Uma Proposta de Solução para Offloading de Métodos entre Dispositivos Móveis

Gabriel B. dos Santos, Paulo A. L. Rego e Fernando Trinta
gabrielsantos@great.ufc.br,pauloalr@ufc.br,fernando.trinta@dc.ufc.br
Universidade Federal do Ceará
Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat)
Fortaleza, Ceará

ABSTRACT

In spite of the increasing processing power of smart handheld devices, their capacity is always a few steps behind their contemporary desktop counterparts. Besides, mobile devices have limited power supplies, which leads software designers to keep energy consumption in mind. An alternative to help overcome this issue is using the offloading (or cyber foraging) 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 (i.e., a cloudlet). This proposal for master's thesis aims to develop a framework – called CAOS D2D – which allows a mobile device to offload tasks to another mobile device, as well as acting as an offloading server too. We briefly describe the development process of our implementation, starting from another offloading solution developed by our research group. We also describe CAOS D2D architecture, and finally, we define some future works to guide the evolution path of our solution.

KEYWORDS

Mobile Cloud Computing, Offloading, Device-to-Device

1 CARACTERIZAÇÃO DO PROBLEMA

O uso de tecnologias digitais está cada vez mais presente na sociedade mundial. Essa tendência é fortemente representa pelo aumento considerável no uso de aplicativos móveis com acesso à Internet por praticamente todas as camadas sociais. 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. Dispositivos móveis precisam ser compactos para serem portáteis, e com isso, possuem limitações em relação aos recursos de processamento e armazenamento. Além disso, seu funcionamento depende de uma bateria, que em geral tem autonomia de funcionamento de poucas horas. Tais restrições são ainda mais problemáticas quando as aplicações têm uma tendência forte de uso intensivo de dados e sensores embutidos nos dispositivos.

Uma das possíveis soluções para contornar este problema é a Computação Móvel em Nuvem (do inglês, *Mobile Cloud Computing* - MCC). Segundo Dinh et al. [2], MCC tem por objetivo provisionar um conjunto de serviços equivalentes aos da nuvem, adaptados à capacidade de dispositivos com recursos restritos, de modo a trazer melhorias de desempenho das aplicações ou 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 [14], 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 [3].

O suporte a *offloading* já foi proposto por vários estudos [1, 4, 7, 10, 17]. Um deles é o CAOS (*Context Acquisition and Offloading System*) [7], 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 [6, 7], porém, sua atual versão impõe restrições no tipo de máquina utilizada como *cloudlet*, o que impede o *offloading* de tarefas entre dois *smartphones*. Com isso, em um cenário como de, por exemplo, uma catástrofe natural, a interação entre dispositivos seria prejudicada, uma vez que a infraestrutura estaria possivelmente perdida, e a comunicação *ad-hoc* seria a única disponível.

Este trabalho aborda essa questão, ao apresentar 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 parte da refatoração de componentes existentes no CAOS, 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 melhorar o desempenho de uma aplicação móvel, o que representa mais um passo dentro do objetivo das aplicações distribuídas e ubíquas.

A partir desta introdução, é apresentado um referencial teórico sobre Mobile Cloud Computing na Seção 2. A Seção 3 discute os trabalhos relacionados; a Seção 4 introduz a solução CAOS D2D, sua arquitetura e principais componentes. Por fim, a Seção 8 conclui o trabalho e apresenta trabalhos futuros para evolução da solução proposta.

In:XVII Workshop de Teses de Dissertações (WTD 2017), Gramado, Brasil. Anais do XXIII Simpósio Brasileiro de Sistemas Multimídia e Web: Workshops e Pôsteres. Porto Alegre: Sociedade Brasileira de Computação, 2017.

ISBN 978-85-7669-380-2.

