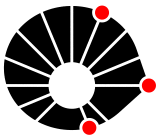

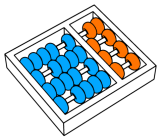

Simulação de Modelos de Provisionamento e Migração de Recursos nas Bordas da Internet Utilizando o Simulador *MobFogSim*

Pietro Ruy Pugliesi *Edmundo Madeira*

Relatório Técnico - IC-PFG-22-13
Projeto Final de Graduação
2022 - Julho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Sumário

1 Introdução **3**

2 Fundamentos **4**
2.1 Nuvem e Névoa 4
2.2 Virtualização, Máquina Virtual e *Container* 5
2.3 Migração 6
2.4 *MobFogSim* 6
2.4.1 Políticas de Migração 7
2.4.2 Estratégia de Migração 7
2.4.3 Política de Migração da Máquina Virtual/*Container* 7
2.4.4 Tipo de Ponto de Migração 8

3 Revisão Bibliográfica **9**

4 Justificativa **9**

5 Objetivos **10**

6 Desenvolvimento do Trabalho **10**
6.1 Antes da Execução das Simulações Propostas 11
6.2 Parâmetros de Entrada do *MobFogSim* 11
6.2.1 Parâmetros Sobre Posição, Direção e Velocidade dos Usuários 12
6.2.2 Configurações Dentro do Código-Fonte 12
6.2.3 Parâmetros de Linha de Comando do *MobFogSim* 13
6.3 Cenário das Simulações Executadas 14
6.4 Parâmetros de Linha de Comando Utilizados nas Simulações 16
6.5 Execução das Simulações 18
6.5.1 Parâmetros Utilizados na Simulação Base 18

7 Resultados **21**
7.1 Resultados de Latência 21
7.2 Discussão dos Resultados de Latência 21
7.2.1 Medida Base 21
7.2.2 Parâmetro #3: Tipo de Ponto de Migração 21
7.2.3 Parâmetro #4: Estratégia para Escolha da Próxima *Cloudlet* em uma Migração 22
7.2.4 Parâmetro #7: Política de Migração da Máquina Virtual/*Container* 22
7.3 Resultados de *Downtime* 23
7.4 Discussão dos Resultados de *Downtime* 23
7.4.1 Parâmetros #3: Tipo de Ponto de Migração, e #4: Estratégia Para Escolha da Próxima *Cloudlet* em uma Migração 23
7.4.2 Parâmetro #7: Política de Migração da Máquina Virtual/*Container* 23
7.5 Medidas Adicionais para os Parâmetros #3 e #4 24

7.5.1 Medidas Adicionais para o Parâmetro #3 Igual a 1: Ponto de Migração Baseado na Velocidade do Usuário 24
7.5.2 Medidas Adicionais para o Parâmetro #4 Igual a 1: Escolha da Próxima *Cloudlet* a Mais Próxima ao Usuário 24
7.5.3 Possível Justificativa para a Dispersão Continuar Alta 24
7.6 Medida com os Melhores Valores 24
7.7 Resultados do Ponto de Vista dos Usuários 25

8 Conclusões **25**

9 Agradecimentos **27**

Simulação de Modelos de Provisionamento e Migração de Recursos nas Bordas da Internet Utilizando o Simulador *MobFogSim*

Pietro Ruy Pugliesi* Edmundo Madeira*

Resumo

Com a expansão e popularização da Internet tem se tornado cada vez mais usada a arquitetura de computação em névoa, que utiliza mini-servidores (*cloudlets*) responsáveis pela alocação dos recursos de armazenamento e processamento dos dados dos usuários, sob a forma de máquinas virtuais ou *containers*, na borda da rede. Mesmo assim, os usuários podem experimentar latências e perdas de conexão. Para melhorar isso, faz-se necessário adaptar a alocação desses recursos da rede conforme o cenário: mais ou menos usuários, com diferentes usos de rede e movimentação. A proposta deste trabalho é investigar se alterações nas políticas de migração de recursos entre as *cloudlets* podem diminuir as latências e perdas de conexão experimentadas pelos usuários. A validação dessa proposta foi feita utilizando o simulador *MobFogSim*.

1 Introdução

Não é desconhecida a popularização e expansão da Internet como meio de comunicação: cada vez mais dispositivos (fixos ou móveis) estão conectados a ela, trocando informações, seja em redes fixas (*Wi-Fi*) ou redes móveis (4G ou o novo 5G). Nessas redes, há servidores — dispositivos que proveem os recursos, ou seja, oferecem serviços computacionais para outros dispositivos na rede; e os usuários que acessam esses serviços. Os servidores, por exemplo, podem oferecer serviços de *Cloud Computing* (computação na nuvem): cenário em que os processamentos dos dados e computação são feitos utilizando recursos de outros computadores que não os que estão acessando o serviço, sem preocupação com localização física ou investimentos em *hardware*[1] [13]. Nesse cenário de *Cloud Computing* os dados têm que percorrer todo o caminho entre cada usuário e o servidor do serviço que cada um está acessando — da borda da rede (usuário), passando até núcleo da rede (servidores) e de volta, constantemente. Isso causa alta latência (e até perdas de conexão) e aumento do tráfego no núcleo da rede, sendo uma limitação da abordagem [5]. Uma maneira de amenizar esses problemas é a arquitetura de *Fog Computing* (computação em névoa): levar o processamento para mais perto dos usuários, na borda da rede com eles[1].

*Instituto de Computação, Universidade Estadual de Campinas, 13081–970 Campinas, SP.

[1]Para explicação mais detalhada, vide Seção 2.1.

Nessa arquitetura, há mini servidores, ou *cloudlets*, nós na rede responsáveis pelo armazenamento e processamento dos dados [5], reduzindo assim o uso dos servidores na nuvem e o tráfego no núcleo da rede. Porém, a mobilidade dos usuários pode reduzir essas vantagens, visto que a distância entre o usuário e a *cloudlet* (o nó) que está fazendo o processamento dos seus dados pode variar. Uma maneira de amenizar o problema é fazer a migração do serviço de névoa responsável por cada usuário de maneira estratégica entre as *cloudlets*, para que o nó responsável esteja sempre próximo o suficiente do usuário [12].

Para simular essa migração de recursos entre as *cloudlets* (nós) de acordo com a mobilidade dos usuários, utilizou-se o simulador ***MobFogSim***[12][2]: um simulador de alocação de recursos na rede, com uso de computação em névoa, para usuários com mobilidade. Nele, temos parâmetros como: localização dos usuários e provedores de rede; quantidade deles; políticas de migração de recursos, latências, larguras de banda para comunicação, entre outros.

No presente trabalho são apresentados estudos e simulações com o *MobFogSim* com o objetivo de reduzir latência e quedas experimentadas pelos usuários no cenário de *Fog Computing*, através de mudanças nas estratégias de migração de recursos de rede entre as *cloudlets*:

1. Tipo de ponto de migração dos máquinas virtuais/*containers*, entre fixo ou calculado com base na velocidade do usuário;

2. Critério para escolha da próxima *cloudlet* em uma migração;

3. Tipo de implementação do gerenciamento de recursos de rede para o usuário (máquina virtual ou *container*) e de que maneira a migração desses recursos é feita (*Cold* ou *Live*).

2 Fundamentos

Nesta seção são apresentados conceitos básicos para entender o funcionamento do *MobFogSim* e as simulações que serão feitas com ele.

2.1 Nuvem e Névoa

Segundo [13], uma definição possível de Computação em Nuvem (*Cloud Computing*) é "um conjunto de recursos como capacidade de processamento, armazenamento, conectividade, plataformas, aplicações e serviços disponibilizados na Internet". Ou seja, disponibilização de recursos computacionais através da Internet, de modo transparente para os usuários.

Uma limitação dessa abordagem é notada em latência, quedas de conexão e perdas de pacotes, pois os dados dos usuários são processados nos servidores de nuvem, muito distantes deles, numa abordagem centralizada. Uma solução proposta é a arquitetura de Computação em Névoa (*Fog Computing*) [3]: "Computação em Névoa é uma plataforma

[2] Mais detalhado em [5], [7].

altamente virtualizada que provê serviços de computação, armazenamento e rede entre os usuários e os tradicionais Data Centers da Computação em Nuvem" [3][3]. Isso é feito através da adição de nós na rede, papel exercido pelas *cloudlets* do presente trabalho, entre os usuários e a nuvem que, como vemos na Figura 1, traz esse acesso para mais próximo dos usuários.

