UNIVERSIDADE FEDERAL DO PAMPA

Ivan Mangini Lopes Junior

Escalonamento de Contêineres com Base em Prioridades para Aplicações em Edge Computing Sensíveis à Latência

Alegrete
Julho de 2026

Ivan Mangini Lopes Junior

Escalonamento de Contêineres com Base em Prioridades para Aplicações em Edge Computing Sensíveis à Latência

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

Orientador: Fábio Diniz Rossi

Alegrete
Julho de 2026

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

L864e Lopes Junior, Ivan Mangini
Escalonamento de contêineres com base em prioridades para aplicações em edge computing sensíveis à latência / Ivan Mangini Lopes Junior.
53 p.

Dissertação(Mestrado)-- Universidade Federal do Pampa, MESTRADO EM ENGENHARIA ELÉTRICA, 2026.
"Orientação: Fábio Diniz Rossi".

1. Provisionamento. 2. Conteiner. 3. Computação em borda. I. Título.

IVAN MANGINI LOPES JUNIOR

ESCALONAMENTO DE CONTÊINERES COM BASE EM PRIORIDADES PARA APLICAÇÕES EM EDGE COMPUTING SENSÍVEIS À LATÊNCIA

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

Dissertação defendida e aprovada em: 06/07/2026.

Banca examinadora:

_____

Prof. Dr. Fábio Diniz Rossi

Orientador

IFFar

_____

Prof. Dr. Marcelo Caggiani Luizelli

Unipampa

_____

Prof. Drª. Josiane Fontoura dos Anjos

IFFar

Assinado eletronicamente por **MARCELO CAGGIANI LUIZELLI**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 14/07/2026, às 15:51, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **FABIO DINIZ ROSSI**, **PESSOAL VOLUNTÁRIO**, em 14/07/2026, às 16:05, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Josiane Fontoura dos Anjos**, **Usuário Externo**, em 14/07/2026, às 18:01, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2072269** e o código CRC **3D2D5262**.

AGRADECIMENTOS

Cada conquista, aprimoramento ou melhora em nosso desenvolvimento pessoal deve ser celebrada. No entanto, cabe reconhecer aqueles que, no processo, apoiaram-me, guiaram-me ou, carinhosamente, confortaram-me em momentos de angústia e dúvida. Este trabalho representa o encerramento de um ciclo e, também, o surgimento de um novo, repleto de oportunidades. Definitivamente, tal realidade era antes inimaginável para quem, por muito tempo, serviu apenas como ferramenta, cedendo sua força de trabalho longe do ambiente acadêmico.

Gostaria de registrar meu profundo agradecimento à minha esposa, Suziane Borges de David, pelo inestimável apoio e incentivo desde o início de minha graduação até a conclusão desta pós-graduação. Ela sempre foi minha maior incentivadora e companheira em todos os momentos. Quero também expressar um imenso agradecimento à Nira Mari Costa Goulart, a quem orgulhosamente chamo de mãe. Você sempre foi um exemplo de resiliência e força; devo a você a pessoa que sou hoje.

Ao meu estimado orientador, Prof. Dr. Fábio Diniz Rossi, um profissional exemplar e uma pessoa incrível, agradeço por ter me guiado e orientado de forma ativa, porém leve e descontraída, mesmo estando, em alguns momentos, do outro lado do mundo.

Por fim, aos professores que me acompanharam desde a graduação até este mestrado, deixo meu mais sincero agradecimento. Vocês foram mentores que, com dedicação e rigor, lapidaram minha curiosidade e me forneceram as ferramentas necessárias para evolução profissional e pessoal. Sou eternamente grato por cada ensinamento e conselho que contribuíram para que eu chegasse a este momento de realização.

RESUMO

Aplicações de borda com alta prioridade exigem análise preditiva em tempo real e sincronização contínua, tornando o provisionamento oportuno um requisito crítico em ambientes com recursos limitados. Garantir a operação ininterrupta dessas aplicações demanda mecanismos inteligentes de escalonamento capazes de lidar com contenção sem comprometer a qualidade de serviço (QoS). Embora tecnologias de conteinerização como o Docker ofereçam implantação leve e modularidade, elas enfrentam gargalos de desempenho em redes de borda com largura de banda restrita, especialmente durante o download de imagens de contêiner. Este estudo introduz o PrIO, um algoritmo de otimização de provisionamento que prioriza dinamicamente os downloads de camadas de contêiner com base em uma avaliação multicritério que considera a criticidade da aplicação, o tempo de espera e a duração estimada de execução. Diferentemente de abordagens naives ou estáticas, o PrIO adapta-se às variações de carga de trabalho e explora inteligentemente a largura de banda ociosa para manter a equidade no provisionamento e o cumprimento de prazos. Os resultados experimentais demonstram que, mesmo sob ocupação elevada do sistema (até 75%), o PrIO sustenta a QoS de aplicações críticas enquanto reduz significativamente os atrasos no provisionamento de serviços com menor prioridade.

Palavras-chave: Computação em borda, Docker, desempenho, provisionamento.

ABSTRACT

High-priority edge applications demand real-time predictive analysis and continuous synchronization, making timely provisioning critical in resource-constrained edge environments. Ensuring uninterrupted operation for such applications requires intelligent scheduling mechanisms capable of handling contention without degrading QoS. Although containerization technologies like Docker offer lightweight deployment and modularity, they face performance bottlenecks in bandwidth-limited edge networks, especially during container image downloads. This study introduces PrIO, a provisioning optimization algorithm that dynamically prioritizes container layer downloads based on a multi-criteria evaluation encompassing application criticality, waiting time, and estimated execution duration. Unlike naive or static approaches, PrIO adapts to workload variations and intelligently exploits idle bandwidth to maintain provisioning fairness and deadline adherence. Experimental results demonstrate that, even under high system occupancy up to 75%, PrIO sustains the QoS of high-priority edge applications while significantly reducing provisioning delays for lower-priority services.

Keywords: Edge computing, Docker, performance, provisioning.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A computação em borda viabiliza processamento próximo aos dados e consolidou-se como um paradigma essencial para a infraestrutura da internet, propondo a descentralização de recursos de processamento e armazenamento para a periferia da rede. Adicionalmente, esse paradigma propicia melhorias na utilização de recursos de rede ao integrar dispositivos finais a capacidades computacionais de proximidade, suportando uma vasta gama de novas aplicações sensíveis ao tempo. Ainda, a computação em borda proporciona baixa latência e redução de custos na transmissão, consumo energético e armazenamento dos dados. [1].

Contudo, a eficácia desse modelo enfrenta restrições severas impostas pela disparidade entre as demandas das aplicações e as capacidades dos nós de borda. Enquanto as aplicações modernas são caracterizadas pelo intensivo uso de recursos, assim sendo, demandam alto poder de processamento, os dispositivos de borda operam frequentemente sob restrições estritas de hardware como processamento e persistência de dados. Diferente do ambiente de nuvem (*Cloud*), a borda é caracterizada como um ecossistema transiente e volátil, no qual a disponibilidade de recursos oscila consideravelmente ao longo do tempo e pode ser garantida apenas por períodos reduzidos. Logo, a computação em borda, impõe restrições em termos de largura de banda, recursos computacionais, provisionamento de tarefas e segurança. [2].

A demanda por provisionamento eficiente e otimização de recursos tem se intensificado em tais ambientes, nos quais diversas aplicações com diferentes níveis de urgência e criticidade competem por recursos limitados. Aplicações como sistemas autônomos, *gaming*, realidade virtual e realidade aumentada demandam processamento em tempo real, ou com o mínimo atraso possível, em localizações distribuídas e com conectividade massiva de dispositivos [3]. Desta forma, tem-se como desafio provisionar aplicações heterogêneas sob restrições de rede e *offloading* de recursos computacionais. Nesses contextos, é essencial garantir que aplicações prioritárias sejam provisionadas com o mínimo de atraso, mesmo sob condições de recursos limitados ou redes com pouca largura de banda. Essas aplicações frequentemente requerem sincronização contínua, implantação rápida e desempenho confiável, uma vez que qualquer atraso ou contenção de recursos pode afetar diretamente sua funcionalidade e a qualidade de serviço (QoS) entregue aos usuários finais [4].

Para lidar com essas restrições, arquiteturas baseadas em contêineres tornaram-se padrão devido à sua leveza e natureza modular. Contêineres encapsulam todos os componentes necessários, como código, ambiente de execução, bibliotecas e dependências, permitindo uma execução consistente e eficiente em infraestruturas heterogêneas. Microsserviços ampliam ainda mais essa modularidade ao decompor aplicações em unidades implantáveis de forma independente, cada uma otimizada para funcionalidades específicas. Essas abordagens possibilitam *pipelines* de implantação escaláveis, flexíveis e ágeis, especialmente quando orquestradas por plataformas como o Docker [5, 6].

Apesar dessas vantagens, o Docker apresenta desafios em cenários com restrições de banda. De acordo com sua documentação oficial [7], o mecanismo do Docker normalmente realiza downloads concorrentes de até três camadas de imagem. Embora essa concorrência acelere a implantação em ambientes com alta largura de banda, ela pode causar falhas e *timeouts* em ambientes restritos. Sistemas de provisionamento que tratam todas as aplicações de forma uniforme falham em considerar diferenças críticas de urgência, tolerância ao atraso e tempo de execução.

Para mitigar esse problema, o número de downloads simultâneos muitas vezes precisa ser reduzido, o que pode introduzir atrasos na fila. Esses atrasos são particularmente prejudiciais quando se trabalha com aplicações que devem ser priorizadas para execução, pois podem comprometer a responsividade e degradar o desempenho geral do sistema [8]. O mecanismo padrão de download de imagens Docker, ao não distinguir criticidade, compromete aplicações sensíveis ao tempo.

Este trabalho propõe o PrIO, um algoritmo que otimiza dinamicamente o download de camadas, aplicando uma métrica de decisão que avalia dinamicamente cada aplicação, com base em múltiplos fatores: a criticidade da aplicação, o tempo de espera acumulado, a duração estimada do download e o tempo máximo de atraso tolerado priorizando a implantação de aplicações críticas e garantindo QoS. Assim, preenchendo uma lacuna metodológica existente na orquestração de borda. De maneira específica, este trabalho busca formalizar o agendamento de downloads na granularidade das camadas de imagem Docker, tratando-o como um problema de otimização com restrições de Qualidade de Serviço QoS e de recursos. Além disso, o PrIO incorpora um mecanismo de *backfilling* que preenche dinamicamente os *slots* ociosos de download com tarefas de menor prioridade, maximizando a utilização dos recursos sem comprometer a responsividade das aplicações sensíveis ao tempo.

