ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

RAFAEL MACHADO RIOS

TÉCNICAS ENERGETICAMENTE EFICIENTES PARA O POSICIONAMENTO DE APLICAÇÕES EM COMPUTAÇÃO NA BORDA

Porto Alegre
2022

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TÉCNICAS ENERGETICAMENTE EFICIENTES PARA O POSICIONAMENTO DE APLICAÇÕES EM COMPUTAÇÃO NA BORDA

RAFAEL MACHADO RIOS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Tiago Coelho Ferreto

Porto Alegre
2022

Ficha Catalográfica

R586t Rios, Rafael Machado

Técnicas energeticamente eficientes para o posicionamento de aplicações em computação na borda / Rafael Machado Rios. – 2022.

77 p.

Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Tiago Coelho Ferreto.

1. Computação na Borda. 2. Eficiência Energética. 3. Posicionamento de Aplicações. I. Ferreto, Tiago Coelho. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS com os dados fornecidos pelo(a) autor(a).

Bibliotecária responsável: Clarissa Jesinska Selbach CRB-10/2051

RAFAEL MACHADO RIOS

TÉCNICAS ENERGETICAMENTE EFICIENTES PARA O POSICIONAMENTO DE APLICAÇÕES EM COMPUTAÇÃO NA BORDA

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação, Escola Politécnica da Pontifícia Universidade Católica do Rio Grande do Sul.

Aprovado(a) em 20 de Dezembro de 2022.

BANCA EXAMINADORA:

Prof. Dr. Fábio Diniz Rossi (IFFar)

Prof. Dr. César A. F. De Rose (PPGCC/PUCRS)

Prof. Dr. Tiago Coelho Ferreto (PPGCC/PUCRS - Orientador)

DEDICATÓRIA

Dedico este trabalho a meus pais, Angelita e Roberto, à minha namorada, Bárbara, e aos demais membros da minha família pelo apoio durante a realização do mestrado.

AGRADECIMENTOS

Primeiramente, gostaria de agradecer ao Prof. Tiago Ferreto pela orientação. Agradeço também ao Luis Knob e ao Carlos Kayser pelas contribuições no desenvolvimento do simulador ECOS, este último em especial pelo auxílio com a redação deste documento.

Agradeço à PUCRS por toda esta jornada, que envolve: a Bolsa Mérito em Engenharia da Computação; a Bolsa ENADE, a qual possibilitou realizar uma especialização em Experiência do Usuário; e esta oportunidade de me aprimorar no mundo acadêmico através do mestrado.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior — Brasil (CAPES) — Código de Financiamento 001.

O presente trabalho foi alcançado em cooperação com a Hewlett Packard Brasil LTDA. e com recursos provenientes da Lei de Informática (Lei nž 8.248, de 1991).

TÉCNICAS ENERGETICAMENTE EFICIENTES PARA O POSICIONAMENTO DE APLICAÇÕES EM COMPUTAÇÃO NA BORDA

RESUMO

O setor de Tecnologia da Informação e Comunicação (TIC) tornou-se relevante na economia global e a tendência é de que ele continue crescendo nos próximos anos. Esse setor utiliza dispositivos cujo funcionamento consome energia e produz calor, o que é problemático por motivos ambientais, socioeconômicos e tecnológicos. As técnicas de redução do consumo de energia destes dispositivos tipicamente envolvem seu desligamento temporário ou a utilização de estados de baixo consumo de energia, os quais podem impactar negativamente o desempenho do sistema. No paradigma de Computação na Borda, no entanto, aplicações usualmente possuem requisitos como baixa latência e baixo tempo de resposta. Este trabalho introduz duas técnicas de posicionamento de aplicações com foco no equilíbrio do compromisso entre eficiência energética e responsividade do sistema. A primeira técnica envolve a criação de um algoritmo de escalonamento baseado em prioridades com informações sobre a disponibilidade de um nodo. A segunda técnica utiliza *Service Level Agreements* (SLAs) de tempo de provisionamento e informações de disponibilidade dos nodos em um algoritmo genético de múltiplos objetivos na decisão de posicionamento das aplicações. Os resultados demonstram a eficácia de ambas as técnicas na redução do consumo de energia e na minimização das violações de SLAs.

Palavras-Chave: computação na borda, eficiência energética, posicionamento de aplicações.

ENERGY-EFFICIENT TECHNIQUES FOR TASK PLACEMENT OF CONTAINER-BASED APPLICATIONS ON EDGE COMPUTING

ABSTRACT

Information and Communication Technology has become an important sector in the global economy and the tendency is that it keeps growing in the foreseeable future. This sector utilizes devices whose functioning wastes energy and produces heat, which is problematic for environmental, socioeconomical and technological reasons. Techniques for reducing energy consumption of such devices typically involve shutting them off or placing them in low-power states, which may impact overall system performance. In the Edge Computing paradigm, however, applications usually have strict requirements such as low latency and low response time. This work presents two application placement techniques with the objective of balancing the tradeoff between energy efficiency and system responsiveness. The first technique involves the creation of a priority-based scheduling algorithm with node-availability information, while the second utilizes a deployment-SLA and node availability-driven scheduler using a multi-objective genetic algorithm. Results show the effectiveness of both application placement techniques in reducing energy consumption while minimizing Service-Level Agreement violations.

Keywords: edge computing, energy efficiency, application placement.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE SIGLAS

AAS – *Availability-Aware Scheduler*

AP – *Availability Priority*

CDN – *Content-Delivery Network*

CPU – *Central Processing Unit*

CVS – *Coordinated Voltage Scaling*

DLSLA – *Deployment Latency SLA Enforcement Scheduler*

DVFS – *Dynamic Voltage and Frequency Scaling*

EA-DLSLA – *Energy-Aware Deployment Latency Enforcement Scheduler*

ECOS – *Edge Computing Orchestrating Simulator*

IOT – *Internet of Things*

IVS – *Independent Voltage Scaling*

MEC – *Mobile Edge Computing*

NSGA-II – *Non-dominated Sorting Genetic Algorithm II*

QOS – *Quality of Service*

SBC – *Single Board Computer*

SDN – *Software-Defined Network*

SLA – *Service-Level Agreement*

TIC – Tecnologia da Informação e Comunicação

VM – *Virtual Machine*

VOVO – *Vary-On Vary-Off*

SUMÁRIO

1. INTRODUÇÃO

O setor de Tecnologia da Informação e Comunicação (TIC) é um dos que mais tem crescido nos últimos anos, impactando diversos outros setores [1]. Os *data centers* e a infraestrutura de rede fazem uso de máquinas de alto desempenho e alta disponibilidade para garantir a qualidade destes serviços. No entanto, acompanhando este crescimento da área de TIC, surgem preocupações quanto ao consumo de energia e a emissão de carbono. Estima-se que este setor tenha sido responsável por 2 a 5% das emissões globais de carbono em 2015 [2]. Logo, a questão da eficiência energética tornou-se uma questão central para as indústrias, devido a motivos econômicos e ambientais.

Devido à miniaturização dos chips de computador e à fabricação de sistemas embarcados com alta capacidade de desempenho [3], a Internet das Coisas (IoT, do inglês, *Internet of Things*) também se popularizou. Isso ocasionou o surgimento de aplicações com um conjunto de requisitos que não se adequam ao modelo tradicional de Computação na Nuvem, como o baixo tempo de resposta e a privacidade dos dados. Sendo assim, o paradigma de Computação na Borda (em inglês, Edge Computing) atua como um intermediário entre aplicações IoT e serviços de Computação na Nuvem através dos chamados *edge devices*.

Assim como na Computação na Nuvem, o gerenciamento e o provisionamento de aplicações são fundamentais para garantir um bom desempenho em sistemas de Computação na Borda [4]. Atualmente, é comum o uso da virtualização baseada em contêineres [5] para providenciar alta escalabilidade, facilidade de manutenção e baixo tempo de provisionamento às aplicações na borda [6].

No entanto, diferentemente da nuvem, onde a infraestrutura é majoritariamente baseada em *data centers*, os quais possuem conjuntos homogêneos de servidores, a borda geralmente possui uma infraestrutura de rede difusa, envolvendo *edge devices* geodistribuídos e com características diferentes quanto aos recursos computacionais e ao consumo de energia [7]. Desse modo, destaca-se a importância de algoritmos eficientes para o posicionamento das aplicações na borda, visando cumprir seus requisitos, assim como minimizar o consumo de energia do sistema.

Data centers tipicamente utilizam em média apenas 30% dos seus recursos [8], uma vez que estes são provisionados acima do necessário para garantir o funcionamento

adequado durante picos de demanda. Esse sobre-provisionamento gera ineficiências do ponto de vista de consumo energético. Sendo assim, diversas técnicas de redução do consumo de energia em sistemas de Computação na Nuvem utilizam técnicas que envolvem o desligamento de máquinas inativas [9, 10, 11, 12], ligando-as novamente conforme aumenta a demanda computacional.

A consolidação das aplicações, a qual envolve agrupá-las em uma quantidade menor de máquinas físicas, concentra a utilização de recursos nestes poucos dispositivos. No entanto, *edge devices* tipicamente possuem menor capacidade computacional e maior tempo de inicialização quando comparados aos servidores utilizados em *data centers* [7]. Além disso, aplicações de Computação na Borda tipicamente possuem características diferentes do que as de Computação na Nuvem, como requisitos de baixa latência.

Portanto, visto que o desligamento de nodos pode impactar negativamente o desempenho em *edge devices*, muitos dos trabalhos os quais visam a redução do consumo de energia em sistemas de Computação na Borda [13, 14, 15, 16, 17, 18, 19, 20, 21] não empregam técnicas deste tipo, utilizando, como alternativa, a migração de serviços (do inglês, *application offloading*) da borda para a nuvem e o posicionamento eficiente de aplicações (do inglês, *application placement*) como formas de aumentar a eficiência energética.

O *offloading* de aplicações, entretanto, não é necessariamente viável em todos os casos de uso da Computação na Borda. Algumas aplicações possuem restrições quanto à confidencialidade dos dados e proximidade de execução [22]. Além disso, ao migrar tarefas da borda para a nuvem, é possível que o desempenho do sistema seja impactado negativamente devido ao aumento do tempo de resposta.

Dessa maneira, um dos objetivos deste trabalho é investigar a discrepância em sistemas de Computação na Nuvem e na Borda quanto às técnicas de aumento de eficiência energética que utilizam o posicionamento de aplicações. Especificamente, propõe-se o desligamento de *edge devices* subutilizados em combinação à algoritmos de otimização dos requisitos de aplicações da Computação na Borda como forma de maximizar tanto a eficiência energética quanto a responsividade do sistema.

As contribuições deste trabalho se dão na forma de dois algoritmos de escalonamento para aplicações em contêineres: o *Availability-Aware Scheduler* (AAS) e o *Energy-Aware Deployment Latency Enforcement Scheduler* (EA-DLSLA). O AAS consiste em pon-

tuar os nodos elegíveis a receber as aplicações baseando-se em prioridades como o estado de energia de uma máquina. O EA-DLSLA utiliza *Service Level Agreements* (SLAs) de tempo de provisionamento e o tempo de inicialização dos nodos, caso estejam desligados, em um algoritmo genético multi-objetivo para a decisão de posicionamento das aplicações.

Os diferentes casos de teste, os quais envolvem as soluções propostas neste trabalho, o escalonador padrão do orquestrador de contêineres *Kubernetes* [23], bem como uma variação eficientemente energética, e a técnica relacionada descrita em [21], foram validados por simulação. Os resultados demonstram a eficácia do AAS e o EA-DLSLA na redução do consumo de energia e na minimização das violações de SLAs.

1.1 Organização do trabalho

O restante deste trabalho está organizado da seguinte maneira: o Capítulo 2 apresenta a fundamentação teórica, abordando as características da computação na borda, a virtualização baseada em contêineres e o funcionamento da plataforma Kubernetes. O Capítulo 3 descreve o estado da arte na área de posicionamento eficientemente energético de aplicações na borda. Os algoritmos para escalonamento de contêineres visando a redução do consumo energético, o AAS e o EA-DLSLA, são apresentados no Capítulo 4, enquanto os Capítulos 5 e 6 apresentam, respectivamente, a metodologia dos experimentos e a avaliação dos resultados. Por fim, o Capítulo 7 expõe as considerações finais e trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentado o embasamento teórico do trabalho desenvolvido. A Seção 2.1 apresenta as características de Computação na Borda bem como suas vantagens, desvantagens e modelos de implementação. A seguir, a Seção 2.2 introduz conceitos relacionados à virtualização baseada em contêineres, assim como descreve as plataformas Docker e Kubernetes, utilizadas para o gerenciamento e orquestração, respectivamente, das aplicações sob este modelo de provisionamento.

2.1 Computação na Borda

Aplicações IoT possuem algumas características típicas [24]: requerem baixa latência, podem envolver informações confidenciais, e devido ao crescente número de dispositivos IoT, têm potencial de produzir uma quantidade de dados capaz de sobrecarregar a rede. A Computação em Nuvem não é suficiente para suportar essas aplicações, uma vez que ela não necessariamente cumpre estes requisitos de baixa latência e localidade dos dados [25].

A Computação na Borda se refere às tecnologias que permitem a realização de computação na borda das redes, ou seja, ela atua como intermediário entre dispositivos IoT e serviços de Computação em Nuvem. A “borda” é definida como quaisquer recursos computacionais e de rede situados entre a origem dos dados, a partir das aplicações IoT, e seu destino, nos *data centers*[26]. A ideia geral do paradigma da Computação na Borda é que a computação ocorra próxima à origem dos dados, isto é, próxima aos dispositivos IoT.

A Figura 2.1 demonstra alguns casos de uso de Computação na Borda. A nuvem localiza-se em um extremo, enquanto os dispositos IoT, como carros inteligentes, sensores da rede de energia e atuadores industriais, localizam-se no outro extremo. Entre as aplicações IoT e os *data centers* na nuvem, os *edge devices* atuam no processamento e na comunicação dos dados no meio destas camadas. Como é possível perceber, a tendência é de que os dados sejam produzidos cada vez mais na borda da rede, sendo assim, é exigida uma maior eficiência no processamento destes dados na própria borda.

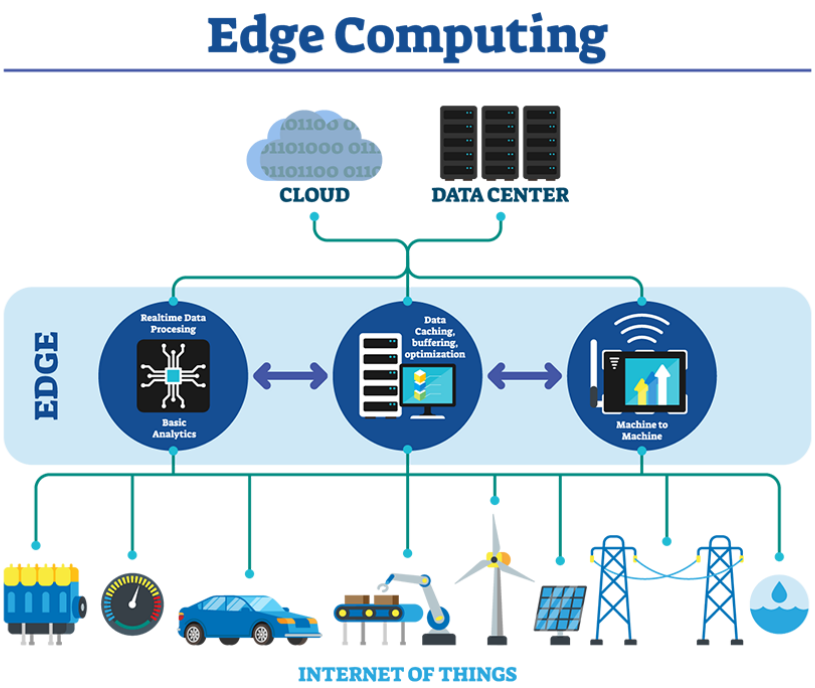

Figura 2.1: Casos de Uso de Computação na Borda

Fonte: IEEE (2019) [27]

Ao processar na nuvem os dados oriundos de aplicações IoT, espera-se que seu processamento ocorra mais rapidamente do que se fosse realizado nos próprios dispositivos IoT, uma vez que os recursos computacionais da nuvem possuem maior desempenho do que os da borda [25]. Porém, comparada com o rápido crescimento da quantidade de dados gerados e de suas demandas de processamento, a largura de banda da rede não acompanhou estes crescimentos [28].

Devido à enorme quantidade de dados gerados por dispositivos IoT, a velocidade e a vazão do transporte de dados através da rede tornam-se o gargalo em sistemas de Computação na Nuvem [7]. Em muitos casos, se toda a informação for enviada para o processamento na nuvem, o tempo de resposta da aplicação não cumprirá os requisitos de latência esperados de aplicações IoT. Desse modo, a informação deve ser processada na borda para se obter um tempo de resposta menor, um processamento mais eficiente dos dados e para que se reduza a sobrecarga da estrutura da rede. Além disso, o requisito de proteger a privacidade das informações torna-se um obstáculo ao enviar os dados para serem processados na nuvem [29]. Por fim, grande parte dos dispositivos IoT possuem limitações quanto ao consumo de energia [30], e os módulos de comunicação sem fio geralmente possuem baixa eficiência energética, por isso justifica-se delegar uma parcela da computação aos dispositivos na borda.

2.1.1 Vantagens

As vantagens da Computação na Borda estão relacionadas à proximidade entre os dispositivos, posto que a proximidade física afeta a latência, largura de banda e confiabilidade. A proximidade de *edge devices* possui, segundo M. Satyanarayanan [7], quatro vantagens em relação à Computação em Nuvem: serviços da nuvem altamente responsivos, escalabilidade através do pré-processamento de dados na borda, imposição de diretrizes de privacidade, e mascaramento de quedas na nuvem.

A proximidade física de um *edge device* à um dispositivo móvel torna mais fácil atingir menores latências, maiores larguras de banda e menor *jitter*. Isso é relevante em aplicações de realidade aumentada ou de realidade virtual, visto que o ser humano é extremamente sensível a atrasos nas interações, e seu desempenho em tarefas cognitivas é rápido e preciso. Um humano necessita de apenas 4ms para identificar um som como uma voz humana [31] e estima-se que aplicações de realidade virtual devem possuir latência de menos de 16ms para gerar uma impressão de fluidez [32].

Edge devices podem reduzir de forma geral o tráfego de dados que são enviados para a nuvem. Em aplicações automotivas, o dispositivo poderia realizar análise de dados em tempo real do fluxo de dados gerados a partir de sensores localizados no motor do automóvel ou de outras fontes de forma a alertar o motorista de uma falha iminente ou da necessidade de manutenção preventiva [33]. Além disso, *analytics* em tempo real podem ser úteis em um sistema colaborativo de segurança no trânsito, como por exemplo, ao compartilhar os locais onde existam buracos ou árvores caídas para o dispositivo localizado na borda, o qual repassaria essa informação a veículos próximos. Por fim, vídeos produzidos por *smartphones* poderiam ser processados localmente. Ao utilizar visão computacional, os dispositivos móveis enviariam para a nuvem somente o produto final, já processado, como faces reconhecidas e etiquetas de conteúdo, assim como os metadados, por exemplo: proprietário, localização e data [7].

Por último, a proximidade física de *edge devices* com os dispositivos móveis e IoT reduz a dependência deles com a nuvem. Atualmente, arquiteturas IoT assumem que a nuvem é acessível a todo instante, ou seja, que a rede sempre é de qualidade e que não existem falhas nela ou na nuvem. Contudo, essa não é a realidade: existem locais onde a infraestrutura de rede é limitada, ou onde há interferências que impossibilitam a comu-

nicação eficiente entre os dispositivos. Através da Computação na Borda, o*edge device* poderia atuar como uma opção de contingência que mascararia falhas temporárias de comunicação com a nuvem. Durante falhas, um servidor localizado na borda poderia, de modo transparente aos dispositivos IoT, assumir as responsabilidades da nuvem e realizar seus serviços críticos.