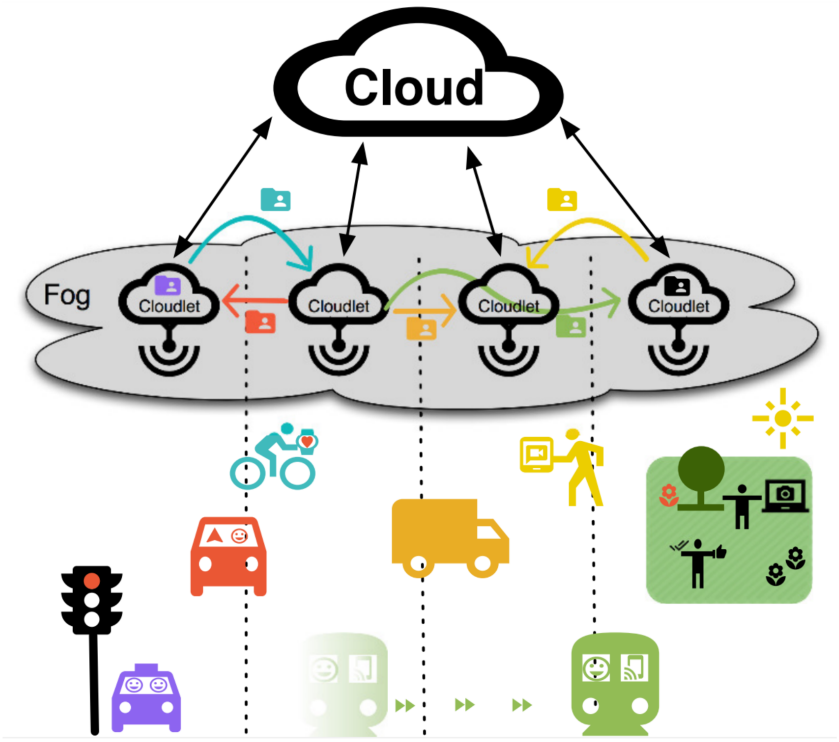

Figura 1: Ilustração do cenário de usuários acessando serviços de computação em névoa. Retirado de [*GONÇALVES, D.* [6]]

2.2 Virtualização, Máquina Virtual e *Container*

Virtualização consiste numa camada de abstração sobre um *hardware* e é feita por um programa específico chamado de *Hypervisor* [4]. O *MobFogSim* trabalha com duas implementações diferentes de virtualização: utilizando máquinas virtuais (VMs), ou utilizando *containers*.

No primeiro caso, as aplicações nas *cloudlets* responsáveis pelo processamento de dados e armazenamento de usuário são instaladas num sistema operacional completo, virtual, "convidado", e a interface entre esse sistema operacional do "anfitrião" (a *cloudlet*) em que roda — o sistema que faz a comunicação com o *hardware* da *cloudlet* — é um programa chamado de *Hypervisor* [4]. Ou seja, temos o sistema operacional base (da *cloudlet*), que

[3]Tradução livre.

faz a comunicação com o seu *hardware*, e, sobre ele, uma máquina virtual com outro sistema operacional instalado, e as aplicações são executadas dentro dos sistemas operacionais convidados. Esse cenário é chamado de virtualização de *hardware* [4].

Um *container*, por sua vez, não tem um sistema operacional virtualizado sobre o qual são instaladas as aplicações; ele é um ambiente isolado criado dentro do sistema operacional do anfitrião (*host*, a *cloudlet*), que executa a aplicação desejada, num cenário que é chamado de virtualização de *sistema operacional* [4].

Essa característica do *MobFogSim* é detalhada por *[PULIAFITO et al.]* em [12]: "[foi considerado que os serviços em névoa simulados do *MobFogSim*] estão disponíveis na forma de ambiente de software virtualizado, como VMs ou *containers*, que encapsulam o acesso a recursos computacionais e de rede"[4]. Ou seja, vemos que o *MobFogSim* oferece suporte a ambos, sendo possível escolher, por um argumento da execução do programa[5], qual implementação usar. É importante notar que é utilizada ou uma ou outra abordagem: ou todos os aplicativos de rede das *cloudlets* são implementados e migrados como máquinas virtuais, ou todos como *containers*.

2.3 Migração

Num cenário de *Fog Computing*, as aplicações usam *cloudlets* (nós) para armazenar e processar dados [12]. Os dados e processamento relativos a um usuário são feitos em uma máquina virtual, ou *container* (mais detalhes na Seção 2.2). No caso em que o usuário está em movimento, essa máquina virtual/*container* deve se mover juntamente com ele, sendo migrada entre as *cloudlets* para manter menor latência e tentar evitar perdas de conexão.

2.4 *MobFogSim*

O *MobFogSim* é um simulador de código aberto[6] de computação em névoa com suporte a mobilidade e fatiamento da rede, proposto por *[PULIAFITO et al.]*, como extensão do `iFogSim` [8], um simulador de computação em névoa.

Nele, podemos fazer simulações de movimentação de usuários numa rede em cenário de *Fog Computing*, com configurações de *cloudlets* gerenciando os recursos de rede modelados por meio de máquinas virtuais ou *containers*; de número de pontos de acesso e *cloudlets*; posições de cada um destes; especificações de *hardware* dos usuários, pontos de acesso e *cloudlets*, entre muitas outras configurações e parâmetros. Mais detalhes são dados na Seção 6.2.2. Após as simulações, temos vários resultados de saída como: latência, *downtime*, energia consumida pelas *cloudlets*, tempos e posições em que ocorreram migrações, entre outros.

O *MobFogSim* também possui [5] integração com o simulador *Simulation of Urban Mobility (SUMO)* [*BEHRISCH et al.* [2]]: os dados dos usuários, usados como entrada do *MobFogSim*, são arquivos de extensão `csv`, com informações sobre posição x, y, direção (em

[4]Tradução livre.
[5]O parâmetro número 7. Conferir (Cf.) Tabela 1.
[6]Disponível em `https://github.com/diogomg/MobFogSim`. Acessado em junho de 2022.

rad) e velocidade (m/s) do usuário a cada segundo. Tais dados `csv` podem vir diretamente da saída do *SUMO*, ou criados seguindo o formato esperado.

Além desses parâmetros acerca dos usuários, algumas outras configurações do simulador estão definidas no código-fonte deste, e outros são passados nos argumentos da sua execução, em linha de comando, detalhados na Tabela 1. Os parâmetros são apresentados na Seção 6.2.

2.4.1 Políticas de Migração

A maneira com que a migração dessas VMs/*containers* — contendo os dados e processamento — é feita depende do *software* implementado nas *cloudlets* [12]. Desse modo, podemos ter cada *cloudlet* responsável pelas suas decisões de migração. Três dos parâmetros relativos à política de migração das *cloudlets* serão discutidos nesse trabalho, para análise de como impactam nas latências e quedas experimentadas pelos usuários:

1. Estratégia para escolha da próxima *cloudlet* em uma migração;
2. Política de migração da máquina virtual;
3. Tipo de ponto de migração;

A seguir, detalhamos mais cada um deles.

2.4.2 Estratégia de Migração

Segundo [*PULIAFITO et al.* [12]], se as migrações são permitidas e o usuário está se movendo na direção de uma *cloudlet* disponível, e está conectado à rede, havendo comunicação das *cloudlets* entre si e com os usuários, o algoritmo decide que pode, e deve, ser feita a migração. A escolha do nó destino dessa migração de VM ou *container* é resultado de uma de três possíveis estratégias de migração:

1. É escolhida a *cloudlet* que oferece a menor latência para o usuário;
2. É escolhida a *cloudlet* que está mais próxima ao usuário;
3. É escolhida a *cloudlet* que está conectada ao ponto de acesso mais próximo ao usuário;

2.4.3 Política de Migração da Máquina Virtual/*Container*

Uma vez decidido que será feita a migração e para onde, há duas [*PULIAFITO et al.* [12]] abordagens de migração das VMs ou *containers*:

1. Migração "Fria" (*Cold*): Primeiro a VM ou *container* é parada ("congelada"); depois o seu estado é salvo e transferido para a *cloudlet* de destino; e apenas quando for totalmente transferida é que a VM é retomada na nova *cloudlet*. É a abordagem com maior *downtime* (intervalo de tempo em que a VM/container não está funcional, ou seja, o usuário fica sem acesso ao serviço);

2. Migração *Post-copy*: É uma migração "*live*", "ao vivo": A VM/*container* continua rodando enquanto seu estado é transferido para a *cloudlet* destino, e só é parada para transmissão de uma quantidade mínima do seu estado geral: quando a migração começa, ela é parada e seus estados de *CPU* e registradores são transferidos para a *cloudlet* destino, e a sua execução é retomada ali. Depois, o *MobFogSim* usa a a estratégia "preguiçosa" (*lazy*): quando a VM/*container* resumido tenta acessar uma página de memória que não existe, ocorre um *page fault*. Então, o servidor migra essa página faltante de memória para o destino.

