PUCRS

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

JOSÉ ANTÔNIO COLPO DE ARAÚJO

VALIDANDO ESCALONAMENTO BASEADO EM INTERFERÊNCIA EM AMBIENTES CONTEINERIZADOS

Porto Alegre
2024

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

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

VALIDANDO ESCALONAMENTO BASEADO EM INTERFERÊNCIA EM AMBIENTES CONTEINERIZADOS

JOSÉ ANTÔNIO COLPO DE ARAÚJO

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. César Augusto Fonticielha De Rose
Co-Orientador: Prof. Dr. Vinícius Meyer

Porto Alegre
2024

Ficha Catalográfica

A663v Araújo, José Antônio Colpo

Validando escalonamento baseado em interferência em ambientes conteinerizados / José Antônio Colpo Araújo. – 2024.
59 p.
Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. César Augusto Fonticielha De Rose.
Coorientador: Prof. Dr. Vinícius Meyer.

1. docker. 2. kubernetes. 3. interferência. 4. label. 5. estratégias de escalonamento. I. De Rose, César Augusto Fonticielha. II. Meyer, Vinícius. III. , . IV. 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

JOSÉ ANTÔNIO COLPO DE ARAÚJO

VALIDANDO ESCALONAMENTO BASEADO EM INTERFERÊNCIA EM AMBIENTES CONTEINERIZADOS

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 em 22 de Março de 2024.

BANCA EXAMINADORA:

Prof. Dr. Gerson Geraldo Homrich Cavalheiro (UFPEL)

Prof. Dr. Avelino Francisco Zorzo (PPGCC/PUCRS)

Prof. Dr. Vinícius Meyer (Univates - Co-Orientador)

Prof. Dr. César Augusto Fonticielha De Rose (PPGCC/PUCRS - Orientador)

DEDICATÓRIA

Dedico este trabalho à minha querida mãe, que sempre me apoiou a concluir todos os objetivos.

“A persistência é o caminho do êxito.”
(Charles Chaplin)

AGRADECIMENTOS

Expresso minha sincera gratidão ao Prof. Dr. César Augusto Fonticielha De Rose e Prof. Dr. Vinícius Meyer, cuja sabedoria, paciência e apoio foi o farol que guiou esta jornada acadêmica. Suas orientações não apenas iluminaram o caminho, mas também me inspiraram a buscar além dos horizontes conhecidos, transformando desafios em oportunidades de aprendizado. Agradeço profundamente por suas inestimáveis contribuições, não só como orientadores, mas como uma fonte de inspiração acadêmica e pessoal. Este trabalho é o reflexo do compromisso e paixão que compartilhamos pela ciência e inovação.

VALIDANDO ESCALONAMENTO BASEADO EM INTERFERÊNCIA EM AMBIENTES CONTEINERIZADOS

RESUMO

O objetivo deste trabalho é validar uma estratégia de gerenciamento de recursos baseada em interferência em um ambiente conteinerizado utilizando Kubernetes e contêineres com Docker. Foram realizadas análises detalhadas sobre escalonamento com base em interferência, e implementações de políticas customizadas para o Kubernetes. Experimentos foram executados para avaliar a eficácia das estratégias em ambientes virtualizados para comparação de resultados com as políticas tradicionais de escalonamento. A implementação de uma política de escalonamento baseada em atribuição estática de interferência demonstrou melhorias significativas no desempenho e na eficiência de recursos em ambientes conteinerizados. O trabalho pretende abrir caminho para pesquisas futuras, avanços no campo do gerenciamento de recursos conteinerizados, e otimização de desempenho nesse ambiente levando em conta a interferência, como por exemplo estratégias que se utilizam de atribuição dinâmica de perfis de interferência.

Palavras-Chave: docker, kubernetes, interferência, estratégias de escalonamento.

VALIDATING INTERFERENCE-BASED SCHEDULING IN CONTAINERIZED ENVIRONMENTS

ABSTRACT

The objective of this work is to validate an interference-based resource management strategy in a containerized environment using Docker and Kubernetes. Detailed analyzes were performed on interference-based scaling and custom policy implementations for Kubernetes. Experiments were carried out to evaluate the effectiveness of the strategies in virtualized environments to compare results with traditional scheduling policies. Implementing a scheduling policy based on static interference assignment has demonstrated significant improvements in performance and resource efficiency in containerized environments. The work aims to pave the way for future research and advances in the field of containerized resource management and performance optimization in this environment taking interference into account, such as strategies that use dynamic assignment of interference profiles.

Keywords: docker, kubernetes, interference, management strategies.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE SIGLAS

API – Application Programming Interface (Interface de Programação de Aplicativos)

CPU – Central Processing Unit (Unidade de Processamento Central)

GB – Giga Byte

HDD – Hard Disk Drive (Disco Rígido)

I/O – Input/Output (Entrada/Saída)

ILS – Iterated Local Search (Busca Local Iterada)

INTP – Interference Profiler

LLC – Last Level Cache

QOS – Quality of Service (Qualidade de Serviço)

RAM – Random Access Memory (Memória de Acesso Aleatório)

VM – Virtual Machine (Máquina Virtual)

SUMÁRIO

1. INTRODUÇÃO

A crescente adoção de ambientes virtualizados na computação em nuvem tem sido uma prática comum para melhorar a eficiência e a flexibilidade no uso de recursos de *hardware*. No entanto, o gerenciamento eficaz desses recursos é essencial para garantir o desempenho e a estabilidade das aplicações executadas nesses ambientes [20].

À medida que as organizações, tanto publicas quanto privadas, migram para ambientes altamente virtualizados, para aproveitar os benefícios que essas tecnologias oferecem, a capacidade de gerenciar recursos de maneira eficaz torna-se um desafio crítico. Em um ambiente onde múltiplas aplicações e serviços compartilham a mesma infraestrutura física, a interferência entre as cargas de trabalho pode levar a uma degradação significativa do desempenho, afetando negativamente a experiência do usuário final e consequentemente os objetivos comerciais da organização. Cliente e provedor podem ser beneficiados ao levar em consideração o impacto da interferência no desempenho, o cliente com uma resposta mais rápida das requisições e o provedor com otimização de recursos e maior performance [11].

Atualmente, existem políticas de balanceamento de carga que se concentram principalmente na alocação de recursos com base em capacidades como CPU e memória. Embora essas políticas sejam úteis para otimizar o uso dos recursos disponíveis, elas não consideram o impacto da interferência entre as aplicações que compartilham um mesmo nó virtualizado [7].

Em pesquisas anteriores, nosso grupo desenvolveu uma estratégia de gerenciamento de recursos baseada na interferência entre aplicações [17]. Essa abordagem demonstrou resultados promissores, superando as políticas tradicionais que se baseiam apenas em capacidades. No entanto, até o momento, essa estratégia ainda não foi validada em ambientes de última geração baseados em contêineres, como Docker e Kubernetes.

O Kubernetes é uma plataforma amplamente adotada para orquestração de contêineres, permitindo o dimensionamento e gerenciamento eficiente de aplicativos em ambientes de *cluster* [13]. No entanto, em um ambiente compartilhado, onde múltiplos contêineres competem pelos mesmos recursos, a interferência entre eles pode afetar significativamente o desempenho e a eficiência das aplicações.

Nesse contexto, apresentamos o projeto de validação da incorporação de política de escalonamento no Kubernetes, baseada na classificação de interferência entre os recursos. Essa política visa mitigar os efeitos negativos da interferência e melhorar o desempenho geral do sistema. Utilizando a API do Kubernetes e a atribuição de *labels* específicas, é possível identificar e classificar o nível de interferência em três recursos críticos: I/O, memória e CPU.

Essa classificação de interferência foi encontrada em um estudo realizado por Uillian L. Ludwig, intitulado "*Optimizing multi-tier application performance with interference and affinity-aware placement algorithms*" [16]. A abordagem de classificação baseada em níveis permite a identificação precisa do impacto da interferência em cada recurso.

O objetivo principal dessa política de escalonamento é aumentar a eficiência e o desempenho dos contêineres, garantindo que os recursos sejam alocados para minimizar a interferência. Ao migrar os contêineres para nós com menor uso de recursos, espera-se reduzir a interferência e melhorar o desempenho geral das aplicações. Essa abordagem que combina a atribuição estática de rótulos, indicando o uso intensivo de recursos por nó, com a dinâmica de orquestração e escalonamento de contêineres. Enquanto os rótulos (*labels*) permanecem fixos, refletindo o perfil de uso de recursos no início, o processo de gestão dos contêineres é flexível, ajustando-se às mudanças na carga de trabalho e na configuração do sistema. Esta estratégia permite um equilíbrio entre a eficiência inicial na distribuição de recursos e a adaptabilidade necessária para responder a novas demandas, assegurando a otimização contínua do desempenho e uso de recursos em ambientes conteinerizados.

Neste trabalho, apresentaremos em detalhes a validação dessa política de escalonamento baseado em interferência, incluindo a lógica de identificação do nó mais eficiente, sendo aquele que consegue aproveitar o máximo possível dos recursos de forma a minimizar a interferência, os desafios enfrentados durante o processo e os resultados obtidos. Essas contribuições não apenas fornecem conhecimentos valiosos para a comunidade científica e profissionais da área de TI, como também abrem caminho para futuras pesquisas em otimização de ambientes conteinerizados, promovendo uma melhor compreensão e gestão da interferência, um desafio persistente nesses ambientes computacionais.

2. BACKGROUND

Nesse capítulo são abordados alguns conceitos e tecnologias relevantes ao contexto deste trabalho.

2.1 Ambientes virtualizados

Em um ambiente virtualizado, várias máquinas virtuais (VMs) ou contêineres são executados em uma única máquina *host* física. Cada VM ou contêiner atua como um sistema separado e isolado, mas compartilham os recursos da máquina *host*, como memória, armazenamento e CPU [1]. Quando várias VMs ou contêineres estão em execução no mesmo *host* físico, eles podem competir por esses recursos compartilhados. Isso pode causar problemas de interferência, em que o uso de um recurso por VM ou contêiner afeta o desempenho de outras VMs ou contêineres em execução no mesmo *host*.

Por exemplo, se uma VM estiver usando uma grande quantidade de memória, isso pode fazer com que a máquina *host* comece a trocar, o que pode diminuir o desempenho de todas as outras VMs em execução no *host*. Da mesma forma, se um contêiner estiver usando uma alta porcentagem da CPU, isso poderá fazer com que outros contêineres tenham desempenho reduzido [17].

Para atenuar esses problemas de interferência, técnicas como alocação e isolamento de recursos para limitar a quantidade de recursos que cada VM ou contêiner pode usar, para que não interfiram entre si. Eles também podem usar monitoramento e ajuste de desempenho para identificar e resolver degradações de desempenho causados pelo alto uso dos recursos.

2.1.1 Contêiner

A tecnologia de contêineres surgiu como uma solução revolucionária para a entrega e a operacionalização de software, permitindo que aplicações sejam empacotadas juntamente com suas dependências, garantindo assim a consistência em diversos ambientes de execução e rapidez na inicialização. Esta abordagem facilita significativamente o desenvolvimento, teste e implantação de aplicações. Além disso, contêineres promovem uma utilização mais eficiente dos recursos do sistema em comparação com máquinas virtuais tradicionais, pois compartilham o mesmo sistema operacional e isolam apenas o necessário para sua execução. Os contêineres são fundamentais para a agilidade, eficiência e portabilidade no desenvolvimento de software moderno e escalável [11].

O Docker é uma tecnologia fundamental para a criação e gerenciamento de imagens de contêineres. Ele oferece uma abordagem padronizada para empacotar aplicativos juntamente com suas dependências [5]. No contexto deste trabalho, o Docker é utilizado para criar as imagens de contêineres a serem executadas em um ambiente conteinerizado.

Essas imagens são construídas a partir de um arquivo chamado Dockerfile [8], o qual especifica as configurações e dependências necessárias para o ambiente de execução. A utilização dessas imagens é feita para implantar e executar os contêineres em diferentes ambientes.

A integração entre o Docker e outras ferramentas, desempenha um papel crucial no ciclo de vida dos contêineres. O Docker fornece as ferramentas e recursos necessários para criar e gerenciar as imagens de contêineres, possibilitando sua execução em diferentes contextos 2.1.

Essa abordagem facilita a criação e o gerenciamento eficientes de contêineres, permitindo que aplicações e serviços sejam empacotados de maneira isolada e escalável [18]. A flexibilidade e a portabilidade dos contêineres gerados pelo Docker são elementos-chave para sua integração e utilização em ambientes distribuídos.

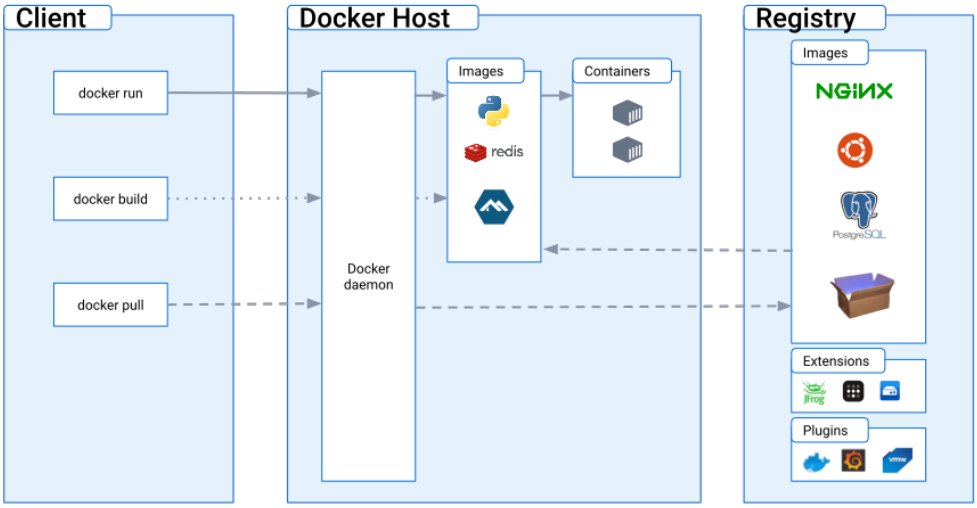

Figura 2.1 – Arquitetura do Docker [5].

2.1.2 Orquestração

No contexto da orquestração de contêineres, o Kubernetes desempenha um papel fundamental. Sua política de escalonamento padrão *(Default Scheduler)* tem como

objetivo distribuir e equilibrar os contêineres entre os nós do *cluster*, considerando requisitos de recursos como CPU e memória, além de levar em conta as restrições e preferências definidas pelos administradores [12].

O Kubernetes segue uma arquitetura de mestre e trabalhador, em que o mestre coordena e gerencia o cluster, enquanto os trabalhadores executam as cargas de trabalho dos contêineres [6]. Para simplificar a configuração inicial do mestre e dos trabalhadores, o kubeadm é uma ferramenta oficial do Kubernetes que facilita esse processo 2.2.

A conexão dos trabalhadores ao mestre é essencial para que eles possam receber as tarefas de implantação e execução dos contêineres. Essa arquitetura mestre-trabalhador é fundamental para a implementação da política de escalonamento baseada em interferência, garantindo a coordenação e execução eficiente dos contêineres no *cluster* [9].

