Uma ferramenta baseada em containers Docker para experimentação em Mobile Cloud Computing

Renan A. Barbosa[1], **Paulo A. L. Rego**[1], **Elvis M. G. Stancanelli**[1], **Marcio Maia**[1] **e José N. de Souza**[2]

[1]Curso de Redes de Computadores
Universidade Federal do Ceará (UFC) - Campus Quixadá

[2]Departamento de Computação
Universidade Federal do Ceará (UFC) - Campus do Pici

renan.alves@alu.ufc.br, {pauloalr,elvis.stancanelli,marcio,neuman}@ufc.br

Abstract. *Mobile Cloud Computing (MCC) is intended to improve the execution of applications on mobile devices through the processing power provided by cloud computing. Several MCC works analyze the influence of the network conditions onto the accomplishment of computation offloading; however, many of these works use single-device scenarios, which do not fit into a classic cloud computing environment, where multiple users interact simultaneously with a service. This work presents a tool that uses Docker containers to create MCC experimentation environments with multiple mobile devices and cloudlets. By using the proposed tool, developers and researchers will be able to create more realistic experimentation scenarios to test their applications.*

Resumo. *Mobile Cloud Computing (MCC) destina-se a melhorar a execução de aplicações em dispositivos móveis através do poder de processamento fornecido pela computação em nuvem. Diversos trabalhos da área analisam a influência das condições da rede junto à realização do offloading de processamento; porém, muitos se limitam a cenários com apenas um usuário, que não condizem com um ambiente clássico de computação em nuvem, em que múltiplos usuários interagem simultaneamente com um serviço e a demanda muda constantemente. Este trabalho apresenta uma ferramenta que utiliza containers Docker para criação de ambientes de experimentação MCC com múltiplos dispositivos móveis e cloudlets.*

1. Introdução

Apesar da enorme evolução desde os primeiros dispositivos computacionais móveis, pesquisadores e desenvolvedores ainda se deparam com muitas restrições quanto ao seu uso. De acordo com [Fernando et al. 2013], fatores como consumo energético e capacidade de processamento dificultam o desenvolvimento de aplicações mais complexas para dispositivos móveis. *Mobile Cloud Computing* (MCC) é uma junção da computação em nuvem e da computação móvel que surgiu com o intuito de superar tais obstáculos. Como exemplo, uma máquina virtual pode ser disponibilizada na nuvem, e seus recursos computacionais poderão ser devidamente aproveitados para a execução de uma aplicação, em partes ou mesmo em sua totalidade [Fernando et al. 2013] [Hoang et al. 2013].

Para isso, faz-se necessário que os dispositivos transmitam os dados a serem processados para a outra máquina. Trata-se do processo denominado *offloading de processamento*, pelo qual um dispositivo transfere carga de trabalho a outro dispositivo para que seja processada e os resultados do processamento devolvidos ao dispositivo original [Kumar et al. 2013].

MCC tem chamado atenção, tanto da indústria como da academia, por ser uma alternativa que reduz os custos de desenvolvimento e execução de aplicações móveis, bem como a atenção também dos usuários de dispositivos móveis por ser uma nova tecnologia que promove uma melhor experiência a um baixo custo para uma série de serviços móveis, tais como aplicações colaborativas e jogos [Hoang et al. 2013].

Diversos estudos, tais como [Kosta et al. 2012], [Rego et al. 2016] e [Gomes et al. 2017] avaliam o impacto do *offloading* no desempenho das aplicações. No entanto, os cenários de testes adotados não condizem com a realidade de um ambiente clássico de computação em nuvem, em que múltiplos usuários interagem simultaneamente com um serviço e a demanda muda constantemente.

Com vistas a essa limitação, o presente trabalho traz uma ferramenta para auxiliar desenvolvedores e pesquisadores de MCC na criação e execução de experimentos em cenários com múltiplos dispositivos móveis. Tal ferramenta é capaz de executar as seguintes operações:

- Criar diferentes cenários de execução;
- Permitir a criação de *cloudlets*;
- Permitir a criação de dispositivos móveis com diferentes configurações de rede;
- Permitir a execução automatizada de experimentos através de uma API.

A solução desenvolvida foca apenas na plataforma Android devido à sua grande abrangência em meio aos desenvolvedores e usuários, tanto que, no final de 2016, ele estava presente em aproximadamente 87% dos *smartphones* do mundo, de acordo com pesquisas da IDC[1].