1.1 PROBLEMA DE PESQUISA

Docker é amplamente adotado em ambientes de computação em borda devido à sua eficiência no gerenciamento e fornecimento de aplicações conteinerizadas. Sua arquitetura leve e o suporte à modularização por camadas de imagem o tornam especialmente adequado para implantações em infraestruturas com recursos limitados. Uma imagem Docker é construída como uma pilha de camadas read-only, cada uma representando um conjunto de alterações no sistema de arquivos. Essa estrutura permite que camadas comuns entre diferentes imagens sejam compartilhadas, reduzindo o consumo de armazenamento e o tráfego de rede. Porém, para que uma aplicação seja inicializada, todas as suas camadas devem estar disponíveis localmente. Assim, o ato de provisionar uma aplicação equivale a garantir a transferência de seu conjunto de camadas ainda ausentes. Adicionalmente, o tempo de download de imagens representa 76% do tempo de inicialização do contêiner [9].

A Figura 1 ilustra a arquitetura típica de um ambiente de computação em borda

baseado em contêineres, onde dispositivos heterogêneos compartilham recursos limitados de CPU, memória e largura de banda. Nesse cenário, aplicações com diferentes níveis de prioridade são orquestradas por um sistema de provisionamento que gerencia o download e a inicialização dos contêineres de acordo com os requisitos de Qualidade de Serviço (QoS). Esse modelo visual reforça os desafios de provisionar aplicações críticas de forma eficiente em condições de infraestrutura restrita.

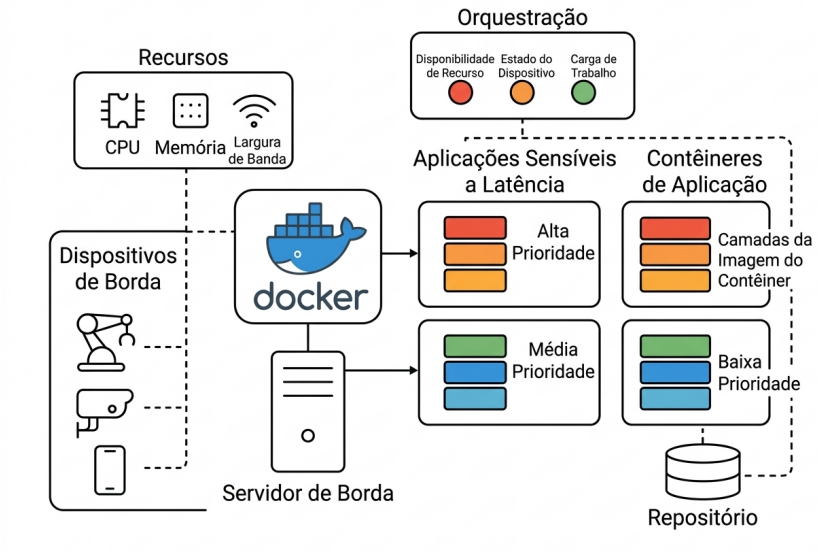

Figura 1 – Arquitetura de um ambiente de borda conteinerizado. Utilizando contêineres Docker, aplicações sensíveis à latência são provisionadas com diferentes níveis de prioridade (alta, média e baixa). Um sistema de provisionamento gerencia o agendamento dos downloads das camadas de imagem com base na criticidade da aplicação, tempo de espera e disponibilidade de recursos.

No entanto, em condições de baixa largura de banda, o Docker impõe um limite ao número de downloads simultâneos de camadas de imagem para mitigar riscos de congestionamento de rede[1]. Embora essa restrição seja configurável, ela representa um desafio em cenários com múltiplas aplicações de borda com requisitos heterogêneos de prioridade. Por padrão, o Docker não distingue entre camadas associadas a aplicações de alta ou baixa prioridade durante o escalonamento dos downloads. Logo, trata todas as camadas de forma indiferenciada, organizando os downloads em filas que não consideram a prioridade da aplicação associada.

Algumas aplicações são críticas, como sistemas de monitoramento de segurança ou controle de equipamentos industriais, exigindo provisionamento rápido para cumprir prazos operacionais; outras são de menor criticidade, como serviços de coleta de telemetria ou atualizações não urgentes, que toleram atrasos maiores. Consequentemente, aplicações críticas podem sofrer atrasos no provisionamento, bloqueadas atrás de camadas de aplicações de baixa prioridade, serviços com menor criticidade. Esses atrasos são agravados pelas

[1] https://docs.docker.com/reference/cli/docker/image/pull/

dependências entre camadas, onde um atraso no download de uma camada fundamental causa atrasos em cascata nas camadas dependentes, potencialmente resultando em perdas de prazos e violações de garantias de QoS. O desafio do escalonamento surge justamente desse limite de concorrência aliado à heterogeneidade de prioridades e à existência de dependências entre camadas.

O problema consiste em gerenciar a alocação dos recursos de download, limitados a um número máximo de transferências simultâneas, de modo a minimizar os atrasos totais, com ênfase em atender aos prazos das aplicações críticas. Isso impõe quatro grandes grupos de restrições. A primeira é o limite de downloads simultâneos, que estabelece que, a qualquer momento, o número de camadas sendo transferidas não pode ultrapassar a capacidade configurada. A segunda é a restrição de dependência: uma aplicação só pode ser considerada pronta para inicialização quando todas as suas camadas pendentes já foram baixadas. A terceira é a conformidade com QoS, que exige que o tempo total de provisionamento de cada aplicação, dado pela soma do tempo de espera acumulado com o maior tempo de download entre suas camadas pendentes, não ultrapasse seu prazo. A quarta é a conformidade com prioridade, que determina que aplicações de maior prioridade devem ser favorecidas no escalonamento, especialmente quando há risco iminente de perda de prazo.

1.2 OBJETIVOS

A principal distinção do PrIO em relação ao *Conservative Backfilling* reside em sua capacidade de combinar inteligência adaptativa com sensibilidade a prazos, tornando-o particularmente eficaz em ambientes de borda com alta contenção e recursos limitados. Enquanto o *Conservative Backfilling* adota uma postura excessivamente cautelosa ao evitar a execução de qualquer tarefa que possa interferir com trabalhos de maior prioridade, mesmo minimamente, isso frequentemente leva à subutilização de *slots* ociosos.

Em contraste, o PrIO implementa uma estratégia mais sofisticada e responsiva. Ele calcula dinamicamente uma métrica de prioridade que leva em conta o peso de criticidade da aplicação, o tempo de espera acumulado, a duração estimada do download e o tempo restante até o prazo da SLA. Isso permite ao PrIO identificar janelas seguras para o agendamento de tarefas de menor prioridade sem comprometer as críticas, promovendo assim uma utilização mais eficiente da infraestrutura. Seu mecanismo oportunista de *backfilling* não apenas melhora a vazão do sistema como um todo, mas também previne a inanição e assegura maior equidade entre as aplicações, algo que o *Conservative Backfilling*, devido à sua rigidez, não consegue alcançar com a mesma eficácia.

As principais contribuições deste trabalho podem ser listadas a seguir da seguinte forma:

- O design do PrIO, um algoritmo de provisionamento com consciência de prioridade e sensível a prazos, voltado para ambientes conteinerizados de borda;
- A formalização do problema de agendamento das camadas de contêineres como um modelo de otimização com restrições de QoS e de recursos;
- A integração de um mecanismo dinâmico de *backfilling* para utilizar oportunamente *slots* ociosos de download sem comprometer tarefas de alta prioridade;
- Uma avaliação abrangente demonstrando que o PrIO supera consistentemente estratégias de última geração sob diferentes níveis de carga da infraestrutura.

2 REFERENCIAL TEÓRICO E TRABALHOS RELACIONADOS

2.1 REFERENCIAL TEÓRICO

A crescente complexidade e diversidade das aplicações em ambientes de borda têm introduzido novos desafios para os mecanismos de provisionamento. As aplicações implantadas na borda da rede variam amplamente em termos de urgência, requisitos de recursos e sensibilidade à latência, exigindo um tratamento diferenciado durante o processo de provisionamento [10]. Por exemplo, aplicações de missão crítica, como monitoramento industrial, coordenação de veículos autônomos ou sistemas de resposta a emergências, exigem implantação rápida, atualizações frequentes e alta confiabilidade. Em contraste, serviços de segundo plano ou tarefas analíticas não sensíveis ao tempo podem tolerar tempos de espera maiores e menor precedência no provisionamento [11].

Infraestruturas de computação em borda são tipicamente limitadas em termos de poder computacional, memória e largura de banda. Essas limitações dificultam o tratamento uniforme de todas as aplicações, especialmente quando cargas de trabalho de alta prioridade precisam coexistir com tarefas de menor prioridade. Atrasos no provisionamento de aplicações críticas podem comprometer a reatividade do sistema e degradar a Qualidade de Serviço (QoS) geral. Portanto, garantir um provisionamento eficiente e sensível à prioridade é essencial para manter o desempenho e a confiabilidade de serviços sensíveis à latência [12].

Figura 2 – Classificação de aplicações de borda por criticidade e requisitos de provisionamento. Serviços críticos demandam baixa latência e alta prioridade de agendamento, enquanto aplicações do tipo *best-effort* toleram atrasos e podem ser adiadas em condições restritas.

A heterogeneidade das cargas de trabalho em borda impõe desafios significativos para a alocação de recursos e o provisionamento. Como ilustrado na Figura 2, aplicações em borda podem ser amplamente categorizadas segundo sua criticidade e tolerância à latência. Aplicações críticas, como veículos autônomos ou sistemas de saúde, exigem garantias rígidas de QoS, demandando provisionamento imediato e execução contínua. Em contraste, aplicações interativas como videoconferência ou streaming toleram atrasos moderados,

enquanto serviços do tipo *best-effort*, como backup de dados e análises periódicas, podem ser agendados oportunisticamente em períodos de menor contenção. Essa diversidade reforça a necessidade de mecanismos inteligentes de orquestração que reconheçam a prioridade das aplicações e gerenciem adaptativamente os recursos restritos da borda para manter responsividade e equidade.

Para enfrentar esses desafios, arquiteturas baseadas em contêineres têm sido amplamente adotadas [13]. Contêineres permitem implantações modulares e leves ao encapsular código, dependências e ambientes de execução em unidades isoladas. Ferramentas como o Docker permitem a orquestração flexível desses contêineres em dispositivos heterogêneos, promovendo escalabilidade e consistência [14]. No entanto, mesmo com esses avanços, o comportamento padrão do Docker, como o download concorrente de até três camadas de imagem, pode introduzir gargalos em ambientes com largura de banda limitada. Essas limitações tornam-se críticas quando múltiplas aplicações competem pelo provisionamento sob diferentes níveis de prioridade, uma vez que serviços de alta prioridade podem ser atrasados devido à contenção com cargas de trabalho de menor prioridade [15].

À medida que aplicações em borda dependem cada vez mais de processamento de dados em tempo real e respostas rápidas, cresce a necessidade por estratégias de provisionamento que considerem a prioridade das aplicações, a disponibilidade de recursos e os prazos de execução. Por exemplo, os sistemas devem ser capazes de priorizar o download e a inicialização de contêineres para aplicações críticas, ao mesmo tempo em que gerenciam de forma inteligente a fila e a execução de tarefas menos urgentes. Vários estudos recentes demonstraram os benefícios de estratégias diferenciadas na melhoria da eficiência energética, na redução de latência e na alocação otimizada de recursos em redes de borda.