2 MOBILE CLOUD COMPUTING

MCC é um paradigma que une duas tecnologias heterogêneas: computação móvel e computação em nuvem [3]. Esse paradigma destina-se a mitigar as limitações dos dispositivos móveis em relação ao consumo de energia e desempenho, utilizando recursos de nuvens para aumentar tanto o poder de processamento como a capacidade de armazenamento dos dispositivos.

Existem várias definições para MCC. Em [16], os autores afirmam que 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. Segundo [13], 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-go*.

2.1 *Offloading*

Existem vários temas de pesquisa relacionados a MCC, sendo destacado na literatura o *offloading* [3], técnica que visa aumentar o desempenho e reduzir o consumo de energia de dispositivos móveis por meio da migração de processamento ou dados de dispositivos móveis para outras infraestruturas, com maior poder computacional e armazenamento. É importante destacar que *offloading* é diferente do modelo cliente-servidor tradicional, em que os clientes sempre delegam a responsabilidade de realizar o processamento ao servidor. No *offloading*, quando não há conexão entre dispositivo móvel e servidor, o processamento é realizado no dispositivo. Quando existe a conexão e existem ganhos em se migrar dados ou processamento, o *offloading* é realizado.

Tipicamente, existem dois tipos principais de *offloading*: processamento e dados [3]. 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 preservar a carga 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 [6].

2.2 Onde Executar o *Offloading*?

Existem várias questões de pesquisa relacionadas ao *offloading*. Dentre as mais importantes, pode-se citar: Por que, Quando, Como e Onde executar o *offloading* [3, 9]. Este trabalho foca na última questão. Em geral, a execução dessa técnica pode acontecer em três locais possíveis: (*i*) uma nuvem pública; (*ii*) um *cloudlet*; e (*iii*) outro dispositivo móvel.

Na execução em nuvem pública, os desenvolvedores de aplicativos móveis podem utilizar os serviços da nuvem para obter um bom desempenho, assim como elasticidade e conectividade a redes sociais para melhorar os serviços dos aplicativos. A execução em *cloudlet* é proposta em [15] e tem como principal objetivo aproximar os serviços de *offloading* dos dispositivos móveis para serem executados em máquinas locais, que podem ser *desktops* ou *laptops* 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*, em que dispositivos mais robustos cedem parte dos seus recursos computacionais para resolver uma tarefa em comum a outros dispositivos [3] mais fracos, como ilustrado na Figura 1.

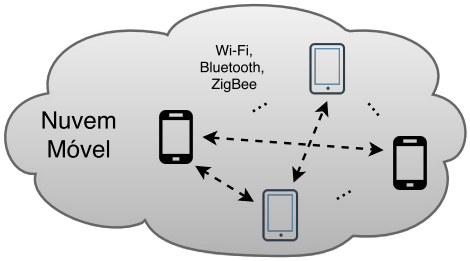


Figura 1: Ambiente de execução de *offloading* formado por uma nuvem de dispositivos.

3 TRABALHOS RELACIONADOS

O CAOS D2D tem como principal requisito de projeto oferecer suporte a *offloading* entre dispositivos móveis. Tal suporte é proposto por vários estudos [1, 4, 7, 10, 17]. O *framework* proposto em [1] oferece suporte a *offloading* D2D, em um modelo de *Cloud Federation*, de maneira que os servidores oferecem serviços dentro da rede, como uma nuvem. A solução conta com mecanismos de privacidade e incentivos, porém o *framework* pressupõe que haja um diretório central de *cloudlets* móveis dentro da rede, o que reduz a disponibilidade do serviço.

A abordagem proposta em [10] propõe um *framework* para *offloading* D2D, usando o protocolo *OpenFlow* e redes definidas por software para abstrair a camada de rede entre os dispositivos. Entretanto, as implementações existentes do *OpenFlow* para dispositivos móveis são relativamente recentes, e ainda enfrentam alguns problemas [8]. A granularidade da solução é a nível de camadas de aplicação, separadas em *front-end* e *back-end*. A camada *back-end* da aplicação pode ser executada remotamente em outro dispositivo móvel.