O simulador *MobFogSim*, no momento da escrita desse trabalho, suporta três configurações de política de migração[7]:

1. Migração *Cold* com implementação de *máquina virtual*;
2. Migração *Cold* com implementação de *container*;
3. Migração *Post-Copy* (*Live*) com implementação de *container*;

2.4.4 Tipo de Ponto de Migração

Agora que a *cloudlet* decidiu para onde que deve ser feita a migração da máquina virtual/*container* e como será feita essa migração, há duas abordagens possíveis para o escolha da localização do ponto de migração, isso é, o ponto onde, quando o usuário chega a ele, o processo de migração é iniciado. Esse ponto se encontra entre o ponto de acesso (AP, no centro da Figura 2) a que o usuário está conectado (por exemplo, a antena de rede 4G) e a fronteira de cobertura da rede desse mesmo ponto de acesso [12] ("Fonteira do sinal wireless" da figura), e está representado na figura como a linha tracejada em vermelho. A localização desse ponto de migração pode ser decidida de duas maneiras: localização definida estática ou dinâmica.

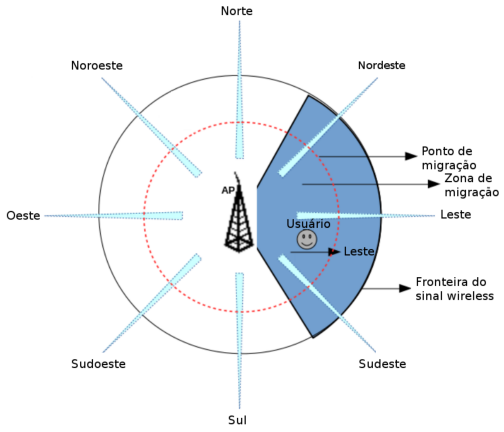

Figura 2: Ilustração do cenário de usuários acessando serviços de computação em névoa. Retirado de [*GONÇALVES, D.* [6]]

[7]Cf. parâmetro número 7 da Tabela 1.

No tipo estático, o ponto de migração é fixado num local do mapa, independentemente de outros parâmetros. Por exemplo, esse ponto pode ser definido para 70% distância entre a localização do ponto de acesso e a da estimativa do ponto em que o usuário cruzará a fronteira de cobertura da rede desse ponto de acesso.

No tipo dinâmico, a posição do ponto de migração para um usuário específico é decidida com base em outros parâmetros, como velocidade desse usuário. Novamente, quando o usuário chega a esse ponto, a migração começa, mas a posição desse ponto é diferente para usuários diferentes, e é decidida pelo algoritmo.

3 Revisão Bibliográfica

O *MobFogSim* foi proposto em [*PULIAFITO et al.* [12]]. Nesse artigo, o simulador é descrito em detalhes, é mostrado como o *MobFogSim* foi criado como extensão do *iFogSim*; feita descrição dos diferentes parâmetros de abordagem de ponto de migração, estratégias e políticas de migração; detalhes de implementação; validação e apresentação de outros simuladores de computação em nuvem. Esse artigo foi a principal referência para o presente trabalho, devido à apresentação da terminologia envolvida e conceitos necessários para entender o funcionamento do simulador. Nele, é mostrado que o *MobFogSim* fornece uma boa base para simulação de aplicações de computação em névoa em que os usuários estão em movimento e é necessário mover dados entre as *cloudlets*; porém, no artigo, ele não é utilizado como ferramenta de análise de um cenário real, com o objetivo de melhorar a experiência dos usuários, por exemplo, reduzindo latência e *downtime*.

Além desse artigo, também foram utilizados os artigos: [*GONÇALVES et al.* [5] e [7]] como fontes secundárias para este presente trabalho. Estes dois últimos artigos são voltados ao estudo das simulações do *MobFogSim* utilizando o recurso de fatiamento de rede, que não foi utilizado em nosso trabalho, uma vez que estamos aqui analisando outra situação: como o as políticas e estratégias de migração afetam latência e perdas de conexão.

Além desses, também há [10], que estuda alocação de recursos em redes com usuários em movimento, mas propõe um novo algoritmo de fatiamento de rede baseado em análise de atrasos — novamente, não é o que estudamos aqui; [1], que propõe e analisa um algoritmo de migração de *containers* baseado em análise de mobilidade; e [10] que propõe um algoritmo baseado em "aprendizado com reforço" [10] para dar suporte a migração de serviços com baixa latência. Mesmo que esses estudos não sejam voltados a fatiamento, estão sendo propostos novos algoritmos, não sendo utilizado um simulador de rede como ferramenta de análise para estudar como as diferentes políticas de migração afetam latência e *downtime* para os usuários.

4 Justificativa

Como mostrado na Introdução (Seção 1), a arquitetura de névoa foi criada para fazer o processamento dos dados de usuários de serviços de computação em nuvem, entre outras vantagens, com menor latência e tempo sem conexão (*downtime*). A depender do serviço

que o usuário esteja utilizando, pode ser preferível diminuir ou um, ou outro, ou os dois[8]. Porém, criar uma infraestrutura para realizar experimentos para estudo desse cenário na vida real seria muito custoso, tanto para implementar a rede de névoa (*cloudlets*) para dar cobertura de rede aos usuários e gerenciar seus recursos de rede, quanto para fazer medições. Portanto, fizemos esses estudos com o simulador *MobFogSim*.

Conforme descrito em [12], o *MobFogSim* foi criado como extensão do *iFogSim* para oferecer suporte a mobilidade e fatiamento de redes, algo não encontrado em outros simuladores. Devido à falta de trabalhos que utilizem o *MobFogSim* como ferramenta de análise sobre uma simulação de cenário real, com o objetivo de estudar e propor melhorias à conexão dos usuários aos serviços que estão utilizando, por via de modificações em parâmetros relativos à migração de máquinas virtuais/*containers* entre as *cloudlets*, foi proposto o presente trabalho com este objetivo: estudar como reduzir latências e perdas de conexão apenas modificando esses parâmetros, numa aproximação de cenário real.

5 Objetivos

O objetivo deste trabalho é, utilizando o *MobFogSim*, analisar se podemos reduzir latências e quedas (*downtime*) experimentadas pelos usuários de um mesmo veículo (ou seja, com padrão de movimentação idêntico), alterando os parâmetros das *cloudlets*: tipo de ponto de migração, estratégia para escolha da próxima *cloudlet*, e política de migração de máquina virtual/*container*.

É esperado que, com os resultados obtidos, possamos ver como esses parâmetros relativos às *cloudlets* influenciam em latência e *downtime*, se há algum deles que não influencia, se o uso dos que retornaram melhores resultados, combinados, melhoram mais ainda a resposta e, assim, saber uma maneira de melhorar latência e *downtime* para os usuários de serviços em rede num cenário real. Além disso, procuramos ter dados para analisar a resposta do simulador em suas configurações atuais, e, se possível, propor melhorias futuras para ele, uma vez que ele possui algumas limitações, como somente ser possível definir essas políticas de migração — as modificadas no presente estudo — para todas as *cloudlets* de uma vez, ao invés de individualmente; e esses parâmetros serem fixos para a *cloudlet* a que o usuário está conectado num determinado momento, independentemente do aplicativo que o usuário esteja utilizando.