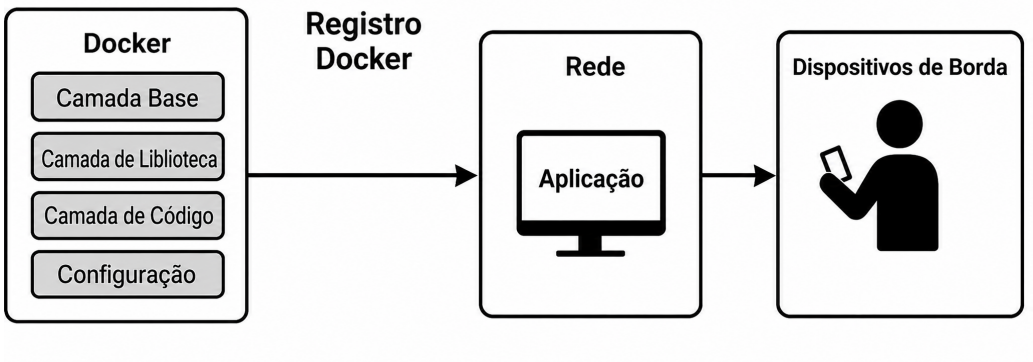

Figura 3 – Arquitetura ponta-a-ponta para provisionamento de aplicações conteinerizadas em ambientes de borda. Imagens Docker são puxadas de um registro, transferidas pela rede para dispositivos de borda, implantadas e executadas para atender usuários móveis.

O provisionamento de aplicações conteinerizadas em ambientes de computação em

borda envolve múltiplos componentes operando em coordenação. Conforme mostrado na Figura 3, as aplicações são empacotadas em contêineres usando Docker, que as organiza em camadas modulares (base, bibliotecas, código e configuração). Essas imagens são armazenadas em um repositório Docker e transferidas pela rede até os dispositivos de borda. Uma vez recebidas, os contêineres são instanciados localmente, oferecendo serviços em tempo real com latência reduzida. Essa arquitetura permite a implantação escalável e flexível de aplicações próximas aos usuários finais, como clientes móveis, minimizando a dependência de infraestrutura centralizada em nuvem e otimizando os tempos de resposta.

Apesar dos avanços, persistem desafios. A priorização deve ser equilibrada com a equidade, de modo a evitar a inanição de tarefas de menor prioridade, e os sistemas devem se adaptar dinamicamente às mudanças nas cargas de trabalho, mobilidade dos usuários e condições da rede. Além disso, a implementação dessas estratégias precisa ser computacionalmente eficiente para não introduzir sobrecarga adicional em ambientes já restritos em recursos. Em resposta a esses desafios, este trabalho propõe o *PrIO*, um algoritmo de otimização de provisionamento que prioriza o download das camadas de contêiner com base na criticidade da aplicação, tempo de espera, duração estimada e atraso tolerado. Ao integrar um mecanismo de *backfilling* que utiliza oportunisticamente *slots* ociosos de banda, o PrIO melhora a responsividade das aplicações críticas ao mesmo tempo em que mantém o uso eficiente dos recursos em todo o sistema.

2.2 TRABALHOS RELACIONADOS

O provisionamento de aplicações em ambientes de borda sob restrições de QoS tem sido objeto de crescente interesse nos últimos anos [16] [17] [18] [19] [20] [21]. A necessidade por escalonamento inteligente de recursos cresce à medida que as infraestruturas de borda passam a ser responsáveis por tarefas sensíveis à latência. Diversos estudos propuseram mecanismos que incorporam consciência de prioridade, restrições de SLA e disponibilidade de recursos para garantir a entrega eficiente de serviços. No entanto, a maioria das soluções existentes concentra-se na alocação de tarefas ou de CPU, com poucos esforços voltados à granularidade do provisionamento de imagens de contêineres — um gargalo crítico em sistemas baseados em Docker [22].

Wang et al. [23] introduziram o DYVERSE, um framework para escalonamento vertical dinâmico em ambientes multi-inquilino de computação em borda. O sistema ajusta os limites de recursos dos contêineres em tempo de execução usando múltiplos modelos de prioridade, incluindo políticas conscientes da carga de trabalho e do sistema. O DYVERSE reduz significativamente as taxas de violação de Objetivos de Nível de Serviço (SLO) em aplicações de detecção de objetos em tempo real e serviços interativos. No entanto, apesar de sua eficácia em ambientes dinâmicos, o DYVERSE se concentra na escalabilidade de CPU e memória, não abordando gargalos de download ou o escalonamento das camadas de imagens de contêineres sob condições de largura de banda restrita.

Madej et al. [24] propuseram diversas heurísticas de escalonamento para conciliar justiça e priorização de tarefas em cenários de borda. A abordagem avalia três estratégias principais: apenas justiça, apenas prioridade e um modelo híbrido que combina ambos. O modelo híbrido, em particular, alcança bons resultados no balanceamento de carga ao mesmo tempo em que respeita os requisitos de prioridade. No entanto, sua análise está centrada no compartilhamento de tempo de CPU e omite etapas-chave do provisionamento de aplicações, como a entrega de imagens pela rede. Isso torna o modelo menos aplicável a sistemas onde a contenção no download é uma limitação significativa, como em ambientes baseados em Docker.

Samanta et al. [25] desenvolveram o ASAP, um sistema de provisionamento consciente de SLA para microsserviços em redes edge-IoT. O ASAP avalia continuamente os requisitos dos microsserviços e os distribui para nós de borda de acordo com as restrições de SLA e sensibilidade à latência. Seu escalonador adaptativo reduz eficazmente os atrasos de microsserviços em cenários com dependências encadeadas entre serviços. No entanto, o ASAP opera em um nível de abstração de serviço e não trata decisões no nível de contêiner, como escalonamento de camadas de imagem ou gerenciamento de *slots* de download. Além disso, a falta de integração com o modelo de provisionamento de imagens do Docker limita sua aplicabilidade em plataformas de borda centradas em contêineres.

Lindner [26] introduziu o PBMECO, um algoritmo de *offloading* de tarefas baseado em prioridade para ambientes de borda móvel em nuvem. O método utiliza um modelo dinâmico de prioridade que considera urgência da tarefa, proximidade e disponibilidade de recursos ao tomar decisões de *offloading*. Embora o PBMECO seja particularmente útil em contextos móveis, seu foco está na alocação de computação e não aborda as complexidades do tempo de inicialização de aplicações relacionadas ao download das camadas de imagem. Assim, o PBMECO não considera o escalonamento de downloads, o reaproveitamento de *slots* ociosos, nem a granularidade de múltiplas camadas, que são cruciais para a responsividade em tempo real em infraestruturas conteinerizadas.

Al-Dulaimy et al. [27] apresentaram um escalonador com pesos de prioridade para serviços conteinerizados em sistemas de borda em tempo real. Seu modelo atribui pesos a cada tarefa com base na criticidade e provisiona recursos de acordo. Os resultados de simulação mostram que o método melhora os tempos médios de resposta e a equidade entre tarefas. No entanto, o modelo assume que os contêineres já estão disponíveis no nó, não modelando o processo de download da imagem, um fator frequentemente relevante na latência de inicialização (*cold start*) em implantações baseadas em Docker. Além disso, janelas ociosas de recursos não são exploradas por cargas de menor prioridade.

Utilizando um framework de otimização de Lyapunov [28], Abouaomar et al. [29] exploraram o provisionamento de recursos para aplicações sensíveis à latência na borda. Seu algoritmo ajusta a alocação de recursos dinamicamente, considerando padrões de chegada de tarefas e o estado do sistema. O trabalho é eficaz na gestão de QoS sob condições variáveis

de carga e heterogeneidade dos nós. No entanto, foca no provisionamento de máquinas virtuais (VMs) e carece de aplicabilidade direta a contêineres Docker e ao escalonamento de camadas de imagem, onde as restrições são mais granulares e a contenção de largura de banda tem papel mais relevante.

O trabalho de Mohammadi [30], apresenta uma abordagem avançada para a gestão de recursos em ambientes de computação em borda, fundamentada em Aprendizado por Reforço (*Reinforcement Learning*). Os autores partem da premissa de que a proliferação de dispositivos IoT exige processamento em tempo real, mas a natureza limitada e heterogênea dos nós de borda torna a alocação de recursos um problema NP-hard. Como solução, a pesquisa propõe a formulação do problema como um Processo de Decisão de Markov (MDP), resolvido por meio do algoritmo Q-learning, que permite ao sistema aprender políticas de alocação otimizadas com base no *feedback* contínuo do ambiente

Tabela 1 – Comparação de Trabalhos Relacionados com o PrIO

Abordagem	**Ciente de Prioridade**	**Backfilling**	**Consciente de SLA**	**Nível de Camada**
DYVERSE [23]	✓	✗	✓	✗
Justiça Híbrida [24]	✓	✗	✗	✗
ASAP [25]	✓	✗	✓	✗
PBMECO [26]	✓	✗	✗	✗
Al-Dulaimy et al. [27]	✓	✗	✓	✗
Abouaomar et al. [29]	✓	✗	✓	✗
Mohammadi et al. [30]	✓	✗	✓	✗
PrIO (proposto)	✓	✓	✓	✓

A Tabela 1 resume as principais contribuições desses trabalhos e os compara com nossa proposta, **PrIO**. Embora muitos desses algoritmos tratem de prioridade de tarefas ou aderência a SLAs em nível elevado, tipicamente não abordam os gargalos de provisionamento introduzidos pelas camadas de imagens Docker, nem implementam estratégias de *backfilling* para uso eficiente de *slots* ociosos. O PrIO preenche essa lacuna metodológica ao introduzir uma estratégia dinâmica de priorização orientada por SLA e um mecanismo de *backfilling* voltado para o gerenciamento de downloads em ambientes de borda com largura de banda restrita. Isso o torna especialmente adequado para serviços em tempo real e sensíveis à latência, nos quais os atrasos de inicialização são uma preocupação crítica.

3 PRIO

Este trabalho apresenta o **PrIO**, um novo algoritmo projetado para otimizar o escalonamento de downloads de camadas de imagens de contêineres para aplicações de borda com requisitos de provisionamento heterogêneos. O PrIO é voltado para cenários nos quais múltiplas aplicações conteinerizadas precisam ser provisionadas simultaneamente em uma infraestrutura de borda com largura de banda limitada, assegurando que aplicações de alta prioridade sejam agendadas de forma oportuna, sem desperdiçar recursos em períodos ociosos.

O PrIO é proposto a ser implantado como um componente intermediário entre a camada de aplicação e a infraestrutura. Desta maneira realiza seu escalonamento nos *Edge nodes*. Este posicionamento melhora a capacidade de respostas rápidas às aplicações de borda onde seus níveis de criticidade são predefinidos.

