UNIVERSIDADE FEDERAL DO PAMPA

Daniel Chaves Temp

MIGRAÇÃO DE REGISTROS DE IMAGENS DE APLICAÇÕES CONTEINERIZADAS BASEADA NA MOBILIDADE DOS USUÁRIOS EM INFRAESTRUTURAS DE BORDA

Alegrete

outubro de 2023

Daniel Chaves Temp

MIGRAÇÃO DE REGISTROS DE IMAGENS DE APLICAÇÕES CONTEINERIZADAS BASEADA NA MOBILIDADE DOS USUÁRIOS EM INFRAESTRUTURAS DE BORDA

Dissertação apresentada ao Programa de Pós-graduação Stricto Sensu em Engenharia Elétrica da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Mestre em Engenharia Elétrica.

Orientador: Fábio Diniz Rossi
Coorientador: Marcelo Caggiani Luizelli

Alegrete
outubro de 2023

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

T282m Temp, Daniel Chaves
Migração de registros de imagens de aplicações conteinerizadas baseada na mobilidade dos usuários em infraestruturas de borda / Daniel Chaves Temp.
49 p.

Dissertação(Mestrado)-- Universidade Federal do Pampa, MESTRADO EM ENGENHARIA ELÉTRICA, 2023.
"Orientação: Fábio Diniz Rossi".

1. edge computing. 2. migração. 3. registros. I. Título.

DANIEL CHAVES TEMP

MIGRAÇAO DE REGISTROS DE IMAGENS DE APLICACOES CONTEINERIZADAS BASEADA NA MOBILIDADE DOS USUARIOS EM INFRAESTRUTURAS DE BORDA

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica (PPGEE) da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Mestre em Engenharia Elétrica.

Dissertação defendida e aprovada em: 19 de outubro de 2023.

Banca examinadora:

_____

Prof. Dr. Fábio Diniz Rossi

Orientador

IFFar

_____

Prof. Dr. Marcelo Caggiani Luizelli

Coorientador

Unipampa

Prof. Dr. Bernardo Henz

IFFar

Prof. Dr. Paulo Silas Severo de Souza

IFFar

Assinado eletronicamente por **MARCELO CAGGIANI LUIZELLI**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 25/10/2023, às 16:28, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Bernardo Henz**, **Usuário Externo**, em 25/10/2023, às 16:43, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Fábio Diniz Rossi**, **Usuário Externo**, em 25/10/2023, às 16:55, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Paulo Silas Severo de Souza**, **Usuário Externo**, em 25/10/2023, às 17:03, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1269182** e o código CRC **594BA4E8**.

RESUMO

A conteinerização é a principal tecnologia de virtualização implementada em ambientes de Edge Computing devido ao seu baixo consumo de recursos e tempos de provisionamento rápidos. O desafio de posicionar de forma inteligente os registros de contêiner na infraestrutura torna-se mais evidente, pois essas entidades afetam muito a velocidade na qual os contêineres são provisionados. A pesquisa existente sobre gerenciamento de registro de contêiner tem se concentrado na definição de locais predefinidos de registro de contêiner, negligenciando a variabilidade potencial na demanda por imagens de contêiner influenciada por fatores como a mobilidade do usuário. Este trabalho propõe uma nova estratégia que provisiona dinamicamente registros de contêineres com base na mobilidade do usuário. Enquanto o provisionamento de novos registros próximos aos usuários garante tempos de provisionamento de aplicativos mais curtos, o desprovisionamento de registros ociosos otimiza a utilização da infraestrutura de borda em termos de consumo de energia. Experimentos simulados demonstram que nossa abordagem de provisionamento de registro dinâmico supera as estratégias atuais em relação ao tempo de provisionamento de aplicativos e economia de energia.

Palavras-chave: Cointêiner, Edge Computing, Registros.

ABSTRACT

Containerization is the leading virtualization technology implemented in Edge Computing environments due to its low resource consumption and fast provisioning times. The challenge of intelligently placing container registries in the infrastructure becomes more apparent as these entities significantly affect the speed at which containers are provisioned. Existing research on container registry management has focused on defining predefined container registry locations, neglecting the potential variability in demand for container images influenced by factors such as user mobility. This work proposes a new strategy that dynamically provisions container records based on user mobility. While provisioning new registries close to users ensures shorter application provisioning times, deprovisioning idle registries optimizes edge infrastructure utilization regarding power consumption. Simulated experiments demonstrate that our dynamic registry provisioning approach outperforms current strategies regarding application provisioning time and power savings.

Keywords: Conteiner, Edge Computing, Registries.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A computação em nuvem[1] é um paradigma que mudou a maneira em que serviços são ofertados pela Internet e atualmente é amplamente utilizada. Nesse paradigma, usuários têm uma maior flexibilidade, pois a nuvem oferece acesso a recursos e dados à partir de qualquer lugar com conexão à Internet. Os usuários podem acessar aplicativos e dados por meio de diferentes dispositivos, facilitando a colaboração remota e o trabalho móvel. Com a computação na nuvem, novas possibilidades surgiram, como terceirização de servidores por provedores de nuvem, serviço prestado pela AWS[1] e benefícios como mais confiabilidade, tolerância a falhas e segurança dos dados [2], escalabilidade da infraestrutura e redução de custos com equipamentos de *data center*, diminuindo suas despesas de capital (*CAPital EXpenditure - Capex*) e investindo apenas em despesas operacionais (*OPerational EXpenditure - Opex*) [3].

No modelo tradicional de infraestrutura de Tecnologia da Informação (TI), as empresas precisavam investir em hardware, servidores, redes e outros equipamentos, representando um alto custo inicial de capital (*Capex*). No entanto, ao migrar para a computação em nuvem, esses investimentos de capital são substituídos por um modelo de pagamento baseado no consumo, onde as despesas operacionais (*Opex*) tornam-se o foco principal. Nesse modelo, as organizações passam a pagar apenas pelo uso dos recursos e serviços em nuvem, como armazenamento, poder de processamento e largura de banda, eliminando a necessidade de grandes desembolsos iniciais. Essa abordagem baseada em *Opex* permite às empresas escalarem seus recursos de TI de acordo com a demanda, reduzindo custos fixos e oferecendo flexibilidade financeira significativa.

Com a evolução da computação na nuvem, surgiram novas perspectivas de uso e também problemas como *delay* [4] (também conhecido como atraso - o tempo que leva para que uma informação percorra um determinado percurso, desde o momento em que é enviado até o momento em que é recebido pelo destinatário) e uma maior necessidade na velocidade do processamento dos dados [5]. Com o advindo da mobilidade dos usuários e dados, as redes tradicionais já não são capazes de transferir as informações na velocidade em que a demanda exige.

Para evitar situações causadas pela alta latência acarretada com a distância física entre os *data centers* em nuvem e os usuários, surgiu a *Edge Computing* [6] ou computação de borda. Nesse paradigma de computação, muitas vezes o processamento dos dados é feito na nuvem. Isto é chamado de *offloading*, ou seja, é o processo de transferir determinadas tarefas de processamento de um dispositivo para outro, geralmente com o objetivo de melhorar o desempenho, economizar energia ou otimizar recursos.

Enquanto que a proximidade entre os recursos de borda e os dispositivos finais quebre a barreira física que causa problemas de latência em aplicativos baseados em nuvem, desafios como a mobilidade do usuário levantam preocupações sobre a alocação

[1] Amazon Web Services www.aws.com

de recursos em infraestruturas de borda. Por exemplo, direcionar decisões de alocação com base em estratégias de posicionamento estático pode gerar resultados satisfatórios no início, mas podem ficar aquém no médio prazo, pois os usuários podem se distanciar de seus locais originais e, consequentemente, dos dispositivos de borda onde seus aplicativos foram alocados.

De um certo modo, é possível dizer que a *Edge computing* é um passo à frente na evolução da computação na nuvem, pois foi para suprir necessidades de mobilidade dos usuários que podem se posicionar longe do núcleo da nuvem, que este paradigma surgiu. Logo, a *Edge Computing* herda recursos da nuvem. Um destes recursos é a virtualização, que facilita o processo de gerenciamento de recursos. Depois que estes recursos de borda são virtualizados, os aplicativos podem ser realocados dinamicamente pela infraestrutura para fornecer baixa latência para os usuários móveis enquanto eles se movem pelo ambiente. A virtualização pode se dar de duas formas: através de Máquinas Virtuais (VMs) ou Contêineres.

As aplicações virtualizadas, tanto em máquinas virtuais quanto em contêineres, podem acompanhar a mobilidade dos usuários através da técnica de migração. Em termos práticos, novas aplicações são provisionadas em locais mais próximos dos usuários; então os usuários são redirecionados para esse novo local; e por fim, os recursos que mantinham a aplicação virtualizada em um local anterior são liberados. Aplicações virtualizadas podem ser *stateful* ou *stateless* [7]. Caso a aplicação seja *stateful*, páginas de memória são trafegadas via rede para alimentar a aplicação virtualizada provisionada, e caso a aplicação seja *stateless*, a nova aplicação virtualizada já estará pronta para atender os clientes. No caso específico de contêineres, as camadas que fazem paret da imagem da aplicação são mantidas e são acessadas através de um contêiner *stateful* chamado registro (*registry*).

Exemplos populares de registros de imagens para contêineres incluem o Docker Hub, o Amazon Elastic Container Registry (ECR), o Google Container Registry (GCR) e o Azure Container Registry (ACR). Esses registros de imagens desempenham um papel fundamental no ecossistema de contêineres, permitindo o compartilhamento, a colaboração e a implantação eficiente de aplicativos baseados em contêineres.

Os contêineres têm emergido como a principal [8] opção para a implantação de *Edge computing* devido ao seu tempo de provisionamento reduzido, o que atende à vários requisitos das arquiteturas de software modernas. A fim de atender aos requisitos dos usuários móveis mantendo a qualidade do serviço, a literatura apresenta duas formas de alocação de serviços conteinerizados em ambientes de *Edge computing*: *placement* [9] e a migração [10] de serviços.

Placement consiste em escolher o melhor local na infraestrutura para provisionar o serviço de forma a atender a demanda do usuário móvel. No entanto, quando a mobilidade é difícil de prever, as técnicas de *placement* tornam-se ineficazes. Acontece que, os usuários podem (e possivelmente irão) se movimentar pelo ambiente e a aplicação acaba ficando

distante destes usuários, afetando a qualidade do serviço.

Alguns trabalhos propõem a migração de serviços para acompanhar a mobilidade dos usuários a fim de solucionar as limitações das técnicas do *placement* de aplicações. Assim, quando os usuários mudam de localização no ambiente, novos serviços são novamente provisionados em dispositivos de borda próximos à nova posição dos usuários. No entanto, a migração de serviços causa outra situação que afeta a qualidade do serviço. Este problema consiste na distância que os usuários podem estar localizados no provedor da nuvem onde o registro está localizado. Essa distância afeta o tempo de provisionamento das imagens de registro que devem percorrer a infraestrutura até o dispositivo de borda onde o novo serviço deve ser migrado.

As abordagens atuais que utilizam técnicas de *placement* estático e técnicas de migração de serviço para registros de contêineres na infraestrutura de rede falham em atender aos requisitos dinâmicos de ambiente de borda móvel. Essas abordagens ignoram a possibilidade de demandas por imagens de contêineres em regiões específicas, que dependem da mobilidade do usuário. Para resolver este problema, uma nova técnica é abordada neste trabalho, que provisiona registros de contêineres dinamicamente na infraestrutura de borda com base na mobilidade dos usuários, pois é inevitável levar em consideração que o usuário do serviço irá movimentar-se através de toda a infraestrutura. Esta movimentação, cada vez mais dinâmica, faz com que o tratamento das informações geradas tenha que acompanhar esta movimentação, caso contrário, atrasos na comunicação poderão afetar a qualidade do serviço prestado e quebrar o acordo de nível de serviço estabelecido.

