

Uma nova Formulação de Programação Inteira para o Problema de Escalonamento de Containers

João Vitor Maués Dias Carneiro

Programa de Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro
jvmaues@cos.ufrj.br

Diego Brandão

Escola de Informática e Computação
Centro Federal de Educação Tecnológica Celso Suckow da Fonseca
diego.brandao@cefet-rj.br

Claudio Miceli Farias

Programa de Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro
cmicelifarias@cos.ufrj.br

Pedro Henrique González

Programa de Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro
pegonzalez@cos.ufrj.br

RESUMO

O trabalho propõe uma nova formulação de Programação Inteira para o escalonamento de containers em clusters Kubernetes, modelada como um Problema de Empacotamento em Múltiplas Mochilas com Penalidade de Realocação ao Longo do Tempo. A formulação busca minimizar o custo total de operação do cluster, considerando custos fixos e variáveis associados à ativação e uso dos nós, além de penalizações por falhas e migrações desnecessárias de *pods* entre períodos. A proposta promove maior estabilidade nas alocações e eficiência no uso dos recursos, superando o kube-scheduler em redução de custos operacionais, embora com maior tempo computacional.

PALAVRAS CHAVE. Escalonamento de Containers, Kubernetes e Otimização Combinatória.

Tópicos (indique, em ordem de PRIORIDADE, o(s) tópico(s) de seu artigo)

ABSTRACT

The work proposes a new Integer Programming formulation for container scheduling in Kubernetes clusters, modeled as a Multiple Knapsack Problem with Reallocation Penalty over Time. The formulation aims to minimize the total operational cost of the cluster, considering fixed and variable costs associated with node activation and usage, as well as penalties for failures and unnecessary *pod* migrations across time periods. The proposed approach promotes greater allocation stability and improved resource utilization, outperforming the kube-scheduler in operational cost reduction, although at the expense of higher computational time.

KEYWORDS. Container Scheduling, Kubernetes and Combinatorial Optimization.

Paper topics (indicate in order of PRIORITY the paper topic(s))

1. Introdução

O escalonamento de containers em clusters Kubernetes (K8s) representa um desafio fundamental para garantir a eficiência operacional de aplicações em ambientes de computação em nuvem. O Kubernetes é atualmente o orquestrador de contêineres mais utilizado na indústria, responsável pelo gerenciamento completo do ciclo de vida de aplicações containerizadas conforme é descrito em Tran et al. [2022]. Para assegurar o desempenho das aplicações e o cumprimento dos Acordos de Nível de Serviço (SLA), mesmo diante de variações dinâmicas na carga de trabalho ou na disponibilidade de recursos, o escalonamento automático é uma funcionalidade essencial Tran et al. [2022].

No entanto, as estratégias padrão de escalonamento automático oferecidas pelo Kubernetes, como o *Horizontal Autoscaler* (HPA) e o *Vertical Pod Autoscaler* (VPA), apresentam limitações significativas, notadamente em sua capacidade de adaptação rápida a cenários de carga variável Tran et al. [2022]. Estudos recentes têm proposto diferentes aprimoramentos para essas estratégias, incluindo métodos reativos e proativos, técnicas baseadas em aprendizado de máquina, e abordagens híbridas que combinam escalonamento horizontal e vertical Tran et al. [2022]. Apesar desses avanços, o kube-scheduler, algoritmo padrão de alocação de *pods*, ainda realiza decisões baseadas em heurística de *best-fit* que não consideram custos operacionais ou impactos de transição entre nós Martins et al. [2023].

Nesse contexto, Martins et al. [2023] propuseram uma formulação matemática de Programação Linear Inteira Mista (PLIM) inspirada no *Capacitated Facility Location Problem* para otimizar a alocação de *pods* em clusters Kubernetes. Embora essa abordagem considere custos fixos e variáveis associados à abertura e uso dos nós, bem como penalizações relacionadas ao histórico de falhas dos nós, ela negligencia explicitamente os custos de transição de *pods* entre nós ao longo do tempo, um aspecto crítico em ambientes dinâmicos e heterogêneos. Essa limitação reduz a aplicabilidade da solução em cenários onde o reescalonamento frequente é necessário, potencialmente gerando custos ocultos devido à migração desnecessária de cargas entre os nós.

Além disso, a literatura evidencia que estratégias de escalonamento que desconsideram os impactos decorrentes de transições entre nós tendem a adotar decisões miopes, suscetíveis a degradação de desempenho, aumento de latência ou elevação de custos operacionais Li et al. [2014]; de Farias et al. [2013]. Em contrapartida, técnicas de otimização combinatória permitem modelar explicitamente tais efeitos, favorecendo decisões de alocação mais robustas e sensíveis ao contexto.