O *framework* Hyrax [17] é voltado para *crowdsourcing*, baseando-se em *Map Reduce*, derivado do *Hadoop*[1]. Apesar da proposta de *crowdsourcing* em dispositivos móveis alcançar resultados satisfatórios, a arquitetura proposta conta com uma série de pressupostos que nem sempre podem ser cumpridos:

- A necessidade de um nó central que não pode falhar.
- Os dados não podem sofrer alterações durante o processamento.
- Todos os dispositivos dentro da nuvem móvel devem possuir de antemão os dados a serem processados.

[1] *http://hadoop.apache.org/*

A arquitetura *AnyRun Computing* (ARC) [4] propõe um modelo de *offloading* D2D dinâmico e oportunista, com granularidade de *offloading* a nível de método. Da mesma forma que o CAOS (D2D), o desenvolvedor de aplicações deve anotar os métodos propensos a *offloading*.

3.1 CAOS

Um trabalho que merece especial atenção é o CAOS. Uma plataforma de software desenvolvida dentro do nosso grupo de pesquisa, com a finalidade de ajudar no desenvolvimento de aplicações móveis e sensíveis ao contexto da plataforma Android; com suporte ao mecanismo de *offloading* para delegar a migração e o processamento de dados contextuais de dispositivos móveis para plataformas de nuvem [7]. O CAOS tem como base duas soluções: *(i) Loosely Coupled Context Acquisition Middleware* (LoCCAM) [11], que é um *middleware* que ajuda 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) [12], 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. Assim como o CAOS D2D, o CAOS provê suporte a *offloading* através de anotações Java a serem aplicadas nos métodos passíveis de *offloading*[2]. Mas diferente do CAOS D2D, esse *offloading* é realizado para *cloudlets* disponibilizados na mesma rede *Wi-fi* do dispositivo cliente, em vez de dispositivos móveis. No CAOS, a tarefa de tomada de decisão (se é necessário ou não o *offloading*) é compartilhada, parte dela é executada no dispositivo móvel e outra parte é executada no servidor. No CAOS D2D, apenas o cliente participa na tomada dessa decisão.

4 CAOS D2D

A ideia de desenvolver um *framework* para dar suporte a *offloading* entre dois dispositivos móveis surge da necessidade de se realizar *offloading* de computação quando não há uma nuvem/*cloudlet* disponível para tal.

O principal objetivo deste trabalho é realizar *offloading* diretamente de um dispositivo móvel para outro. Por isso, foi adotada a abordagem de adaptar componentes existentes no CAOS para suportar tal funcionalidade.

O CAOS D2D, assim como o CAOS, foi desenvolvido como um *framework* para o sistema operacional Android. Com respeito ao suporte de *offloading* do CAOS, a granularidade adotada é a nível de método, que deve ser marcado com a anotação *@Offloadable*. O CAOS D2D suporta execução local ou remota de métodos anotados, dependendo da disponibilidade de servidores na mesma rede *Wi-Fi* do cliente. O CAOS D2D possui um mecanismo de descoberta para procurar dispositivos móveis fazendo o papel de *cloudlet* na mesma rede do usuário.

4.1 Arquitetura

O CAOS D2D possui componentes que são executados no dispositivo móvel cliente e no dispositivo móvel que fará o papel de *cloudlet*. A Figura 2 ilustra esses componentes.