2.1.2 Desvantagens

O paradigma da Computação na Borda implica uma maior complexidade na infraestrutura da rede [25]. Apesar de todas as vantagens, é preciso adaptar as soluções Cloud-Iot para se comunicarem através de um intermediário, o *edge device*, sendo necessário introduzi-lo ao sistema, configurá-lo e realizar manutenção sua quando for preciso. Além do mais, é preciso proteger o *edge device*, não só virtualmente, mas também fisicamente: como o dispositivo localiza-se na borda, é provável que não possua toda a segurança de um *datacenter* tradicional, sendo mais passível de furto e de depredação [29].

Dispositivos como *smartphones* e *tablets* frequentemente são utilizados para compartilhar fotos e vídeos de alta resolução em redes sociais, como o *Instagram* [34], o *Facebook* [35], ou o *Youtube*, [36]. Além disso, essas mídias também são comumente armazenadas em serviços como *OneDrive* [37] e *Google Drive* [38], os quais utilizam a infraestrutura da Computação na Nuvem.

Tendo em vista essa situação, os arquivos de mídia gerados pelos dispositivos móveis podem ter um tamanho considerável, consumindo consideráveis recursos da rede para realizar a transferência IoT-Nuvem. Neste caso, através de *edge devices*, as mídias podem ser convertidas para uma resolução menor ou com um formato de maior compressão antes de serem enviadas para a nuvem. Outro exemplo são dispositivos médicos IoT: uma vez que as informações coletadas por esses dispositivos são geralmente particulares, a privacidade do usuário seria melhor protegida ao processar os dados localmente em vez de enviá-los à nuvem. No entanto, a adição dessas funcionalidades de pré-processamento aos *edge devices* exige mais recursos computacionais e de transmissão de dados na infraestrutura da borda.

2.1.3 Implementações

A classificação dos sistemas que atuam como intermediários entre a nuvem e dispositivos IoT varia quanto ao tipo de dispositivos utilizados como *edge devices*, aos protocolos de rede e comunicação aplicados à camada de borda, e aos serviços oferecidos pelo sistema. Os modelos de implementação de Computação na Borda são [39]:*Mobile Edge Computing* (MEC), *Fog Computing* e *Cloudlet Computing*.

MEC pode ser definida como a implementação da Computação na Borda junto a Estações de Rádio Base (ERBs) [39], isto é, os equipamentos que fazem a conexão entre os telefones celulares e a companhia telefônica [40]. ERBs são compostas por antenas, equipamentos de transmissão e recepção, e infraestruturas auxiliares como estações de energia e sistemas de segurança. Sendo assim, os *edge devices* utilizados em sistemas de MEC não necessariamente possuem restrições quanto ao tamanho ou consumo de energia, sendo possível utilizar servidores convencionais, porém em menor quantidade do que em um data center. A Figura 2.2 demonstra um sistema de MEC desenvolvido pela empresa *American Tower* [41].

Figura 2.2: Visões externa e interna de um sistema de *Mobile Edge Computing*

Fonte: Data Center Frontier (2020) [42]

A *Fog Computing* é uma implementação decentralizada da infraestrutura computacional [39]. Os *edge devices* possuem natureza heterogênea e podem ser baseados em dispositivos tanto como servidores convencionais quanto roteadores, *switches*, pontos de acesso e *gateways* IoT.

Por fim, *cloudlets* podem ser descritos como *data centers* em pequena escala, tendo a capacidade de se conectar a serviços da nuvem e a dispositivos IoT [7]. As *cloudlets* são compostas por *clusters* de computadores de tamanho reduzido e de baixo consumo de energia, denominados *Single Board Computers* (SBCs). Esses *clusters* ficam localizados próximos aos usuários, disponibilizando aplicações de baixa latência.

2.2 Virtualização Baseada em Contêineres

Contêineres são um modo de garantir a consistência e portabilidade ao executar aplicações [43]. Esta tecnologia aproxima os ambientes de desenvolvimento, teste e produção ao aumentar a previsibilidade do sistema. Quando se gera um contêiner contendo uma aplicação ou serviço, assegura-se de que o software e suas dependências serão executados sobre o mesmo ambiente, evitando conflitos de versionamento ou de variáveis de ambiente entre componentes das aplicações.

Assim como máquinas virtuais (VMs, do inglês, *Virtual Machines*), contêineres executam sob uma máquina hospedeira. A aplicação encapsulada no contêiner, por padrão, não tem acesso a recursos externos a ela, como rede e sistema de arquivos da máquina hospedeira [43]. No entanto, como pode se observar na Figura 2.3, um contêiner não realiza a virtualização de todo um sistema operacional, uma vez que os contêineres lançados em uma máquina hospedeira compartilham o mesmo *kernel*. O *kernel* do Linux permite isolamento de recursos como memória, CPU, *Input/Output* (I/O) e rede. LinuX Containers (LXC) [44], surgida em 2008, é a implementação do gerenciador de contêineres Linux, tendo sido desenvolvida com base nas funcionalidades de *cgroups* e *namespaces* do Linux. O LXC disponibiliza ferramentas para o gerenciamento de contêineres, provê suporte a rede e armazenamento, e oferece uma biblioteca, denominada *liblxc*, para desenvolver aplicações que façam uso do LXC [45].

2.2.1 Plataforma Docker

Atualmente, uma das plataforma de gerenciamento de contêineres mais utilizada é a Docker [47]. A ferramenta Docker facilita a criação de imagens de contêineres, além de permitir a gerência do ciclo de vida dos contêineres, através de comandos como iniciar,

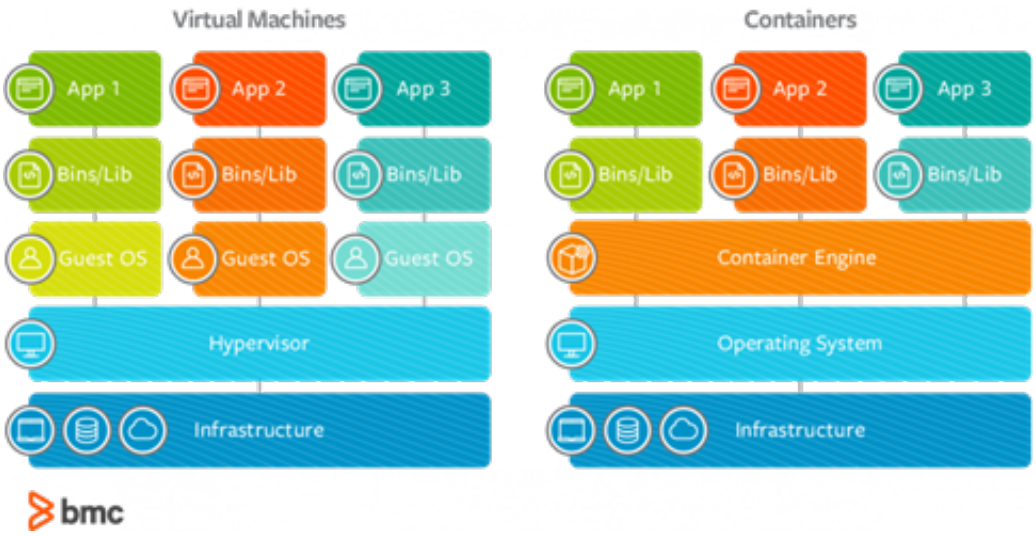

Figura 2.3: Comparação entre arquitetura de uma máquina virtual e de um contêiner

Fonte: BMC (2020) [46]

pausar, reiniciar e remover [48]. Um contêiner Docker mantém os itens necessários para a execução da aplicação, tipicamente contendo o *userspace* do sistema operacional, os arquivos adicionados pelo usuário e alguns metadados que são utilizados para gerenciamento do contêiner [49].

As imagens Docker podem ser consideradas como modelos a partir dos quais são gerados os contêineres. Essas imagens são construídas através de um conjunto de instruções contidas em um arquivo chamado Dockerfile. As instruções executam ações como: basear a nova imagem em outra já existente, adicionar arquivos ou pastas à imagem, executar uma série de comandos, definir variáveis de ambiente ou selecionar o comando inicial que será executado ao criar o contêiner.

Além dos benefícios como inicialização rápida e menor consumo de recursos em comparação a VMs [43], as imagens dos contêineres tendem a ocupar menos espaço de armazenamento devido a suas composições em camadas. Cada instrução em um Dockerfile cria uma camada unicamente identificável, as quais são compartilhadas em imagens que possuem estas camadas em comum.

Além da Docker, outros componentes são frequentemente utilizados em conjunto com a plataforma: o Docker CLI e o Docker Registry. A ferramenta de linha de comando (CLI - do inglês, *Command Line Interface*) Docker CLI pode ser utilizada para realizar *upload* ou *download* de novas imagens a partir do Registry, o qual é um repositório para armazenar e distribuir imagens Docker [50].

2.2.2 Orquestradores de Contêineres

Orquestradores de contêineres são ferramentas que visam solucionar alguns dos desafios na criação de aplicações e serviços baseados em contêineres, como provisionamento, escalabilidade, versionamento e estabilidade [51]. Os orquestradores proveem mecanismos para gerenciar contêineres com o mínimo de intervenção dos administradores do sistema, como a configuração declarativa, provisionamento, descoberta de serviços e monitoramento.

A configuração declarativa permite que o usuário da ferramenta de orquestração defina o estado desejado da aplicação ao invés de manualmente configurar os contêineres. O orquestrador constantemente tenta atingir esse estado baseado no estado atual do sistema e no plano declarativo, o qual contém dados sobre como deveria ser o comportamento do sistema [23].

O orquestrador assume a responsabilidade de distribuir aplicações e seus contêineres pelas diversas máquinas físicas que compõe o *cluster*. Esse provisionamento deve ser feito de maneira inteligente, utilizando métricas como o uso de recursos e tempo de resposta [51].

2.2.3 Kubernetes

O Kubernetes é um *middleware* que possibilita a execução e coordenação de aplicações encapsuladas em contêineres através de um *cluster* de máquinas [52]. É uma plataforma confeccionada para gerenciar o ciclo de vida de aplicações e serviços utilizando métodos que forneçam escalabilidade, confiabilidade e alta disponibilidade. Segundo (Ellingwood, 2018) [52]:

> O Kubernetes pode ser visto como um sistema construído em camadas, com cada camada mais alta abstraindo a complexidade encontrada nos níveis mais baixos. [...] Em sua base, o Kubernetes constrói um *cluster* a partir de máquinas físicas individuais através do uso de uma rede compartilhada para a comunicação entre cada máquina. Esse *cluster* é a plataforma física onde todos os objetos, componentes, e cargas de trabalho do Kubernetes são configurados.

Os diversos componentes do Kubernetes certificam-se de que o estado desejado das aplicações corresponda ao estado real do *cluster* através da configuração declarativa[23]. Os usuários interagem com o *cluster* através da API disponibilizada pela máquina

mestre. O servidor mestre utiliza a configuração declarativa e toma decisões com base nos recursos e no estado atual do sistema para atingir o estado desejado.

As aplicações e serviços propriamente ditos executam no *cluster* encapsulados em contêineres. A 2.4 demonstra como aplicações e serviços poderiam ser distribuídos em um *cluster*. Os contêineres são contidos em Pods, que são a unidade básica para construir aplicações no Kubernetes, e essas Pods podem ser enviadas para execução em qualquer nodo disponível no *cluster*.

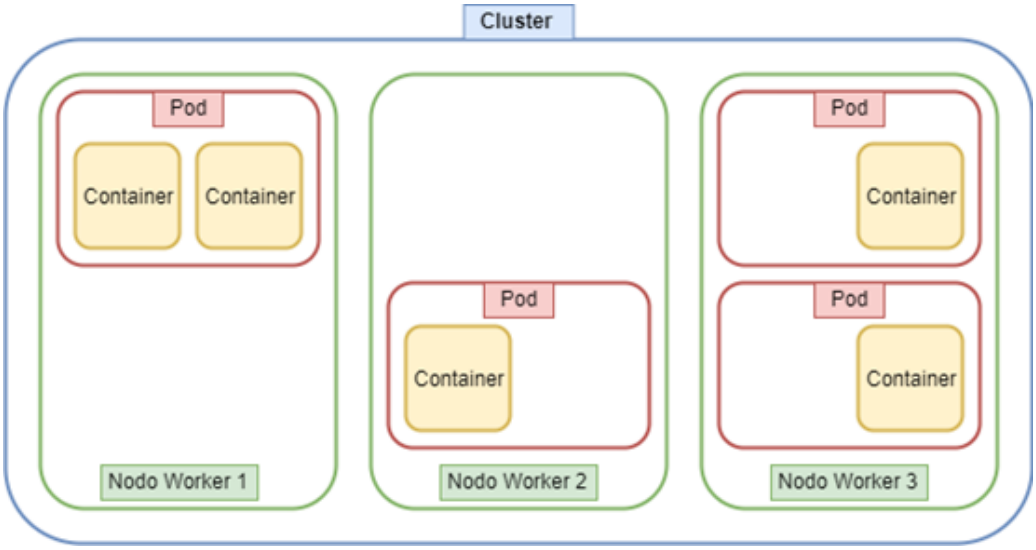

Figura 2.4: Ilustração simplificada da arquitetura do Kubernetes

Fonte: O Autor (2019)

A Figura 2.5 demonstra a relação entre alguns dos objetos da plataforma Kubernetes. A documentação do Kubernetes define um objeto como [53] um registro de intenção. Ao criar um objeto, o Kubernetes trabalha de maneira constante para garantir que o objeto exista, efetivamente tentando atingir o estado desejado do *cluster*.

O estado das entidades do sistema é representado pelos objetos do Kubernetes [55]. Esses objetos atuam como uma camada adicional de abstração sobre a interface de contêineres. Sendo assim, a interação com as aplicações e serviços é feita com os objetos do Kubernetes ao invés de interagir diretamente com os contêineres. Os objetos básicos do Kubernetes são [55]:

- **Pod** — Uma Pod é um conjunto de um ou mais contêineres com rede e armazenamento compartilhados o qual possui uma especificação sobre como executá-los.

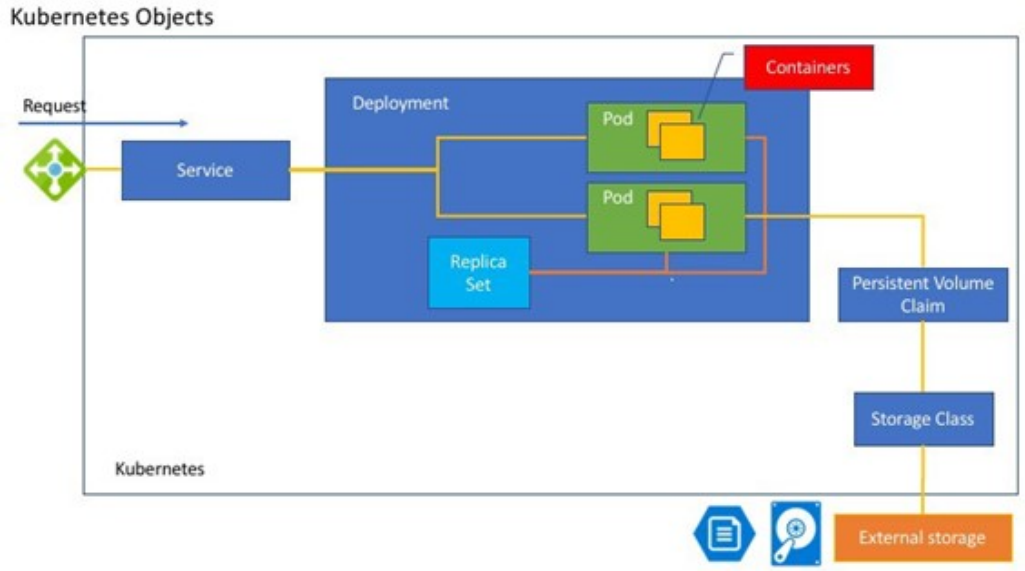

Figura 2.5: Diagrama das relações entre os objetos do Kubernetes

Fonte: Ushio (2018) [54]

- **Service** — É uma abstração que define um conjunto lógico de Pods e como acessá-las.
- **Volume** — É um espaço de armazenamento acessível a todos os contêineres dentro de uma Pod específica.
- **Namespaces** — Possibilitam a orquestração de *clusters* virtuais em um único *cluster* físico.

O Kubernetes também disponibiliza diversos controladores, os quais são baseados nos objetos básicos e possuem funcionalidades adicionais. O Kubernetes inclui, por padrão, os seguintes controladores [55]:

- **ReplicaSet** — Compromete-se de que um certo número de Pods sempre estará disponível e operacional no *cluster*.
- **Deployment** — É responsável pelo provisionamento de Pods e ReplicaSets. Este controlador monitora o estado do provisionamento para atingir o estado desejado.
- **StatefulSet** — É responsável por garantir um provisionamento ordenado e armazenamento persistente.
- **DaemonSet** — Garante que todos os nodos do *cluster* executem uma cópia de uma Pod.

- **Job** — É utilizado para realizar uma tarefa e encerra-se ao completá-la ou após um determinado intervalo de tempo.

A Figura 2.6 exibe uma versão detalhada da arquitetura do Kubernetes. No desenvolvimento deste trabalho, são relevantes os componentes do nodo mestre, especificamente o *kube-scheduler*. De acordo com a documentação do Kubernetes (The Kubernetes Authors, 2022) [56] o nodo mestre é definido como:

> O "mestre"se refere à coleção de processos gerenciando o estado do *cluster*. Tipicamente, estes processos rodam todos em um único nodo no *cluster*, e este nodo também é referenciado como o mestre. O [nodo] mestre também pode ser replicado para disponibilidade e redundância. O nodo mestre do Kubernetes controla e coordena todos os nodos no *cluster* com o auxílio de alguns processos [56]: o *etcd*, o *kube-apiserver*, o *kube-controller-manager* e o *kube-scheduler*.

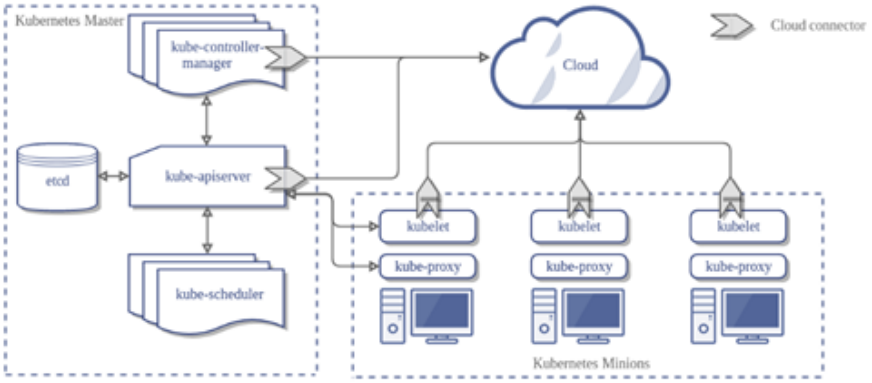

Figura 2.6: Diagrama dos componentes do Kubernetes

Fonte: Kubernetes (2022) [56]

O *etcd* é uma aplicação auxiliar que possui a função de armazenar as informações de configuração e torná-las acessíveis aos nodos do *cluster*. O *kube-apiserver* implementa uma interface REST (do inglês, Representation State Transfer) [57] para possibilitar a comunicação de ferramentas e bibliotecas com o *cluster*. O controlador *kube-controller-manager* é responsável por gerenciar o estado do *cluster* através da coordenação dos diferentes tipos de controladores. Por fim, o *kube-scheduler* distribui e programa a carga de trabalho para os nodos trabalhadores do *cluster*.

