Impactos do *Offloading* de Processamento no Tempo de Execução e Consumo Energético de Dispositivos Móveis

Gabriel B. dos Santos[1], **Fernando A. M. Trinta**[1] **e Paulo A. L. Rego**[1]

[1]Group of Computer Networks, Software Engineering and Systems (GREat)
Universidade Federal do Ceará (UFC) – Fortaleza, CE – Brasil

{gabrielsantos, fernandotrinta}@great.ufc.br, pauloalr@ufc.br

Abstract. *With the goal of simplifying the design and development of applications that use computation offloading, we developed the framework CAOS D2D to provide an abstraction layer for dealing with low-level tasks related to offloading of methods between Android mobile devices. This paper presents experiments performed to evaluate different aspects of the CAOS D2D framework, such as execution time and power consumption of devices during method offloading. Besides, we also performed experiments to evaluate the applications dependency deployment. The experiments show that computation offloading improves execution times by up to 78% and reduces power consumption by up to 88% in comparison to local executions of the same methods.*

Resumo. *Com o objetivo de facilitar o projeto e desenvolvimento de aplicações que usufruam do offloading de processamento, o framework CAOS D2D foi desenvolvido para abstrair tarefas de baixo nível relacionadas ao offloading de métodos entre dispositivos móveis Android. Este trabalho apresenta experimentos realizados para avaliar diferentes aspectos do framework CAOS D2D, como o tempo de execução de tarefas e o consumo de energia de dispositivos durante o offloading de métodos. Além disso, também foram realizados experimentos para avaliar o procedimento de deploy de dependências de aplicações. Os experimentos mostraram que o offloading de processamento proporcionou melhorias de tempos de execução de até 78% e economia de energia de até 88%, com relação às execuções locais dos mesmos métodos.*

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* 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, 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 com relação aos seus 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], 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 consideração a necessidade de poupar o consumo de energia, de modo a aumentar a autonomia de funcionamento do aparelho. Todas estas restrições são ainda mais problemáticas quando usuários se engajam em utilizar aplicações com tendência de uso intensivo de dados e sensores embutidos nos dispositivos [Júnior et al. 2016, Gonçalves et al. 2016]. 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 e/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* obtém resultados ainda melhores quando se dá entre dispositivos próximos, já que a latência de comunicação é um dos principais empecilhos para que a migração de tarefas seja mais 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 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* para dispositivos móveis já foi proposto por vários estudos [Artail et al. 2015, Liao et al. 2015, Teo 2012, Ferrari et al. 2016]. Um deles é o CAOS D2D [Santos et al. 2017], uma plataforma para *offloading* de métodos entre dispositivos Android. O CAOS D2D permite que o desenvolvedor marque quais métodos são candidatos a migrar para um outro dispositivo, e de acordo com a disponibilidade de servidores aptos para o *offloading* na rede, o sistema decide quando migrar a tarefa. Nesse contexto, este trabalho tem como objetivo realizar experimentos, utilizando o *framework* CAOS D2D, para avaliar o impacto do *offloading* de processamento no tempo de execução de métodos e consumo energético de dispositivos móveis, bem como o procedimento de *deploy* de dependências de aplicações – uma das característica do *framework*.

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

[2]http://translate.google.com

[3]http://www.nike.com.br/running/nrc-app

O restante deste artigo está dividido da seguinte forma: a Seção 2 apresenta os trabalhos relacionados, comparando as principais características deles a este trabalho. A Seção 3 apresenta uma visão geral do CAOS D2D, *framework* utilizado nos experimentos realizados. A Seção 4 discute os experimentos que foram realizados para, além de avaliar impactos do *offloading* nos tempos de execução e consumo de energia de dispositivos móveis, avaliar também diferentes aspectos do CAOS D2D. E finalmente, a Seção 5 expõe as considerações finais sobre os experimentos e lista os trabalhos futuros.

2. Trabalhos Relacionados

Nos últimos anos, uma variedade de estudos propôs o suporte a *offloading* de processamento de um dispositivo móvel 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.