[2]Ver seção 4 (CAOS D2D), na página 3

O *Offloading Monitor* é o componente responsável por monitorar a execução das aplicações e interceptar o fluxo de execução do método marcado. No CAOS D2D (e no CAOS), os métodos são marcados com anotação Java (*@Offloadable*), o qual define que o método deve executar preferencialmente fora do dispositivo móvel. Ao interceptar a execução do método, o *Offloading Monitor* analisa se há algum servidor CAOS D2D disponível na mesma rede. 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*/nuvem móvel. O componente *Discovery Client* usa um mecanismo baseado em UDP/Multicast para descobrir servidores CAOS D2D (Ou CAOS) disponíveis na rede. Ao descobrir novos servidores, dados sobre eles – tipo de servidor (CAOS ou dispositivo móvel), nível de carga da bateria, endereço IP – são armazenados em uma estrutura de dados; O *Discovery Client* também é responsável por disparar o *deploy* de dependências do cliente para o servidor. O módulo *Profile Monitor* é responsável por monitorar a conectividade entre cliente e servidor (e.g., *ping*). O módulo *Deploy Client*, é responsável por realizar a injeção de dependências em formato .apk dinamicamente do cliente para o servidor, de forma que este se torne apto a receber chamadas de *offloading* feitas por aquele.

Os principais componentes do lado remoto são: *Discovery Service* , *Profile Services* , *Authentication Service* , *Offloading Service* e *Offloading Method Invocation Service*.

O componente *Discovery Service* responde às chamadas do *Discovery Client*, e fornece o endereço da parte servidora do CAOS D2D para os dispositivos móveis clientes. O *Profile Service* é responsável por responder às chamadas do componente *Profile Monitor*, do lado cliente. O *Deploy Service* recebe dependências enviadas pelo *Deploy Client* e os armazena em um diretório no sistema de arquivos, para que o servidor possa executar adequadamente as chamadas de execução remota feitas pelo cliente. O *Offloading Service* recebe pedidos de *offloading* diretamente do *Offloading Client* e encaminha para o *Offloading Method Invocation Service*, responsável por carregar as dependências e executar o método.

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 *Deploy Service* e *Deploy Client* foram desenvolvidos exclusivamente para esta 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.

4.2 Implementação

Para implementação do lado servidor do CAOS D2D, a abordagem adotada foi partir da implementação existente do *Offloading Method Invocation Service* – que é o componente do CAOS que reside na máquina virtual que executa os métodos remotos, e adicionar serviços a ela, de forma a tornar o novo componente completamente autônomo, e posteriormente, fazer a adição do *Deploy Service*.

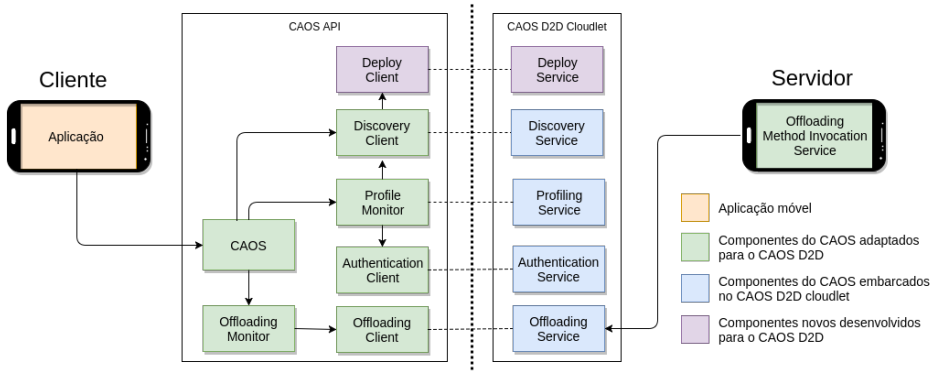


Figura 2: Visão geral da arquitetura do CAOS D2D

No lado cliente, foram feitas adaptações nos componentes *Offloading Client* e *Discovery Client*, de modo que eles se comuniquem diretamente com a versão modificada do *Offloading Method Invocation Service*. O *Offloading Client* foi modificado de forma a agregar parte da responsabilidade do *Offloading Service*, para fazer o *offloading* diretamente para o dispositivo móvel em vez de um *cloudlet*. O *Discovery Client* foi alterado para se comunicar com o *Discovery Service* incorporado no lado servidor. Posteriormente, foi implementado o componente *Deploy Service*, para receber as dependências enviadas pelo *Deploy Client*.