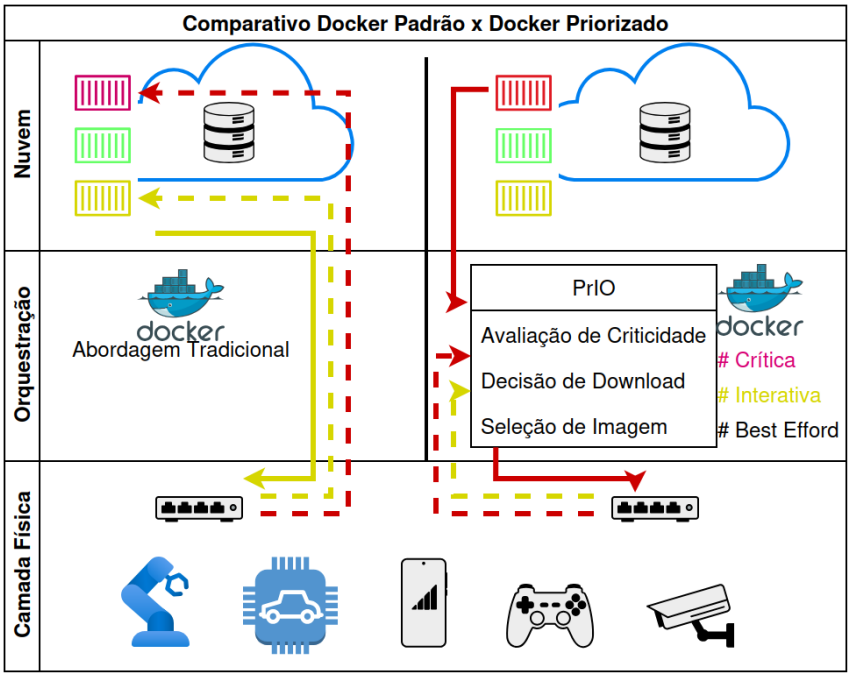

Figura 4 – Diferença entre a abordagem de download de imagens de containers de forma tradicional comparada a determinação de prioridade com o PrIO.

Métodos tradicionais, como o *Conservative Backfilling*, priorizam a equidade ao permitir que tarefas de menor prioridade utilizem recursos ociosos, desde que não atrasem tarefas de maior prioridade. No entanto, esses métodos carecem de responsividade dinâmica frente a flutuações na carga de trabalho e geralmente não incorporam restrições de QoS. Em contraste, o PrIO introduz uma estratégia sensível a prioridades e prazos, que se adapta continuamente ao estado do sistema, priorizando aplicações com base em sua urgência, tempo de espera acumulado e duração estimada de provisionamento. A métrica central de decisão no PrIO avalia cada aplicação pendente a_i da seguinte forma:

$$\text{Prioridade}(a_i) = W(a_i) \times \frac{T_c(a_i)}{T_a(a_i) + \sum_{l \in d(a_i)} T(l)}, \quad (1)$$

onde:

- $W(a_i)$: Peso de prioridade da aplicação a_i,
- $T_c(a_i)$: Tempo acumulado de espera da aplicação,
- $T_a(a_i)$: Prazo estabelecido pelas restrições de QoS,
- $\sum_{l \in d(a_i)} T(l)$: Tempo estimado para download das camadas pendentes.

Essa formulação garante que aplicações com prazos mais apertados e maior criticidade sejam priorizadas, enquanto o mecanismo de *backfilling* incorporado mitiga a inanição de tarefas de baixa prioridade.

A complexidade no pior caso do PrIO por decisão de escalonamento é $O(m \cdot n)$, onde m é o número de aplicações pendentes e n é o número médio de camadas por aplicação. Essa complexidade decorre da avaliação de todas as aplicações para o cálculo da função de prioridade e da identificação da melhor opção de *backfilling*. Embora essa complexidade seja aceitável para cargas típicas em ambientes de borda, o algoritmo continua sendo viável devido ao tamanho moderado das filas nesses contextos.

A Figura 5 ilustra a arquitetura proposta. A camada física consiste em dispositivos reais e sensores que geram dados continuamente. A camada de borda executa aplicações conteinerizadas com diferentes níveis de sensibilidade à latência e criticidade. Um sistema de provisionamento intermediário monitora o uso de recursos e as filas de download em tempo real, orquestrando quais camadas de imagem serão recuperadas a seguir. Ele utiliza informações de prioridade e tempo para assegurar o provisionamento oportuno de tarefas críticas, ao mesmo tempo em que preenche *slots* ociosos com operações secundárias quando seguro.

O PrIO apresenta diversas propriedades desejáveis que o tornam adequado para ambientes de borda dinâmicos e com recursos limitados. Ele é sensível a QoS, garantindo que as aplicações sejam escalonadas conforme seus prazos de provisionamento e pesos de prioridade. O Algoritmo 1 promove equidade por meio de um mecanismo de *backfilling* que evita a inanição de tarefas de baixa prioridade, permitindo seu progresso sempre que há recursos ociosos. Além disso, ele é adaptável, respondendo dinamicamente a variações na intensidade da carga de trabalho e nas condições de rede. O PrIO também melhora a eficiência do uso de recursos ao utilizar prontamente *slots* de download disponíveis, reduzindo a latência geral de provisionamento e aumentando o *throughput* do sistema.

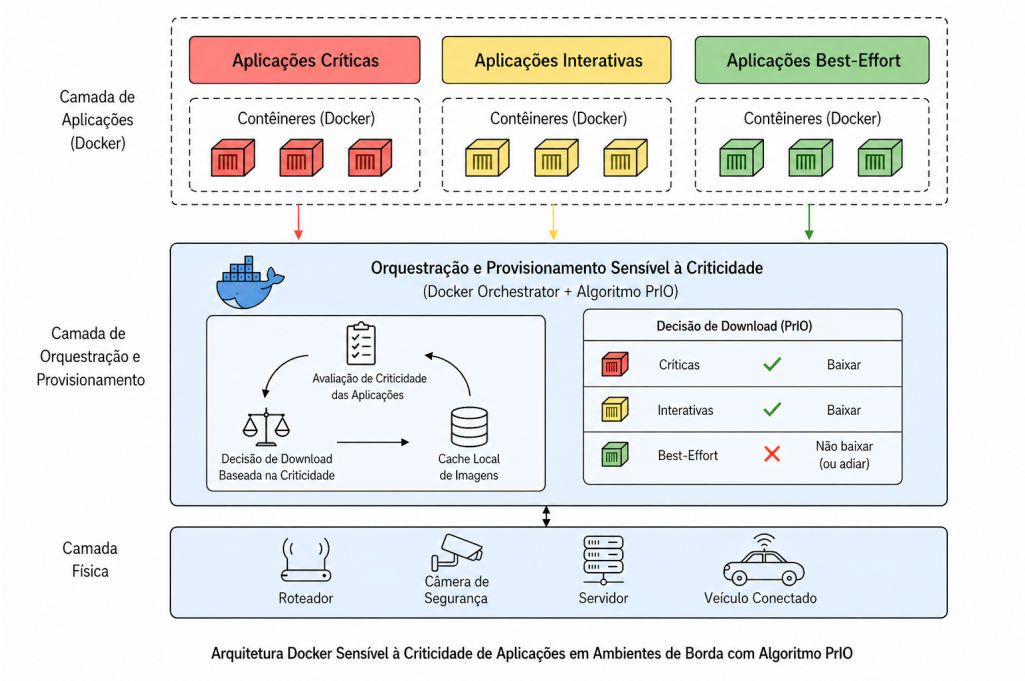

Figura 5 – Provisionamento sensível a prioridade de aplicações em borda. Aplicações de alta prioridade são agendadas e provisionadas antes de tarefas não críticas, com base na disponibilidade de recursos e nos requisitos de cada aplicação. O sistema de provisionamento gerencia dinamicamente os downloads e execuções de contêineres usando informações em tempo real do ambiente de borda.

Algorithm 1 Algoritmo PrIO

function PRIO(D, A_w)
  Entrada: D (downloads ativos), A_w (aplicações pendentes)
  Saída: Camada selecionada para download
  $a_{\text{next}} \leftarrow \arg\max_{a \in A_w} W(a) \times \frac{T_c(a)}{T_a(a) + \sum_{l \in d(a)} T(l)}$
  if a_{next} possui camadas ativas em D **then**
    $prov\_time \leftarrow \max\left(\sum_{l \in d(a_{\text{next}})} T(l), \quad T(l) \in D \cap d(a_{\text{next}})\right)$
    $T_{\text{remaining}} \leftarrow T_a(a_{\text{next}}) - T_c(a_{\text{next}}) - prov\_time$
    $b_{\text{best}} \leftarrow 0$, $w_{\text{best}} \leftarrow 0$, $l_{\text{best}} \leftarrow \arg\min_{l \in d(a_{\text{next}})} T(l)$
    for $a \in A_w \setminus \{a_{\text{next}}\}$ (ordenadas por $W(a)$ decrescente) **do**
      $l \leftarrow \arg\min_{l_i \in d(a)}$
      $T_{\text{effective}} \leftarrow T(l) + \frac{T_c(a)}{T_a(a)} \times \frac{W(a)}{W(a_{\text{next}})}$
      if $T_{\text{remaining}} - T_{\text{effective}} > b_{\text{best}}$ & $w_{\text{best}} \leq W(a)$ **then**
        $b_{\text{best}} \leftarrow T_{\text{remaining}} - T_{\text{effective}}$
        $l_{\text{best}} \leftarrow l$
        $w_{\text{best}} \leftarrow W(a)$
      end if
    end for
    return l_{best}
  end if
  return $first(d(a_{\text{next}}))$
end function

Como apresentado no Algoritmo 1, o PrIO decide qual camada baixar a seguir quando um *slot* de download se torna disponível. Para tal, o PrIO executa as seguintes

ações:

- Recebe como entrada o conjunto atual de downloads ativos (D) e a fila de aplicações pendentes A_w;
- Inicia computando uma pontuação de prioridade para cada aplicação, considerando seu peso $W(a)$, tempo de espera acumulado $T_c(a)$, prazo $T_a(a)$ e duração estimada de download de suas camadas necessárias. A aplicação com maior prioridade, a_{next}, é então selecionada;
- Se a aplicação selecionada já possui downloads ativos em andamento, o PrIO tenta preencher oportunamente qualquer largura de banda restante escalonando uma tarefa de baixa prioridade por meio de *backfilling*;
- As demais aplicações são avaliadas em ordem decrescente de prioridade, buscando o candidato cuja camada tenha tempo de download efetivo ($T_{\text{effective}}$) que caiba dentro da janela restante de provisionamento. O algoritmo retorna a camada desse melhor candidato;
- Caso não encontre aplicação com camada que atenda o passo anterior, prossegue com o download da primeira camada de a_{next}.

O PrIO fornece uma estratégia eficaz de orquestração para ambientes de borda com cargas de trabalho heterogêneas e requisitos rigorosos de QoS. Ele equilibra a responsividade para aplicações críticas com a utilização eficiente de recursos limitados.

3.0.1 PARÂMETROS E VARIÁVEIS DO PROBLEMA

A formulação resultante caracteriza um problema de otimização combinatória, representada pela equação 2, com restrições de recursos, precedências e prazos. A função objetivo é expressa como o problema de redução do tempo de provisionamento para as aplicações onde D representa o subconjunto de camadas atualmente em processo de download.

$$\min_{D \subseteq L} \sum_{a_k \in A} \left(T_c(a_k) + \max_{l_i \in d(a_k)} T(l_i) \right) \tag{2}$$

- $T(l_i)$: Tempo estimado de download da camada l_i;
- $d(a_k)$: Conjunto de camadas de imagem requeridas por a_k que ainda não foram baixadas;
- $T_a(a_k)$: Prazo acordado de provisionamento da aplicação a_k, conforme suas restrições de QoS;
- $T_c(a_k)$: Tempo total de espera acumulado da aplicação a_k.