1.1 PROBLEMA DE PESQUISA

Muitos autores já exploraram o tema que envolve otimizar a alocação de serviços em ambientes conteinerizados. Quando é realizado o *placement* destes serviços, um local na infraestrutura é escolhido a fim de melhor atender a demanda do usuário móvel. Geralmente, algum algoritmo de previsão de mobilidade é usado para estimar quais caminhos os usuários podem seguir para otimizar o posicionamento do serviço em um dispositivo de borda. Porém, com o dinamismo das movimentações e comunicações atuais, as técnicas de *placement* acabam apresentando-se não tão eficientes quanto esperado, tornando imprescindível a investigação dos desafios associados ao *placement* de serviços em ambientes conteinerizados para encontrar soluções que possam lidar adequadamente com a volatilidade das interações. A Figura 1 ilustra um exemplo de *placement*. Na primeira parte da imagem, os usuários do serviço (a), estão localizados próximos a um dispositivo de borda, e o provedor de nuvem aloca o serviço (b) o mais próximo possível desses usuários. Na segunda parte da Figura 1, que apresenta a movimentação dos usuários que estavam acessando o serviço, fica evidente a ineficácia da técnica de *placement*.

Para então sanar esta deficiência do *placement* em relação à movimentação dos usuários, autores propuseram a migração de serviços, para que estes possam acompanhar os

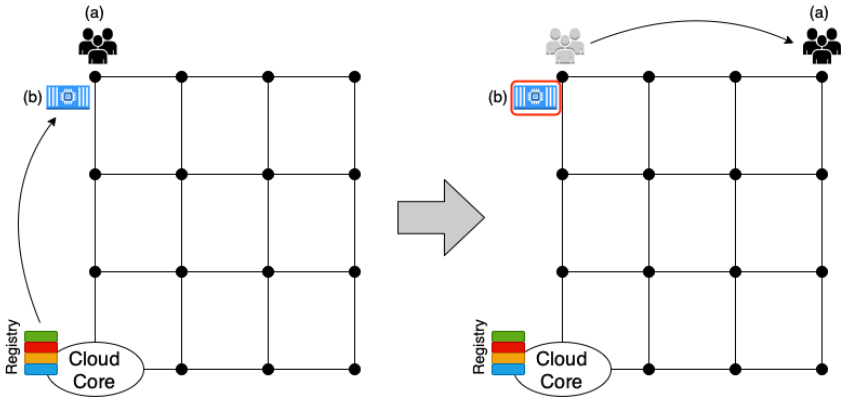


Figura 1 – Problema de *placement* de serviços

usuários móveis em sua mobilidade pela infraestrutura. Nesse contexto, quando os usuários mudam sua localização no ambiente, novos serviços são provisionados em dispositivos de borda próximos à nova posição dos usuários, como ilustrado na Figura 2 (b). No entanto, a migração de serviços apresenta um desafio adicional, que afeta principalmente a qualidade do serviço: a distância entre os usuários e o provedor de nuvem onde o registro está armazenado, que pode ser visto na segunda parte da Figura 2. Essa distância influencia no tempo de provisionamento das imagens de registro, que precisam percorrer a infraestrutura até o dispositivo de borda onde o novo serviço será provisionado.

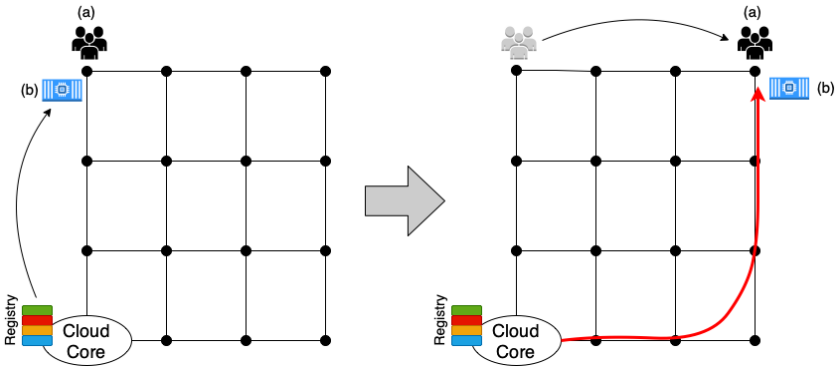


Figura 2 – Problema de migração de serviços

Devido às necessidades dinâmicas dos ambientes de borda em que os usuários se movimentam cada vez mais ativamente através da infraestrutura da rede, as abordagens existentes que adotam esquemas de posicionamento estático para registros de contêineres não são atendidas satisfatoriamente. Estas abordagens negligenciam o fato de que a exigência por imagens de contêineres em determinadas regiões pode mudar ao longo do tempo, dependendo da mobilidade do usuário e de qual serviço este usuário está utilizando. Cada aplicação possui especificidades próprias, como por exemplo, determinada aplicação pode ser executada satisfatoriamente com um delay maior, mas precisa de um tempo de provisionamento menor, ou outra aplicação necessita de delay e tempo de provisionamento

reduzidos. Estar particularidades pelo SLA (Service Level Agreement ou Acordo de Nível de Serviço) que nada mais é que um contrato formal estabelecido entre um provedor de serviços e um cliente, que define os níveis de serviço que o cliente pode esperar receber. Este acordo estabelece as métricas de desempenho e os padrões de qualidade que o provedor de serviços deve atender, como tempo de resposta, disponibilidade do serviço, tempo de atividade, entre outros indicadores relevantes para a prestação do serviço.

Como proposta, a migração de registros de contêineres em infraestrutura de borda é uma técnica utilizada para otimizar o consumo de energia, garantir a disponibilidade dos serviços e reduzir a latência das aplicações. Em um ambiente de borda, os registros de contêineres podem ser migrados para diferentes dispositivos de borda de processamento, dependendo da demanda de mobilidade dos usuários pela infraestrutura da rede. Isso permite uma melhor utilização dos recursos disponíveis e, portanto, uma redução no consumo de energia.

1.2 PROPOSTA DE SOLUÇÃO

Para endereçar esse problema de pesquisa, este trabalho propõe uma nova estratégia que provisiona registros de imagens de contêineres dinamicamente na infraestrutura, baseando-se na mobilidade dos usuários. Esta estratégia detecta quando os tempos de provisionamento estão crescendo excessivamente e ativa novos registros ao passo que provisiona serviços em locais mais próximos aos usuários. Por outro lado, os registros que agora estão ociosos, podem ser desativados para evitar o desperdício de recursos. A Figura 3 mostra esta proposta, na qual o serviço é provisionado em um local mais próximo ao usuário. Um registro, Figura 3 (c), com as camadas de imagens referentes ao serviço também é provisionado em um dispositivo de borda intermediário que suporta a cópia de tal registro. Quando o serviço, Figura 3 (b), é migrado, ao invés de as imagens serem buscadas no núcleo da nuvem, o novo serviço é provisionado a partir do registro intermediário.

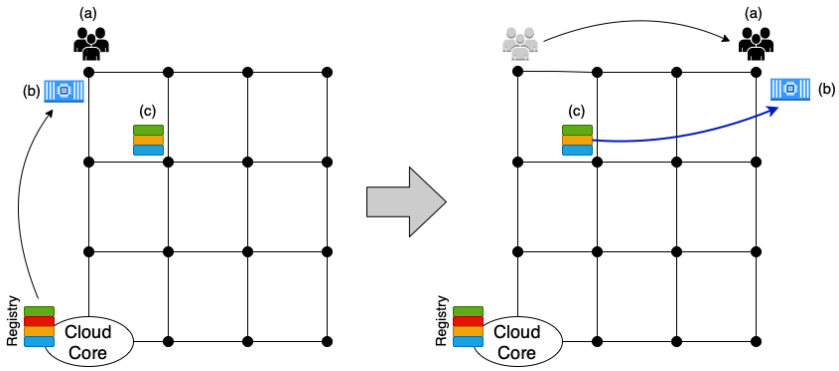


Figura 3 – Migração de registro

A questão de pesquisa que orientou este trabalho foi: Como melhorar o desempenho

no provisionamento de aplicações de borda e ao mesmo tempo reduzir o consumo de energia? Para responder a essa questão, foram realizadas análises e avaliações de desempenho em diferentes cenários e ambientes de uso através de simulações. A resposta para essa questão poderá contribuir para o desenvolvimento de estratégias mais eficientes e sustentáveis em ambientes de computação de borda, que permitam a economia de energia sem comprometer a disponibilidade dos serviços e a qualidade das aplicações.

O restante deste trabalho está organizado da seguinte forma: no Capítulo 2 é apresentado o Referencial Teórico que permeia o tema e Trabalhos Relacionados, em que são discutidas tratativas de otimizações de *placement* e migração de serviços em ambientes de borda; O Capítulo 3 exibe os procedimentos metodológicos, implementação das simulações, as avaliações e discussão de resultados; e por fim, no Capítulo 4, temos as Conclusões do Trabalho.

2 REFERENCIAL TEÓRICO E TRABALHOS RELACIONADOS

Esse capítulo apresenta o referencial teórico em termos de conceituação e tecnologias necessárias para o entendimento do trabalho, bem como uma discussão sobre os trabalhos relacionados que propõem soluções para o problema de pesquisa.

2.1 REFERENCIAL TEÓRICO

A computação em nuvem é amplamente adotada no campo da tecnologia da informação (TI) e se estabeleceu como uma infraestrutura padrão [11] para oferecer serviços nessa área. Ela representa um paradigma que combina o modelo orientado a serviços, e mais recentemente, o uso de microsserviços, com a flexibilidade proporcionada pelas características da camada de infraestrutura elástica. Em sua essência, a computação em nuvem pode ser definida como um modelo que possibilita o acesso sob demanda a um conjunto de recursos de computação configuráveis, incluindo redes, servidores, dispositivos de armazenamento, aplicativos e serviços. Esses recursos podem ser adquiridos e liberados de forma instantânea, com um mínimo de esforço de gerenciamento.

Este conceito revolucionou a forma como os serviços de TI são entregues, oferecendo vantagens significativas, como escalabilidade, flexibilidade e redução de custos. Com sua capacidade de disponibilizar recursos de forma rápida e eficiente, a computação em nuvem tem impulsionado a inovação e possibilitado o desenvolvimento de uma ampla gama de aplicações e serviços baseados em nuvem.

A camada de infraestrutura, também conhecida como Infraestrutura como Serviço (*IaaS*), desempenha um papel fundamental nesse contexto. Esta camada fornece recursos de computação sob demanda, como servidores, rede e armazenamento, para atender às solicitações dos clientes, garantindo o cumprimento dos acordos de nível de serviço estabelecidos. A capacidade em atender às demandas dos clientes é amplamente possibilitada pela elasticidade proporcionada pela virtualização, permitindo a alocação dinâmica e escalonamento dos recursos de acordo com as necessidades específicas.

A elasticidade é a capacidade fundamental da computação em nuvem, que permite o provisionamento rápido e flexível de recursos, de acordo com as demandas dos clientes. Essa capacidade de elasticidade pode ser alcançada por meio do escalonamento vertical ou horizontal. Na elasticidade vertical, recursos como processadores e memória são adicionados ao mesmo *host*, aumentando sua capacidade. Já na elasticidade horizontal, novos *hosts* são adicionados ao ambiente para ampliar a capacidade de processamento. Quando a demanda dos clientes diminui, é possível liberar recursos tanto verticalmente (reduzindo a capacidade do *host*) quanto horizontalmente (removendo *hosts* do ambiente).

No entanto, apesar das vantagens oferecidas pela elasticidade na computação em nuvem, a proposta desse modelo envolve um ambiente computacional centralizado. Essa centralização traz consigo desafios significativos no que diz respeito à qualidade de serviço (*Quality of Service - QoS*) em relação a novas aplicações. Diversos fatores

têm dificultado que os ambientes em nuvem, mesmo com todas as capacidades elásticas, atendam adequadamente a essas novas aplicações com os níveis desejados de *QoS*. Esses desafios podem estar relacionados à latência, ao desempenho, à segurança ou até mesmo à limitações físicas da infraestrutura em nuvem. A complexidade de garantir *QoS* em um ambiente centralizado e compartilhado, que lida com uma ampla variedade de aplicações, pode resultar em dificuldades para atender a requisitos específicos de desempenho e confiabilidade. Dessa forma, apesar dos avanços proporcionados pela elasticidade na computação em nuvem, é importante considerar os desafios e limitações inerentes ao fornecimento de *QoS* para novas aplicações em ambientes centralizados.

O surgimento de novas tecnologias e aplicações, como a Internet das Coisas [12], 5G [13] e Aprendizado de Máquina [14], juntamente com a mobilidade do usuário, especialmente em cenários de mobilidade veicular, resultou em um aumento significativo no envio de grandes volumes de dados heterogêneos para nuvens centralizadas. E muitas dessas aplicações têm restrições de processamento em tempo real, o que torna o modelo centralizado inadequado para atender às suas necessidades. Diante desse cenário, houve uma necessidade de reavaliar o modelo de prestação de serviços em nuvem, buscando uma abordagem mais descentralizada e próxima do cliente. Essa evolução impulsionou o surgimento da computação em borda, que visa levar o processamento de dados mais próximo do local onde são gerados, a fim de reduzir a latência e acelerar as operações em tempo real.

