Estratégia de alocação dinâmica de recursos no Kubernetes para ambientes multi-inquilinos

Gabriel Maciel, Carlos Eduardo, Raphael Gomes, Leandro Alexandre, Antonio Oliveira

***Resumo*— A alocação dinâmica de recursos em ambientes compartilhados apresenta desafios significativos, surgindo a necessidade de soluções que possam gerenciar a alocação atendendo os requisitos conflitantes entre os inquilinos. Este trabalho propõe uma estratégia de multi-inquilino para o Kubernetes, que incorpora um algoritmo multicritério baseado em *Technique for Order of Preference by Similarity to Ideal Solution* (TOPSIS) para a alocação dinâmica de recursos entre os inquilinos. Os resultados obtidos demonstraram a eficácia da solução, destacando sua capacidade de alocação de recursos em ambientes multi-inquilinos, evidenciando a melhoria significativa na eficiência operacional e na utilização dos recursos.**

***Palavras-Chave*— Kubernetes, Alocação de Recursos, Multi-inquilino, TOPSIS, Ambientes Compartilhados.**

***Abstract*— Dynamic resource allocation in shared environments presents significant challenges, requiring solutions that can handle conflicting tenant requirements. This work proposes a multi-tenant strategy for Kubernetes, integrating a multi-criteria algorithm based on the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) for dynamic resource allocation. The results demonstrate the effectiveness of the proposed solution, highlighting its ability to manage resource allocation in multi-tenant environments, with significant improvements in operational efficiency and resource utilization.**

***Keywords*— Kubernetes, Resource Allocation, Multi-Tenant, TOPSIS, Shared Environments.**

I. Introdução

No ecossistema dinâmico da computação, o Kubernetes emergiu como uma plataforma essencial para orquestrar e escalar aplicações [1]. À medida que sua adoção cresce, surgem desafios no gerenciamento eficiente de ambientes multi-inquilinos, exigindo soluções dinâmicas para otimizar a alocação de recursos, como CPU e memória [1]. O objetivo é garantir equidade, eficiência, isolamento e escalabilidade para aplicações concorrentes no cluster [2].

A arquitetura modular, com suporte a *Custom Resource Definitions* (CRDs), facilita a integração de soluções personalizadas. No Kubernetes, o escalonamento utiliza mecanismos como *Resource Requests* e *Limits* para alocar Pods em nós disponíveis, oferecendo previsibilidade e simplicidade. No entanto, em cenários de carga dinâmica, esses mecanismos podem levar à subutilização ou superdimensionamento de recursos, impactando a eficiência e os custos [3].

Este trabalho propõe a aplicação do método *Technique for Order of Preference by Similarity to Ideal Solution* (TOPSIS) para a alocação de recursos, integrado ao Kubernetes via CRDs. O TOPSIS permite decisões multicritério, considerando múltiplos fatores, e avalia a melhor opção com base na proximidade de uma solução ideal [4].

Em ambientes multi-inquilinos, é essencial garantir equidade, desempenho e isolamento entre inquilinos[3]. O *Horizontal Pod Autoscaler* (HPA), embora amplamente utilizado no Kubernetes, realiza apenas ajustes automáticos com base em métricas simples, como CPU e memória, e geralmente não atende cenários com múltiplos critérios e demandas dinâmicas, podendo resultar em subutilização de recursos ou alocação desigual sob cargas variáveis. Já o método TOPSIS possibilita decisões mais completas ao considerar, simultaneamente, critérios como CPU, memória, políticas de rede e latência. A escolha desse método se apoia na sua robustez comprovada na literatura para decisões com múltiplos requisitos conflitantes. Diante dessas limitações, diversas soluções alternativas têm sido propostas na literatura, discutidas a seguir.