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 SaaS tradicional, 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 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 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 do tempo de execução de 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* implementado em cima de uma rede definida por software para abstrair a comunicação de rede entre os dispositivos. Entretanto, as implementações existentes do *OpenFlow* para dispositivos móveis são relativamente recentes e enfrentam alguns problemas em dispositivos móveis [Kolias et al. 2013]. Além disso, a divisão da aplicação em duas grandes camadas é um pressuposto para utilização da solução, o que pode exigir alterações de projeto em alguns aplicativos.

O *framework* Hyrax [Teo 2012] é uma solução de *offloading device-to-device* voltada para *crowdsourcing*, com foco na melhoria do tempo de execução de aplicações utilizando uma implementação de *Map Reduce* derivada do *Hadoop*. O Hyrax busca dar suporte à formação de *clusters* compostos por dispositivos Android. Apesar da proposta de *crowdsourcing* com dispositivos móveis alcançar resultados satisfatórios, Hyrax conta com uma série de pressupostos que nem sempre podem ser cumpridos, 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, para utilizar o Hyrax, deve-se implementar as funções *Map* e *Reduce* – cerne do paradigma MapReduce – para as funcionalidades que devem ter as melhorias de tempo de execução.

A arquitetura *Anyrun Computing* (ARC) [Ferrari et al. 2016] propõe um modelo de *offloading* métodos dinâmico e oportunista entre dispositivos Android, com foco em melhoria do tempo de execução e economia de energia. Para utilizar o ARC, o desenvolvedor deve refatorar os códigos-fonte, de maneira que as classes do ARC sejam usadas para realizar as chamadas dos métodos candidatos a *offloading*.

[Ghasemi-Falavarjani et al. 2015] desenvolveram um *middleware* que coleta informações contextuais para tomada de decisão de *offloading* com foco na melhoria do tempo de execução e consumo de energia. Um protótipo foi implementado para a plataforma Android, onde é possível fazer *offloading* para dispositivos móveis. O protótipo foi desenvolvido para uma aplicação de processamento de imagem, onde foi preciso desenvolver código do cliente e do servidor, não sendo, assim, fácil a adaptação em outras aplicações/cenários

O HyMobi [Flores et al. 2017] é um *framework* para *offloading* D2D em que os dispositivos móveis estão organizados em uma espécie de ”rede social”, em que cada dispositivo (dispositivo móvel, *cloud* ou *cloudlet*) possui um perfil e pontuação, sobre os quais as decisões de *offloading* são tomadas. O particionamento de aplicações é a nível de métodos, que devem ser adaptados para que sejam utilizados os recursos do *framework*. Essa adaptação expõe ao desenvolvedor aspectos da reflexão utilizada pela API, o que pode ser um ponto para surgimento de bugs.

A Tabela 1 apresenta um resumo das principais características de cada trabalho, com relação a: sistema operacional, 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:

***Baixo*:** Quando as alterações em código-fonte se resumem 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 para usar 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.

3. CAOS D2D

O CAOS D2D (*Device-to-Device*) [Santos et al. 2017] é uma infraestrutura de software projetada no Grupo de Redes de Computadores, Engenharia de Software e Sistemas da Universidade Federal do Ceará, para auxiliar no desenvolvimento de aplicações móveis e sensíveis ao contexto na plataforma Android. Ele surge como uma versão estendida do CAOS (*Context Acquisition and Offloading System*) [Gomes et al. 2017], e conta com

Tabela 1. Comparação das principais características dos trabalhos citados

Trabalho	Sistema operacional	Foco de melhoria	Abordagem de particionamento	Esforço para adaptação
Artail et. Al (2015)	Multiplataforma	Tempo de execução e armazenamento	Serviços	Alto
Liao, Qiu e Leung (2015)	Multiplataforma	Tempo de execução	Camadas Front end e back end	Alto
Hyrax	Android	Tempo de execução	Map reduce	Alto
ARC	Android	Tempo de execução e economia energética	Métodos (dinâmico)	Médio
Ghasemi-Falavarjani et al. 2015	Android	Tempo de execução e economia energética	Não se aplica*	Alto
HyMobi	Android	Tempo de execução e economia energética	Métodos (dinâmico)	Médio
CAOS D2D	Android	Tempo de execução e economia energética	Métodos (estático e dinâmico)	Baixo

suporte ao mecanismo de *offloading* para delegar a execução de tarefas de dispositivos móveis Android para outros dispositivos Android.