A computação em borda permite que os serviços e recursos de computação sejam disponibilizados em nós de processamento distribuídos em locais estrategicamente posicionados, próximos aos pontos de geração e consumo de dados [15]. Essa abordagem descentralizada possibilita um processamento mais eficiente e ágil das informações, evitando a sobrecarga das redes de comunicação e reduzindo a dependência de conexões de longa distância até nuvens centralizadas. Assim, as aplicações que demandam tempo de resposta rápidos e processamento em tempo real podem se beneficiar significativamente, uma vez que os dados são processados localmente, próximos à fonte, minimizando os atrasos de rede e proporcionando uma experiência mais ágil e responsiva para os usuários.

Portanto, a computação em borda surge como uma solução promissora para enfrentar os desafios impostos pela crescente demanda de aplicações que requerem processamento em tempo real, possibilitando um ambiente mais eficiente e adaptado às necessidades específicas de cada cenário. É possível afirmar que computação em borda é uma complementação ao modelo de computação em nuvem, trazendo benefícios significativos para o processamento de dados. Ao contrário da abordagem tradicional de processar os dados em um centro de dados remoto, na computação em borda, o objetivo é processá-los em dispositivos próximos aos clientes, onde os dados são gerados e consumidos. Como se pode ver na Figura 4, os servidores de *Edge Computing* ficam entre os dispositivos finais do usuário móveis e os servidores na nuvem. Estes recebem os dados e os enviam para a

nuvem. Já na nuvem, estes dados são processados e armazenados [16].

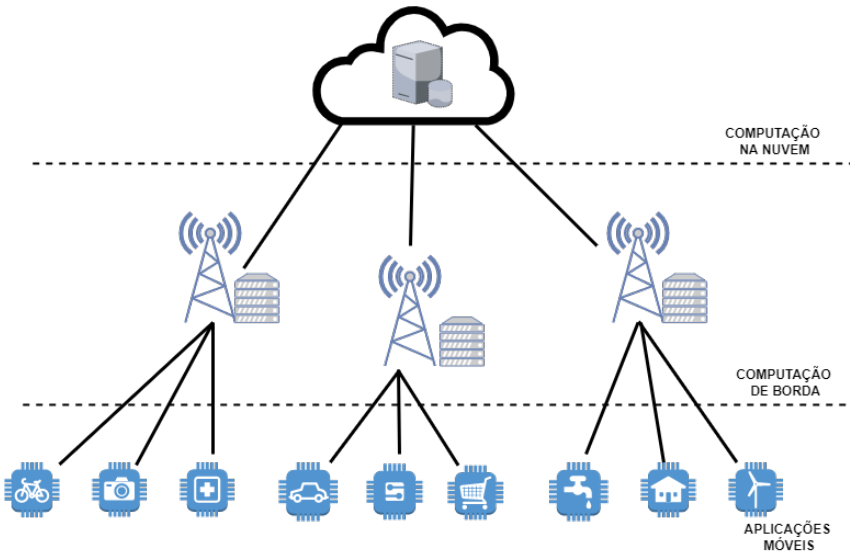


Figura 4 – Hierarquia com computação na nuvem, borda e aplicações móveis

Essa proximidade geográfica entre os dispositivos de borda e os clientes resulta em distâncias de comunicação menores, o que por sua vez reduz drasticamente a latência para apenas alguns milissegundos. Essa redução na latência é extremamente importante para aplicações e serviços que exigem respostas rápidas e processamento em tempo real. Por exemplo, nas redes 5G, onde a baixa latência é essencial para a conectividade e experiência do usuário, a computação em borda desempenha um papel fundamental ao processar os dados mais próximo das fontes de geração e consumo.

Além disso, a computação em borda é particularmente relevante na mobilidade dos usuários [17] [18]. Com a crescente adoção de aplicações em dispositivos móveis, há uma necessidade cada vez maior de processamento de dados em tempo real para suportar recursos avançados, como jogos eletrônicos [19], *streaming* de dados, navegação e tomada de decisões em tempo real. Em suma, a computação em borda é um fator crucial para o desenvolvimento eficiente de várias aplicações e tecnologias, como redes 5G e mobilidade dos usuários. Ao permitir o processamento de dados próximo aos clientes, ela reduz significativamente a latência e melhora a experiência do usuário, viabilizando aplicações de tempo real e impulsionando a inovação em diversos setores.

À medida que a infraestrutura de nuvem e computação em borda se expande, é essencial que as arquiteturas de desenvolvimento de software se adaptem a esse cenário em constante evolução. Nos ambientes de nuvem e borda, as arquiteturas de software predominante utilizadas têm sido a arquitetura orientada a serviços (*Service-oriented Architecture - SoA*) e os microsserviços [20]. Essas abordagens arquiteturais permitem uma maior flexibilidade, escalabilidade e eficiência na entrega de serviços e aplicativos em ambientes distribuídos. Ao adotar a *SoA* e os microsserviços, as organizações podem

aproveitar ao máximo os benefícios da infraestrutura de nuvem e borda, garantindo um desenvolvimento ágil e adaptável às demandas em constante mudança.

SoA fornece aplicativos como serviços, baseados em computação distribuída e protocolos de solicitação e resposta da Internet, além de permitir a integração de aplicativos de diferentes fontes e linguagens de programação por meio dos mesmos protocolos de comunicação. No entanto, com o avanço das novas tecnologias e aplicações de *big data*, a *SoA* tem suas limitações, como dependência de desempenho do servidor, vulnerabilidade à sobrecarga do usuário e dificuldades em testar e depurar devido à heterogeneidade dos aplicativos, além de erros incontroláveis e desafiadores de tratar.

Os microsserviços são uma abordagem arquitetônica que usa componentes independentes e autônomos para criar aplicações. Cada microsserviço possui uma função específica e pode ser desenvolvido, implantado e escalado de forma independente. Isso traz benefícios como escalabilidade seletiva e facilidade de desenvolvimento, teste e implantação de alterações em partes específicas do sistema sem afetar os outros componentes.

Existem dois modelos para executar microsserviços em ambientes de nuvem e borda: baseados em contêineres e baseados em *serverless*. Este trabalho se concentra no modelo de microsserviços baseados em contêineres, que permite uma migração mais natural entre dispositivos de borda usando virtualização. Ao contrário da virtualização tradicional [21], onde máquinas virtuais são usadas com uma camada de virtualização chamada *hypervisor*, os contêineres são executados diretamente no sistema operacional host e são gerenciados por um mecanismo específico.

Aplicativos que são executados através de máquinas virtuais tradicionais estão instalados dentro do um sistema operacional completo, incorrendo em um carregamento de componentes por vezes desnecessários. Já aplicativos baseados em contêineres, compartilham bibliotecas e binários do sistema operacional do equipamento hospedeiro, resultando em menor uso de recursos e, consequentemente, maior agilidade ao carregar o serviço [22]. A Figura 5[1] ilustra as diferenças entre as duas arquiteturas.

Ambas arquiteturas são baseadas em imagens, que são *templates* contendo binários e dependências usadas pelos aplicativos. A principal diferença entre as imagens utilizadas por cada uma destas arquiteturas é que as imagens de VMs são, geralmente, de uma camada única e as de contêineres são baseadas em um sistema de arquivo de várias camadas, onde cada camada tem uma função específica. A Figura 6 ilustra um exemplo de um sistema de contêiner multicamadas. Na figura, na camada I é apresentado o sistema operacional, que pode ser, por exemplo, Windows, Linux, Ios, etc... Na camada II estão armazenados os *runtimes*, que são conjuntos de código pré-compilado que fornecem funções e rotinas comuns para auxiliar na execução de programas. A camada III é onde estão as aplicações ou dependências que possibilitam o processamento de elementos escritos na linguagem que são nativamente suportada pelo sistema operacional. Já a camada IV hospeda o ambiente

[1] Baseado na imagem retirada de www.docker.com/resources/what-container

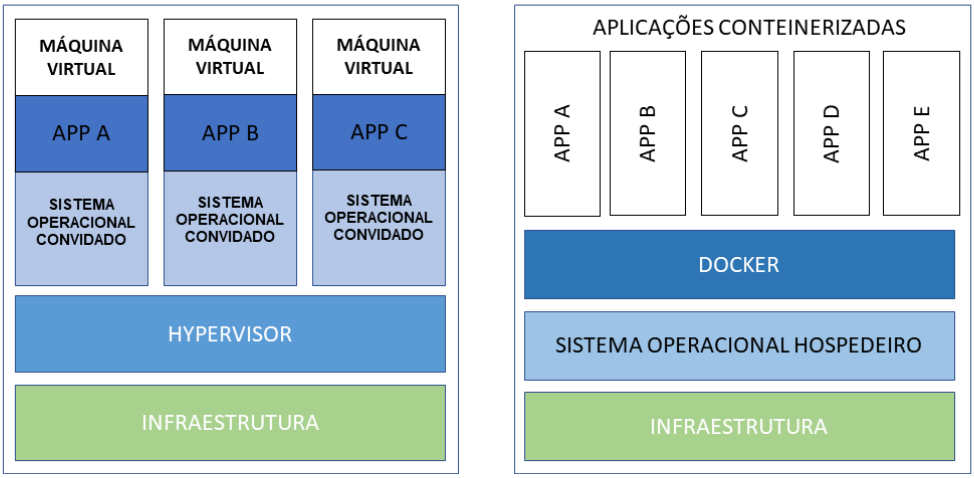


Figura 5 – Diferenças das arquiteturas de máquinas virtuais e contêineres

em que o usuário pode efetuar leitura/escrita de dados. Com exceção desta camada, todas as outras são de apenas leitura. Contêineres executados dentro do mesmo equipamento hospedeiro podem compartilhar entre si as imagens, tendo o uso de recursos reduzido, assim como o tempo de provisionamento dos contêineres em *hosts* com imagens em cache.

Além disso, a virtualização por Máquinas Virtuais possui maior isolamento e, consequentemente, maior segurança, porém apresenta uma demanda maior de armazenamento. Diante disto, a virtualização por contêineres surgem como a principal opção para implantar aplicativos de borda [23]. Por ocuparem menos espaço em disco que as Máquinas Virtuais, eles reduzem os tempos de provisionamento, atendendo aos requisitos de flexibilidade e descartabilidade de arquiteturas de software modernas, principalmente os microsserviços [24].

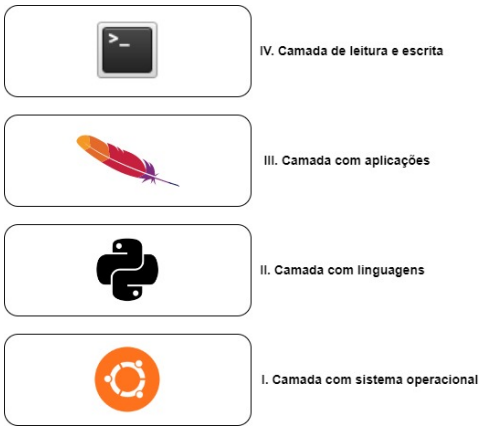


Figura 6 – Exemplo de um arquivo de imagem baseado em camadas

Com as particularidades entre máquinas virtuais e contêineres já em mente, sabe-se que aplicativos são migrados de formas diferentes dependendo do tipo de virtualização em que estão hospedados. Os aplicativos baseados em máquinas virtuais são realocados por meio de técnicas de migração a frio, que normalmente transferem os dados do aplicativo

de sua origem para seu *host* de destino. Contêineres são considerados uma virtualização leve, isto, é, os aplicativos baseados em contêineres geralmente são gerados no *host* de destino, enquanto suas imagens de contêineres são extraídas de repositórios de imagens de contêineres, chamados de *registries*.

