UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

TECNÓLOGO EM REDES DE COMPUTADORES

RENAN ALVES BARBOSA

UMA FERRAMENTA BASEADA EM CONTAINERS DOCKER PARA EXPERIMENTAÇÃO EM MOBILE CLOUD COMPUTING

QUIXADÁ

2017

RENAN ALVES BARBOSA

UMA FERRAMENTA BASEADA EM CONTAINERS DOCKER PARA EXPERIMENTAÇÃO EM MOBILE CLOUD COMPUTING

Monografia apresentada no curso de Redes de Computadores da Universidade Federal do Ceará, como requisito parcial à obtenção do título de tecnólogo em Redes de Computadores. Área de concentração: Computação.

Orientador: Dr. Paulo Antonio Leal Rego

QUIXADÁ

2017

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

B211f Barbosa, Renan Alves.
Uma ferramenta baseada em containers docker para experimentação em mobile cloud computing / Renan Alves Barbosa. – 2017.
37 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Redes de Computadores, Quixadá, 2017.
Orientação: Prof. Dr. Paulo Antonio Leal Rego.

1. Computação em nuvem móvel. 2. Offloading computacional. I. Título.

CDD 004.6

RENAN ALVES BARBOSA

UMA FERRAMENTA BASEADA EM CONTAINERS DOCKER PARA EXPERIMENTAÇÃO EM MOBILE CLOUD COMPUTING

Monografia apresentada no curso de Redes de Computadores da Universidade Federal do Ceará, como requisito parcial à obtenção do título de tecnólogo em Redes de Computadores. Área de concentração: Computação.

Aprovada em: __/__/__

BANCA EXAMINADORA

Dr. Paulo Antonio Leal Rego (Orientador)
Universidade Federal do Ceará – UFC

Prof. Dr. Elvis Miguel Galeas Stancanelli
Universidade Federal do Ceará - UFC

Prof. Dr. Márcio Espíndola Freire Maia
Universidade Federal do Ceará - UFC

A minha avó Maria de Lourdes (in memorian) por todos os momentos que passamos juntos e a todos aqueles que de alguma forma estiveram e estão próximos de mim, fazendo esta vida valer cada vez mais a pena.

AGRADECIMENTOS

A Deus por minha vida, família e amigos. Aos meus pais, pelo amor, incentivo e apoio incondicional. A Instituição pelo ambiente criativo e amigável que proporciona.

Meus agradecimentos aos meus amigos, em especial Ana Lucia, Fábio Correia, Ueliton Souza e Walafi Ferreira, companheiros de trabalhos e irmãos na amizade que fizeram parte da minha formação e que vão continuar presentes em minha vida com certeza.

Ao Prof. Dr. Paulo A. L. Rego pela oportunidade e apoio na elaboração deste trabalho. Aos professores participantes da banca examinadora pelo tempo, pelas valiosas colaborações e sugestões. A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

"*Only courage in the face of doubt can lead one to the answer.*"

(Nyx Avatar)

RESUMO

Mobile Cloud Computing surgiu a partir da colaboração entre a computação em nuvem e a computação móvel visando melhorar a execução das aplicações em dispositivos móveis através do poder de processamento fornecido pela computação em nuvem. Diversos trabalhos na área de *Mobile Cloud Computing* analisam o impacto das condições da rede na execução do *offloading*. Porém, muitos o fazem em cenários com apenas um usuário, seja por falta de recursos ou pela dificuldade de gerenciamento de vários dispositivos. Este trabalho teve como objetivo o desenvolvimento de uma ferramenta que permitisse a criação e configuração de ambientes de teste com múltiplos dispositivos móveis. Para avaliar a ferramenta proposta, um experimento foi executado, onde foram criados clientes e servidores, com diferentes configurações de rede. Os clientes executaram uma aplicação para calcular a multiplicação de duas matrizes e durante o experimento cada dispositivo executou 30 vezes. Os resultados obtidos demonstram que a solução proposta cumpriu os objetivos e é uma opção viável para desenvolvedores e pesquisadores.

Palavras-chave: Computação em nuvem móvel. *Offloading* computacional.

ABSTRACT

Mobile Cloud Computing emerged from the collaboration between cloud computing and mobile computing aiming to improve the execution of applications on mobile devices with the processing power provided by cloud computing. Many works on Mobile Cloud Computing study the impact of network conditions on the offloading exection. However, ones analyze scenarios with just one user, be it because no resources are available or because of the difficult of managing several devices. This work aimed to develop a tool that allowed the making and setting up of test environments with multiple mobile devices. To evaluate the proposed tool, an experiment was executed, where clients and servers were created, with different network configurations. Clients ran an application to calculate the multiplication of two matrices and during the experiment each device performed 30 times. The obtained results demonstrate that the proposed solution fulfilled the objectives and is a viable option for developers and researchers.

Keywords: Mobile Cloud Computing. Computational Offloading.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

MCC	*Mobile Cloud Computing*
ADB	Android Debugging Bridge

SUMÁRIO

1 INTRODUÇÃO

Nos dias atuais, os dispositivos móveis ainda apresentam diversas restrições. De acordo com Fernando, Loke e Rahayu (2013), fatores como uso de energia e capacidade de processamento dificultam o desenvolvimento de aplicações mais complexas para dispositivos móveis. *Mobile Cloud Computing* (MCC) surge como uma junção da computação em nuvem e computação móvel no intuito de solucionar tais problemas usando os recursos de outros dispositivos, por exemplo uma máquina virtual na nuvem, para a execução de uma aplicação por completa ou partes dela (FERNANDO; LOKE; RAHAYU, 2013) (HOANG et al., 2013).

Para que isso aconteça, é necessário que os dispositivos enviem os dados a serem processados para a outra máquina. Isso é feito pelo processo de *offloading*, pelo qual um dispositivo envia cargas de trabalho a outro dispositivo para que as mesmas sejam processadas e os resultados do processamento enviados de volta ao primeiro dispositivo (KUMAR et al., 2013).

MCC tem chamado a atenção da indústria, por ser uma alternativa que reduz os custos de desenvolvimento e execução de aplicações móveis, e a dos usuários de dispositivos móveis, por ser uma nova tecnologia que possibilita proporcionar uma melhor experiência para uma série de serviços móveis a baixo custo (HOANG et al., 2013).

Diversos estudos na área de MCC avaliam o impacto do *offloading* no desempenho das aplicações. No entanto, a maioria dos cenários de testes usados não condiz com a realidade de um ambiente clássico de computação em nuvem, onde múltiplos usuários interagem simultaneamente com o servidor e a demanda muda constantemente.

Nesse contexto, vê-se a necessidade de uma ferramenta para auxiliar desenvolvedores e pesquisadores de MCC na criação e execução de experimentos em cenários com múltiplos dispositivos móveis. Tal ferramenta deve ser 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.

Sabendo disso, o objetivo deste trabalho é desenvolver uma ferramenta para permitir que desenvolvedores e pesquisadores criem cenários de experimentação para testar suas aplicações. A solução desenvolvida tem foco na plataforma Android, ela foi escolhida por sua grande abrangência em meio aos desenvolvedores e usuários.

Os testes feitos utilizaram a ferramenta desenvolvida para criar dois cenários de teste com configurações distintas e comparar o tempo de execução de um método de uma aplicação

Android nos dois cenários.

O trabalho está organizado da seguinte maneira: a fundamentação teórica é apresentada no Capítulo 2. No Capítulo 3 são apresentados os trabalhos relacionados. O Capítulo 4 detalha a arquitetura da solução proposta e o Capitulo 5 explica como usá-la. No Capítulo 6 são discutidos os resultados da avaliação da ferramenta e por último a conclusão é feita no Capítulo 7.