6 Desenvolvimento do Trabalho

Para realizar esse estudo, será primeiro feita uma medida base, à qual serão comparados os resultados obtidos modificando os parâmetros relativos a: tipo de ponto de migração, estratégia para escolha da próxima *cloudlet*, e política de migração de máquina virtual/*container*; um parâmetro por vez, e uma vez para cada opção — por exemplo, temos três opções de critério de escolha do próximo nó numa migração. Ao fim, tomaremos

[8] Por exemplo, menor latência em um *streaming* de vídeo; menor *downtime* em acesso a uma *VPN* a que o usuário necessite estar conectado; os dois menores em um jogo *online* ou conexão remota a uma área de trabalho virtual.

os valores dos parâmetros que mostraram os resultados de menor média de latência para os usuários, e os que mostraram os de menor *downtime* (tempo total em que a máquina virtual/*container* está fora do ar, o que se traduz em tempo em que o usuário conectado está sem acesso ao serviço) e faremos simulações com esses valores, para verificar o máxima melhora possível em latência e *downtime* modificando esses três parâmetros.

Para realizar as simulações, primeiro foi necessário instalar as dependências do simulador e fazê-lo funcionar na máquina, depois entender como o simulador funciona (Seção 6.1); quais os três tipos de parâmetros de entrada e como modificá-los para o cenário que desejamos simular (6.2); entender esse cenário referente às simulações executadas (6.3); configurar os parâmetros de linha de comando para as simulações planejadas (6.4); para depois fazer as simulações propostas (6.5).

Também foi necessário aprender *Shell Script* para entender o arquivo `script.sh` do repositório, que executa as simulações em lote. Para facilitar a execução das várias simulações planejadas e organizar os arquivos de saída, foi criado um *script* personalizado, utilizando o outro já disponibilizado como base.

Por fim, foi feito também um tratamento estatístico dos resultados de latência e *downtime* obtidos nas simulações executadas. Todo esse processo é descrito nos próximos itens. Os resultados são apresentados na Seção 7.

6.1 Antes da Execução das Simulações Propostas

Para executar o simulador, baixamos o *MobFogSim* pelo seu repositório disponível no *GitHub*[9]. Para o simulador funcionar, é preciso ter o `JAVA SDK`[10] instalado. Também foi instalada a *IDE* `Eclipse`[11] para leitura e análise do código-fonte. Depois, para executar uma simulação de teste, foram seguidas as instruções no arquivo `README.md` disponível no repositório baixado.

Uma vez entendido o funcionamento do simulador para o cenário proposto e saída analisada, foi necessário modificar os parâmetros de entrada do simulador para o cenário que desejamos simular.

6.2 Parâmetros de Entrada do *MobFogSim*

Os parâmetros para descrever uma simulação no *MobFogSim* estão separados em três lugares: os relativos à posição, direção e velocidade dos usuários estão dentro do diretório `input` da pasta do programa baixado (Seção 6.2.1); alguns já vêm configurados para algum valor dentro do próprio código-fonte do programa (6.2.2); e outros[12], os alterados para cada simulação do trabalho, são passados na execução do programa (em linha de comando do terminal ou *script shell*) (6.2.3). A seguir, detalhamos mais esses três tipos de parâmetros e como foram modificados para as simulações executadas.

[9] `https://github.com/diogomg/MobFogSim`. Acessado em junho de 2022.
[10] `https://java.com/en/download/`
[11] `https://www.eclipse.org/downloads/`
[12] Cf. Tabela 1.

6.2.1 Parâmetros Sobre Posição, Direção e Velocidade dos Usuários

Os arquivos que contêm a posição, direção e velocidade dos usuários devem estar dentro da pasta `input` do projeto. Esses arquivos, um para cada usuário, têm os dados de tempo (em segundos), direção (em *rad*), posição x e y, em m e velocidade, em m/s, de cada um deles, em formato `.csv`[13] Esses dados podem ser o resultado de saída do *SUMO*, e nesse trabalho foi utilizado o arquivo de entrada `1702log.csv`, que já vem no repositório, como padrão de mobilidade para todos os usuários, uma vez que todos os passageiros de um ônibus, nosso planejamento inicial de cenário a simular, têm o movimento igual. Como queremos simular um cenário com vários passageiros desse mesmo ônibus — ou seja, com movimentação idêntica, foram feitas cópias deste arquivo: cada arquivo é correspondente a um passageiro do ônibus. Os arquivos dessa pasta foram nomeados `1.csv`, `2.csv` ... `40.csv`.

Dentro dessa pasta `input` também deve haver um arquivo `ordem_entrada.csv`, com um número indicando a quantidade de arquivos com dados de movimentação dessa pasta que devem ser lidos: 1 para 1 usuário (será lido 1 arquivo de entrada), 2 para 2 usuários (serão lidos 2), etc. Este arquivo também foi modificado para a simulação planejada com 40 usuários.

Da maneira que o código-fonte foi escrito, esse arquivo deve ser o último da ordenação por nome da pasta `input`: nomeando os arquivos de entrada dos usuários como foi feito, `1.csv`, `2.csv`, etc. já é suficiente.

A ideia inicial era fazer simulações para 1, 2, 5, 10 e 40 usuários de movimento idêntico, como num ônibus. Porém, o simulador tem limitação em código para simulações com mais de 7 usuários[14]. Além disso, no cenário proposto, o simulador também passou a demorar um tempo muito longo para executar simulações com 3 ou mais usuários na máquina utilizada, um MacBook Pro modelo 2017: mais de meia hora, o tempo limite de execução de uma simulação definido no código-fonte[15]. Fora isso, também pretendíamos ter o tempo máximo para a execução de uma simulação de 1 hora, devido ao grande número de simulações a serem executadas para o presente trabalho. Ou seja, somente foi possível rodar simulações para 1 ou 2 usuários na máquina utilizada.

Uma vez que o simulador dá os resultados de latência e *downtime* para cada usuário das simulações, temos medidas de latência e *downtime* para o usuário único da primeira, e também para cada um dos dois usuários da outra simulação.

6.2.2 Configurações Dentro do Código-Fonte

Outras configurações da simulação estão dentro do código-fonte do projeto, no arquivo `AppExample.java`, como: configuração de localização dos pontos de acesso (*APs, Access Points*) e *cloudlets* e cobertura destes, configurações de *hardware* dos usuários, *AP*s e *cloudlets* (como tamanho das *VMs* alocadas nas *cloudlets* para cada usuário, banda de *downlink* e *uplink*, entre outros), e muitas outras.

[13] *Comma Separated Values*, no nosso caso, valores separados por espaços.
[14] As constantes de valores máximos e mínimos estão no arquivo `MaxAndMin.java` do código-fonte.
[15] Também no arquivo `MaxAndMin.java`.

Não foi feita, porém, nenhuma modificação nos parâmetros do código fonte do simulador, apenas redução do `log` impresso na linha de comando e correções de localização de salvamento dos arquivos de saída.

6.2.3 Parâmetros de Linha de Comando do *MobFogSim*

Os parâmetros para execução do programa, que se dá por linha de comando em terminal, estão descritos e detalhados na Tabela 1. Esses são os parâmetros principais, e entre eles estão os que são alterados a cada simulação executada para o presente trabalho.

O planejamento é executar, para 1 e depois 2 usuários, uma simulação base com os parâmetros mostrados na Tabela 3, e depois alterar um parâmetro por vez, entre os #3, #4 e #7, e comparar os resultados com a medida base para verificar se, e como eles interferem nos resultados de latência e *downtime*; e, ao final, executar uma simulação com os valores dos parâmetros desses parâmetros que retornaram os melhores resultados, para chegarmos à melhor configuração desses três parâmetros para latência e *downtime*.

Nota sobre o parâmetro *Seed*: o *Seed* é um número inteiro utilizado para inicializar o gerador de números pseudo-aleatórios. Se o valor do *Seed* é conhecido, os valores gerados também o serão[16].

[16] "[...]to generate random number sequences, computers rely on algorithms known as "pseudo-random number generators" or PRNG.A critical component of the functions and quality of PRNG is "seeding", whereby a number or vector "seed" is used to initialize the PRNG to produce a random number sequence. Effectively, if the seed values are known, the entire sequence generated by a PRNG will be known, and the seeding process is a critical point of vulnerability for cybersecurity purposes"[9].