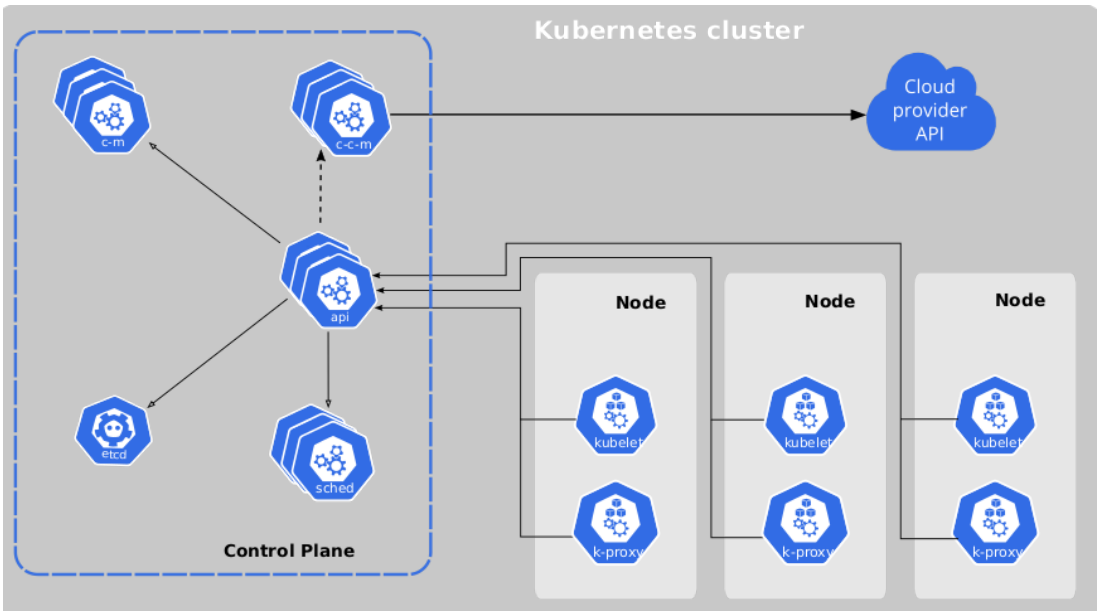

Figura 2.2 – Componentes principais do Kubernetes [13].

2.1.3 Políticas de balanceamento de carga tradicionais

Políticas de balanceamento de carga são essenciais para otimizar a distribuição de recursos em ambientes virtualizados, como o Kubernetes [9]. No ecossistema do Kubernetes, existem diversas políticas disponíveis que buscam equilibrar a carga de trabalho entre os nós de um *cluster*. No entanto, essas políticas ainda se baseiam em capacidades para realizar a alocação de recursos.

A capacidade de recursos refere-se à quantidade de recursos computacionais disponíveis para execução de tarefas. Esses recursos incluem, mas não se limitam a, poder de processamento (CPU), memória (RAM) e armazenamento (disco). A capacidade

de recursos é crucial para garantir que as aplicações executem eficientemente, cumprindo os requisitos de desempenho e escalabilidade.

Essas políticas tradicionais são úteis para garantir que cada nó tenha recursos suficientes para executar as aplicações atribuídas a ele. Por exemplo, ao usar políticas de balanceamento de carga baseadas em CPU e memória, é possível evitar que um nó fique sobrecarregado enquanto outros têm recursos ociosos. Contribuindo para um melhor aproveitamento dos recursos disponíveis no *cluster*.

No entanto, essas políticas não levam em consideração o efeito da interferência entre as aplicações, que ocorre quando múltiplas aplicações em execução no mesmo nó utilizam recursos de hardware compartilhado, isso pode levar a um impacto negativo no desempenho de outras aplicações devido à competição por recursos.

Dessa forma, políticas de balanceamento de carga baseadas apenas em capacidades podem não ser suficientes para garantir um desempenho eficiente das aplicações em ambientes de alta escalabilidade, onde várias aplicações são executadas em um mesmo nó.

2.2 Interferência

Com a implementação de técnicas de compartilhamento de recursos, máquinas físicas agora têm a capacidade de hospedar vários aplicativos simultaneamente. No entanto, ao utilizar métodos de compartilhamento de recursos, como a virtualização e conteinerização, surge a questão da interferência de recursos, quando vários serviços intensivamente usam um recurso simultaneamente, resultando em problemas de contenção (interferência no desempenho), podendo causar degradação na performance [17].

A virtualização e a consolidação de servidores são impulsionadores essenciais da alta utilização de recursos em *data centers* modernos. Quando múltiplas VMs são combinadas em um mesmo servidor, pode ocorrer uma degradação significativa do desempenho, conhecida como interferência da máquina virtual [20].

A interferência no desempenho também afeta os ambientes baseados em contêineres. Aplicativos que apresentam características de uso intensivo de disco, quando executados em contêineres, promovem degradação no desempenho ao competirem intensivamente por diferentes recursos. Xavier et al. [22] testaram várias combinações de cargas de trabalho co-hospedadas e observaram que algumas dessas combinações levaram a uma degradação no desempenho de até 38%, enquanto outras puderam ser combinadas com baixa interferência.

Em sistemas de *cluster*, vários aplicativos executam simultaneamente, muitas vezes de diferentes usuários, competindo pelo acesso a recursos compartilhados, como o sistema de arquivos ou a memória. O baixo desempenho de um aplicativo pode ser resul-

tado da interferência de diferentes fontes. Shah et al. [19] afirmam que o mapeamento de dados de desempenho relacionados a recursos compartilhados em intervalos de tempo pode evidenciar a simultaneidade de uso de aplicativos entre trabalhos, o que pode indicar interferência entre eles. Em algumas situações, a interferência entre aplicações pode causar uma degradação significativa no desempenho.

Para mitigar os efeitos da interferência e aprimorar o desempenho da aplicação em ambientes conteinerizados, um escalonador que considera problemas de interferência se mostra uma solução promissora. Zhu e Tung [24], Bu et al. [4], Zang et al. [23] apresentam estratégias de escalonamento que incorporam aspectos de interferência, baseadas em modelos de previsão de desempenho para tomar decisões mais eficazes sobre o posicionamento da carga de trabalho. Esses modelos propostos alcançam uma margem de erro média inferior a 8% e uma aceleração de 1,5 a 6,5 vezes para trabalhos individuais, respectivamente.

A identificação da interferência pode ser um desafio, pois os contêineres possuem isolamento e podem não ser capazes de visualizar diretamente as atividades uns dos outros. Portanto, é necessário utilizar ferramentas específicas para monitorar e medir o impacto da interferência. Métricas como tempo de resposta *(Makespan)* e uso de recursos podem ser analisadas para identificar possíveis interferências.

2.2.1 A ferramenta IntP

O IntP é uma ferramenta que quantifica a interferência causada pelas tarefas de aplicativos nos recursos de *hardware* e fornece métricas sobre as demandas dos aplicativos durante a execução. Sua arquitetura e componentes integrados operam no nível do sistema operacional (modo *kernel*), permitindo a coleta de métricas de diferentes subsistemas de *hardware* e níveis do sistema operacional [21].

O IntP foi projetado para não ser intrusivo em cargas de trabalho, executar durante a execução do aplicativo e instrumentar aplicativos independentemente da quantidade executada na mesma máquina. Sua estrutura no modo *kernel* permite a instrumentação de todos os componentes do sistema operacional, desde *drivers* até filas de agendamento, com mínima intrusão no desempenho. O IntP gera métricas de interferência para CPU, disco, memória, rede e cache, retornando valores normalizados em porcentagem. Quanto maior a métrica, maior a interferência gerada pelo aplicativo. Essas métricas fornecem *insights* sobre o desempenho do aplicativo e sua interferência nos recursos do sistema:

- Camada de Bloco (Armazenamento): Analisa o tempo de serviço médio para requisições de I/O e a taxa de chegada para quantificar interferências na fila de armazenamento;

- Rede: Analisa o tempo de serviço médio para fila de envio e recebimento, bem como a capacidade de banda para diferenciar tarefas intensivas em rede;
- CPU: Coleta métricas de trocas de contexto (*context-switch*) para inferir interferência na CPU e no despachante do sistema operacional;
- Memória: Avalia a interferência durante o acesso à memória, medindo a quantidade de leituras e escritas de memória por unidade de tempo;
- LLC (*Last Level Cache*): Monitora a ocupação do cache para inferir a interferência causada por aplicativos em relação ao uso do cache;

A utilização dessa ferramenta é crucial para tomar decisões de escalonamento e balanceamento de carga mais eficientes. É importante destacar que, originalmente, ela não foi projetada para suportar virtualização, mas, está sendo aprimorada nesse sentido. No entanto, até o momento deste trabalho, ainda não tivemos acesso a essa versão aprimorada. Portanto, foi necessário encontrar uma alternativa para utilizar a ferramenta no contexto proposto.

Em resumo, a ferramenta IntP é uma peça-chave na validação da política de escalonamento proposta, fornecendo dados precisos e confiáveis sobre a interferência. Sua utilização permite a implementação de uma política de escalonamento com base em interferência e afinidade, resultando em uma melhoria significativa no desempenho de aplicações.

3. ESTADO DA ARTE E TRABALHOS RELACIONADOS

No campo da gerência de recursos em ambientes virtualizados, diversas abordagens têm sido desenvolvidas para otimizar a alocação de recursos entre as aplicações. Políticas de balanceamento de carga baseadas em capacidades, como CPU e memória, são comumente utilizadas para garantir um uso eficiente dos recursos do *cluster* Kubernetes [10].

Diversos estudos têm se dedicado a desenvolver algoritmos de escalonamento inteligentes e adaptativos que levem em consideração a interferência entre os contêineres. Essas políticas visam otimizar a alocação de recursos, distribuindo as cargas de trabalho de forma a minimizar a interferência e maximizar o desempenho geral do sistema.

Alguns dos trabalhos recentes têm explorado o uso de técnicas de *machine learning* para prever e mitigar a interferência entre contêineres [11]. Essas abordagens utilizam dados históricos de desempenho e características dos aplicativos para ajustar dinamicamente o escalonamento e evitar conflitos entre as cargas de trabalho.

Outras políticas têm se concentrado na criação de métricas mais sofisticadas para avaliar a interferência entre os contêineres e identificar possíveis degradações de desempenho no *cluster* [20]. Essas métricas permitem que o sistema tome decisões mais precisas sobre como distribuir as cargas de trabalho, alocando recursos de forma mais equilibrada, evitando o impacto negativo da interferência.

3.1 Trabalhos relacionados do grupo

O artigo "*Optimizing multi-tier application performance with interference and affinity-aware placement algorithms*" de Uillian L. Ludwig [16] ofereceu uma contribuição crucial para realização deste trabalho. Através da utilização dos níveis de interferência exploradas nesse artigo foi possível categorizar o impacto da interferência. Este estudo ofereceu um auxilio conceitual para avaliar e mitigar interferências em cargas de trabalho dinâmicas e sensíveis à latência, fornecendo um alicerce fundamental para a abordagem adotada neste trabalho. O artigo em questão propõe um conjunto de algoritmos de *placement* para aplicações *multi-tier* que consideram tanto a interferência de recursos quanto a afinidade de rede. As decisões de *placement* são as decisões sobre onde colocar diferentes componentes de uma aplicação. Essas decisões são importantes para o desempenho da aplicação, pois podem afetar a comunicação entre os componentes, o uso de recursos e a escalabilidade.

O artigo "*IADA: Interference-aware cloud scheduling architecture for dynamic latency-sensitive workloads*" de Vinicius Meyer [17] propõe uma nova arquitetura de agendamento em nuvem que leva em consideração a interferência entre cargas de trabalho. A arquitetura é baseada em um modelo de interferência que estima o impacto da coexistência de duas cargas de trabalho em um mesmo nó. O modelo leva em consideração os recursos compartilhados pelos nós, como CPU, memória e rede. A arquitetura foi avaliada em um ambiente virtualizado e os resultados da avaliação mostraram que ela é capaz de melhorar o desempenho de cargas de trabalho dinâmicas e sensíveis à latência, e também a escalabilidade das cargas de trabalho. A proposta é uma contribuição importante para o campo de agendamento em nuvem.

3.2 Trabalhos relacionados externos

O artigo "*Adaptive Resource Allocation for Scalable Scientific Workflows in Heterogeneous Clusters*" de Jonathan Bader [3] apresenta um sistema para alocação de recursos adaptativa para *workflows* científicos escaláveis em *clusters* heterogêneos. O sistema, chamado Tarema, usa *labels* do Kubernetes para classificar os nós do *cluster* em grupos de desempenho semelhantes. Isso é feito atribuindo *labels* aos nós com base em sua capacidade de CPU, memória e rede. As tarefas de *workflow* são então alocadas aos nós com base em suas demandas de recursos e na capacidade dos nós. O uso de *labels* do Kubernetes permitiu que o sistema desenvolvido alocasse as tarefas de *workflow* de forma mais eficiente e eficaz em *clusters* heterogêneos. Isso ocorre porque o sistema é capaz de levar em consideração a capacidade dos nós. Como resultado, foi possível reduzir o tempo de execução total dos *workflows* e melhorar a utilização do *cluster*.

O artigo "*Proactive Autoscaling for Edge Computing Systems with Kubernetes*" de Li Ju [11] apresenta uma abordagem de escalonamento proativo para sistemas de computação usando o Kubernetes. O artigo foca na implementação de um escalonador personalizado para o Kubernetes, que permite prever as demandas futuras de recursos do sistema e ajustar dinamicamente a capacidade dos nós do *cluster* antes que problemas de desempenho ocorram. O escalonador personalizado é projetado para monitorar métricas relevantes, como a utilização de CPU e a latência de rede, em todos os nós do *cluster*. Com base nessas métricas, o mecanismo de previsão de carga analisa os dados coletados e faz projeções sobre a carga de trabalho futura. Com as previsões em mãos, o controlador toma decisões de escalonamento pró-ativo, escalando os nós do *cluster* para cima ou para baixo, conforme necessário, para atender às demandas. Essa abordagem proativa permite que o sistema se adapte rapidamente a variações de carga, garantindo um desempenho otimizado e evitando sobrecargas. O escalonamento customizado do Kubernetes baseado em previsões ajuda a melhorar a eficiência e a utilização de recursos

do ambiente, garantindo um funcionamento mais eficiente e responsivo das aplicações, ajudando a enfrentar os desafios únicos desses cenários.

O artigo "*An interference-aware virtual machine placement strategy for high-performance computing applications in clouds*" de Melo Alves [2] aborda a questão da interferência em ambientes *cloud*, especificamente em máquinas virtuais utilizadas para aplicações de alto desempenho, utilizando diferentes plataformas na nuvem para exemplificar os benefícios do uso de cada uma. O principal objetivo do estudo é propor uma estratégia de alocação de máquinas virtuais que leve em consideração a interferência entre as instâncias para melhorar o desempenho dessas aplicações na nuvem utilizando o *framework Iterated Local Search* (ILS). A estratégia considera o uso de métricas de interferência para avaliar o impacto de cada máquina virtual no mesmo *host* físico para posicionar as instâncias de forma a minimizar a interferência. Os resultados mostraram que a estratégia proposta reduziu a interferência em média 40%, para alguns cenários específicos com alto uso de disco durante a execução das tarefas.

O artigo "*Paragon: QoS-aware scheduling for heterogeneous datacenters*", escrito por Delimitrou e Kozyrakis [7], propõe um escalonador de tarefas chamado Paragon projetado para datacenters heterogêneos. O objetivo principal é garantir a Qualidade de Serviço (QoS) das aplicações em ambientes de *datacenter* com múltiplos tipos de recursos computacionais. O escalonador Paragon considera a QoS de cada aplicação e atribui tarefas às máquinas com recursos necessários para atender aos requisitos de desempenho estabelecidos. O escalonador utiliza informações sobre o perfil de desempenho dos recursos das máquinas para tomar decisões inteligentes de escalonamento. Além disso, o Paragon é capaz de lidar com a variação dinâmica da carga de trabalho e as características heterogêneas das máquinas. Os resultados experimentais demonstram que o Paragon supera as políticas de escalonamento tradicionais em termos de QoS, garantindo que as aplicações tenham acesso aos recursos necessários para executar com alto desempenho. O escalonador Paragon é uma contribuição significativa para a otimização do desempenho e eficiência em *datacenters* heterogêneos, melhorando a experiência do usuário e a efetividade dos serviços em ambientes de nuvem e computação distribuída.