O principal objetivo do CAOS D2D é fornecer uma camada de abstração de tarefas de baixo nível relacionadas ao *offloading*, para facilitar o projeto de aplicações que usem deste. Ele suporta o *offloading* a nível de métodos, que são marcados com uma anotação Java. Além disso, ele possui um mecanismo de descoberta de serviço para identificar dispositivos dispostos a atuar como servidores de processamento disponíveis na mesma rede em que o dispositivo cliente está conectado.

Os componentes do CAOS D2D são executados no dispositivo móvel cliente (CAOS D2D API) e no dispositivo servidor (CAOS D2D Server), como ilustrado na Figura 1. Em sua maioria, os componentes do CAOS D2D são formados por módulos correspondentes – presentes tanto no lado cliente quanto no lado servidor, de forma que esses módulos se comunicam entre si, através da abordagem cliente/servidor.

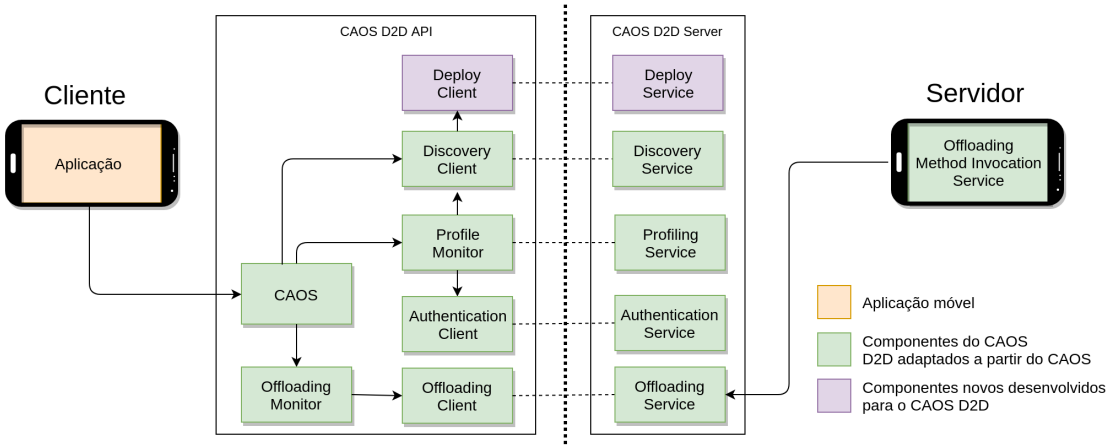


Figura 1. Visão geral da arquitetura do CAOS D2D

Os módulos presentes no CAOS D2D API têm a iniciativa de iniciar todos os fluxos de comunicação da solução. Eles têm a responsabilidade de:

- Anunciar que o dispositivo está presente na rede, através do *Discovery Client*;
- Autenticar o dispositivo cliente no servidor, através do *Authentication Client*;
- Injetar dependências no CAOS D2D Server, por meio do *Deploy Client*. A dependência é uma cópia do aplicativo móvel (arquivo .APK do Android);
- Fazer monitoramento da qualidade da rede, por meio do *Profile Monitor*, onde são monitoradas as métricas: taxa de *download, upload*, latência, dentre outras;
- Iniciar o processo de *offloading* de um método, por meio do *Offloading Client*.

O CAOS D2D Server responde às requisições do cliente e tem responsabilidade de:

- Receber requisições do cliente, tratá-las e respondê-las apropriadamente;
- Manter-se visível para dispositivos clientes, através do *Discovery Service*; requisições de *offloading*;
- Por meio do *Authentication Service*, informar ao dispositivo cliente qual a sua configuração de hardware, estado de bateria e se o servidor possui as dependências necessárias para a aplicação cliente;
- Através do *Deploy Service*, receber dependências de aplicações de clientes, e armazená-las no sistema de arquivos do dispositivo servidor, para ficar apto a realizar o *offloading* para essas aplicações;
- Através do *Offloading Service*, receber requisições de *offloading*, processar os métodos sob demanda, e retornar seus resultados.