Tabela 1: Parâmetros de linha de comando do *MobFogSim*

Número do Parâmetro	**Parâmetro**	**Valores Permitidos**
1	Migração permitida ou não	0 (não permitida) ou 1 (permitida)
2	*Seed* para geração de números aleatórios	Número inteiro maior que 0
3	Tipo de ponto de migração	0 (ponto de migração fixo) ou 1 (ponto de migração baseado na velocidade do usuário)
4	Estratégia para escolha da próxima *cloudlet* em uma migração	0 (escolhe a *cloudlet* com menor latência para o usuário); 1 (escolhe a *cloudlet* que está mais próxima do usuário); e 2 (escolhe a *cloudlet* conectada ao ponto de acesso que está mais próximo ao usuário)
5	Número de usuários	Número inteiro maior que 0
6	*Bandwidth* (largura de banda) base para comunicação entre as *cloudlets*, em *Mbps*	Número inteiro maior que 0
7	Política de migração da máquina virtual/*container*	0 (É utilizada Máquina Virtual, e a migração é do tipo *Cold*); 1 (É utilizado *container*, e a migração é do tipo *Cold*); ou 2 (É utilizado *container*, e a migração é do tipo *Live*)
8	Número de segundos para as predições	Número inteiro maior ou igual a 0
9	Imprecisão da medição, em metros (m)	Número inteiro maior ou igual a 0
10	Latência base de rede entre as *cloudlets*, em milissegundos (ms)	Número inteiro maior que 0

6.3 Cenário das Simulações Executadas

Nessa seção oferecemos uma descrição do cenário simulado e descrição das características de pontos de acesso, *cloudlets* e usuários definidos no código fonte e arquivos de entrada de padrão de movimentação dos usuários. Como mostrado na Seção 6.2.2, essas configurações não foram modificadas, sendo utilizadas nas simulações com os valores que vieram com o projeto baixado do repositório.

A configuração do mapa da rede — tamanho, posicionamento e alcance da cobertura dos pontos de acesso e *cloudlets* — são definidos no código fonte do projeto. O padrão de movimentação dos usuários é definido nos arquivos `.csv` de entrada na pasta `input` do simulador.

Na Figura 3 apresentamos um mapa em escala com o cenário da rede utilizado para as simulações. Temos uma área de 16.000 $m \times 16.000$ m, com doze fileiras de doze pontos de acesso/*cloudlets* (pois estão nos mesmos pontos) nas posições representadas pelos pontos pretos. As suas coberturas são de raio igual a 500 m, representadas pelos círculos azuis. Por fim, temos também o padrão de movimentação dos usuários nas simulações, representado pela linha vermelha, de ponto inicial na cruz e ponto final no X.

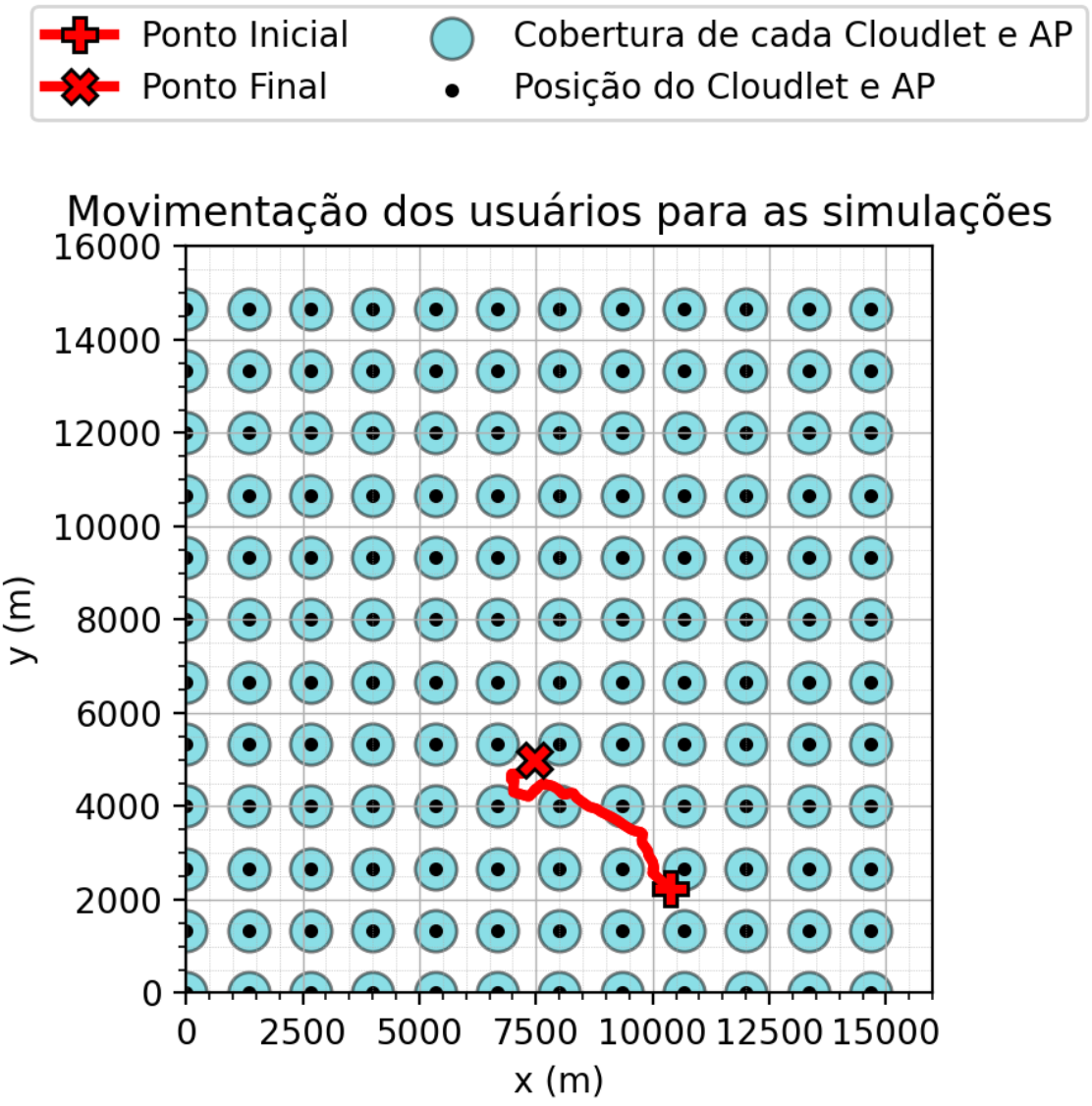

Figura 3: Mapa da rede simulada, com posições de *cloudlets*, Pontos de Acesso, e movimentação dos usuários

Além das configurações do mapa da rede, as características dos dispositivos — pontos de acesso, *cloudlets* e dispositivos dos usuários — também são definidas no código fonte do simulador.

Os pontos de acesso têm banda de *uplink* e *downlink* igual a 100 $Mbps$[17], com 4 ms de latência de *uplink*.

As *cloudlets* são definidas como dispositivos com *1 GB*[18] de memória *RAM*, *16 GB* de armazenamento, com suporte a largura de banda de até *1 GBps*[19] e capacidade de processamento de 3234 $MIPS$ (milhões de instruções executadas por segundo), executando sistemas *Linux* de arquitetura `x86`, isso é, de 32 *bits*.

Por fim, os dispositivos dos usuários são definidos com *16 MB* de *RAM*, *512 MB* de armazenamento, executando sistemas operacionais *Android* de arquitetura `x86` (32 *bits*), capacidade de processamento de 46533 $MIPS$, e têm largura de banda de até *1 Mbps* de *uplink* e 2 *Mbps* de *downlink*.

6.4 Parâmetros de Linha de Comando Utilizados nas Simulações

Os parâmetros de linha de comando de interesse para os resultados são os #3, #4 e #7. Eles que são modificados para cada cenário analisado. Para cada um deles, foram usados 3 ou 9 valores de *Seed* (parâmetro #2) diferentes. O parâmetro #5, número de usuários, também foi alterado para os cenários simulados de 1 ou 2 usuários.

Os parâmetros #1, #6, #8, #9 e #10 não foram modificados nas simulações do presente trabalho, uma vez que não são de interesse para o estudo proposto. Os valores deles foram mantidos os que vieram no *script* `script.sh` do repositório baixado. Na Tabela 2 abaixo, temos a justificativa para o uso de todos os parâmetros.

[17] *Megabits* por segundo.