A. Trabalhos Relacionados

Dentre os trabalhos encontrados, ressalta-se [5], que propõe a criação de clusters virtuais para isolar inquilinos, promovendo segurança e confidencialidade das cargas de trabalho. Essa abordagem reduz custos operacionais ao permitir o gerenciamento de múltiplos clusters virtuais em um único cluster físico.

A solução [6], oferece funcionalidades avançadas para gerenciamento de inquilinos, com políticas flexíveis e granularidade no controle de acesso e recursos. Sua automação reduz a carga operacional, aumentando a eficiência.

O trabalho [3], um meta-agendador que utiliza métricas como demanda de recursos e tempo médio de espera para melhorar a equidade na alocação em clusters multi-inquilinos. Ele complementa o escalonador padrão do Kubernetes com políticas orientadas à justiça .

Em [7], os autores descrevem a necessidade de isolamento entre inquilinos, especialmente em termos de rede, para garantir segurança e privacidade. Apesar dos avanços, a solução apresenta limitações em testes de desempenho.

Este artigo está estruturado da seguinte forma: a Seção II detalha a estratégia proposta; A Seção III detalha a metodologia utilizada; a Seção IV apresenta os resultados; e a Seção V oferece as conclusões e sugestões para futuros trabalhos.

Gabriel Eduardo, Instituto de Informática, Universidade Federal de Goiás (UFG), GO, Brasil, e-mail: gabrieleduardo10102@gmail.com; Carlos Eduardo, Instituto Federal de Educação, Ciência e Tecnologia do Tocantins (IFTO), TO, Brasil, e-mail: carlosedu@ifto.edu.br; Raphael Gomes, Instituto Federal de Educação, Ciência e Tecnologia do Goiás (IFG), GO, Brasil, e-mail: raphael.gomes@ifg.edu.br; Leandro Alexandre, Instituto Federal de Educação, Ciência e Tecnologia do Goiás (IFG), GO, Brasil, e-mail: leandro.freitas@ifg.edu.br; Leandro Alexandre, Instituto de Informática, Universidade Federal de Goiás (UFG), GO, Brasil, e-mail: antoniojr@ufg.br.

II. Proposta da Solução

A proposta deste trabalho visa aprimorar a alocação de inquilinos em ambientes Kubernetes, por meio da criação de um CRD denominado EMK. O EMK incorpora um algoritmo de decisão multicritério, baseado no método TOPSIS, desenvolvido para otimizar a alocação de inquilinos com base em diversos fatores, como utilização de recursos e requisitos de desempenho. O objetivo é proporcionar uma alocação dinâmica e eficiente, garantindo equidade entre os inquilinos e maximizando a utilização dos recursos disponíveis no *cluster* Kubernetes. Os CRDs são extensões customizadas que expandem as funcionalidades do Kubernetes, permitindo a automação de tarefas operacionais complexas, como provisionamento, escalonamento e atualização de aplicações [8].

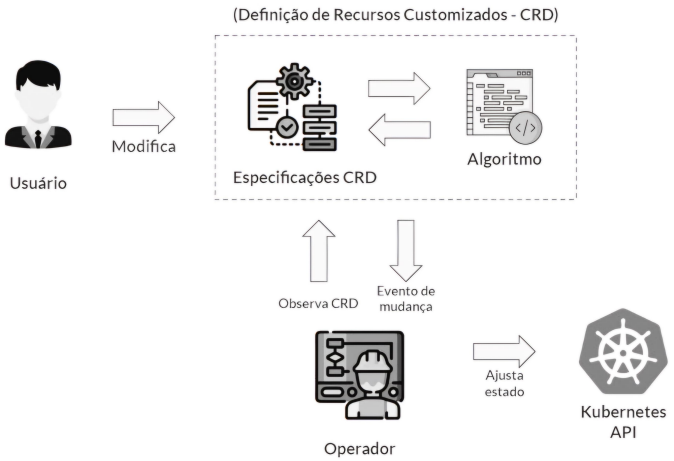

Fig. 1. Arquitetura EMK [8]

A implantação de uma aplicação em um ambiente Kubernetes multi-inquilino inicia-se com a requisição do usuário, que submete um manifesto YAML contendo definições de Pods, *Deployments* e demais recursos. O *API Server* recebe essa solicitação e a persiste no *etcd*, tornando-a acessível aos demais componentes do *cluster*.

