Um Mecanismo de *Offloading* de Dados com Tomada de Decisão

Joari S. L. Filho, Manuel G. da S. Neto, Paulo A. L. Rego, Danielo G. Gomes

[1]Universidade Federal do Ceará (UFC)
Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat)
Av. Mister Hull, s/n – Campus do Pici – Bloco 942-A
60455-760 – Fortaleza – CE – Brasil

{joarilima, manuelsilva, paulo.rego, dgomes}@great.ufc.br

Abstract. *In the mobile cloud computing paradigm, offloading techniques are used to augment computation and power capacities of mobile devices as well as to reduce the execution time of tasks. In this paper, we propose a data offloading mechanism that selects and migrates files to a local infrastructure (cloudlet), assisting computation offloading frameworks to reduce the amount of data sent over the network. The mechanism uses the application methods execution history, as well as the network condition, to create decision trees that help deciding when and which files used by these methods should be transferred. The experiments results indicate that our mechanism reduces the processing offloading time by up to 19.5%.*

Resumo. *No paradigma de mobile cloud computing, as técnicas de offloading permitem a extensão das capacidades energética e computacional de dispositivos móveis, bem como a redução do tempo de execução de procedimentos. Neste artigo, propomos um mecanismo de offloading de dados que seleciona e migra arquivos para uma infraestrutura local (cloudlet) auxiliando os frameworks de offloading de processamento a reduzirem a quantidade de dados enviados pela rede. O mecanismo utiliza-se do histórico de execuções dos métodos dos aplicativos, assim como das condições da rede, para criar árvores de decisão que auxiliam na deliberação de quando e quais arquivos utilizados por estes métodos devem ser transferidos. Os resultados dos experimentos indicam que a utilização do mecanismo proposto reduz o tempo do offloading de processamento em até 19,5%.*

1. Introdução

Uma das características que contribuiram para a popularidade dos dispositivos móveis, em especial os *smartphones* é o suporte a um grande número de aplicativos, tais como jogos, comércio eletrônico e redes sociais *on-line*. No entanto, apesar destes aplicativos estarem cada vez mais exigentes em termos de processamento, armazenamento e banda, as melhorias no *hardware* dos dispositivos móveis e na vida útil das suas baterias não têm acompanhado a crescente demanda por recursos computacionais [Khan et al. 2014]. Diante destes obstáculos, uma das soluções em perspectiva é migrar o processamento e o armazenamento dos dados para dispositivos remotos ou nuvens computacionais [Kemp et al. 2009].

Neste contexto, um dos benefícios providos pela computação móvel em nuvem (*Mobile Cloud Computing*, MCC) é o uso das nuvens, as quais disponibilizam serviços para a expansão das capacidades físico-computacionais dos dispositivos móveis [Schüring 2011, Dinh et al. 2013]. Através da MCC, os dispositivos móveis podem expandir seus recursos de processamento, memória, armazenamento e melhorar sua autonomia energética [Khan et al. 2014]. A migração das tarefas de processamento de um dispositivo móvel para uma nuvem computacional remota ou para uma infraestrutura computacional de alcance local é conhecida por *offloading* de processamento ou *cyber foraging* [Kumar et al. 2013].

Em [Lewis and Lago 2015] foi realizada uma revisão sistemática para identificar e classificar os trabalhos de *offloading* de processamento e de dados sob o ponto de vista arquitetural. Eles executaram uma *string* de busca em setembro de 2013 na base *Google Scholar*, a qual retornou 430 resultados[1]. Nós executamos a mesma *string* de busca em novembro de 2016, cujo retorno foi de 2.770 resultados, i.e. um aumento de mais de 540% com relação ao artigo de [Lewis and Lago 2015]. Apesar da realização de uma revisão sistemática estar fora do escopo do trabalho aqui proposto, tal aumento indica interesse no tema e presença de problemas ainda em aberto.

Para que o *offloading* economize energia ou reduza tempo de execução das tarefas, de acordo com [Kumar et al. 2013], é necessário avaliar as características instantâneas da rede em que o dispositivo móvel está inserido, o poder de processamento (MFLOPS) dos dispositivos envolvidos na operação e a quantidade de dados em potencial de migração. Um sistema de *offloading* eficiente é aquele capaz de inferir onde uma tarefa computacional deve ser executada (se local ou remotamente) de modo que o dispositivo móvel economize seus recursos computacionais [Flores et al. 2015].

Neste sentido, uma das possíveis alternativas para diminuir a quantidade de dados transferidos durante o *offloading* de processamento é o seu emprego conjunto com mecanismos de *offloading* de dados [Lewis and Lago 2015]. Através destes mecanismos, os dispositivos móveis migram seu dados para uma infraestrutura computacional local (*cloudlet*) a qual funciona como um intermediário entre o dispositivo móvel e a nuvem remota. Entretanto, notamos que os atuais *frameworks* de *offloading* de dados normalmente não filtram quais dados devem ser migrados para uma *cloudlet*, i.e. apesar deles decidirem *quando* migrar os dados, não avaliam *o que* devem migrar [Lewis and Lago 2015].

Neste artigo, propomos um mecanismo de *offloading* de dados com tomada de decisão que leva em consideração as condições momentâneas da rede e o histórico de execuções dos métodos de aplicativos para selecionar quais arquivos do dispositivo móvel devem ser migrados para uma *cloudlet*. O mecanismo se utiliza de árvores de decisão para escolha de quais arquivos e de quando estes arquivos devem ser migrados. Os arquivos selecionados para a migração variam dependendo dos aplicativos utilizados e das diferentes condições de rede.

2. Trabalhos Relacionados

Atualmente, encontra-se na literatura uma variedade de estudos abordando *offloading*, seja ele de processamento ou de dados [Ali et al. 2016], [Silva Jr. et al. 2015], [Enzai

[1] https://www.andrew.cmu.edu/user/gritter/slr-online-material.pdf

and Tang 2014], [Fernando et al. 2013], [Kumar et al. 2013], [Chun et al. 2011], [Cuervo et al. 2010] e [Satyanarayanan et al. 2009].