Os componentes do CAOS D2D, incluindo componentes novos e componentes do CAOS alterados para implementação do CAOS D2D, são apresentados na Figura 2. As alterações para implementação do lado cliente do CAOS D2D foram feitas sem alterações na API do CAOS. Isso significa que códigos-fonte desenvolvidos para serem executados com o CAOS também podem ser usados com o CAOS D2D, fazendo-se necessária apenas a mudança de biblioteca, mas não de código-fonte.

5 OBJETIVOS E CONTRIBUIÇÕES ESPERADAS

O objetivo deste trabalho é apresentar um *framework* para prover *offloading* de maneira transparente para desenvolvedores, com particionamento de aplicações a nível de método. A solução apresentada tem como principais pontos de foco o aspecto da mobilidade, apresentando a possibilidade de *offloading* para dispositivos móveis que estejam usando a mesma rede *Wi-fi* do dispositivo cliente; e a simplicidade, permitindo que desenvolvedores marquem métodos como elegíveis a *offloading* usando uma anotação *Java*[3].

6 ESTADO ATUAL DO TRABALHO

Na implementação atual do CAOS D2D, todos os módulos[4] estão operando normalmente. Atualmente, a solução está entrando na segunda fase de testes[5], maturação e correção de *bugs*.

[3]Ver seção 4 (CAOS D2D), na página 3
[4]Ver seção 4.1 (Arquitetura), na página 3
[5]Ver seção 7 (Testes e Avaliação da Solução), na página 4

7 TESTES E AVALIAÇÃO DA SOLUÇÃO

A avaliação do CAOS D2D foi dividida em duas grandes fases: A primeira fase foi executada em uma momento inicial da implementação da solução, no qual apenas os módulos relacionados diretamente ao *offloading* estavam implementados. Essa fase teve como objetivos mensurar de maneira preliminar ganhos de desempenho de aplicativos, bem como melhorias na preservação da vida útil de bateria em condições ideais de conectividade.

A segunda fase está sendo iniciada no momento da escrita desse *paper* – momento no qual a implementação do CAOS D2D está mais completa, com todos os seus módulos funcionais. Essa fase tem como objetivos:

- Obter resultados mais definitivos sobre ganhos de desempenho e economia de energia durante o *offloading* de processamento.
- Mensurar o desempenho do *deploy* dinâmico de dependências.
- Aferir o desempenho de diferentes estratégias de *class loading*.

Os testes da segunda fase estão sendo realizados considerando-se diversas condições de conectividade de rede, e diferentes tamanhos de aplicativos.

O tópico a seguir mostra como foi conduzida a primeira fase de testes, e apresenta os resultados obtidos nela.

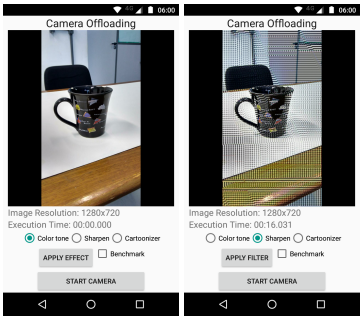
7.1 Primeira Fase

Esta seção apresenta os experimentos realizados para avaliar o desempenho de uma aplicação Android que foi configurada para utilizar o CAOS D2D, a fim de permitir o *offloading* de métodos. O tempo de execução do método escolhido e o consumo de energia do dispositivo foram calculados em diferentes cenários. Para calcular o último, foi utilizado o equipamento *Monsoon Power Monitor*[6].

Os equipamentos utilizados foram: Um *smartphone* – chamado aqui de *Handset* A, com as seguintes características: Android 4.4.2, 1 GB de RAM, chipset Qualcomm Snapdragon 400 (CPU Cortex A7, 1.2 GHz quad-core). Também foi usado um *smartphone* – chamado aqui de *Handset* B, dotado de Android 6.0.1, 2 GB de RAM, chipset Qualcomm Snapdragon 410 (CPU Cortex A53, 1.2 GHz quad-core).