[18] 1 *Gigabyte*, igual a *1024 MB*.

[19] Capacidade máxima. A largura de banda para as *cloudlets* é definida pelo parâmetro #6 na execução do programa, via linha de comando. Cf. Tabela 1.

Tabela 2: Justificativas dos parâmetros de linha de comando usados nas simulações

Número do Parâmetro	**Parâmetro**	**Justificativa**
1	Migração permitida ou não	1, migração permitida, pois testamos cenários de migração
2	*Seed* para geração de números aleatórios	Foram utilizados os três primeiros do *script* do repositório
3	Tipo de ponto de migração	Parâmetro de interesse, 0 na medida base e depois modificado para 1 para analisar como influencia nos resultados
4	Estratégia para escolha da próxima *cloudlet* em uma migração	Parâmetro de interesse, 0 na medida base, depois modificado para 1 e 2 para analisar como influencia nos resultados
5	Número de usuários	Fizemos simulações com cenários de 1 ou 2 usuários. Com mais do que isso não foi possível
6	*Bandwidth* (largura de banda) base para comunicação entre as *cloudlets*, (*Mbps*)	11, o valor que veio no *script* `script.sh` do repositório do projeto. Já é suficiente para o estudo deste trabalho
7	Política de migração da máquina virtual/*container*	Parâmetro de interesse, 0 na medida base e depois modificado para analisar como influencia nos resultados
8	Número de segundos para as predições	0, pois, já é suficiente para o nosso estudo
9	Imprecisão da medição (m)	0, pois queremos os resultados mais precisos possíveis para fazer a nossa análise sobre os parâmetros de entrada modificados
10	Latência base de rede entre as *cloudlets* (ms)	61, o valor que veio no *script* `script.sh` do repositório do projeto. Substitui a configuração de latência de comunicação entre as *cloudlets* no código fonte. Não foi modificado, porque já é um valor suficiente para o nosso estudo

6.5 Execução das Simulações

Para verificar se esses parâmetros reduzem latência e quedas experimentadas pelos usuários (*downtime*), primeiramente foi feita uma simulação base, com os parâmetros descritos na Tabela 3: ponto de migração fixo (parâmetro número 3 igual a 0), escolha de próximo nó o de menor latência para o usuário (parâmetro número 4 igual a 0), e implementação de máquina virtual com migração *Cold* (parâmetro número 7 igual a 0). Depois, foram modificados **somente** esses parâmetros 3, 4 e 7, um por vez, para verificar se a latência e *downtime* aumentaram ou não em relação aos resultados obtidos na medida base.

6.5.1 Parâmetros Utilizados na Simulação Base

As configurações dos parâmetros para a simulação base estão descritos abaixo, na Tabela 3.

Tabela 3: Parâmetros de linha de comando para a medida de base

Número do Parâmetro	**Parâmetro**	**Valores para as simulações base**
1	Migração permitida ou não	1 (permitida)
2	*Seed* para geração de números aleatórios	290538; 909538; 310420
3	Tipo de ponto de migração	0 (ponto de migração fixo)
4	Estratégia para escolha da próxima *cloudlet* em uma migração	0 (escolhe a *cloudlet* com menor latência para o usuário)
5	Número de usuários	1; 2
6	*Bandwidth* (largura de banda) base para comunicação entre as *cloudlets* (*Mbps*)	11
7	Política de migração da máquina virtual/*container*	0 (É utilizada Máquina Virtual, e a migração é do tipo *Cold*)
8	Número de segundos para as predições	0
9	Imprecisão da medição (m)	0
10	Latência base de rede entre as *cloudlets* (ms)	61

Todas as simulações foram feitas seis vezes cada, cada uma com um valor diferente de *Seed*: 290538, 909538, e 310420 — os três valores de *Seed* iniciais do *script* `script.sh` disponibilizado na pasta do projeto; e com o número de um e dois usuários. Ou seja, temos, para cada simulação, 9 valores de latência e 6 de *downtime* — o resultado de latência apresentado é para cada usuário, e o de *downtime* é a média dos valores, para simulação de

dois usuários[20].

Os valores apresentados são o resultado da média dos 9 ou 6 valores, com desvio padrão populacional[21]. Para automatizar a execução das várias simulações com a mesma configuração, alterando somente o parâmetro *Seed* para o tratamento estatístico, e também organizar e salvar os arquivos de saída, foi feito um *script* `shell`.

Depois de terminadas as três simulações, são criados pelo simulador os arquivos com os valores de latência para o/cada um dos dois usuários — no arquivo de saída `out.txt`, gerado na pasta raiz do projeto, na seção `APPLICATION LOOP DELAYS`; e *downtime* médio para o/os dois usuários no arquivo `downtime_xxx`, gerado dentro da pasta `averages` da raiz do projeto. O nome desse arquivo é diferente para cada configuração de tipo de ponto de migração e estratégia de escolha do próximo nó: para ponto de migração fixo e escolha do nó destino o de menor latência, é `downtime_FIXED_MIGRATION_POINT_with_LOWEST_LATENCY.txt`.

Por fim, foi calculada a média e desvio padrão de população com os nove (latências) ou seis (*downtime*) resultados para cada configuração a seguir:

1. **Medida base**: parâmetros #3, #4 e #7 iguais a 0 — ponto de migração fixo, seleção de nó destino de menor latência, implementação de máquina virtual com migração *Cold*;

2. **Parâmetro #3 igual a 1**, parâmetros #4 e #7 iguais a 0: ponto de migração da máquina virtual/*container* de cada usuário baseado na sua velocidade, seleção de nó destino de menor latência, implementação de máquina virtual com migração *Cold*;

3. **Parâmetro #4 igual a 1**, parâmetros #3 e #7 iguais a 0: ponto de migração fixo, seleção de nó destino de menor distância em relação ao usuário, implementação de máquina virtual com migração *Cold*;

4. **Parâmetro #4 igual a 2**, parâmetros #3 e #7 iguais a 0: ponto de migração fixo, seleção de nó destino o conectado ao ponto de acesso que está mais próximo ao usuário, implementação de máquina virtual com migração *Cold*;

5. **Parâmetro #7 igual a 1**, parâmetros #3 e #4 iguais a 0: ponto de migração fixo, seleção de nó destino de menor latência, implementação de *container* com migração *Cold*;

6. **Parâmetro #7 igual a 2**, parâmetros #3 e #4 iguais a 0: ponto de migração fixo, seleção de nó destino de menor latência, implementação de *container* com migração *Live*;

As simulações executadas estão sumarizadas na Tabela 4. Em negrito, os parâmetros #3, #4 e #7.

[20]Para a latência, temos 3 *Seeds* × 2 usuários + 3 *Seeds* × 1 usuário = 9 valores. Para o *downtime* temos 3 *Seeds* × 2 medidas (1 usuário ou média dos 2) = 6 valores.

[21]Estamos tratando de toda a população dos valores medidos, não uma amostra dessa população que depois será extrapolada.

Após execução das simulações base e alteração de um parâmetro por vez, com os três valores iniciais de *Seed*, ocorreu de os valores de latência para as configurações de parâmetro #3 = 1 e parâmetro #4 = 1 estarem com desvio padrão relativamente alto, então, foram feitas mais medidas para estas configurações de parâmetros, com os 6 valores de *Seed* seguintes no *script* **script.sh** da pasta do projeto. Esses seis valores estão entre parêntesis na tabela, nos valores do parâmetro #2. Mais detalhes na Seção 7.5.

Tabela 4: Sumarização dos parâmetros de linha de comando para as simulações executadas

Número do Parâmetro	**Parâmetro**	**Valores para as simulações**
1	Migração permitida ou não	1 (permitida)
2	*Seed* para geração de números aleatórios	290538; 909538; 310420; +(793815 299994; 201337; 768912; 768794; 148536)
3	**Tipo de ponto de migração**	**0 (ponto de migração fixo); 1 (ponto de migração baseado na velocidade do usuário)**
4	**Estratégia para escolha da próxima *cloudlet* em uma migração**	**0 (escolhe a *cloudlet* com menor latência para o usuário); 1 (escolhe a *cloudlet* que está mais próxima do usuário); e 2 (escolhe a *cloudlet* conectada ao ponto de acesso que está mais próximo ao usuário)**
5	Número de usuários	1; 2
6	*Bandwidth* (largura de banda) base para comunicação entre as *cloudlets* (*Mbps*)	11
7	**Política de migração da máquina virtual/*container***	**0 (É utilizada Máquina Virtual, e a migração é do tipo *Cold*); 1 (É utilizado *container*, e a migração é do tipo *Cold*); ou 2 (É utilizado *container*, e a migração é do tipo *Live*)**
8	Número de segundos para as predições	0
9	Imprecisão da medição (m)	0
10	Latência base de rede entre as *cloudlets* (ms)	61

7 Resultados

Os resultados são apresentados e discutidos abaixo, primeiro os de latência — resultados na Seção 7.1 e discussão na Seção 7.2; depois os de *downtime* (7.3 e 7.4). As discussões estão divididas de modo a analisar um parâmetro por vez, nas subseções.