O *Kube Controller Manager* detecta a criação do *Deployment* e inicia a geração dos respectivos Pods. Em seguida, o *Kube Scheduler* avalia a alocação desses Pods com base em critérios como disponibilidade de CPU, memória e políticas de afinidade. No entanto, em um ambiente multi-inquilino com alocação dinâmica, essa decisão é aprimorada pelo EMK, conforme os critérios definidos na Seção II-A.

O EMK, ilustrado na Figura 1, é responsável por definir parâmetros que caracterizam o inquilino e sua carga de trabalho, aplicando uma estratégia de alocação baseada em critérios de prioridade e equidade. O *Custom Controller*, que integra o EMK, intercepta a criação de novos Pods e executa uma normalização dos critérios, atribuindo pesos às métricas de alocação.

Para determinar o nó mais adequado, o controlador aplica o método TOPSIS, levando em consideração as condições atuais do cluster e as demandas do inquilino. O resultado dessa análise influencia diretamente o processo de escalonamento, promovendo uma distribuição equilibrada dos recursos.

Algoritmo 1: Algoritmo com integração TOPSIS

```
   Input: CRD com matriz de decisão e pesos
   Output: Estado ajustado do cluster Kubernetes
1  Função CRD_TOPSIS():
2      crd ← DefinirCRD("TOPSIS_CRD");
3      while MonitorarCRD(crd) do
4          matriz_decisao ← ObterMatrizDecisao(crd);
5          matriz_normalizada ←
             NormalizarMatriz(matriz_decisao);
6          pesos ← ObterPesos(crd);
7          matriz_ponderada ←
             AplicarPesos(matriz_normalizada, pesos);
8          solucao_ideal, solucao_antiideal ←
             CalcularSolucoesIdeais(matriz_ponderada);
9          distancias ←
             CalcularDistancias(matriz_ponderada,
             solucao_ideal, solucao_antiideal);
10         scores ← CalcularScoresTOPSIS(distancias);
11         AtualizarCRD(crd, scores);
12         AjustarEstadoKubernetes(scores);
13     end
14     return "Integração CRD-TOPSIS concluída";
```

O Algoritmo 1 exemplifica a integração do método TOPSIS ao Kubernetes por meio do EMK, permitindo decisões baseadas em múltiplos critérios para ajuste do estado do *cluster*. Inicialmente, o EMK é configurado para armazenar os pesos dos critérios (Seção II-A) e a matriz de decisão (Seção II-B). O sistema monitora continuamente o CRD, extraindo e normalizando a matriz de decisão para garantir a comparabilidade entre os critérios.

Com os pesos aplicados, gera-se a matriz ponderada. A partir dela, são determinadas as soluções ideal e anti-ideal, que representam os melhores e piores cenários possíveis. Em seguida, as distâncias de cada alternativa em relação às soluções são calculadas e transformadas em *scores* TOPSIS. Esses valores são atualizados no EMK, possibilitando que o Kubernetes ajuste dinamicamente a alocação de recursos de forma eficiente, justa e otimizada entre os serviços do *cluster*.

A. Critérios de seleção para alocação dos multi-inquilinos

Os critérios de seleção visam auxiliar na alocação dos inquilinos, identificando o nó que oferece equilíbrio entre o estado da infraestrutura de nuvem e as necessidades do usuário [9]. Esses critérios são organizados em uma matriz de decisão, normalizados para balancear suas influências e usados para determinar a proximidade das alternativas com as soluções ideais, possibilitando decisões objetivas na alocação de recursos no Kubernetes.