3.3 A proposta deste trabalho

Ao revisar o estado da arte e os trabalhos relacionados, observamos uma variedade de abordagens voltadas para a otimização do escalonamento e gerenciamento de recursos em ambientes virtualizados. Muitos desses trabalhos focam no desenvolvimento de algoritmos inteligentes e adaptativos, que consideram a interferência e a alocação de recursos com base em métricas de desempenho e previsões. Estratégias como o uso de *machine learning* para antecipar demandas de recursos e a implementação de políticas de

escalonamento que levam em conta a afinidade e interferência entre tarefas demonstram avanços significativos no campo.

No entanto, este trabalho distingue-se dos demais ao focar especificamente na avaliação e validação de estratégias de escalonamento que mitigam a interferência em ambientes conteinerizados. Diferentemente de outras propostas que podem aplicar-se genericamente a ambientes virtualizados ou considerar apenas teoricamente a interferência, este estudo incorpora uma abordagem prática e detalhada para medir e mitigar os efeitos da interferência entre contêineres através de políticas customizadas para o Kubernetes, utilizando *labels* para gerenciar e otimizar o escalonamento.

Enquanto os trabalhos relacionados fornecem uma base teórica e metodológica sólida para o entendimento e a abordagem da interferência e do escalonamento em sistemas computacionais distribuídos, este trabalho avança na aplicação prática desses conceitos, demonstrando como estratégias específicas de escalonamento podem ser efetivamente implementadas e validadas em ambientes conteinerizados modernos. Este trabalho destaca-se pela sua abordagem focada na implementação e validação em um ambiente real, contribuindo assim para a evolução prática das técnicas de gerenciamento de recursos em ambientes conteinerizados.

4. ESCALONAMENTO BASEADO EM ATRIBUIÇÃO ESTÁTICA DE PERFIS DE INTERFERÊNCIA

A estratégia explorada no trabalho de Uillian L. Ludwig [16] foi escolhida como base para adaptação ao ambiente conteinerizado, devido ao seu enfoque detalhado e abrangente na otimização de desempenho de aplicações através da análise meticulosa de interferência e afinidade entre recursos. Ludwig propõe uma metodologia que não apenas identifica, mas também quantifica o impacto da interferência entre aplicações que compartilham o mesmo ambiente virtualizado 4.1, oferecendo estratégias de *placement* (*placement*) que consideram essas interações complexas para melhorar o desempenho geral do sistema. Embora o trabalho de Vinicius Meyer [17] também aborde a questão da interferência, focando em cargas de trabalho sensíveis à latência, o estudo de Ludwig foi considerado mais alinhado com os objetivos deste trabalho, especialmente sobre à aplicabilidade direta e à capacidade de adaptar seus algoritmos para o contexto específico do Kubernetes e dos contêineres. A escolha reflete um interesse em explorar e validar uma abordagem compreensiva que pode ser diretamente aplicada para melhorar a orquestração e o escalonamento em sistemas conteinerizados, garantindo assim uma otimização mais efetiva e prática dos recursos computacionais disponíveis.

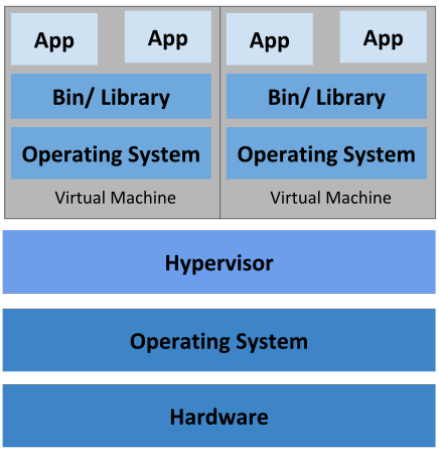

Figura 4.1 – Arquitetura do ambiente virtualizado, camadas de execução de uma máquina virtual [13].

A estratégia central do trabalho de Uillian L. Ludwig [16] é focada em otimizar o desempenho de aplicações multi-camadas em ambientes virtualizados, levando em consideração dois fatores principais: a interferência entre as cargas de trabalho e a afinidade entre as aplicações. Essas camadas representam diferentes partes funcionais de um sistema distribuído, como, por exemplo, uma aplicação *web* dividida em camadas de *front-end*, *back-end*, banco de dados, entre outras.

O estudo reconhece que diferentes cargas de trabalho competindo pelos mesmos recursos de *hardware* (como CPU, memória, disco, e rede) podem causar interferência entre si, levando a uma degradação do desempenho. A interferência ocorre quando várias aplicações ou processos tentam acessar intensivamente um mesmo recurso, resultando em conflitos que afetam negativamente a execução eficiente das aplicações.

Além da interferência, o estudo considera a afinidade entre aplicações, que se refere à relação de proximidade ou dependência entre diferentes componentes de uma aplicação multi-camadas ou entre diferentes aplicações que se beneficiam de serem alocadas no mesmo *host* físico ou virtual. As “camadas“ referem-se aos diferentes componentes ou partes de uma aplicação de software, conhecidos como *“tiers“*. Afinidades podem ser baseadas em requisitos de latência de rede, comunicações intensivas entre componentes ou dependências de dados 4.2.

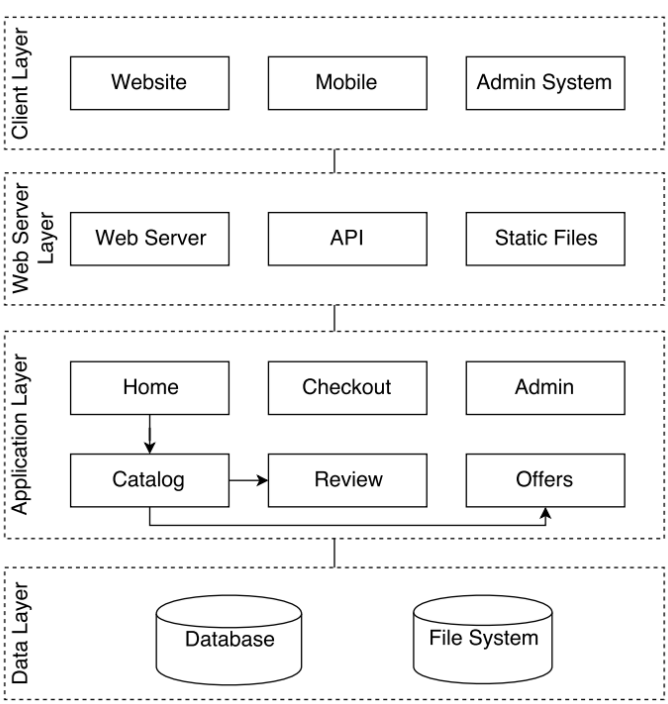

Figura 4.2 – Exemplo de arquitetura multi-camadas de um sistema de comércio eletrônico [16].

A pesquisa propõe algoritmos de *placement* que levam em conta tanto a interferência quanto a afinidade para tomar decisões informadas sobre onde alojar máquinas virtuais (VMs). *Placement* de uma VM refere-se ao processo de alocação e distribuição estratégica de uma VM em um servidor. Esses algoritmos buscam equilibrar a carga entre os recursos disponíveis, minimizando a interferência entre as cargas de trabalho enquanto maximizam a afinidade entre componentes relacionados para melhorar o desempenho geral do sistema.

Estes algoritmos são projetados para avaliar e categorizar as VMs com base em dois critérios críticos: os níveis de interferência e o uso de recursos. A interferência é classificada em diversos níveis, desde ausente a alta, permitindo uma análise do impacto potencial de diferentes cargas de trabalho compartilhando os mesmos recursos físicos. Simultaneamente, o uso de recursos é avaliado para determinar as demandas específicas de cada VM, seja em termos de CPU, memória ou I/O.

Os limiares *(thresholds)* para a atribuição de níveis de interferência para cada recurso desempenham um papel crucial na estratégia proposta. Esses *thresholds* são definidos como limites específicos para a utilização de recursos, que quando excedidos indicam um aumento significativo na interferência entre as cargas de trabalho. Ao estabelecer esses *thresholds*, é possível categorizar os níveis de interferência, com base na intensidade da competição por recursos compartilhados. Ao considerar esses *thresholds*, os algoritmos propostos são capazes de identificar e mitigar eficientemente potenciais degradações no desempenho, garantindo uma distribuição equilibrada dos recursos e uma melhor utilização dos recursos disponíveis.

A implementação desses algoritmos envolve a categorização das VMs em grupos, de acordo com seus perfis de interferência e requisitos de recursos. O modelo de repulsão visa minimizar a interferência entre as cargas de trabalho, esse modelo é baseado na idéia de que a co-localização de VMs com requisitos de recursos semelhantes pode levar a conflitos de recursos e, consequentemente, à degradação do desempenho das aplicações. Portanto, o modelo de repulsão propõe uma abordagem na qual as VMs são distribuídas de forma a evitar a proximidade de outras VMs que demandam recursos similares 4.3. Isso é alcançado através da introdução de uma força de repulsão entre VMs que compartilham recursos semelhantes, o que os afasta uns dos outros no espaço de alocação. Por exemplo, VMs com alto uso de CPU, mas baixa interferência, podem ser colocadas juntas em um mesmo nó físico, otimizando a utilização da CPU sem prejudicar o desempenho de cada VM. Em contrapartida, VMs que apresentam alta interferência, especialmente em operações de I/O, são distribuídas de forma a evitar a sobreposição em nós que já sustentam cargas de trabalho intensivas de I/O, mitigando assim o risco de degradação do desempenho.

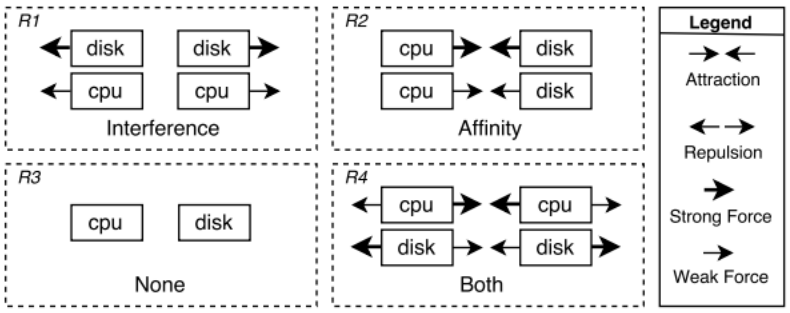

Figura 4.3 – Ilustração das políticas de *placement* baseadas na repulsão [16].

A estratégia inclui modelar o ambiente de execução e simular diferentes cenários de *placement* para identificar configurações com melhor desempenho. Utiliza-se uma combinação de análise teórica e experimentação prática para avaliar o impacto das políticas de *placement* no desempenho das aplicações. Isso permite uma compreensão detalhada de como diferentes fatores, como a configuração do *hardware* e a natureza das cargas de trabalho, influenciam a eficiência do escalonamento.

Os resultados encontrados no trabalho do Ludwig demonstram a eficácia significativa dos algoritmos propostos na melhoria do desempenho de aplicações multi-camadas por meio de uma gestão estratégica da interferência entre cargas de trabalho e da exploração da afinidade de recursos. A aplicação desses algoritmos levou a uma distribuição mais eficiente das VMs dentro do ambiente computacional, resultando em uma redução notável da interferência no desempenho. Consequentemente, observou-se uma melhoria no tempo de resposta das aplicações e uma utilização mais equilibrada dos recursos do sistema, colaborando com a hipótese do *placement* estratégico, que considera tanto a interferência quanto a afinidade, pode levar a ganhos substanciais em eficiência e desempenho em ambientes virtualizados. Esses resultados reforçam a importância de estratégias para otimização, destacando o potencial de tais abordagens para aprimorar a experiência do usuário final e a eficácia operacional das aplicações.

5. MAPEAMENTO DA ESTRATÉGIA PARA UM AMBIENTE CONTEINERIZADO

Esse capítulo aborda a adaptação da estratégia criada por Uillian L. Ludwig no trabalho *"OPTIMIZING THE PERFORMANCE OF MULTI-TIER APPLICATIONS USING INTERFERENCE AND AFFINITY-AWARE PLACEMENT ALGORITHMS"* para os desafios e peculiaridades dos ambientes conteinerizados. Esta transição representa uma evolução das práticas de otimização de desempenho, refletindo um avanço em direção à agilidade, escalabilidade e eficiência que os contêineres oferecem em comparação com as abordagens tradicionais baseadas em máquinas virtuais.

A adaptação da estratégia de Ludwig para o contexto conteinerizado não é apenas uma questão de transposição de conceitos; é uma reimaginação que leva em conta a dinâmica única dos contêineres e a orquestração sofisticada fornecida por sistemas como o Kubernetes. O trabalho base, com seu foco em algoritmos de *placement* que consideram minuciosamente a interferência entre as cargas de trabalho e a afinidade de recursos, fornece um sólido alicerce teórico. No entanto, a implementação em ambientes conteinerizados demanda uma consideração sobre como esses princípios podem ser efetivamente aplicados em um ecossistema que privilegia a portabilidade e a escalabilidade.

Refletir sobre como as estratégias abordadas no trabalho base são aplicáveis e foram adaptadas ao contexto de ambientes conteinerizados revela a profundidade da inovação requerida. As características dos contêineres, como seu ciclo de vida, a comunicação entre os contêineres e o compartilhamento do *kernel* do sistema operacional *host*, apresentam tanto desafios quanto oportunidades para otimização. A estratégia adaptada, portanto, abraça essas peculiaridades, empregando mecanismos de orquestração para dinamicamente alocar contêineres de maneira que se otimize o uso dos recursos enquanto se minimiza a interferência prejudicial ao desempenho.

Este capítulo explora o processo de adaptação, destacando a importância de manter a fidelidade aos princípios fundamentais de minimização da interferência, ao mesmo tempo em que se navega pelas novas realidades dos ambientes conteinerizados.

5.1 Atribuição estática de perfis de interferência

Os perfis de interferência são atribuídos de forma estática aos contêineres no tempo 0, antes do início do escalonamento, e permanecem inalterados durante todo o ciclo de vida do contêiner. Esses perfis definem a intensidade da interferência causada por cada contêiner em relação aos recursos compartilhados. Por outro lado, o escalonamento

em si é dinâmico, o que significa que os contêineres podem ser movidos de uma máquina para outra ao longo da execução do sistema.

Ao adaptar a estratégia estática de perfis de interferência para o contexto de contêineres, buscamos enfrentar os desafios únicos apresentados por essa tecnologia emergente. Os contêineres oferecem vantagens significativas em relação à virtualização tradicional para validação dessa estratégia, como tempos de inicialização mais rápidos, menor sobrecarga e maior densidade de implantação. No entanto, também trazem consigo novos desafios de gerenciamento de recursos, especialmente em ambientes com várias aplicações e cargas de trabalho heterogêneas.

É importante ressaltar que o objetivo não é substituir completamente os algoritmos de escalonamento padrão do Kubernetes, mas sim aprimorá-los para considerar a interferência estática e a afinidade de comunicação entre os contêineres 5.1. Permitindo uma alocação mais inteligente dos recursos.