[6]https://www.msoon.com/LabEquipment/PowerMonitor

E por fim, foi utilizado uma máquina virtual Android x86 4.4.2 com 2 GB RAM, 2 VCPUs. rodando no VirtualBox v5.1, em um laptop com Ubuntu 16.04, Intel Core i5-3210m, 8 GB de RAM.



(a) Imagem sem efeito aplicado **(b) Imagem com efeito *Sharpen* aplicado**

Figura 3: Capturas de tela do aplicativo Camera Offloading, tirados no *Handset* B

Os dispositivos foram conectados à mesma rede 802.11n, através de um ponto de acesso da marca TP-Link[7] (modelo TL-WA901ND). A rede estava isolada, de forma que os dispositivos móveis Android não estavam conectados a outros dispositivos externos aos testes. A máquina virtual foi criada com o hipervisor VirtualBox[8] e executava o sistema operacional Android x86 [9] 4.4.2. Além disso, a interface de rede da máquina virtual estava conectada em modo *bridge* à interface de rede Wi-Fi do laptop.

Para os experimentos, foi usado o aplicativo *Camera Offloading*, um aplicativo de processamento de imagens que permite ao usuário tirar uma foto com a câmera e aplicar 3 diferentes efeitos – Em ordem crescente de custo computacional: *Color Tone*, *Sharpen*, e *Cartoonizer*. Tais efeitos têm diferentes requisitos de computação e, por conseguinte, tempos de execução diferentes. Esse aplicativo foi originalmente desenvolvido com suporte ao CAOS, e por consequência, ao CAOS D2D. Durante os experimentos foram utilizadas fotos com as seguintes resoluções: 1MP, 3MP e 6MP, aplicando-se o efeito *Sharpen*.

No cenário C1, foram utilizados o *Handset* A e a máquina virtual. Para cada resolução de fotos o filtro *Sharpen* foi aplicado 30 vezes no *Handset* A, que fez o *offloading* para a máquina virtual. Essa configuração foi escolhida porque dentre os três dispositivos utilizados nos testes, a máquina virtual é o de maior poder computacional, e a intenção do teste era medir o potencial de ganho de desempenho e economia de energia proporcionado pelo *offloading* . No cenário C2, foram utilizados os *Handsets* A e B. O filtro foi aplicado 30 vezes cada no *Handset* B, que fez o *offloading* para o *Handset* A. Por fim, no cenário C3, foi utilizado apenas o *Handset* A, sem fazer *offloading*. O filtro *Sharpen* foi aplicado localmente 30 vezes. O cenário C3 serve como controle, para servir de base de comparação com os cenários C1 e C2. Durante os três experimentos, houve uma pausa programada de 1 segundo entre todas as execuções de métodos, e o consumo de energia do *Handset* A foi medido com o *Power Monitor*. As medições foram feitas para, além de termos métricas sobre o desempenho da aplicação, medir o consumo de energia do *Handset* A quando ele atua como cliente, servidor, e quando executa os processamentos localmente.

Tabela 1: Sumário dos experimentos

Fatores	Dispositivo móvel, dispositivo servidor, método, resolução da foto, e onde executar o método.
Níveis	2 dispositivos móveis (*Handset* A e *Handset* B), 1 método (*Sharpen*), 3 resoluções de fotos (1 MP, 3 MP e 6 MP), e 3 locais para executar os métodos (localmente, em uma máquina virtual, e em um outro dispositivo móvel.).
Projeto	O experimento foi repetido 30 vezes para as combinações de níveis dos fatores em que o *handset* A está envolvido, pois as medições foram feitas de acordo com o ponto de vista do *Handset* A.
Resposta	Tempo decorrido para a execução dos métodos e o consumo de energia por parte do dispositivo móvel.