Em [Kumar et al. 2013] foi realizado um estudo do impacto das tarefas de *offload* no desempenho global do sistema. Foi apresentado um modelo analítico que propõe responder *quando* que a realização do *offloading* de processamento seria realmente viável levando em conta uma visão global do dispositivo móvel e do processo de *offloading*. As equações propostas em seu modelo assumem que mesmo no *offload* voltado a melhoria do processamento, uma quantidade significante de dados deve ser transferida do dispositivo móvel para o dispositivo remoto. O artigo aqui proposto utilizou-se da mesma ideia geral de [Kumar et al. 2013], onde se aponta a possibilidade de realizar previamente o *offloading* dos dados envolvidos nas tarefas de processamento remoto, diminuindo assim esta sobrecarga.

Em [Hung et al. 2012] foi proposto um *framework* para execução de aplicações móveis em um ambiente virtual na nuvem. Um agente instalado no dispositivo local e no ambiente virtualizado se responsabiliza por orquestrar a sincronização dos dados necessários para correta execução da aplicação na nuvem. Para diminuir a sobrecarga na transferência dos dados os agentes obtêm de forma prévia informações apenas do estado do dispositivo móvel e do dispositivo virtualizado. Estas informações são então utilizadas para elencar quais dados devem efetivamente passar pelo *offloading*.

Em [Gomes et al. 2016] foi proposto um serviço de *offloading* de dados com suporte à privacidade a fim de realizar a disseminação de dados contextuais entre os dispositivos móveis e o ambiente da nuvem. Esse serviço se utiliza do conceito de *cloudlets* apresentado em [Satyanarayanan et al. 2009] para permitir essa disseminação. O serviço realiza *offloading* de processamento e faz uma tomada de decisão pela execução ou não na nuvem a fim de melhorar o desempenho da aplicação que é realizada com uso de métricas de rede. Ele se utiliza de informações como a qualidade de conexão entre o dispositivo móvel e o servidor remoto, ou seja, a latência da conexão para mensurar a viabilidade das tarefas de *offloading*.

Nos trabalhos aqui citados, a problemática da sobrecarga adicional gerada pela transferência dos dados necessários durante as tarefas de *offloading* de processamento foi tratada utilizando a ideia de disponibilizar tais dados de forma prévia, a fim de obter um ganho de desempenho global. O mecanismo que propomos possui dois diferenciais: (*i*) Adicionou-se uma tomada de decisão antes do *offloading* de dados, a qual é feita tomando por base uma árvore de decisão. Esta árvore é criada na infraestrutura local (*cloudlet*) a partir do histórico de execuções local (dispositivo móvel) e remota (*cloudlet*); (*ii*) o *offloading* de dados é utilizado em conjunto com o *offloading* de processamento com intuito de melhorar o seu desempenho global. Este processo, alinhado ao mecanismo de tomada de decisão adotado, são os diferenciais deste artigo.

3. O Mecanismo Proposto

Neste trabalho, o mecanismo de *offloading* de dados objetiva auxiliar no *offloading* de processamento. Para isto, é realizada a persistência de arquivos que são utilizados como parâmetros na chamada de métodos dos aplicativos móveis. Nesta fase de desenvolvimento do mecanismo não foi implementada a comunicação com a nuvem, se restringindo apenas aos dispositivos que estão na rede local.

O desenvolvimento do mecanismo resultou em 2 (dois) aplicativos. O primeiro, chamado de *FileOffApi*, deve ser executado em um dispositivo móvel, e o segundo, chamado de *FileOffCloudlet*, deve ser executado em um *cloudlet* (como um *desktop*). Para que o mecanismo desempenhe sua função, os dois aplicativos precisam se comunicar periodicamente e a infraestrutura de comunicação utilizada foi a conexão Wi-Fi entre os dois dispositivos, neste caso, entre o dispositivo móvel e a infraestrutura local.

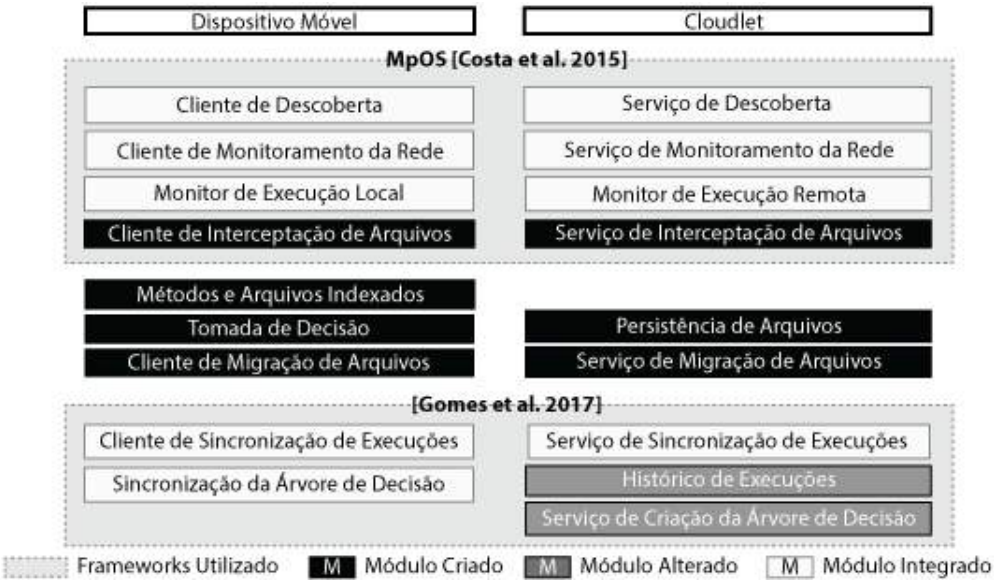

Figura 1. Arquitetura do mecanismo proposto. Módulos do dispositivo móvel e da *cloudlet*.