Através dessa pesquisa, esperamos contribuir para o avanço da ciência de gerenciamento de recursos em ambientes de contêineres e proporcionar soluções mais eficazes e eficientes para a implantação de aplicações em larga escala.

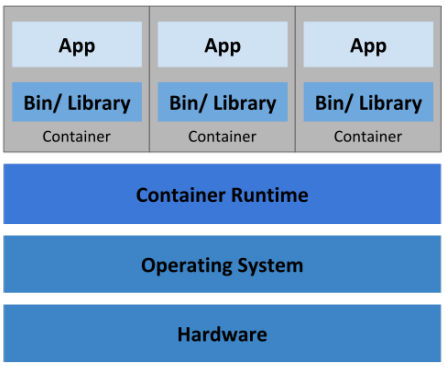

Figura 5.1 – Arquitetura do ambiente conteinerizado, camadas de execução de um contêiner [13].

5.2 Categorização dos níveis de interferência

Um aspecto crucial herdado do trabalho base é a abordagem sistemática dos níveis de interferência. A interferência, no âmbito do trabalho original, é categorizada em diferentes níveis, variando de ”Ausente” a ”Alta”. Essa categorização serve como um mecanismo para quantificar o impacto que diferentes cargas de trabalho podem ter umas nas outras quando compartilham os mesmos recursos físicos ou virtuais. A adaptação desse conceito para ambientes conteinerizados se concentra em identificar e mitigar a interferência potencial entre contêineres, especialmente considerando a natureza compartilhada do *kernel* do sistema operacional e os recursos subjacentes.

- Interferência Ausente: Reflete um cenário ideal, onde as cargas de trabalho coexistem sem qualquer impacto negativo perceptível no desempenho uma da outra. Este nível é raramente alcançado em ambientes densamente virtualizados;
- Interferência Baixa: Indica uma mínima degradação no desempenho, que pode ser aceitável para a maioria das aplicações. A interferência a este nível não impede significativamente as operações normais;
- Interferência Moderada: Representa um nível de interferência que começa a afetar o desempenho das aplicações de maneira mais evidente. Estratégias de mitigação podem ser necessárias para evitar impactos adversos significativos;
- Interferência Alta: Caracteriza-se por uma severa degradação no desempenho, onde a competição por recursos resulta em impactos substanciais na eficiência das aplicações. A intervenção para realocação de recursos ou reescalonamento de cargas de trabalho é crucial;

Tabela 5.1 – Intervalos de interferência e seus respectivos níveis, criado por Ludwig.

Interferência:	Categoria:
0%	Ausente
1% - 20%	Baixa
21% - 50%	Moderada
51% - 100%	Alta

5.3 Atribuição dos níveis de interferência

A atribuição de *labels* (rótulos) aos nós representa uma etapa crucial para implementar a política de escalonamento que considera a interferência. Este processo envolve categorizar os nós do *cluster* Kubernetes com base nos níveis de interferência de cada recurso.

Labels são marcadores em formato de chave-valor que podem ser atribuídos a objetos no Kubernetes, como nós. Eles servem para organizar e selecionar subconjuntos desses objetos com base em critérios definidos, facilitando a gestão e o escalonamento em um *cluster*.

A utilização de *labels* é feita para indicar o uso intensivo de recursos específicos dos contêineres alocados em cada nó. Esses *labels* servem como uma ferramenta de sinalização importante, informando ao sistema de orquestração sobre as demandas de recursos predominantes em cada nó do *cluster*, o que é fundamental para uma alocação

de contêineres otimizada e para a mitigação da interferência entre cargas de trabalho. As atribuições são feitas aos nós com base em sua capacidade e tipo de recurso, como:

- *CPU-intensive*;
- *Memory-intensive*;
- *I/O-intensive*;

Por exemplo, quando um contêiner é responsável por utilizar intensivamente operações de entrada e saída (I/O), o nó que aloca esse contêiner recebe o *label* "I/O-intensive". Isso não apenas identifica o nó como um ponto de alta demanda por operações de I/O, mas também orienta decisões futuras de *placement* de contêineres, evitando a alocação de mais contêineres com perfil similar de uso intenso de I/O no mesmo nó. Essa prática visa distribuir de maneira equilibrada as cargas de trabalho que demandam recursos semelhantes, evitando a sobrecarga de um único nó com múltiplas cargas de trabalho intensivas do mesmo recurso.

Nesta fase, a ferramenta IntP será empregada para identificar os níveis de interferência para cada aplicação e atribuí-los aos contêineres que hospedam essas aplicações. Através da análise realizada pela ferramenta, serão definidos os perfis de interferência de forma estática associados a cada tipo de carga de trabalho.

A atribuição de *labels* baseada no perfil de uso dos recursos pelos contêineres é, portanto, uma técnica proativa para a eficiência do ambiente de computação como um todo. Ao informar explicitamente sobre o tipo de uso intensivo de recursos, facilita-se a implementação de políticas de escalonamento e *placement* que levam em consideração a necessidade de balancear a carga entre os nós, promovendo uma utilização de recursos mais eficaz e mantendo altos níveis de desempenho das aplicações. Este método representa um avanço significativo na orquestração de contêineres, contribuindo para a otimização contínua de ambientes conteinerizados através de uma gestão mais informada e adaptativa dos recursos disponíveis.

Com a utilização dos dados de interferência estáticos e atribuídos à *labels*, a política de escalonamento será capaz de analisar e identificar o nó mais eficiente para executar os contêineres, levando em conta o mínimo de interferência possível entre os recursos.

5.4 Estratégia de alocação de contêineres

A alocação de contêineres, orientada por *labels* e demandas de recursos, estabelece um método robusto para a gestão de recursos em um *cluster* conteinerizado. Ao

alinhar as necessidades específicas dos contêineres com os perfis de recursos dos nós, esta abordagem promove uma utilização mais eficaz da infraestrutura disponível, minimizando conflitos de recursos e potencializando o desempenho das aplicações.

Quando um novo contêiner é solicitado para execução, a lógica de alocação estabelece os seguintes passos:

- Análise de Demanda de Recursos: Determina-se qual recurso é intensivamente demandado pelo contêiner. Esta análise é crucial para identificar o tipo de *label* de nó a ser buscado para uma alocação ideal;
- Identificação de Nós Elegíveis: Procura-se por nós que não possuam o *label* correspondente ao uso intensivo do recurso demandado pelo contêiner. Isso assegura que o contêiner seja alocado em um ambiente onde a competição pelo recurso específico seja minimizada;
- Decisão de *Placement* Baseada em *Labels*: Se existirem nós sem o *label* de uso intensivo do recurso requisitado, o contêiner será inicializado em um desses nós, promovendo uma distribuição equilibrada dos recursos;
- Estratégia para Nós com Recursos Intensivos: Na eventualidade de todos os nós estarem marcados como intensivos no recurso demandado pelo contêiner, opta-se pelo nó que apresenta o menor número de contêineres em execução. Essa abordagem visa diluir a carga de trabalho e mitigar potenciais interferências;

```
1: function AlocarConteiner(C)
2: {Aloca o conteiner C baseando-se no uso intensivo de recursos e menor número de contêineres em execução no nó}
3: NodosDisponiveis ← IdentificarNodosSemLabelUsoIntensivo(C)
4: if NodosDisponiveis ≠ ∅ then
5:     NodoAlvo ← EscolherNodoComMenorNumeroDeConteineres(NodosDisponiveis)
6: else
7:     TodosOsNodos ← ListarTodosOsNodos()
8:     NodoAlvo ← EscolherNodoComMenorNumeroDeConteineres(TodosOsNodos)
9: end if
10: AlocarConteinerNoNodo(C, NodoAlvo)
11: return NodoAlvo
```

Algoritmo 5.1 – Algoritmo para a alocação de contêineres com base no rótulo de uso intensivo de recursos e no número de contêineres em execução.

5.5 Ferramentas usadas na implementação

A ferramenta ”node-tiers” [15] desenvolvida por Ludwig em NodeJs, foi utilizada para criar cargas de trabalho que forçam intensivamente os recursos de sistemas computacionais, com base em requisições a essa aplicação. Ela permite simular ambientes de alta demanda, testando o desempenho e a capacidade de resposta de sistemas em cenários variados, onde cada *endpoint*, foca em estressar diferentes recursos, como disco, memória e CPU.

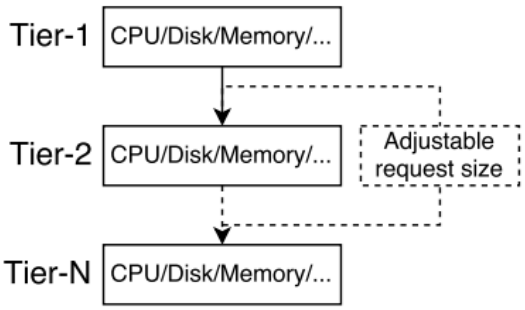

Figura 5.2 – Visão geral da arquitetura do node-tiers [15].

O ”node-tiers” opera como um servidor web que aguarda por requisições HTTP na porta especificada. Cada requisição deve informar qual camada será executada e, se necessário, quais outras camadas deverão ser comunicadas subsequente à execução da tarefa inicial. Essa configuração flexível permite simular diversos cenários de aplicativos multi-camadas, adaptando-se a diferentes requisitos de interferência de recursos e padrões de comunicação de rede. Essa ferramenta pode ser amplamente utilizada para análise detalhada do impacto da *placement* de aplicações em infraestruturas.

Tabela 5.2 – Exemplo de camadas disponíveis no node-tiers.

Recurso:	Método:	Descrição:
CPU	PI	Calcular iterações pi.
Memory	Memory	Alocar a desalocar blocos de memória.
Disk	I/O	Inserção de texto na base de dados.
Cache	zlib	Compressão e descompressão de texto.
Mixed	Matrix	Multiplicação matriz.
None	Nothing	Nada é executado.

A API do Kubernetes é a interface oficial disponibilizada pelo Kubernetes para interagir com o *cluster* por meio de código Python. Ela fornece uma série de recursos e métodos para criar, atualizar e gerenciar objetos do Kubernetes, como nós, *pods*, serviços, entre outros [14]. Através desta API, é possível realizar operações de configuração, monitoramento e escalonamento do *cluster* 5.3.

No contexto deste trabalho, a API oficial do Kubernetes foi empregada para orquestrar a execução de contêineres, gerenciar requisições, coletar métricas de uso de recursos dos nós, adicionar *labels* aos nós e realocar contêineres.

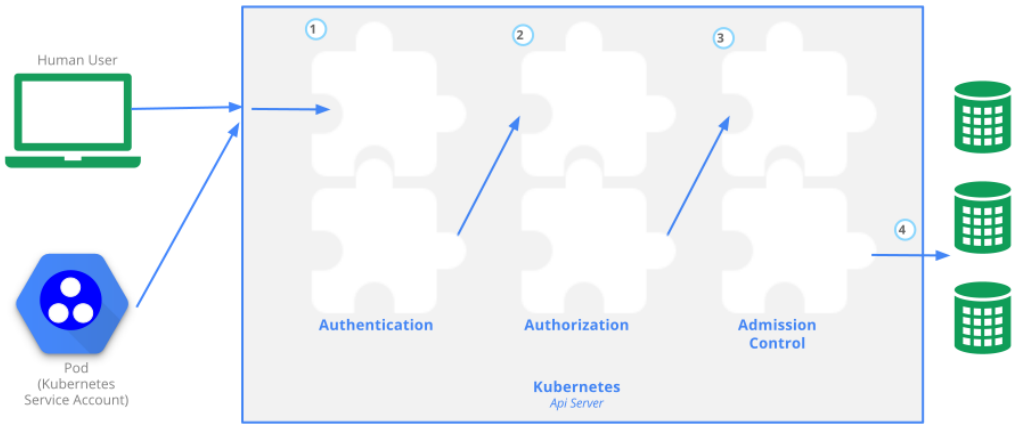

Figura 5.3 – Visão geral do controle de acesso a API Kubernetes [14].

Uma das principais funções implementadas é a função de processamento dos rótulos dos nós. Essa função recebe como parâmetro as informações dos nós do *cluster*, como os *labels* que indicam o nível de interferência de cada recurso. Também foi desenvolvido uma função para facilitar a migração de contêineres para os nós apropriados com base na identificação dos *labels* atribuídos. Essa função é responsável por verificar se os contêineres em execução estão alocados no nó onde o *label* de uso intensivo correspondente não está presente. Caso seja identificada essa condição, indicando a necessidade de realocação do contêiner, a função procede com a migração. Para isso, o contêiner é primeiro finalizado no nó atual e, em seguida, reiniciado no nó onde o *label* de uso intensivo está ausente. Nesse processo também é considerado o número de contêineres em execução em cada nó, garantindo que o contêiner seja realocado para o nó com a menor carga de trabalho possível.

Essas funções são essenciais para a implementação da estratégia proposta nesse trabalho. Elas permitem a análise dos recursos e a tomada de decisões inteligentes sobre a alocação de contêineres, contribuindo para a redução da interferência e o aumento da eficiência do *cluster*.

6. VALIDANDO ESCALONAMENTO BASEADO EM INTERFERÊNCIA EM AMBIENTES CONTEINERIZADOS

Neste capítulo, apresentamos os resultados obtidos a partir da simulação de cargas de trabalho intensivas em contêineres operando em um ambiente distribuído. O foco destas simulações foi avaliar o desempenho de contêineres submetidos a cargas de trabalho que demandam intensivamente CPU, memória e operações de I/O, e quantificar o impacto da interferência destes recursos no tempo de execução das tarefas e utilização dos recursos do Nó.

Exploramos a eficácia de uma política de escalonamento de interferência projetada para mitigar a degradação de desempenho. Um dos principais objetivos é visualizar e quantificar os benefícios dessa política proposta em contêineres, validando a estratégia proposta pelo grupo, demonstrando como ela pode otimizar a utilização dos recursos e melhorar a eficiência geral do sistema. Ao analisar os dados coletados de experimentos controlados, este capítulo fornece informações úteis sobre o impacto da política de escalonamento na redução da interferência entre contêineres, garantindo assim um desempenho mais previsível e estável das aplicações. Através de uma abordagem baseada em evidências, buscamos compreender os diferentes aspectos do escalonamento em ambientes altamente dinâmicos e as implicações práticas de implementar políticas de escalonamento mais inteligentes e adaptativas.

6.1 Ambiente de validação

O ambiente de teste foi configurado nos servidores da PUCRS, composto por cinco máquinas, sendo eles, um nó mestre e quatro nós trabalhadores. As máquinas possuem especificações variadas para simular um ambiente heterogêneo: dois nós Dell R810 com 32 *threads* e 64GB de RAM, e três nós Dell R610, cada um com 16 *threads* e 16GB de RAM. O HDD utilizado nas máquinas é o Dell PowerEdge R610/R810 de 300GB, sua velocidade é de 10.000 RPM.

A limitação de recursos foi estrategicamente implementada no *namespace* dos trabalhadores, garantindo um balanceamento e uma performance equilibrada entre todos os nós, independentemente de suas capacidades individuais. O *namespace* do Kubernetes refere-se a uma forma de realizar configurações de um *cluster* entre múltiplos nós compartilhando as mesmas definições.

Cada máquina virtual foi configurada com o Kubeadm que simplifica o processo de configuração do *cluster* Kubernetes. Utilizando o Kubeadm, foi realizado o provisiona-

mento e a configuração das máquinas. Esse processo envolveu a instalação dos componentes principais do Kubernetes para sua execução apropriada.

Esse ambiente proporcionou as condições necessárias para quantificar o nível de degradação que interferência pode causar em ambientes conteinerizados, e também validar a implementação da política de escalonamento com base em interferência, permitindo a análise de dados simulados e seus benefícios.