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta alguns trabalhos relacionados; a Seção 3 introduz a ferramenta proposta; a Seção 4 descreve como será a demonstração e apresenta os endereços para acessar o código fonte, guias de instalação e utilização da ferramenta. Por fim, a Seção 5 conclui o trabalho e menciona algumas frentes de investigação para trabalhos futuros.

2. Trabalhos Relacionados

Nos últimos anos, vários *frameworks* e *middlewares* têm sido propostos para facilitar o desenvolvimento de aplicações com suporte ao *offloading* de processamento. Em [Kosta et al. 2012], os autores apresentam a ferramenta *ThinkAir*, um *framework* que visa simplificar o trabalho de desenvolvedores na tarefa de portar suas aplicações Android para a nuvem. Para isso, o *ThinkAir* virtualiza o sistema operacional Android utilizando uma versão modificada do sistema desenvolvido pelo projeto Android x86[2].

Em [Rego et al. 2016], é apresentado o MpOS (*Multiplatform Offloading System*), um *framework* para fazer *offloading* nas plataformas Android e Windows Phone. O MpOS

[1]IDC Smartphone Market Share Report: http://www.idc.com/promo/smartphone-market-share/os.

[2]`http://www.android-x86.org`

segue uma arquitetura cliente-servidor e os métodos candidatos ao *offloading* precisam ser marcados pelos desenvolvedores com a anotação *@Remotable*, assim, quando a aplicação for executar o método em questão, a API do MpOS reconhece que tal método pode ser executado fora do dispositivo móvel. O CAOS (*Context Acquisition and Offloading System*) [Gomes et al. 2017] é uma extensão do MpOS que inclui também o tratamento de dados contextuais, ao possibilitar o *offloading* dos dados, bem como o processamento deles fora do dispositivo.

Nenhum dos trabalhos mencionados realizou experimentos com múltiplos dispositivos. Tais soluções foram validadas com experimentos em que apenas um dispositivo móvel real é utilizado, e ele faz *offloading* para um servidor dedicado (geralmente usando uma rede Wi-Fi também dedicada). A ferramenta aqui proposta possibilitará que pesquisadores façam avaliações em cenários mais realistas.

3. Ferramenta proposta

A ferramenta proposta foi desenvolvida por meio da linguagem de programação Python e conta com três componentes principais: *CLI* (*Command Line Interface*), *Criador* e *Gerente*. Além deles, são também utilizadas as ferramentas: ADB (*Android Debugging Bridge*), responsável pela comunicação com os dispositivos Android; Docker, tecnologia de software responsável pela criação, gerenciamento e comunicação com os *containers*; e um banco de dados *SQLite*, que é utilizado para guardar todas as informações sobre *containers* e os cenários.

A Figura 1 mostra como estão organizados os principais componentes da ferramenta. Nela, o usuário utiliza uma *CLI* para se comunicar com o *Criador* e o *Gerente*. O *Criador* é responsável pela criação e destruição de cenários (conjunto de *containers*), sendo responsável por se comunicar com o banco de dados *SQLite*, com o *Gerente* e com o Docker. O *Gerente* é responsável pelos processos de listagem, validação de nomes, controle de execução de cenários e conexão com os *containers*. Ele se relaciona com o banco de dados *SQLite*, com o Docker e também com a ferramenta ADB. Por último, o usuário pode criar *scripts* em Python junto à API desenvolvida para executar métodos das aplicações instaladas nos *containers* Android.

A tecnologia de *containers* é usada na ferramenta para criar cenários com dispositivos clientes e servidores com base em imagens do Docker. Para criar os clientes, a imagem utilizada foi retirada do repositório de imagens Docker Hub[3], tendo sido alterada para facilitar a integração com a ferramenta (e.g., foram configuradas as ferramentas ADB, emulador Android e servidor VNC, bem como algumas regras de filtragem de pacotes e redirecionamento de portas). Para os servidores, foi utilizada uma imagem Ubuntu Server 16.10 com o Java versão 8.