Para verificar melhor o comportamento com os parâmetro #3 e #4 iguais a 1, foram feitas medidas adicionais. Essa discussão está na Seção 7.5.

Por fim, a discussão para medida com os parâmetros que retornaram melhores valores está na Seção 7.6; e o resultado do ponto de vista dos usuários na 7.7.

As médias e desvios-padrões estão apresentados nas tabelas com 3 dígitos decimais, e os valores mínimo e máximo com 95% de confiança (média ± 2dp) formatados segundo o padrão com a incerteza de 1 algarismo significativo.

7.1 Resultados de Latência

Tabela 5: Resultados de **Latência** obtidos para os três primeiros *Seeds*

Medida	**Latência Média** (ms)	**Desvio padrão** (ms)	**Mínimo – Máximo** (ms)
Base	60,778	3,529	$54 - 68$
Parâmetro #3 = 1	67,146	11,325	$44 - 90$
Parâmetro #4 = 1	72,875	11,797	$49 - 96$
Parâmetro #4 = 2	60,970	3,601	$54 - 68$
Parâmetro #7 = 1	39,820	1,922	$36 - 44$
Parâmetro #7 = 2	162,848	1,937	$159 - 167$

7.2 Discussão dos Resultados de Latência

7.2.1 Medida Base

Com medida base vemos que a latência fica em torno de 61 ms, o valor definido como latência base de rede entre as *cloudlets*, no parâmetro #10.

7.2.2 Parâmetro #3: Tipo de Ponto de Migração

Com 3 *Seeds*, para o ponto de migração baseado em velocidade do usuário (parâmetro $\#3 = 1$), vemos um grande desvio padrão nos resultados, então foi decidido fazer simulações adicionais, com 6 outros valores de *Seed*, para tentar diminuir a dispersão. Os resultados dessas simulações são apresentados e discutidos na Seção 7.5.

7.2.3 Parâmetro #4: Estratégia para Escolha da Próxima *Cloudlet* em uma Migração

Para a escolha da próxima *cloudlet* a mais próxima ao usuário (parâmetro #4 igual a 1), vemos também um grande desvio padrão nos resultados com 3 *Seeds*, então também foi decidido, para essa medida,fazer simulações adicionais, com 6 outros valores de *Seed*, para tentar diminuir a dispersão (Seção 7.5).

Por outro lado, para a escolha da próxima *cloudlet* a conectada ao ponto de acesso que está mais próximo ao usuário (parâmetro #4 igual a 2), temos um resultado: vemos que essa configuração de escolha de *cloudlet* destino de migração não faz diferença na latência experimentada pelo(s) usuário(s).

7.2.4 Parâmetro #7: Política de Migração da Máquina Virtual/*Container*

Com as medidas usando 3 *Seeds* temos resultados interessantes (e consistentes — de desvio padrão pequeno) para o parâmetro #7, política de migração: para política de migração utilizando *container* e migração *Cold*, o melhor resultado em latência: menor latência para os usuários em relação a todas as outras medidas; mas com política de migração utilizando *container* e migração *Live*, o pior resultado: latência média bem acima das outras medidas.

Vemos que a mudança de implementação de máquina virtual para *container* foi responsável pela melhoria (em relação à medida base), por essa implementação ser mais leve do que a primeira, de máquina virtual. Por outro lado, a migração *Live* apresentou o pior resultado para latência ao(s) usuário(s), talvez por causa da estratégia *"lazy"* utilizada após a migração da quantidade mínima do estado do *container* — migração das páginas de memória para o destino após o *page fault*, todo esse processo demorando muito mais até ser completo do que o processo de migração com implementação de máquina virtual.

7.3 Resultados de *Downtime*

Tabela 6: Resultados de **downtime** obtidos para os três primeiros *Seeds*

Medida	***Downtime* Médio** (ms)	**Desvio padrão** (ms)	**Mínimo−Máximo** (ms)
Base	99.698,917	3.124,902	93.449 – 105.949
Parâmetro #3 = 1	99.698,546	3.124,902	93.449 – 105.948
Parâmetro #4 = 1	99.698,917	3.124,902	93.449 – 105.949
Parâmetro #4 = 2	99.698,917	3.124,902	93.449 – 105.949
Parâmetro #7 = 1	59.292,958	1.235,596	56.822 – 61.764
Parâmetro #7 = 2	19.440,983	609,370	18.222 – 20.660

7.4 Discussão dos Resultados de *Downtime*

7.4.1 Parâmetros #3: Tipo de Ponto de Migração, e #4: Estratégia Para Escolha da Próxima *Cloudlet* em uma Migração

Com 3 *Seeds* já podemos verificar, pois o desvio padrão para essas medidas foi baixo em relação à média, que esses dois parâmetros, mesmo o #4 tendo três opções de valores, não fazem diferença nenhuma no resultado de *downtime* para o(s) usuário(s) — as médias e até os desvios padrões obtidos foram quase que exatamente iguais aos da medida base.

Isso já era esperado, pois o *downtime* diz respeito ao tempo em que o serviço não está disponível, por estar em migração. O momento em que essa migração começa, definido pelo parâmetro #3, tipo de ponto de migração, e a escolha do destino dessa migração, definido pelo parâmetro #4, não interferem no processo de migração de fato.

7.4.2 Parâmetro #7: Política de Migração da Máquina Virtual/*Container*

Vemos que ao trocar a implementação de máquina virtual para *container*, seja migração *Cold* ou *Live*, já houve grande melhora em resultado de *downtime* para o(s) usuário(s). Além disso, a melhora ao se utilizar a migração *Live* é muito maior do que a em *Cold*.

Esses resultados são ainda mais interessantes quando comparados aos desse parâmetro #7 em latência: enquanto com o parâmetro #7 igual a 1 (*container* com migração *Cold*), a latência foi a menor (melhor), o *downtime* é melhor que o da medida base (implementação em *máquina virtual* e migração *Cold*). Ainda mais, com o parâmetro #7 igual a 2 (*container* com migração *Live*), tivemos o resultado de latência pior que o da medida base, mas o *downtime* foi o melhor de todas as medidas.

7.5 Medidas Adicionais para os Parâmetros #3 e #4

Como as medidas com parâmetros #3 e #4 iguais a 1 retornaram resultados com desvio padrão grande demais para chegarmos a uma conclusão, fizemos as medidas com outros seis valores de *Seed*, os próximos no *script* `script.sh`.

7.5.1 Medidas Adicionais para o Parâmetro #3 Igual a 1: Ponto de Migração Baseado na Velocidade do Usuário

Ocorreu, porém, que as medidas de latência continuaram dispersas demais para chegarmos a um valor que possibilitasse uma conclusão, com os valores com a mudança do parâmetro #3 obtidos entre 38,173[22] e 1.732,258[23] ms. Mesmo fazendo a média e desvio removendo o valor 1.732,258, o desvio padrão continuou muito alto, acima de 20.

7.5.2 Medidas Adicionais para o Parâmetro #4 Igual a 1: Escolha da Próxima *Cloudlet* a Mais Próxima ao Usuário

Mesmo após fazer as medidas adicionais com os 6 novos valores de *Seed*, o desvio padrão considerando os resultados com todos os 9 *Seeds* continuou acima de 10, inviabilizando uma conclusão sobre como esse tipo de seleção de próxima *cloudlet* em uma migração afeta as latências experimentadas pelos usuários.