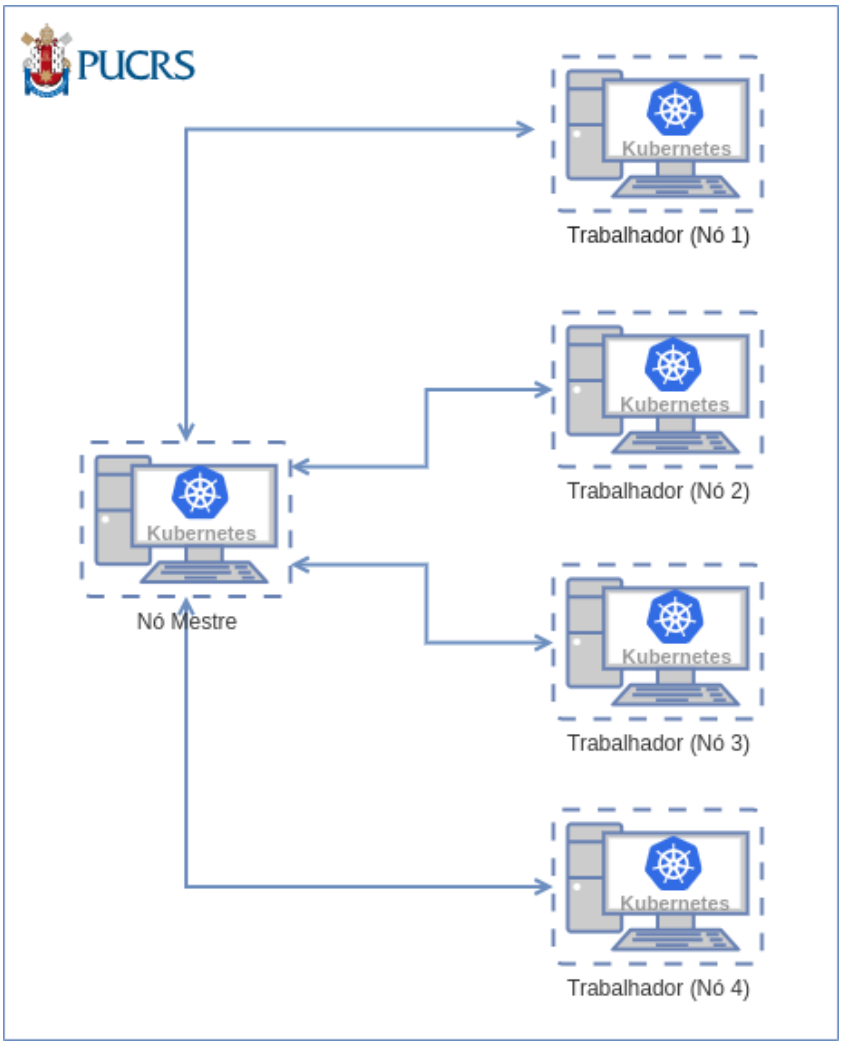

Figura 6.1 – Visão geral do ambiente de teste PUCRS.

6.2 Metodologia de teste

A abordagem metodológica para os testes de avaliação do escalonamento baseado em interferência em ambientes conteinerizados foi projetada para tentar reproduzir cenários reais com as complexidades e desafios enfrentados em ambientes de produção. Como exemplos, consideramos a execução de aplicações web escaláveis, que demandam intensivamente recursos de CPU e memória durante picos de tráfego, ou sistemas de pro-

cessamento de dados em larga escala, que realizam operações de I/O intensivas para análise de grandes volumes de informações. Outro cenário relevante inclui a implantação de microserviços em arquiteturas orientadas a eventos, onde a gestão eficiente de recursos é crucial para o processamento em tempo real de fluxos de dados. Esses exemplos refletem a variedade de aplicações e cargas de trabalho que os sistemas conteinerizados são capazes de suportar de uso intenso de recursos em um *cluster* Kubernetes.

O nó mestre, um componente crítico no controle do *cluster*, foi configurado para administrar a distribuição de cargas de trabalho e coletar métricas de desempenho dos nós trabalhadores. A fim de manter a integridade da coleta de dados, o nó mestre foi isolado da execução das cargas de trabalho. A sua função permaneceu estritamente ligada à coordenação e ao monitoramento das atividades do *cluster*.

Os nós trabalhadores, denominados Nó 1, Nó 2, Nó 3 e Nó 4, foram encarregados da execução das cargas de trabalho intensivas. Para simular a alta demanda em recursos computacionais, cargas de trabalho que exigem o uso intensivo de CPU, memória e I/O foram distribuídas de forma igualitária entre eles.

Essa estratégia permitiu que observássemos o comportamento dos contêineres sob condições de estresse e a maneira como os recursos eram alocados e utilizados durante as operações. Scripts em Python, desenvolvidos para integrar-se com a API oficial do Kubernetes, foram implantados no nó mestre. Esses scripts tinham múltiplas funções: iniciar as cargas de trabalho, monitorar em tempo real a utilização dos recursos nos nós trabalhadores e registrar o tempo de execução de cada tarefa. A coleta de dados foi feita de maneira contínua e automática, garantindo a captura de informações precisas e relevantes para a análise. Todos os arquivos, incluindo código e versões, estão disponíveis no repositório público do Github [1].

A integridade dos dados coletados foi uma prioridade, e uma análise preliminar foi conduzida imediatamente após a coleta para verificar a consistência e a exatidão das métricas registradas. Inicialmente, a política de escalonamento padrão do Kubernetes foi posta à prova para estabelecer um ponto de referência para as comparações futuras. Posteriormente, com a implementação da política de escalonamento que leva em consideração a interferência entre os contêineres, procedemos com uma nova rodada de testes para avaliar as melhorias em relação à linha de base. A metodologia empregada assegurou que os testes fossem replicáveis e controlados, proporcionando uma base sólida para uma análise confiável.

[1] https://github.com/jose-colpo/kubernetes_labels_interferencia

6.3 Simulação de cargas de trabalho

O núcleo da simulação foi o aplicativo node-tiers [15], que contém múltiplos *endpoints*, cada um projetado para exercitar intensivamente um recurso específico do sistema (I/O, memória, CPU). Uma imagem Docker foi criada a partir deste aplicativo para facilitar a distribuição e execução uniforme das cargas de trabalho sintéticas em todos os contêineres do *cluster*. Todos os arquivos, incluindo código e versões, estão disponíveis no repositório público do Dockerhub [2].

Os contêineres foram configurados para executar a aplicação node-tiers, permitindo que cada contêiner executasse operações requisitadas via *endpoints* específicos. Essa configuração permitiu uma simulação precisa de diferentes tipos de cargas de trabalho em um ambiente distribuído, para impor cargas de trabalho que demandam intensivamente diferentes recursos do sistema, a simulação de um cenário de utilização de recursos envolvendo vários recursos é necessária para proporcionar uma representação mais realista e abrangente do comportamento de um sistema, pois aplicações reais geralmente demandam múltiplos recursos simultaneamente durante sua execução.

As cargas nomeadas ”I/O + I/O”, significam que todos os contêineres foram submetidos a um uso intensivo de disco, envolvendo operações significativas de escrita e leitura no banco de dados. Por outro lado, na configuração ”I/O + CPU”, metade dos contêineres foi designada para realizar operações intensivas de disco, enquanto a outra metade focou em tarefas intensivas de CPU, por meio da execução de algoritmos computacionalmente intensivos, como a aproximação do valor de PI utilizando séries matemáticas. Para a carga de memória, houve uma intensiva utilização do espaço de endereçamento de memória para execução de operações de leitura e escrita, simulando o acesso a estruturas de dados complexas e de grande volume.

A execução das cargas de trabalho foi automatizada por meio de um script Python, enviando um volume definido de requisições para os *endpoints* do node-tiers em cada contêiner [3]. Inicialmente, as cargas foram executadas sem intervenção na política padrão de escalonamento do Kubernetes, permitindo uma avaliação base da distribuição e gestão de contêineres. Posteriormente, a política de interferência foi aplicada para analisar as mudanças no comportamento do escalonamento e na utilização dos recursos.

Para garantir a consistência e a integridade dos dados em todas as fases dos testes de uso intensivo de recursos, o mesmo número de requisições foi mantido em cada etapa. Essa abordagem padronizada assegura que as variações observadas no desempenho e na eficiência dos recursos possam ser atribuídas diretamente às diferenças na política de escalonamento e na alocação de recursos, e não a variações na carga de

[2] https://hub.docker.com/r/josecolpo/node-tiers/tags
[3] https://github.com/jose-colpo/kubernetes_labels_interferencia

trabalho. Ao manter um volume constante de requisições através das fases de teste, a análise se torna mais robusta e confiável, permitindo uma comparação direta e justa entre diferentes configurações e políticas de gerenciamento de recursos.

A metodologia começou com a execução de apenas dois contêineres distribuídos em dois nós, de maneira isolada, permitindo uma análise detalhada do uso dos recursos sob condições controladas. Esta abordagem isolada foi crucial para entender o comportamento individual de cada contêiner e sua demanda específica por recursos. Sendo possível observar, com clareza, como cada tipo de carga de trabalho intensiva influenciava o uso dos recursos do sistema, Após essa fase, as cargas de trabalho foram executadas conjuntamente escalando progressivamente para 8, 16 e depois 32 contêineres.

Executar as cargas de trabalho em conjunto permitiu a avaliação do impacto da interferência entre os contêineres. Esta abordagem combinada ofereceu uma visão abrangente do desempenho do sistema sob condições de carga elevada, destacando os desafios e as oportunidades para otimização no escalonamento e na gestão de recursos em ambientes conteinerizados ao levar em conta a interferência.

O principal objetivo dessas simulações foi observar o comportamento do sistema sob uso intensivo de recursos em um ambiente conteinerizado, maximizando as demandas para identificar e quantificar a interferência. As métricas de interesse incluíram o *Makespan* (tempo total de execução das cargas de trabalho) e a utilização dos recursos dos nós, permitindo uma avaliação criteriosa da eficácia da política de escalonamento proposta em mitigar os efeitos negativos da interferência e melhorar o desempenho geral do sistema.

6.4 Apresentação dos dados coletados

Nessa seção exploramos as descobertas e avanços resultantes das simulações conduzidas para avaliar a eficácia de políticas de escalonamento na mitigação da interferência em ambientes conteinerizados. Através de uma análise detalhada, destacaremos não apenas os casos em que a aplicação de uma política de escalonamento de interferência resultou em melhorias notáveis na eficiência e na estabilidade dos contêineres, mas também situações em que o impacto foi menos pronunciado. Por meio dessa comparação, buscamos proporcionar uma compreensão abrangente de como diferentes cenários e configurações influenciam a dinâmica da interferência, e como estratégias eficazes de escalonamento podem ser empregadas para otimizar o desempenho em ambientes conteinerizados.

Os dados apresentados na tabela 6.1 refletem o *Makespan*, medido em segundos, para diferentes cargas de trabalho submetidas a uma intensidade de recursos variada em ambientes conteinerizados, especificamente sob as configurações de 8, 16 e 32 con-

Tabela 6.1 – *Makespan* em segundos das cargas de trabalho em diferentes quantidades de contêineres, sem intervenção na política de escalonamento tradicional do Kubernetes.

Carga de Trabalho Intensiva	**8 *Containers***	**16 *Containers***	**32 *Containers***
I/O + I/O	267	318	377
MEM + MEM	289	463	523
CPU + CPU	366	383	413
I/O + MEM	473	581	671
I/O + CPU	528	651	788
CPU + MEM	300	323	396

têineres em execução. Estes números representam o tempo total necessário para completar as tarefas designadas (uma por contêiner) utilizando a política de escalonamento padrão *(Default Scheduler)* do Kubernetes. É importante destacar que o foco desta politica padrão é na capacidade dos recursos disponíveis nos nós, para determinar a alocação de contêineres. No entanto, um aspecto crítico observado durante a inicialização dos contêineres sob esta política tradicional é a tendência do Kubernetes em agrupar múltiplos contêineres em um único nó, quando este parece ter capacidade de recursos suficiente. Esse comportamento ocorre porque a política padrão não considera os rótulos aplicados que poderiam indicar necessidades específicas de alocação.

Observa-se que, para cada tipo de recurso intensivo — seja I/O, memória ou CPU — o *Makespan* aumenta à medida que o número de contêineres cresce. Esta tendência é evidente nas combinações de carga de trabalho ”I/O + I/O”, onde as operações intensivas de entrada e saída elevam o tempo de execução de 267 segundos com 8 contêineres para 377 segundos com 32 contêineres. No caso da configuração ”CPU + CPU”, que implica em cargas de trabalho intensivas em processamento, a diferença não é marcante em comparação com outras combinações de recursos intensivos. Esse padrão sugere que, para cargas de trabalho intensivas em CPU, o sistema conseguiu manter um nível de desempenho relativamente estável, mesmo com um aumento na densidade de contêineres.

Tabela 6.2 – *Makespan* em segundos das cargas de trabalho em diferentes quantidades de contêineres, com execução da política de escalonamento baseada em interferência.

Carga de Trabalho Intensiva	**8 *Containers***	**16 *Containers***	**32 *Containers***
I/O + I/O	191	224	184
MEM + MEM	285	422	466
CPU + CPU	346	364	381
I/O + MEM	415	426	477
I/O + CPU	495	438	471
CPU + MEM	273	285	294

Na tabela 6.2 é possível visualizar o tempo de execução das cargas de trabalho após a aplicação da política de interferência, os dados do *Makespan* mostram resultados

notáveis em diferentes configurações de contêineres em execução. A configuração 'I/O + I/O' demonstra uma tendência única, onde o *Makespan* diminui de 224 segundos com 16 contêineres para 184 segundos com 32 contêineres, sugerindo uma eficiência notável da política de interferência em cenários de alta densidade de contêineres com uso intensivo de disco, devido ao balanceamento equilibrado dos contêineres entre os nós, além disso as plataformas de conteinerização podem ter mecanismos de *cache* inteligentes que se tornam mais eficazes quando há mais operações de I/O, potencialmente reduzindo o tempo total de execução conforme as operações de I/O aumentam. Outro ponto importante é que quando há menos contêineres em execução, eles podem tentar usar mais recursos do que lhes é permitido, o que pode resultar em limitações e espera, enquanto um número maior de contêineres pode levar a um uso mais equilibrado e a menos contenção dos recursos.

Em contrapartida, nos casos de "CPU + CPU" e "MEM + MEM", os incrementos moderados no *Makespan* apontam para uma menor eficácia relativa da política de interferência em lidar com a concorrência por essa combinação de recursos, sugerindo que tais configurações podem ser menos suscetíveis a interferências ou mais facilmente otimizadas pela política tradicional do Kubernetes.

Tabela 6.3 – Porcentagem de aumento no tempo de conclusão das tarefas sem a execução da política escalonamento que leva em consideração a interferência.

Carga de Trabalho Intensiva	**8 *Containers***	**16 *Containers***	**32 *Containers***
I/O + I/O	39,8%	42,0%	104,9%
MEM + MEM	1,4%	9,7%	12,2%
CPU + CPU	5,8%	5,2%	8,4%
I/O + MEM	14,0%	36,4%	40,7%
I/O + CPU	6,7%	48,6%	67,3%
CPU + MEM	9,9%	13,3%	34,7%

Os dados apresentados na tabela 6.3 destacam a porcentagem de aumento do *Makespan* em diferentes configurações de contêineres em execução, na ausência de uma política de escalonamento baseada em interferência. Observações chave revelam tendência sobre o impacto da interferência nos diversos tipos de cargas de trabalho intensivas.