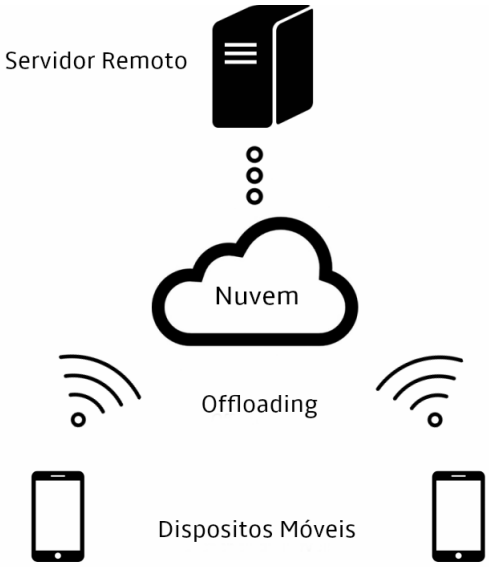
2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são apresentados os conceitos de *Mobile Cloud Computing* e *offloading*, e são comentados os serviços oferecidos pelo *framework* MpOS, que é o *framework* MCC que foi utilizado neste trabalho.

2.1 *Mobile Cloud Computing*

Segundo Fernando, Loke e Rahayu (2013), existem diversas definições para *Mobile Cloud Computing* (MCC). A mais comumente citada define MCC como o ato de executar uma aplicação, por exemplo o Google tradutor, em um servidor remoto que possui maior capacidade computacional enquanto o dispositivo móvel atua como cliente magro, que se conecta ao servidor remoto por meio de uma conexão 3G, 4G ou *Wi-Fi*. A Figura 1 ilustra o cenário citado.

Figura 1 – Visão Geral de MCC.



Fonte – Autor, 2017

Rahimi et al. (2014) se referem a *Mobile Cloud Computing* como um conjunto de técnicas que usam recursos da nuvem para melhorar a experiência de usuários que possuem dispositivos com recursos e capacidades limitados como processamento, armazenamento e bateria. Semelhante a essa definição, Hoang et al. (2013) conceituam o paradigma de MCC como a conexão entre os usuários móveis e os recursos oferecidos pela computação em nuvem. Isso implicaria a necessidade de dispositivos móveis com maior poder de processamento, uma vez que todo o processamento e armazenamento seria feito na nuvem.

Kovachev, Cao e Klamma (2011), por sua vez, separam as aplicações móveis em três tipos:

- Aplicações *offline*, onde todo o processamento de informações é feito no dispositivo cliente. Este tipo de aplicação tem, entre outras vantagens, o fato de estar sempre disponível para seus usuários. No entanto, elas possuem código mais complexo e não possuem portabilidade para outras plataformas;
- Aplicações *online* assumem que o usuário está conectado durante a maior parte do tempo. Geralmente são aplicações *web*, o que permite que seus usuários possam acessá-las de qualquer lugar e em diferentes plataformas. No entanto, algumas delas podem ser afetadas pela latência da rede (e.g, aplicações em tempo real) e podem ter problemas com cenários em que a comunicação precisa ficar ativa durante longos períodos de tempo;
- Aplicações híbridas podem executar tarefas tanto localmente como remotamente, dessa forma elas podem ter os mesmo benefícios de uma aplicações *online* e *offline*.

Segundo Fernando, Loke e Rahayu (2013), a operação principal em qualquer nuvem móvel é a ação de fazer *offloading* de trabalhos muito pesados. Por esse motivo, o presente trabalho foca em aplicações híbridas, visto que elas podem se beneficiar do uso de *offloading*.

2.1.1 Offloading

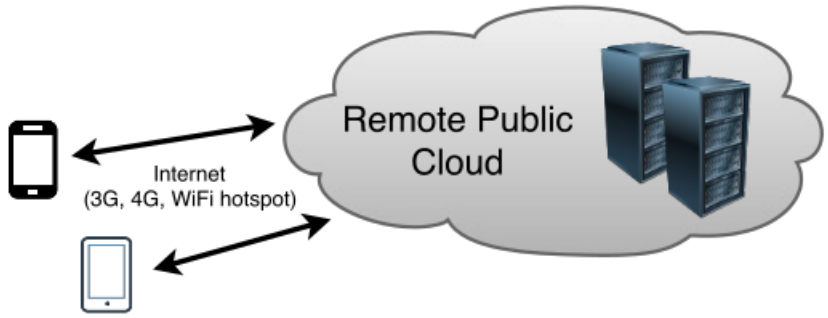
Offloading é o ato de enviar cargas de trabalho para serem processadas em um máquina que disponha de maior poder computacional. Após o processamento das informações, a máquina retorna os resultados encontrados (KUMAR et al., 2013).

O *offloading* pode ser feito para diferentes locais. Por exemplo, um dispositivo pode fazer *offloading* para outro dispositivo que se encontra na mesma rede que ele (caracterizando assim uma *cloudlet*) ou para um dispositivo que não faz parte da sua rede local (e.g., uma máquina na nuvem). Os ambiente de *offloading* disponíveis são:

- Nuvem pública:

  A execução de tarefas na nuvem pública é bastante comum. Diversos trabalhos, como (KOSTA et al., 2012) e (KEMP et al., 2012), descrevem *frameworks* que permitem aplicações móveis executarem *offloading* na nuvem. Para se comunicar com a nuvem pública os dispositivos podem utilizar redes móveis (e.g., 3G, 4G) ou *Wi-Fi* (REGO, 2016). A Figura 2 demonstra o cenário descrito.
- *Cloudlet*:

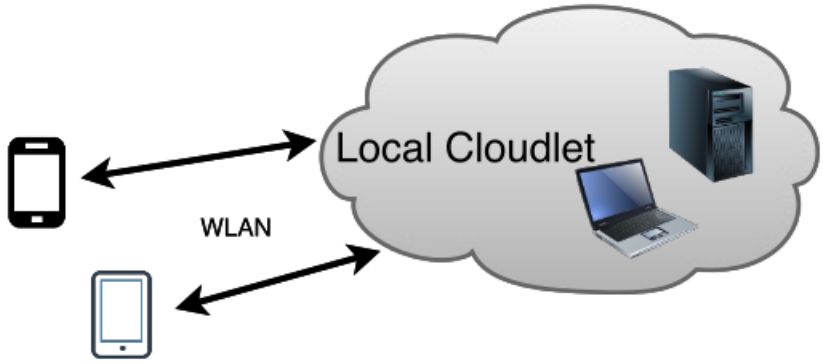
Figura 2 – Execução de *offloading* na nuvem pública



Fonte – Rego (2016)

A Figura 3 monstra a arquitetura de uma *Cloudlet*, cujo intuito é permitir que os dispositivos móveis executem *offloading* em outros dispositivos que se encontram na mesma rede local, diminuindo assim atrasos referentes à velocidade de transmissão da rede (SATYANARAYANAN et al., 2009).

Figura 3 – Execução de *offloading* em *cloudlet*.



Fonte – Rego (2016)

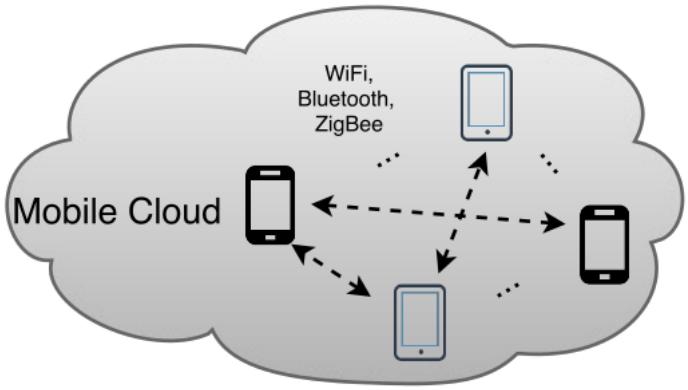
- Nuvem móvel/*Cluster* de dispositivos:

  Outra arquitetura é a nuvem móvel, apresentada na Figura 4. Nela os dispositivos móveis estão conectados uns aos outros por meio de uma rede *peer-to-peer*, criando uma nuvem móvel. Isso implica que todo dispositivo que faz parte dessa rede *peer-to-peer* pode ser considerado como provedor de recursos para outro dispositivo que também conectados a essa rede *peer-to-peer* (FERNANDO; LOKE; RAHAYU, 2013).