Em ambientes multi-inquilino no Kubernetes [3], os critérios adotados para a proposta seguem as diretrizes do Kubernetes: (i) disponibilidade de CPU, (ii) memória, (iii) disco, (iv) uso de *ResourceQuota*, (v) controle de acesso com *Role* e *RoleBinding*, (vi) políticas de rede (*NetworkPolicy*), (vii) isolamento de nós, (viii) isolamento de armazenamento (PVs e PVCs), (ix) QoS, e (x) prioridade e justiça na alocação.

A disponibilidade de recursos (CPU, memória e disco) é essencial para evitar sobrecarga e garantir balanceamento entre as *workloads* dos inquilinos. O *ResourceQuota* define limites por namespace, evitando a monopolização de recursos, enquanto *Role* e *RoleBinding* controlam permissões, assegurando o acesso adequado aos recursos.

O *NetworkPolicy* assegura o isolamento de tráfego entre pods e namespaces, reforçando a segurança. O isolamento de nós, com *node affinity* e *tolerations*, previne falhas em um inquilino afetando os outros. O isolamento de armazenamento, por meio de *PVs* e *PVCs*, assegura a segregação dos dados.

A QoS é crucial para garantir que requisições prioritárias recebam recursos, conforme os pesos definidos na Subseção II-C. A atribuição de prioridade e justiça assegura um ambiente equilibrado, evitando desigualdade na alocação de recursos. Esses critérios, alinhados às melhores práticas do Kubernetes, garantem um ambiente multi-inquilino seguro e eficiente.

B. Normalização dos critérios de QoS e Prioridade e Justiça

Para a aplicação dos critérios e os seus respectivos valores ao algoritmo, os critérios de QoS e Prioridade e Justiça são normalizados. Visto que, a operação precisa estar na mesma unidade de medida e a múltiplos fatores que podem ser utilizados para definir a prioridade de um inquilino e também os fatores para definir a QoS para cada inquilino. Disso isso, o usuário iniciando o processo de alocação de recurso informando os requisitos necessários do inquilino para o sistema. A API informa em paralelo os recursos disponíveis do sistema.

A normalização evita que diferentes escalas influenciem os resultados. Para isso, a matriz de decisão X, com m alternativas e n critérios, é transformada na matriz R, onde:

$$R_{ij} = \frac{X_{ij}}{\sqrt{\sum_{i=1}^{m} X_{ij}^2}} \tag{1}$$

Aplicam-se então os pesos w_j, obtendo a matriz ponderada T:

$$T_{ij} = R_{ij} \cdot w_j \tag{2}$$

Com T, determinam-se os vetores ideal positivo e negativo:

$$A_j^+ = \max(T_{ij}), \quad A_j^- = \min(T_{ij}) \tag{3}$$

Calculam-se as distâncias euclidianas às soluções ideais:

$$S_i^+ = \sqrt{\sum_{j=1}^{n} (T_{ij} - A_j^+)^2}, \quad S_i^- = \sqrt{\sum_{j=1}^{n} (T_{ij} - A_j^-)^2} \tag{4}$$

Por fim, o índice de similaridade é dado por:

$$C_i = \frac{S_i^-}{S_i^+ + S_i^-} \tag{5}$$

As alternativas são então classificadas com base nesses índices, priorizando aquelas com valores mais próximos de 1. O método TOPSIS, amplamente utilizado em decisões multicritério, garante uma alocação eficiente ao balancear múltiplos fatores relevantes para a escolha dos recursos no ambiente Kubernetes.