O gerenciamento de download de imagens pelo escalonador opera determinando, em cada momento, quais camadas elencar para download respeitando essas restrições, de forma a minimizar o atraso total das aplicações. Esse objetivo pode ser entendido como uma função que pondera os tempos de espera acumulados e os tempos restantes de download, dando maior peso às aplicações críticas.

3.1 EXEMPLO PRÁTICO

Para ilustrar de forma prática como o PrIO determina qual a melhor aplicação atender, descreve-se um cenário hipotético onde um servidor de borda trabalha para atender três aplicações diferentes que solicitam provisionamento ao mesmo tempo. A largura de banda é restrita a três *slots* de download simultâneos definidos por $C = 3$. Para as aplicações, define-se:

- Aplicação A, contempla um veículo autônomo com alta prioridade. Logo, seu peso é expresso por $W_a = 5$ e seu prazo de entrega por $T_a = 7$ segundos.
- Aplicação B, monitoramento de tráfego com média prioridade. Seu peso é definido por $W_c = 3$ e seu prazo de entrega por $T_a = 10$ segundos.
- Aplicação C, voltada a backup de log diário com baixa prioridade. O peso atribuído é $W_c = 1$, seu prazo de entrega por $T_a = 15$ segundos.

Considera-se que todas aplicações esperam um segundo na fila ($T_c = 1$) e todas necessitam baixar imagens que totalizam um tempo (T) de 4 segundos, $\sum T(l) = 4$. Inicialmente, o PrIO calcula a pontuação de prioridade de cada uma das aplicações expressa pela equação 1.

Aplicando os valores de peso de prioridade, prazo de entrega e tempo total de download para as aplicações, obtemos como resultado 0.45, 0.21 e 0.05 para A, B e C, respectivamente. O PrIO seleciona a aplicação A para ocupar o primeiro *slot* de download, pois essa possui a maior pontuação de prioridade.

Posteriormente, ocorre a verificação de *slots* ociosos para preenchimento oportuno, *Backfilling*. Nesta etapa, o PrIO identifica uma janela segura, calcula o tempo restante de aplicação A e procura qual, dentre as demais aplicações menos prioritárias, cabe se encaixam na janela de tempo da aplicação A. Se o prazo para download da aplicação mais urgente é de 7 segundos e seu tempo total de download é de 4 segundos, existe uma janela de tempo segura. O algoritmo verifica as aplicações B e C. É então selecionada a aplicação C por ter o maior prazo e o menor peso, assim, com um custo de interferência menor que aplicação B. Portanto, a Aplicação C é considerada a opção mais segura para preencher um *slot* ocioso sem comprometer a prioridade ou colocar em risco o prazo rigoroso da Aplicação A.

4 EXPERIMENTOS E DISCUSSÃO

Esta seção avalia o desempenho do PrIO, com foco em três métricas principais: tempo de provisionamento, tempo total de provisionamento e tempo de espera. Comparamos o PrIO com três estratégias de base: FIFO, uma abordagem naive por prioridade e o *Conservative Backfilling*.

4.1 CONFIGURAÇÃO EXPERIMENTAL

Todas as simulações foram conduzidas com o EdgeSimPy [31], um simulador de eventos discretos projetado para avaliar estratégias de escalonamento em ambientes realistas de computação em borda. Para configuração das variáveis aleatórias no experimento, utilizou-se o mesmo *seed*, valor de inicialização randômico. Desta forma, todos os experimentos são caracterizados por utilizar o mesmo *dataset*, topologia e condições. Esta abordagem foi adotada para evitar que alterações no comportamento dos elementos da simulação interferissem no resultado do experimento.

A topologia de rede consiste em 100 estações base, 50 servidores de borda e 100 switches, interconectados em uma grade hexagonal 10×10. Cada servidor de borda suporta provisionamento de contêineres baseados em Docker com canal de download limitado a até três camadas simultâneas (configurável). Para o uso da rede, foram utilizadas 3 configurações distintas, 25%, 50% e 75% de taxa de utilização da rede onde tais valores são referentes a quantidade aplicações 65, 130 e 200, respectivamente. Não foi considerado a utilização total, 100%, pois pode ocasionar uma condição de saturação total ou congestionamento crítico, onde as filas tendem ao infinito e as métricas de latência tornam-se imprevisíveis. A Tabela 2 apresenta a configuração utilizada.

Tabela 2 – Configuração do cenário em diferentes níveis de utilização.

	Taxa de Utilização		
	25%	**50%**	**75%**
Estações Base	100	100	100
Servidores de Borda	50	50	50
Switches	100	100	100

A simulação foi realizada com cargas de trabalho em diferentes níveis de sensibilidade à latência e prioridade. As aplicações são classificadas em três categorias — alta, média e baixa prioridade — de acordo com seus requisitos de SLA, conforme a Tabela 3.

Tabela 3 – Aplicações, níveis de SLA e parâmetros de simulação.

Tipo de App	**Prioridade**		
SLA	Alta	Média	Baixa
W	5	3	1
T_a	7	10	15

Cada aplicação é mapeada para uma imagem de contêiner composta por 2 a 5 camadas, com tempo de download entre 1 e 5 segundos, simulando variabilidade real de rede e tamanho de imagem.

Avaliam-se três métricas: **Tempo de Provisionamento** (tempo entre submissão e finalização do download de todas as camadas), **Tempo Total de Provisionamento** (soma do tempo para todas as aplicações) e **Tempo de Espera** (tempo até o início do primeiro download da aplicação).

Para avaliar a eficácia do algoritmo **PrIO**, comparamos com três estratégias base:

- **FIFO**: ordem de chegada, sem consideração de prioridade.
- **Conservative Backfilling (CB)**: permite execução de tarefas de baixa prioridade somente se não houver risco de interferência com aplicações críticas.
- **Prioridade naive**: escalonamento estritamente por prioridade, sem *backfilling*.

Embora frameworks como ASAP [25] e PBMECO [26] tenham sido propostos para alocação de microsserviços, eles não atuam no nível de provisionamento de camadas de imagem Docker, por isso focamos em estratégias de escalonamento para a comparação. Os níveis de SLA e parâmetros de QoS estão descritos nas Tabelas 2 e 3.

4.2 TEMPO DE PROVISIONAMENTO

O tempo de provisionamento inclui o tempo de fila e o tempo de transferência de todas as camadas de uma aplicação. Como mostrado nas Figuras 6, 7 e 8, o PrIO obtém os menores tempos de provisionamento em todos os cenários.

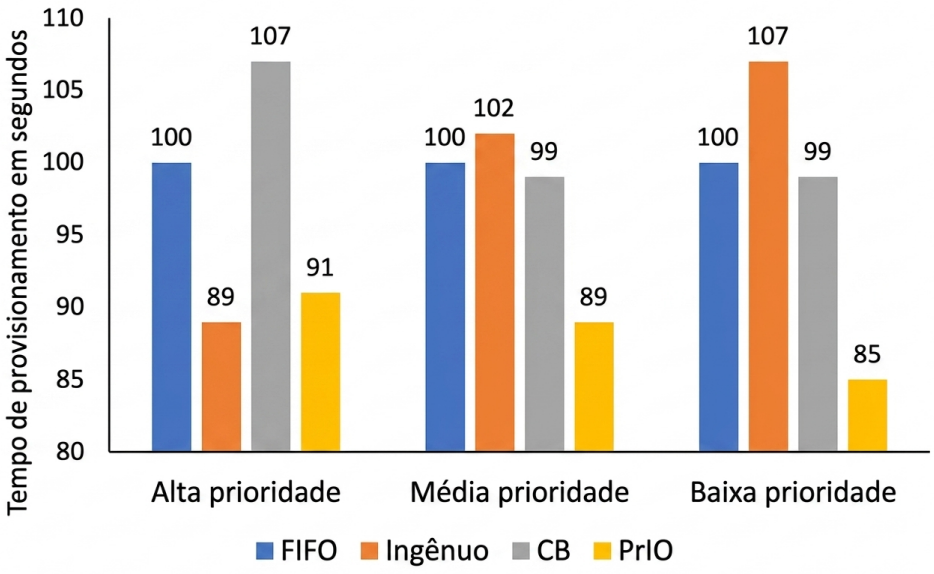

Figura 6 – Tempo de provisionamento com 25% de utilização.

No cenário de baixa carga (25%), PrIO e Prioridade naive têm desempenho similar para tarefas críticas. Mas a estratégia naive desperdiça recursos ociosos, enquanto o PrIO usa o *backfilling* para atender aplicações de baixa prioridade.

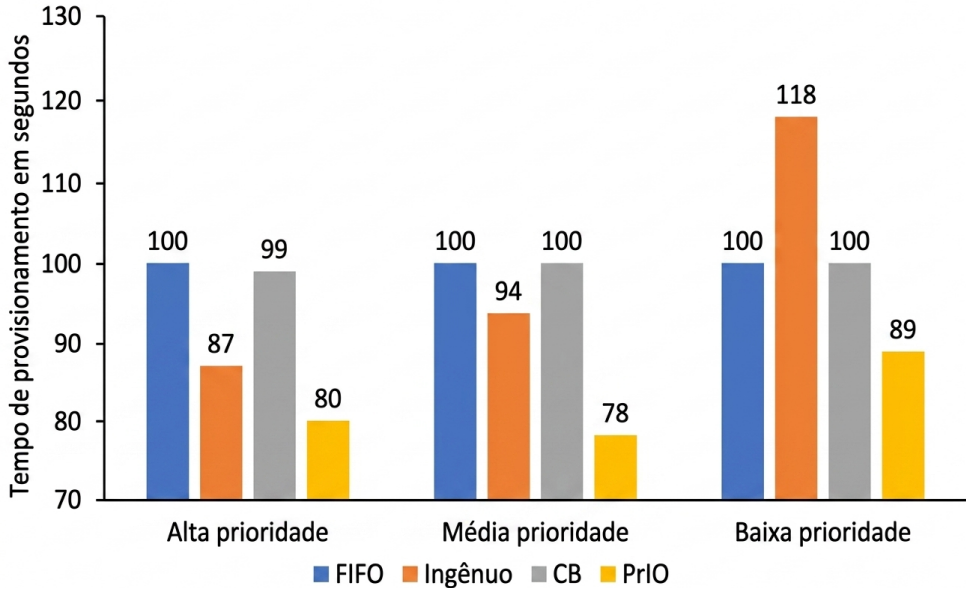

Figura 7 – Tempo de provisionamento com 50% de utilização.

Com 50% de carga, PrIO mostra vantagens mais claras, adaptando o escalonamento com base no SLA, tempo de espera e duração prevista de download. No entanto, as abordagens FIFO e *Conservative Backfilling* não apresentam variações mesmo frente as variações de prioridades.