2.2.4 Diretrizes do *kube-scheduler*

A função do *kube-scheduler* é selecionar um nodo para realizar o provisionamento de Pods recém criadas ou que ainda não foram escalonadas. A partir da versão 1.23 do

Kubernetes, lançada em dezembro de 2021, o comportamento do escalonador é definido através de perfis de escalonamento [58]. No entanto, ao longo do desenvolvimento deste trabalho, ainda utiliza-se o método de seleção baseado nas etapas de filtragem e classificação [59].

Uma vez que aplicações possuem requisitos diferentes quanto à utilização de recursos, somente devem ser selecionados os nodos que possam cumprir os requisitos dessas aplicações. Sendo assim, a etapa de filtragem elimina os nodos inválidos através de diretrizes baseadas em predicados, como:

- **PodFitsResources**: classifica o nodo como elegível baseado na disponibilidade dos recursos computacionais;
- **MatchNodeSelector**: define a elegibilidade de acordo com *labels* e *tags* definidas nas Pods e nos nodos, permitindo atribuições customizadas, como localidade;
- **MatchInterPodAffinity**: verifica se o nodo já está executando uma Pod incompatível com a que se pretende provisionar.

Após a etapa de filtragem, o *kube-scheduler* atribui pontuações aos nodos elegíveis através de um conjunto de prioridades. Algumas destas prioridades são:

- **SelectorSpreadPriority**: prioriza a distribuição de Pods em vários nodos;
- **InterPodAffinityPriority**: assim como na *MatchInterPodAffinity*, atribui pontuações aos nodos dependendo da compatibilidade das Pods já instaladas nele com a que deverá ser provisionada;
- **LeastRequestedPriority**: atribui maior pontuação a nodos com as menores solicitação de recursos;
- **MostRequestedPriority**: contrariamente à *LeastRequestedPriority*, os nodos com as maiores solicitações de recursos recebem as melhores pontuações;
- **BalancedResourceAllocation**: visa equilibrar o consumo de recursos nos nodos.

O *kube-scheduler* é configurável tanto em termos de quais predicados e prioridades serão utilizados, quanto na relevância de cada prioridade ao somar as pontuações para um determinado nodo. Sendo assim, a criação de novas prioridades e configurações

no cálculo do *score* são formas relativamente simples de alterar o algoritmo de escalonamento para atingir algum objetivo específico.

2.3 Considerações Finais

Neste capítulo apresentou-se uma visão geral do paradigma de Computação na Borda, expondo suas características, vantagens e desvantagens, destacando as diferenças em relação à Computação na Nuvem. Estima-se que as técnicas de aumento da eficiência energética introduzidas neste trabalho, embora não tenham sido desenvolvidas especificamente para um determinado modelo de implementação de Computação na Borda, sejam especialmente eficazes em sistemas de *Fog Computing*, visto que os algoritmos de escalonamento propostos consideram a heterogeneidade dos *edge devices*.

Sistemas MEC não necessariamente possuem restrições quanto ao uso de energia, enquanto *cloudlets* possuem algumas características semelhantes a *data centers*, as quais podem facilitar o uso de estratégias de redução do consumo de energia comuns a sistemas de Computação na Nuvem.

Por fim, descreveu-se o ecossistema de gerenciamento e provisionamento de aplicações em contêineres através das plataformas Docker e Kubernetes. As contribuições deste trabalho, embora não sejam restritas a ele, utilizam o *kube-scheduler* como base para o modelo de escalonamento.

3. TRABALHOS RELACIONADOS

Neste capítulo encontra-se a revisão e análise da literatura quanto às técnicas de redução do consumo de energia no contexto do posicionamento de aplicações nos paradigmas de computação tanto na na nuvem quanto na borda. A Seção 3.1 apresenta uma classificação das técnicas utilizadas em *data centers*, bem como descreve algumas delas. Já a Seção 3.2 discorre sobre a aplicabilidade na borda das técnicas voltadas à Computação na Nuvem, assim como descreve o estado da arte nas técnicas de aumento da eficiência energética específicas a sistemas de Computação na Borda.

3.1 Eficiência Energética em Computação na Nuvem

As técnicas de redução do consumo de energia da Computação na Nuvem, isto é, em *data centers*, são um assunto bem explorado na literatura [60, 61, 62, 63, 9, 10, 11, 12]. No estudo desenvolvido por Elnozahy et al [60], o gerenciamento de energia em *clusters* de servidores foi organizado em 5 categorias:

- *Independent Voltage Scaling* (IVS)
- *Coordinated Voltage Scaling* (CVS)
- *Vary-On Vary-Off* (VOVO)
- *Combined Policy* (VOVO-IVS)
- *Coordinated Policy* (VOVO-CVS)

A IVS agrupa as técnicas que envolvem cada máquina física individualmente, como o *Dynamic Voltage and Frequency Scaling* (DVFS), em que a frequência e a tensão de operação dos processadores variam de acordo com a demanda de processamento. Nessa categoria, todos os nodos do *cluster* ficam ativos o tempo todo, mesmo em períodos de baixa carga computacional. Visto que o gerenciamento de energia em cada nodo é independente dos demais, cada dispositivo pode operar em frequências e tensões diferentes.

Em CVS, a redução do consumo de energia também se dá através de DVFS. No entanto, diferentemente da IVS, os nodos do *cluster* coordenam-se para operar praticamente na mesma tensão e frequência. Um monitor centralizado observa a frequência e tensão média dos nodos e as propaga para todos os servidores do *cluster*. A seguir, cada nodo restringe suas faixas de frequência e tensão para operar próximo à média do *cluster*. Assim como na IVS, todos os dispositivos do *cluster* permanecem ligados o tempo todo.

As técnicas da categoria VOVO envolvem o desligamento de nodos, de modo que fique ativa a menor quantidade possível de dispositivos capaz de suportar a carga de trabalho demandada. A VOVO não utiliza DVFS, no entanto, ela exige do sistema alguma funcionalidade que dê suporte a dinamicamente ativar nodos desligados, como interface de rede *Wake-On-LAN*, assim como um mecanismo de controle que envie comandos de desligamento aos dispositivos subutilizados.

A categoria VOVO-IVS simplesmente abrange sistemas que apliquem técnicas VOVO e IVS em conjunto. Sendo assim, os nodos subutilizados são desligados, enquanto aqueles que não estiverem com utilização máxima têm suas frequências e tensões reduzidas individualmente de acordo com a carga de trabalho.

Analogamente, a VOVO-CVS engloba sistemas que utilizam ambas VOVO e CVS. Ao utilizá-las em conjunto, forma-se uma estratégia que define uma faixa de operação de frequência e tensão a qual varia de acordo com a quantidade de máquinas ativas, de modo que um nodo é dinamicamente desligado se sua frequência ou tensão caírem abaixo do limite inferior. De maneira semelhante, quando todos os nodos estiverem operando acima do limite superior, uma máquina é reativada para suprir a demanda computacional.

Muitos dos trabalhos que tratam da consolidação de VMs ou de aplicações [61, 62, 63, 9, 10, 11, 12] introduzem técnicas que envolvem VOVO. Ao concentrar a computação em apenas alguns nodos do sistema, conclui-se que é possível desligar as máquinas inutilizadas para aumentar a eficiência energética.

O trabalho desenvolvido por Rajamani et al [61] é um dos primeiros exemplos de implementação de uma técnica VOVO. O algoritmo responsável pelas decisões de ligar ou desligar as máquinas de um *data center* analisa fatores-chave tanto dos servidores, como capacidade de processamento e tempos de inicialização e de desligamento, quanto das tarefas, como carga de trabalho e tempo estimado de execução.

Em [63], Mathew et al propõe técnicas de otimização do consumo de energia com o foco em desligar servidores de *Content-Delivery Networks* (CDNs) durante períodos de baixa demanda. As soluções desenvolvidas, através de diretrizes de escalonamento, têm como objetivos maximizar a redução do consumo de energia, minimizar o impacto de disponibilidade percebida pelos clientes, e limitar a frequência de transições de inicialização e desligamento dos servidores para reduzir o desgaste do *hardware*.

A diferenciação do trabalho de Mathew et al [63] em relação aos que o antecedem é: o foco em CDNs, as quais apresentam características comuns em sistemas de Computação na Borda, como geodistribuição e baixo tempo de resposta; otimização do *tradeoff* entre eficiência energética e violações de SLAs; e análise do impacto de inicializações e encerramentos sucessivos, o que pode reduzir a vida útil dos servidores.

Tabela 3.1: Trabalhos sobre redução do consumo de energia em Computação na Nuvem

Artigo	Categoria	Métrica de desempenho	Estratégia
[9]	VOVO-IVS	Tempo de resposta	Consolidação de VMs
[10]	VOVO*-IVS	Transações por segundo	Consolidação de VMs e DVFS
[11]	VOVO	Utilização de CPU	Consolidação de VMs
[12]	VOVO-IVS	Tempo de execução	Posicionamento de VMs e DVFS

Os estudos desenvolvidos por Rajamani et al [61] e Mathew et al[63], embora relevantes, foram publicados há mais de uma década. Desde então, diversas novas técnicas de redução do consumo de energia em Computação na Nuvem foram introduzidas. A Tabela 3.1 apresenta trabalhos mais recentes que tratam do posicionamento energeticamente eficiente de VMs ou aplicações.

O principal objetivo do trabalho desenvolvido por Sharma et al [9] é formular a questão de alocação de VMs como sendo um problema multidimensional de otimização combinatória com múltiplos objetivos. Desse modo, os autores propõem um algoritmo capaz de alocar múltiplas instâncias de VMs na menor quantidade de máquinas físicas de forma que seja possível reduzir o consumo de energia e otimizar a utilização de recursos do *data center*.

Sendo assim, o trabalho de Sharma et al [9] possui duas contribuições. Primeiramente, introduz-se um algoritmo híbrido baseado em algoritmos genéticos, *Particle Swarm Optimization* (PSO) e distância euclidiana para atingir o ponto ótimo entre eficiência energética e utilização de recursos. Por fim, apresenta-se uma diretriz de migração

de VMs como foco na redução tanto do consumo de energia quanto das violações de SLAs no *data center*.

Rossi et al [10] apresenta o e-eco, um orquestrador energeticamente eficiente para a Computação na Nuvem, o qual tenta equilibrar o *tradeoff* entre economia de energia e desempenho das aplicações através do gerenciamento inteligente de um conjunto de técnicas de redução do consumo de energia. Esse orquestrador permite uma resposta imediata do sistema quanto a quais técnicas devem ser utilizadas em um dado instante de forma que não prejudiquem o desempenho das aplicações. Embora estas técnicas não envolvam o desligamento dos nodos, do ponto de vista do sistema, os nodos colocados em *sleep states* comportam-se de modo semelhante a máquinas desligadas, necessitando de um sinal externo para que sejam reativadas, porém com menor latência de inicialização.

As principais contribuições do trabalho apresentado por Rossi et al [10] são: elaboração de um modelo aprimorado dos estados de baixo consumo de energia da *Advanced Configuration and Power Interface* (ACPI); a análise detalhada dos impactos de consolidação de servidores e técnicas de DVFS no consumo de energia e no desempenho de aplicações; e a implementação de um orquestrador multiobjetivo com foco em eficiência energética e alto desempenho.

Em [11], Yadav et al introduz 2 algoritmos baseados em regressões lineares, o Gdr e o MCP, a fim de definir dinamicamente um limiar de utilização de CPU de modo a detectar nodos sobrecarregados. A partir destes nodos, faz-se a migração de VMs para outras máquinas para minimizar a degradação do desempenho do sistema.

Ao concentrar as aplicações em poucos nodos, desde que a utilização da CPU de cada um deles seja menor que o limiar definido pelos algoritmos, é possível otimizar a utilização de recursos e minimizar o número de nodos ativos, reduzindo, dessa forma, o consumo de energia. Quando a carga computacional ultrapassa o limite superior de utilização da CPU, a migração de aplicações para máquinas subutilizadas pode gerar uma ineficiência quanto ao consumo de energia, no entanto, reduz-se a quantidade de violações de SLA ao aumentar o desempenho do sistema.

Além disso, Yadav et al [11] propõe um algoritmo denominado *Bandwidth-Aware Dynamic VM Selection Policy* (Bw) com o intuito de selecionar as VMs que serão migradas dos nodos sobrecarregados. Esse algoritmo equilibra o *tradeoff* entre consume de energia, número de migrações, desempenho dos nodos e quantidade de nodos desligados. Por

fim, o Gdr, MCP e Bw utilizam como base dados históricos de utilização da CPU dos nodos no sistema.

Em [12], Chen et al introduz um mecanismo que combina um algoritmo de consolidação de VMs com um método de DVFS, de modo a reduzir o consumo de energia do sistema. O *energy delay product* (EDP), o qual é definido como o produto do tempo de execução de uma tarefa com o consumo de energia atribuído a ela, foi utilizado como critério de avaliação. Segundo os autores, o EDP expõe informações tanto do desempenho do sistema quanto do consumo de energia. Sistemas com um menor valor de EDP ou consomem menos energia enquanto mantêm o mesmo tempo de execução, ou consomem a mesma quantidade de energia, porém em menos tempo quando comparados a sistemas com maior EDP.

Embora muitos dos trabalhos descritos nesta seção tenham como objetivo não só reduzir o consumo de energia, mas também maximizar o desempenho do sistema, as aplicações de Computação na Nuvem tipicamente possuem propriedades diferentes do que as de Computação na Borda. Em *cloudlets*, por exemplo, geralmente são utilizados *edge devices* com menor poder computacional do que os servidores de *data centers* [7]. Além disso, aplicações da borda usualmente requerem uma latência menor quando comparadas às da nuvem [7].

Nodos distribuídos geograficamente, máquinas com limitações quanto à capacidade de processamento e requisitos como baixo tempo de provisionamento demandam técnicas eficientemente energética específicas à Computação na Borda. A próxima seção apresenta algumas destas técnicas.

3.2 Eficiência Energética em Computação na Borda

Conforme apresentado por Mathew et al [63], maximizar a eficiência energética e satisfazer os SLAs são objetivos conflitantes, visto que desligar nodos visando a redução do consumo de energia diminui a capacidade computacional do sistema. Na Computação na Borda, essa dicotomia é acentuada: o tempo de inicialização dos *edge devices* é maior do que em servidores de *data centers*, além de que os requisitos de latência, em geral, são mais rigorosos em aplicações da borda.

Para evitar violações de SLAs devido ao tempo de inicialização de um *edge device*, podem-se utilizar técnicas IVS ou CVS, as quais não envolvem o desligamento do nodo. No entanto, alguns SBCs, como a Raspberry Pi 4 [64], não possuem modos *sleep*/*standby* [65], enquanto outros dispositivos como a Orange Pi Zero [66] tornam-se inoperantes ao utilizar DVFS com frequências muito reduzidas [67].

Trabalhos sobre o posicionamento de aplicações na borda visando a redução do consumo de energia raramente aplicam técnicas do tipo VOVO, VOVO-IVS ou VOVO-CVS, visto que o tempo de inicialização da máquinas pode causar um alto impacto negativo no tempo de provisionamento. Alguns deles, listados na Tabela 3.2, aumentam a eficiência energética de partes do sistema através de *task offloading*, isto é, através da migração da computação seja dos dispositivos IoT para a borda, seja dos edge devices para a nuvem.

Tabela 3.2: Trabalhos relacionados à *task offloading* na borda

Artigo	Sentido da Migração	Métrica de desempenho	Estratégia
[13]	IoT $\rightarrow$ *Edge* $\rightarrow$ *Cloud*	Latência	Migração de dados e aplicações de *IOT devices* para a borda ou nuvem
[14]	IoT $\rightarrow$ *Edge* $\leftrightarrow$ *Cloud*	Tempo de execução	Otimização entre execução de aplicações na borda ou na nuvem
[15]	IoT $\rightarrow$ *Edge*	Definições de QoS	Particionamento da tarefa entre IoT e *edge devices*
[16]	IoT $\rightarrow$ *Edge*	Tempo de provisionamento	Migração de tarefas de dispositivos IoT para a borda

O objetivo da investigação realizada por Jabour et al [13] foi cumprir os requisitos de Qualidade de Serviço (QoS, do inglês, Quality of Service) das aplicações IoT sensíveis a atrasos através da migração de tarefas dos dispositivos IoT para a borda. No entanto, também foi considerado o *tradeoff* entre baixa latência e consumo de energia. Sendo assim, de forma a alocar recursos eficientemente nos *edge devices*, foi proposta uma técnica de escalonamento baseada no *Particle Swarm Optimization Algorithm*.

A migração de tarefas em [13] ocorre da camada de IoT para a borda, ocasionando uma redução do consumo de energia nos *IoT devices* a detrimento da eficiência energética nos nodos da borda. No entanto, este estudo e os demais mencionados na Tabela 3.2 mantêm-se relevantes devido à aplicabilidade das técnicas propostas no *offloading* de aplicações da borda para a nuvem.

Em [14], Sharifi et al. analisaram o consumo de energia em *edge devices* de acordo com o modelo de execução das aplicações. Para fins de comparação, foram definidos 3 modos de operação quanto ao processamento das tarefas: diretamente na nuvem, exclusivamente na borda, e dinamicamente através de *offloading* Borda-Nuvem. Sendo assim, introduziu-se uma diretriz simples de migração de tarefas baseada no tempo de

espera para execução de uma tarefa. Este tempo de espera é definido como o intervalo em que cada solicitação aguarda na fila de processamento dos nodos.

O artigo de Ergun et al. [15] apresentam uma técnica de gerenciamento dinâmico de confiabilidade (DRM, do inglês *Dynamic Relability Management*) voltada aos dispositivos IoT e da borda, de modo a utilizar a migração de tarefas da camada IoT para a borda. O objetivo desta estratégia é manter a QoS e o cumprimento dos requisitos de confiabilidade enquanto maximiza-se a eficiência energética dos dispositivos do sistema.

A técnica de DRM implementada em [15] consiste em formular um problema de controle ótimo não-linear de horizonte finito com objetivos de maximizar a duração da bateria, manter a QoS e cumprir requisitos de confiabilidade dos terminais. Diferentemente das técnicas de DRM aplicadas em dispositivos individualmente, esta solução utiliza estratégias tanto individuais, como DVFS, quanto coletivas a nível de rede, através de *task offloading*.

Em [16], Hu et al. investigaram o *tradeoff* entre eficiência energética e atraso de serviço em sistemas IoT-MEC durante o provisionamento de serviços em um cenário onde os usuários deslocam-se geograficamente. A solução proposta envolve a formulação de um problema de otimização estocástico com o objetivo de maximizar a eficiência energética de rede sem comprometer os requisitos de tempo de resposta e os limites superiores potência de transmissão, frequência de CPU e número de usuários. Para isso, foi desenvolvido um algoritmo online de migração de tarefas e alocação de recursos (OORAA) que envolve a decomposição do problema original em diversos subproblemas baseados na teoria de otimização de Lyapunov, os quais são resolvidos através de decomposição convexa e métodos submodulares.

O *offloading* de tarefas, entretanto, não é necessariamente viável em todos os casos de uso de Computação na Borda. Algumas aplicações possuem restrições quanto à confidencialidade dos dados e à proximidade de execução. Além disso, ao migrar tarefas da borda para a nuvem, é possível que o desempenho do sistema seja impactado negativamente devido à latência entre os dispositivos IoT e a nuvem.

Sendo assim, na literatura, encontram-se outras estratégias de aumento da eficiência energética em sistemas de Computação na Borda que não envolvam a migração Borda-Nuvem de aplicações. Grande parte das técnicas estudadas utilizam estratégias que envolvem o posicionamento das aplicações nos dispositivos da borda como forma de

minimizar o consumo de energia, sem exigir necessariamente que *edge devices* sejam desligados ou que devam ser colocados em *sleep states*. Alguns destes trabalhos estão descritos na Tabela 3.3.