C. Definição dos pesos para os critérios

A atribuição de pesos aos critérios é essencial para refletir sua influência relativa na tomada de decisão. Seja um conjunto de $n = 10$ critérios $C = \{C_1, C_2, ..., C_{10}\}$, os pesos $W = \{w_1, w_2, ..., w_{10}\}$ são definidos de forma a garantir que:

$$\sum_{j=1}^{10} w_j = 1 \tag{6}$$

Os critérios relacionados ao desempenho do sistema, como CPU e memória, além dos critérios de QoS, prioridade e justiça, possuem maior peso devido à sua relevância para a estabilidade e equidade do cluster. Assim, definimos pesos diferenciados, onde critérios críticos possuem maior influência na decisão.

Os critérios e seus respectivos pesos são definidos da seguinte maneira: CPU (C_1) e Memória (C_2) possuem pesos $w_1 = w_2 = 0.15$; QoS (C_3), Prioridade (C_4) e Justiça (C_5) têm $w_3 = w_4 = w_5 = 0.12$; Armazenamento (C_6) recebe $w_6 = 0.10$; *ResourceQuota* (C_7), *RoleBinding* (C_8) e *Network Policy* (C_9) são alocados com $w_7 = w_8 = w_9 = 0.08$; e, finalmente, outros critérios (C_{10}) têm $w_{10} = 0.07$.

Assim, a matriz de pesos W pode ser expressa como um vetor:

$$W = \begin{bmatrix} 0.15 & 0.15 & 0.12 & 0.12 & 0.12 & 0.10 & 0.08 & 0.08 & 0.08 & 0.07 \end{bmatrix} \tag{7}$$

III. METODOLOGIA EXPERIMENTAL

Esta seção descreve a metodologia utilizada para validar experimentalmente a proposta de alocação dinâmica via EMK, incluindo o ambiente computacional, os cenários de avaliação e os procedimentos de coleta e análise dos dados. Para avaliar os métodos propostos de alocação dinâmica de recursos no Kubernetes, os experimentos foram realizados em ambiente experimental utilizando um cluster composto por dois nós físicos com especificações distintas, conforme descrito na Tabela I.

TABELA I
ESPECIFICAÇÕES DO CLUSTER EXPERIMENTAL

Componente	Servidor 1	Servidor 2
Processador	Intel Core i7-9750H	Intel Core i7-1355U
Frequência	2.60 GHz	1.20-5.00 GHz
Memória RAM	32 GB	32 GB
Armazenamento	256 GB	512 GB
Sistema Operacional	Ubuntu 22.04 LTS	Ubuntu 22.04 LTS

Em ambientes multi-inquilinos, como o Kubernetes, a gestão adaptativa de recursos é crucial. A **adaptabilidade** refere-se à capacidade de ajuste às variações na demanda de cada inquilino; já a **flexibilidade** está relacionada à possibilidade de adaptação a diferentes perfis de carga sem modificar a arquitetura subjacente.

A validação foi conduzida em um cluster Kubernetes de alta disponibilidade distribuído em seis nós virtualizados, implementando as melhores práticas de separação de funções conforme descrito em [13]. O cluster foi configurado com dois nós *control plane* dedicados para garantir tolerância a falhas e quatro nós workers especializados para execução de cargas de trabalho.

Nó Control Plane (2 unidades): - CPU: 3 vCPUs cada (total: 6 vCPUs) - Memória: 4 GB RAM cada (total: 8 GB) - Armazenamento: 50 GB SSD cada - Função: *etcd* + API *Server + Controller Manager + Scheduler*

Nós Worker (4 unidades): - CPU: 4 vCPUs cada (total: 16 vCPUs) - Memória: 11 GB RAM cada (total: 44 GB) - Armazenamento: 100 GB SSD cada - Função: Execução de pods de aplicação e EMK

Esta arquitetura distribui os 22 vCPUs e 52 GB de RAM totais disponíveis no hardware físico, garantindo isolamento de funções, tolerância a falhas de nós e capacidade adequada para experimentos multi-inquilinos. O sistema operacional Ubuntu 22.04 LTS foi utilizado em todos os nós, com Kubernetes v1.28 para compatibilidade com CRDs avançados.