- Ambiente híbrido:

  O ambiente híbrido é formado quando o dispositivo móvel dispõe simultaneamente de acesso a dois ou mais dos cenários citados anteriormente. A Figura 5 mostra um cenário

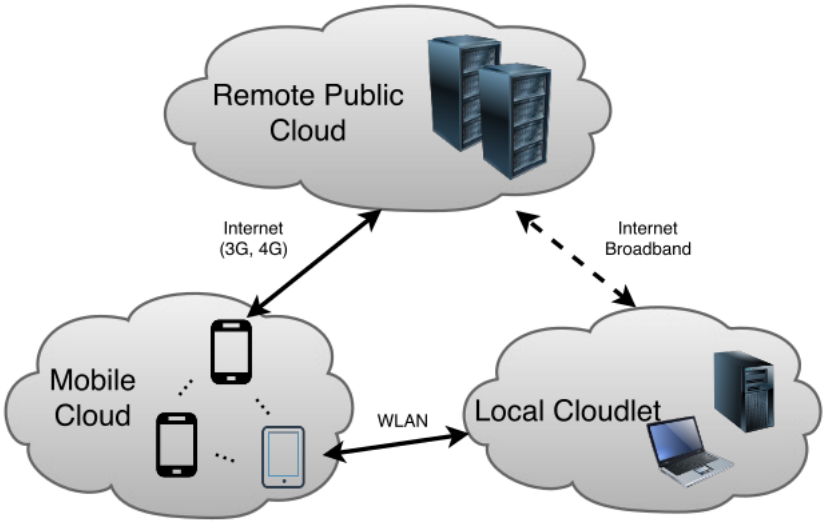
Figura 4 – Execução de *offloading* em outro dispositivo móvel.



Fonte – Rego (2016)

híbrido em que um dispositivo faz parte de uma nuvem móvel e tem acesso tanto a uma *cloudlet* quanto a uma nuvem remota.

Figura 5 – Execução de *offloading* com arquitetura híbrida.



Fonte – Rego (2016)

Como dito na seção anterior, dispositivos móveis possuem capacidade computacional limitada. Kumar et al. (2013) afirmam que recursos como a duração da bateria, capacidade de armazenamento e desempenho do processador podem ser aliviados por meio de *offloading*.

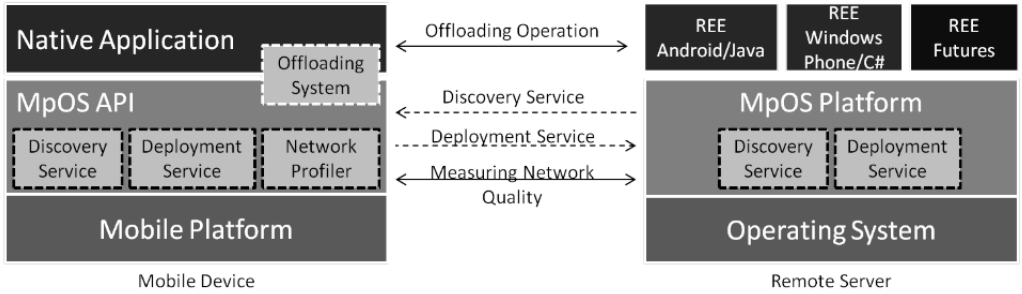
No entanto, podem haver situações em que o processo de *offloading* é mais lento do que a execução no próprio dispositivo. Isso pode acontecer devido à taxa de transmissão da conexão do dispositivo. Em (KUMAR et al., 2013), os autores apresentam uma fórmula para

determinar quando o *offloading* de cargas de trabalho será benéfico para a execução da aplicação, levando em conta a quantidade de informações a ser processada, o poder de processamento do dispositivo móvel, a taxa de transmissão do dispositivo móvel e a velocidade de processamento do dispositivo para onde os dados estão sendo enviados. Com essa fórmula, pode-se determinar que o *offloading* será mais rápido quando: o tempo de processamento da carga de trabalho no dispositivo móvel for maior que o tempo necessário para enviar as informações para outro dispositivo, realizar o processamento e retornar o resultado.

2.1.2 MpOS

O MpOS é um *framework* para realizar *offloading* de métodos de aplicações Android e Windows Phone. A Figura 6 mostra como está organizada a arquitetura do MpOS, tanto para a parte do cliente (representado pelo lado esquerdo da Figura 6) quanto para a parte servidor (representado pelo lado direito da Figura 6).

Figura 6 – Visão Geral da Arquitetura do MpOS.



Fonte – Costa et al. (2015)

No lado cliente, encontra-se a API do MpOS, que é adicionada à aplicação. Nesta, existem quatro serviços que também são executados na parte servidor do MpOS:

- *Discovery Service*: responsável por detectar se há algum servidor MpOS operando na rede local *wireless*;
- *Deployment Service*: caso o *Discovery Service* falhe em achar um serviço de *offloading* em um servidor remoto, o *Deployment Service* é responsável por enviar as dependências necessárias (e.g., binários, *libs*, informações) para que o servidor remoto inicie um serviço de *offloading* para uma aplicação móvel;
- *Network Profiler*: serviço responsável por medir a qualidade da conexão entre o dispositivo móvel e o servidor remoto;

- *Offloading Service*: serviço responsável por interceptar os métodos candidatos a sofrer *offloading* e, usando as informações obtidas pelo serviço *Network Profiler*, determinar se eles serão executados no própio dispositivo móvel ou em um servidor remoto.

O MpOS foi o *framework* de *offloading* escolhido para ser usado neste trabalho. A escolha se deve ao fato do MpOS apresentar configurações e execução bastante simples e, ainda assim, proporcionar aumento satisfatório na velocidade de execução de aplicações, características que chamam a atenção de desenvolvedores que não tem tanta experiência de mercado e que facilitam o trabalho de pesquisadores durante a montagem de um cenário de testes.

3 TRABALHOS RELACIONADOS

Em Costa et al. (2014) é conduzido um teste para avaliação da Internet móvel em três capitais do nordeste do Brasil, Forataleza, Salvador e Recife. Em seguida é feita uma comparação entre o uso de *cloudlets* e a nuvem pública para a realização de *offloading*, além de analisar o desempenho das redes celulares 4G em aplicações que utilizam *offloading*.

Para avaliar a qualidade da internet móvel de quatro operadoras, foi desenvolvida a aplicação NETester para a plataforma Android. Para o primeiro teste, a aplicação foi executada 10 vezes para determinar taxas de *upload*, *download*, RTT (tempo decorrido entre o momento do envio do pacote, sua chegada no destino e o retorno para a origem), total de pacotes perdidos, e *jitter* entre o dispositivo móvel e máquinas virtuais da Amazon EC2. Para o segundo teste, foi utilizada a função SignalHunter, que coleta informações sobre o tipo de conexão usada pelo dispositivo no momento (COSTA et al., 2014).

Para comparar a nuvem pública e a *cloudlet*, foi desenvolvida, também para a plataforma Android, a aplicação BenchImage, que faz a aplicação de efeitos em imagens, podendo executar apenas no dispositivo móvel ou através de *offloading* para uma nuvem pública ou uma *cloudlet*. No experimento, a aplicação foi executada dez vezes com cinco tamanhos diferentes de imagem e em seis ambientes de execução diferentes. Para realizar os experimentos localmente, foram usados dois *smartphones* de modelos diferentes; já os testes de *offloading* foram realizadas utilizando apenas um *smartphone* (COSTA et al., 2014).