O mecanismo de *offloading* de dados precisa estar integrado a um *framework* que faça *offloading* de processamento para que a estrutura como um todo entre em funcionamento. Ele foi organizado de forma que, é necessária a integração com o código fonte do *framework* para *offloading* de processamento que se deseja auxiliar. Utilizamos o *framework* MpOS [Costa et al. 2015] na integração, pois, além de prover a funcionalidade de *offloading* de processamento, este possui os módulos que fazem o monitoramento da rede e o processo de descoberta de serviço que é necessário para o mecanismo. A utilização do MpOS é representada na Figura 1 pelo retângulo com linhas tracejadas.

Para o funcionamento do mecanismo, além da implementação do código responsável pelo *offloading* dos dados, foi necessário o desenvolvimento ou reuso de funcionalidades existentes em outros *frameworks*. Além do uso do *framework* MpOS [Costa et al. 2015], foram utilizados os trabalhos [Rego et al. 2017] e [Gomes et al. 2017] para a criação da árvore de decisão e sua sincronização com o dispositivo móvel. A Figura 1 exibe a arquitetura, divida em módulos, utilizada para a integração dessas funcionalidades.

Na Figura 1 os módulos desenvolvidos especificamente para este trabalho estão na cor preta. Os módulos que foram alterados (cor cinza) e os legados (cor branca) foram retirados de *frameworks* já existentes, conforme foi apresentado. O funcionamento dos módulos são explicados em detalhes a seguir.

O funcionamento do mecanismo começa com os módulos de descoberta que são

responsáveis pela busca e registro em um serviço de *offloading* de dados que está em execução em uma rede local. Em seguida, os módulos de monitoramento trocam dados para aferirem as condições da rede: *Round Trip Time* (RTT), taxa de *Upload* e taxa de *Download*. A aferição desses dados é feita periodicamente a cada 60 segundos e seus valores são armazenados e compartilhados entre o dispositivo móvel e a *cloudlet*.

Os arquivos envolvidos na operação de *offloading* são identificados através da função *hash* criptográfica SHA1. Para cada arquivo é gerada sua função *hash* utilizando como entrada os bytes que compõem o próprio arquivo.

O módulo **Métodos e Arquivos Indexados** é responsável por listar todos os arquivos, junto com suas funções *hash*, que estão em um diretório definido pelo usuário do mecanismo. Esse módulo mantém duas listas que relacionam os arquivos que estão apenas no dispositivo móvel e os arquivos que já foram transferidos para a *cloudlet*. Outra função desse módulo é manter a listagem dos métodos que foram invocados. O módulo **Cliente de Migração de Dados** faz a conexão com a *cloudlet* através do módulo **Serviço de Migração de Dados** para a transferência dos arquivos. Além de enviar os arquivos, o módulo de transferência também tem a função de sincronizar a lista de arquivos que foram enviados para a *cloudlet*. Na *cloudlet* há o módulo **Persistência de Arquivos** que armazena os arquivos recebidos.

É necessário inspecionar os parâmetros passados para um método e verificar se os arquivos presentes nos parâmetros estão disponíveis na *cloudlet*. Caso essa situação seja verdadeira, o arquivo que seria transferido junto com o método é substituído por um identificador (função *hash*). Quando a invocação do método, junto com os parâmetros, chega à *cloudlet*, o identificador do arquivo é substituído pelo arquivo e a execução do método prossegue normalmente. O módulo **Cliente de Interceptação de Arquivos** inspeciona os parâmetros dos métodos e verifica no módulo **Métodos e Arquivos Indexados** se o arquivo passado nos parâmetros já está na *cloudlet*. O módulo **Serviço de Interceptação de Arquivos** inspeciona os parâmetros do método que chegou na *cloudlet* e substitui o identificador pelo arquivo depois de solicitar ao módulo **Persistência de Arquivos** o arquivo correspondente ao identificador informado.

Os módulos **Monitor de Execução Local** e **Monitor de Execução Remota** são responsáveis pelo monitoramento do tempo de execução de métodos que são executados no dispositivo móvel e na *cloudlet*, respectivamente. Eles calculam quanto tempo decorreu desde a chamada de um método até seu término. O módulo **Monitor de Execução Remota**, além de fazer a medição do tempo de execução, junta esses dados com as condições da rede no momento da medição. As condições da rede são obtidas através do módulo **Serviço de Monitoramento da Rede** que detalhamos anteriormente.

As informações obtidas pelos módulos de monitoramento são persistidas em uma base de dados que armazena o histórico de execuções dos métodos. Essas informações são utilizadas para o treinamento da árvore. Para que essas informações cheguem à base de dados, foram utilizados dois módulos do trabalho [Gomes et al. 2017]: o **Cliente de Sincronização de Execuções** e o **Serviço de Sincronização de Execuções**. O primeiro módulo obtém os tempos de execuções dos métodos executados no dispositivo móvel e se comunica com o segundo módulo para repassar as novas informações. O módulo **Serviço de Sincronização de Execuções** recebe as informações do módulo **Cliente de**

Sincronização de Execuções e também é responsável por receber os tempos de execuções dos métodos executados na *cloudlet*. A persistência na base de dados é feita pelo módulo **Serviço de Sincronização de Execuções**.

3.1. Tomada de decisão e criação da árvore

O processo de criação da árvore de decisão foi introduzido em [Rego et al. 2017] e adaptado neste trabalho. O **Serviço de Criação da Árvore de Decisão** utiliza o tempo de execução dos métodos no dispositivo móvel e o tempo de execução na *cloudlet* (capturado pelo **Serviço de Sincronização de Execuções**) para classificar cada instância do histórico de execuções de *offloading* entre as classes *Local* e *Cloudlet* (Figura 2 (I)). Tais classes indicam que, em condição de rede similar, o método deve ser executado, respectivamente, no dispositivo móvel ou na *Cloudlet*.