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.

4. Planejamento e Resultados dos Experimentos

Com o objetivo de avaliar os impactos do *offloading* de processamento no consumo de energia e tempo de execução de métodos de aplicações, bem como validar as funcionalidades e módulos do CAOS D2D, foram planejados experimentos para realizar medições desses aspectos em aplicações que foram configuradas para utilizar o *framework*. Em resumo, os experimentos foram realizados com os seguintes propósitos:

- Obter resultados sobre impactos de tempo de execução e economia de energia durante o *offloading* de processamento;
- Mensurar o consumo de energia e o tempo decorrido durante a implantação de dependências de aplicações do cliente para o servidor (*deploy*);

Para medir o tempo de execução, os tempos foram coletados com o auxílio de logs de aplicações fornecidos pela plataforma Android. Já a métrica usada para avaliação do consumo de energia pelos dispositivos móveis foi calculada através de medições feitas no aparelho *Monsoon Power Monitor*[4]. O 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", em que fatores como voltagem e capacidade podem ser definidos pelo usuário, e também

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

fornece dados em tempo real sobre o consumo de energia, potência, voltagem, corrente, e estimativa de tempo de duração da "bateria".

Como o *Power Monitor* coleta o consumo de todo o dispositivo, sem discriminar por aplicativo, os processos em segundo plano foram finalizados durante os testes, mantendo em execução apenas os aplicativos diretamente envolvidos nos experimentos.

4.1. Avaliação de *offloading*

Para avaliar os tempos de execução de tarefas e o consumo de energia, experimentos foram realizados em diferentes cenários com duas aplicações Android, configuradas com o CAOS D2D API para permitir o *offloading* de métodos.

Os dispositivos Android utilizados nessa fase foram: um *smartphone* LG G3 Beat, aqui referenciado como "*Handset* A"; um *smartphone* Motorola 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. Os dispositivos foram conectados à mesma rede sem fio 802.11n, através de um ponto de acesso TP-Link TL-WA901ND. A rede estava isolada, de forma que apenas os dispositivos móveis utilizados nos testes estavam conectados à rede.

Tabela 2. Descrição dos equipamentos utilizados

Dispositivo	Configuração
Handset A	Android 4.4.2, 1 GB de RAM, Qualcomm Snapdragon 400 (Cortex A7, 1.2 GHz 4 core)
Handset B	Android 6.0.1, 2 GB de RAM, Qualcomm Snapdragon 410 (Cortex A53, 1.2 GHz 4 core)
TV Box	Android 6.0, 1 GB RAM, Amlogic S905X (Cortex-A53 2 GHz 4 core)

Um dos aplicativos escolhidos para os testes é o Matrix Operations, que gera matrizes quadradas com valores aleatórios, e mede o tempo decorrido em operações de soma e multiplicação. 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 baixo custo computacional; considerando-se N o número de elementos da matriz, a complexidade da adição é calculada como $O(N)$; enquanto a multiplicação de matrizes é uma operação de custo mais alto, com complexidade calculada como $O(N^{\frac{1}{5}})$. O uso desses dois métodos, com diferentes tamanhos de argumentos, pode 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 executadas operações de soma e multiplicação de matrizes quadradas de tamanho 200x200 e 800x800.

O outro aplicativo utilizado nos testes é o Camera Offloading, um aplicativo de processamento de imagens que permite ao usuário tirar uma foto com a câmera e aplicar filtros de imagens da biblioteca PhotoFilter[5]. Dois filtros foram utilizados nos testes: *Snow* e *Emboss*. Esses filtros foram escolhidos por representarem dois exemplos distintos: enquanto o filtro *Snow* é de baixo custo computacional, com execução rápida, mesmo localmente; o filtro *Emboss* tem de custo computacional mais elevado, devido ao fato de ele ser uma combinação de outros filtros. Durante os experimentos, os dois filtros foram