Os *registries*, ou registros de contêineres, são semelhantes a repositórios de código-fonte, mas são específicos para imagens de contêineres (ou seja, são contêineres *stateless*). Eles fornecem um local centralizado onde os desenvolvedores podem armazenar e compartilhar suas imagens de contêineres com outras equipes ou usuários. Esses registros permitem que os desenvolvedores compartilhem suas imagens publicamente ou restrinjam o acesso apenas a usuários autorizados. No contexto de contêineres, é comum mencionar o processo de "build" de uma imagem de contêiner, que envolve a criação de uma imagem a partir de um arquivo de definição, geralmente chamado de *Dockerfile*. Este arquivo contém instruções para configurar o ambiente e as dependências necessárias para o aplicativo que será executado no contêiner. Este processo envolve a execução dessas instruções para criar a imagem de contêiner final. A imagem resultante é então usada para criar e executar contêineres.

2.2 TRABALHOS RELACIONADOS

O armazenamento de imagens nos registros de contêineres é fundamental para o ecossistema de borda virtualizada, pois é nela que as imagens de contêiner são hospedadas e distribuídas pela rede. No entanto, um desempenho ineficiente dos registros pode causar atrasos significativos no provisionamento de serviços e até mesmo interrupções deste serviço, caso o registro deixe de funcionar. Por isso, muitos esforços têm sido feitos para otimizar a forma como as imagens de contêineres são distribuídas dos registros para os dispositivos de borda.

Um dos primeiros estudos a abordar uma análise de cargas de trabalho em registros de *Docker* foi em Anwar et al.[25]. A partir da infraestrutura de nuvem da IBM, os autores coletaram e analisaram cargas reais de trabalho e como elas afetam a qualidade de serviço de aplicações em infraestruturas de grande escala. Foram propostas estratégias eficazes de armazenamento em cache e pré-busca com base em suas descobertas para melhorar o desempenho e o uso dos registros do *Docker*.

A análise de quais tipos de aplicações demandam mais recursos do contêiner foi feita em Harter et al.[26]. No artigo, os autores discutem o desenvolvimento de um novo *benchmark* de contêineres chamado *HelloBench* para avaliar os tempos de inicialização de 57 diferentes aplicativos conteinerizados. Os autores analisaram as cargas de trabalho em detalhes, estudando os padrões de E/S do conjunto exibidos durante a inicialização e a compressibilidade das imagens do contêiner. Eles descobriram que a extração de pacotes representa 76% do tempo de inicialização do contêiner, mas apenas 6,4% desses dados são lidos. Com base em suas descobertas, eles projetaram um novo driver de armazenamento

Docker chamado *Slacker*. Este driver, que nada mais é que um componente de software que permite o armazenamento persistente e o gerenciamento de volumes de dados dentro de um ambiente de contêineres. O *Slacker* é otimizado para inicialização rápida de contêineres e acelera o ciclo médio de desenvolvimento de contêineres em 20 vezes e o ciclo de implantação em 5 vezes.

Como no trabalho anterior, outros estudos também demonstraram soluções para reduzir o tempo de provisionamento, como em Chen et al.[27] em que os autores apresentam o *Starlight*, um acelerador para provisionamento de contêineres que aborda os desafios de links de alta latência, largura de banda limitada e usuários com recursos limitados em ambientes móveis e de computação de borda. O *Starlight* dissocia o provisionamento do desenvolvimento ao redesenhar o protocolo de implantação do contêiner, o sistema de arquivos e o formato de armazenamento de imagens. Na avaliação, foram utilizados 21 contêineres populares e mostrou que, em média, o *Starlight* implanta e inicia contêineres 3 vezes mais rápido do que a implementação atual de última geração, sem incorrer em sobrecarga de tempo de execução e com pouca sobrecarga de armazenamento (5%).

Para muitos usuários, quando se fala em contêineres, os mesmos pensam em imensas infraestruturas com recursos computacionais quase que ilimitados, o que não é exatamente a realidade de todos. Estudos como Nathan et al.[28], em que os autores discutiram os desafios de provisionar aplicativos em contêineres em infraestruturas com recursos limitados e apresentaram o *CoMICon*, uma solução que permite o gerenciamento cooperativo de imagens *Docker* em um pool de servidores. Em resumo, todos os servidores hospedam um registro de contêiner e são elegíveis para fornecer imagens de contêiner com seus vizinhos.

Ainda abordando a perspectiva de poucos recursos, Becker, Schmidt e Kao[29] apresentam o *EdgePier*, um registro de contêineres descentralizado, e é capaz de diminuir os tempos de implantação de contêineres utilizando conexões ponto a ponto entre os dispositivos participantes. O registro é capaz de sincronizar imagens entre serviços de registro dentro de um dispositivo de borda próximo sem a necessidade de serviços centralizados de orquestração ou metadados. Semelhante ao *EdgePier*, a utilização de servidores descentralizados também pode ser vista em Ahmed e Pierre[30], em que os autores apresentam uma solução em que é utilizado um cluster de servidores de borda separados geograficamente para distribuir as imagens dos contêineres. Outro trabalho que distribuiu geograficamente os registros foi em Knob et al.[31], no qual os autores, através de simulações apresentaram um novo método de implantação baseado em comunidades para distribuir registros de contêineres em uma topologia de borda.

Todos os três trabalhos abordaram situações com heterogeneidade e infraestrutura com recursos limitados que podem aumentar o atraso das aplicações, porém, apesar de suas contribuições, as abordagens ainda não conseguem lidar com as necessidades dinâmicas dos ambientes de borda, adotando esquemas de registros estáticos na infraestrutura, o que

negligencia que a demanda por imagens de contêiner em determinadas regiões pode mudar ao longo do tempo, dependendo da mobilidade do usuário.

Até o momento, grande parte dos esforços de pesquisa na área de contêineres têm sido direcionada à melhoria das operações internas dos registros de contêineres ou a implementação de protocolos ponto a ponto, com o objetivo de reduzir o tempo necessário para transferir imagens de contêineres para os dispositivos de borda. No entanto, é importante notar que essas soluções pressupõem que os registros de contêineres estejam fisicamente próximos aos *hosts* que solicitam as imagens. Caso essa suposição não seja válida e haja uma grande distância de rede entre os registros e os dispositivos de borda de destino, os ganhos de otimização podem ser parcialmente anulados.

O nosso trabalho complementa as abordagens existentes ao propor uma nova política de gerenciamento de recursos que permite a ativação dinâmica de novos registros quando os tempos de provisionamento de aplicativos começam a aumentar de forma excessiva, ao mesmo tempo em que desprovisiona os registros subutilizados. O trabalho mais próximo ao nosso é Knob et al.[31], que apresenta uma política de alocação de registros. Porém, o trabalho assume que o número ótimo de registros é conhecido e que esse valor permanece o mesmo ao longo do tempo, o que não necessariamente ocorre em infraestruturas de borda que atendem a usuários móveis.

Na Tabela 1 é possível ver uma breve comparação entre os trabalhos relacionados e a nossa abordagem. Nenhum destes trabalhos abordam a redução do tempo de provisionamento de aplicativos em contêineres em infraestruturas de borda utilizando migração dinâmica dos registros de imagens de acordo com a mobilidade do usuário.

Tabela 1 – Comparação entre este trabalho e a literatura

Trabalho	**Abordagem**	**Foco em Mobilidade**	**Paradigma**
Anwar et al. [25]	Registry design optimizations	×	Cloud Computing
Harter et al. [26]	Registry design optimizations	×	Cloud Computing
Chen et al. [27]	Registry design optimizations	×	Edge Computing
Nathan et al. [28]	Peer-to-peer image distribution	×	Cloud Computing
Becker et al. [29]	Peer-to-peer image distribution	×	Edge Computing
Ahmed et al. [30]	Peer-to-peer image distribution	×	Edge Computing
Knob et al. [31]	Registry placement	×	Edge Computing
Este trabalho	**Registry migration**	✓	**Edge Computing**

3 MIGRAÇÃO DE REGISTROS

Este capítulo descreve detalhadamente a implementação do uso do simulador e avaliação dos dos resultados separados por taxas de ocupação da infraestrutura obtidos a partir das simulações geradas.

3.1 IMPLEMENTAÇÃO DAS SOLUÇÕES

A metodologia deste trabalho foi executada através de um simulador chamado **EdgeSimPy** [32], tendo em vista a impossibilidade de executar testes em um ambiente real com um grande número de servidores de borda. O simulador pode ser encontrado neste endereço[1], assim como o dataset utilizado. Logo, este software simulou o ambiente de uma infraestrutura de rede com estações base distribuídas geograficamente e interligadas entre si. O simulador foi desenvolvido em linguagem *Python* e utilizou a biblioteca *NetworkX*, usada para criar, manipular e estudar a estrutura, dinâmica e funções de redes complexas. A biblioteca fornece ferramentas para a criação de grafos, análise de redes e algoritmos de manipulação de grafos.

Na simulação, o ambiente é representado como um conjunto de células hexagonais que dividem o mapa, como no modelo apresentado pelo autores Aral, Demaio e Brandic[33]. Algumas destas estações base possuem alocados à ela servidores de borda. E são nestes servidores que estão armazenados os repositórios de imagens de contêineres, chamados de registros. Nestes registros estão armazenadas as imagens que serão utilizadas pelas aplicações do usuário, enquanto ele se desloca pela infraestrutura. Para melhor explicar o cenário, a Figura 7 apresenta um exemplo da infraestrutura utilizada para a simulação.

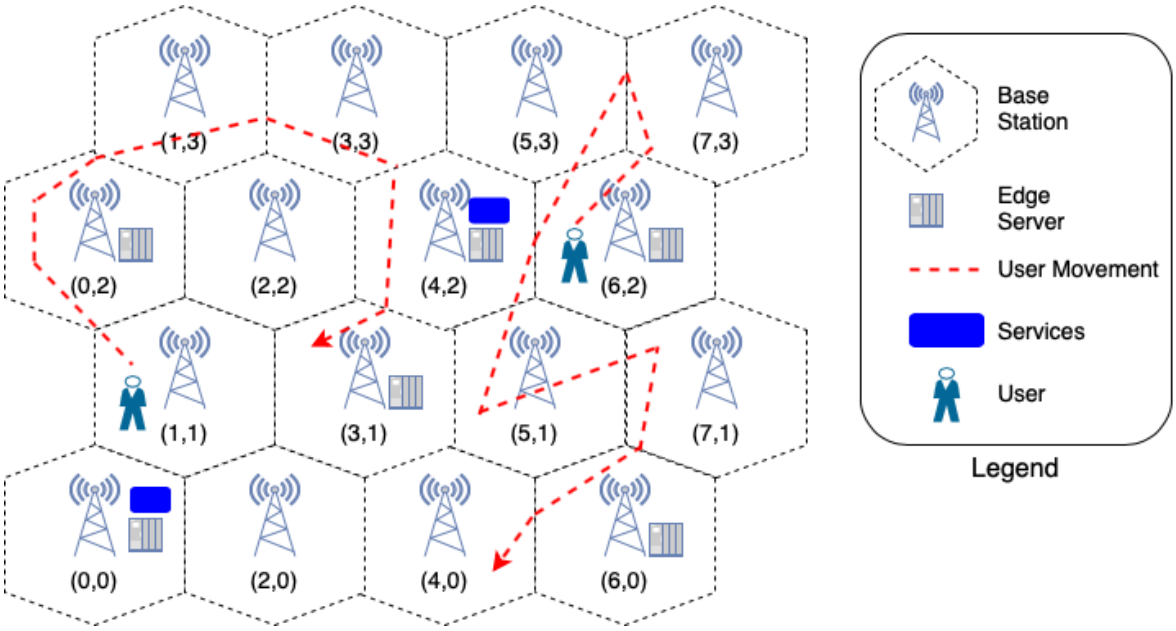


Figura 7 – Exemplo de um cenário de computação de borda

[1] Simulador: <https://github.com/paulosevero/registry_migration>.

A infraestrutura de rede compreende um conjunto de estações base $\mathcal{B}$ interligadas por um conjunto de links $\mathcal{L}$. Nestas configurações, uma estação base é representada por $\mathcal{B}_o = (w_o)$, em que o atributo w_o representa o delay da banda de $\mathcal{B}_o$, e o link da rede é representado por $\mathcal{L}_v = (d_v, b_v)$, onde os atributos d_v e b_v representam delay e capacidade de larguram de banda de $\mathcal{L}_v$. A Tabela 2 resume todas estas notações.