Uma vez que as instâncias são classificadas, os dados formam o conjunto de treinamento (típico de algoritmos de aprendizagem supervisionados) que é utilizado para gerar a árvore de decisão. Como o processo de aprendizagem de árvores de decisão depende do conjunto de treinamento, o número de instâncias do conjunto pode afetar o poder de generalização e precisão da árvore de decisão. Assim, antes de criar a árvore de decisão, o algoritmo analisa o histórico de execuções e considera diferentes condições de rede (Figura 2 (II)) para criar novas instâncias para o conjunto de treinamento (e.g., considerando redes com a metade da taxa de download/upload).

Dentre os diversos algoritmos para construção de árvores de decisão (e.g. ID3, C4.5 e CART), a seleção do atributo mais útil para classificar as instâncias é um fator crítico de escolha. Neste artigo utilizamos as árvores C4.5, que baseiam-se na *information gain*, uma propriedade estatística da teoria da informação, a qual permite a seleção do melhor atributo para dividir o conjunto de treinamento [Quinlan 1993]. C4.5 foi escolhida porque seu algoritmo lida tanto com atributos categóricos (ordinais ou não-ordinais), como com atributos contínuos, além de gerar árvores não necessariamente binárias (diferente do CART) e alcançar ótimos resultados em problemas de classificação. Tais características, fazem do C4.5 um dos algoritmos de árvore de decisão mais utilizados na literatura [Wu et al. 2008].

Uma vez que o conjunto de treinamento está pronto, uma árvore C4.5 é criada utilizando o algoritmo J48 da biblioteca Java Weka[2] (Figura 2 (III)). A Figura 2 apresenta um exemplo de árvore de decisão de *offloading* para a aplicação "Dummy". Neste exemplo, se a taxa de upload for igual a 300 Kbps, o método `m1` deve ser executado no dispositivo móvel. Além disso, independente das métricas, o método `m2` deve ser executado na *Cloudlet*.

O módulo **Serviço de Criação da Árvore de Decisão** periodicamente verifica na base de dados se há novas medições. Nem sempre a inclusão de novas medições vai mudar a estrutura da árvore de decisão, no entanto, quando ocorre esta mudança é preciso notificar o módulo **Sincronização da Árvore de Decisão** e junto com a notificação enviar a nova árvore de decisão.

A tomada de decisão ocorre no módulo **Tomada de Decisão**. Inicialmente, faz-se acesso ao módulo **Métodos e Arquivos Indexados** em busca dos arquivos que estão ape-

[2]Weka website: `http://www.cs.waikato.ac.nz/ml/weka`.

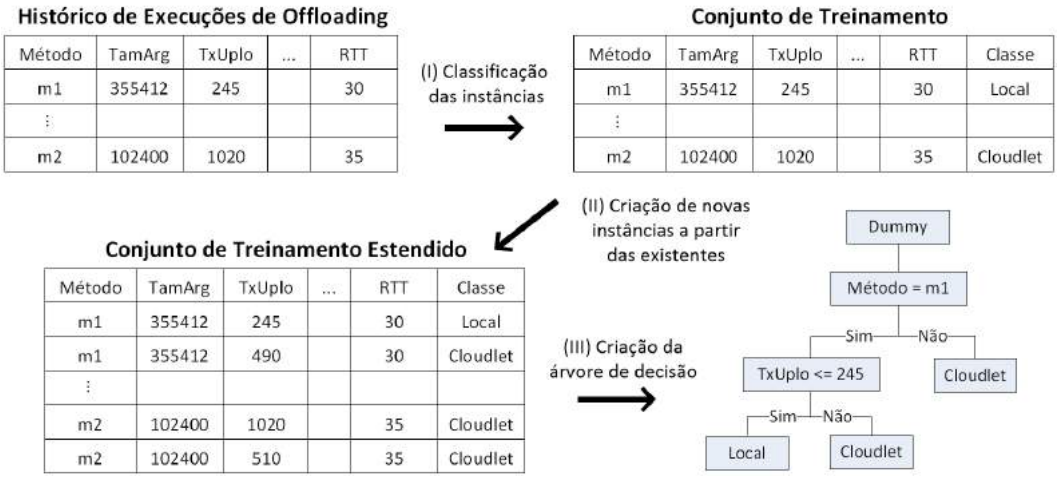

Figura 2. Exemplo de árvore de decisão de *offloading*.

nas no dispositivo móvel e os métodos que podem ser executados (*offloaded*) na *cloudlet*. Em seguida, o módulo **Sincronização da Árvore de Decisão** é acessado e é solicitada a última árvore de decisão disponível. O próximo passo é solicitar ao módulo **Cliente de Monitoramento da Rede** as condições da rede. Em posse desses dados, são passadas para árvore de decisão as condições da rede junto com o tamanho de cada arquivo no dispositivo móvel e os métodos que podem executados na *cloudlet*. Com essas informações a árvore é capaz de inferir se um arquivo deve ou não ser migrado para a *cloudlet*.

A tomada de decisão resulta na listagem dos arquivos que devem ser migrados para a *cloudlet*. Essa listagem é passada para o módulo **Cliente de Migração de Dados** que faz o *offloading* dos arquivos, como explicado anteriormente.

4. Material e Métodos

Os filtros de imagem utilizados pertencem à biblioteca de código aberto PhotoFilter[3], a qual possui 30 filtros. Essa biblioteca foi utilizada nos experimentos por dar suporte a versões antigas do sistema operacional Android (versão 2.3 e superiores) e pela disponibilidade do código fonte online. Para nossos experimentos foram usados somente os filtros *Inverter*, *Snow*, *Emboss* e *Engrave* devido aos seus tempos de execução em comparação aos outros da biblioteca. Entre os 30 filtros disponíveis, os filtros *Inverter* e *Snow* foram os que apresentaram menor tempo para serem executados, já os filtros *Emboss* e *Engrave* foram os que demandaram mais tempo.

Além de utilizar os filtros de imagem, o aplicativo é capaz de migrar as fotos que estão no dispositivo móvel para a *cloudlet* e de executar os filtros tanto no dispositivo móvel quanto na *cloudlet* com a API do *framework* MpOS [Costa et al. 2015]. Os filtros do aplicativo necessitam que arquivos de imagem sejam passados como parâmetros. Para verificar como o tempo de execução dos filtros variavam de acordo com o tamanho dos arquivos, foram utilizados 10 arquivos com tamanhos distintos.