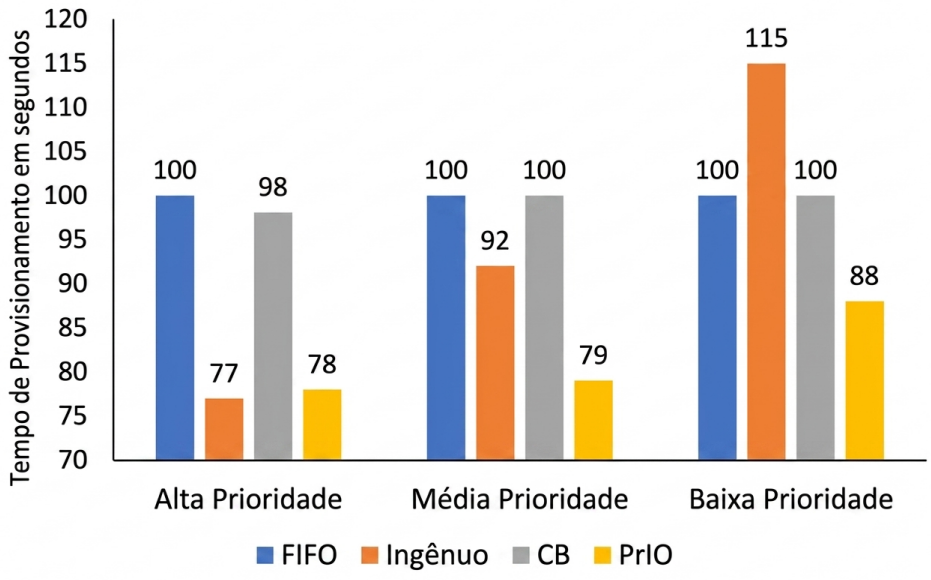

Figura 8 – Tempo de provisionamento com 75% de utilização.

Sob carga alta (75%), PrIO evita o colapso do fairness observado na Prioridade naive e a ineficiência do *Conservative Backfilling*, mantendo alto *throughput*. Em cenários de baixa utilização (25%), a abordagem Naive atinge um tempo de 89 segundos para aplicações de alta prioridade, enquanto o PrIO registra 91 segundos. Esta diferença marginal, aproximadamente 2%, decorre da ausência de overhead de cálculo de janelas de backfilling no modelo ingênuo. Contudo, ao elevar a carga para 75%, o PrIO mantém a estabilidade em 78 segundos para alta prioridade, enquanto a abordagem naive, embora similar na prioridade alta, demonstra uma redução na equidade em tarefas de baixa prioridade, elevando seu tempo para 115 segundos, contra os estáveis 88 segundos do PrIO, uma eficiência 23,4% superior do PrIO na preservação da fluidez para aplicações não críticas.

O modelo CB apresenta uma rigidez que penaliza a vazão; em 50% de carga, o tempo de provisionamento para alta prioridade no CB é de 99 segundos, comparado aos 80 segundos do PrIO. O FIFO, por sua natureza agnóstica à prioridade, mantém tempos uniformes em torno de 100 segundos, falhando em proteger aplicações sensíveis à latência em qualquer nível de estresse de rede.

4.3 TEMPO TOTAL DE PROVISIONAMENTO

As Figuras 9, 10 e 11 mostram que o PrIO supera os demais em tempo total de provisionamento. Sua abordagem combinada de prioridade e backfilling otimiza uso de *slots* e respeita SLAs.

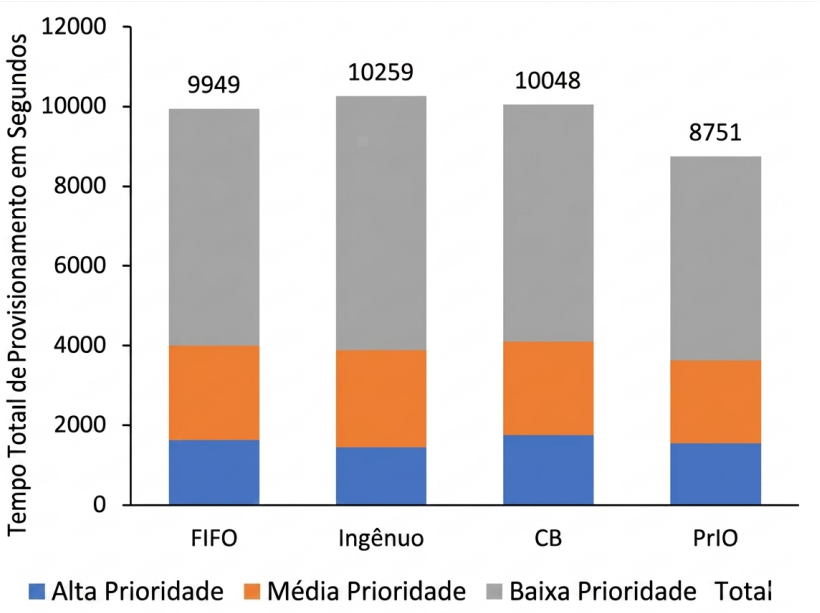

Figura 9 – Tempo total de provisionamento com 25% de utilização.

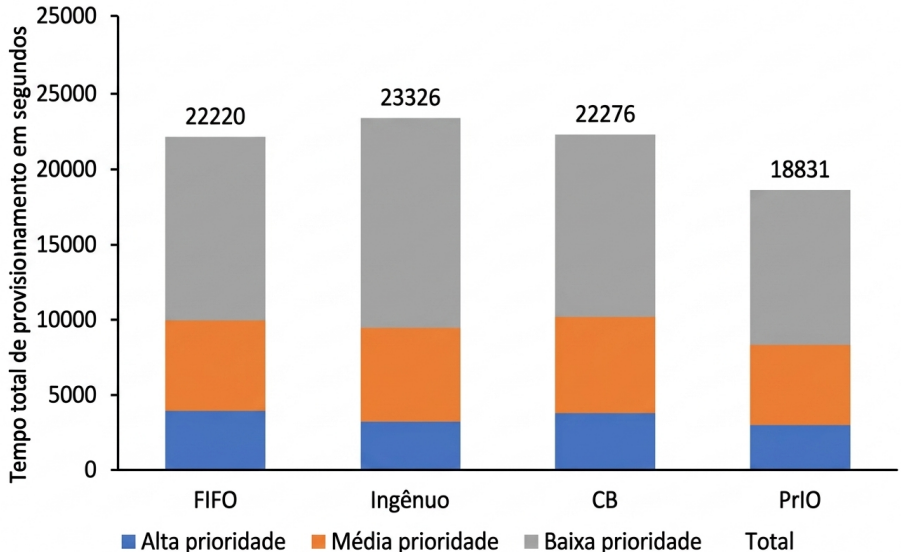

Figura 10 – Tempo total de provisionamento com 50% de utilização.

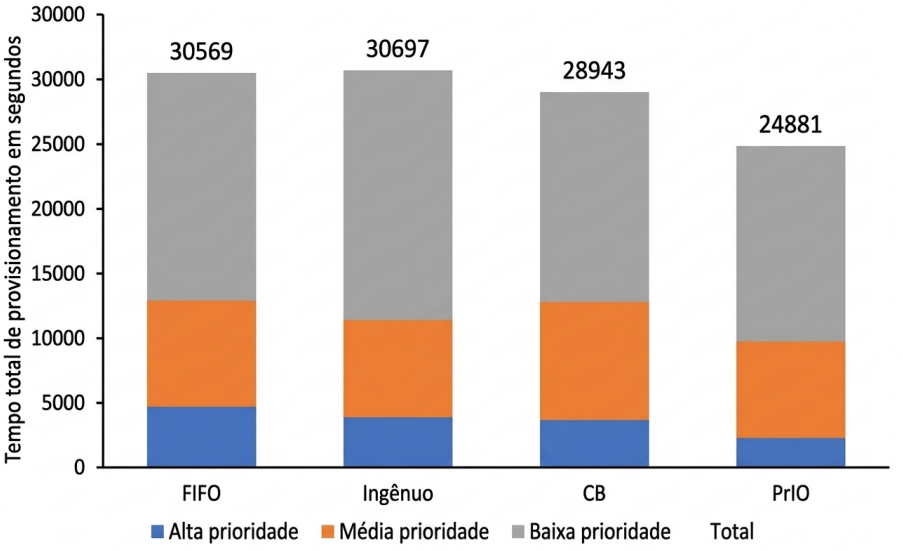

Figura 11 – Tempo total de provisionamento com 75% de utilização.

Em carga alta, o PrIO preserva o desempenho de tarefas críticas sem comprometer aplicações menos prioritárias. Análise de Vazão Acumulada: Sob carga alta (75%), o PrIO atinge um tempo total de 24.881 segundos, enquanto a abordagem naive e o FIFO ultrapassam a marca dos 30.500 segundos. Proporcionalmente, o PrIO apresenta uma melhoria de aproximadamente 18,9% na eficiência global em relação à abordagem naive. Ao observar todos os cenários de utilização de rede, PrIO apresenta o menor tempo total de provisionamento frente as demais abordagens.

4.4 TEMPO DE ESPERA

Nas Figuras 12, 13 e 14, demonstram os resultados do tempo referente ao tempo de requisição da aplicação até o início do download da primeira camada de imagem. O PrIO apresenta reduções significativas no tempo médio de espera, especialmente sob cargas moderada e alta.

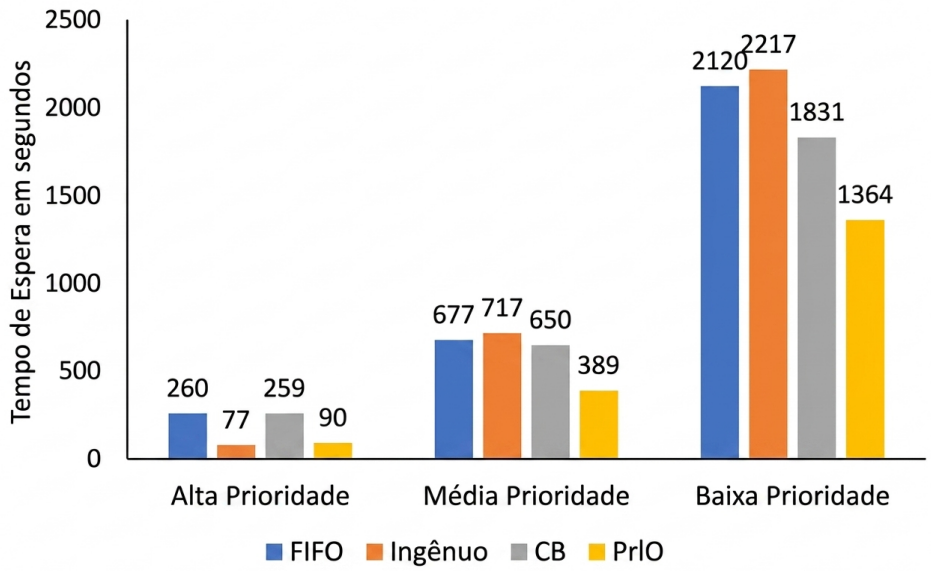

Figura 12 – Tempo de espera com 25% de utilização.