Assim como no trabalho citado, o presente trabalho fará uma análise do uso de *offloading* em *cloudlets*, no entanto, os testes serão realizados com múltiplos dispositivos simultaneamente. Pretende-se avaliar o desempenho do servidor e avaliar como ele responderá às diversas requisições simultâneas.

Kosta et al. (2012) apresentam a ferramenta *ThinkAir*, um *framework* que visa simplificar o trabalho de desenvolvedores na tarefa de portar suas aplicações para a nuvem. Para isso, o *ThinkAir* virtualiza o sistema operacional Android em seis configurações diferentes, utilizando uma versão modificada do Android x86.

O *ThinkAir* possui três componentes principais. O primeiro é denominado ambiente de execução, nele ocorre a reescrita da aplicação, onde o desenvolvedor marca os métodos candidatos a *offloading*, e a tomada de decisão de *offloading*, feita através de dados do ambiente atual e de execuções anteriores. O segundo componente é o servidor da aplicação, o qual é responsável por gerenciar a comunicação entre cliente e servidor, gerenciar o estado das

máquinas virtuais (desligada, em pausa e executando) e por possibilitar o paralelismo. O último componente é o *profiling*, que é responsável por coletar informações e guardar as estatísticas de execuções passadas (KOSTA et al., 2012).

Kosta et al. (2012) afirmam que o *ThinkAir* visa a utilização da nuvem no cenário comercial, dando suporte a múltiplos usuários. O presente trabalho também foca na ideia de múltiplos usuários, e os cenários que serão avaliados terão vários dispositivos executando *offloading* simultaneamente, podendo impactar no desempenho do servidor.

Em Costa et al. (2015) é apresentado o MpOS (*Multiplatform Offloading System*), um *framework* para fazer *offloading* de diferentes plataformas. O MpOS funciona utilizando a arquitetura cliente-servidor, e os métodos que terão chance de *offloading* são 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.

Para decidir se vale ou não a pena fazer o *offloading* do processamento, o MpOS coleta as informações de *jitter*, RTT e taxa de transferência da atual conexão do dispositivo móvel com a internet e, baseado nelas, faz a tomada de decisão, utilizando regras simples definidas pelo desenvolvedor (e.g., fazer *offloading* se RTT < 50 ms) (COSTA et al., 2015).

Este trabalho utilizará o *framework* MpOS para configurar aplicações Android a suportarem *offloading*. Ele foi escolhido pela sua simplicidade e eficiência em relação a outros *frameworks*. Diferente dos testes executados por (COSTA et al., 2015), o presente trabalho utilizará múltiplos dispositivos para avaliar a ferramenta desenvolvida em cenários mais variados.

Baseado nos trabalhos acima relacionados, o presente estudo, semelhante ao estudo de Costa et al. (2014), visa analisar o desempenho da técnica de *offloading*, mas, o fará em um cenário onde múltiplos dispositivos estão atuando simultaneamente, semelhante a um dos testes descritos em Kosta et al. (2012). Para isso, será desenvolvida uma ferramenta para a criação dos dispositivos virtuais e dos ambientes de testes. O MpOS (COSTA et al., 2015) estará presente na aplicação que os dispositivos móveis executarão, sendo responsável pelo *offloading* dos métodos além de ser o servidor responsável por responder às requisições dos dispositivos. Ressalta-se que o canal de comunicação utilizado pelos dispositivos e o tipo de conxeão são definidos pelo emulador utilizado.

4 SOLUÇÃO PROPOSTA

A ferramenta desenvolvida neste trabalho propõe a utilização de *containers* Docker para a criação de cenários de testes, onde o usuário pode desenvolver aplicações *Mobile Cloud Computing* e testar seu desempenho em um ambiente com múltiplos dispositivos móveis.

A Secção 4.1 descreve os requisitos funcionais e não funcionais da ferramenta. Em seguida, na Seção 4.2, a organização e funcionamento da solução proposta são descritos.

4.1 Requisitos funcionais e não funcionais

Dentre os requisitos funcionais da ferramenta desenvolvida, tem-se:

1. O usuário deve poder criar cenários de execução;
2. O usuário deve poder listar, excluir e configurar cenários já existentes;
3. O usuário deve poder listar os *containers* que fazem parte de um cenário;
4. O usuário deve poder salvar as informações relacionadas aos cenários e os *containers* que fazem parte dos mesmos em um banco de dados;
5. O usuário deve poder adicionar *containers* que farão papel de cliente (dispositivo móvel), sendo possível definir nome e o tipo de rede utilizada pelo mesmo;
6. O usuário deve poder adicionar *containers* que ficarão encarregados de responder às requisições vindas dos *containers* clientes;
7. O usuário deve poder excluir *containers* criados;
8. O usuário deve poder se conectar aos *containers* clientes via ADB;
9. O usuário deve poder instalar aplicações nos *containers* clientes via ADB;
10. O usuário deve poder iniciar ou parar a execução de todos os *containers* de um cenário.

Definiu-se como requisitos não funcionais os seguintes itens:

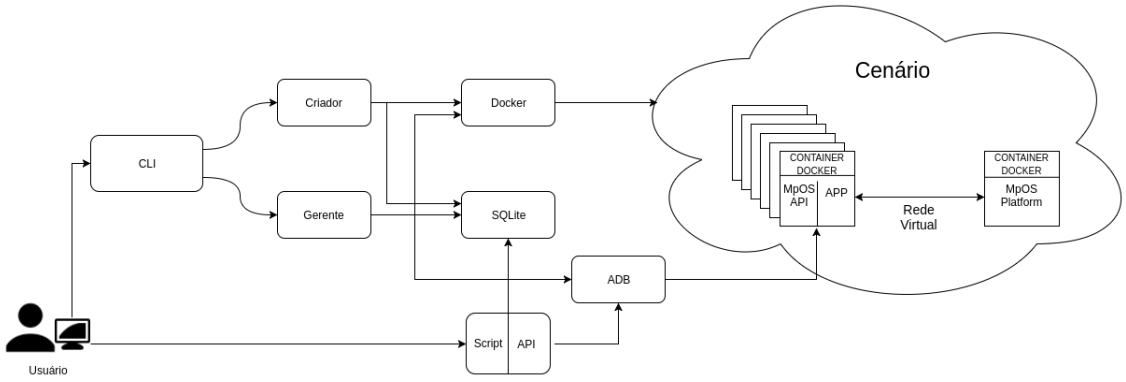
1. A ferramenta deve permitir ao usuário executar métodos de aplicações Android via linha de comando;
2. A ferramenta deve recuperar informações de experimentos executados em um cenário.

4.2 Arquitetura e componentes

A ferramenta proposta foi desenvolvida utilizando a linguagem de programação Python e é composta por três componentes principais: *Command Line Interface (CLI)*, *Criador* e *Gerente*. Além deles, é também utilizada a ferramenta *Android Debugging Bridge (ADB)* responsável pela comunicação com os dispositivos Android, o Docker, responsável pela comunicação com os *containers*, além de um banco de dados *SQLite*, que é utilizado para guardar todas as informações sobre *containers* e os cenários.

A Figura 7 mostra como estão organizados os 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 *containers* e cenários, sendo responsável por se comunicar com o banco de dados *SQLite* 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*, 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.

Figura 7 – Arquitetura da solução proposta.



Fonte – Autor, 2017

A tecnologia de *containers* Docker é utilizada na ferramenta para criar dispositivos clientes e servidores. O Docker utiliza imagens para criar os *containers*, a imagem utilizada para criar os clientes foi retirada do repositório de imagens do Docker, o *dockerhub*, e foi alterada para interagir melhor com a ferramenta. Para os servidores foi utilizada uma imagem Ubuntu 16.10 com o Java versão 8.