Devido à sua simplicidade e serviços oferecidos (e.g., descoberta automática de *cloudlets* na mesma rede Wi-Fi, *deploy* automático de dependências e *offloading* de métodos) [Rego et al. 2016], o *framework* MpOS foi incorporado à ferramenta. Os *containers* clientes representam dispositivos móveis, possuindo um emulador Android instalado e as aplicações Android devem utilizar a API do MpOS para identificar os métodos candidatos à operação de *offloading*. Os *containers* servidores, que fazem o papel *cloudlets*, possuem os arquivos necessários para executar o servidor da solução *MpOS* (*MpOS*

[3] `https://hub.docker.com`

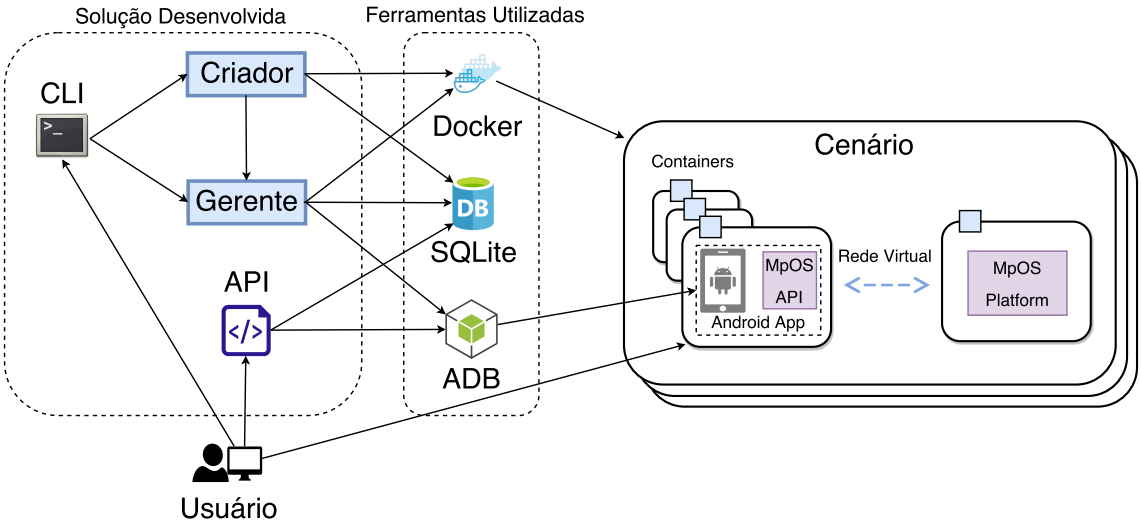
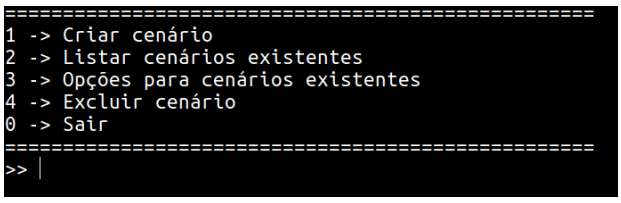


Figura 1. Visão geral dos componentes da ferramenta.

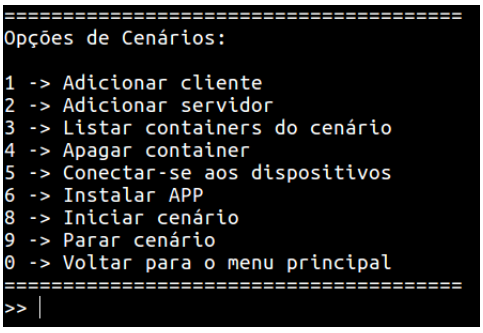
Platform). A comunicação entre Python e Docker é feita utilizando a API *Docker* para a versão 3 do Python.

Além da ferramenta acessível via *CLI*, foi desenvolvida uma API para que o usuário possa automatizar testes na sua aplicação. Utilizando a linguagem Python e a API, o desenvolver pode criar *scripts* que executam métodos específicos da sua aplicação em todos os dispositivos do cenário.

A Figura 2(a) demonstra a tela que o usuário visualiza quando executada a *CLI*, fornecendo acesso a quatro opções: (1) Criar cenário, (2) Listar cenários existentes, (3) Opções para cenários existentes e (4) Excluir cenário.



(a) Menu principal



(b) Menu de configuração de cenários

Figura 2. Visão geral dos menus do CLI da ferramenta.