Tabela 3.3: Trabalhos relacionados ao posicionamento eficientemente energético na borda

Artigo	Modelo de *Edge Computing*	VOVO	Métrica de desempenho	Estratégia
[17]	*Fog Computing*	✗	Utilização de Recursos	Consolidação de aplicações
[18]	*Mobile Edge Computing*	✗	Consumo de Energia	Migração de VMs para nodos energeticamente eficientes
[19]	*Fog Computing*	✗	Tempo de Execução	Posicionamento de aplicações
[20]	*Fog Computing*	✗	Definições de QoS	Consolidação de VMs
[21]	*Fog Computing*	✗	Tempo de Execução	Posicionamento de aplicações
[68]	*Mobile Edge Computing*	✓	Tempo de provisionamento	Migração de tarefas e desligamento de nodos
[69]	*Cloudlet*	✓	*Round-Trip Time*(RTT)	Desligamento de nodos subutilizados

Em [17], Mahaptra et al. propõem um método energeticamente eficiente para que dados oriundos de dispositivos IoT sejam processados na borda. A estratégia utiliza uma fila multinível com *feedback* para a classificação do nodo que receberá a aplicação em conjunto com uma técnica baseada em *Fuzzy C-means* para realizar o *clustering* dos nodos disponíveis para processamento paralelo. A técnica desenvolvida, todavia, não altera o estado de energia dos nodos não utilizados.

O trabalho de Gu et al [18] investiga o problema da minimização do consumo de energia através da migração de VMs e alocação de tarefas com o objetivo de reduzir a quantidade de energia obtida através de fontes não renováveis. A partir de um modelo baseado na otimização de escalonamento com faixas de tempo discretas, os autores propõem um algoritmo de escalonamento capaz de gerenciar a variabilidade tanto de fontes de energia renováveis quanto da demanda dos usuários. Contudo, este trabalho não foca no aumento da eficiência energética do sistema como um todo, tendo como objetivo principal a redução do consumo de energia oriundo de fontes de energia não renováveis.

Na solução proposta por Vadde et al. [19], uma nova heurística, denominada *Levy flight*, é introduzida ao algoritmo JAYA com o fim de produzir uma distribuição probabilística do tipo passeio aleatório. Esta estratégia, chamada de LJAYA, é, então, utilizada para garantir o equilíbrio do *tradeoff* entre desempenho e consumo de energia em sistemas de Computação na Borda através do posicionamento eficiente de aplicações nos *edge devices*.

Alnoman et al. propõem em [68] um mecanismo de suspensão assistido por *Software Defined Networks* (SDNs) voltado à minimização do consumo de energia em sistemas de Computação na Borda. A técnica garante o cumprimento das restrições de probabilidade de enfileiramento de tarefas de modo a reduzir o impacto percebido pelos usuários

devido ao atraso do processamento. Sendo assim, a estratégia utiliza a *square-root staffing rule* e a função Halfin-Whitt para determinar a quantidade de *edge devices* que podem ser suspensos para uma determinada probabilidade de enfileiramento.

Uma vez que as tarefas de um nodo que entrará em suspensão deverão ser migradas, os autores em [68] implementam também um mecanismo de balanceamento de carga de forma a reduzir a variação de carga de processamento entre os *edge devices* ativos. Logo, a estratégia proposta utiliza um algoritmo baseado numa cadeia de Markov de tempo discreto.

Em [20], Alnoman et al. propõem um esquema de redução do consumo de energia através da suspensão de VMs. A estratégia utilizada para a decisão de suspensão e consolidação de VMs em *edge devices* é a mesma empregada em [68], com a diferenciação de que, no trabalho desenvolvido em [20], os *edge devices* não são desligados. Sendo assim, é possível reduzir o tempo de resposta para a inicialização de novas tarefas mantendo alta eficiência energética.

Ghanavati et al. [21] desenvolvem um algoritmo baseado em *Ant Mating Optimization* (AMO) para otimizar em conjunto os problemas de seleção de *edge device* e atribuição de tarefas a esses dispositivos. A estratégia implementada emprega uma função de custo que possui como variáveis o tempo de execução e o consumo de energia, obtendo, desse modo, um equilíbrio entre o *tradeoff* de desempenho e eficiência energética do sistema.

Por fim, Rausch et al. [69] propõem uma arquitetura de infraestrutura para sistemas de Computação na Borda baseados em *clusters* com alta eficiência energética. No núcleo da arquitetura apresentada, encontra-se um mecanismo de monitoramento de energia e de utilização de recursos nos nodos do sistema. Este mecanismo também é responsável pelo balanceamento de carga e pela consolidação de aplicações nos *edge devices*. Uma diretriz implementada para redução do consumo de energia é a *Reactive Autoscaling*, que consiste na ativação ou suspensão de um nodo se uma métrica do sistema ultrapassa um limiar pré-definido por um determinado intervalo de tempo. Na configuração padrão da arquitetura apresentada, um nodo adicional é ativado caso a média da utilização de CPU em todos os nodos do *cluster* mantenha-se acima de 85% por mais de 10 segundos. Já a suspensão de um nodo ocorre caso a utilização de CPU fique abaixo de 25%.

Dos trabalhos estudados que tratam do posicionamento de aplicações energeticamente eficiente em Computação na Borda, vários [19, 20, 68, 21, 17] utilizam um modelo em que o consumo de energia possui uma relação linear com a utilização da CPU. Este modelo é relevante em servidores convencionais, como os utilizados em *data centers* [62, 70], no entanto, ele não representa adequadamente o comportamento do consumo de energia de dispositivos de Computação na Borda. Em [71], Rausch et al demonstra que o consumo de energia em uma Raspberry Pi 4 possui uma relação logarítmica com a utilização da CPU.

3.3 Considerações Finais

Este capítulo abordou a revisão da literatura quanto às técnicas de eficiência energética voltadas a sistemas de Computação na Nuvem e na Borda. Dos trabalhos mencionados no capítulo, podem-se realizar as seguintes observações:

- Técnicas voltadas a *data centers* e à Computação na Nuvem tendem a utilizar estratégias VOVO ou alguma de suas variações, como VOVO-IVS ou VOVO-CVS;
- Técnicas com foco em Computação na Borda tendem a não desligar os nodos subutilizados, empregando *application placement* ou *task offloading* como alternativas;
- A maioria dos trabalhos relacionados, inclusive o desenvolvido em [68], o qual utiliza uma estratégia VOVO, consideram que o consumo de energia de *edge devices* é proporcional à utilização da CPU.

Embora empregue uma estratégia VOVO, o algoritmo de escalonamento utilizado em [69] é um simples *round-robin*, em que as aplicações são provisionadas de acordo com uma lista estática de nodos. Sabe-se que, devido à heterogeneidade de *edge devices* em sistemas de Computação na Borda, bem como a possibilidade de existirem restrições quanto à localidade das aplicações, este tipo de escalonamento simples impacta negativamente o desempenho do sistema [6].

Sendo assim, este trabalho pretende introduzir novas técnicas para o aumento da eficiência energética em sistemas de Computação na Borda que envolvam o desligamento de nodos subutilizados, a fim de maximizar a redução do consumo de energia. Não

obstante, as soluções propostas minimizam as violações de SLA de tempo de provisionamento, procurando um equilíbrio do *tradeoff* entre eficiência energética e desempenho do sistema.

4. CONTRIBUIÇÕES

Este trabalho apresenta como contribuição duas técnicas para aumento da eficiência energética no posicionamento de aplicações de Computação na Borda ao mesmo tempo em que visa minimizar as violações de SLA de tempo de provisionamento. Diferentemente da maioria dos trabalhos analisados no Capítulo 3 específicos à Computação na Borda, as técnicas descritas neste capítulo resgatam o mecanismo VOVO presente nas estratégias voltadas à Computação na Nuvem.

Este Capítulo está organizado da seguinte forma: a Seção 4.1 apresenta o modelo do consumo de energia utilizado durante o desenvolvimento deste trabalho; a Seção 4.2 descreve o mecanismo de gerenciamento de energia com características VOVO; as Seções 4.3 e 4.4 esclarecem, respectivamente, as técnicas AAS e EA-DLSLA; por fim, a Seção 4.5 traz as considerações finais.

4.1 Modelo do Consumo de Energia

Computadores em geral, tanto os *edge devices*, quanto aqueles usados em *data centers*, consomem menos energia quando a utilização da CPU não está em seu limite superior [70, 71, 72, 73]. No entanto, a relação entre utilização de CPU e consumo de energia não é a mesma para servidores convencionais e dispositivos como SBCs [71].

Aliás, no contexto de eficiência energética em *data centers*, não há um consenso sobre a porcentagem da potência máxima de um servidor que é produzida quando ele está em inatividade. Em [70], Judge et al postula que o consumo de energia de uma máquina inativa é 70% do consumo sob utilização máxima.

Porém, em [72], Vasques et al estima a partir do *SPEC power benchmark*[74] que o consumo de máquinas inativas situa-se entre 15% a 34% do consumo máximo. Ainda, em [73], Fuchs et al afirma que os dados do *SPEC power benchmark* podem ser enviesados, isto é, contêm em sua maioria apenas servidores cuja eficiência energética é maior do que os encontrados no mercado como um todo. Aliás, a Figura 4.1 apresenta comparações da relação da utilização da CPU com a mediana da potência de servidores entre os *datasets* da SPEC, da ENERGY STAR [75], e com os dados obtidos a partir de *web scraping* de *websites* de varejo.

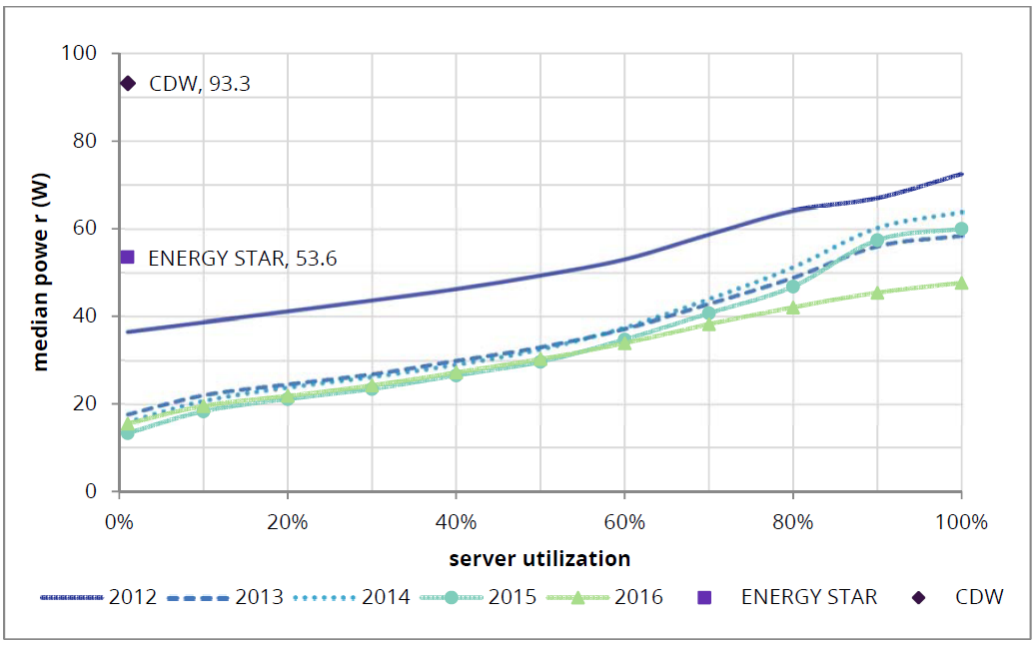

Figura 4.1: Relação entre a utilização da CPU e mediana da potência produzida em servidores

Fonte: Fuchs et al (2022) [73]

Em [76], Shehabi et al determinou que servidores convencionais com apenas um soquete de CPU produzem 118W em média com utilização máxima. Sendo assim, a potência produzida em inatividade representa 79%, 45% e 15% da potência máxima de acordo com os dados obtidos por *web scraping*, da ENERGY STAR e da SPEC, respectivamente.

Em relação aos *edge devices*, em [71], Rausch et al observou que a relação entre utilização de CPU e consumo de energia não é linear, demonstrando um comportamento logarítmico. A Figura 4.2 ilustra esta relação com base em dados coletados por prototipagem.

Sendo assim, considerando uma topologia de *Fog Computing*, em que existam máquinas com desempenho similar tanto a servidores convencionais, as quais serão denominadas de *workers*, quanto a SBCs, chamadas de *edge nodes*, define-se: P_{min} como a potência produzida pelos nodos quando estiverem inativos, isto é, sem executarem nenhuma aplicação; e P_{max} como a potência máxima produzida pelas máquinas. A Figura 4.3 exemplifica esta topologia heterogênea de Computação na Nuvem.

Além disso, w_a, s_a e e_a representam, respectivamente, a carga de trabalho em mCPUs, o tempo de início e o tempo de término da execução de uma aplicação a. Sob este modelo, assim que as aplicações forem provisionadas, elas executam uma tarefa por um determinado período de tempo. Já o controle do gerenciamento de energia nos

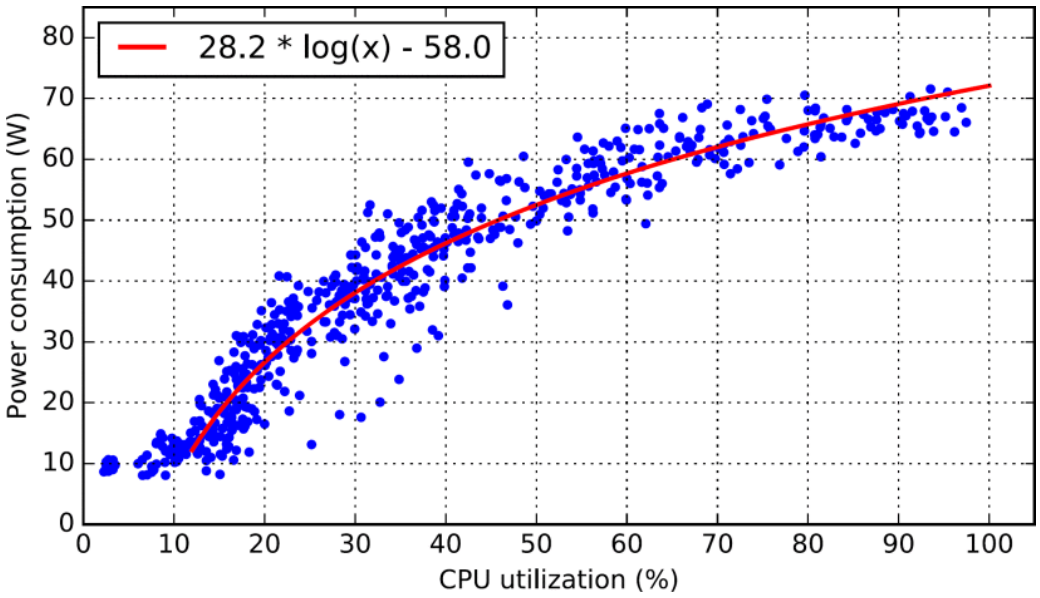

Figura 4.2: Relação entre utilização de CPU e potência produzida em *edge devices*

Fonte: Rausch et al (2018) [71]

nodos pode ser modelado através de mensagens de inicialização (*wakeup*) e desligamento (*shutdown*), definidas respectivamente como M_w e M_s. Estas mensagens, assim como as aplicações, contêm informações de tempo de início (M^s) e tempo de término (M^e).

Desse modo, para um nodo n, definem-se respectivamente os intervalos de tempo de *wakeup*, de *shutdown* e no estado *awake*, isto é, já inicializado, através das Equações 4.1, 4.2 e 4.3, onde u representa a quantidade de mensagens em um determinado nodo:

$$t_w^n = \sum_{i=1}^{u} M_{w_i}^e - M_{w_i}^s \tag{4.1}$$

$$t_s^n = \sum_{i=1}^{u} M_{s_i}^e - M_{s_i}^e \tag{4.2}$$

$$t_\alpha^n = \sum_{i=1}^{u} M_{s_i}^s - M_{w_i}^e \tag{4.3}$$

O intervalo de tempo ativo, t_{act}^n, é definido para *worker nodes* como o período em que existe uma aplicação sendo executada na máquina. Todavia, simplificando o modelo de consumo introduzido por Rausch et al [71], neste trabalho, um *edge node* só é considerado ativo se possuir utilização de CPU acima de 30%. Já o intervalo de tempo em que

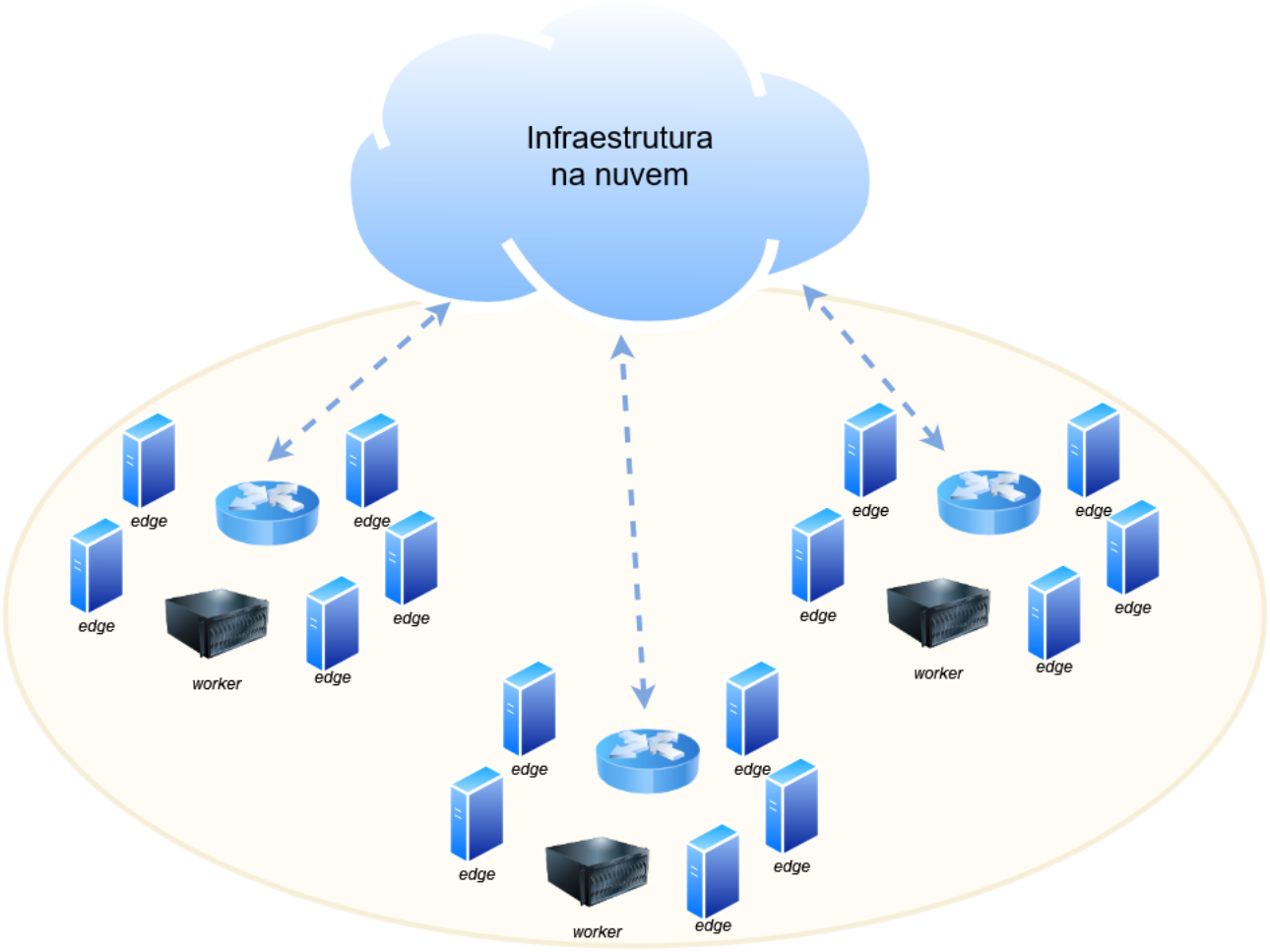

Figura 4.3: Diagrama da topologia de Computação na Borda

Fonte: O autor (2022)