Os *containers* clientes representam dispositivos móveis, que possuem um emulador

Android instalado, enquanto os *containers* servidores possuem os arquivos necessários para executar o *MpOS Platform*. A comunicação entre Python e Docker é feita utilizando a API *Docker* para a versão 3 do Python.

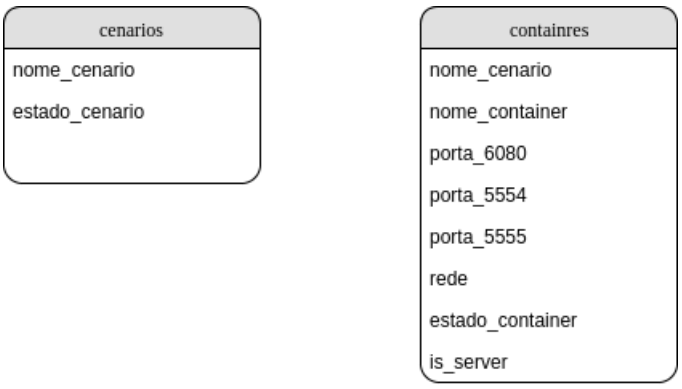
Além da ferramenta acessível via CLI, foi desenvolvida uma API para que o usuário possa automatizar testes na sua aplicação. Utilizando a linguagem Python e a API, o desenvolver pode criar *scripts* que executam métodos específicos da sua aplicação em todos os dispositivos do cenário.

Quando o usuário executa a *CLI*, ele tem acesso a quatro opções: (1) Criar cenário, (2) Listar cenários existentes, (3) Opções para cenários existentes e (4) Excluir cenário.

Selecionando a opção 1, será pedido que o usuário digite o nome do novo cenário. Nesse momento, o componente *Criador* é chamado, que por sua vez chamará 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 no banco de dados.

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 no Figura 8.

Figura 8 – Esquema das tabelas do banco de dados.



Fonte – Autor, 2017

Na opção 3, o usuário é solicitado a digitar o nome de um cenário existente e terá acesso a um novo menu onde estão disponíveis as seguintes opções:

- Adicionar cliente: recebe um nome único e uma configuração de rede, essas informações são passadas para ao componente *Criador*, gravadas no banco de

dados e o *container* é instanciado;

- Adicionar servidor: semelhante a 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 o componente *Criador* executará a exclusão *container* especificado juntamente com as informações do banco de dados referentes a ele;
- Conectar-se aos dispositivos: o componente *Gerente* iniciará o ADB, permitindo a conexão com os dispositivos móveis que fazem parte do cenário;
- Instalar APP: utilizando o ADB, o componente *Gerente* irá instalar 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, 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.

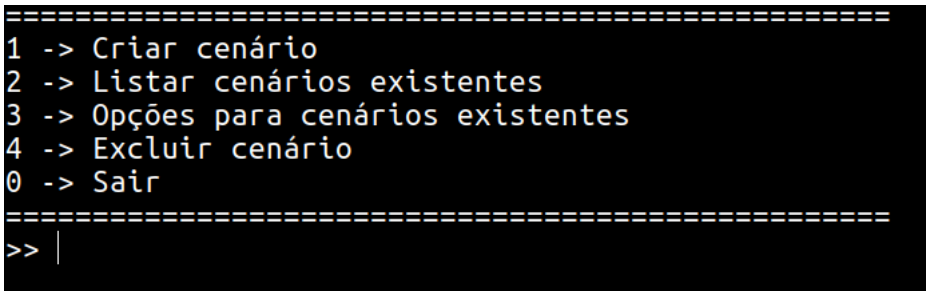
5 UTILIZANDO A SOLUÇÃO PROPOSTA

Este Capítulo apresenta como utilizar a solução desenvolvida. Na Seção 5.1, são descritos os passos para a criação e configuração de um cenário. A Seção 5.2 exemplifica o uso da API desenvolvida para a automação de testes.

5.1 Criação e configuração de um cenário

A Figura 9 representa a *CLI* apresentada quando o usuário executa a ferramenta. Ao selecionar a opção *Criar cenário*, o usuário deve inserir um nome para o novo cenário. Caso o nome digitado já exista, uma mensagem avisará o usuário e solicitará um novo nome. Escolhido um nome único, o cenário será criado e a ferramento voltará ao menu inicial. Ao utilizar a opção *Listar cenários*, uma lista com os cenários existentes é apresentada.

Figura 9 – CLI da ferramenta.

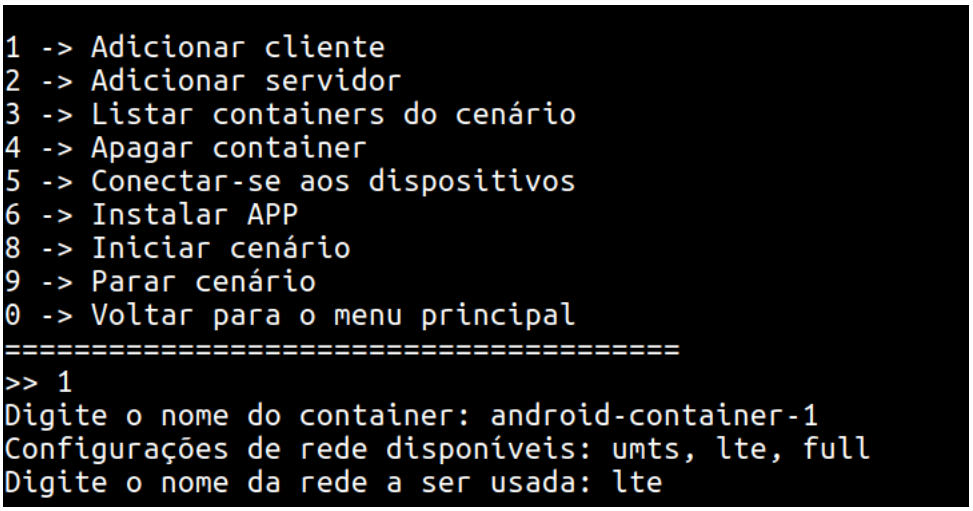


Fonte – Autor, 2017

Para configurar um cenário, o usuário deve selecionar a opção *Opções para cenários existentes* e digitar o nome do cenário que ele deseja configurar. Como dito na Seção 4.2, a ferramenta é capaz de criar dois tipos diferentes de *containers*: clientes e servidores. Os clientes possuem o emulador com a versão 5.1.1 do Android, já os servidores possuem os arquivos para a execução do *MpOS Platform*. Para adicionar um novo *container* cliente, deve-se escolher a opção *Adicionar cliente*, e então deve-se fornecer um nome único e escolher entre uma das três configurações de rede disponíveis, são elas: lte (rede móvel 4G), umts (rede móvel 3G) ou full (velocidade de comunicação da rede interna). Definidos esses parâmetros, o componente *Criador* é chamado, as informações são então gravadas no banco de dados e o *container* é criado. A Figura 10 demonstra a adição de um cliente utilizando a ferramenta.