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

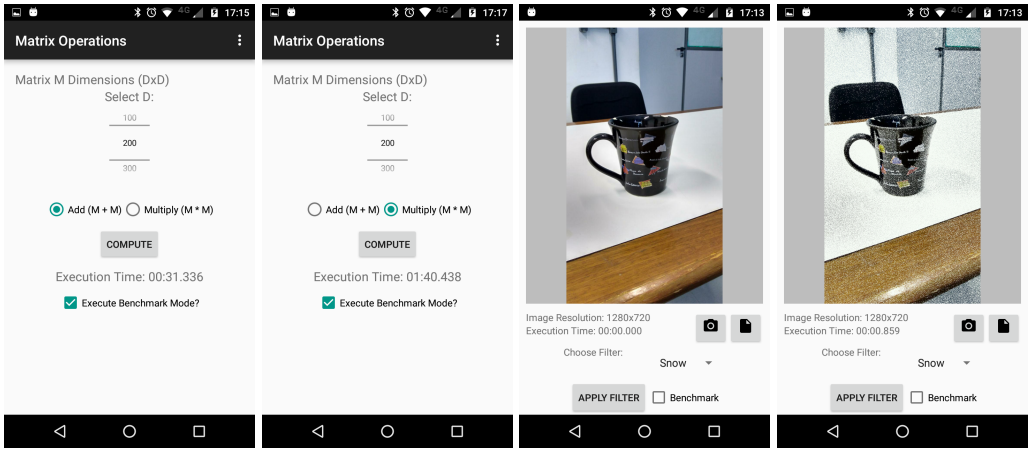


Figura 2. Capturas de tela dos aplicativos utilizados, tiradas no *Handset* B. Matrix Operations, (a) após realizar 30 operações de soma, e (b) 30 operações de multiplicação, com matrizes 200x200. Camera Offloading, (c) antes e (d) após aplicar o efeito *Snow* em uma foto de 1MP

utilizados com fotos de 1,6MP. Os dois aplicativos utilizados nos testes são ilustrados na Figura 2.

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 (cenário de controle, que serve de base de comparação com os outros dois cenários).

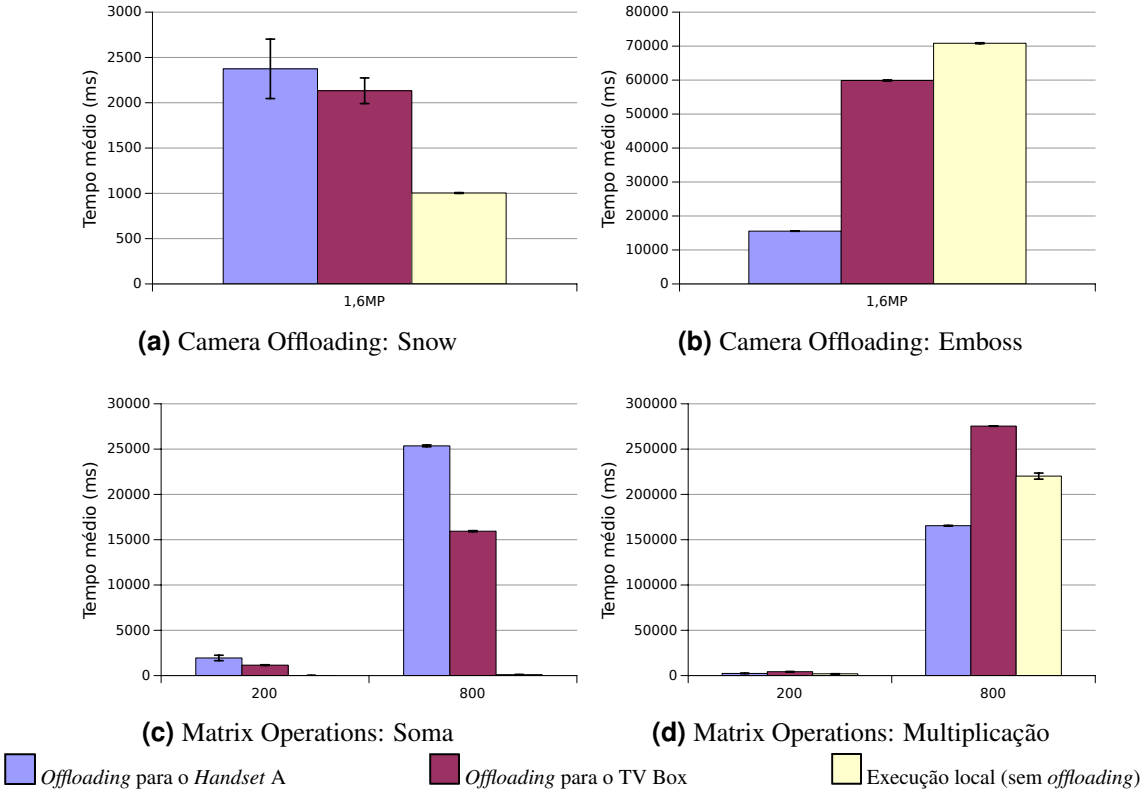
Nos três cenários, foram realizados os mesmos procedimentos: No aplicativo Matrix Operations, foi realizada a soma e a multiplicação de matrizes, com dimensão 200x200 e 800x800, resultando em doze diferentes casos. No aplicativo Camera Offloading, foram aplicados os filtros *Snow* e *Emboss* em imagens de 1,6MP, resultando em 6 casos. No total, foram realizados 18 casos, repetidos trinta vezes, o que totaliza 540 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 coletar 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.