um nodo está inativo é obtido por meio da equação abaixo:

$$t_i^n = t\alpha^n - t_{act}^n \tag{4.4}$$

Uma vez obtidos esses valores, é possível realizar os cálculos de consumo de energia. Primeiramente, o consumo de energia durante a inicialização da máquina é calculado como:

$$C_w^n = \frac{t_w^n}{3600} * 1000 P_{max}^n \tag{4.5}$$

Dado que t_w^n encontra-se em segundos e P_{max}^n em watts, para obter C_w^n em mWh é necessário realizar conversões de unidade. Sendo assim, a Equação 4.5 pode ser simplifi-

cada da seguinte forma:

$$C_w^n = \frac{t_w^n * P_{max}^n}{3,6} \tag{4.6}$$

Analogamente, obtêm-se os valores de consumo de energia durante o desligamento da máquina e quando ela estiver inativa, respectivamente, a partir das Equações 4.7 e 4.8:

$$C_s^n = \frac{t_s^n * P_{max}^n}{3,6} \tag{4.7}$$

$$C_i^n = \frac{t_i^n * P_{min}^n}{3,6} \tag{4.8}$$

Em *worker nodes*, os quais possuem características de servidores convencionais, considera-se que o consumo de energia possui uma relação linear com a utilização da CPU. Sendo assim, é possível calcular o valor do consumo de energia quando o nodo estiver ativo através da Equação 4.9, onde v representa a quantidade de aplicações executando no nodo n e $\mathcal{W}^n$ é a sua capacidade de processamento:

$$C_{act_w}^n = \frac{P_{max}^n}{3.6} * \frac{1}{\mathcal{W}^n} \sum_{a=1}^{v} w_a * (e_a - s_a) \tag{4.9}$$

$$\mathrm{c}_{act_e}^n(t) = \begin{cases} P_{max}^n, & \text{se } \frac{1}{\mathcal{W}^n} \sum_{a=1}^{v} w_a(t) \geq 0.30 \\ P_{min}^n, & \end{cases} \tag{4.10}$$

$$C_{act_e}^n = \frac{1}{3,6} \int \mathrm{c}_{act_e}^n(t) \tag{4.11}$$

Baseando-se nos dados da Figura 4.2, como em *edge nodes* a potência produzida é máxima quando a utilização da CPU for superior a 30% e mínima abaixo disso, o cálculo do consumo de energia em um dado instante é obtida pela Equação 4.10. Sendo assim, o consumo de energia ativa de um *edge node* calcula-se conforme a Equação 4.11. Por fim, o consumo de energia total de um nodo se dá através da seguinte equação:

$$C^n = C_w^n + C_s^n + C_i^n + C_{act}^n \tag{4.12}$$

Concluindo, o desligamento das máquinas inativas reduz drasticamente os valores C_i^n. No entanto, dependendo do número de inicializações e desligamentos, ao longo

tempo de inicialização das máquinas podem ocasionar violações de SLA, além de causar aumentos de C_w^n e C_s^n.

4.2 Mecanismo de Gerenciamento de Energia

Por padrão, o Kubernetes não provisiona aplicações para nodos que estão desligados, visto que, para o kube-controller-manager, não há diferenciação entre uma máquina desligada ou que possua uma falha na comunicação com a rede. Entretanto, uma vez que o Kubernetes é extensível através de *plugins*, ao longo do desenvolvimento deste trabalho, supõe-se a existência de uma extensão que permita o escalonamento de aplicações a nodos desligados.

Esta extensão, também considerada como o mecanismo de gerenciamento de energia, é responsável por solicitar a inicialização dos nodos de um *cluster*. As técnicas introduzidas neste trabalho não solicitam diretamente a este mecanismo, nem às máquinas físicas, as operações de inicialização ou desligamento. Em vez disso, pressupõe-se que o gerenciador de energia monitora a lista de aplicações provisionadas para determinar se deverá ativar um nodo, além de desligar as máquinas que não estiverem executando nenhuma aplicação.

O mecanismo de inicialização e encerramento das máquinas é independente das técnicas introduzidas neste trabalho. Isto é, o escalonador não envia diretamente ao nodo uma solicitação de inicialização ou encerramento.

Caso o nodo esteja desligado e uma aplicação tenha sido atribuída a ele, então o gerenciador de energia solicita a inicialização da máquina. Analogamente, quando um nodo ligado não tiver nenhuma tarefa pendente, então o gerenciador de energia o desliga. O Algoritmo 4.1 descreve o funcionamento do mecanismo de gerenciamento de energia.

4.3 *Availability-Aware Scheduler*

Em [77], Fu et al propõe que escalonadores devam considerar as dependências das aplicações ao realizar as decisões de posicionamento. Especificamente, sugere-se que um escalonador deva atentar-se em provisionar uma aplicação $\mathcal{A}$ no nodo que tiver

Algoritmo 4.1: Mecanismo de Gerenciamento de Energia

Entrada: *nodos*: lista de nodos do sistema; *aplicacoesPendentes*: lista de aplicações a serem provisionadas;

```
para cada n ∈ nodos faça
    se estado(n) = desligado então
        para cada a ∈ aplicacoesPendentes faça
            se nodo(a) = n então
                ligarNodo()
                break
            fim
        fim
    fim
    senão se tamanho(aplicacoes(n)) = 0 então
        desligarNodo()
    fim
fim
```

a maior quantidade das dependências de $\mathcal{A}$, desde que ele possua recursos livres para receber $\mathcal{A}$. Deste modo, ao utilizar esta estratégia, chamada de *dependency scheduling*, estima-se que seja possível reduzir o tempo de provisionamento das aplicações.

Ainda segundo Fu et al [77], o *makespan* $\mathcal{M}$ de uma tarefa, isto é, o tempo desde a solicitação inicial até a conclusão de sua execução, pode ser decomposto conforme a Equação 4.13, onde $\mathcal{S}$ representa o tempo necessário para a inicialização da tarefa e $\mathcal{R}$ é seu tempo de execução após ter sido inicializada. Além disso, supõe-se que $\mathcal{S}$ é formado majoritariamente pelo tempo de *download* dos contêineres da aplicação.

$$\mathcal{M} = \mathcal{S} + \mathcal{R} \tag{4.13}$$

O *dependency scheduling* visa reduzir $\mathcal{S}$ através do *caching* de imagens previamente provisionadas em um determinado nodo. Desse modo, escalonam-se aplicações para nodos que já possuírem a imagem do contêiner da aplicação, estratégia esta chamada de *ImageLocalityPriority*. Além disso, em maior nível de granularidade, Fu et al [77] propõe também o *caching* das camadas de contêineres, em uma técnica denominada *LayerLocalityPriority*. A pontuação de cada nodo é calculada a partir de quantas dependências da aplicação a ser provisionada encontram-se na *cache* do nodo.

Embora as camadas de contêineres possam conter configurações e implementações específicas a determinadas aplicações, muitas destas camadas são genéricas, sendo

reutilizadas em diversas imagens [49, 6, 5]. Ao provisionar uma aplicação, o nodo não realiza o *download* da imagem em um único pacote, em vez disso, são solicitadas apenas as camadas que não se encontram na *cache* do dispositivo. Desse modo, através da *LayerLocalityPriority*, é possível reduzir o tráfego desnecessário na rede e minimizar o tempo de provisionamento das aplicações.

No entanto, ao considerar o mecanismo VOVO, em que máquinas subutilizadas são desligadas, o *makespan* $\mathcal{M}$ de uma tarefa é melhor representado pela Equação 4.14, onde $\mathcal{W}$ representa o tempo de inicialização do nodo onde a tarefa será executada. Desse modo, o AAS propõe uma nova prioridade que considera a disponibilidade de um nodo e o valor de $\mathcal{W}$ durante a etapa de classificação do *kube-scheduler*. O Algoritmo 4.2 descreve o funcionamento desta prioridade, denominada *Availability-Priority* (AP).

$$\mathcal{M} = \mathcal{W} + \mathcal{S} + \mathcal{R} \tag{4.14}$$

Algoritmo 4.2: Funcionamento da *Availability Priority*

Entrada: *nodo*: nodo atual; *peso*: peso da AP na pontuação geral do nodo
Saída: pontuação do nodo

score $\leftarrow 0$
se *estado*(*nodo*) $=$ *ligado* **ou** *estado*(*nodo*) $=$ *aguardandoEncerramento* **então**
    score $\leftarrow$ *peso*
fim
senão se *estado*(*nodo*) $=$ *inicializando* **então**
    score $\leftarrow$ *peso*$/2$
fim
retorna *score*

Como a AP tende a realizar a consolidação de aplicações, ou seja, concentra-as em poucos nodos, antes de sua execução, utiliza-se o predicado *PodFitsResources* na etapa de filtragem do *kube-scheduler*. Desse modo, o AP só recebe como entrada os nodos elegíveis, os quais não serão sobrecarregados com o provisionamento de uma nova aplicação.

Por fim, o *Availability-Aware Scheduler* consiste na utilização tanto da *Availability-Priority*, como da *LayerLocalityPriority*, proposta por [77] Fu et al. Sendo assim, a Equação 4.15 descreve o cálculo da pontuação de um nodo *n* sob o AAS, onde se deu maior peso à *AvailabilityPriority*, de modo a priorizar os nodos disponíveis, mesmo que possuam menos camadas em *cache* do que nodos desligados.

$$score = \frac{availabilityPriority(n, 6) + 4 * layerLocality(n)}{10} \tag{4.15}$$

4.4 *Energy-Aware Deployment Latency SLA Enforcement Scheduler*

Assim como com o AAS, o EA-DLSLA baseia-se em um algoritmo de escalonamento cujo objetivo é minimizar o tempo de provisionamento de aplicações em sistemas de computação na borda, neste caso, o DLSLA [78]. Uma vez que cada nodo do sistema pode receber múltiplas solicitações para provisionar novas aplicações, uma fila de *downloads* determina a ordem em que as camadas dos contêineres contendo as aplicações serão solicitadas para o registro de imagens.

Sendo assim, o objetivo do DLSLA consiste em definir o posicionamento de aplicações e arranjos das filas de *download* nos nodos de modo a minimizar a quantidade de violações de SLA por causa da prolongação do tempo de provisionamento. As técnicas VOVO incrementam ainda mais este atraso de provisionamento devido à inicialização das máquinas anteriormente desligadas. Logo, o *Energy-Aware DLSLA* adiciona informações sobre o tempo de inicialização e consumo de energia do nodo na função objetivo do algoritmo.

Sabe-se que o posicionamento de aplicações é um problema de otimização NP-*hard* [6]. Apesar disso, heurísticas podem, alternativamente, ser utilizadas para encontrar soluções aceitáveis em tempo polinomial [79]. Desse modo, o DLSLA baseia-se no *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) [80].

Algoritmos evolucionários usam modelos computacionais baseados nos processos de evolução biológica como ferramenta de resolução de problemas [81]. Logo, algoritmos genéticos (GA, do inglês — *Genetic Algorithms*), os quais são uma subcategoria dos evolucionários, possuem inspiração na genética e na teoria da evolução das espécies, empregando termos como *cromossomo*, *gene*, *seleção*, *crossover* e *mutação*.

Desse modo, no EA-DLSLA, um *cromossomo* ou *indivíduo*, isto é, uma possível solução do problema, representa a fila de *downloads* de um nodo, sendo constituído por uma lista de *contêineres*. Cada contêiner desta lista é denominado de *gene*, sendo que um *cromossomo* deve possuir apenas *genes* únicos. Essas representações podem ser observadas a partir do diagrama exposto na Figura 4.4.

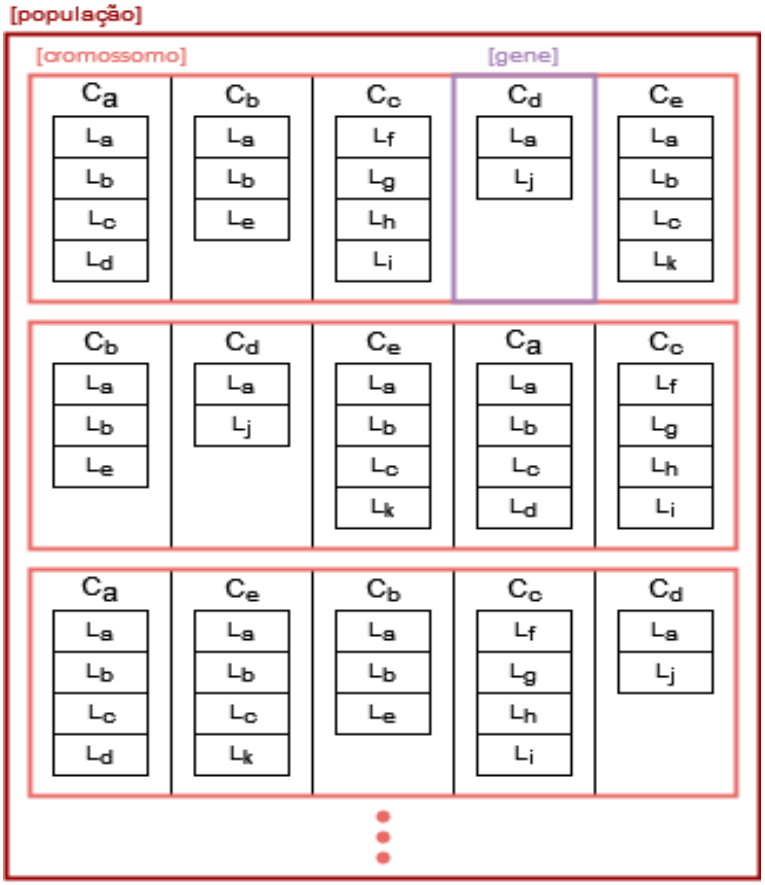

Figura 4.4: Representação da população de cromossomos no EA-DLSLA

Fonte: O autor (2022)

A Figura 4.5 demonstra genericamente as etapas realizadas por um GA. A inicialização da população de indivíduos ocorre após a etapa de filtragem do *kube-scheduler*, selecionando todos os nodos elegíveis. Uma vez que a execução do EA-DLSLA é uma operação custosa, se a fila de *downloads* do nodo contiver menos de 3 contêineres, então aplica-se diretamente a função objetivo para obter a pontuação deste nodo. Para filas maiores, a população inicial e a quantidade de gerações do algoritmo genético são definidos, respectivamente, como $min((tamanhoFila-1)!, 100)$ e $min((tamanhoFila-2)!, 100)$.

Em razão do EA-DLSLA basear-se no NSGA-II, o qual é projetado para resolver problemas de otimização de múltiplos objetivos, a função objetivo (em inglês, *fitness function*) é representada por 4 variáveis a serem minimizadas: *violacoesSLA*, *tempoTotal*, *mudancasNaFila* e *potenciaMaxima*, sendo esta última uma adição específica à variação *Energy-Aware* do DLSLA. O Algoritmo 4.3 exibe a implementação da função objetivo, onde calcula-se, para a fila de *downloads* de um dado nodo: a quantidade de violações de SLA

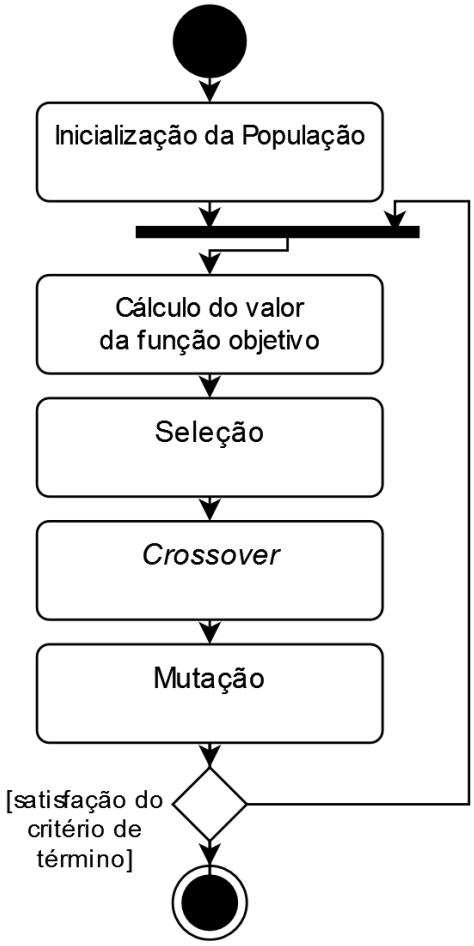

Figura 4.5: Diagrama das etapas de um algoritmo genético

Fonte: O autor (2022)

geradas por esse cromossomo, o tempo necessário para provisionar todos os contêineres, o número de alterações na fila de *downloads*, e a potência máxima do nodo.

Nesta função objetivo, caso o nodo selecionado esteja desligado, o seu atraso de inicialização é acrescido ao tempo total de provisionamento. Então, para cada contêiner de um cromossomo, verifica-se se suas camadas já encontram-se na fila de *downloads* do nodo. Caso uma camada deva ser adicionada a esta fila, o tempo de *download* da camada é incorporado ao tempo de provisionamento. Por fim, caso este atraso ultrapasse as restrições de tempo de provisionamento das aplicações, incrementam-se as violações de SLA.

Após a seleção dos indivíduos mais aptos da população de acordo com as diretrizes do NSGA-II, o EA-DLSLA aplica uma técnica de cruzamento, denominada de *crossover*, com a pretensão de gerar uma nova geração de cromossomos com pontuações ainda maiores que a anterior. Cada par de cromossomos consecutivos representa os pais de

Algoritmo 4.3: Função Objetivo do EA-DLSLA

Entrada: *nodo*: nodo atual; *cromossomo*: cromossomo atual; *larguraBanda*: largura de banda disponível no nodo; *filaDownloads*: fila de *downloads* no nodo
Saída: Pontuação dos objetivos para o nodo

```
violacoesSLA ← 0
mudancasNaFila ← 0
potenciaMaxima ← P^n_max
se estado(nodo) = desligado então
    tempoTotal ← tempoInicializacao(nodo)
fim
senão tempoTotal ← 0
para cada gene ∈ genes(cromossomo) faça
    para cada camada ∈ camadas(gene) faça
        se id(camada) ∉ filaDownloads então
            tempoTotal+ = tamanho(camada) ÷ larguraBanda
            concatenar(filaDownloads, tamanho(camada))
        fim
    fim
    se tempoTotal > sla(gene) então
        violacoesSLA+ = 1
    fim
fim
para cada aplicacao ∈ filaAplicacoes(nodo) faça
    indiceTemp ← indice(filaDownloads, aplicacao)
    remover(filaDownloads, indiceTemp)
    mudancasNaFila+ = indiceTemp
fim
objetivos ← {violacoesSLA, tempoTotal, mudancasNaFila, potenciaMaxima}
retorna objetivos
```

um novo indivíduo, o *filho*. Então, cada gene do cromossomo filho recebe uma cópia de um gene localizado no mesmo *locus*, isto é, na mesma posição da lista de contêineres de um de seus pais, selecionados aleatoriamente. Se o gene copiado já estiver presente no *genótipo*, ou seja, no conjunto de genes do filho, então selecionam-se novos genes dos pais até que o genótipo do cromossomo filho não contenha nenhum gene repetido.

Durante o processo de *crossover*, também ocorre a mutação dos genes, de forma que seja evitada uma solução ótima local. A função de mutação, a qual altera metade do genótipo de um cromossomo, é aplicada em uma quantidade aleatória de novos indivíduos. Após serem executadas iterações do GA pelo número de gerações definido em conjunto com a inicialização da população, os indivíduos mais aptos são selecionados como a melhor solução para um dado nodo.