A opção *Adicionar servidor* funciona de maneira semelhante à mencionada anteriormente. Nela, é necessário apenas que o usuário defina um nome único para o novo

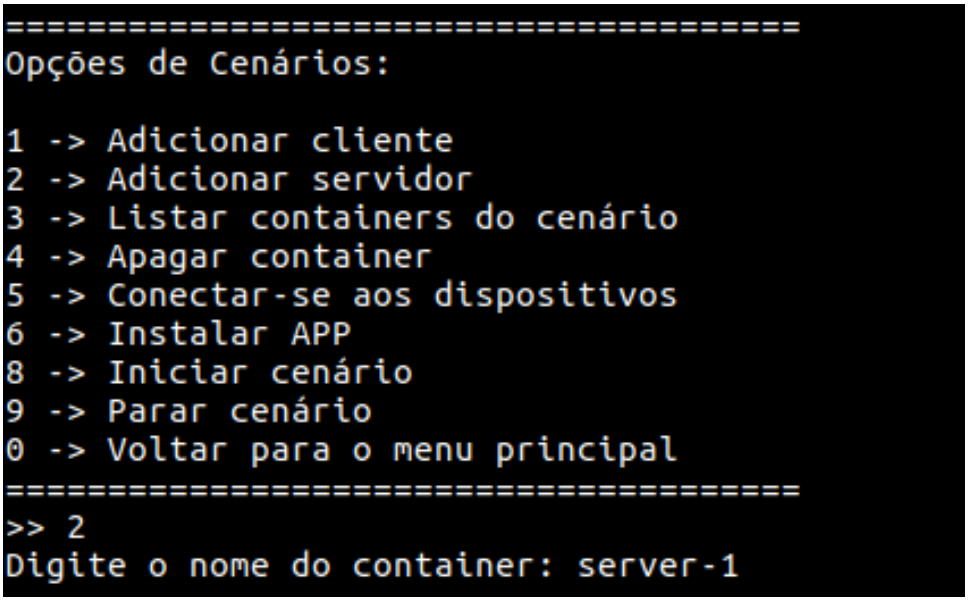
Figura 10 – Exemplo da criação de um *container* cliente.



Fonte – Autor, 2017

servidor, então o componente *Criador* salva as informações no banco de dados e cria o novo *container*. A Figura 11 exemplifica a criação de um *container* servidor.

Figura 11 – Exemplo da criação de um *container* servidor.

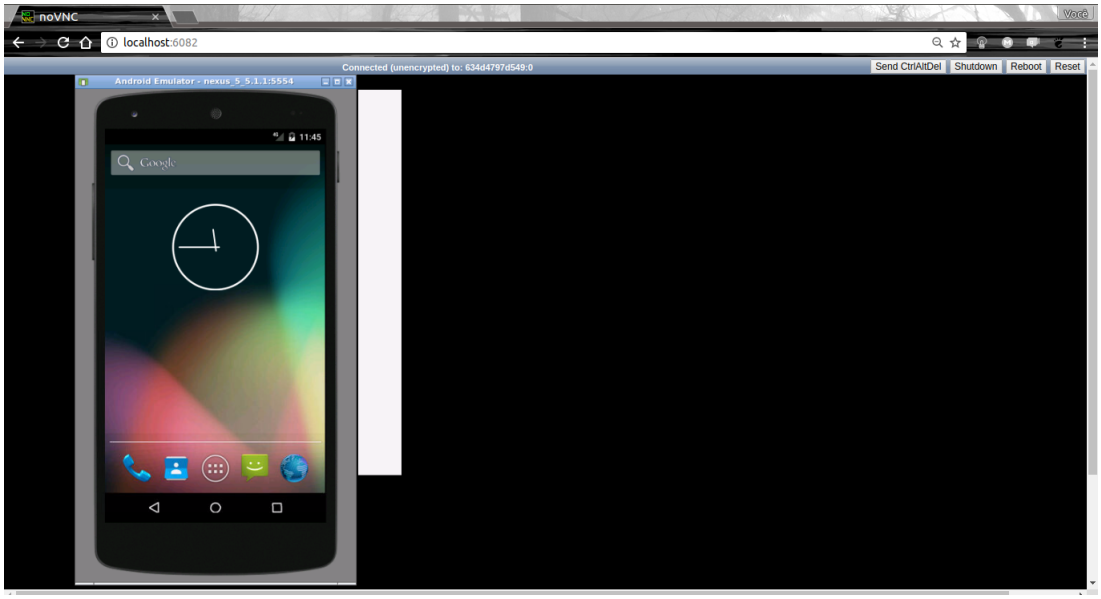


Fonte – Autor, 2017

Quando o número de dispositivos desejados for criado, a opção *Iniciar cenário* deve ser executada. Ela executará o componente *Gerente* para recuperar do banco de dados as informações referentes aos *containers* que fazem parte do cenário atual. Em posse das informações, a ferramenta iniciará o emulador Android com a configuração de rede escolhida previamente nos clientes e iniciará o serviço do *MpOS Platform* nos *containers* servidores.

Após o cenário ser iniciados já é possível acessar a interface do emulador Android via VNC como visto na Figura 12.

Figura 12 – Acessando a interface do emulador via VNC.



Fonte – Autor, 2017

Em seguida o usuário deve usar a opção *Conectar-se aos dispositivos* tal qual o exemplo da Figura 13. Ela executará dois comandos *shell*, sendo que o primeiro encerrará o ADB e segundo irá inicia-lo novamente. Tal ação é necessária para que o ADB se conecte automaticamente a todos os emuladores em execução.

Depois de estar conectado aos emuladores, a opção *Instalar APP* pode ser utilizada. Ao executá-la, o usuário deve indicar o caminho para o aquivo *apk* que será instalado nos clientes.

5.2 Automatizando a execução de um experimento

Caso não haja erros na execução dos passos da Seção 5.1, já será possível executar as aplicações instaladas nos *containers* clientes.

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 14 mostra o esquema de classes implementado pela API.

O código a baixo foi usado para automatizar os testes feitos neste trabalho, e pode

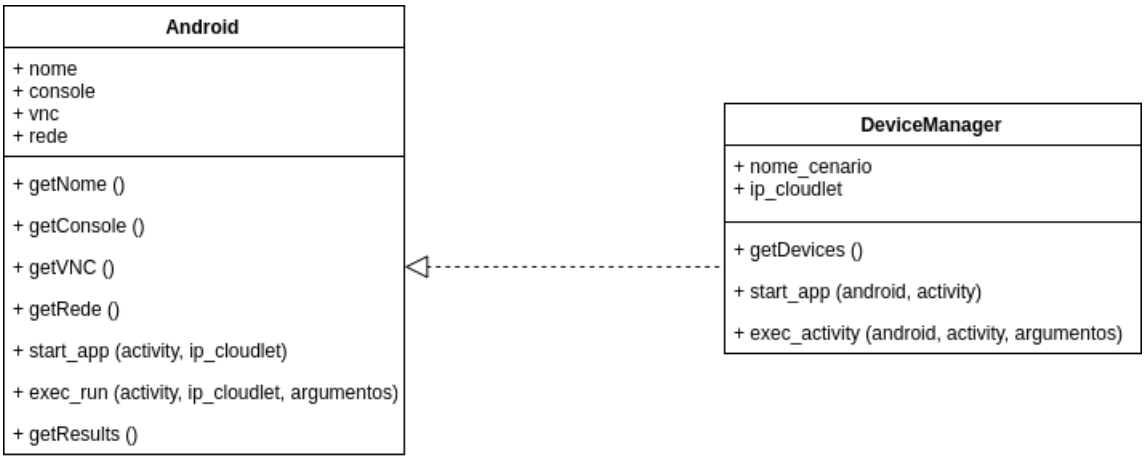
Figura 13 – Exemplo de saída da opção *Conectar-se aos dispositivos*.

```
========================================
Opções de Cenários:

1 -> Adicionar cliente
2 -> Adicionar servidor
3 -> Listar containers do cenário
4 -> Apagar container
5 -> Conectar-se aos dispositivos
6 -> Instalar APP
8 -> Iniciar cenário
9 -> Parar cenário
0 -> Voltar para o menu principal
========================================
>> 5
List of devices attached
emulator-5558   device
```

Fonte – Autor, 2017

Figura 14 – Classes implementadas pela API desenvolvida.