Tabela 2 – Lista de notações utilizadas neste trabalho

Símbolo	Descrição
w_o	Delay da estação base $\mathcal{B}_o$
d_v	Delay do link $\mathcal{L}_v$
b_v	Capacidade da largura de banda do link $\mathcal{L}_v$
c_i	Capacidade do servidor de borda $\mathcal{S}_i$
μ_j	Lista das camadas que compõe a imagem da aplicação $\mathcal{A}_j$
σ_j	Demanda de capacidade da aplicação $\mathcal{A}_j$
β_j	SLA de delay da aplicação $\mathcal{A}_j$
γ_j	SLA de tempo de provisionamento da aplicação $\mathcal{A}_j$
q_k	Tamanho da camada do contêiner $\mathcal{I}_k$
g_u	Lista de camadas de contêiner hospedadas no registro $\mathcal{R}_u$
h_u	Demanda do registro $\mathcal{R}_u$
$x_{e,i,j}$	Placement das aplicações
$z_{e,i,u}$	Placement dos registros
$\eta(\mathcal{T}_t, \mathcal{A}_j)$	Delay da aplicação $\mathcal{A}_j$ na etapa de tempo $\mathcal{T}_t$
$\xi(\mathcal{T}_t, \mathcal{A}_j)$	Links usados para rotear os dados do aplicativo $\mathcal{A}_j$ na etapa de tempo $\mathcal{T}_t$
$\rho(\mathcal{T}_t, \mathcal{S}_i)$	Demanda do servidor de borda $\mathcal{S}_i$ na etapa do tempo $\mathcal{T}_t$

Uma aplicação é representada por $\mathcal{A}_j = (\mu_j, \sigma_j, \beta_j, \gamma_j)$ e neste cenário as aplicações são conteinerizadas. Assim, $\mathcal{A}_j$ é desenvolvido em cima de um imagem de contêiner composta por uma ou várias camadas de contêiner $\mu_j \in \mathcal{I}$, na qual cada uma desta camada é representada como $\mathcal{I}_k = (q_k)$, e atributo q_k representando o tamanho da camada. A demanda total de capacidade de $\mathcal{A}_j$, representado por σ_j, é o tamanho agregado de todas as suas camadas, neste caso, $\sigma_j = \sum_{k \in \mu_j} q_k$.

O delay da aplicação $\mathcal{A}_j$ na etapa de tempo $\mathcal{T}_t \in \mathcal{T}$ é dado por $\eta(\mathcal{T}_t, \mathcal{A}_j)$, como em Eq. 3.1, considerando o delay da wireless $\mathcal{B}_o$ (que representa a estação base usada pelo usuário $\mathcal{A}_j$) ae o delay agregado de um conjunto de links de rede $\xi(\mathcal{T}_t, \mathcal{A}_j)$, usado para rotear os dados de $\mathcal{A}_j$ da sua estação base para a estação base do usuário. Nesta modelagem, $\xi(\mathcal{T}_t, \mathcal{A}_j)$ é encontrado através do algoritmo do caminho mais curto de Dijkstra [34] (delays como peso). Neste contexto, o SLA de delay de $\mathcal{A}_j$ é violado sempre que seu delay $\eta(\mathcal{T}_t, \mathcal{A}_j)$ ultrapassa o SLA de delay, dado por β_j.

$$\eta(\mathcal{T}_t, \mathcal{A}_j) = w_o + \sum_{v \in \xi(\mathcal{T}_t, \mathcal{A}_j)} d_v \tag{3.1}$$

Além de hospedar aplicações, os servidores de borda acomodam um conjunto de registros de contêineres $\mathcal{R}$, que contêm as camadas de contêiner que compõem as imagens de contêiner usadas pelos aplicativos. Um registro de contêiner é representado por $\mathcal{R}_u = (g_u, h_u)$, onde os atributos g_u e h_u representam as camadas hospedadas de $\mathcal{R}_u$ e a demanda de $\mathcal{R}_u$ (soma do tamanho de todas as camadas hospedadas por $\mathcal{R}_u$). Sempre que uma aplicação $\mathcal{A}_j$ precisa ser provisionada ou realocada, as camadas que compõem

sua imagem devem ser buscadas em um registro de contêiner disponível na infraestrutura. Este provisionamento de aplicativo é detalhado em Alg. 1.

Algoritmo 1: Processo de provisionamento das aplicações

```
1 Function provisionApp(A_j, S_i):
2     foreach I_k ∈ μ_j do
3         R' = Registro ordenados pela distância (asc.) to S_i
4         foreach R'_u ∈ R' do
5             if R'_u hospera a camada I_k then
6                 Download camada I_k de R'_u para S_i
7                 break
8     return Tempo necessário para provisionar A_j
```

Estudos anteriores mostram que o download de camadas em paralelo pode aumentar o tempo de provisionamento, pois uma camada só é descompactada e extraída após o download da camada anterior [35]. Então, nesta modelagem, as camadas são baixadas sequencialmente (Alg. 1, lines 2–7). Para cada camada que compõe a imagem do contêiner $\mathcal{A}_j$, é iterado na lista de registros, procurando o registro mais próximo que hospeda essa camada. Assim que o registro mais próximo que hospeda as camadas for encontrado, ele começará a a fazer o download. Este processo é repetido até que toda a imagem do contêiner seja baixada. O SLA de tempo de provisionamento de $\mathcal{A}_j$ é violado sempre que o resultado de Alg. 1 exceder seu limite γ_j.

Um servidor de borda é representado por $\mathcal{S}_i = (c_i)$, onde c_i simboliza a capacidade de $\mathcal{S}_i$. O placement da aplicação é representado por $x_{e,i,j}$, no qual recebe 1 se o servidor de borda $\mathcal{S}_i$ hospeda a aplicação $\mathcal{A}_j$ no passo de tempo $\mathcal{T}_t$, e 0, caso contrário. Da mesma forma, a localização do registro é dada por $z_{e,i,u}$, onde recebe 1 se o servidor de borda $\mathcal{S}_i$ hospeda o registro $\mathcal{R}_u$ no passo de tempo $\mathcal{T}_t$, e 0 caso não. A demanda geral de servidores de borda $\mathcal{S}_i$ no passo de tempo $\mathcal{T}_t$ é dada por $\rho(\mathcal{T}_t, \mathcal{S}_i)$, como pode ser visto na Eq. 3.2.

$$\rho(\mathcal{T}_t, \mathcal{S}_i) = \sum_{j=1}^{|\mathcal{A}|} \sigma_j \cdot x_{e,i,j} + \sum_{u=1}^{|\mathcal{R}|} h_u \cdot z_{e,i,u} \tag{3.2}$$

Os usuários se movimentarão pela infraestrutura acessando aplicações com diferentes distâncias e tempos de provisionamento. O simulador utiliza o modelo de mobilidade Pathway [36]. Este modelo é uma abordagem para entender e analisar os padrões de mobilidade urbana de pessoas em uma cidade. Ele se concentra na ideia de que as pessoas não apenas se movem de um ponto a outro, mas também seguem rotas específicas ou "caminhos" ao longo do tempo. Os usuários transitam apenas pelos caminhos do mapa. Este mapa é representado como um grafo em que as ruas são constituídas pelas arestas e os nodos são pontos que afetam a mobilidade, como por exemplo, esquinas.

Os servidores são interligados de acordo com a topologia Barabási-Albert, apresentada pelos autores em Barabási e Albert[37]. Esta topologia é amplamente utilizada para descrever redes de escala livre, onde alguns nós têm um número significativamente maior de conexões do que outros. Esta topologia baseia-se no princípio do crescimento

preferencial, o que significa que a probabilidade de um novo nó se conectar a um nó existente é proporcional ao número de conexões que o nó existente já possui. Isto é, os nós que têm mais conexões têm uma maior probabilidade de receber novas conexões, tornando-se "hubs" da rede.

Foram utilizados três cenários, de acordo com a taxa de ocupação do ambiente (25%, 50% e 75%). O que diferencia estes três cenários é o número de usuários e o número de aplicações distribuídas pelo ambiente. Na Tabela 3 são apresentados os quantitativos de cada cenário utilizado na simulação. Em todos os cenários, foram utilizados três perfis diferentes de consumo de energia para os dispositivos de borda: o primeiro perfil é com um consumo bem baixo de energia, com um equipamento equivalente a um Raspberry PI.

O perfil intermediário, com um equipamento equivalente a um servidor multicore e por fim, um consumo mais alto de energia, equivamente a um pequeno cluster com servidores multiprocessado.

O registro inicial é composto por três camadas (sistema operacional, runtimes e aplicação) com imagens e tamanhos diferentes para cada camada. Isto torna a simulação mais realista, pois imita infraestruturas reais de computação de borda.

Tabela 3 – Quantitativos dos cenários

Taxa de Ocupação	**25%**	**50%**	**75%**
Estações Base	100	100	100
Servidores de Borda	50	50	50
Imagens de Contêineres	315	315	315
Registros	15	15	15
Serviços	55	235	415

Neste trabalho, foram utilizadas quatro heurísticas como parâmetro de comparação entre as estratégias para a migração de registros. A opção de utilizar heurísticas para este trabalho se dá no fato que a alocação de serviços em ambientes de contêiner é um problema complexo e desafiador, que envolve muitas variáveis e incertezas. As heurísticas são abordagens simplificadas e práticas para resolver problemas complexos, que podem fornecer soluções razoáveis em tempo hábil.

Não há parâmetro de comparação na literatura entre as estratégias abordadas neste trabalho, pois nenhuma outra abordagem aloca registros dinamicamente em infraestruturas de borda. Assim, as heurísticas Never Follow (NF) e Follow User (FU), apresentadas pelos autores em Yao et al.[38], foram utilizadas como base de comparação. A estratégia Never Follow não realiza decisões de migração de registros de imagens com base na mobilidade dos usuários. Essa estratégia serve como um limite inferior para decisões de migração, permitindo identificar o quanto as políticas sensíveis à mobilidade reduzem problemas relacionados a atrasos em comparação com um cenário de referência sem migrações. Por outro lado, a estratégia Follow User complementa a avaliação com uma abordagem oposta à Never Follow. Follow User satura a infraestrutura ao migrar desnecessariamente aplicativos

quando seus usuários ainda estão suficientemente próximos a eles e seu SLA de delay ainda está sendo respeitado.

A heurística Performance (P), já apresentada por Temp et al.[10], e a heurística Energy-Aware (EA) são implementadas para usar a migração de registros baseadas na mobilidade dos usuários. Estas duas heurísticas utilizam-se de limites para coordenar a necessidade de migração de novos registros na infraestrutura. O foco aqui é a necessidade de evitar as violações do SLA. O SLA de tempo de provisionamento é um limite que define o tempo máximo permitido para que um aplicativo seja provisionado. Se o tempo de provisionamento exceder esse limite, isso será considerado uma violação do SLA. Estas heurísticas fazem isso ordenando as aplicações pelo atraso de provisionamento (quão distante o serviço está do registro localizado no núcleo da nuvem ou do registro intermediário provisionado na infraestrutura de borda). Quando este atraso ultrapassa um limite estabelecido, significa que um novo registro deve ser provisionado mais próximo do serviço. Este é o conceito por trás da heurística Performance. A heurística Energy-Aware (EA) leva em consideração os mesmos requisitos, porém, verifica o consumo de energia dos dispositivos de borda que estão próximos ao serviço e escolhe aquele que tem maior potencial de economia de energia.

Cada uma das quatro estratégias possuem suas particularidades, que, resumidamente, são:

- **Never Follow (NF)**: após o *placement* inicial do serviço, o registro e os serviços provisionados permanecem estáticos no local inicial, mesmo que o usuário se movimente pelo ambiente. Os registros de contêineres permanecem no núcleo da nuvem. logo, nesta heurística, não há nenhuma migração de registro e nem de serviços.

- **Follow User (FU)**: após o serviço ser inicialmente provisionado, este serviço sempre migrará para um local mais próximo do usuário de acordo com a mobilidade dos usuários pela infraestrutura. Porém, os registros ainda permanecem estáticos no núcleo da nuvem.

- **Performance (P)**: após a alocação inicial, esta heurística sempre migrará os registros para um local mais próximo do usuário, de acordo com a sua mobilidade. Conforme a mobilidade do serviço, os registros também serão migrados (total ou parcialmente) para dispositivos de borda que estão entre o núcleo da nuvem e o local onde o serviço está hospedado. Esta heurística leva sempre em consideração o melhor desempenho possível.