O DLSLA utiliza a Equação 4.16 para pontuar os nodos. Ao adaptá-la para adicionar as informações sobre eficiência energética, a partir da razão entre a potência máxima produzida pela máquina e a maior potência entre todos os nodos, obtém-se a Equação 4.17:

$$\begin{aligned} \mathit{score}_{\mathit{dlsla}} \leftarrow\ & 30 \\ & - \mathit{tamanho}(\mathit{filaAplicacoes}(\mathit{nodo})) \\ & - 0,5 * \mathit{mudancasNaFila} \\ & - 0,5 * \mathit{violacoesSLA} \\ & + 10 * \mathit{tempoTotal} \end{aligned} \tag{4.16}$$

$$\mathit{score} \leftarrow \mathit{score}_{\mathit{dlsla}} - 10 * \frac{\mathit{potenciaMaxima}}{\mathit{max}(P^1_{max}, ..., P^n_{max})} \tag{4.17}$$

4.5 Considerações Finais

Este capítulo descreveu o modelo de consumo de energia, bem como introduziu o AAS e o EA-DLSLA como meios de aumentar a eficiência energética através do posicionamento de aplicações no contexto de Computação na Borda. Devido ao longo tempo de inicialização dos nodos, essas técnicas também visam minimizar o tempo de provisionamento das aplicações, de modo a garantir um bom desempenho do sistema.

O próximo capítulo aborda a metodologia aplicada para avaliar a efetividade dessas estratégias. Além disso, apresentam-se as funcionalidades adicionadas ao simulador ECOS para realizar as medições de consumo de energia.

5. METODOLOGIA

Este capítulo apresenta a metodologia utilizada para avaliar as técnicas de redução do consumo de energia em aplicações baseadas em contêineres mencionadas no Capítulo 4. As seções do capítulo descrevem: o simulador utilizado e as funcionalidades adicionadas a ele para o desenvolvimento deste trabalho, a topologia considerada para as simulações, a carga de trabalho e os casos de teste.

5.1 *Edge Computing Orchestrating Simulator*

O ECOS, introduzido em [78], é um simulador de aplicações de Computação na Borda desenvolvido com o uso da biblioteca NetworkX [82] através da linguagem de programação Python. Baseando-se em eventos discretos, ele permite a modelagem dos processos de provisionamento, distribuição e transferência de imagens das aplicações em meio à topologia configurável de rede.

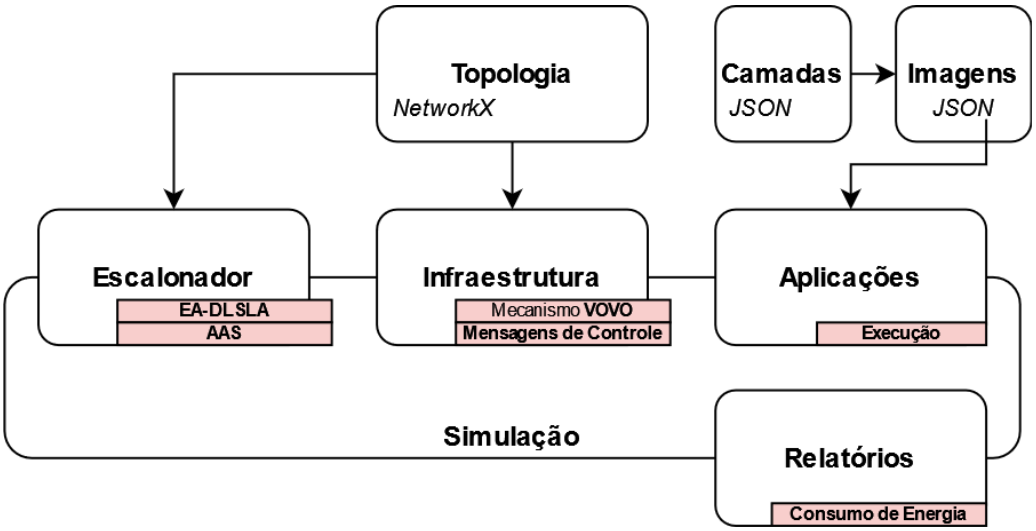

Figura 5.1: Diagrama do Simulador ECOS

Fonte: O autor (2022)

A Figura 5.1 apresenta o *framework* do ECOS, exibindo a relação entre os diversos módulos que compõem o ambiente do simulador. Em destaque estão os submódulos customizados ou adicionados para o desenvolvimento deste trabalho.

O módulo de Simulação é responsável por gerenciar o tempo de simulação e enviar a fila de mensagens para o módulo de Infraestrutura. Esta fila representa as solici-

tações dos usuários para executar novas tarefas ou aplicações. Sob o modelo padrão de orquestração de contêineres, demonstrado na descrição do Kubernetes no Capítulo 2, tipicamente, existe um nodo atribuído com a função de "mestre“, encarregado de armazenar as informações de todos os nodos *workers* e manter os *logs* do escalonador e das aplicações. No ECOS, as funções do nodo mestre são implementadas diretamente no módulo de Infraestrutura.

Ao módulo de Infraestrutura são delegadas as seguintes funções: simular o comportamento de rede através do gerenciamento do compartilhamento da largura de banda na topologia; e gerenciar os recursos dos nodos, como CPU, memória e armazenamento. No desenvolvimento deste trabalho, contribui-se à este módulo com a adição do mecanismo de gerenciamento de energia e com as funções de inicialização e desligamento dos nodos, bem como a criação de diversos tipos de mensagens de controle para facilitar a depuração sobre a operação das máquinas.

O módulo de Aplicações gerencia o ciclo de vida das aplicações durante a simulação. Cada aplicação é controlada por operações de ciclo de vida como *criação* e *provisionamento*, descritas no arquivo de configuração da simulação. Neste arquivo, cada operação relacionada às aplicações possui um *timestamp* que marca o instante em que ela deverá ser aplicada durante uma iteração do simulador. Quando não houverem mais operações de ciclo de vida pendentes, a simulação é encerrada e o controle do programa é passado para o módulo de Relatórios.

O modelo de aplicação neste trabalho baseia-se em na instanciação de tarefas, as quais são executadas por um determinado período, e depois são concluídas, de forma que as imagens de contêineres são mantidas nas máquinas, mas as aplicações são removidas, de forma que sejam liberados os recursos computacionais dos nodos. No entanto, como o ECOS foi desenvolvido com o foco na análise e no desenvolvimento de técnicas de redução do tempo de provisionamento em Computação na Borda, não foi originalmente implementada uma operação de ciclo de vida que representasse a execução de uma tarefa. Sendo assim, neste trabalho, adicionou-se essa funcionalidade.

A implementação do *kube-scheduler* no ECOS baseia-se no código-fonte oficial do Kubernetes [23]. No entanto, o ECOS disponibiliza uma interface de customização que facilita a customização e o desenvolvimento de novas estratégias de escalonamento. As técnicas introduzidas neste trabalho, o AAS e o EA-DLSLA, utilizaram esta interface de

modo que apenas a etapa de classificação fosse alterada, mantendo a implementação padrão da etapa de filtragem e das demais funcionalidades do escalonador.

Por fim, o módulo de Relatórios é responsável por armazenar todos os dados relevantes desde as configurações iniciais até os capturados durante a simulação. Então, diversos relatórios e gráficos podem ser gerados a partir desses dados. Novamente, este módulo foi customizado como requisito para o desenvolvimento deste trabalho, de forma que fossem adicionadas as funcionalidades necessárias para mensurar quanto tempo cada nodo passa em um determinado estado, além do cálculo de consumo de energia.

A configuração do simulador se dá através de um arquivo JSON o qual contém todas os objetos e opções que serão usados como parâmetros da simulação. O Anexo A exibe um exemplo de configuração, a qual envolve diversos vetores, cada qual fornecendo informações sobre um determinado módulo, como: nodos e *links* da topologia; camadas e imagens de contêineres; aplicações e suas operações de ciclo de vida; configurações do escalonador; e níveis de *logs* a serem gerados.

5.2 Topologia

Para simular a infraestrutura de computação na borda, criou-se uma topologia baseada na Rede Ipê [83], demonstrada na 5.2. Essa rede possui 27 pontos de presença (em inglês, Points of Presence PoPs) localizados em cada uma das unidades federativas do Brasil. Além disso, em 2010, foi incorporado um ponto de apoio situado em João Pessoa, no estado de Paraíba, o qual, neste trabalho, também foi considerado como um PoP.

Na topologia utilizada, cada ponto de presença possui 6 nodos computacionais. Um deles exerce a função de um *worker node*, com características semelhantes a um servidor convencional, enquanto os outros 5 são considerados como *edge nodes*, possuindo especificações típicas de um edge device como a Raspberry Pi 4.

A Tabela 5.1 apresenta os parâmetros da simulação em relação às características dos nodos computacionais. Dado que os valores de utilização de CPU no ECOS são representados em mCPU, considera-se que um núcleo de processamento equivale a 1000 mCPU. Os *worker nodes* possuem 16GB de memória RAM, 16 núcleos de processamento e 100Mbps de largura de banda. Quanto à eficiência energética, esses nodos produzem 118W sob utilização máxima da CPU. Ao ficarem inativos, supõe-se que eles produzam

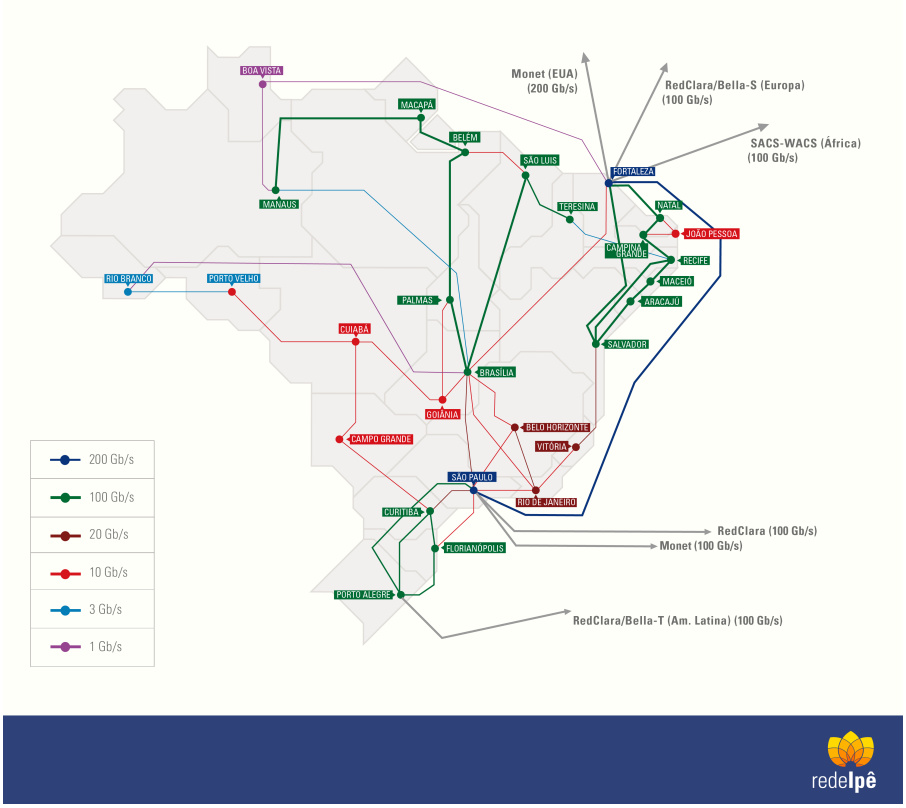

Figura 5.2: Topologia da Rede Ipê

Fonte: Sistema RNP [83]

Tabela 5.1: Características dos nodos utilizados na simulação

Parâmetro	*worker node*	*edge node*
Capacidade de processamento (mCPU)	16000	4000
Memória RAM (GB)	16	4
Potência máxima (W)	118	7
Potência mínima (W)	71	3
Tempo de inicialização (s)	30	45
Tempo de desligamento (s)	15	15

60% da potência máxima, isto é, 71W. Considerou-se o tempo de inicialização e o de desligamento como 30s e 15s, respectivamente.

Os *edge nodes* possuem 4GB de memória RAM, 4 núcleos de processamento e entre 10Mbps e 60Mbps de largura de banda, distribuídos uniformemente. Baseando-se nos dados de consumo de energia de uma Raspberry Pi 4 obtidos em [84], a potência ao provisionar ou executar uma aplicação é de 7W, enquanto a potência quando o edge device está inativo é de 3W, considerando-se em ambos os casos a potência introduzida pelo uso da conexão Ethernet. Por fim, os tempos de inicialização e de desligamento foram considerados como 45s e 15s, respectivamente.

5.3 Carga de Trabalho

Neste trabalho foram utilizadas 3 cargas de trabalho de forma que fosse possível avaliar o desempenho das soluções propostas em diferentes níveis de demanda dos recursos computacionais. A Tabela 5.2 apresenta os parâmetros da simulação em relação às cargas de trabalho.

Tabela 5.2: Parâmetros da simulação

Parâmetro	Carga de Trabalho Leve		Carga de Trabalho Média		Carga de Trabalho Pesada	
	média	desvio	média	desvio	média	desvio
Quantidade de aplicações	600	—	600	—	600	—
Aplicações por lote	3	—	3	—	3	—
Intervalo de chegada do lote (s)	3	—	3	—	3	—
Tamanho da Imagem (MB)	58	11,6	116	23,2	232	46,4
Tempo de Execução (s)	20	8	40	16	80	32
SLA de provisionamento (s)	90	18	90	18	90	18

Em todas as cargas de trabalho foram utilizadas 600 aplicações com um intervalo de chegada de 3 segundos entre cada 3 aplicações. Considerando a carga de trabalho média, o tamanho das imagens baseia-se numa distribuição uniforme das 20 imagens com mais downloads do Docker Hub [85], possuindo média de 116MB e desvio padrão de 23,2.

O tempo de execução das aplicações foi definido como uma distribuição uniforme com média de 40s e desvio padrão de 16. Para fins desta simulação, este tempo de execução independe das características do nodo. Enfim, o SLA de tempo de provisionamento foi estabelecido também por uma distribuição normal com uma média de 90s.

Na carga de trabalho leve, os parâmetros da simulação foram ajustados como uma razão de 50% daqueles da carga de trabalho média, com exceção do SLA, o qual manteve-se constante. Ou seja, as médias do tamanho das imagens e do tempo de execução das aplicações, no caso de *workload* leve, são 58MB e 20s, respectivamente. Na carga de trabalho pesada, multiplicaram-se os parâmetros da carga de trabalho média por 2.

5.4 Casos de Teste

Neste trabalho, além do AAS e do EA-DLSLA, foram utilizados 3 casos de teste, de forma que fosse possível avaliar individualmente a influência tanto da técnica de esca-

lonamento quanto do mecanismo de gerenciamento de energia do sistema. Dois destes casos, descritos abaixo, não realizam o desligamento de máquinas inativas:

- **Kube-Scheduler (KS)**: este é o mecanismo padrão de escalonamento do Kubernetes. Os nodos são selecionados de acordo com as diretrizes de predicados e prioridas descritas no Capítulo 2;
- **Ghanavati et al [21]**: é uma técnica de redução do consumo de energia em aplicações de *Fog Computing* através do posicionamento de aplicações, considerando o tempo de execução como métrica de desempenho do sistema.

O outro caso de teste, denominado **DLSLA-PM**, envolve o uso do DLSLA original como algoritmo de escalonamento, bem como a utilização do mecanismo VOVO proposto no Capítulo 4, em que os nodos computacionais podem ser individualmente ligados ou desligados.

O próximo Capítulo descreve as métricas consideradas na avaliação dos casos de teste. Além disso, nele são apresentados os resultados obtidos a partir das simulações sob diversos cenários com variações da carga de trabalho.

6. RESULTADOS

Este capítulo apresenta os resultados dos casos de teste descritos no Capítulo 5. Na primeira seção, descrevem-se as métricas analisadas. Em seguida, expõem-se os resultados obtidos. Por fim, a última seção exibe comparações entre as diferentes técnicas avaliadas.

6.1 Métricas

As métricas utilizadas em relação à eficiência energética são os valores de consumo quanto aos diversos estados das máquinas, os quais podem ser: ativo, inativo, em inicialização e em desligamento. Cada uma destas métricas, cujos cálculos estão descritos no Capítulo 4, representa a soma do respectivo tipo de consumo em todos os nodos do sistema.

Quanto ao tempo de simulação, as métricas consideradas são o intervalo de tempo dos nodos sob os estados: ativo, inativo, em transição, isto é, inicialização ou desligamento, e completamente desligados. Diferentemente das métricas de consumo de energia, as quais utilizam a soma dos valores dos nodos, estas utilizam as médias de todas as máquinas. Por exemplo, o tempo médio com máquinas no estado inativo calcula-se conforme a Equação 6.1, onde N representa o conjunto de nodos na topologia.

$$T_i = \frac{1}{|N|} \sum_{n=1}^{N} t_i^n \tag{6.1}$$

Por fim, outra métrica utilizada é o número de violações de SLA de provisionamento, de modo que o impacto no desempenho do sistema devido ao emprego de estratégias VOVO possa ser quantificado. Além disso, também considera-se a quantidade de inicializações e desligamentos de nodos do sistema.

6.2 Carga de Trabalho Leve

Ao analisar os dados sobre eficiência energética nas simulações com carga de trabalho leve, expostos na Figura 6.1, percebe-se que o *kube-scheduler* (KS) possui o maior

consumo de energia. Devido às prioridades *LeastRequestedPriority*, que aumenta a pontuação de nodos pouco requisitados, e *SelectorSpreadPriority*, a qual prioriza a distribuição das aplicações por todos os nodos do *cluster*, são utilizados tanto os *edge nodes* quanto os *worker nodes*. No entanto, o consumo de energia de um *worker node* em inatividade é 71Wh, atingindo 118Wh com 100% de utilização máxima de CPU, enquanto a potência máxima produzida por um *edge node* é de apenas 7W.