Aplicações genéricas, compostas por microserviços com consumo variável de CPU e memória, foram executadas com limites e solicitações configurados por contêiner. A carga foi gerada pela ferramenta Locust, simulando tráfego HTTP e MQTT com controle de requisições simultâneas e duração.

Cada experimento foi repetido 10 vezes sob as mesmas condições para cada cenário avaliado, a fim de capturar a variação natural dos resultados. Para cada métrica analisada, foram registradas as médias e os desvios padrão obtidos ao longo dessas repetições, garantindo uma avaliação estatística mais robusta da solução proposta.

IV. Resultados e Discussão

Nesta seção, apresentamos os resultados do estudo, destacando as melhorias proporcionadas pelo método de automação desenvolvido em comparação com abordagens tradicionais de alocação de recursos. A análise abrange diferentes aspectos, como a redução de intervenções manuais, a eficiência na alocação de recursos, a redução do tempo de resposta e a flexibilidade adaptativa do sistema. As métricas consolidadas referem-se à média e ao desvio padrão obtidos a partir de 10 execuções independentes para cada cenário

A. Automatização do Processo Decisório

A alocação tradicional de recursos no Kubernetes depende de decisões manuais dos administradores, o que pode causar inconsistências e atrasos. O EMK automatiza esse processo, monitorando e respondendo em tempo real às mudanças no cluster, eliminando a latência entre a necessidade identificada e a implementação da decisão.

A Figura 2 compara o número de intervenções manuais entre o método tradicional e o EMK. O método tradicional inicia com 1000 intervenções, diminuindo gradualmente para 5, enquanto o EMK começa com 2 e reduz para 1 após a terceira requisição. Essa redução se deve à eliminação de intervenções manuais, tornando o processo mais eficiente.

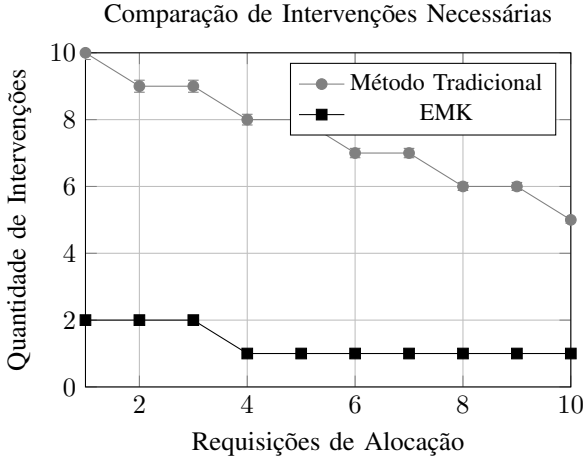

Fig. 2. Comparação entre o método tradicional e o EMK.

O EMK requer apenas duas intervenções críticas: a configuração inicial dos critérios e a validação final das decisões sugeridas pelo algoritmo.

B. Impacto na Gestão de Recursos

A alocação tradicional de recursos frequentemente resulta em alocações subótimas devido à complexidade de considerar múltiplos critérios. O EMK, ao integrar um algoritmo com CRD, realiza decisões baseadas em múltiplos critérios, otimizando a utilização da infraestrutura e eliminando vieses na priorização de recursos.

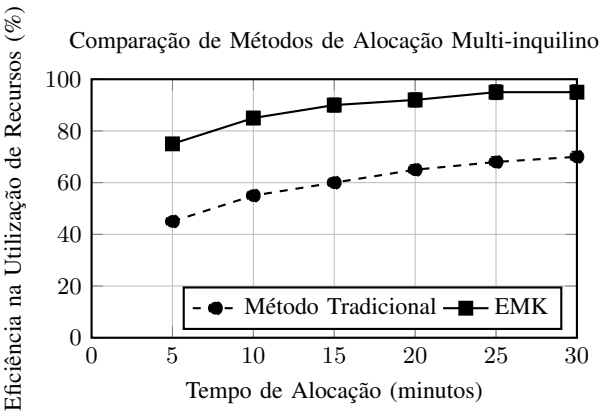

Fig. 3. Eficiência na alocação para inquilinos com diferentes requisitos.