- **Energy-Aware (EA)**: esta heurística é semelhante à heurística anterior, em termos de migração. Os registros ficarão entre o núcleo da nuvem e o local onde o serviço está hospedado, porém, leva sempre em consideração a melhor economia de energia possível. A diferença está em que esta heurística busca o dispositivo de borda com o

menor perfil de consumo de energia possível, desde que este atenda os requisitos de armazenamento do registro e tempo de provisionamento do serviço.

Estas duas estratégias propostas (Performance e Energy-Aware) inicialmente organizam a lista de aplicações de acordo com uma função de pontuação ∂ (Eq. 3.3), que as classifica com base em quanto seu atraso real excede seu limite de atraso (Alg. 1, linha 2). Dessa forma, os aplicativos com problemas de atraso mais intensos são migrados primeiro.

$$\partial(\mathcal{T}_t, \mathcal{A}_j) = \beta_j - \eta(\mathcal{T}_t, \mathcal{A}_j) \tag{3.3}$$

No Algoritmo 1 é possível ver como é todo o funcionamento das heurísticas Performance e Energy-Aware.

Algoritmo 2: Estratégia Heurística Performance

```
1  foreach time step T_t ∈ T do
2      A = Aplicações em A ordenadas por Eq. 3.3 (asc.)
3      φ = {}
4      foreach A_j ∈ A do
5          if η(T_t, A_j) > β_j · λ then
6              ϰ = Usuário que acesssa A_j
7              S = Servidores de borda ordenados por atraso até ϰ (asc.)
8              foreach S_i ∈ S do
9                  if A_j já está hospedado em S_i then
10                     break
11                 else
12                     if c_i − ρ(T_t, S_i) ≥ σ_j then
13                         pt = provisionApp()
14                         if pt > γ_j · ρ then
15                             φ = φ ∪ {ϰ}
16                         break
17     Desprovisionar registros de contêineres distantes dos usuários
18     ψ = Lista dos servidores com capacidade para hospedar um registro
19     while |φ| > 0 and |ψ| > 0 do
20         θ = {}
21         foreach S_i ∈ ψ do
22             foreach U_e ∈ φ do
23                 A_j = Aplicação acessada por U_e
24                 b = Largura de banda entre S_i e U_e
25                 if σ_j / b ≤ γ_j · ρ then
26                     θ_i = θ_i ∪ {U_e}
27         S_i = Servidor de borda em ψ com a maior θ
28         if |θ_i| > 0 then
29             Provisionamento de um registro em S_i
30             ψ = ψ − {S_i}
31             φ = φ − θ_i
32         else
33             ψ = {}
```

Em vez de migrar todas as aplicações, o algoritmo proposto tenta migrar apenas aquelas cujo atraso está próximo o suficiente de seus SLAs de delay. O limite de proximidade entre o atraso atual e o atraso do SLA é definido por meio de um limiar λ (Alg.2, linha 5). Quando uma aplicação $\mathcal{A}_j$ é migrada, o algoritmo proposto reúne a lista de servidores de borda candidatos a hospedá-la, classificando esses servidores de acordo com seu atraso em relação ao usuário de $\mathcal{A}_j$ (Alg.2, linha 7). Se o servidor de borda mais próximo já

hospeda $\mathcal{A}_j$, nenhuma migração é realizada (Alg.2, linhas 9–10). Caso contrário, $\mathcal{A}_j$ é migrada para o servidor de borda mais próximo de seu usuário (Alg.2, linhas 12–16). Se o tempo de migração de $\mathcal{A}_j$ exceder um limite ρ, o usuário de $\mathcal{A}_j$ é adicionado a uma lista ϕ, que corresponde à lista de usuários que acessam aplicações cujos SLAs de tempo de provisionamento estão prestes a ser violados (Alg. 2, linha 15).

Após realizar as migrações de aplicativos, nosso algoritmo desprovisiona todos os registros que não são os mais próximos de nenhum dos usuários no ambiente (Alg.2, linha 17). Em seguida, ele começa a tomar decisões para provisionar novos registros para evitar que ocorram violações de SLAs de tempo de provisionamento. Nesse processo, o algoritmo começa reunindo uma lista de servidores de borda elegíveis para hospedar um registro de contêiner (Alg.2, linha 18). Em seguida, ele inicia um processo iterativo que busca encontrar locais adequados para os registros (Alg. 2, linhas 19–33). Esse processo iterativo se repete até que os registros sejam provisionados com proximidade suficiente de todos os usuários em ϕ, ou nenhum servidor de borda na infraestrutura tenha recursos suficientes para hospedar um registro.

Quando escolhe servidores de borda para hospedar registros, o algoritmo mantém uma lista θ_i para cada servidor $\mathcal{S}_i \in \psi$, com o número de usuários cujas violações de SLA de tempo de provisionamento poderiam ser evitadas se $\mathcal{S}_i$ hospedasse um registro (Alg.2, linhas 21–26). Uma vez obtido θ para cada $\mathcal{S}_i \in \psi$, o servidor com o maior valor de θ (ou seja, aquele que poderia evitar o maior número de futuras violações de SLA de tempo de provisionamento) é escolhido para hospedar um registro (Alg.2, linha 29).

A principal diferença dos algoritmos das heurísticas Performance e Energy-Aware se dá em (Alg. 1, linha 7). Na heurística Performance, os servidores de borda são ordenados crescentemente pelo delay. Já na heurística Energy-Aware, os servidores são ordenados decrescentemente pelo seu consumo de energia.

3.2 AVALIAÇÃO E RESULTADOS

Na simulação, foram avaliadas três métricas resultantes das quatro heurísticas aplicadas:

- **Violações de SLA de delay**: quantidade de vezes em que o aplicativo ultrapassou o limite aceitável de atraso do SLA. Esse limite indica que o usuário está distante demais de onde o serviço está provisionado e, portanto, é necessário provisionar um novo serviço em um local mais próximo ao usuário.

- **Violações de SLA de Tempo de Provisionamento**: quantidade de vezes em que o SLA estabelecido como aceitável para migração ou provisionamento de cada aplicativo foi violado. Quando esse limite é ultrapassado, indica que os usuários estão se conectando a um local menos próximo de onde o serviço está provisionado

e, portanto, é necessário provisionar um novo serviço em outro dispositivo de borda mais próximo dos usuários.

- **Consumo de Energia**: consumo total de energia utilizada pelos equipamentos de borda na infraestrutura.

As duas primeiras métricas impactam diretamente nas questões de desempenho da aplicação. Já a terceira, afeta a questão de consumo de energia da infraestrutura. As três métricas permitem a avaliação das heurísticas de migração de registros propostas neste trabalho. A Figura 8 ilustra a diferença dessas duas primeiras métricas. É possível ver de onde que é medido o tempo de provisionamento das imagens do registro e também é possível ver entre quais partes surgem o delay da execução da aplicação. Somando o tempo de provisionamento com o delay temos o tempo total da aplicação.

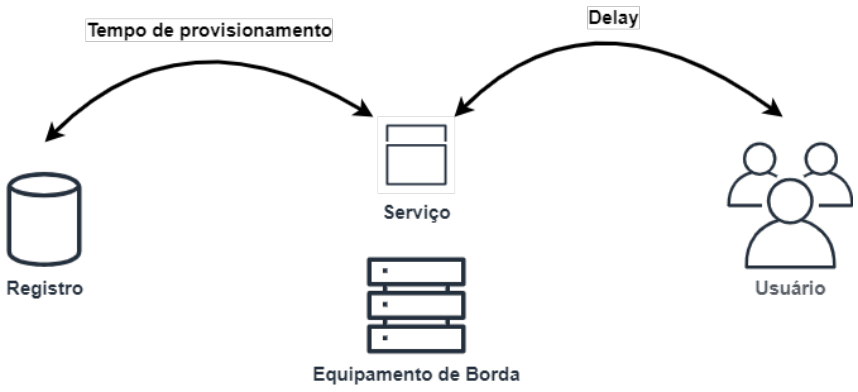


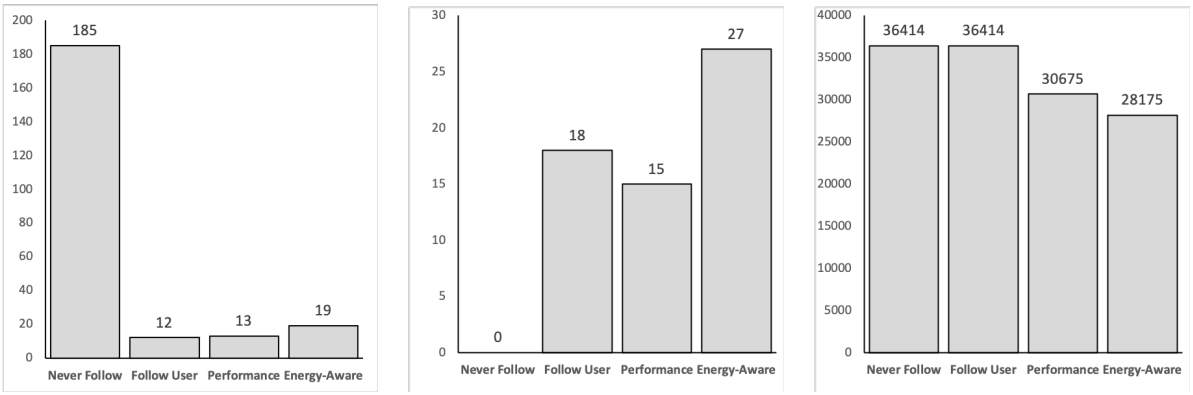
Figura 8 – Diferenças entre tempo de provisionamento e delay

Para uma melhor organização do trabalho, cada cenário referente às taxas de ocupação será discutido individualmente.

3.2.1 EXPERIMENTOS COM TAXA DE OCUPAÇÃO DE 25%

Esta seção apresenta os resultados alcançados no primeiro cenário de avaliação, onde os 50 servidores de borda na infraestrutura hospedam 55 aplicações, resultando em uma demanda média de infraestrutura de 25%.

Heurística Never Follow: A heurística Never Follow é a que mais realiza as violações de SLA de delay nos cenários com taxa de ocupação de 25% da infraestrutura, como se pode ver na Figura 8 (a). Isso ocorre porque, após o provisionamento inicial, a heurística não migra mais os serviços para acompanhar a mobilidade dos usuários. Em ambientes altamente móveis e dinâmicos, como os atuais, o serviço acaba ficando distante dos usuários, causando atrasos nas trocas de informações. Por causa disto, a heurística não possui violações de SLA de tempo de provisionamento e o também o consumo de energia é mais elevado.



(a) Violações de SLA de delay (b) Violações de tempo de provisionamento (c) Consumo de Energia

Figura 9 – Experimentos com taxa de 25% de ocupação da infraestrutura

Heurística Follow User: A heurística Follow User apresenta poucas violações de SLA de delay para a taxa de ocupação de 25%. Isso se deve à sua natureza, que constantemente provisiona os serviços em locais mais próximos aos usuários, de acordo com sua mobilidade. No entanto, as violações de SLA de tempo de provisionamento ainda são consideráveis. A heurística sempre provisiona novos recursos para provisionar um novo serviço, causando uma sobrecarga desnecessária na infraestrutura de borda. O consumo de energia é semelhante à heurística Never Follow, como pode ser visto na Figura 8 (c).

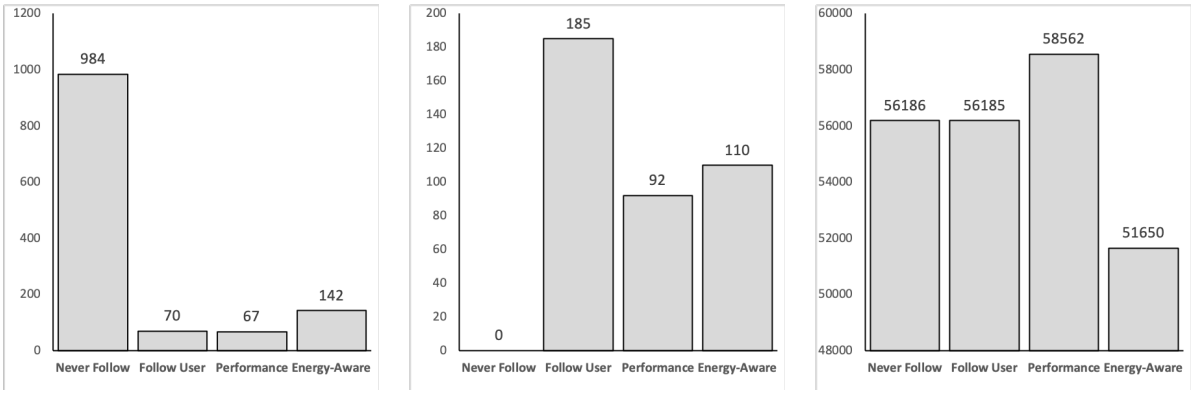
Heurística Performance: A heurística Performance reduz apenas um pouco as violações de SLA de delay em comparação com a Follow User para a taxa de ocupação de 25%. No entanto, há uma redução considerável nas violações de SLA de tempo de provisionamento, como ilustrado na Figura 8 (b). Isso ocorre porque os registros intermediários são continuamente distribuídos na infraestrutura de borda, mantendo as imagens dos serviços próximas aos usuários. Porém, a heurística Performance não se preocupa com a economia de energia, distribuindo os registros pela infraestrutura sem considerar a capacidade e o consumo de energia dos dispositivos de borda.