As combinações que envolvem operações intensivas de disco, como "I/O + I/O", exibem uma degradação marcante, especialmente com 32 contêineres em execução, atingindo 104,9%. Sugerindo que a interferência no acesso ao disco é particularmente problemática, levando a um aumento substancial no tempo de execução.

As cargas de trabalho que demandam intensivamente CPU ("CPU + CPU") ou memória ("MEM + MEM") individualmente apresentam uma degradação relativamente baixa, com variações menores nos percentuais à medida que o número de contêineres aumenta.

Indicando que, embora exista interferência, os sistemas são capazes de gerenciar efetivamente a concorrência por esses recursos isoladamente.

Quando CPU e memória são demandadas conjuntamente em ”CPU + MEM”, observa-se uma degradação notável, particularmente com 32 contêineres (34,7%). Isso ressalta que a interferência se torna mais evidente quando múltiplos tipos de recursos são intensivamente requisitados simultaneamente.

Em geral, todas as cargas de trabalho apresentam uma tendência de aumento na porcentagem de degradação à medida que o número de contêineres cresce, sublinhando a relação direta entre a densidade de contêineres e o impacto da interferência no desempenho.

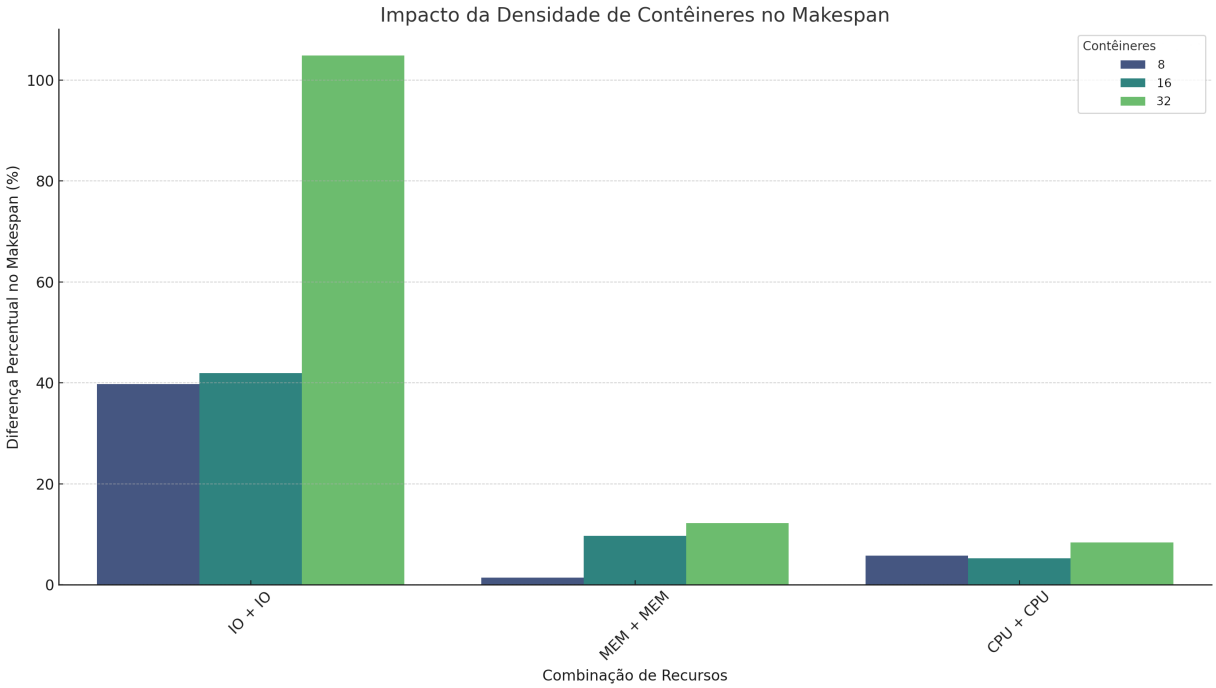

Figura 6.2 – Impacto no *Makespan* por contêineres, sem a política de escalonamento proposta, para ”I/O + I/O”, ”MEM + MEM” e ”CPU + CPU”.

Na figura 6.2 está a análise dos dados referentes ao impacto no prolongamento do tempo de execução por quantidade de contêineres em execução, sem a aplicação da política de escalonamento proposta, para três cenários distintos de carga de trabalho intensiva: ”I/O + I/O”, ”MEM + MEM” e ”CPU + CPU”. Os dados revelam uma tendência clara de aumento do *Makespan* à medida que o número de contêineres cresce. Especificamente, a combinação ”I/O + I/O” demonstrou uma sensibilidade particularmente alta à densidade de contêineres. Enquanto isso, ”MEM + MEM” e ”CPU + CPU” também exibiram incrementos na degradação do desempenho com o aumento de contêineres, embora em menor escala, sugerindo uma melhor capacidade de gerenciamento da memória e da CPU em cenários de alta densidade.

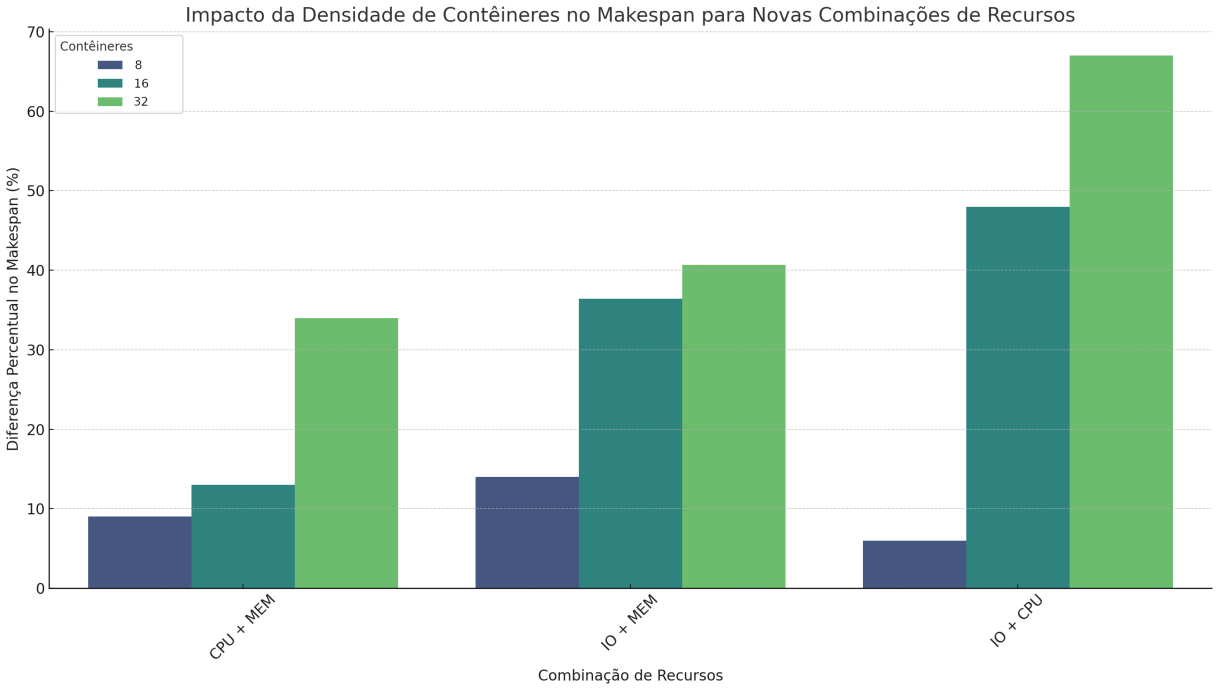

Figura 6.3 – Impacto no *Makespan* por contêineres, sem a política de escalonamento proposta, para ”CPU + MEM” e combinações de recursos com I/O.

A análise do impacto da densidade de contêineres nas combinações de cargas de trabalho ”CPU + MEM”, ”I/O + MEM” e ”I/O + CPU” revelam padrões distintos de degradação do *Makespan*. No caso de ”CPU + MEM”, a combinação de demandas por processamento e memória mostrou uma escalada moderada na degradação à medida que o número de contêineres aumentou, indicando uma contenção significativa quando ambos os recursos são intensivamente requisitados.

Por sua vez, ”I/O + MEM” apresentou um aumento mais acentuado no *Makespan*, evidenciando a sensibilidade desta configuração à densidade de contêineres, possivelmente devido à complexa interação entre operações de memória e I/O. A configuração ”I/O + CPU”, responsável pelas cargas intensivas de I/O e processamento, exibiu o impacto mais pronunciado, com um aumento substancial no tempo de execução em cenários de alta densidade de contêineres.

Esses resultados destacam os desafios associados à otimização de ambientes que simultaneamente impõem alta demanda em múltiplos tipos de recursos, principalmente quando combinados com operações em disco.

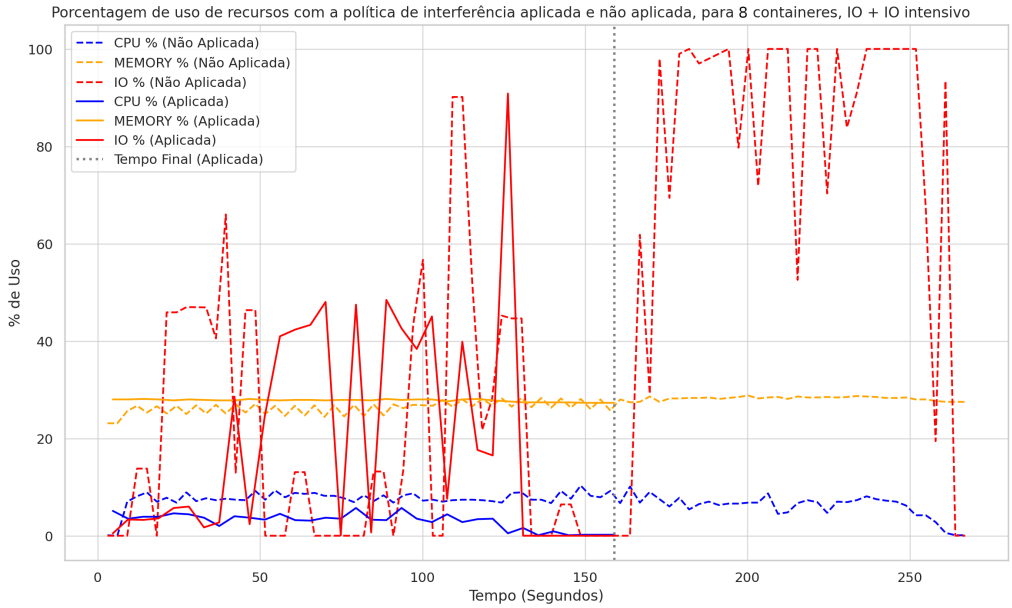

Figura 6.4 – Comparação uso de recursos, carga de trabalho ”I/O + I/O”, 8 contêineres.

O gráfico na figura 6.4 mostra a porcentagem de uso de recursos em um ambiente de teste com oito contêineres e carga de trabalho ”I/O + I/O”, comparando o uso ao longo do tempo dos recursos da política tradicional do Kubernetes contra a customizada de interferência. As linhas tracejadas representam a utilização dos recursos quando a política de interferência não é aplicada, e as linhas contínuas quando a política está em vigor. O *Makespan* é indicado pela linha pontilhada vertical dando ênfase no momento que a execução da carga de trabalho com a politica de interferência é finalizada. Há uma redução notável na utilização de I/O com a política customizada aplicada, indicando que a política pode estar efetivamente mitigando a interferência no uso desse recurso. Já os recursos CPU e memória são menos voláteis com a política customizada aplicada.

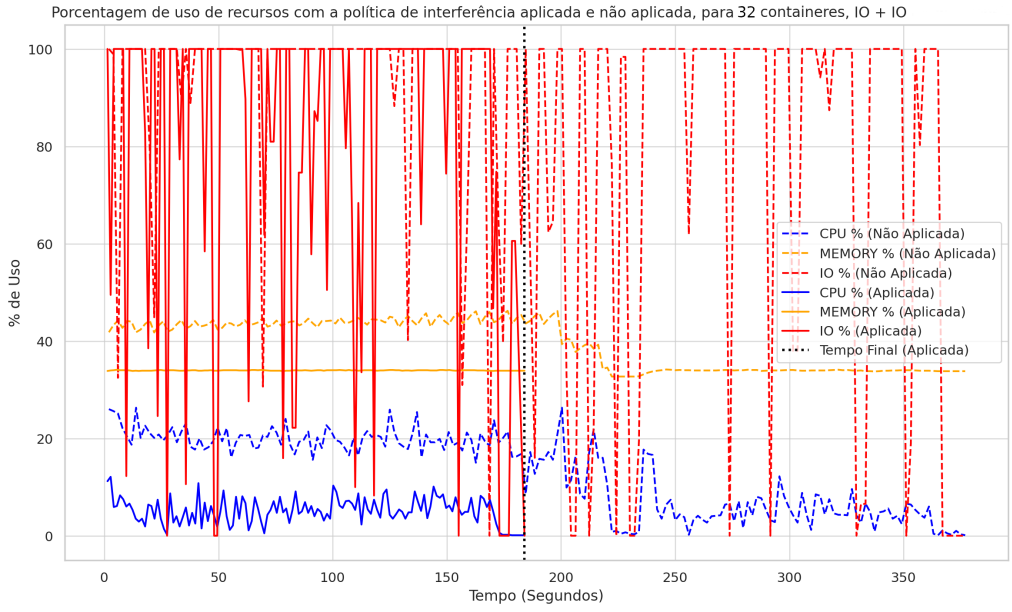

Figura 6.5 – Comparação uso de recursos, carga de trabalho ”I/O + I/O”, 32 contêineres.

Analisando o gráfico na figura 6.5 de uso intensivo de I/O em um ambiente com 32 contêineres em execução e comparando-o com o cenário anterior de 8 contêineres, observam-se diferenças significativas no comportamento do sistema na utilização intensiva de recursos, principalmente de disco. No ambiente com 32 contêineres, a utilização de I/O mostra picos muito altos e frequentes, indicando períodos intensos de atividade de disco e potencial contenção de I/O. Com a política de interferência aplicada, ainda há picos, mas são menos frequentes e com menor amplitude, sugerindo que a política contribui para mitigar a interferência.

