MPRV.2009.82>.

SHIRAZ, M.; GANI, A.; KHOKHAR, R.; BUYYA, R. A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. **Communications Surveys Tutorials, IEEE**, v. 15, n. 3, p. 1294–1313, Third 2013. ISSN 1553-877X.

TEO, C. L. V. **Hyrax: Crowdsourcing Mobile Devices to Develop Proximity-Based Mobile Clouds**. 97 p. Tese (Doutorado) — Carnegie Mellon University, Pittsburgh, 2012.

WEISER, M. The computer for the 21st century. **Scientific american**, New York, v. 265, n. 3, p. 94–104, 1991.

APÊNDICE A – DADOS COLETADOS NOS EXPERIMENTOS

Esses são os dados coletados durante os experimentos descritos no Capítulo 5. Cada uma das linhas na tabela abaixo apresenta a média de tempos médios das execuções de métodos de cada caso, com intervalo de confiança de 95%. Os dados na íntegra para todos os exerimentos estão disponíveis em https://github.com/gabrielgbs/cd2d-tests

Os dados coletados nos experimentos para avaliação de *offloading* estão dispostos na tabela abaixo. As linhas representam casos com variações de aplicativo, método, argumento, e onde o método foi executado:

- C1: *Offloading* realizado para o *Handset* A;
- C2: *Offloading* realizado para o *TV Box*;
- C3: Métodos realizados sem *offloading* (caso de controle).

Cenário	Aplicativo	Método	Argumento	Tempo decorrido (s)	Potência média (mW)	Energia consumida (J)	Tempo médio por execução (ms)	Intervalo de confiança tempo médio (95%)	Energia média consumida por método (J)
C1	Camera Offloading	snow	1.6MP	166.84	767.49	128.05	2374.23	328.17	1.82
		emboss	1.6MP	553.42	594.34	328.92	15541.80	40.65	9.24
	Matrix Operations	add	200	91.98	1006.41	92.57	1946.63	299.77	1.96
			800	801.78	1109.15	889.29	25359.63	98.74	28.13
		multiply	200	104.35	858.32	89.57	2379.77	27.38	2.04
			800	2501.08	612.61	1532.19	165487.13	342.50	101.38
C2	Camera Offloading	snow	1.6MP	187.52	781.22	146.49	2132.43	141.64	1.67
		emboss	1.6MP	1928.96	569.50	1098.54	59885.43	164.56	34.10
	Matrix Operations	add	200	69.80	978.87	68.33	1153.37	41.13	1.13
			800	497.50	1364.62	678.90	15937.80	59.89	21.75
		multiply	200	162.69	751.49	122.26	4335.67	25.23	3.26
			800	4156.73	590.16	2453.14	275420.73	245.05	162.54
C3	Camera Offloading	snow	1.6MP	75.36	937.14	70.62	1003.70	4.31	0.94
		emboss	1.6MP	2167.00	1135.17	2459.91	70844.27	143.61	80.42
	Matrix Operations	add	200	35.26	978.87	34.51	8.37	0.35	0.01
			800	56.67	834.97	47.32	105.93	1.74	0.09
		multiply	200	93.40	910.78	85.07	1987.57	22.70	1.81
			800	3341.53	1101.38	3680.29	220197.60	3358.73	242.52

Os dados coletados nos experimentos para avaliação de diferentes estratégias de carregamento de dependências estão dispostos na tabela abaixo. As linhas apresentando a média de tempo de execução em cada caso, com variações de método executado e tamanho do arquivo APK da aplicação a ser transmitido.

Método	Tamanho APK	Classloader em cache	Tempo (ms)	IC tempo (ms)
Emboss	52MB	não	4653.79	119.52
Snow	52MB	não	1293.99	211.47
Emboss	52MB	sim	4710.29	111.53
Snow	52MB	sim	1468.80	585.17
Emboss	1.7MB	não	4768.78	120.13
Snow	1.7MB	não	1711.63	218.50
Emboss	1.7MB	sim	4744.12	108.09
Snow	1.7MB	sim	1423.05	95.13