A Figura 3 mostra que o EMK atinge 75% de eficiência nos primeiros 5 minutos e chega a 95% após 25 minutos, mantendo-se estável. O método tradicional, por outro lado, começa com 45% e atinge apenas 70% após 30 minutos. A diferença reflete uma melhoria de 35% na eficiência global do sistema.

A Figura 4 destaca a redução da latência com o EMK, que, após 500 requisições de alocação, apresenta tempos de resposta significativamente menores do que o método tradicional. Essa otimização é resultado da eliminação das verificações manuais e da implementação de um processo

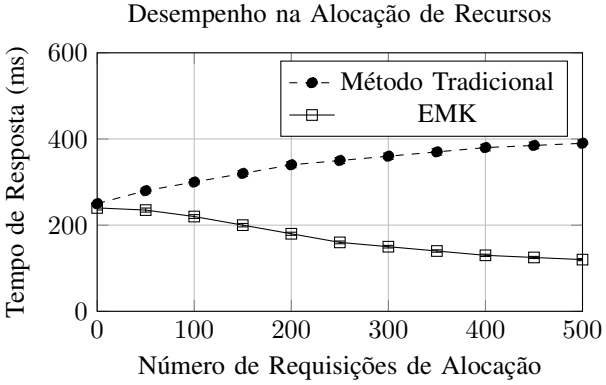

Fig. 4. Comparação do tempo de resposta entre método tradicional e EMK.

decisório automatizado, permitindo decisões mais rápidas e eficientes.

C. Flexibilidade e Adaptabilidade

A Figura 5 ilustra a flexibilidade e adaptabilidade do EMK em diferentes cenários. A flexibilidade refere-se à capacidade de ajustar rapidamente os critérios sem alterar a arquitetura, enquanto a adaptabilidade diz respeito à resposta eficiente a mudanças operacionais. Em um cenário de carga leve, o sistema apresenta flexibilidade no nível 3 e adaptabilidade no nível 2. Com uma carga moderada, ambos os níveis aumentam para 4, e em cargas avançadas, atingem o máximo de 5, refletindo a maturidade do sistema em gerenciar cenários complexos com múltiplos inquilinos.

Essa flexibilidade é essencial para atender às necessidades específicas de diferentes grupos de usuários ou aplicações, permitindo ajustes contínuos e personalizados sem interrupções no serviço.

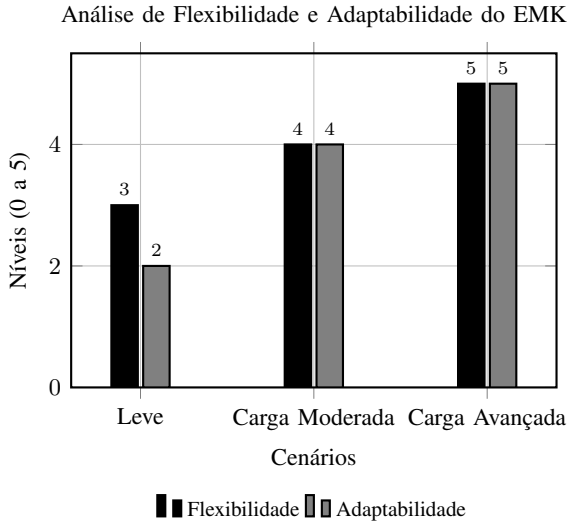

Fig. 5. Análise de flexibilidade e adaptabilidade do EMK em três cenários distintos.