Selecionando-se a opção 1, é solicitado que o usuário digite o nome do novo cenário. Nesse momento, o componente *Criador* é chamado, o qual consulta o componente *Gerente* para validar se já existe um cenário cadastrado no banco de dados utilizando o nome especificado. Caso o nome já esteja em uso, será pedido que o usuário defina outro nome; caso o nome não conste no banco de dados, o cenário será criado e armazenado.

A opção 2 retorna a lista dos cenários existentes e seus estados (e.g., Parado ou

Executando). As informações são retiradas da tabela *cenarios* do banco de dados pelo componente *Gerente* e apresentadas em forma de tabela usando a biblioteca *Texttable* do Python. O esquema das tabelas do banco de dados pode ser visto na Figura 3, nela pode-se ver também a tabela *containers* onde são armazenadas as informações de todos os *containers* criados pela ferramenta, tornando possível definir qual *container* pertence a determinado cenário e permitindo diferenciar *containers* clientes e servidores.

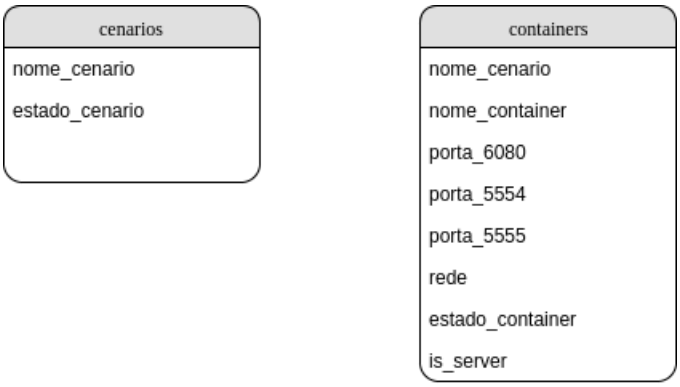


Figura 3. Esquema das tabelas do banco de dados.

Na opção 3, solicita-se ao usuário que digite o nome de um cenário existente para ter acesso ao menu visto na Figura 2(b), onde estão disponíveis as seguintes opções:

- Adicionar cliente: recebe um nome único e uma configuração de rede. Essas informações são passadas ao componente *Criador*, gravadas no banco de dados e o *container* é instanciado;
- Adicionar servidor: semelhante à opção anterior, recebe um nome único e é trabalho do *Criador* gravar as informações no banco e instanciar o novo *container*;
- Listar *containers* do cenário: passando como argumento o nome do cenário que está sendo configurado, o componente *Gerente* realiza uma consulta no banco de dados e retorna uma tabela contendo o nome do *container*, endereço IP, endereço para acesso via VNC, tipo de rede utilizada e estado do *container*;
- Apagar *container*: recebe um nome e, então, o componente *Criador* executa a exclusão do *container* especificado juntamente com as informações do banco de dados referentes ao mesmo;
- Conectar-se aos dispositivos: o componente *Gerente* inicia o ADB, permitindo a conexão com os dispositivos móveis que fazem parte do cenário;
- Instalar APP: utilizando o ADB, o componente *Gerente* instala uma aplicação em todos os dispositivos móveis que fazem parte do cenário;
- Iniciar cenário: inicia a execução dos *containers* do cenário, caso seja um servidor é iniciado o *MpOS Platform*, caso seja cliente é iniciado o emulador Android;
- Parar cenário: para a execução de todos os *containers* do cenário;
- Voltar para o menu principal: volta para o menu inicial da ferramenta.

Uma vez que o cenário for configurado e iniciado, o usuário pode acessar os emuladores dos *containers* clientes por meio de um cliente VNC baseado em HTML5 (noVNC[4]). A Figura 4 demonstra o acesso a dois *containers* clientes por meio do VNC.

[4]https://github.com/novnc/noVNC.