7.1.1 Resultados Preliminares. Os dados das médias de medições de tempo de execução dos métodos dos 9 casos analisados estão dispostos na figura 4b. A partir desses dados, podemos chegar a algumas conclusões.

Primeiramente, notamos que os casos em que o *Handset* A apresentou maiores tempos de execução de método foram aqueles em que ele agiu como servidor (C2); onde ele fez os processamentos mais custosos e teve a tarefa adicional de receber e enviar os dados pela rede *Wi-Fi*. É notável o custo dessa tarefa adicional em termos de desempenho.

Por esse mesmo motivo, percebemos também que quando a foto a ser processada era pequena – 1MP, o *Handset* A teve um melhor desempenho quando aplicou os efeitos localmente. O ganho de desempenho proporcionado pelo *offloading* não compensou o custo de enviar e receber dados pela rede *Wi-fi*. O tempo médio por execução de método com *offloading* foi 25% maior no efeito *Sharpen*.

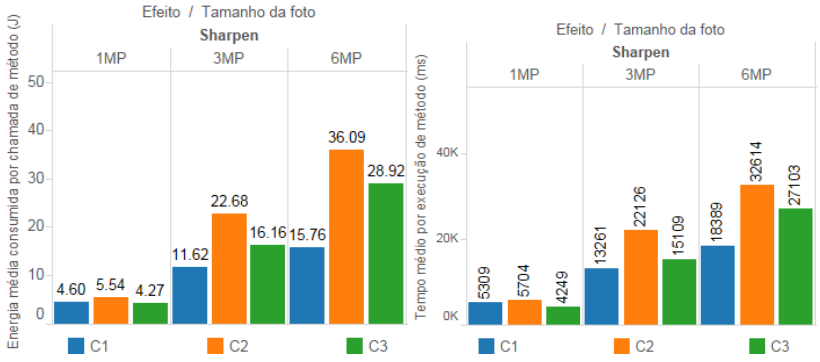
Para as fotos maiores – 3MP e 6MP, o *offloading* se mostra mais interessante, apresentando melhora de desempenho proporcional ao custo computacional da aplicação dos efeitos. Os ganhos médios de desempenho em fotos de 3MP e 6MP foram respectivamente de 12,2% e 32,1%.

Os resultados das medições de energia feitas no *Handset* A estão organizados na Figura 4a.

A partir desses dados, podemos concluir que o consumo médio de energia nos diferentes cenários de teste se comporta de maneira similar ao consumo de tempo. Para fotos de 1MP, foi mais vantajoso aplicar o efeito *Sharpen* localmente. No caso de fotos maiores – 3MP e 6MP, o *offloading* representou economia de energia com relação à execução local.

Apesar do ambiente controlado onde foram realizados os testes, o estudo feito em [5] 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

[7] http://www.tp-link.com
[8] https://www.virtualbox.org.
[9] www.android-x86.org



(a) Energia Consumida (J) **(b) Tempo Gasto (ms)**

Figura 4: Medições feitas no *Handset* A nos diferentes cenários com o aplicativo *CameraOffloading*

de rede de distância ainda pode ser vantajoso, o que endossa os benefícios do *offloading* entre dispositivos móveis em situações do mundo real.

8 CONSIDERAÇÕES FINAIS

Neste artigo, foi apresentado 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.

Em trabalhos futuros, pretende-se implantar um algoritmo de decisão mais robusto, que faça uma consideração mais ajustada de fatores como complexidade do método a ser executado, tamanho dos argumentos e métricas de conectividade de rede. 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; além da implementação do suporte a dados contextuais também pelo CAOS D2D.

REFERÊNCIAS

[1] Abdalla Artail, Karim Frenn, Haidar Safa, and Hassan Artail. 2015. A Framework of Mobile Cloudlet Centers Based on the Use of Mobile Devices as Cloudlets. In *Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference on*. IEEE, 777–784.