V. Conclusão

Frente às limitações na alocação eficiente de recursos em ambientes Kubernetes multi-inquilinos, este estudo apresentou o EMK, uma abordagem inovadora baseada em algoritmo multicritério para otimizar o desempenho do sistema. A arquitetura possibilitou avaliar o coeficiente de proximidade na alocação dinâmica de inquilinos, promovendo equidade e melhor uso dos recursos do cluster.

Os testes comprovaram a eficácia da solução na melhoria da alocação em ambientes compartilhados. A análise também identificou desafios e oportunidades para aprimorar o Kubernetes conforme as demandas dos fluxos de trabalho multi-inquilinos, impulsionando avanços tecnológicos contínuos.

Assim, o EMK se destaca como contribuição relevante para tornar o Kubernetes mais robusto e adaptável na gestão eficiente de recursos em cenários multi-inquilinos.

As medidas estatísticas mostram que a abordagem oferece robustez mesmo frente à variabilidade de carga. Contudo, em situações com muitos inquilinos e pods, torna-se importante avaliar o impacto no tempo de decisão e resposta do controlador, sinalizando caminhos futuros para otimização da escalabilidade do sistema.

Agradecimentos

O presente trabalho foi realizado com o apoio do Instituto de Informática (INF-UFG) e da empresa Aliare para a participação no Simpósio.

Referências

[1] B. Burns, J. Beda, K. Hightower, e L. Evenson, *Kubernetes: Up and Running*, O'Reilly Media, 2022.

[2] G. Superbo *et al.*, "Hard multi-tenancy Kubernetes approaches in a local 5G deployment: Testing and evaluation of the available solutions," Aalto University, 2022.

[3] Kubernetes, Multi-tenancy, disponível em: `https://kubernetes.io/docs/concepts/security/multi-tenancy/`. Acessado em: 23 mai. 2025.

[4] J. Papathanasiou, N. Ploskas, *Topsis. Multiple Criteria Decision Aid: Methods, Examples and Python Implementations*, pp. 1–30, 2018.

[5] vCluster, "What are virtual clusters," disponível em: `https://www.vcluster.com/docs/what-are-virtual-clusters`. Acessado em: 23 mai. 2025.

[6] Cloud-Ark, "KubePlus," disponível em: `https://github.com/cloud-ark/kubeplus`. Acessado em: 23 mai. 2025.

[7] N. T. Nguyen e Y. Kim, "A design of resource allocation structure for multi-tenant services in Kubernetes cluster," *Proc. 27th Asia Pacific Conf. on Communications (APCC)*, pp. 651–654, 2022.

[8] Kubernetes, Padrão Operador, disponível em: `https://kubernetes.io/pt-br/docs/concepts/extend-kubernetes/operator/`. Acessado em: 23 mai. 2025.

[9] T. Menouer, "KCSS," *The Journal of Supercomputing*, vol. 77, no. 5, pp. 4267–4293, 2021.

[10] A. Araldo, A. D. Stefano, e A. D. Stefano, "Resource allocation for edge computing with multiple tenant configurations," *Proc. 35th ACM Symposium on Applied Computing*, pp. 1190–1199, 2020.

[11] A. Beltre, P. Saha, e M. Govindaraju, "Kubesphere," *IEEE*, vol. 14, 2019.

[12] Kubernetes, Kubernetes components, disponível em: `https://kubernetes.io/docs/concepts/overview/components/`. Acessado em: 23 mai. 2025.

[13] X. Nguyen *et al.*, "Network isolation for Kubernetes hard multi-tenancy," Aalto University, 2020.

[14] K. Senjab, S. Abbas, N. Ahmed, e A. U. R. Khan, "A survey of Kubernetes scheduling algorithms," *Journal of Cloud Computing*, vol. 12, no. 1, art. 87, 2023.

[15] Ł. Wojciechowski *et al.*, "Netmarks: Network metrics-aware Kubernetes scheduler powered by service mesh," *IEEE INFOCOM 2021 - Conf. on Computer Communications*, pp. 1–9, 2021.