Heurística Energy-Aware: A heurística Energy-Aware escolhe o dispositivo de borda com menor consumo de energia e capacidade suficiente para armazenar um registro intermediário próximo aos serviços provisionados. No entanto, em relação ao atraso, apresenta violações de SLA piores do que a heurística Performance para a taxa de ocupação de 25%. Também há consideráveis violações de SLA de tempo de provisionamento. Apesar disso, a heurística Energy-Aware é consideravelmente mais econômica em termos de consumo de energia em comparação com as outras heurísticas.

3.2.2 EXPERIMENTOS COM TAXA DE OCUPAÇÃO DE 50%

Esta seção apresenta os resultados para o segundo cenário considerado durante nossa avaliação, onde os 50 servidores de borda hospedam 235 aplicações, resultando em

uma demanda média de infraestrutura de 50%.



(a) Violações de SLA de delay (b) Violações de tempo de provisionamento (c) Consumo de Energia

Figura 10 – Experimentos com taxa de 25% de ocupação da infraestrutura

Heurística Never Follow: Esta heurística continua apresentando as maiores violações de SLA de delay para a taxa de ocupação de 50% da infraestrutura. O serviço fica distante dos usuários devido à falta de migração após a alocação inicial. Como já discutido, a heurística não efetua nenhuma migração, portanto, não tem tempo de provisionamento. O consumo de energia também é alto neste cenário, como visto na Figura 9.

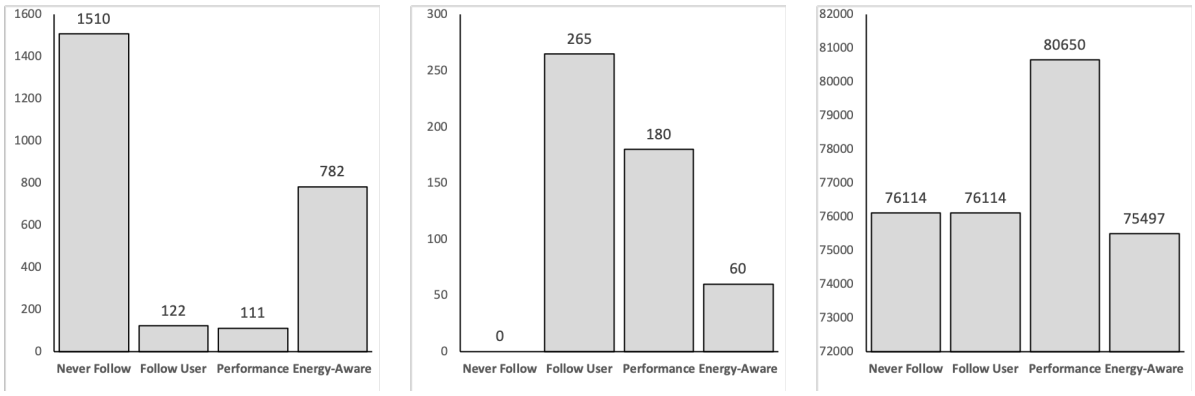
Heurística Follow User: A heurística Follow User ainda possui poucas violações de SLA de delay, mas as violações de SLA de provisionamento são consideráveis para a taxa de ocupação de 50%. O consumo de energia é semelhante à heurística Never Follow, como visto na Figura 9 (c).

Heurística Performance: A heurística Performance apresenta pouca redução nas violações de SLA de delay em comparação com a Follow User para esta taxa de ocupação, como é possível ver na Figura 9 (a). As violações de SLA de tempo de provisionamento são menores, pois os registros intermediários são distribuídos na infraestrutura, mantendo as imagens dos serviços próximas aos usuários. No entanto, a heurística não considera a economia de energia.

Heurística Energy-Aware: A heurística continua apresentando violações de SLA de delay piores do que a heurística Performance para a taxa de ocupação de 50%. As violações de SLA de tempo de provisionamento também ocorrem. No entanto, o consumo de energia é consideravelmente menor em comparação com as outras heurísticas, como é possível ver na Figura 9 (c).

3.2.3 EXPERIMENTOS COM TAXA DE OCUPAÇÃO DE 75%

Esta seção apresenta os resultados alcançados no terceiro cenário de avaliação, onde os 50 servidores de borda na infraestrutura hospedam 415 aplicações, resultando em uma demanda média de infraestrutura de 75%.



(a) Violações de SLA de delay (b) Violações de tempo de provisionamento (c) Consumo de Energia

Figura 11 – Experimentos com taxa de 25% de ocupação da infraestrutura

Heurística Never Follow: Assim como nos outros cenários, a heurística Never Follow continua a apresentar no cenário com taxa de ocupação de 75% as maiores violações de SLA de delay, como mostrado na Figura 10 (a). O tempo de provisionamento continua zerado, como mostra a Figura 10 (b) e o consumo de energia é alto.

Heurística Follow User: A heurística Follow User possui poucas violações de SLA de delay para a taxa de ocupação de 75%. No entanto, as violações de SLA de tempo de provisionamento são consideráveis, como visto na Figura 10 (b). O provisionamento constante de novos recursos causa sobrecarga na infraestrutura de borda. O consumo de energia é semelhante à heurística Never Follow.

Heurística Performance: A heurística Performance apresenta pouca alteração em relação à violações de SLA de delay em comparação com a Follow User para a taxa de ocupação de 75%. As violações de SLA de tempo de provisionamento são menores, mas o consumo de energia continua sendo alto, pois a heurística não considera a economia de energia.

Heurística Energy-Aware: A heurística Energy-Aware continua exibindo níveis mais elevados de violações de SLA de delay em comparação com as heurísticas Performance e Follow User para uma taxa de ocupação de 75%. Além disso, ainda ocorrem violações de SLA de tempo de provisionamento, mesmo que não tão consideráveis do que em comparação às duas heurísticas anteriores. No entanto, é importante destacar que o consumo de energia é semelhante quando comparado às duas primeiras heurísticas, como observado na Figura 10 (c).

3.3 CONSIDERAÇÕES PARCIAIS

A análise das simulações revela a complexidade inerente ao gerenciamento de recursos em infraestruturas de servidores de borda. Embora a migração de registros tenha surgido como uma abordagem intrigante para aprimorar o desempenho dos serviços de

borda, não podemos negligenciar métricas críticas, como consumo de energia e tempo de provisionamento, que desempenham papéis igualmente importantes no sucesso da operação dessas infraestruturas.

Em resumo, as simulações mostram que a migração de registros é um conceito interessante para melhorar o desempenho dos serviços de borda, mas também é importante considerar métricas como consumo de energia e tempo de provisionamento.

Cada heurística apresenta suas próprias vantagens e desvantagens em relação a essas métricas cruciais, tornando a escolha da estratégia um desafio que depende dos requisitos específicos do sistema. À medida que exploramos essas heurísticas em detalhes, fica evidente que suas características únicas podem ter um impacto significativo no desempenho geral do sistema.

Como esperado, as heurísticas Never Follow e Follow User apresentaram os piores resultados. A Never Follow, com sua abordagem estática que não envolve migração após o provisionamento inicial de registros, inevitavelmente resulta em grandes violações de SLA de delay em todos os cenários simulados. Isso ocorre porque os serviços permanecem distantes dos usuários devido à falta de adaptação às mudanças nas demandas e na mobilidade dos usuários.

Já a heurística Follow User, por ser exatamente o contrário da Never Follow, isto é, ela sempre seguirá os usuários em suas movimentações pela infraestrutura, mesmo que não haja necessidade, resulta em grande consumo de energia, porém, apresenta uma média interessante em termos de tempo de provisionamento. Essa abordagem dinâmica visa garantir que os serviços estejam sempre próximos dos usuários, mas às custas de um consumo de energia significativo.

A estratégia "Performance" se destaca por provisionar serviços rapidamente, mas ao fazer isso, consome mais energia do que todas as outras estratégias. Isso ocorre porque prioriza o desempenho dos serviços em detrimento da eficiência energética. Notavelmente, a estratégia de desempenho mostrou consistência na redução das violações de SLA de delay em cenários com diferentes níveis de carga de trabalho, destacando-se especialmente em ambientes com carga leve e moderada.

Por sua vez, a estratégia "Energy-Aware" concentra-se em alocar recursos que consomem menos energia, contribuindo para a redução do consumo total de energia da infraestrutura. No entanto, essa abordagem de eficiência energética pode comprometer o desempenho das aplicações, tornando-a mais adequada para cenários de alta carga de trabalho, onde a minimização das violações de SLA de atraso é crucial.

Nestas considerações, é possível admitir que a escolha da estratégia de gerenciamento de recursos para servidores de borda dependerá das necessidades específicas de um ambiente e das metas de desempenho, eficiência energética e conformidade com SLAs. A diversificação das estratégias pode ser benéfica para lidar com a variabilidade nas cargas de trabalho e garantir a melhor adaptação a diferentes cenários operacionais.

Além disso, a diversificação das estratégias pode ser benéfica para lidar com a variabilidade nas cargas de trabalho e para garantir que a infraestrutura de servidores de borda seja capaz de se adaptar eficazmente a diferentes cenários operacionais. A combinação adequada de estratégias pode proporcionar um equilíbrio entre desempenho, eficiência energética e conformidade com SLAs, atendendo às demandas variáveis de um ambiente dinâmico.

4 CONCLUSÕES

A computação em borda, uma faceta inovadora da computação distribuída, tem se destacado como uma tecnologia de vanguarda que está moldando o futuro da computação. Em seu cerne, a computação em borda envolve a descentralização do processamento de dados e a aproximação da capacidade de computação e armazenamento mais perto dos dispositivos e sensores que geram esses dados. Isso permite uma série de benefícios significativos, incluindo latência reduzida, maior eficiência de largura de banda e uma resposta mais rápida a eventos em tempo real. Além disso, a computação em borda desempenha um papel crítico no suporte à Internet das Coisas (IoT), permitindo que bilhões de dispositivos interconectados coletem, processem e respondam aos dados localmente, antes de enviar informações valiosas para a nuvem. Esse paradigma de computação está revolucionando setores inteiros, desde a saúde até a indústria automotiva, e promete abrir novas oportunidades emocionantes em áreas como realidade aumentada, cidades inteligentes e automação industrial. No entanto, desafios relacionados à segurança, gerenciamento de dados e escalabilidade ainda precisam ser superados à medida que a computação em borda continua a evoluir e se expandir.

A virtualização baseada em contêineres foi reconhecida como a principal arquitetura para alcançar elasticidade e melhor uso de recursos em infraestruturas de computação em borda. Depois que os aplicativos são encapsulados em contêineres, que ocupam menos espaço do que as máquinas virtuais, é menos provável que os servidores de borda sejam afetados pela sobrecarga de virtualização, o que é bom devido às restrições de recursos das infraestruturas de borda. Ao mesmo tempo, a agilidade dos contêineres permite mover aplicativos dinamicamente pela infraestrutura à medida que seus usuários se movem pelo ambiente. No entanto, gerenciar aplicativos em contêineres na borda implica em lidar com um conjunto significativo de desafios. Por exemplo, o provisionamento oportuno de aplicativos em contêineres de acordo com a mobilidade dos usuários resulta em comunicação intensa em toda a infraestrutura de borda, pois as imagens de contêineres devem ser transferidas dos registros de contêineres para onde quer que os aplicativos devam ser provisionados.