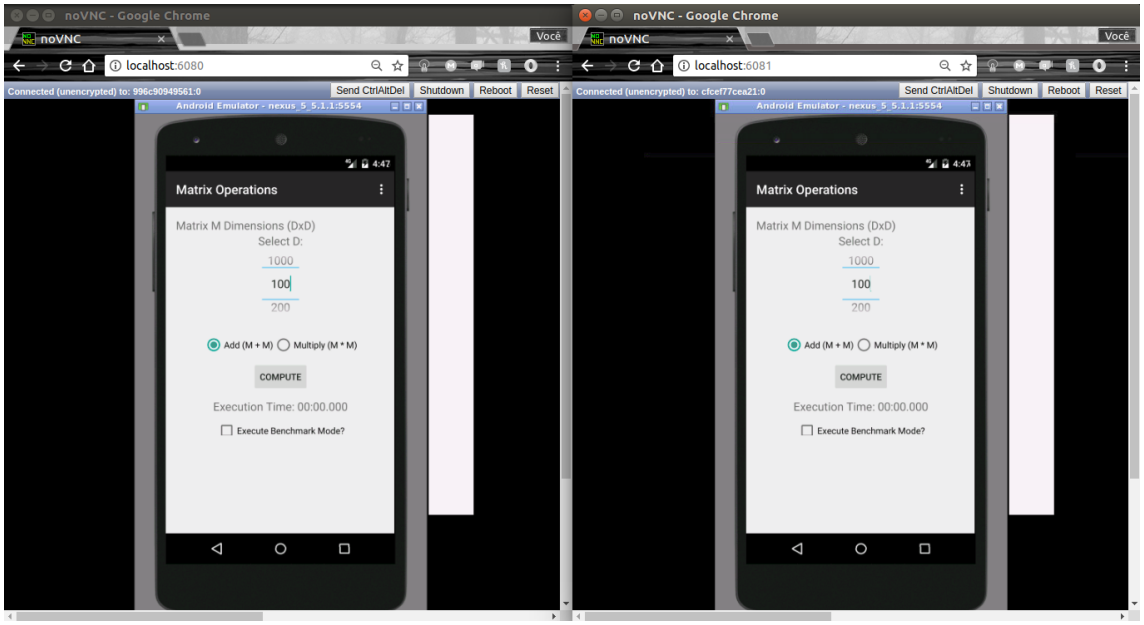


Figura 4. Acessando a interface dos emuladores através de um navegador Web. A aplicação *Matrix Operations* está sendo executada nos dois emuladores.

3.1. Automatizando os experimentos

Para automatizar a execução de testes, o usuário pode utilizar a API desenvolvida para programar experimentos utilizando *scripts* Python. A API, intitulada *DeviceManagerAPI*, depende do arquivo *comandos.py* e de informações do banco de dados, sendo assim necessário que todos os arquivos estejam no mesmo diretório. A Figura 5 mostra o esquema de classes implementado pela API.

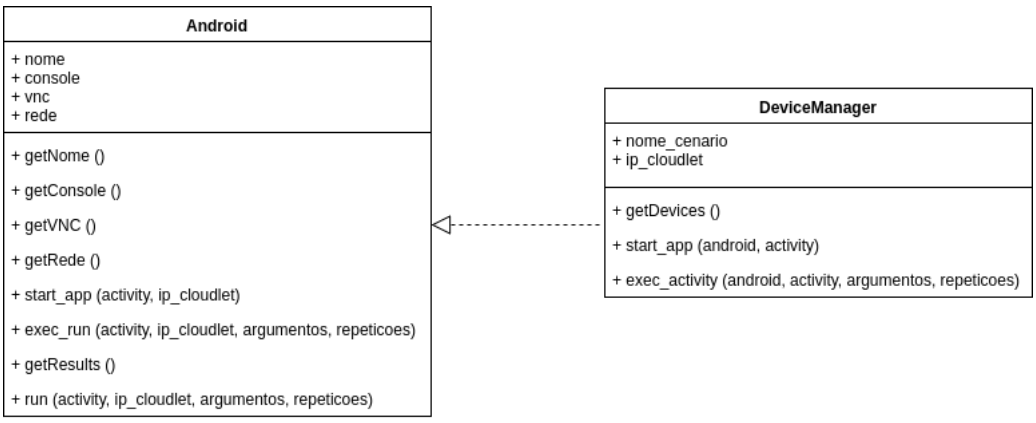


Figura 5. Classes implementadas pela API desenvolvida.