Para que o mecanismo proposto fosse capaz de decidir quais arquivos deveriam ser migrados, foram coletados os tempos de execução dos filtros. Os filtros foram executados

[3]Código Fonte PhotoFilter: https://github.com/mukeshsolanki/photofilter

em um dispositivo móvel e também em uma *cloudlet* (detalhes na Tabela 1). Os tempos de execução foram salvos em um banco de dados com o histórico de todas as execuções.

Tabela 1. Detalhes do ambiente de testes.

DISPOSITIVOS	HARDWARE			SOFTWARE
Dispositivo Móvel Smartphone Samsung Galaxy S6 (SM-G920i)	**Processador** Quad-core 1.5 GHz Cortex-A53 + Quad-core 2.1 GHz Cortex-A57	**RAM** 3GB	**WIFI** 802.11 a/b/g/n/ac	Android 5.0 TouchWiz UI Lollipop
Cloudlet Ultrabook Samsung (NP53U3B-AD1BR)	**Processador** Intel® Core™ i5-2467M	**RAM** 8GB	**WIFI** 802.11 b/g/n	Windows® 7 Home Premium Original 64 Bits
Roteador DLink DSL-2740E	**WIFI** 802.11 b/g			

Nas subseções a seguir, são descritos os estudos de caso utilizados na validação do mecanismo.

4.1. Experimento #1: Ganho de tempo com o *offloading* de dados

O primeiro experimento consistiu na implementação do *offloading* de dados e seu teste em um ambiente real. Para o teste foram utilizados os dispositivos descritos na Tabela 1. O objetivo deste experimento é verificar se a operação de *offloading* de dados resulta em ganho de tempo para o usuário quando é feito o *offloading* de processamento dos filtros de imagem.

Tabela 2. Métrica e fatores utilizados no experimento de tempo de execução de métodos na *cloudlet*

MÉTRICA		DETALHES
Tempo do *Offloading* de Processamento		Tempo em milissegundos para o *offloading* de processamento dos métodos de filtro de imagem
FATORES	**NÍVEIS**	**DETALHES**
Métodos de Filtro de Imagem	4	Filtro ApplySnow Filtro Inverter Filtro Emboss Filtro Engrave
Tamanhos de Arquivos	10	25KB 90KB 192KB 342KB 524KB 870KB 1,2MB 1,5MB 1,9MB 4,2MB
Largura Máxima de Banda	2	4Mbps 8Mbps
Offloading de Dados	2	Ativado Desativado
15 Repetições	**x**	**160 testes**

As etapas do Experimento #1 são ilustradas na Figura 3 e iniciam na etapa 1 onde os arquivos de imagem que estão no dispositivo móvel são migrados para a *cloudlet*. Em seguida, na etapa 2, o dispositivo móvel faz o *offloading* de processamento dos filtros de imagem e passa como parâmetros as referências das imagens que foram migradas na etapa 1. Por fim, na etapa 3, o dispositivo móvel faz novamente o *offloading* de processamento dos filtros de imagem, porém, dessa vez, são passadas as imagens ao invés das referências. Para verificar se houve ganho de tempo, foi comparado o tempo gasto para fazer o *offloading* de processamento quando o mecanismo de *offloading* de dados estava ativado (Figura 3, etapa 2) com o tempo gasto quando o mecanismo estava desativado (Figura 3, etapa 3).

Figura 3. Etapas do Experimento #1.

A métrica e fatores utilizados para a execução do Experimento #1 estão descritos na Tabela 2. Além de serem utilizados 4 métodos de filtro de imagem e 10 arquivos de imagem de tamanhos distintos, foram utilizadas duas larguras de banda para verificar o quanto a vazão da rede impactou no tempo para se realizar o *offloading* de processamento. Por último, variou-se a ativação do mecanismo proposto para verificar em que situação o *offloading* de processamento se beneficiou do mecanismo e apresentou menor tempo. Foram realizados 160 testes e para cada teste foram feitas 15 repetições.

4.2. Experimento #2: Tempo de execução no dispositivo móvel e na *cloudlet*

O segundo experimento tem o objetivo de comparar os tempos obtidos no Experimento #1 (execução na *cloudlet*) com o tempo de execução dos mesmos métodos no dispositivo móvel. Como a tomada de decisão do mecanismo depende do histórico de execuções tanto no dispositivo móvel quanto na *cloudlet*, então, com este experimento, foi possível analisar quando o *offloading* de dados seria benéfico. A métrica e os fatores utilizados no experimento estão descritos na Tabela 3.

Tabela 3. Métrica e fatores utilizados no experimento de tempo de execução de métodos no dispositivo móvel.

MÉTRICA		DETALHES
Tempo de Execução no Dispositivo Móvel		Tempo em milissegundos para a execução dos métodos de filtro de imagem
FATORES	**NÍVEIS**	**DETALHES**
Métodos de Filtro de Imagem	4	Filtro ApplySnow Filtro Inverter Filtro Emboss Filtro Engrave
Tamanhos de Arquivos	10	25KB 90KB 192KB 342KB 524KB 870KB 1,2MB 1,5MB 1,9MB 4,2MB
30 Repetições	**x**	**40 testes**

Ao comparar o Experimento #1 com o Experimento #2, é possível observar em que situações a execução dos filtros no dispositivo móvel é mais rápida do que realizar o *offloading* de processamento e, consequentemente, em que circunstâncias não é vantajoso fazer o *offloading* dos dados.

4.3. Experimento #3: Seleção dos arquivos utilizando a árvore de decisão

O terceiro experimento utiliza os dados gerados nos experimentos anteriores para criar a árvore de decisão. O objetivo deste experimento é verificar se o mecanismo de tomada de decisão seleciona corretamente que arquivos devem ser migrados para a *cloudlet*. Os fatores utilizados foram os 4 filtros de imagem e os 10 tamanhos distintos de arquivos utilizados nos experimentos anteriores. A migração ou não dos arquivos é a métrica do experimento.