Estratégias de posicionamento que distribuem registros na infraestrutura para mitigar possíveis pontos de acesso que levam a tempos de provisionamento prolongados. No entanto, à medida que o número de usuários no ambiente cresce, mais registros são provisionados para atender à demanda crescente, o que pode se tornar inconveniente em infraestruturas de borda com recursos limitados, pois os registros podem saturar recursos que poderiam ser melhor usados para hospedar aplicativos. Logo, pensar em soluções que abordem a mobilidade dinâmica baseada nos comportamentos dos usuários abre um grande leque para futuras pesquisas envolvendo a área de contêineres e movimentação dos usuários.

Este trabalho prevê uma nova abordagem baseada em limites que ativam novos

registros quando os tempos de provisionamento de aplicativos estão se tornando muito altos, enquanto os registros existentes longe dos usuários são desprovisionados para evitar o desperdício de recursos. Foram utilizadas heurísticas com foco em economia de energia e desempenho e comparadas com heurísticas já abordadas na literatura, porém, que não utilizavam de mobilidade dinâmica dos registros. Os resultados experimentais demonstram que nossa abordagem reduz os problemas de tempo de provisionamento de aplicativos em 50% em média em comparação com estratégias que alocam registros de contêineres estáticos e uma redução do consumo de energia de até 23% em certos cenários.

4.1 TRABALHOS FUTUROS

O trabalho apresentado abre um leque de oportunidades para futuras pesquisas, como:

- Utilização de inteligência artificial para a predição da mobilidade dos usuários, aliado ao uso de heurísticas.
- Projetar algoritmos de gerenciamento de recursos que tomem decisões otimizadas em cenários onde vários usuários em diferentes locais acessam o mesmo serviço.
- Abordar heurísticas que tratem especificamente do *trade-off* entre consumo de energia e desempenho.
- Abordar cenários com tecnologias emergentes, como por exemplo, 6G.

4.2 PUBLICAÇÕES

Pelo regimento do PPGEE - Unipampa, para a obtenção do título de Mestre em Engenharia Elétrica, é necessário ter obtido o aceite de artigo em um congresso de nível nacional ou internacional ou revista científica com estrato Qualis na área de avaliação Engenharias IV referente ao trabalho desenvolvido. Ao longo destes dois anos de trabalho no grupo de pesquisa, foi possível a publicação dos seguintes artigos:

- **TEMP, D. C.** ; SOUZA, P. S. S. ; LORENZON, ARTHUR F. ; LUIZELLI, MARCELO C. ; ROSSI, F. D. . Mobility-aware registry migration for containerized applications on edge computing infrastructures. JOURNAL OF NETWORK AND COMPUTER APPLICATIONS, p. 103676, 2023. **Publicado**. Qualis: **A1**.
- **TEMP, DANIEL** ; CAPELETTI, IGOR ; GOES DE CASTRO, ARIEL ; SEVERO DE SOUZA, PAULO ; LORENZON, ARTHUR ; LUIZELLI, MARCELO ; ROSSI, FÁBIO . Latency-Aware Cost-Efficient Provisioning of Composite Applications in Multi-Provider Clouds. In: 13th International Conference on Cloud Computing and

Services Science, 2023, Prague. Proceedings of the 13th International Conference on Cloud Computing and Services Science, 2023. v. 1. p. 297. **Publicado**. Qualis: **A3**.

- CAPELETTI, IGOR ; GOES DE CASTRO, ARIEL ; **TEMP, DANIEL** ; SEVERO DE SOUZA, PAULO ; LORENZON, ARTHUR ; ROSSI, FÁBIO ; LUIZELLI, MARCELO . Towards Optimizing the Edge-to-Cloud Continuum Resource Allocation. In: 13th International Conference on Cloud Computing and Services Science, 2023, Prague. Proceedings of the 13th International Conference on Cloud Computing and Services Science, 2023. v. 1. p. 90. **Publicado**. Qualis: **A3**.

- **TEMP, D. C.** ; SOUZA, P. S. S. ; LORENZON, ARTHUR F. ; LUIZELLI, MARCELO C. ; ROSSI, F. D. . Dynamic Provisioning of Container Registries in 6G-Enabled Edge Computing Infrastructures. IEEE Network. **Submetido - Em revisão**. Qualis: **A1**.

REFERÊNCIAS

1 ZHANG, L. C. . R. B. Q. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, v. 1, n. 7, p. 7–18, 2010. ISSN 1869-0238. Citado na página 15.

2 BUYYA, R. et al. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, Elsevier, v. 25, n. 6, p. 599–616, 2009. Citado na página 15.

3 GOVINDA, K.; OKEREAFOR, K. Capex and opex minimization using cloud technology. *International Journal of Computer Applications*, v. 6, p. 7 – 15, 10 2012. Citado na página 15.

4 LI, G. et al. Data processing delay optimization in mobile edge computing. *Wireless Communications and Mobile Computing*, v. 2018, p. 1–9, 02 2018. Citado na página 15.

5 SATYANARAYANAN, M. et al. The seminal role of edge-native applications. In: *IEEE International Conference on Edge Computing*. [S.l.: s.n.], 2019. p. 33–40. Citado na página 15.

6 SATYANARAYANAN, M. et al. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, IEEE, v. 8, n. 4, p. 14–23, 2009. Citado na página 15.

7 LI, Z. Long live the image: Container-native data persistence in production. In: *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*. [S.l.: s.n.], 2021. p. 82–85. Citado na página 16.

8 CAPROLU, M. et al. Edge computing perspectives: Architectures, technologies, and open security issues. In: *2019 IEEE International Conference on Edge Computing (EDGE)*. [S.l.: s.n.], 2019. p. 116–123. Citado na página 16.

9 CHEN, Y. et al. Mobility-aware edge server placement for mobile edge computing. *Computer Communications*, v. 208, p. 136–146, 2023. ISSN 0140-3664. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0140366423001913>. Citado na página 16.

10 TEMP, D. C. et al. Mobility-aware registry migration for containerized applications on edge computing infrastructures. *Journal of Network and Computer Applications*, v. 217, p. 103676, 2023. ISSN 1084-8045. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804523000954>. Citado 2 vezes nas páginas 16 e 33.

11 BOHN, R. B. et al. Nist cloud computing reference architecture. In: *IEEE World Congress on Services*. [S.l.: s.n.], 2011. p. 594–596. Citado na página 21.

12 GUBBI, J. et al. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, v. 29, n. 7, p. 1645–1660, 2013. ISSN 0167-739X. Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond. Citado na página 22.

13 AGIWAL, M.; ROY, A.; SAXENA, N. Next generation 5g wireless networks: A comprehensive survey. *IEEE Communications Surveys and Tutorials*, v. 18, n. 3, p. 1617–1655, 2016. Citado na página 22.

14 CHEN, X.-W.; LIN, X. Big data deep learning: Challenges and perspectives. *IEEE Access*, v. 2, p. 514–525, 2014. Citado na página 22.

15 SHI, W. et al. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, v. 3, n. 5, p. 637–646, 2016. Citado na página 22.

16 SHI, W.; DUSTDAR, S. The promise of edge computing. *Computer*, IEEE, v. 49, n. 5, p. 78–81, 2016. Citado na página 23.

17 COSTA, J. et al. Escalonamento de tarefas ciente de contexto para computação de borda veicular. In: *Anais do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Porto Alegre, RS, Brasil: SBC, 2022. p. 15–28. ISSN 2177-9384. Disponível em: <https://sol.sbc.org.br/index.php/sbrc/article/view/21158>. Citado na página 23.

18 COSTA, J. et al. Nemesis: Mecanismo para formação de nuvens veiculares baseado em previsão de mobilidade. In: *Anais do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Porto Alegre, RS, Brasil: SBC, 2022. p. 280–293. ISSN 2177-9384. Disponível em: <https://sol.sbc.org.br/index.php/sbrc/article/view/21177>. Citado na página 23.

19 ROBAINA, G.; FIORESE, A. Gaming on the edge: Uma arquitetura de computação na borda para jogos em dispositivos móveis. In: *Anais do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Porto Alegre, RS, Brasil: SBC, 2023. p. 574–587. ISSN 2177-9384. Disponível em: <https://sol.sbc.org.br/index.php/sbrc/article/view/24566>. Citado na página 23.

20 BANIJAMALI, A. et al. Software architectures of the convergence of cloud computing and the internet of things: A systematic literature review. *Information and Software Technology*, v. 122, p. 106271, 2020. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584920300215>. Citado na página 23.

21 BARI, M. F. et al. Data center network virtualization: A survey. *IEEE Communications Surveys and Tutorials*, v. 15, n. 2, p. 909–928, 2013. Citado na página 24.

22 XAVIER, M. G. et al. Performance evaluation of container-based virtualization for high performance computing environments. In: *Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. [S.l.: s.n.], 2013. p. 233–240. Citado na página 24.

23 PALLEWATTA, S.; KOSTAKOS, V.; BUYYA, R. Microservices-based iot application placement within heterogeneous and resource constrained fog computing environments. In: *IEEE/ACM International Conference on Utility and Cloud Computing*. [S.l.: s.n.], 2019. p. 71–81. Citado na página 25.

24 GANNON, D.; BARGA, R.; SUNDARESAN, N. Cloud-native applications. *IEEE Cloud Computing*, IEEE, v. 4, n. 5, p. 16–21, 2017. Citado na página 25.

25 ANWAR, A. et al. Improving docker registry design based on production workload analysis. In: USENIX. *USENIX Conference on File and Storage Technologies*. [S.l.], 2018. p. 265–278. Citado 2 vezes nas páginas 26 e 28.

26 HARTER, T. et al. Slacker: Fast distribution with lazy docker containers. In: USENIX. *USENIX Conference on File and Storage Technologies*. [S.l.], 2016. p. 181–195. Citado 2 vezes nas páginas 26 e 28.

27 CHEN, J. L. et al. Starlight: Fast container provisioning on the edge and over the {WAN}. In: USENIX. *USENIX Symposium on Networked Systems Design and Implementation*. [S.l.], 2022. p. 35–50. Citado 2 vezes nas páginas 27 e 28.

28 NATHAN, S. et al. Comicon: A co-operative management system for docker container images. In: IEEE. *International Conference on Cloud Engineering*. [S.l.], 2017. p. 116–126. Citado 2 vezes nas páginas 27 e 28.

29 BECKER, S.; SCHMIDT, F.; KAO, O. Edgepier: P2p-based container image distribution in edge computing environments. In: IEEE. *International Performance, Computing, and Communications Conference*. [S.l.], 2021. p. 1–8. Citado 2 vezes nas páginas 27 e 28.

30 AHMED, A.; PIERRE, G. Docker image sharing in distributed fog infrastructures. In: IEEE. *International Conference on Cloud Computing Technology and Science*. [S.l.], 2019. p. 135–142. Citado 2 vezes nas páginas 27 e 28.

31 KNOB, L. A. D. et al. Community-based placement of registries to speed up application deployment on edge computing. In: *International Conference on Cloud Engineering*. [S.l.: s.n.], 2021. p. 147–153. Citado 2 vezes nas páginas 27 e 28.

32 SOUZA, P. S.; FERRETO, T.; CALHEIROS, R. N. Edgesimpy: Python-based modeling and simulation of edge computing resource management policies. *Future Generation Computer Systems*, 2023. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X23002340>. Citado na página 29.

33 ARAL, A.; DEMAIO, V.; BRANDIC, I. Ares: Reliable and sustainable edge provisioning for wireless sensor networks. *IEEE Transactions on Sustainable Computing*, p. 1–12, 2021. Citado na página 29.

34 DIJKSTRA, E. W. et al. A note on two problems in connexion with graphs. *Numerische mathematik*, v. 1, n. 1, p. 269–271, 1959. Citado na página 30.

35 AHMED, A.; PIERRE, G. Docker container deployment in fog computing infrastructures. In: *IEEE International Conference on Edge Computing*. [S.l.: s.n.], 2018. p. 1–8. Citado na página 31.

36 BAI, F.; HELMY, A. A survey of mobility models. *Wireless Adhoc Networks. University of Southern California, USA*, v. 206, p. 147, 2004. Citado na página 31.

37 BARABÁSI, A.-L.; ALBERT, R. Emergence of scaling in random networks. *science*, American Association for the Advancement of Science, v. 286, n. 5439, p. 509–512, 1999. Citado na página 31.

38 YAO, H. et al. Migrate or not? exploring virtual machine migration in roadside cloudlet-based vehicular cloud. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, v. 27, n. 18, p. 5780–5792, 2015. Citado na página 32.