[2] Hoang T. Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. 2013. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing* 13, 18 (2013), 1587–1611. https://doi.org/10.1002/wcm.1203

[3] Niroshinie Fernando, Seng W. Loke, and Wenny Rahayu. 2013. Mobile Cloud Computing. *Future Gener. Comput. Syst.* 29, 1 (Jan. 2013), 84–106. https://doi.org/10.1016/j.future.2012.05.023

[4] Alan Ferrari, Silvia Giordano, and Daniele Puccinelli. 2016. Reducing your local footprint with anyrun computing. *Computer Communications* 81 (2016), 1–11.

[5] Colin Funai, Cristiano Tapparello, and Wendi Heinzelman. 2016. Mobile to Mobile Computational Offloading in Multi-hop Cooperative Networks. In *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 1–7.

[6] Francisco A.A. Gomes, Windson Viana, Lincoln S. Rocha, and Fernando Trinta. 2016. A Contextual Data Offloading Service With Privacy Support. In *Proceedings of the 22Nd Brazilian Symposium on Multimedia and the Web (Webmedia '16)*. ACM, New York, NY, USA, 23–30. https://doi.org/10.1145/2976796.2976860

[7] Francisco A A Gomes, Paulo A L Rego, Lincoln Rocha, Jose N de Souza, and Fernando Trinta. 2017. CAOS: A Context Acquisition and Offloading System. In *2017 IEEE 41th Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1.

[8] C Kolias, S Ahlawat, C Ashton, et al. 2013. OpenFlow-Enabled Mobile and Wireless Networks. *White Paper* (2013).

[9] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. 2013. A survey of computation offloading for mobile systems. *Mobile Networks and Applications* 18, 1 (2013), 129–140.

[10] Lingxia Liao, Meikang Qiu, and Victor CM Leung. 2015. Software Defined Mobile Cloudlet. *Mobile Networks and Applications* 20, 3 (2015), 337–347.

[11] Marcio E. F. Maia, Andre Fonteles, Benedito Neto, Romulo Gadelha, Windson Viana, and Rossana M. C. Andrade. 2013. LOCCAM - Loosely Coupled Context Acquisition Middleware. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC '13)*. ACM, New York, NY, USA, 534–541. https://doi.org/10.1145/2480362.2480465

[12] Paulo A L Rego, Philipp B Costa, Emanuel F Coutinho, Lincoln S Rocha, Fernando AM Trinta, and Jose N de Souza. 2016. Performing computation offloading on multiple platforms. *Computer Communications* (2016). https://doi.org/10.1016/j.comcom.2016.07.017

[13] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya. 2014. Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges. *IEEE Communications Surveys Tutorials* 16, 1 (First 2014), 369–392. https://doi.org/10.1109/SURV.2013.050113.00090

[14] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. 2009. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing* 8, 4 (Oct. 2009), 14–23. https://doi.org/10.1109/MPRV.2009.82

[15] Mahadev Satyanarayanan, P. Bahl, R Caceres, and N. Davies. 2009. The Case for VM-Based Cloudlets in Mobile Computing. *Pervasive Computing, IEEE* 8, 4 (Oct 2009), 14–23. https://doi.org/10.1109/MPRV.2009.82

[16] M. Shiraz, A. Gani, R.H. Khokhar, and R. Buyya. 2013. A Review on Distributed Application Processing Frameworks in Smart Mobile Devices for Mobile Cloud Computing. *Communications Surveys Tutorials, IEEE* 15, 3 (Third 2013), 1294–1313. https://doi.org/10.1109/SURV.2012.111412.00045

[17] Vincent Teo and Chye Liang. 2012. *Hyrax: Crowdsourcing Mobile Devices to Develop Proximity-Based Mobile Clouds*. Ph.D. Dissertation. Carnegie Mellon University Pittsburgh, PA.