5. Resultados e Discussões

s resultados do Experimento #1 foram compilados na Tabela 4. Os dados exibidos na Tabela 4 são as médias dos tempos gastos em milissegundos para o *offloading* de processamento dos quatro filtros em diferentes condições da rede, com o mecanismo de *offloading* de dados ativado ou desativado e com diferentes tamanhos de arquivos.

Tabela 4. Tempos de *offloading* de processamento (em milissegundos) com o mecanismo de *offloading* de dados ativado e desativado.

Cloudlet			Tamanho dos Arquivos									
Filtro	Banda	Offloading de Dados	25KB	90KB	192KB	342KB	524KB	870KB	1,2MB	1,5MB	1,9MB	4,2MB
Inverter	4 Mbps	Desativado	296	474	1272	3164	4168	9033	12307	17720	20799	45158
	4 Mbps	Ativado	**416**	**673**	1232	2524	3816	7852	9174	12363	14597	32311
	8 Mbps	Desativado	176	458	1017	1808	2963	4648	6515	10121	11508	28696
	8 Mbps	Ativado	**381**	**620**	855	1765	2586	3676	6206	8392	9853	22660
Emboss	4 Mbps	Desativado	386	842	2056	4256	7176	12800	18701	23522	26865	64780
	4 Mbps	Ativado	**550**	**966**	2012	3498	5703	9975	15281	19356	22285	51985
	8 Mbps	Desativado	314	823	1426	2748	4361	7236	10418	15019	18868	39425
	8 Mbps	Ativado	**565**	**970**	1421	2621	4082	6452	8812	12857	16256	37542
Engrave	4 Mbps	Desativado	386	753	1806	4314	5896	10585	16674	20305	25092	57806
	4 Mbps	Ativado	**560**	**905**	1678	3297	5363	9775	13111	18498	21037	44678
	8 Mbps	Desativado	325	813	1431	2699	4544	6875	10628	13144	17045	39034
	8 Mbps	Ativado	**590**	**944**	1364	2625	3833	5653	9557	11619	14512	32901
Snow	4 Mbps	Desativado	178	411	1033	2236	4218	7071	11450	14297	15852	37499
	4 Mbps	Ativado	**381**	**539**	934	1962	3203	5039	6745	9879	11119	27168
	8 Mbps	Desativado	127	381	725	1471	2193	3730	5117	7795	9655	23398
	8 Mbps	Ativado	**399**	**532**	723	1271	2125	2642	4074	4884	6995	16280

Nos testes realizados, quando o *offloading* de dados estava ativado, os arquivos utilizados nos filtros não precisavam ser enviados pela rede a cada requisição já que era feita a persistência na *cloudlet*. No início dos testes esperava-se que o tempo para o *offloading* de processamento dos filtros fosse menor quando o mecanismo estivesse ativado pelo motivo da persistência dos arquivos. No entanto, nos testes foi possível observar que há situações em que fazer a persistência de arquivos não é vantajoso (tempos em vermelho na Tabela 4).

Os resultados do Experimento #1 mostraram que a migração dos arquivos economiza tempo em 80% dos casos. No entanto, podemos observar nas colunas 25KB e 90KB da Tabela 4 que o *offloading* de processamento dos filtros de imagem é mais lento quando o mecanismo estava ativado (tempos em vermelho). O funcionamento do mecanismo introduz na operação de *offloading* de processamento uma sobrecarga para realizar o *hash* dos arquivos e em seguida de trocar o *hash* pelo arquivo correspondente na *cloudlet*. O tempo necessário para realizar a função *hash* impactou no tempo de *offloading* de processamento dos filtros com arquivos de tamanho 25KB e 90KB, e assim, a operação de *offloading* de dados se tornou desvantajosa para estes testes.

Já para as execuções com tamanho de arquivos maiores ou iguais a 192KB é vantajoso utilizar o *offloading* de dados. A Figura 4 exibe a diferença de tempos do *offloading* de processamento do filtro *Snow* quando o *offloading* de dados está ativado e desativado para duas larguras de banda distintas. É possível observar na Figura 4 que a utilização do mecanismo se torna mais vantajosa de acordo com o aumento do tamanho dos arquivos. Quando comparamos os tempos de execução com e sem o mecanismo, em média, o tempo ganho com a migração dos dados foi de 19,5%.

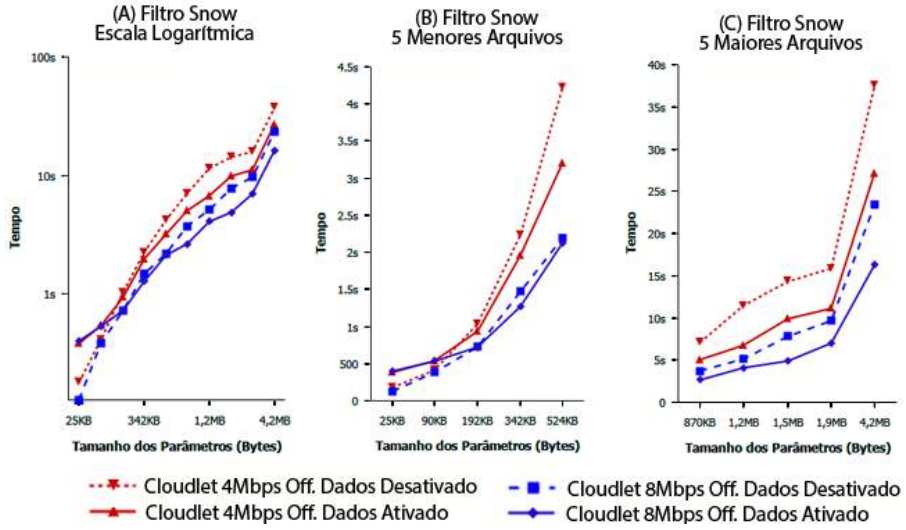

Figura 4. Comparativo do tempo de *offloading* de processamento do filtro *Snow* com o *offloading* de dados ativado e desativado.