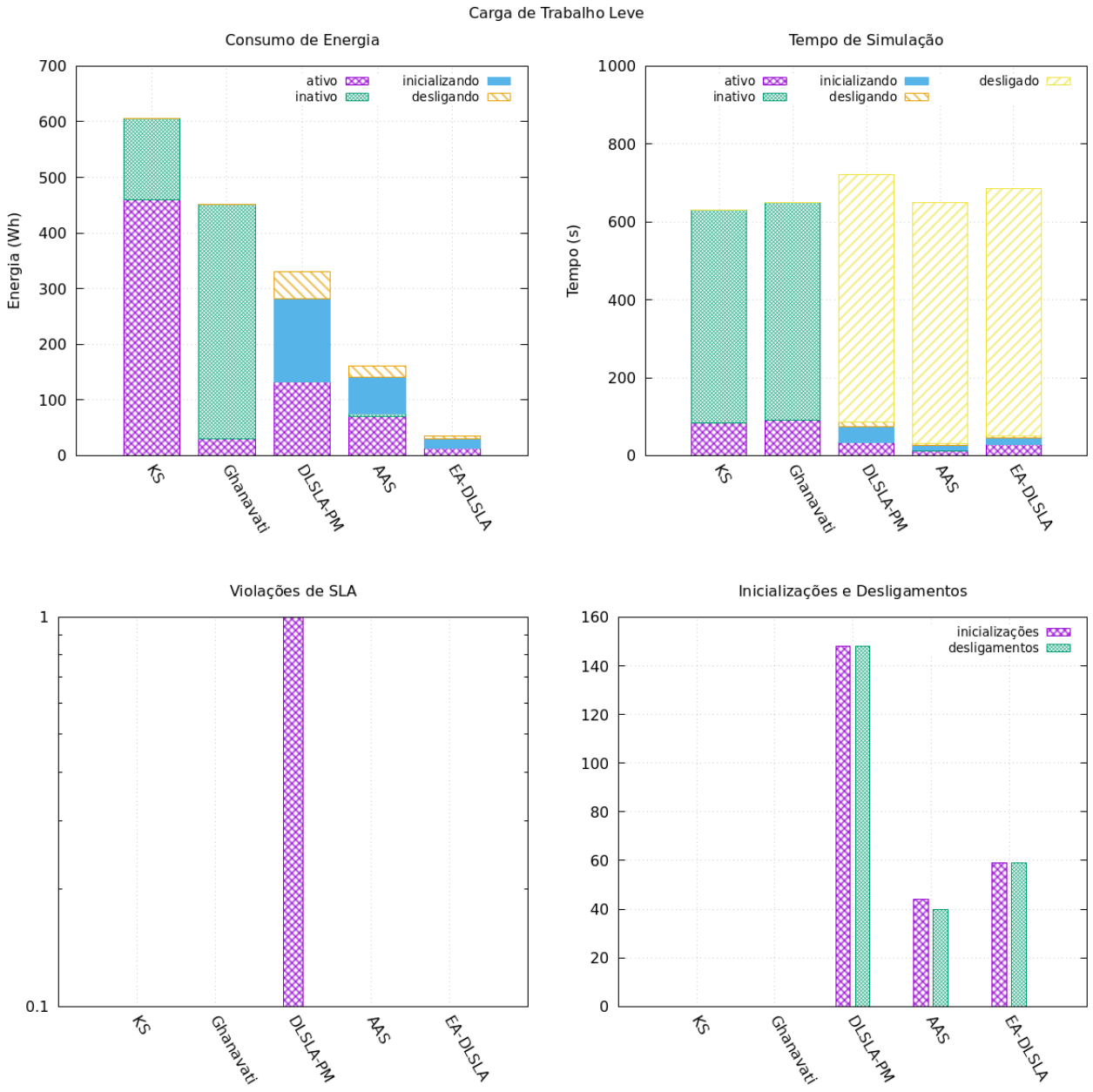

Figura 6.1: Resultados com carga de trabalho leve

Fonte: O autor (2022)

Observa-se que a técnica proposta por Ghanavati[21] et al (GH) reduz drasticamente o consumo ativo, sendo que quase todo o consumo é proveniente de máquinas em

inatividade. Além do mais, o valor de consumo ativo indica que esta técnica prioriza o provisionamento das aplicações para *edge nodes* ao invés de *worker nodes*.

O DLSLA-PM não apresenta consumo inativo, visto que o mecanismo de gerenciamento de energia solicita o encerramento dos nodos assim que entrarem no estado de inatividade. No entanto, como esta técnica não prioriza os nodos ligados, mais da metade do consumo de energia total é proveniente das operações de inicialização e desligamento.

Em contraste ao DLSLA-PM, o AAS prioriza as máquinas ligadas, obtendo um menor consumo de nodos em transição. Ademais, percebe-se que tanto o DLSLA-PM quanto o AAS apresentam consumo ativo significativamente maior do que em GH, indicando que aplicações são escalonadas a *worker nodes*, porém com menor frequência do que no *kube-scheduler*.

Por fim, o EA-DLSLA obteve o menor consumo de energia de todos os casos de teste, apresentando resultados semelhantes à GH, porém sem o consumo em inatividade. Uma vez que o EA-DLSLA prioriza as máquinas ligadas e penaliza os nodos com potência máxima elevada, como os *worker nodes*, percebe-se que as aplicações executaram majoritariamente nos *edge nodes*, como pode ser observado na Tabela 6.1.

Tabela 6.1: Porcentagem das aplicações provisionadas em *edge nodes* ou *worker nodes*

Dispositivos	KS	Ghanavati	DLSLA-PM	KS-AP	EA-DLSLA
edge nodes	11%	100%	32%	17%	100%
worker nodes	89%	0%	68%	83%	0%

Quanto às médias dos tempo de simulação, percebe-se que o *kube-scheduler* e a técnica de Ghanavati et al possuem valores maiores de intervalo de tempo ativo. O intervalo de tempo em que uma tarefa executa em um determinado nodo resulta da velocidade em que o contêiner é adquirido do registro, o que depende das capacidades do nodo e das condições da rede, bem como do tempo de execução da aplicação, definido no arquivo de entrada da simulação. Uma vez que este intervalo de execução é constante para uma dada aplicação, independente do nodo onde ela é provisionada, estipula-se que um valor alto de média de tempo ativo indica que vários nodos estão executando poucas tarefas. No DLSLA-PM, AAS e EA-DLSLA, nota-se que os pequenos intervalos de tempo ativo demonstram que estas técnicas realizam a consolidação das aplicações, onde poucos nodos executam várias aplicações.

Sob a carga de trabalho leve, o DLSLA-PM foi o único caso de teste em que houve alguma violação de SLA. Embora tenha obtido apenas uma violação no provisionamento de 600 aplicações, estima-se que isso ocorra devido a esta técnica não considerar se o nodo está desligado ao decidir pelo escalonamento de uma aplicação nele. Sendo assim, o tempo de inicialização da máquina faz com que não sejam cumpridos os requisitos de tempo de provisionamento.

Quanto à quantidade de inicializações e desligamentos, o DLSLA-PM obteve mais que o dobro das transições do AAS ou do EA-DLSLA, visto que esta técnica não possui a informação de que estará solicitando a inicialização dos nodos. Em contrapartida, o AAS diretamente prioriza as máquinas já ligadas, enquanto o EA-DLSLA penaliza os nodos desligados.

6.3 Carga de Trabalho Média

Os dados das simulações sob a carga de trabalho média estão expostos na Figura 6.2. Ao analisá-los, percebe-se que o comportamento do consumo de energia é semelhante ao da carga de trabalho leve, porém, há um aumento de aproximadamente 15% do consumo total nos diferentes casos de teste. Além disso, a proporção entre o consumo de inicialização/desligamento e o consumo ativo dos DLSLA-PM e AAS são diferente em relação à carga de trabalho leve: ao aumentar a demanda computacional, passa a predominar o consumo de energia enquanto os nodos estão em atividade.

Assim como o consumo de energia, o tempo de simulação teve um leve aumento em relação à carga de trabalho leve. A relação entre os diferentes intervalos de tempo mensurados permaneceu a mesma entre essas cargas de trabalho.

Tanto o DLSLA-PM quanto o AAS tiveram violações de SLA, no entanto, em comparação à carga de trabalho leve, este último obteve 5 vezes mais violações ao aumentar a demanda computacional. A quantidade de inicializações e desligamentos é semelhante à da carga leve. Porém, como as aplicações na carga de trabalho média possuem um maior tempo de execução, há maiores chances do escalonador atribuir uma aplicação para um nodo ainda ligado, reduzindo assim o número de desligamentos em relação ao cenário leve.

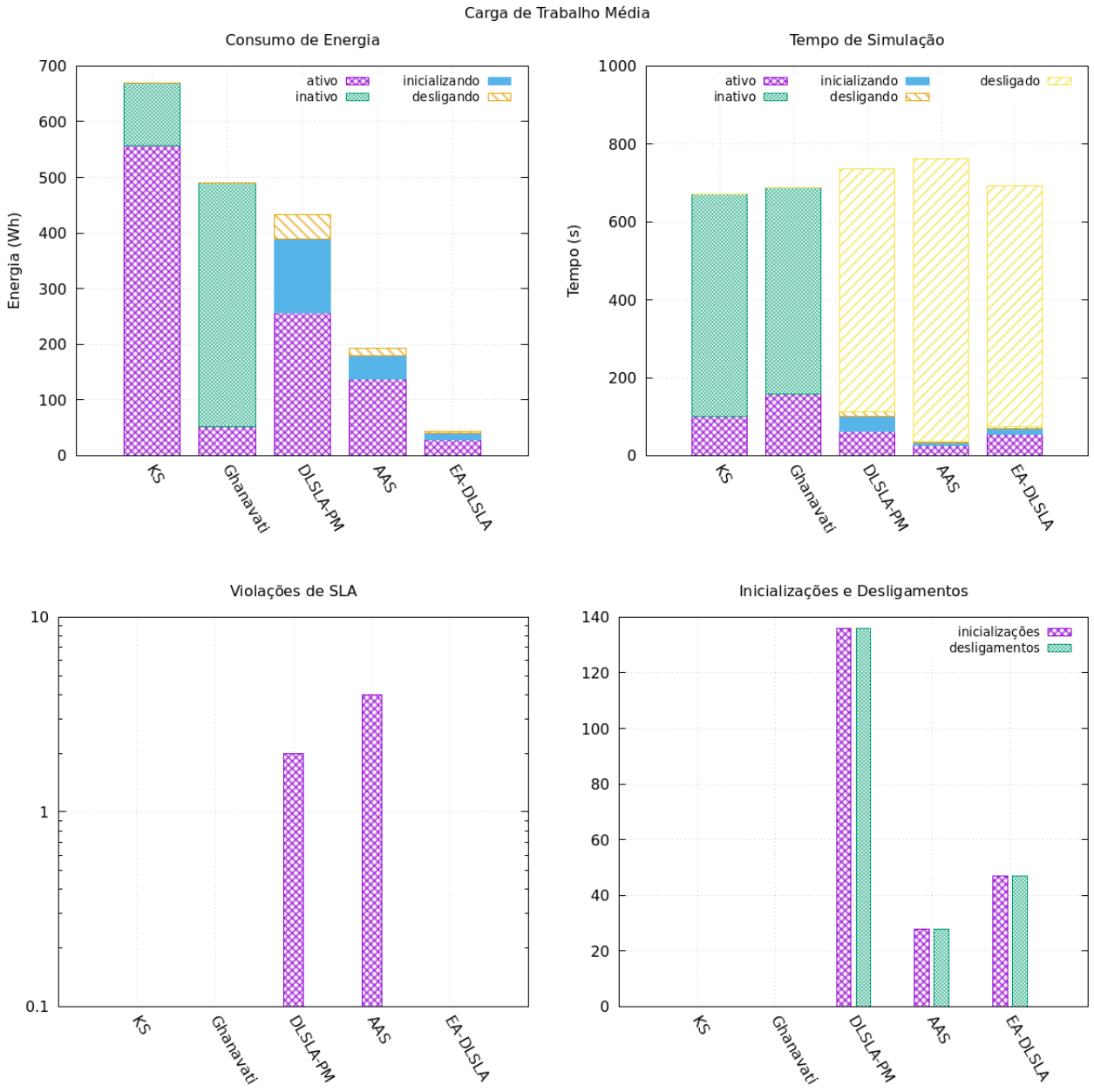

Figura 6.2: Resultados com carga de trabalho média

Fonte: O autor (2022)

6.4 Carga de Trabalho Pesada

Sob uma carga de trabalho pesada, novamente percebe-se um crescimento no consumo de energia nos diferentes casos de teste. Houve um aumento de aproximadamente 8% e 12% em KS e GH, respectivamente. As técnicas que utilizam o mecanismo VOVO, no entanto, tiveram crescimentos mais acentuados: 42% no DLSLA-PM, 80% no AAS e 110% no EA-DLSLA.

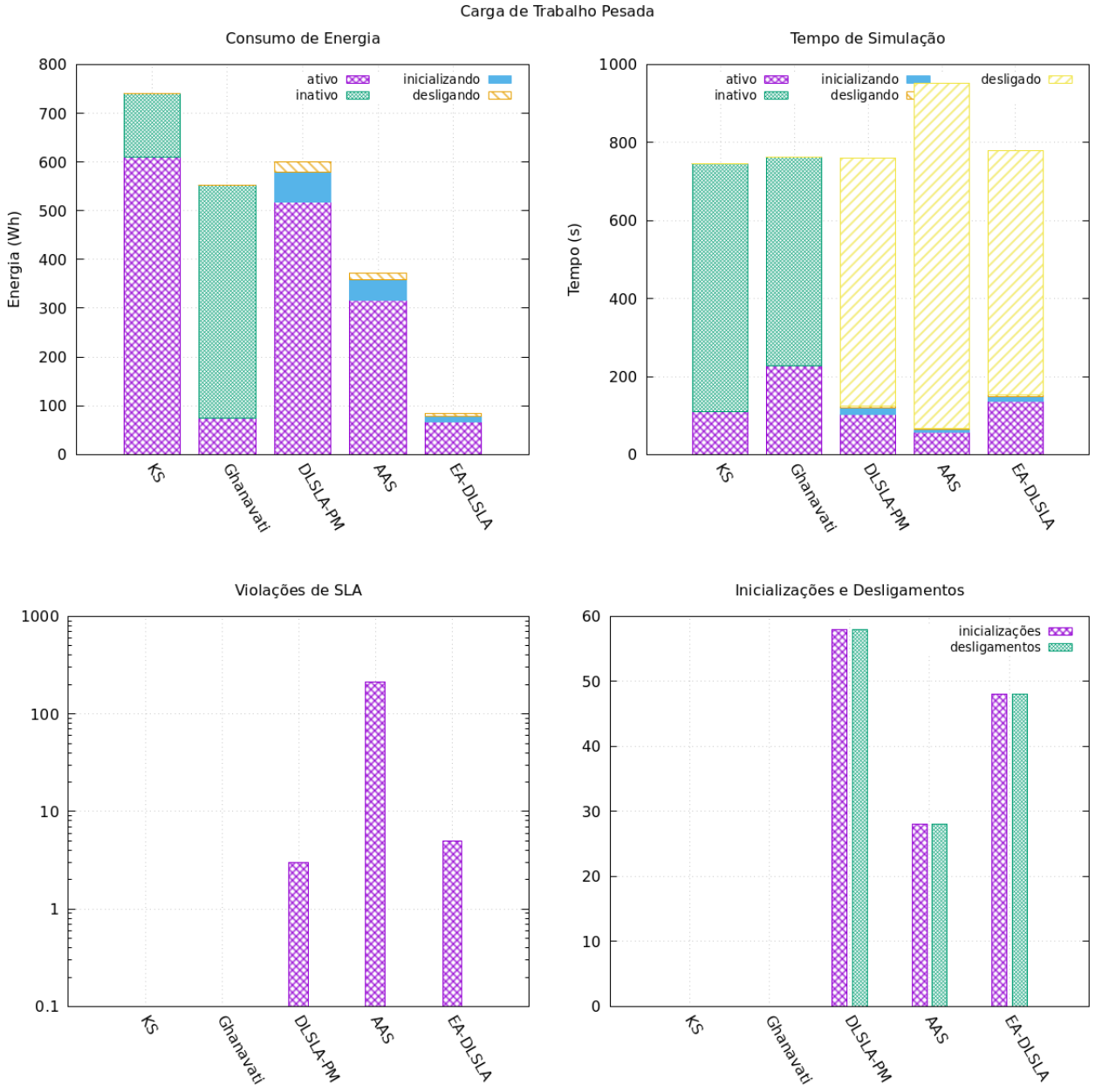

Figura 6.3: Resultados com carga de trabalho pesada

Fonte: O autor (2022)

Diferentemente das outras cargas de trabalho, neste cenário de maior demanda computacional, o consumo de energia na técnica de Ghanavati et al é menor do que no DLSLA-PM, indicando que o mecanismo VOVO não necessariamente obterá a maior eficiência energética. Novamente, nota-se em DLSLA-PM e AAS um aumento da razão entre o consumo ativo e o consumo devido à inicialização e ao desligamento dos nodos. Por fim, o consumo ativo do EA-DLSLA permanece semelhante ao de GH.

Percebe-se um aumento na proporção do tempo de simulação ativo em GH. Uma vez que *edge nodes* possuem menos capacidade computacional, são necessários mais destes dispositivos para executar a mesma quantidade de aplicações que um *worker node* em um dado momento. Como a técnica de Ghanavati et al. envolve a priorização de *edge nodes*, existem mais nodos realizando computações do que nos casos de teste em que os *worker nodes* são utilizados. Dessa forma, a média do tempo em consumo ativo é maior em GH. Aliás, o EA-DLSLA também apresenta este comportamento, devido à grande demanda computacional deste cenário.

Em relação às violações de SLA, o AAS falhou em cumprir os requisitos de tempo de provisionamento de mais de 200 das 600 aplicações. O DLSLA-PM e o EA-DLSLA, respectivamente, obtiveram 3 e 6 violações. Quanto à quantidade de inicializações e desligamentos, nota-se que, embora os valores do AAS e EA-DLSLA sejam próximos dos demais cenários, o número de transições em DLSLA-PM caiu pela metade. Uma vez que as aplicações são concentradas nos poucos *worker nodes* em DLSLA-PM, há menos condições para que um destes nodos fique inativo, e consequentemente seja desligado, em um cenário de alta demanda computacional.

6.5 Considerações Finais

Em relação ao escalonamento padrão do *kube-scheduler*, a técnica de Ghanavati et al. se mostrou eficiente em reduzir o consumo de energia ativo em todos os cenários. Porém, ao considerar a eficiência energética desta estratégia como um todo, o consumo de energia devido aos *worker nodes* inativos a torna ineficiente. O DLSLA-PM obteve resultados intermediários, demonstrando que o desligamento de máquinas subutilizadas reduz o consumo de energia, porém percebe-se um aumento das violações de SLA de tempo de provisionamento.

O AAS apresentou bons resultados de eficiência energética, possuindo, em todos os cenários, o segundo menor consumo de energia, sendo maior apenas que o do EA-DLSLA. No entanto, houveram violações de SLA em todas as cargas de trabalho, tendo ocorrido para mais de um terço das aplicações no cenário de alta demanda computacional. Por fim, o EA-DLSLA apresentou a maior eficiência energética e poucas violações dos requisitos de tempo de provisionamento.

7. CONCLUSÃO E TRABALHO FUTUROS

Neste trabalho, foram introduzidas duas novas técnicas para aumento da eficiência energética no posicionamento de aplicações de Computação na Borda. Diferentemente da maioria dos trabalhos analisados no Capítulo 3 voltados à Computação na Borda, as técnicas propostas realizam o desligamento de máquinas subutilizadas.

As contribuições deste trabalho se dão na forma de dois algoritmos de escalonamento para aplicações em contêineres: o *Availability-Aware Scheduler* (AAS) e o *Energy-Aware Deployment Latency Enforcement Scheduler* (EA-DLSLA). O AAS consiste em pontuar os nodos elegíveis a receber as aplicações baseando-se em prioridades como o estado de energia de uma máquina. O EA-DLSLA utiliza SLAs de tempo de provisionamento, o tempo de inicialização e a potência máxima dos nodos em um algoritmo genético multiobjetivo para a decisão de posicionamento das aplicações.

As soluções propostas, o AAS e o EA-DLSLA, foram validados em um ambiente simulado de Computação na Borda sob diferentes níveis de carga de trabalho. Os resultados demonstram a eficácia do AAS em cenários de baixa demanda computacional, enquanto o EA-DLSLA obteve redução do consumo de energia sem grandes impactos no tempo de provisionamento em todos os cenários. Além disso, avaliaram-se a fim de comparação a eficiência energética e desempenho do sistema do *kube-scheduler*, do DLSLA em combinação ao mecanismo de gerenciamento de energia, e da estratégia desenvolvida por Ghanavati et al.

7.1 Trabalhos Futuros

Neste trabalho, o mecanismo de gerenciamento de energia solicita o desligamento dos nodos assim que ficam inativos. No entanto, futuramente se propõe abordar estratégias diferentes para este mecanismo, como aguardar um determinado período de inatividade para iniciar o encerramento de uma máquina. Desse modo, é possível que uma aplicação seja escalonada a um nodo inativo de forma que o seu desligamento seja cancelado.

Além disso, a eficiência energética do AAS é atribuída à *AvailabilityPriority*, a qual, diferentemente do EA-DLSLA, não considera o consumo de energia de cada nodo. Como

pode-se observar nos comportamentos na técnica de Ghanavati et al. e no EA-DLSLA, ao priorizar as máquinas com baixo consumo de energia, é possível obter maior eficiência energética. Sendo assim, propõe-se avaliar o impacto energético e de tempo de provisionamento das aplicações ao estender o AAS com esta priorização por nodos de baixo consumo.