Fonte – Autor, 2017

ser utilizado como exemplo de uso da API.

```
1  import time
2  from DeviceManagerAPI import DeviceManager
3
4  DM = DeviceManager("cenario-2", "10.0.0.2")
5
6  dispositivos = DM.getDevices()
7
8  for android in dispositivos:
9      DM.start_app(android, "br.ufc.great.matrixoperation/.MainActivity")
10     time.sleep(5)
11
12 cont = 0
13 while cont < 30:
14     print("\n Iniciando o teste numero {}".format(cont))
15
16     for android in dispositivos:
17         DM.exec_activity(android, "br.ufc.great.matrixoperation/.MainActivity
               ", "--es operation mul --ei size 500")
18
19     time.sleep(10)
20
21     cont += 1
22
23 for android in dispositivos:
24     android.getResults()
```

Código-fonte 1 – Exemplo de Script.

Na linha 2 a API é importada e é instanciada na linha 4 passando como argumentos o nome do cenário desejado e o IP da *cloudlet* do cenário. Na linha 6 é salvo na variável *dispositivos* uma lista de objetos da classe Android, onde cada *container* cliente representa um objeto da lista retornada pelo método *getDevices*. Na linha 8 são iniciadas as aplicações para cada dispositivo, o método *start_app* deve receber um objeto Android e a *activity* para iniciar a aplicação desejada usando o método *start_app* do objeto. Na linha 16, para cada dispositivo, é executado o método *exec_activity*, deve receber além do objeto Android os argumentos que serão usados na execução da *activity*, só então é executado o método *exec_run* do objeto Android. Por último, na linha 23, o método *getResults* da classe Android captura as informações do *log* do dispositivo, local onde os resultados dos experimentos executados estavam sendo guardados.

5.3 Outros detalhes importantes

O *script* apresentado na seção anterior 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 indicava o tamanho da matriz que seria gerada.

Para que isso seja possível, é preciso configurar a aplicação Android para aceitar *extras* e suportar esse tipo de operação, caso contrário só será possível executar aplicações por meio da interface VNC do dispositivo.

6 AVALIAÇÃO DA FERRAMENTA

Este capítulo descreve como foram executados os testes para avaliação da ferramenta desenvolvida e os resultados obtidos com eles.

6.1 Metodologia de experimentação

Para avaliar a ferramenta desenvolvidas, utilizou-se uma aplicação para o cálculo de multiplicação de matrizes. A métrica utilizada em todos os experimentos foi o tempo necessário para que a aplicação enviasse a matriz gerada para o *cloudlet* mais o tempo de processamento na *cloudlet* mais o tempo de recuperação do resultado. A Tabela 1 especifica os fatores e níveis usados nos experimentos realizados.

Tabela 1 – Fatores e níveis dos experimentos.

Fator	Nível
Tipo de rede móvel	LTE e UMTS
Velocidade das redes utilizadas (kbps)	LTE: *up* 58,000 *down* 173,000; UMTS: *up* 384,0 *down* 384,0
Quantidade de dispositivos móveis no cenário	2 e 4
Dimensão das matrizes utilizadas	500x500
Aplicação utilizada	MatrixOperation
Método da aplicação utilizado	br.ufc.great.matrixoperation/.MainActivity

Fonte – Produzido pelo autor

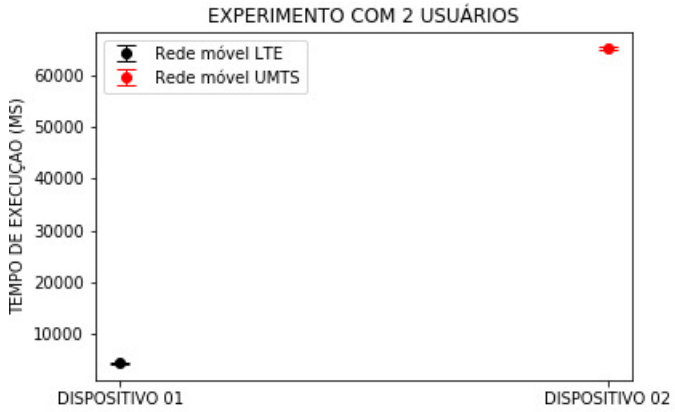
Foram utilizados dois cenários de testes: o primeiro era composto por dois dispositivos móveis, um utilizando rede móvel LTE e o outro a rede móvel UMTS, e um servidor executando o *MpOS Platform*; o segundo cenário era composto por quatro dispositivos móveis, dois deles utilizando móvel LTE e dois móvel UMTS, e um servidor. Em cada cenário, cada dispositivo móvel executou o método citado na Tabela 1 30 vezes, a quantidade pela facilidade de automação de testes provida pela API desenvolvida e também para obter um número maior de amostras e obter possíveis variações nos resultados. Em função de uma limitação da ferramenta, foi necessário que os dispositivos móveis e o servidor estivessem em máquinas físicas diferentes pois, quando executados na mesma máquina, os dispositivos móveis não sofriam a limitação de rede quando comunicavam-se com o servidor.

6.2 Resultados

Ao todo, nos dois cenários, o método da aplicação foi executado 180 vezes. Os resultados obtidos nas 30 execuções de cada dispositivo foram somados, as médias foram calculadas e a varição na média corresponde a um intervalo de confiança de 95%.

A Figura 15 apresenta o resultado dos testes feitos utilizando o primeiro cenário. Nela fica visível a diferença entre o tempo de execução do método da aplicação nos dois dispositivo que compunham o cenário. Nota-se também que houve uma mínima variação ao longo de cada execução do método da aplicação.

Figura 15 – Resultados dos testes com dois dispositivos móveis.



Fonte – Autor, 2017

Na Tabela 2 tem-se a média dos tempos de processamento no *cloudlet*, tempo de *donwload* e *upload* dos dispositivos mostrados no gráfico anterior. Com ela é fácil ver que a disparidade entre os resultados se deve ao fato do tempo médio de *download* e *upload* da rede móvel LTE ser muito maior do que a rede móvel UMTS, o tempo de processamento no *cloudlet* não apresentam grandes diferenças entre os dois dispositivos.

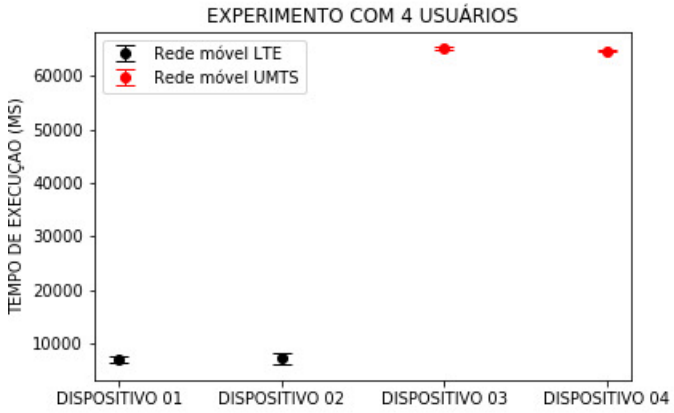
Tabela 2 – Média dos resultados do cenário com 2 usuários.

Dispositivo	Rede utilizada	Média do tempo de processamento no *cloudlet* (ms)	Média do tempo de *download* (ms)	Média do tempo de *upload* (ms)
Dispositivo 1	LTE	607,90 ± 24,16	894,0 ± 501,76	2.937,40 ± 480,40
Dispositivo 2	UMTS	621,83 ± 41,37	42.716,0 ± 56,96	21.509,96 ± 61,40

Fonte – Produzido pelo autor

Na Figura 16 é apresenta o resultado dos testes feitos utilizando o segundo cenário. Novamente os dispositivos que utilizavam a rede móvel LTE obtiveram um menor tempo de execução em relação aos que utilizavam a rede UMTS. Com quatro dispositivos é possível ver que a ocorrência de uma leve variação no tempo de execução dos Dispositivos 1 e 2.