A Figura 6 apresenta um *script* que exemplifica o uso da API. Na linha 3, a API é importada e é instanciada na linha 6, passando como argumentos o nome do cenário desejado e o IP da máquina onde o *cloudlet* está executando. Na linha 8, salva-se na variável *dispositivos* uma lista de objetos da classe Android, onde cada *container* cliente que faz parte do cenário especificado representa um objeto na lista retornada pelo método *getDevices* da API. Na linha 10 são iniciadas as aplicações para cada dispositivo; o método

start_app deve receber um objeto Android e o nome da *activity* que a aplicação deverá iniciar. Na linha 13 o comando *sleep* é usado para dar tempo suficiente para a aplicação descobrir o serviço do MpOS e estabelecer uma conexão com ele. Na linha 15, o método *exec_activity*, em cada um dos dispositivos clientes, recebe como parâmetros um objeto Android, a *activity* a ser executada, os argumentos que serão usados na execução da *activity*, e a quantidade de execuções a serem feitas. Ao fim do processo, os resultados estarão disponíveis em arquivos que possuem o mesmo nome dos *containers* do cenário definido.

```
1  #!/usr/bin/python3
2
3  from DeviceManagerAPI import DeviceManager
4  import time
5
6  DM = DeviceManager("cenario-1", "192.168.1.36")
7
8  dispositivos = DM.getDevices()
9
10 for android in dispositivos:
11   DM.start_app(android, "br.ufc.great.matrixoperations/.MainActivity")
12
13 time.sleep(15)
14
15 for android in dispositivos:
16   DM.exec_activity(android, "br.ufc.great.matrixoperations/.
       MainActivity", "--es 'operation' 'mul' --ei 'size' 500", 5)
```

Figura 6. Script com exemplo de uso da API para automatizar experimentos

O *script* apresentado executa a *activity br.ufc.great.matrixoperation/.MainActivity* passando para ela dois parâmetros: *operation*, que indica o tipo de operação que será executada, no exemplo foi usado "mul" para multiplicação; *size*, que indica o tamanho da matriz que seria gerada.

Para que o *script* possa executar métodos, como no exemplo apresentado, é preciso configurar a aplicação Android para aceitar e tratar *extras*, caso contrário só será possível executar aplicações por meio da interface VNC do dispositivo.

4. Demonstração

O código fonte e o guia de instalação da ferramenta estão disponíveis para a comunidade no Github: `https://github.com/alvesRenan/androidTestBed`. Além disso, foram preparados dois vídeos demonstrativos: um com o guia de instalação (`https://youtu.be/7nu-24ESTl0`) e outro para apresentar as funcionalidades da ferramenta (`https://youtu.be/uZj6Gl6R19Q`).

Durante o evento, será feita uma demonstração das funcionalidades da ferramenta, oportunidade esta em que será criado um cenário com um *cloudlet* e quatro dispositivos Android, dois deles configurados com uma rede LTE e os outros dois com uma rede UMTS. Uma aplicação que faz operações matriciais será instalada nos quatro dispositivos e um *script* de experimentação será criado para executar um método de multiplicação de matrizes, com matrizes de dimensões variadas. Será possível observar a diferença no

tempo de execução do *offloading* quando redes diferentes são utilizadas, bem como a diferença entre utilizar um servidor dedicado e um compartilhado com até 4 dispositivos.

5. Conclusão

Diante da carência de ferramentas que possibilitem a experimentação das soluções propostas na área de MCC, este trabalho apresentou uma ferramenta que utiliza o Docker para criar ambientes de experimentação com múltiplos dispositivos móveis e *cloudlets*, além de permitir a aplicação de diferentes configurações de rede aos dispositivos. Com a solução proposta, desenvolvedores e pesquisadores podem criar cenários complexos e testar aplicações e validar suas teorias.

Como trabalhos futuros, pretende-se implementar uma funcionalidade que forneça ao usuário um maior controle sobre os *containers* clientes, permitindo que ele defina fatores como a quantidade de memória e CPU disponível para o emulador Android, além de configurar, com uma menor granularidade, as características da rede (e.g., taxas de *upload* e de *download*, patamares de perda de pacotes e de latência etc.).

Referências

Fernando, N., Loke, S. W., and Rahayu, W. (2013). Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84 – 106. Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

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.

Hoang, D. 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.

Kosta, S., Aucinas, A., Hui, P., Mortier, R., and Zhang, X. (2012). Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In Greenberg, A. G. and Sohraby, K., editors, *INFOCOM*, pages 945–953. IEEE.

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.

Rego, P. A. L., Costa, P. B., Coutinho, E. F., Rocha, L. S., Trinta, F. A., and de Souza, J. N. (2016). Performing computation offloading on multiple platforms. *Computer Communications*.