Apesar do ganho médio observado no Experimento #1, foi necessário comparar o tempo do *offloading* de processamento com o tempo de execução no dispositivo móvel para averiguar onde é mais vantajoso realizar a execução dos filtros de imagem. Essa comparação foi feita no Experimento #2 e os dados obtidos foram compilados na Tabela 5.

Os resultados do Experimento #2 mostram que o *offloading* de processamento dos filtros consome menos tempo do que a execução no dispositivo móvel em 40% dos casos (tempos na cor verde). No entanto, quando analisamos a Tabela 5 percebemos que o filtro *Inverter* e *Snow* são executados mais rapidamente no dispositivo móvel do que na *cloudlet*. Nessa situação, assumimos que não existe ganho em fazer a migração dos arquivos quando os filtros são executados mais rapidamente no dispositivo móvel. Já que não haverá execução na *cloudlet* para essas situações, então não há necessidade de migrar arquivos para essas operações.

Tabela 5. Comparação dos tempos de *offloading* de processamento (em milissegundos) com a execução no dispositivo móvel.

Execução		Tamanho dos Arquivos									
Filtro	Dispositivo	25KB	90KB	192KB	342KB	524KB	870KB	1,2MB	1,5MB	1,9MB	4,2MB
Inverter	Cloudlet (4Mbps Off. Dados Ativado)	416	673	1232	2524	3816	7852	9174	12363	14597	32311
	Cloudlet (8Mbps Off. Dados Ativado)	381	620	855	1765	2586	3676	6206	8392	9853	22660
	Dispositivo Móvel	**47**	**202**	**528**	**1063**	**1608**	**3615**	**5676**	**6703**	**7552**	**18012**
Emboss	Cloudlet (4Mbps Off. Dados Ativado)	550	966	2012	3498	5703	9975	15281	19356	22285	51985
	Cloudlet (8Mbps Off. Dados Ativado)	565	970	**1421**	**2621**	**4082**	**6452**	**8812**	**12857**	**16256**	**37542**
	Dispositivo Móvel	**201**	**840**	2192	4290	6603	14802	23631	28209	32615	63758
Engrave	Cloudlet (4Mbps Off. Dados Ativado)	560	905	1678	3297	5363	9775	13111	18498	21037	44678
	Cloudlet (8Mbps Off. Dados Ativado)	590	944	**1364**	**2625**	**3833**	**5653**	**9557**	**11619**	**14512**	**32901**
	Dispositivo Móvel	**205**	**849**	2218	4410	6734	15050	24138	28659	32688	72077
Snow	Cloudlet (4Mbps Off. Dados Ativado)	381	539	934	1962	3203	5039	6745	9879	11119	27168
	Cloudlet (8Mbps Off. Dados Ativado)	399	532	723	1271	2125	2642	4074	4884	6995	16280
	Dispositivo Móvel	**6**	**29**	**78**	**156**	**235**	**549**	**870**	**1040**	**1186**	**2683**

Na Tabela 5, podemos observar que a execução na *cloudlet* (com o *offloading* de dados ativado) depende do tamanho do arquivo e do filtro selecionado para que seja vantajosa. Para os filtros *Emboss* e *Engrave* só devem ser migrados os arquivos de tamanho

maiores ou iguais a 192KB. O tempos na cor vermelha indicam que para dado filtro e para dado tamanho de arquivo não vale a pena realizar a sua migração para a *cloudlet*. Já os tempos na cor verde indicam que para esses filtros e esses tamanhos de arquivos a migração contribui para o ganho de tempo.

Pode-se observar nos dois primeiros experimentos que a migração de todos os arquivos não é vantajosa, pois existem situações onde a execução dos filtros no dispositivo móvel é mais rápida ou a própria sobrecarga do mecanismo torna o *offloading* de processamento mais lento. É nesse cenário que a seleção dos arquivos se faz necessária. Assim, a fim averiguar a eficácia da tomada de decisão do mecanismo, foi gerada uma árvore com os dados extraídos dos Experimentos #1 e #2 (que pode ser observada na Figura 5).

Pelos resultados dos dois experimentos, esperava-se que o mecanismo fizesse a seleção de arquivos maiores ou iguais a 192KB e apenas para os filtros *Emboss* e *Engrave*. A Tabela 6 apresenta o resultado do Experimento #3, onde pode-se perceber que a seleção foi feita de forma esperada.

Tabela 6. Arquivos selecionados pelo mecanismo de tomada de decisão.

	Tamanho dos Arquivos									
Filtro	**25KB**	**90KB**	**192KB**	**342KB**	**524KB**	**870KB**	**1,2MB**	**1,5MB**	**1,9MB**	**4,2MB**
Inverter	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO
Emboss	NÃO	NÃO	SIM	SIM	SIM	SIM	SIM	SIM	SIM	SIM
Engrave	NÃO	NÃO	SIM	SIM	SIM	SIM	SIM	SIM	SIM	SIM
Snow	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO

Os seguintes itens podem ser caracterizados como as principais ameaças à validação dos resultados deste trabalho: *(i)* Existe a necessidade de ampliar o ambiente de testes com uma maior variação de configurações e quantidade de dispositivos móveis; *(ii)* Para fins de redução no escopo, o experimento utilizou aplicativos que realizam operações em imagens. Ampliar os testes com outros tipos de aplicativos proporcionaria uma visão mais precisa dos possíveis ganhos ao utilizar o método proposto.