A seguir, são analisados os dados relacionados aos tempos de execução de métodos e consumo de energia por parte do *Handset* B, nos casos possíveis no que diz respeito a *offloading*: quando ele realiza *offloading* para o *Handset* A, para o TV Box, e quando ele executa as tarefas localmente.

4.1.1. Avaliação de tempo de execução

A Figura 3 apresenta as médias de tempo de execução dos métodos para os diferentes casos analisados. Ao analisar os resultados, chega-se às seguintes conclusões. Para realizar o

offloading de um método, o CAOS D2D encapsula o método e seus argumentos em um objeto a ser transmitido pela rede. Pelo fato da transmissão desse objeto ter o seu custo de tempo, nota-se que operações de baixo custo computacional não se beneficiam com o *offloading* em termos de tempo de execução.



(a) Camera Offloading: Snow

(b) Camera Offloading: Emboss

(c) Matrix Operations: Soma

(d) Matrix Operations: Multiplicação

Figura 3. Tempos médios de operações avaliadas nos testes de *offloading*

No caso do aplicativo Matrix Operations, a execução local da soma de matrizes de dimensão 800x800, 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 do objeto transmitido pelo CAOS D2D para realização do *offloading* ser, nesse caso em especial, 5,12MB; e o objeto representando o resultado dessa soma ter em média 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 baixo custo.

No caso do aplicativo Camera Offloading, nota-se que o *Handset* B teve melhores tempos de execução quando aplicou o efeito menos custoso – *Snow* – localmente. A execução local desse efeito foi em média 57,7% mais rápida do que no *Handset* A, e 52,9% mais rápida do que no TV Box.

Em termos de tempos de execução, o *offloading* se mostrou uma opção mais proveitosa para casos em que o custo de execução maior apresenta 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 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 comparação com a a execução local.

No aplicativo Matrix Operations, a multiplicação de matrizes de dimensão 800x800 no TV Box foi em média 25,1% mais lenta do que quando foi executada localmente; 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 diferença de comportamento para diferentes métodos 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 do que para o TV Box.

Esses casos indicam que um dispositivo que executa uma determinada tarefa mais rápido do que outros nem sempre será a melhor opção para o *offloading*.

4.1.2. Avaliação de consumo de energia

Os dados acerca do consumo energético nos experimentos estão organizados na Tabela 3. Partindo da análise desses dados, pode-se observar que o consumo de energia por parte do *Handset* B foi em geral proporcional ao tempo de execução das tarefas. Adicionalmente, medições foram realizadas no *Handset* B para aferir seu consumo de energia em modo *idle*, apenas com a tela ligada. Em 77,42s a potência média mensurada foi de 532,95mW.