7.5.3 Possível Justificativa para a Dispersão Continuar Alta

Podemos concluir que os valores de latência obtidos para as configurações de parâmetros #3 e #4 iguais a 1 são muito sensíveis ao *Seed* utilizado na geração dos números aleatórios. Entendemos como uma possível justificativa para isso ocorrer o fato de *Seeds* diferentes levarem a níveis de congestionamento de rede diferentes, podendo levar a sobrecarga da rede para uma ou algumas *cloudlets*. Realmente, analisando essas configurações para os dois parâmetros, vemos que pode ser que isso ocorra: com o parâmetro #3 igual a 1, ponto de migração baseado em velocidade do usuário, esse ponto para cada velocidade de usuário é dinâmico; dependendo da velocidade, pode ocorrer de um determinada *cloudlet* ter a sua rede sobrecarregada devido a excesso de recebimento de migrações em um curto período de tempo. Com o parâmetro #4 igual a 1, escolha da próxima *cloudlet* a de menor latência, vemos que também pode ocorrer essa sobrecarga, se um determinada *cloudlet* tiver latência baixa, ela será alvo de várias migrações de máquina virtual/*container*.

7.6 Medida com os Melhores Valores

Analisando os resultados obtidos em latência, vemos que não é necessário fazer novas medidas para chegar a uma configuração melhor considerando os três parâmetros estudados, já que em latência não conseguimos concluir como os parâmetros #3 e #4 iguais a 1

[22] Latência para o usuário da simulação com 1 usuário e *Seed* 148536.
[23] Latência para o usuário 1 da simulação com 2 usuários e *Seed* 793815.

influenciam no resultado, e o #4 não tem influência, portanto a medida com parâmetro #7 igual a 1 já é a ideal.

Com os resultados em *downtime* ocorre o mesmo, mas podemos chegar a essa resposta, uma vez que vemos a dispersão é relativamente baixa, e por isso podemos concluir que os parâmetros #3 e #4 não têm influência no resultado. Portanto, o melhor resultado dentro do nosso escopo já é o com a mudança do parâmetro #7 para 2 (utilização de *container* e *Live*).

Chegamos assim à conclusão que a configuração de parâmetros #3 = 0, #4 = 0 e #7 = 1 — ponto de migração fixo, próxima *cloudlet* a de menor latência e uso de *container* com migração *Cold* — é uma melhor para latência; e a de parâmetro #3 = 0, #4 = 0 e #7 = 2 — ponto de migração fixo, próxima *cloudlet* a de menor latência e uso de *container* com migração *Live* — é uma melhor para *downtime*.

7.7 Resultados do Ponto de Vista dos Usuários

Como as melhores configurações de parâmetros, da seção anterior, são uma para menor latência e outra para melhor *downtime*, e o *MobFogSim* utiliza obrigatoriamente mesma política de migração da máquina virtual/*container* para **todas** as *cloudlets*, devemos escolher uma ou outra política de migração dos *containers* para menor latência ou menor *downtime* para os usuários.

Dentro dessa limitação do simulador, obviamente é preferível usar a implementação de menor *downtime*, mesmo que a latência seja maior, pois dessa maneira os usuários não ficam sem acesso aos serviços que estejam usando, ainda que os atrasos experimentados sejam maiores. Isso vale mesmo em relação à medida base, pois o ganho em redução de *downtime* com a migração *Live* é bem pronunciado — redução de 99 s para 19 s em relação a ela —, sendo muito mais perceptível para os usuários do que o aumento de latência de 61 ms para 163 ms que ocorre nesse caso.

8 Conclusões

Analisando os resultados obtidos, concluímos que apenas modificar o parâmetro de tipo de política de migração da máquina virtual/*container* (parâmetro #7) já é o suficiente para melhorar as latências experimentadas pelos usuários (com uso de máquina virtual e migração *Cold*) e *downtime* (uso de máquina virtual e migração *Live*), ainda que não seja possível fazer ambas ao mesmo tempo, para o simulador. Com essa limitação, uma melhor combinação possível para esses três parâmetros, considerando a usabilidade dos clientes dos serviços de rede, é a de ponto de migração fixo, escolha da próxima *cloudlet* a de menor latência e uso de *container* com migração *Live*, que proporcionou redução de *downtime* de 99 s para 19 s em relação à medida base —, ainda que a latência nessa configuração tenha subido de 61 ms para 163 ms.

Uma vez que temos uma configuração ideal para latência e outra para *downtime*, podemos chegar a outra conclusão sobre o simulador: seria interessante que este permitisse que uma *cloudlet* fizesse uso de uma configuração específica de parâmetros de migração para cada aplicativo diferente utilizado por um usuário conectado a ela. Para isso, também

seria necessário que o *MobFogSim* tivesse suporte a esses parâmetros de migração (os #1, #3, #4 e #7[24] da linha de comando) específicos para cada *cloudlet* — o que não ocorre no momento da escrita desse trabalho: atualmente, os parâmetros de linha de comando se aplicam a todas as *cloudlets* de uma simulação. Por exemplo, se tivermos um cenário com um usuário de uma aplicação em nuvem que precise de menor latência e outro com uma que precise de menor *downtime*, seria interessante que as *cloudlets* a que eles estejam conectados, mesmo se os dois estiverem conectados à mesma, oferecessem suporte a diferentes parâmetros de migração para as máquinas virtuais/*containers* dos dois diferentes usuários, dependendo das necessidades das aplicações de nuvem utilizadas por eles dois.

[24] #1: migração permitida ou não; #3: tipo de ponto de migração; #4: critério para escolha da *cloudlet* destino em uma migração; #7: política de migração da máquina virtual/*container*

9 Agradecimentos

Gostaria de deixar aqui um grande agradecimento ao professor Edmundo pela oportunidade de fazer este trabalho final de graduação sob sua orientação; e também ao Diogo Gonçalves, pela grande ajuda no entendimento do simulador. Sem eles este projeto não seria possível.

Referências

[1] AL-TARAWNEH, M.. Mobility-aware container migration in cloudlet-enabled iot systems using integrated muticriteria decision making. International Journal of Advanced Computer Science and Applications, 11(9), 2020.

[2] BEHRISCH, M. et al.. SUMO–simulation of urban mobility: an overview. In: Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation. ThinkMind, 2011.

[3] BONOMI, F. et al.. Fog computing and its role in the internet of things. In: Proceedings of the first edition of the MCC workshop on Mobile cloud computing. p. 13–16, 2012.

[4] DESAI, P. R.. A survey of performance comparison between virtual machines and containers. Int. J. Comput. Sci. Eng, v. 4, n. 7, p. 55-59, 2016.

[5] GONÇALVES, D., BITTENCOURT, L., MADEIRA, E.. Fatiamento Dinâmico de Redes em Computação em Névoa para Usuários Móveis, 2021.

[6] GONÇALVES, D.. Migração Proativa de Máquinas Virtuais em Computação em Névoa Para Usuários Móveis, 2018.

[7] GONÇALVES, D., PULIAFITO, C., MINGOZZI, E., RANA, O., BITTENCOURT, L., and MADEIRA, E.. Dynamic network slicing in fog computing for mobile users in mobfogsim. In 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC), pages 237–246. IEEE., 2020.

[8] GUPTA, H. et al.. iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments. Fog and Edge Computing: Principles and Paradigms, R. Buyya and S. Srirama (eds), 433-466pp, ISBN: 978-111-95-2498-4, Wiley Press, New York, USA, 2019.

[9] HONG, S., LIU, C.. "Sensor-Based Random Number Generator Seeding", in IEEE Access, vol. 3, pp. 562-568, doi: 10.1109/ACCESS.2015.2432140, 2015.

[10] JUN L. et al.. Delay-Aware Bandwidth Slicing for Service Migration in Mobile Backhaul Networks J. Opt. Commun. Netw. 11, B1-B9, 2019.

[11] JUN L. et al.. Enabling technologies for low-latency service migration in 5G transport networks [Invited], J. Opt. Commun. Netw. 13, A200-A210, 2021.

[12] PULIAFITO, C., GONÇALVES, D., LOPES, M., MARTINS, L., MADEIRA, E., MINGOZZI, E., RANA, O., and BITTENCOURT, L.. Mobfogsim: Simulation of mobility and migration for fog computing. Simulation Modelling Practice and Theory, 101:102062, 2020.

[13] TAURION, C. Cloud computing-computação em nuvem. Brasport, 2009.