Com 25% de carga, a estratégia naive favorece as tarefas críticas, mas penaliza fortemente as demais. O PrIO usa o *backfilling* para distribuir melhor o uso da fila.

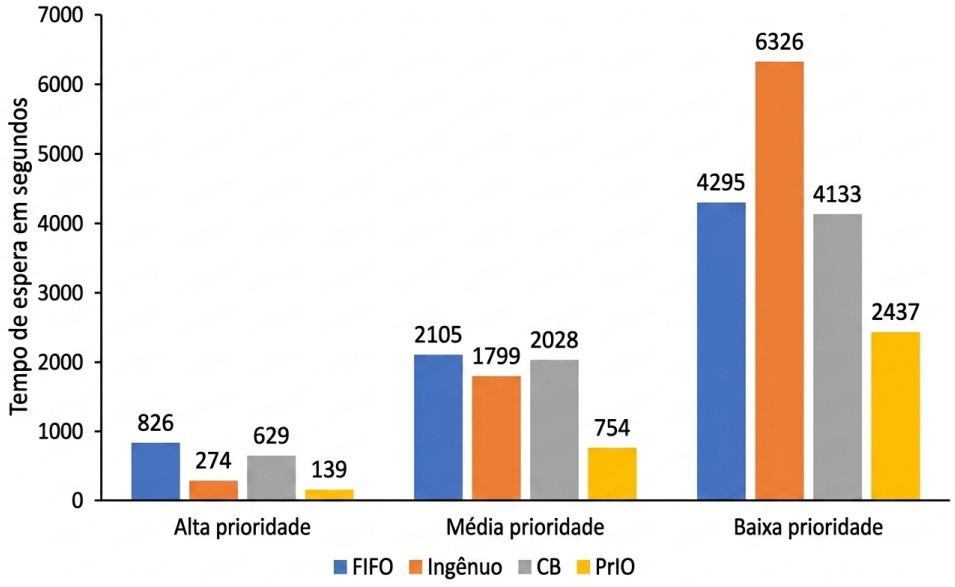

Figura 13 – Tempo de espera com 50% de utilização.

A 50%, o PrIO equilibra prioridades e ociosidade, reduzindo o tempo de espera em todos os níveis de SLA.

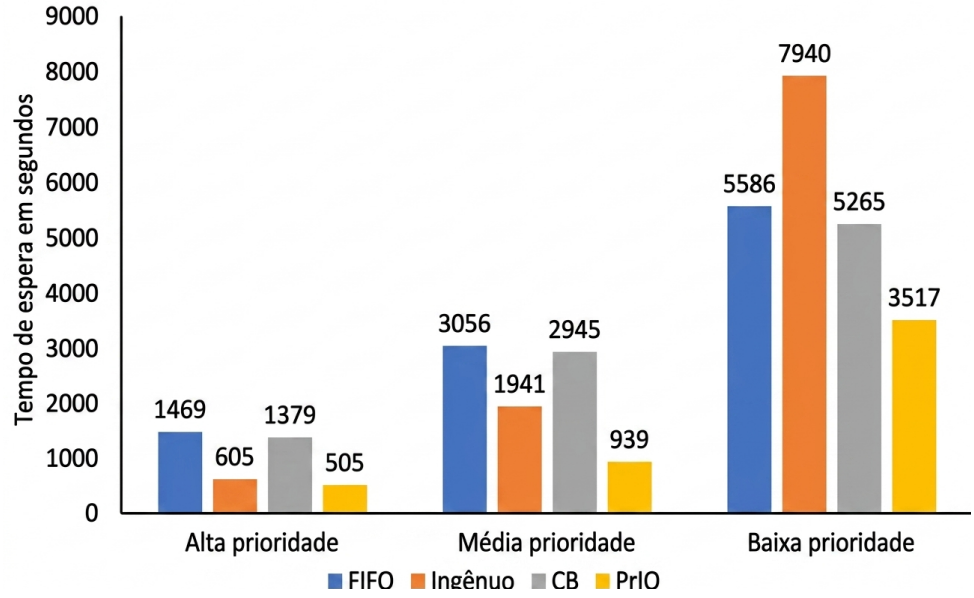

Figura 14 – Tempo de espera com 75% de utilização.

Em 75%, o PrIO mantém sua capacidade de resposta e evita a inanição de tarefas menos críticas, provando sua resiliência. Em tal cenário, o abismo estatístico acentua-se, a abordagem naive gera uma espera de 7.940 segundos para tarefas secundárias, indicando uma tendência crítica à inanição, ao passo que o PrIO reduz essa métrica para 3.517 segundos — uma redução drástica de 55,7% no atraso de espera. Na análise de 50% de utilização, figura 13, observa-se que o tempo de espera para aplicações de baixa prioridade na estratégia naive dispara para 6.326 segundos, apresentando uma aumento exponencial, enquanto o PrIO o contém em 2.437 segundos de forma mais linear e equilibrada.

4.4.1 DISCUSSÃO DOS RESULTADOS

Quanto aos resultados voltados ao de tempo de provisionamento, o PrIO manteve o tempo baixo em todos os cenários de níveis de prioridade comparado às demais abordagens. A prioridade naive também apresentou resultados similares ao PrIO, registrando 89 e 92 segundos, para aplicações de alta prioridade devido a falta de fila em níveis baixos de utilização de rede (25%). No entanto, a estratégia naive revela negligencia perante a recursos ociosos, adicionalmente, seu tempo de provisionamento aumentou significativamente em aplicações de baixa e média prioridade, independentemente da taxa de utilização da rede. Tal diferença evidencia a eficiência do *backfilling* do PrIO quando há uma grande quantidade de tarefas de baixa prioridade.

Ao elevar a carga para 75%, o diferencial técnico do PrIO fica evidente, enquanto ele mantém a estabilidade em 78 segundos para alta prioridade, a abordagem naive sofre uma degradação na equidade, elevando o tempo das tarefas de baixa prioridade para 115

segundos, o que confere ao PrIO uma eficiência 23,4% superior na preservação da fluidez para serviços não críticos.

Referente ao tempo total de provisionamento, métrica fundamental para aferir a eficiência global e a capacidade de vazão, *throughput* da infraestrutura de borda, o PrIO superou consistentemente todas as demais estratégias comparadas. A análise da vazão acumulada sob carga alta (75%) revela que o PrIO atingiu um tempo total de 24.881 segundos, valor consideravelmente inferior aos mais de 30.500 segundos registrados pelas abordagens FIFO e Prioridade naive.

Estatisticamente, isso representa uma melhoria de aproximadamente 18,9% na eficiência global do sistema em relação à estratégia naive. Essa superioridade advém da função de decisão dinâmica do PrIO, que identifica janelas seguras de tempo para o agendamento de tarefas, permitindo uma utilização mais inteligente dos slots de download disponíveis sem comprometer os requisitos rigorosos de Qualidade de Serviço das aplicações.

A terceira métrica avaliada foi o tempo de espera, referente ao tempo transcorrido desde a submissão da aplicação até o início download de sua primeira camada de imagem, reforça a capacidade do PrIO em mitigar a latência de inicialização *cold start*. Novamente, o PrIO apresenta um bom desempenho em todos os níveis de carga de trabalho e pesos de prioridade com destaque mais acentuado em aplicações de média e baixa prioridade. A estratégia naive apresentou um tempo de espera levemente inferior para 25% de utilização com aplicações de alta prioridade, pois foca exclusivamente na hierarquia sem processamento adicional. Que é justificável pela ausência de fila, nesse cenário são instanciadas 65 aplicações, não gerando grande concorrência pelo baixo uso da rede. Contudo, esta abordagem penaliza severamente aplicações de baixa prioridade que enfrentam um tempo de 2.217 segundos frente a 1.364 segundos do PrIO, evidenciando uma distribuição de recursos muito mais equitativa no algoritmo proposto.

Em cenários de estresse máximo (75% de utilização), o PrIO provou sua resiliência ao evitar o fenômeno de *starvation* de tarefas secundárias, mantendo o início do provisionamento de aplicações críticas em 505 segundos. Enquanto a abordagem naive gera um abismo estatístico ao forçar tarefas de baixa prioridade a esperarem por 7.940 segundos, o PrIO contém essa métrica em 3.517 segundos, o que representa uma redução drástica de 55,7% no atraso de espera. Esse comportamento demonstra que a sensibilidade a prazos integrada ao mecanismo de backfilling oportunista permite que o sistema mantenha a operacionalidade de serviços essenciais sem paralisar o fluxo de tarefas menos urgentes, garantindo a escalabilidade necessária para ambientes de borda dinâmicos. Sob carga de 50% e 75%, o algoritmo equilibrou a urgência das tarefas com a ociosidade da rede, reduzindo o tempo de espera em todos os níveis de SLA. Demonstrando assim, sua superioridade no início download em todos os níveis de criticidade.

Em conclusão, as simulações realizadas confirmam que a proposta do PrIO é eficaz para melhorar a eficiência, escalabilidade e confiabilidade dos sistemas de provisionamento

em borda, provando ser superior a estratégias que tratam as aplicações de forma uniforme ou puramente estática. Sua combinação de avaliação dinâmica de prioridades e *backfilling* oportunista supera os limites dos métodos tradicionais. Enquanto abordagens naives causam iniquidade e *Conservative Backfilling* deixa recursos ociosos, o PrIO mantém o equilíbrio entre responsividade e justiça.

5 CONSIDERAÇÕES FINAIS

Esta pesquisa abordou o desafio crítico do provisionamento sensível à prioridade em infraestruturas de computação em borda, um ambiente comprovadamente caracterizado pela escassez de recursos e pela volatilidade das condições de rede. O problema central reside na latência de inicialização *cold start* gerada pelo download de camadas de imagens de contêineres, embora tecnologias como o Docker facilitem a implantação modular, seu mecanismo padrão falha ao tratar de forma indiferenciada aplicações com requisitos de urgência heterogêneos. Em cenários de largura de banda restrita onde aplicações concorrem pelo provisionamento de suas imagens, essa ausência de consciência de prioridade resulta em contenções severas. Nessas condições, serviços com aplicações críticas são frequentemente bloqueados ou atrasados por tarefas secundárias, levando a violações de SLA e à degradação da Qualidade de Serviço.

Com objetivo de reduzir tais limitações, foi desenvolvido e proposto o PrIO, um algoritmo de escalonamento que atua na granularidade das camadas de imagem Docker. A solução diferenciou-se ao aplicar uma métrica de decisão multicritério de forma dinâmica que pondera o peso de criticidade da aplicação, o tempo de espera acumulado e os prazos de entrega rigorosos estabelecidos pelo SLA. Além da priorização adaptativa, o PrIO integra um mecanismo de *backfilling* oportunista, que identifica janelas seguras de tempo para preencher slots ociosos de download com tarefas de menor prioridade. Essa abordagem permite maximizar a vazão do sistema *throughput* sem comprometer a responsividade necessária para aplicações sensíveis à latência, superando a rigidez de métodos tradicionais como o *Conservative Backfilling*. A eficácia da proposta foi rigorosamente validada por meio de simulações no ambiente EdgeSimPy, abrangendo cenários com níveis de utilização de rede de até 75%.