Cenário	Aplicativo	Método	Argumento	Potência média (mW)	Energia média consumida por método (J)
C1	Camera Offloading	emboss	1.6MP	594.34	9.24
	Matrix Operations	add	200x200	1006.41	1.96
			800x800	1109.15	28.13
		multiply	200x200	858.32	2.04
			800x800	612.61	101.38
C2	Camera Offloading	emboss	1.6MP	569.50	34.10
	Matrix Operations	add	200x200	978.87	1.13
			800x800	1364.62	21.75
		multiply	200x200	751.49	3.26
			800x800	590.16	162.54
C3	Camera Offloading	emboss	1.6MP	1135.17	80.42
	Matrix Operations	add	200x200	978.87	0.01
			800x800	834.97	0.09
		multiply	200x200	910.78	1.81
			800x800	1101.38	242.52

Tabela 3. Cálculos da energia média consumida em operações avaliadas nos testes de *offloading*

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. A execução do filtro em *offloading* representou para o *Handset* B um aumento na potência elétrica média de 11,5% em C1 e de apenas 6,9% em C2, com relação ao modo *idle*. Para efeito de comparação, em C3 (onde a execução do método foi local)

o aumento de potência elétrica foi de 113%. a caso do aplicadtivo Matrix Operations, a multiplicação de matrizes de dimensão 800x800, por exemplo, se mostrou 58,2% mais econômica em termos de consumo de energia quando realizada em *offloading* para o *Handset* A. Nesse caso, o aumento da potência elétrica média foi de 15% com relação ao modo *idle*. 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%, com aumento de potência com relação ao modo *idle* de 10,7% (para comparação, em C3, essa tarefa demanda um aumento de 106,7% na potência elétrica média). 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, pode-se considerar que, no aspecto de melhora no consumo, o *offloading* é mais proveitoso à medida que a tarefa é mais custosa.

Por outro lado, nos casos em que a tarefa migrada tem custo computacional baixo, percebe-se 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 a TV Box. Para a Matrix Operations, a soma de matrizes de dimensão 800x800, quando executado localmente, apresentou consumo 99,7% menor do que ao fazer *offloading* para o *Handset* A, e 99,6% menor do que realizando o *offloading* para a TV Box.

4.2. Avaliação de *deploy* de dependências

Para realizar os experimentos com finalidade de avaliar o consumo energético e o tempo gasto pelo módulo de *deploy* de dependências do CAOS D2D, foram utilizados os mesmos dispositivos *Handset* A e *Handset* B, utilizados para os experimentos de avaliação de *offloading*. Esses dispositivos foram conectados à uma rede sem fio 802.11n, provido através de um computador que foi configurado para atuar como um ponto de acesso, 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 configurados 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*.

Para realizar o *deploy* de dependências no dispositivo servidor, o CAOS D2D envia

o arquivo APK do aplicativo no qual está sendo usado, em vez de um arquivo dentro do diretório de recursos do aplicativo. 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.

Os resultados obtidos nos experimentos descritos estão organizados na Tabela 4. Partindo da observação desses dados, pode-se perceber que, primeiramente, os tempos decorridos para transmitir os arquivos APK pela rede são próximos dos tempos ideais calculados a partir dos tamanhos dos arquivos APK e da largura de banda da rede. As perdas médias se situaram entre 5,3% e 10,2% acima do tempo ideal, sendo que essas diferenças se tornam menos significativas à medida que o tamanho do arquivo APK aumenta.

Observando a análise de variância na Tabela 5, nota-se que os fatores tamanho do arquivo a ser transmitido, a velocidade de rede e tempo decorrido para transmissão têm impacto na energia consumida pelo dispositivo móvel cliente.

Tabela 4. 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

Tabela 5. 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	9.91×10^{10}	9.91×10^{10}	1.94×10^{32}	$<2 \times 10^{-16}$
Tempo (s)	1	1.73×10^{10}	1.73×10^{10}	3.38×10^{31}	$<2 \times 10^{-16}$
Largura banda	1	4.13×10^{5}	4.13×10^{5}	8.07×10^{26}	$<2 \times 10^{-16}$
Resíduos	112	0.00	0.00		

$\alpha = 0.05$

Um detalhe que chama a atenção é o fato da 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, a 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 indica que a transmissão continuada em alta velocidade aumenta o consumo médio de energia por parte do dispositivo móvel [Barbera et al. 2013].