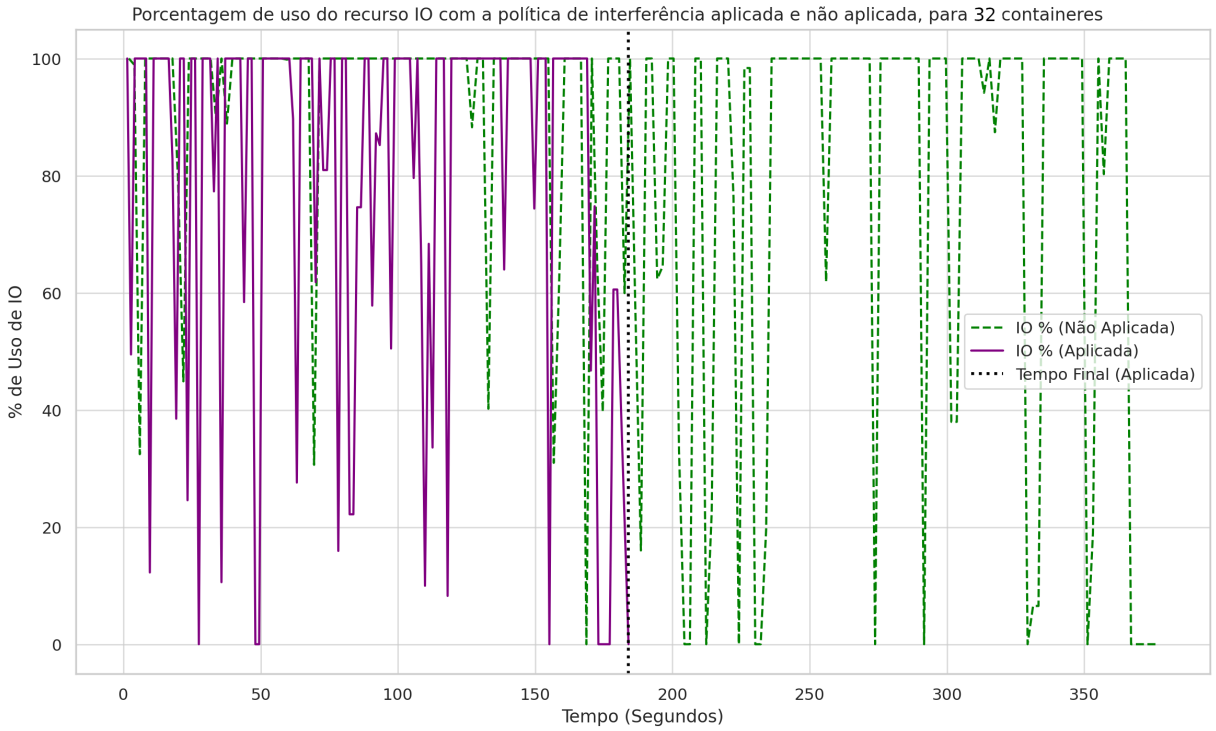

Figura 6.6 – Comparação com a política de interferência aplicada e não aplicada, apenas do uso de I/O ao longo da execução da carga de trabalho intensiva ”I/O + I/O”, em 32 contêineres.

O gráfico da figura 6.6 fornece uma visão detalhada do uso do recurso de I/O da carga de trabalho intensiva ”I/O + I/O”, em um ambiente com 32 contêineres em execução, destacando a eficácia da política de interferência aplicada especificamente sobre as operações de disco. A linha verde tracejada representa a utilização de I/O sem a política de interferência, enquanto a linha roxa sólida indica utilização com a política de interferência aplicada.

Um aspecto notável é a frequência com que a política aplicada conseguiu mitigar os picos de utilização de I/O que atingiam 100%. Com a política em vigor, é evidente uma redução significativa na ocorrência desses picos máximos, o que sugere uma gestão eficaz das operações de I/O, evitando o esgotamento da capacidade do disco e possíveis degradações no desempenho. Essa mitigação é crítica, pois os picos de 100% de utiliza-

ção podem levar a um aumento do tempo de espera das operações de I/O, atrasando a execução de outras tarefas e, consequentemente, prolongando o *Makespan*.

A capacidade da política de interferência em manter a utilização de I/O abaixo dos limites máximos contribui diretamente para diminuição do tempo de processamento total, pois assegura que as operações de leitura e escrita sejam executadas de forma mais fluida e com menos interrupções. Isso é ilustrado pela linha pontilhada que marca o fim da execução quando a política de interferência está aplicada, onde se pode observar que o término das operações ocorre de forma mais precoce em comparação com a ausência da política, refletindo um desempenho aprimorado e uma eficiência operacional elevada.

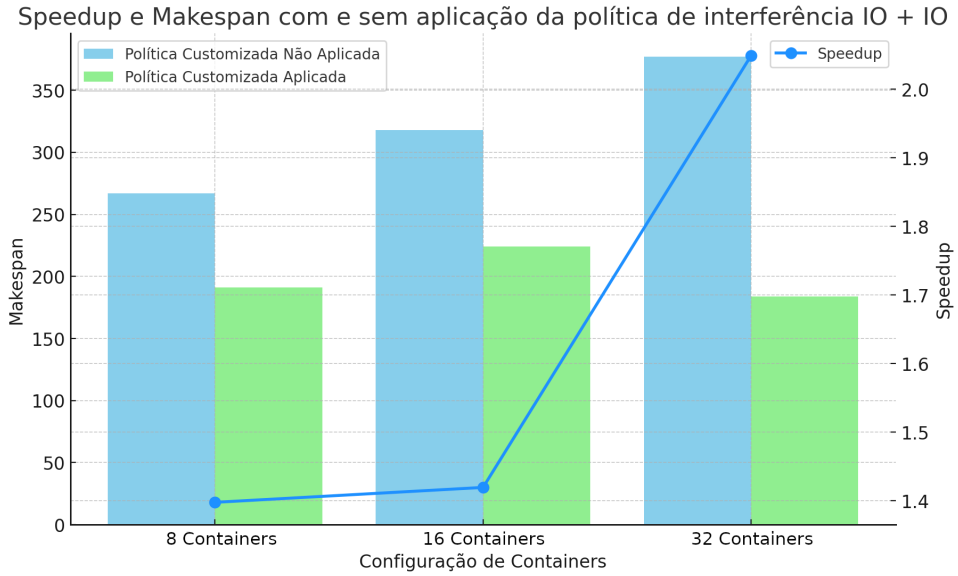

Figura 6.7 – Métrica *Speedup* comparando a melhoria no tempo de execução (*Makespan*), da carga de trabalho ”I/O + I/O” para diferentes quantidades de contêineres em execução.

O gráfico apresentado na figura 6.7 compara o *Makespan* e o Speedup de cargas de trabalho intensivas em I/O, em cenários com 8, 16 e 32 contêineres em execução, com e sem a aplicação da política de interferência. As barras azuis representam o *Makespan* sem a política de interferência aplicada, e as verdes com ela aplicada. A linha com marcadores de círculo azul representa o *Speedup* alcançado com a política aplicada em relação ao cenário sem a política.

Há uma diminuição notável no *Makespan* com a política de interferência aplicada em todas as configurações de contêineres, indicando que a política contribui para a eficiência na execução das cargas de trabalho. O *Speedup* aumenta consistentemente à medida que o número de contêineres cresce, evidenciando que a política de interferência tem um impacto proporcionalmente maior em ambientes com maior densidade de contêineres.

Figura 6.8 – Uso de recursos, carga de trabalho ”CPU + CPU”, oito contêineres.

O gráfico na figura 6.8 exibe a utilização dos recursos de CPU e memória em um cenário de testes com oito contêineres, enfrentando cargas intensivas de CPU. Nota-se uma estabilidade notável na utilização da memória ao longo do teste, com a política de interferência aplicada e sem ela, evidenciando que as operações intensivas de CPU não impactaram significativamente o uso de memória. Isso sugere uma baixa correlação entre a intensidade das operações de CPU e a demanda por memória. O uso do recurso CPU também demonstrou um impacto relativo baixo sob uso intensivo, contribuindo para um controle mais refinado das flutuações na sua utilização.

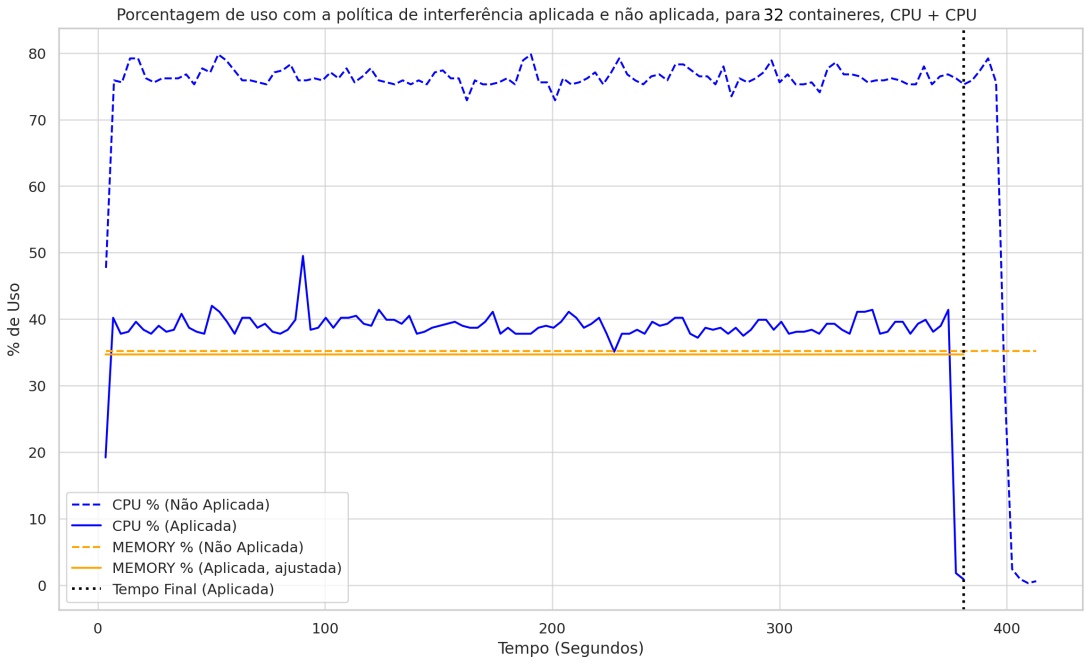

Figura 6.9 – Uso de recursos, carga de trabalho ”CPU + CPU”, 32 contêineres.

O gráfico na figura 6.9 mostra a porcentagem de uso de utilização de recursos durante a execução da carga de trabalho ”CPU + CPU,” com 32 contêineres em execução. Em contraste com a mesma carga de trabalho em menor número de contêineres em execução, o gráfico de 32 contêineres exibe uma linha de utilização de CPU relativamente estável com a política de interferência aplicada. A comparação entre as diferentes densidades de contêineres em execução para essa carga de trabalho indicam que os efeitos podem ser mais pronunciados em ambientes com maior número de contêineres, onde a contenção de recursos é mais provável.

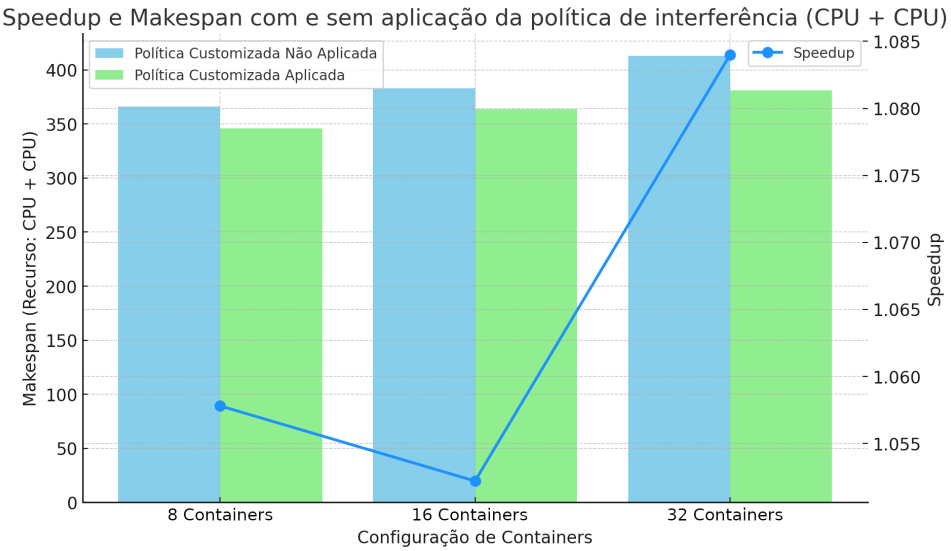

Figura 6.10 – Métrica *Speedup* comparando a melhoria no tempo de execução (*Makespan*), da carga de trabalho ”CPU + CPU” para diferentes quantidades de contêineres em execução.

O gráfico mostrado na figura 6.10 revela que o *Speedup* é ligeiramente maior para 8 contêineres em comparação com 16, o que pode ser atribuído a uma eficiência operacional há menos contêineres competindo por recursos de CPU. A diferença no *Makespan* e no *Speedup* não foi significativa para o uso intensivo desse recurso, o que sugere que a política de interferência pode ter uma capacidade limitada de otimizar ainda mais um recurso que já é eficientemente gerenciado pelo escalonador padrão do Kubernetes. A CPU é frequentemente o recurso menos afetado pela interferência devido ao design moderno dos sistemas de computação, que inclui técnicas avançadas de multitarefa e escalonamento de processos, permitindo um uso mais eficiente e equitativo dos núcleos de CPU disponíveis, mesmo sob carga pesada.

Esse comportamento indica que, embora a política de interferência possa oferecer melhorias, sua eficácia é mais restrita para cargas de trabalho intensivas em CPU, possivelmente devido à natureza menos conflitante das operações de CPU em comparação com outros recursos como I/O, onde a contenção e a interferência são mais perceptíveis e têm um impacto maior no desempenho do sistema.

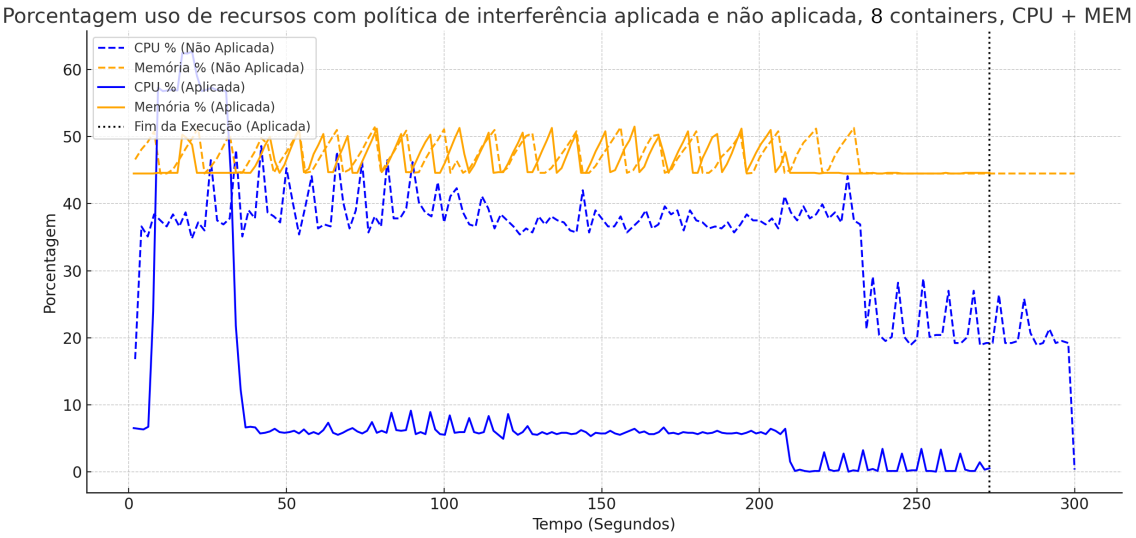

Figura 6.11 – Comparação com a política de interferência aplicada e não aplicada, do uso de recursos ao longo da execução da carga de trabalho intensiva ”CPU + MEM”, em oito contêineres.

O gráfico na figura 6.11 fornece uma representação visual do uso de recursos em um ambiente com oito contêineres, focando na combinação de cargas de trabalho de CPU e memória. O pico inicial no uso de CPU, que ocorre no início da execução da carga de trabalho, pode ser explicado pelo aumento temporário na demanda de processamento pelas aplicações, mas é rapidamente mitigado, resultando em uma estabilização da utilização de CPU em níveis inferiores quando a política de interferência está ativa.