Os resultados demonstraram a robustez do PrIO, que superou consistentemente as estratégias FIFO, Prioridade naive e *Conservative Backfilling* em todas as métricas avaliadas. Sob carga elevada (75%), o PrIO demonstrou manter a estabilidade no tempo de provisionamento de tarefas críticas em 78 segundos, enquanto evitou o colapso de equidade observado em outras abordagens. Notadamente, o algoritmo reduziu o atraso de espera de tarefas secundárias em 55,7% em comparação com a abordagem naive, provando sua capacidade de mitigar a inanição e promover a justiça sistêmica.

Em suma, dados experimentais confirmam que o PrIO representa uma contribuição metodológica significativa para a orquestração de borda, oferecendo uma melhoria de aproximadamente 18,9% na eficiência global, referente ao tempo total de provisionamento do sistema sob alta carga. O trabalho demonstra que é possível conciliar o atendimento imediato de demandas críticas com a utilização otimizada da infraestrutura. Conclui-se que o agendamento inteligente no nível de camadas é essencial para a viabilidade de serviços em tempo real, pavimentando o caminho para futuras implementações em plataformas reais de orquestração e a incorporação de modelos preditivos baseados em aprendizado de

máquina.

5.1 TRABALHOS FUTUROS

Apesar desses resultados promissores, diversas direções permanecem abertas para exploração futura. Primeiramente, a versão atual do PrIO utiliza parâmetros de SLA e pesos de prioridade (W) e prazos de SLA (Ta) estáticos, predefinidos no momento da submissão. Trabalhos futuros podem investigar mecanismos adaptativos de ajuste de prioridade que respondam a mudanças em tempo real no comportamento das aplicações, nível de urgência ou contexto do usuário, baseando-se em dados de telemetria contínua. Essa flexibilidade permitiria que o sistema redefinisse a criticidade de uma aplicação com base na urgência operacional imediata ou na degradação acumulada da Qualidade de Experiência (QoE) [25]. Adicionalmente, pode-se melhorar a configuração do Ambiente referente a escolha das aplicações de rede e implementação dos códigos em P4, eBPF e DPDK.

Outra direção que pode ser tomada é quanto ao cenário de testes e validação da solução. Embora a avaliação tenha sido feita via simulação, utilizando o simulador EdgeSimPy, futuros trabalhos devem validar o PrIO em plataformas reais como Kubernetes (K3s ou MicroK8s), considerando como a variação de atraso *jitter*, o impacto de rajadas inesperadas de requisições *bursts* e a heterogeneidade arquitetural dos processadores de borda, como ARM e x86, que influenciam diretamente a velocidade de processamento e descompactação das camadas de imagem [6]

Por fim, a incorporação de modelos preditivos baseados em técnicas de aprendizado de máquina, como previsão de carga de trabalho ou estimativa de congestionamento, pode aumentar a capacidade de resposta e a inteligência do PrIO, permitindo uma orquestração mais proativa e eficiente das tarefas de provisionamento em ambientes dinâmicos de borda. O desenvolvimento de heurísticas capazes de realizar a previsão da carga de trabalho futura ou a estimativa do congestionamento da rede permitiria que o orquestrador agisse de forma proativa em vez de reativa. Tal inteligência poderia refinar o mecanismo de *backfilling*, identificando com maior precisão janelas seguras para o download de tarefas de baixa prioridade, otimizando o uso da largura de banda antes que o congestionamento ocorra [30].

REFERÊNCIAS

1 HUA, H. et al. Edge computing with artificial intelligence: A machine learning perspective. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 55, n. 9, jan. 2023. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3555802>. Citado na página 19.

2 SHAMIM, M. Z. M. Hardware deployable edge-ai solution for prescreening of oral tongue lesions using tinyml on embedded devices. *IEEE Embedded Systems Letters*, v. 14, n. 4, p. 183–186, 2022. Citado na página 19.

3 BHASKARAN, S.; MUTHURAMAN, S. A comprehensive study of resource provisioning and optimization in edge computing. *Computers, Materials Continua*, v. 83, p. 5037–5070, 05 2025. Citado na página 19.

4 AL-MASRI, E.; OLMSTED, J. Enhancing resource provisioning across edge-based environments. In: *2020 IEEE International Conference on Big Data (Big Data)*. [S.l.: s.n.], 2020. p. 3459–3463. Citado na página 19.

5 FAVA, F. B. et al. Assessing the performance of docker in docker containers for microservice-based architectures. In: *2024 32nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. [S.l.: s.n.], 2024. p. 137–142. Citado na página 19.

6 KAISER, S. et al. Benchmarking container orchestrations for arm-powered edge computing. In: *2025 IEEE 22nd Consumer Communications Networking Conference (CCNC)*. [S.l.: s.n.], 2025. p. 1–9. Citado 2 vezes nas páginas 19 e 48.

7 INC., D. *Docker Documentation*. 2025. Accessed on: January 18, 2025. Disponível em: <https://docs.docker.com/>. Citado na página 20.

8 SOLLFRANK, M.; LOCH, F.; VOGEL-HEUSER, B. Exploring docker containers for time-sensitive applications in networked control systems. In: *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*. [S.l.: s.n.], 2019. v. 1, p. 1760–1765. Citado na página 20.

9 HARTER, T. et al. Slacker: fast distribution with lazy docker containers. In: *Proceedings of the 14th Usenix Conference on File and Storage Technologies*. USA: USENIX Association, 2016. (FAST'16), p. 181–195. ISBN 9781931971287. Citado na página 20.

10 ALEKSEEVA, D.; OMETOV, A. Hybrid edge-cloud computational offloading for xr medical applications. In: *2024 9th International Conference on Fog and Mobile Edge Computing (FMEC)*. [S.l.: s.n.], 2024. p. 63–68. Citado na página 25.

11 ARABI, S.; SABIR, E.; ELBIAZE, H. Information-centric networking meets delay tolerant networking: Beyond edge caching. In: *2018 IEEE Wireless Communications and Networking Conference (WCNC)*. [S.l.: s.n.], 2018. p. 1–6. Citado na página 25.

12 JAIN, A.; JAT, D. S. An edge computing paradigm for time-sensitive applications. In: *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*. [S.l.: s.n.], 2020. p. 798–803. Citado na página 25.

13 ABDELRAHMAN, M. S.; HUSSEIN, H.; MOHAMMED, O. A. Container-based platform for edge-computing in energy system applications. In: *2024 IEEE Power and Energy Society General Meeting (PESGM)*. [S.l.: s.n.], 2024. p. 1–5. Citado na página 26.

14 XAVIER, M. G. et al. Performance evaluation of container-based virtualization for high performance computing environments. In: *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. [S.l.: s.n.], 2013. p. 233–240. Citado na página 26.

15 B, K. S.; MAHALAKSHMI, K. Exploring large scale docker image vulnerability and storage performance analysis using high performance container-based docker environment. In: *2024 International Conference on System, Computation, Automation and Networking (ICSCAN)*. [S.l.: s.n.], 2024. p. 1–7. Citado na página 26.

16 MALIK, S. U. R. et al. A user-centric qos-aware multi-path service provisioning in mobile edge computing. *IEEE Access*, v. 9, p. 56020–56030, 2021. Citado na página 27.

17 LU, S. et al. Online service provisioning and updating in qos-aware mobile edge computing. In: *2022 18th International Conference on Mobility, Sensing and Networking (MSN)*. [S.l.: s.n.], 2022. p. 247–254. Citado na página 27.

18 SHESHADRI, K. R.; LAKSHMI, J. Qos aware faas for heterogeneous edge-cloud continuum. In: *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. [S.l.: s.n.], 2022. p. 70–80. Citado na página 27.

19 MA, X. et al. Cost-efficient resource provisioning in cloud assisted mobile edge computing. In: *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. [S.l.: s.n.], 2017. p. 1–6. Citado na página 27.

20 GUAN, S.; BOUKERCHE, A. Intelligent edge-based service provisioning using smart cloudlets, fog and mobile edges. *IEEE Network*, v. 36, n. 2, p. 139–145, 2022. Citado na página 27.

21 WANG, Y.; GU, Y.; TAO, X. Edge network slicing with statistical qos provisioning. *IEEE Wireless Communications Letters*, v. 8, n. 5, p. 1464–1467, 2019. Citado na página 27.

22 ROSA, G. et al. What quality aspects influence the adoption of docker images? *ACM Trans. Softw. Eng. Methodol.*, Association for Computing Machinery, New York, NY, USA, v. 32, n. 6, set. 2023. ISSN 1049-331X. Disponível em: <https://doi.org/10.1145/3603111>. Citado na página 27.

23 WANG, N. et al. Dyverse: Dynamic vertical scaling in multi-tenant edge environments. *arXiv preprint arXiv:1810.04608*, 2018. Citado 2 vezes nas páginas 27 e 29.

24 MADEJ, A. et al. Priority-based fair scheduling in edge computing. *arXiv preprint arXiv:2001.09070*, 2020. Citado 2 vezes nas páginas 28 e 29.

25 SAMANTA, A.; ESPOSITO, F.; NGUYEN, T. G. Asap: Adaptive and scalable microservice provisioning for edge-iot networks. In: *Proceedings of the IEEE International Conference on Edge Computing*. [S.l.: s.n.], 2022. p. 1–8. Citado 4 vezes nas páginas 28, 29, 38 e 48.

26 LINDNER, M. *Priority-Based Mobile Edge Cloud Offloading (PBMECO) Algorithm*. Tese (Doutorado) — TU Wien, 2020. Citado 3 vezes nas páginas 28, 29 e 38.

27 AL-DULAIMY, A.; AL-DOGHMAN, F.; ELKHATIB, Y. Providing real-time support for containerized edge services via priority scheduling. *Future Generation Computer Systems*, v. 117, p. 367–379, 2021. Citado 2 vezes nas páginas 28 e 29.

28 NEELY, M. J. Stability and probability 1 convergence for queueing networks via lyapunov optimization. *Journal of Applied Mathematics*, Wiley Online Library, v. 2012, n. 1, p. 831909, 2012. Citado na página 28.

29 ABOUAOMAR, A. et al. Resource provisioning in edge computing for latency sensitive applications. *arXiv preprint arXiv:2201.11837*, 2022. Citado 2 vezes nas páginas 28 e 29.

30 MOHAMMADI, S.; AKBARI, B. Priority-aware resource reallocation in edge computing using reinforcement learning. *Journal of Network and Systems Management*, Springer, v. 34, n. 1, p. 20, 2026. Citado 2 vezes nas páginas 29 e 48.

31 SOUZA, P. S.; FERRETO, T.; CALHEIROS, R. N. Edgesimpy: Python-based modeling and simulation of edge computing resource management policies. *Future Generation Computer Systems*, Elsevier, v. 148, p. 446–459, 2023. ISSN 0167-739X. Citado na página 37.