5. Considerações finais

Este trabalho descreveu e discutiu dois experimentos que foram realizados com o *framework* CAOS D2D para avaliar os impactos do *offloading* de processamento no tempo de

execução e consumo de energia de dispositivos móveis.

No que diz respeito ao tempo de execução de tarefas em *offloading*, os resultados 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. Em alguns casos o *offloading* para o *Handset* A apresentava os melhores tempos de execução, enquanto em outros o *offloading* demonstrava tempos de execução menores se realizado para o *TV Box*.

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, foi possível perceber 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 do tamanho do arquivo APK da aplicação e da velocidade da rede sem fio.

Uma limitação do CAOS D2D 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*, e novos experimentos dentro desses cenários podem ser realizados.

Um trabalho interessante 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.

Referências

Abolfazli, S., Sanaei, Z., Sanaei, M. H., Shojafar, M., and Gani, A. (2016). Mobile cloud computing. *Encyclopedia of Cloud Computing*, page 29.

Artail, A., Frenn, K., Safa, H., and Artail, H. (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*, pages 777–784. IEEE.

Barbera, M. V., Kosta, S., Mei, A., and Stefa, J. (2013). To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In *INFOCOM, 2013 Proceedings IEEE*, pages 1285–1293. IEEE.

Chaffey, D. (2016). Mobile marketing statistics compilation. http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/. Acessado em 09/10/2017.

Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P. (2010). Maui: Making smartphones last longer with code offload. In *MobiSys 2010, Proceedings ACM*, pages 49–62, New York, NY, USA. ACM.

Dinh, H. T., Lee, C., Niyato, D., and Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 13(18):1587–1611.

Fernando, N., Loke, S. W., and Rahayu, W. (2013). Mobile cloud computing. *Future Gener. Comput. Syst.*, 29(1):84–106.

Ferrari, A., Giordano, S., and Puccinelli, D. (2016). Reducing your local footprint with anyrun computing. *Computer Communications*, 81:1–11.

Flores, H., Sharma, R., Ferreira, D., Kostakos, V., Manner, J., Tarkoma, S., Hui, P., and Li, Y. (2017). Social-aware hybrid mobile offloading. *Pervasive and Mobile Computing*, 36:25–43.

Ghasemi-Falavarjani, S., Nematbakhsh, M., and Ghahfarokhi, B. S. (2015). Context-aware multi-objective resource allocation in mobile cloud. *Computers & Electrical Engineering*, 44:218–240.

Gomes, F. A., Rego, P. A., Rocha, L., de Souza, J. N., and Trinta, F. (2017). CAOS: A Context Acquisition and Offloading System. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, pages 957–966. IEEE.

Gonçalves, G. D., Vieira, A. B., da Silva, A. P. C., and Almeida, J. M. (2016). Trabalho colaborativo em serviços de armazenamento na nuvem: Uma análise do dropbox. In *XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*.

Júnior, A. M. S., Sousa, M. L., Xavier, F. Z., Xavier, W. Z., Almeida, J. M., Ziviani, A., Rangel, F., Avila, C., and Marques-Neto, H. T. (2016). Caracterizacao do serviço de táxi a partir de corridas solicitadas por um aplicativo de smartphone. In *XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*.

Kolias, C., Ahlawat, S., Ashton, C., et al. (2013). Openflow-enabled mobile and wireless networks. *White Paper*.

Liao, L., Qiu, M., and Leung, V. C. (2015). Software defined mobile cloudlet. *Mobile Networks and Applications*, 20(3):337–347.

Santos, G., Trinta, F., Rego, P., and Gomes, F. (2017). CAOS D2D: Uma solução para offloading de métodos entre dispositivos móveis. In *XVI WFA*, pages 164–168.

Satyanarayanan, M. (1993). Mobile computing. *Computer*, 26(9):81–82.

Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23.

Teo, C. L. V. (2012). *Hyrax: Crowdsourcing Mobile Devices to Develop Proximity-Based Mobile Clouds*. PhD thesis, Carnegie Mellon University, Pittsburgh.