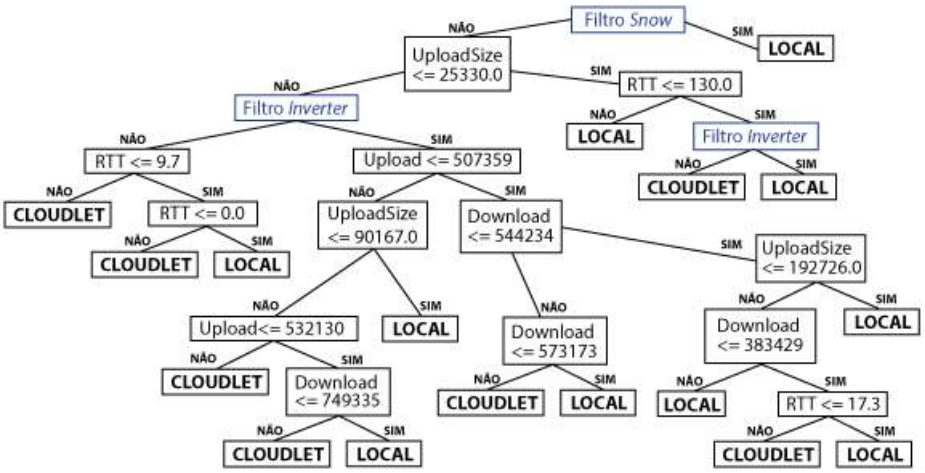

Figura 5. Árvore gerada a partir do histórico de execuções dos filtros de imagem no dispositivo móvel e na *cloudlet*

6. Conclusão

Neste artigo, propomos um mecanismo de auxílio a *frameworks* de *offloading* de processamento na persistência dos arquivos utilizados como parâmetros na invocação de métodos. De acordo com os resultados obtidos, o mecanismo diminuiu o tempo de *offloading* em 19,5% (em média). O mecanismo proposto leva em conta a tomada de decisão acerca de quais arquivos devem ser migrados. De acordo com os resultados, observamos que arquivos com tamanho inferiores a 192 KB não devem ser migrados pois: *(i)* o *offloading* de processamento é mais rápido quando o mecanismo de *offloading* de dados encontrava-se desativado, e *(ii)* os filtros de imagem que envolviam esses tamanhos de arquivos são executados mais rapidamente no dispositivo móvel.

A principal contribuição deste artigo é a utilização de técnicas de seleção e tomada de decisão no *offloading* de dados para dar suporte ao *offloading* de processamento, o que propicia redução no tempo de execução das tarefas executadas fora do dispositivo móvel.

Como perspectiva de estudos futuros, sugere-se a criação de perfis de acesso aos arquivos e perfis de funcionalidades utilizadas para que a tomada de decisão leve em consideração não só o histórico de todo o período de acesso, mas também o comportamento do usuário em intervalos de tempo mais curtos. Sugere-se ainda, com base nos resultados obtidos e nas ameaças à validação, a ampliação dos casos de uso e quantitativo de experimentos a fim de proporcionar uma visão mais ampla e precisa dos ganhos obtidos pelo uso do mecanismo e de sua aplicação em ambientes não controlados.

Agradecimentos

Agradecemos à CAPES (#1522784) e ao CNPq (#311878/2016-4).

Referências

Ali, F. A., Simoens, P., Verbelen, T., Demeester, P., and Dhoedt, B. (2016). Mobile device power models for energy efficient dynamic offloading at runtime. *Journal of Systems and Software*, 113:173–187.

Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., and Patti, A. (2011). Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM.

Costa, P. B., Rego, P. A. L., Rocha, L. S., Trinta, F. A. M., and de Souza, J. N. (2015). Mpos: A multiplatform offloading system. In *30th Annual ACM Symposium on Applied Computing*, SAC '15, page 577–584, New York, NY, USA. ACM.

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 *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. 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.

Enzai, N. I. M. and Tang, M. (2014). A taxonomy of computation offloading in mobile cloud computing. In *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*, pages 19–28. IEEE.

Fernando, N., Loke, S. W., and Rahayu, W. (2013). Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29:84–106.

Flores, H., Hui, P., Tarkoma, S., Li, Y., Srirama, S., and Buyya, R. (2015). Mobile code offloading: from concept to practice and beyond. *IEEE Communications Magazine*, 53(3):80–88.

Gomes, F. A. A., Rego, P. A. L., Rocha, L., de Souza, J. N., and Trinta, F. (2017). Caos: A context acquisition and offloading system. In *2017 IEEE 41th Annual Computer Software and Applications Conference (COMPSAC)*, volume 1.

Gomes, F. A. A., Viana, W., Rocha, L. S., and Trinta, F. (2016). A Contextual Data Offloading Service With Privacy Support. In *WebMedia*, Teresina-PI , Brazil. Sociedade Brasileira de Computação.

Hung, S.-h., Shih, C.-s., Shieh, J.-p., Lee, C.-p., and Huang, Y.-h. (2012). Executing mobile applications on the cloud : Framework and issues. *Computers and Mathematics with Applications*, 63(2):573–587.

Kemp, R., Palmer, N., Kielmann, T., Seinstra, F., Drost, N., Maassen, J., and Bal, H. (2009). eyedentify: Multimedia cyber foraging from a smartphone. In *Multimedia, 2009. ISM'09. 11th IEEE International Symposium on*, pages 392–399. IEEE.

Khan, A. R., Othman, M., Madani, S. A., and Khan, S. U. (2014). A survey of mobile cloud computing application models. *Communications Surveys & Tutorials, IEEE*, 16(1):393–413.

Kumar, K., Liu, J., Lu, Y.-H., and Bhargava, B. (2013). A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140.

Lewis, G. and Lago, P. (2015). Architectural tactics for cyber-foraging: Results of a systematic literature review. *Journal of Systems and Software*, 107:158–186.

Quinlan, J. R. (1993). C4. 5: Programming for machine learning. *Morgan Kauffmann*, page 38.

Rego, P. A. L., Cheong, E., Coutinho, E. F., Trinta, F. A., Hasan, M. Z., and de Souza, J. N. (2017). 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)*.

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.

Schüring, M. (2011). Mobile cloud computing–open issues and solutions. In *15thTwente Student Conference on IT, Enschede, The Netherlands*.

Silva Jr., L. S. d., Magalhães, D. M. V., and Gomes, D. G. (2015). Modelagem e simulação de offloading para computação móvel em nuvem. In *Anais / XXXV Congresso da Sociedade Brasileira de Computação*, pages 91–100, Porto Alegre. SBC.

Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Philip, S. Y., et al. (2008). Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37.