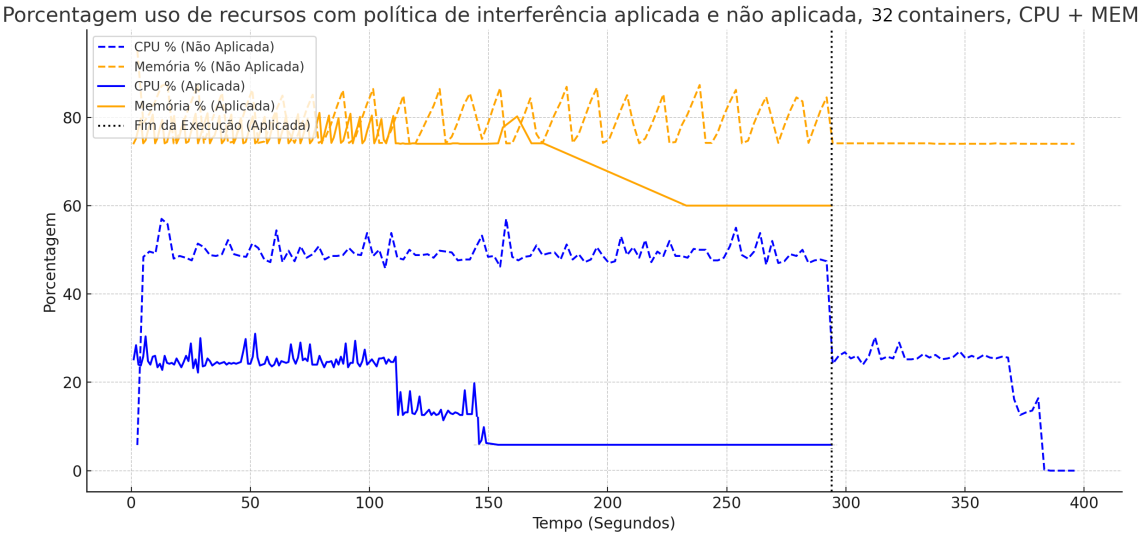

Figura 6.12 – Comparação com a política de interferência aplicada e não aplicada, do uso de recursos CPU e memória ao longo da execução da carga de trabalho intensiva ”CPU + MEM”, em 32 contêineres.

Ao comprar os dados mostrados da figura 6.12 de utilização de recursos durante execução da carga de trabalho ”CPU + MEM” em um ambiente com 8 contêineres, com

o cenário de 32 contêineres, mantém-se a tendência observada de que a memória não é substancialmente influenciada pela densidade dos contêineres, mesmo com a política de interferência aplicada. Indicando que o consumo de memória permanece relativamente estável, independentemente do número de contêineres.

Além disso, em contraste com o cenário de 8 contêineres, onde um pico de utilização de CPU foi observado, o gráfico atual para 32 contêineres não mostra picos de CPU com a política de interferência aplicada. Essa ausência de picos reflete uma maior estabilidade na utilização da CPU, o que pode ser atribuído à eficiência da política de interferência, especialmente em ambientes com uma maior densidade de contêineres. A política parece regular de maneira eficaz a alocação de CPU, evitando os extremos de utilização que podem levar à contenção de recursos e prejudicar o desempenho geral.

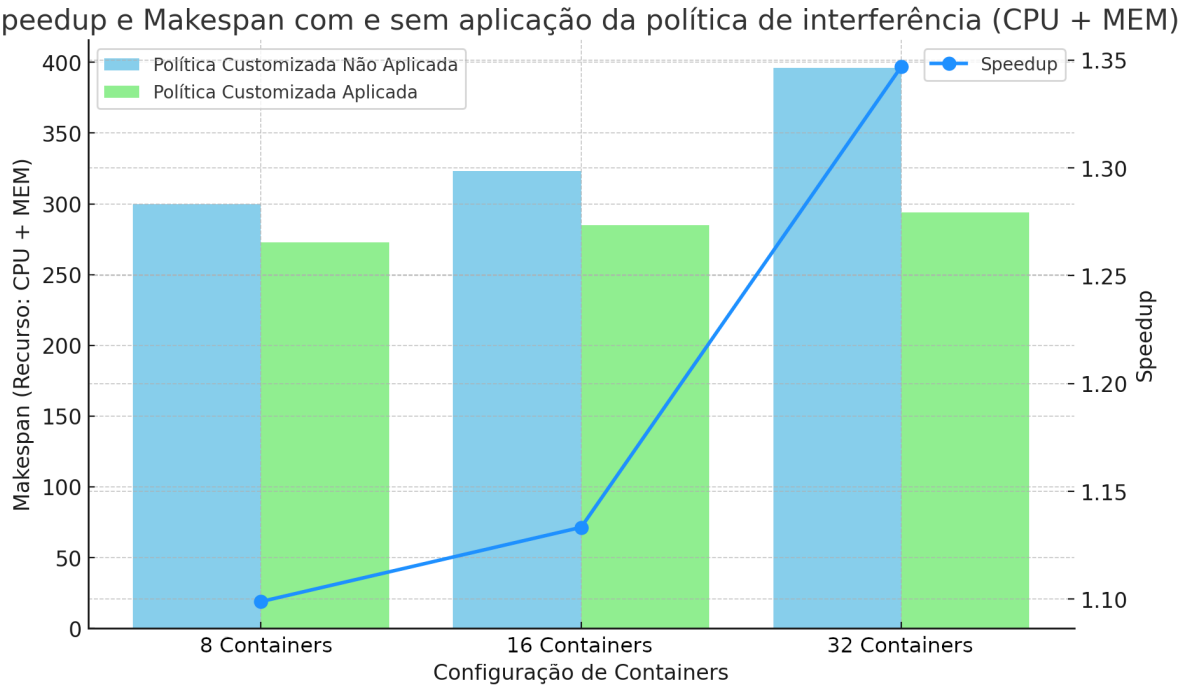

Figura 6.13 – Métrica *Speedup* comparando a melhoria no tempo de execução (*Makespan*), da carga de trabalho ”CPU + MEM” para diferentes quantidades de contêineres em execução.

Com base nos dados exibidos na figura 6.13, é notável que a combinação de CPU e memória atingiu um *Speedup* melhor com a política de interferência aplicada em comparação com o uso isolado da CPU ou da memória. Esse fenômeno pode ser atribuído a uma interação mais complexa entre as operações, onde a interferência pode ter um impacto mais pronunciado no desempenho devido à contenção simultânea de múltiplos recursos. A política de interferência, ao ser aplicada, efetivamente alivia a contenção de recursos, permitindo que as operações de CPU e memória sejam executadas com mais eficiência.

A eficácia da política, portanto, destaca-se particularmente quando dois recursos críticos estão sendo intensivamente utilizados, sublinhando a importância de estratégias de escalonamento que consideram múltiplos aspectos do uso de recursos em ambientes conteinerizados.

6.5 Análise dos dados coletados

Os dados coletados revelam entendimentos cruciais sobre a dinâmica de escalonamento em sistemas conteinerizados. Primeiramente, observou-se que as combinações de cargas de trabalho que envolvem uso intensivo de disco (I/O) foram as que mais sofreram com o impacto da interferência, resultando em uma degradação significativa na performance. Esse fenômeno destaca o desafio do gerenciamento de operações de I/O em ambientes de alta densidade de contêineres, onde a contenção de acesso aos dispositivos de armazenamento pode levar a um aumento significativo no tempo de execução das tarefas.

Por outro lado, cargas de trabalho que demandam recursos isolados de CPU e memória mostraram-se mais eficazes na mitigação da interferência, indicando que esses recursos podem ser gerenciados de maneira mais eficiente pelo escalonador tradicional, mesmo sob carga intensa. Sugerindo que a política de interferência padrão do Kubernetes já possui mecanismos robustos para lidar com a alocação desses recursos de maneira a minimizar a competição e o bloqueio entre contêineres concorrentes. Por outro lado a combinação de cargas de trabalho que simultaneamente utilizam esses dois recursos CPU e memória, apresentou um cenário desafiador para a política de escalonamento tradicional do Kubernetes. Os dados demonstram que essa política padrão não foi capaz de mitigar a interferência de maneira eficaz nessas circunstâncias, registrando casos de aumento de até 34,7% no tempo de execução.

Além disso, a implementação de uma política de escalonamento mais inteligente no Kubernetes propiciou resultados promissores, alcançando uma melhoria de até 104,9% na performance em alguns casos. Esse avanço significativo é uma evidência do potencial da política de interferência para otimizar a utilização dos recursos em ambientes conteinerizados.

Essas evidências sublinham a importância de estratégias adaptativas e inteligentes que considerem o comportamento específico das cargas de trabalho e a configuração do sistema. Esses achados são fundamentais para aprimorar as abordagens de escalonamento e garantir a eficiência e a confiabilidade de sistemas conteinerizados em escala.

7. CONCLUSÃO

Este trabalho buscou validar estratégias de gerenciamento de recursos em ambientes conteinerizados, especificamente no contexto do Docker e Kubernetes. Ao longo do estudo, exploramos a interferência entre aplicações compartilhando diferentes tipos de recursos e densidade de contêineres, um desafio muitas vezes negligenciado pelas políticas tradicionais de alocação de recursos, que se concentram apenas em capacidades individuais, como CPU e memória.

A implementação desenvolvida, baseada na classificação e identificação da interferência por meio de rótulos (*labels*) no Kubernetes, apresentou resultados promissores. Ao priorizar a mitigação da interferência e ao alocar recursos de maneira inteligente, observamos melhorias significativas no desempenho e na estabilidade das aplicações.

Ao adaptar uma estratégia prévia de gerenciamento de recursos para o ambiente conteinerizado, identificamos que a análise detalhada dos rótulos atribuídos aos Nós desempenha um papel crucial na alocação eficiente de recursos. Essa abordagem permite uma distribuição mais balanceada dos recursos, reduzindo degradações no desempenho e melhorando a utilização dos recursos do *cluster* Kubernetes.

Os resultados obtidos na simulação de cargas de trabalho intensivas destacaram a importância de considerar a interferência entre os recursos ao realizar o escalonamento de contêineres. A política de escalonamento customizada demonstrou capacidade para mitigar a interferência, proporcionando um ambiente mais estável e confiável para as aplicações em execução.

É importante salientar que este estudo oferece uma base sólida para futuras pesquisas no campo do gerenciamento de recursos conteinerizados, como a atribuição de interferência dinâmica. Dessa forma, este trabalho não apenas oferece *insights* valiosos sobre a importância da consideração da interferência na alocação de recursos, mas também serve como um ponto de partida para o desenvolvimento de políticas de escalonamento mais eficazes e adaptáveis em ambientes de contêineres, contribuindo para um melhor aproveitamento dos recursos disponíveis e para o aprimoramento da eficiência das aplicações em larga escala.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Alves, M.; Drummond, L. “A multivariate and quantitative model for predicting cross-application interference in virtual environments”, *Journal of Systems and Software*, vol. 128, 04 2017.

[2] Alves, M.; Teylo, L.; Frota, Y.; Drummond, L. “An interference-aware virtual machine placement strategy for high performance computing applications in clouds”, 2018, pp. 94–100.

[3] Bader, J.; Thamsen, L.; Kulagina, S.; Will, J.; Meyerhenke, H.; Kao, O. “Tarema: Adaptive resource allocation for scalable scientific workflows in heterogeneous clusters”. In: 2021 IEEE International Conference on Big Data (Big Data), 2021, pp. 65–75.

[4] Bu, X.; Rao, J.; Xu, C.-Z. “Interference and locality-aware task scheduling for mapreduce applications in virtual clusters”, *HPDC 2013 - Proceedings of the 22nd ACM International Symposium on High-Performance Parallel and Distributed Computing*, 06 2013.

[5] Docker. “Get started and overview”. Capturado em: https://docs.docker.com/get-started/overview/, Dez 2023.

[6] Figueiredo, R.; Subratie, K. “Demo: Software-defined virtual networking across multiple edge and cloud providers with edgevpn.io”. In: 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS), 2021, pp. 1088–1090.

[7] Guan, W.; Ababei, C. “Unified cross-layer cluster-node scheduling for heterogeneous datacenters”. In: 2022 IEEE 13th International Green and Sustainable Computing Conference (IGSC), 2022, pp. 1–8.

[8] Inc., D. “Docker hub container image library | app containerization”. Capturado em: https://hub.docker.com/, Dez 2023.

[9] Jabbari, A.; Masoumiyan, F.; Hu, S.; Tang, M.; Tian, Y.-C. “A cost-efficient resource provisioning and scheduling approach for deadline-sensitive mapreduce computations in cloud environment”. In: 2021 IEEE 14th International Conference on Cloud Computing (CLOUD), 2021, pp. 600–608.

[10] Jing Hui, A. N.; Lee, B. S. “Epsilon: A microservices based distributed scheduler for kubernetes cluster”. In: 2021 18th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2021, pp. 1–6.

[11] Ju, L.; Singh, P.; Toor, S. “Proactive autoscaling for edge computing systems with kubernetes”. In: Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion, 2022.

[12] Kautsar, I. A.; Maika, M. R.; Budiman, A. N.; Setyawan, A. B.; Awali, J. Y. “Microservice based architecture: The development of rapid prototyping supportive tools for project based learning”. In: 2023 IEEE World Engineering Education Conference (EDUNINE), 2023, pp. 1–6.

[13] Kubernetes. “Concepts, overview and components”. Capturado em: https://kubernetes.io/docs/concepts/overview/components/, Dez 2023.

[14] Kubernetes. “Security and controlling access”. Capturado em: https://kubernetes.io/pt-br/docs/concepts/security/controlling-access/, Dez 2023.

[15] Ludwig, U. “Node-tiers: A multi-tier benchmark”. Capturado em: https://github.com/uillianluiz/node-tiers, Dez 2023.

[16] Ludwig, U.; Xavier, M.; Kirchoff, D.; Cezar, I.; De Rose, C. “Optimizing multi-tier application performance with interference and affinity-aware placement algorithms”, *Concurrency and Computation: Practice and Experience*, vol. 31, 1 2019, pp. e5098.

[17] Meyer, V.; da Silva, M. L.; Kirchoff, D. F.; De Rose, C. A. “Iada: A dynamic interference-aware cloud scheduling architecture for latency-sensitive workloads”, *Journal of Systems and Software*, vol. 194, 2022, pp. 111491.

[18] Saha, P.; Beltre, A.; Uminski, P.; Govindaraju, M. “Evaluation of docker containers for scientific workloads in the cloud”. In: Proceedings of the Practice and Experience on Advanced Research Computing, 2018.

[19] Shah, A.; Wolf, F.; Zhumatiy, S.; Voevodin, V. “Capturing inter-application interference on clusters”. In: 2013 IEEE International Conference on Cluster Computing (CLUSTER), 2013, pp. 1–5.

[20] Tzenetopoulos, A.; Masouros, D.; Xydis, S.; Soudris, D. “Interference-aware orchestration in kubernetes”. In: High Performance Computing, Jagode, H.; Anzt, H.; Juckeland, G.; Ltaief, H. (Editores), 2020, pp. 321–330.

[21] Xavier, M.; Cano, C.; Meyer, V.; De Rose, C. “Intp: Quantifying cross-application interference via system-level instrumentation”, 2022, pp. 231–240.

[22] Xavier, M. G. “Data processing with cross-application interference control via system-level instrumentation”, Tese de Doutorado, Pontifícia Universidade Católica do Rio Grande do Sul, 2019, 100p.

[23] Zhang, W.; Rajasekaran, S.; Wood, T.; Zhu, M. “Mimp: Deadline and interference aware scheduling of hadoop virtual machines”. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2014, pp. 394–403.

[24] Zhu, Q.; Tung, T. “A performance interference model for managing consolidated workloads in qos-aware clouds”. In: 2012 IEEE Fifth International Conference on Cloud Computing, 2012, pp. 170–179.

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