Figura 16 – Resultados dos testes com quatro dispositivos móveis.



Fonte – Autor, 2017

Observando a Tabela 3 que contém, assim como a Tabela 2, a média dos tempos de processamento no *cloudlet*, tempo de *donwload* e *upload* para os dispositivos mostrados no gráfico anterior, novamente a o tempo médio de processamento no *cloudlet* foi bastante semelhante para os quatro dispositivos e também a rede móvel LTE obteve os menores tempos de *download* e *upload* em comparação a rede móvel UMTS.

Tabela 3 – Média dos resultados do cenário com 4 usuários.

Dispositivo	Rede utilizada	Média do tempo de processamento no *cloudlet* (ms)	Média do tempo de *download* (ms)	Média do tempo de *upload* (ms)
Dispositivo 1	LTE	654,36 ± 40,86	1025,0 ± 502,26	5.176,20 ± 1.497,56
Dispositivo 2	LTE	755,63 ± 587,64	1.581,66 ± 1.609,86	4.944,46 ± 1.932,68
Dispositivo 3	UMTS	664,06 ± 63,98	42.626,30 ± 56,90	21.797,13 ± 646,36
Dispositivo 4	UMTS	695,86 ± 151,93	42.226,66 ± 59,61	21.775,96 ± 707,92

Fonte – Produzido pelo autor

7 CONCLUSÃO E TRABALHOS FUTUROS

Cada vez mais os dispositivos móveis estão atrelados as tarefas do nosso dia a dia. Para que possamos usufruir desse fato cada vez mais, desenvolvedores e pesquisadores precisam de ferramentas que os auxiliem no desenvolvimento e teste de suas aplicações.

Sabendo disso, este trabalho propôs o desenvolvimento de uma ferramenta para facilitar a criação de cenários de testes e experimentação para aplicações que usam a técnica de *offloading*. Essa ferramenta deve possibilitar que os usuários criem e configurem cenários com múltiplos dispositivos móveis executando sobre diferentes condições de rede.

A solução descrita ao longo desta monografia implementa todos os objetivos propostos para este trabalho. A ferramenta desenvolvida está disponível para a comunidade e pode ser encontrada no GitHub no repositório *RenanData/androidTestBed*[1] e as imagens utilizadas podem ser baixadas do *DockerHub*. O usuário pode também visitar o site do MpOS[2], onde estão disponíveis exemplos de implementação e mais informações sobre o *framework*.

Os experimentos feitos nos cenários criados com a ferramenta apresentaram resultados esperados para os fatores e níveis utilizados. A rede móvel LTE, em função da sua maior taxa de transmissão em relação a rede móvel UMTS, obteve os melhores resultados quanto a velocidade de execução do método da aplicação utilizada.

A versão atual da ferramenta ainda é simples, visto que não é possível definir outras configurações para o emulador como quantidade de memória e utilização de CPU. Uma grande limitação é o fato de que quando servidor e dispositivos móveis são executados na mesma máquina não ocorre limitação na rede dos mesmos.

Como trabalhos futuros pretende-se permitir que o usuário tenha um maior controle sobre os *containers* clientes, permitindo que ele defina fatores como a velocidade da rede que será utilizada além da quantidade de memória e CPU disponível para o emulador. Pretende-se encontrar um meio que permita que clientes e servidores, mesmo quando executados na mesma máquina, executem as condições de rede desejadas. Há interesse também em realizar mais experimentos com uma maior variação de fatores e níveis.

[1] https://github.com/RenanData/androidTestBed
[2] http://mpos.great.ufc.br/

REFERÊNCIAS

COSTA, P. B.; REGO, P. A.; COUTINHO, E. F.; TRINTA, F. A.; SOUZA, J. N. de. Uma análise do impacto da qualidade da internet móvel na utilizaç ao de cloudlets. **Anais do 32º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2014**, 2014.

COSTA, P. B.; REGO, P. A. L.; ROCHA, L. S.; TRINTA, F. A. M.; SOUZA, J. N. de. Mpos: A multiplatform offloading system. In: **Proceedings of the 30th Annual ACM Symposium on Applied Computing**. New York, NY, USA: ACM, 2015. (SAC '15), p. 577–584. ISBN 978-1-4503-3196-8. Disponível em: <http://doi.acm.org/10.1145/2695664.2695945>. Acesso em: 15 maio. 2017.

FERNANDO, N.; LOKE, S. W.; RAHAYU, W. Mobile cloud computing: A survey. **Future Generation Computer Systems**, v. 29, n. 1, p. 84 – 106, 2013. ISSN 0167-739X. Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X12001318>. Acesso em: 25 mar. 2017.

HOANG, D. T.; LEE, C.; NIYATO, D.; WANG, P. A survey of mobile cloud computing: architecture, applications, and approaches. **Wireless Communications and Mobile Computing**, v. 13, n. 18, p. 1587–1611, 2013. Disponível em: <http://dblp.uni-trier.de/db/journals/wicomm/wicomm13.html#HoangLNW13>. Acesso em: 15 jun. 2017.

KEMP, R.; PALMER, N.; KIELMANN, T.; BAL, H. Cuckoo: A computation offloading framework for smartphones. In: ____. **Mobile Computing, Applications, and Services: Second International ICST Conference, MobiCASE 2010, Santa Clara, CA, USA, October 25-28, 2010, Revised Selected Papers**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 59–79. ISBN 978-3-642-29336-8. Disponível em: <http://dx.doi.org/10.1007/978-3-642-29336-8_4>. Acesso em: 13 jun. 2017.

KOSTA, S.; AUCINAS, A.; HUI, P.; MORTIER, R.; ZHANG, X. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In: GREENBERG, A. G.; SOHRABY, K. (Ed.). **INFOCOM**. IEEE, 2012. p. 945–953. ISBN 978-1-4673-0773-4. Disponível em: <http://dblp.uni-trier.de/db/conf/infocom/infocom2012.html#KostaAHMZ12>. Acesso em: 8 abr. 2017.

KOVACHEV, D.; CAO, Y.; KLAMMA, R. Mobile cloud computing: A comparison of application models. **CoRR**, abs/1107.4940, 2011. Disponível em: <http://arxiv.org/abs/1107.4940>. Acesso em: 15 abr. 2017.

KUMAR, K.; LIU, J.; LU, Y.-H.; BHARGAVA, B. A survey of computation offloading for mobile systems. **Mobile Networks and Applications**, v. 18, n. 1, p. 129–140, 2013. ISSN 1572-8153. Disponível em: <http://dx.doi.org/10.1007/s11036-012-0368-0>. Acesso em: 15 jun. 2017.

RAHIMI, M. R.; REN, J.; LIU, C. H.; VASILAKOS, A. V.; VENKATASUBRAMANIAN, N. Mobile cloud computing: A survey, state of art and future directions. **Mobile Networks and Applications**, v. 19, n. 2, p. 133–143, 2014. ISSN 1572-8153. Disponível em: <http://dx.doi.org/10.1007/s11036-013-0477-4>. Acesso em: 25 mar. 2017.

REGO, P. A. L. **Applying Smart Decisions, Adaptive Monitoring and Mobility Support for Enhancing Offloading Systems**. Tese (Doutorado) — Universidade Federal do Ceará, Fortaleza, Brasil, 12 2016.

SATYANARAYANAN, M.; BAHL, V.; CACERES, R.; DAVIES, N. The case for vm-based cloudlets in mobile computing. **IEEE Pervasive Computing**, IEEE, November 2009. Disponível em: <https://www.microsoft.com/en-us/research/publication/the-case-for-vm-based-cloudlets-in-mobile-computing/>. Acesso em: 15 abr. 2017.