Por fim, considerou-se que apenas os *edge devices* e *worker nodes* produzem potência significativa na simulação. No entanto, sabe-se que roteadores e *switches* também são responsáveis pelo consumo de energia em infraestruturas de *Fog Computing*, e poderiam ser considerados em uma nova solução.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Ahmed, F.; Naeem, M.; Iqbal, M. “ICT and renewable energy: a way forward to the next generation telecom base stations”, *Telecommunication Systems*, vol. 64, Janeiro 2017, pp. 44–56.

[2] Malmodin, J.; Lundén, D. “The Energy and Carbon Footprint of the Global ICT and E&M Sectors 2010–2015”, *Sustainability*, vol. 10–9, Agosto 2018, p. 3027.

[3] Johnston, S. J.; Basford, P. J.; Perkins, C. S.; Herry, H.; Tso, F. P.; Pezaros, D.; Mullins, R. D.; Yoneki, E.; Cox, S. J.; Singer, J. “Commodity single board computer clusters and their applications”, *Future Generation Computer Systems*, vol. 89, Dezembro 2018, pp. 201–212.

[4] Jiang, C.; Fan, T.; Gao, H.; Shi, W.; Liu, L.; Cérin, C.; Wan, J. “Energy aware edge computing: A survey”, *Computer Communications*, vol. 151, Fevereiro 2020, pp. 556–580.

[5] Kayser, C. H. “Minimizing Container-Based Applications SLA Violations on Edge Computing Environments”, (Dissertação de Mestrado), Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2022, 46p.

[6] Knob, L. A. D. “Improving Container Deployment Latency in Distributed Edge Infrastructures”, (Tese de Doutorado), Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2022, 122p.

[7] Satyanarayanan, M. “The Emergence of Edge Computing”, *Computer*, vol. 50–1, Janeiro 2017, pp. 30–39.

[8] Bilal, K.; Khan, S.; Zomaya, A. “Trends and Challenges in Cloud Data centers”, *IEEE Cloud Computing*, vol. 1–1, Junho 2014, pp. 10–20.

[9] Sharma, N. K.; Reddy, G. R. M. “Multi-Objective Energy Efficient Virtual Machines Allocation at the Cloud Data Center”, *IEEE Transactions on Services Computing*, vol. 12–1, Jan-Fev 2019, pp. 158–171.

[10] Rossi, F. D.; Xavier, M. G.; De Rose, C. A.; Calheiros, R. N.; Buyya, R. “E-eco: Performance-aware energy-efficient cloud data center orchestration”, *Journal of Network and Computer Applications*, vol. 78, Janeiro 2017, pp. 83–96.

[11] Yadav, R.; Zhang, W.; Kaiwartya, O.; Singh, P. R.; Elgendy, I. A.; Tian, Y.-C. “Adaptive Energy-Aware Algorithms for Minimizing Energy Consumption and SLA Violation in Cloud Computing”, *IEEE Access*, vol. 6, Outubro 2018, pp. 55923–55936.

[12] Chen, L.; Li, J.; Ma, R.; Guan, H.; Jacobsen, H.-A. “Balancing Power And Performance In HPC Clouds”, *The Computer Journal*, vol. 63–1, Janeiro 2020, pp. 880–899.

[13] Jabour, I. M.; Al-Libawy, H. “An Optimized Approach for Efficient-Power and Low-Latency Fog Environment Based on the PSO Algorithm”. In: Information Technology To Enhance e-learning and Other Application, 2021, pp. 52–57.

[14] Sharifi, F.; Hessabi, S.; Rasaii, A. “The Effect of Fog Offloading on the Energy Consumption of Computational Nodes”. In: International Symposium on Real-Time and Embedded Systems and Technologies, 2022, pp. 1–6.

[15] Ergun, K.; Ayoub, R.; Mercati, P.; Liu, D.; Rosing, T. “Energy and QoS-Aware Dynamic Reliability Management of IoT Edge Computing Systems”. In: Asia and South Pacific Design Automation Conference, 2021, pp. 561–567.

[16] Hu, H.; Song, W.; Wang, Q.; Hu, R. Q.; Zhu, H. “Energy Efficiency and Delay Tradeoff in an MEC-Enabled Mobile IoT Network”, *IEEE Internet of Things Journal*, vol. 9–17, Setembro 2022, pp. 15942–15956.

[17] Mahapatra, A.; Mishra, K.; Majhi, S. K.; Pradhan, R. “EFog-IoT: Harnessing Power Consumption in Fog-Assisted of Things”. In: IEEE Region 10 Symposium, 2022, pp. 1–6.

[18] Gu, L.; Cai, J.; Zeng, D.; Zhang, Y.; Jin, H.; Dai, W. “Energy efficient task allocation and energy scheduling in green energy powered edge computing”, *Future Generation Computer Systems*, vol. 95, Junho 2019, pp. 89–99.

[19] Vadde U, K. V. “Energy efficient service placement in fog computing”, *PeerJ Computer Science 8*, vol. 8, Julho 2022, p. e1035.

[20] Alnoman, A.; Anpalagan, A. “QoS-aware Energy Saving Scheme and Traffic Management in Mobile Edge Computing Networks”. In: International Wireless Communications and Mobile Computing, 2021, pp. 1925–1930.

[21] Ghanavati, S.; Abawajy, J.; Izadi, D. “An Energy Aware Task Scheduling Model Using Ant-Mating Optimization in Fog Computing Environment”, *IEEE Transactions on Services Computing*, vol. 15–4, Jul-Ago 2022, pp. 2007–2017.

[22] Mach, P.; Becvar, Z. “Mobile Edge Computing: A Survey on Architecture and Computation Offloading”, *IEEE Communications Surveys Tutorials*, vol. 19–3, Jul-Set 2017, pp. 1628–1656.

[23] The Kubernetes Authors. “Kubernetes”. Recuperado de: https://kubernetes.io/, Novembro 2022.

[24] Chen, S.; Xu, H.; Liu, D.; Hu, B.; Wang, H. “A Vision of IoT: Applications, Challenges, and Opportunities With China Perspective”, *IEEE Internet of Things Journal*, vol. 1–4, Agosto 2014, pp. 349–359.

[25] Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. “Edge Computing: Vision and Challenges”, *IEEE Internet of Things Journal*, vol. 3–5, Outubro 2016, pp. 637–646.

[26] Mahmoudi, C.; Mourlin, F.; Battou, A. “Formal definition of edge computing: An emphasis on mobile cloud and IoT composition”. In: International Conference on Fog and Mobile Edge Computing, 2018, pp. 34–42.

[27] IEEE. “Real-Life Use Cases for Edge Computing”. Recuperado de: https://innovationatwork.ieee.org/real-life-edge-computing-use-cases/, Novembro 2022.

[28] Al Mtawa, Y.; Haque, A.; Bitar, B. “The Mammoth Internet: Are We Ready?”, *IEEE Access*, vol. 7, Setembro 2019, pp. 132894–132908.

[29] Choudhury, T.; Gupta, A.; Pradhan, S.; Kumar, P.; Rathore, Y. S. “Privacy and Security of Cloud-Based Internet of Things (IoT)”. In: International Conference on Computational Intelligence and Networks, 2017, pp. 40–45.

[30] Kaur, N.; Sood, S. K. “An Energy-Efficient Architecture for the Internet of Things (IoT)”, *IEEE Systems Journal*, vol. 11–2, Junho 2017, pp. 796–805.

[31] Centanni, T.; Sloan, A.; Reed, A.; Engineer, C.; Rennaker, R.; Kilgard, M. “Detection and identification of speech sounds using cortical activity patterns”, *Neuroscience*, vol. 258, Janeiro 2014, pp. 292 – 306.

[32] Elbamby, M. S.; Perfecto, C.; Bennis, M.; Doppler, K. “Toward Low-Latency and Ultra-Reliable Virtual Reality”, *IEEE Network*, vol. 32–2, Março 2018, pp. 78–84.

[33] Avino, G.; Giordanino, M.; Franzoudis, P. A.; Vitale, C.; Casetti, C.; Chiasserini, C. F.; Gebru, K.; Ksentini, A.; Stojanovic, A. “A MEC-based Extended Virtual Sensing for Automotive Services”. In: International Conference of Electrical and Electronic Technologies for Automotive, 2019, pp. 1–6.

[34] Meta. “Instagram”. Recuperado de: https://www.instagram.com/, Agosto 2022.

[35] Meta. “Facebook”. Recuperado de: https://www.facebook.com/, Agosto 2022.

[36] Google. “Youtube”. Recuperado de: https://www.youtube.com, Agosto 2022.

[37] Microsoft. “Armazenamento em nuvem pessoal do OneDrive”. Recuperado de: https://www.microsoft.com/pt-br/microsoft-365/onedrive/online-cloud-storage, Agosto 2022.

[38] Google. “Google Drive”. Recuperado de: https://www.google.com/intl/pt-br/drive/about.html, Agosto 2022.

[39] Dolui, K.; Datta, S. K. “Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing”. In: Global Internet of Things Summit, 2017, pp. 1–6.

[40] Zhao, P.; Tian, H.; Qin, C.; Nie, G. “Energy-Saving Offloading by Jointly Allocating Radio and Computational Resources for Mobile Edge Computing”, *IEEE Access*, vol. 5, Junho 2017, pp. 11255–11268.

[41] Hardesty, L. “American tower has deployed 6 edge data centers at tower sites”. Recuperado de: https://www.fiercewireless.com/tech/american-tower-has-deployed-6-edge-data-centers-at-tower-sites, Agosto 2022.

[42] Miller, R. “American tower begins deploying edge data centers at towers”. Recuperado de: https://www.datacenterfrontier.com/edge-computing/article/11428863/american-tower-begins-deploying-edge-data-centers-at-towers, Novembro 2022.

[43] Morabito, R.; Kjällman, J.; Komu, M. “Hypervisors vs. lightweight virtualization: A performance comparison”. In: IEEE International Conference on Cloud Engineering, 2015, pp. 386–393.

[44] Canonical. “Container and virtualization tools”. Recuperado de: https://linuxcontainers.org/, Novembro 2022.

[45] Canonical. “What is LXD?” Recuperado de: https: //linuxcontainers.org/lxd/introduction/, Novembro 2022.

[46] Raza, M.; Kidd, C. “Virtual Machines (VMs) vs Containers: Whats The Difference?” Recuperado de: https://www.bmc.com/blogs/containers-vs-virtual-machines/, Novembro 2022.

[47] Vaughan-Nichols, S. “What is Docker and why is it so darn popular?” Recuperado de: https://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/, Novembro 2022.

[48] Docker. “Use the Docker command line”. Recuperado de: https://docs.docker.com/ engine/reference/commandline/cli/, Novembro 2022.

[49] Docker. “About storage drivers”. Recuperado de: https://docs.docker.com/storage/ storagedriver/, Novembro 2022.

[50] Docker. “Docker Registry”. Recuperado de: https://docs.docker.com/registry/, Novembro 2022.

[51] MSV, J. “From Containers to Container Orchestration”. Recuperado de: https://medium.com/@janakiramm/ from-containers-to-container-orchestration-407eca9663d3, Novembro 2022.

[52] Ellingwood, J. “Uma Introdução ao Kubernetes”. Recuperado de: https: //www.digitalocean.com/community/tutorials/uma-introducao-ao-kubernetes-pt, Novembro 2022.

[53] The Kubernetes Authors. “Understanding kubernetes objects”. Recuperado de: https: //kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/, Novembro 2022.

[54] Ushio, T. “Kubernetes in three diagrams”. Recuperado de: https://tsuyoshiushio.medium.com/kubernetes-in-three-diagrams-6aba8432541c, Novembro 2022.

[55] Bashir, F. “A friendly introduction to Kubernetes”. Recuperado de: https://www.freecodecamp.org/news/a-friendly-introduction-to-kubernetes-670c50ce4542/, Novembro 2022.

[56] The Kubernetes Authors. “Concepts”. Recuperado de: https://kubernetes.io/docs/concepts/, Novembro 2022.

[57] W3C. “Web Services Architecture”. Recuperado de: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, Novembro 2022.

[58] The Kubernetes Authors. “Scheduling Policies”. Recuperado de: https://kubernetes.io/docs/reference/scheduling/policies/, Novembro 2022.

[59] Fu, P. “Kubernetes Scheduler Introduction”. Recuperado de: https://www.cncf.io/blog/2022/03/28/kubernetes-scheduler-introduction/, Set 2021.

[60] Elnozahy, E. N. M.; Kistler, M.; Rajamony, R. “Energy-Efficient Server Clusters”. In: International Workshop on Power-Aware Computer Systems, 2002, pp. 179–197.

[61] Rajamani, K.; Lefurgy, C. “On evaluating request-distribution schemes for saving energy in server clusters”. In: IEEE International Symposium on Performance Analysis of Systems and Software, 2003, pp. 111–122.

[62] Duy, T. V. T.; Sato, Y.; Inoguchi, Y. “Performance evaluation of a Green Scheduling Algorithm for energy savings in Cloud computing”. In: IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, 2010, pp. 1–8.

[63] Mathew, V.; Sitaraman, R. K.; Shenoy, P. “Energy-aware load balancing in content delivery networks”. In: IEEE International Conference on Computer Communications, 2012, pp. 954–962.

[64] Raspberry Pi Foundation. “Raspberry Pi”. Recuperado de: https://www.raspberrypi.com/, Agosto 2022.

[65] Raspberry Pi Foundation. “Raspberry Pi 4 - Sleep Mode”. Recuperado de: https://forums.raspberrypi.com/viewtopic.php?t=243421, Novembro 2022.

[66] Shenzhen Xunlong Software Co. “Orange Pi”. Recuperado de: http://www.orangepi.org/index.html, Agosto 2022.

[67] Armbian. “Orange Pi Zero Randomly Hangs”. Recuperado de: https://forum.armbian.com/topic/5237-orange-pi-zero-random-hangs/, Novembro 2022.

[68] Alnoman, A.; Anpalagan, A. “A SDN-Assisted Energy Saving Scheme for Cooperative Edge Computing Networks”. In: IEEE Global Communications Conference, 2019, pp. 1–6.

[69] Rausch, T.; Raith, P.; Pillai, P.; Dustdar, S. “A System for Operating Energy-Aware Cloudlets: Demo”. In: ACM/IEEE Symposium on Edge Computing, 2019, pp. 307–309.

[70] Judge, J.; Pouchet, J.; Ekbote, A.; Dixit, S. “Reducing data center energy consumption”, *ASHRAE Journal*, vol. 50–11, Novembro 2008, p. 14.

[71] Rausch, T.; Avasalcai, C.; Dustdar, S. “Portable Energy-Aware Cluster-Based Edge Computers”. In: IEEE/ACM Symposium on Edge Computing, 2018, pp. 260–272.

[72] Vasques, T.; Moura, P.; Almeida, A. “A review on energy efficiency and demand response with focus on small and medium data centers”, *Energy Efficiency*, vol. 12, Junho 2019, pp. 1399–1428.

[73] Fuchs, H.; Shehabi, A.; Ganeshalingam, M.; Desorches, L.-B.; Lim, B.; Roth, K.; Tsao, A. “Characteristics and Energy Use of Volume Servers in the United States”, (Relatório Técnico), Lawrence Berkeley National Laboratory, 2022, 50p.

[74] SPEC. “SPECpower_ssj2008 benchmark”. Recuperado de: http://www.spec.org/power_ssj2008/results/, Novembro 2022.

[75] ENERGY STAR. “ENERGY STAR”. Recuperado de: https://www.energystar.gov/, Novembro 2022.

[76] Shehabi, A.; Smith, S.; Sartor, D.; Brown, R.; Herrlin, M.; Koomey, J.; Masanet, E.; Horner, N.; Azevedo, I.; Lintner, W. “United States Data Center Energy Usage Report”, (Relatório Técnico), Lawrence Berkeley National Laboratory, 2016, 65p.

[77] Fu, S.; Mittal, R.; Zhang, L.; Ratnasamy, S. “Fast and Efficient Container Startup at the Edge via Dependency Scheduling”. In: USENIX Workshop on Hot Topics in Edge Computing, 2020, p. 7.

[78] Knob, L. A. D.; Kayser, C. H.; de Souza, P. S. S.; Ferreto, T. “Enforcing Deployment Latency SLA in Edge Infrastructures through Multi-Objective Genetic Scheduler”. In: IEEE/ACM International Conference on Utility and Cloud Computing, 2021, pp. 1–9.

[79] Oliveto, P. S.; He, J.; Yao, X. “Evolutionary algorithms and the Vertex Cover problem”. In: IEEE Congress on Evolutionary Computation, 2007, pp. 1870–1877.

[80] Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. “A fast and elitist multiobjective genetic algorithm: NSGA-II”, *IEEE Transactions on Evolutionary Computation*, vol. 6–2, Abril 2002, pp. 182–197.

[81] Linden, R. “Algoritmos Genéticos”. Brasport, 2008, 398p.

[82] NetworkX. “Networkx — networkx documentation”. Recuperado de: https://networkx.org/, Novembro 2022.

[83] Rede Nacional de Ensino e Pesquisa - RNP. “Conexão atual - Rede Ipê”. Recuperado de: https://www.rnp.br/sistema-rnp/rede-ipe, Novembro 2022.

[84] Bekaroo, G.; Santokhee, A. “Power consumption of the Raspberry Pi: A comparative analysis”. In: IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies, 2016, pp. 361–366.

[85] Docker. “Docker Hub Container Image Library”. Recuperado de: https://hub.docker.com/, Novembro 2022.

APÊNDICE A – EXEMPLO DE ARQUIVO DE ENTRADA DO ECOS

```
    {
    "nodes": [
        {"type":"netabs", "name":"switchA"},
        {"type":"registry", "name":"registry", "position": [0, 0],
            "images": ["postgres:latest", "mysql:latest"]},
        {"type":"chost", "name":"edge1", "registry": ["registry"], "position": [0, 0],
            "cpuCapacity": 4000, "memoryCapacity": "4GB",
            "storageCapacity": "10GB", "simultaneousDownload":1,
            "labels": { "bandwidth": "100Mb", "wakeUpTime" : 15, "shutdownTime": 30,
                        "powerConsumptionActive" : 7, "powerConsumptionIdle" : 3
            }
        }
    ],
    "edges":[
        {"bandwidth":"100Mb", "from":"switchA", "to":"registry", "bidirectional":true},
        {"bandwidth":"100Mb", "from":"switchA", "to":"edge1", "bidirectional":true}
    ],
    "layers": [
        {"size":"50MB", "digest":"1111"}, {"size":"50MB", "digest":"2222"}
    ],
    "images":[
        {"name":"postgres", "versions":[{"digest":"6666",  "tag":"latest",
            "layers": ["2222","1111"]}], "startDelay": 15}
    ],
    "applications":[
        {"name": "postgres", "image": "postgres",
            "cpuRequired": 500, "memoryRequired": "256MB"}
    ],
    "lifecyleOperations": [
        {"application": "postgres", "name": "deployment", "type": "creation",
            "tag": "latest", "numReplicas": 1, "time": 0,
            "labels": { "sla": 20, "executionTime": 40}}
    ],
    "infrastructure": {
        "scheduler": {
            "name": "EnergyAwareSLADeploymentLatencyScheduler",
            "config": {
                "predicates": ["PodFitsResources"],
                "priorities": [{"name": "LeastRequestedPriority", "weight": 10}]
            }
        },
        "logLevel": "DEBUG",
        "log": ["netstat"]
    }
}
```

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Pesquisa e Pós-Graduação
Av. Ipiranga, 6681 – Prédio 1 – Térreo
Porto Alegre – RS – Brasil
Fone: (51) 3320-3513
E-mail: propesq@pucrs.br
Site: www.pucrs.br