No contexto de Kubernetes, diversos trabalhos têm explorado tanto a melhoria do *kube-scheduler* quanto o desenvolvimento de escalonadores customizados para atender objetivos específicos. Carrión Carrion [2022] apresenta uma taxonomia abrangente de objetivos, métricas e desafios associados ao escalonamento, enquanto Rejiba e Chamanara [2022] categorizam problemas recorrentes e abordagens de solução na personalização do escalonador. Senjab et al. [2023] realizam uma análise experimental comparativa de algoritmos, ressaltando métricas de desempenho e limitações dos métodos existentes. Outras revisões Ahmad et al. [2022]; Tran et al. [2022] abrangem desde abordagens matemáticas e heurísticas até técnicas de aprendizado de máquina e *deep reinforcement learning* aplicadas ao agendamento e ao *autoscaling*, destacando lacunas como a ausência de modelagem de custos de migração e de estabilidade temporal.

Especificamente, alguns estudos incorporam explicitamente penalidades que associam realocações a modelos de programação inteira ou linear, seja em ambientes de contêineres como em Zhang et al. [2017] ou de máquinas virtuais como em Shahryari et al. [2022]; Koubàa et al. [2024], demonstrando que tais mecanismos reduzem o *churn* e promovem maior estabilidade. Ou-

tras propostas exploram métricas adicionais, como latência onde é mostr Carrion et al. [2023], e a implementação direta no *scheduling framework* oficial Furnadzhiev et al. [2025], o que possibilita comparações justas com o escalonador padrão do Kubernetes.

Este trabalho propõe uma nova formulação matemática que estende o modelo de Martins et al. Martins et al. [2023] ao incorporar penalizações explícitas para mudanças de alocação entre nós ao longo do tempo. Essa extensão visa superar as limitações do kube-scheduler e das soluções existentes ao permitir decisões adaptativas que equilibram eficiência operacional, estabilidade de alocação e custos de transição. Ao integrar tais considerações, a proposta busca contribuir para o estado da arte em escalonamento de containers, oferecendo uma abordagem mais adequada para ambientes dinâmicos e heterogêneos típicos de infraestruturas de nuvem modernas.

2. Descrição do Problema

O problema de escalonamento de contêineres em clusters Kubernetes consiste em decidir, a cada instante de tempo, como distribuir os *pods*, unidades de execução que encapsulam contêineres, entre os nós disponíveis no cluster. O objetivo é minimizar o custo operacional, garantindo, ao mesmo tempo, o atendimento às necessidades dos *pods* e o respeito às capacidades dos nós Carrion [2022]; Rejiba e Chamanara [2022]. Cada nó do cluster possui recursos limitados, como CPU e memória, e sua ativação implica custos fixos e variáveis, associados à inicialização, ao uso por *pods* e à gestão de falhas acumuladas Ahmad et al. [2022]. Em cada intervalo de tempo, a dinâmica de execução modifica o estado do cluster: alguns *pods* são finalizados, novos *pods* surgem, e o sistema precisa reavaliar quais nós devem permanecer ativos ou ser desativados, de modo a otimizar a utilização dos recursos Tran et al. [2022]; Senjab et al. [2023].

O desafio central está em determinar quais nós ativar e em quais deles alocar os *pods*, assegurando que os requisitos de recursos sejam atendidos, que o número de nós ativos seja o mínimo necessário e que o custo total da solução seja reduzido. Esse custo inclui o valor fixo de manter um nó ativo, o custo de alocar *pods* a esse nó e uma penalidade associada ao histórico de falhas dos *pods* alocados Shahryari et al. [2022]; Koubàa et al. [2024]. O problema guarda semelhança com o *Capacitated Facility Location Problem* (CFLP), no qual se busca a localização ótima de instalações para atender demandas ao menor custo, respeitando as capacidades de cada instalação Li et al. [2014]; de Farias et al. [2013]. De forma análoga, no Kubernetes, procura-se o nó mais adequado para cada pod, considerando tanto os custos quanto as restrições de capacidade Zhang et al. [2017]; Furnadzhiev et al. [2025].

Algorithm 1 Algoritmo Simplificado do Kube-Scheduler

```

1: procedure SCHEDULEPOD(Pod)
2:   nodes  $\leftarrow$  Listar todos os nós disponíveis
3:   feasibleNodes  $\leftarrow$  Filtrar nós que atendem aos requisitos do Pod
4:   if feasibleNodes está vazio then
5:     Rejeitar o agendamento do Pod
6:   else
7:     scoredNodes  $\leftarrow$  Aplicar funções de pontuação em feasibleNodes
8:     bestNode  $\leftarrow$  Selecionar o nó com a maior pontuação
9:     Atribuir o Pod ao bestNode
10:  end if
11: end procedure

```

Trata-se de um processo dinâmico e iterativo, pois, a cada novo intervalo de tempo, o estado do cluster se altera e novas decisões de escalonamento precisam ser tomadas. Um exemplo

típico ocorre quando nós que estavam em uso em instantes anteriores tornam-se desnecessários após a finalização de *pods*. Nesses casos, mantê-los ativos apenas aumenta o custo operacional. Em síntese, o problema pode ser descrito como a busca pela configuração ótima do cluster ao longo do tempo, respondendo à seguinte questão: qual alocação de *pods* permite ao cluster atender à demanda com o menor número de nós ativos e o menor custo possível? A resposta a essa questão impacta diretamente a eficiência das infraestruturas em nuvem, influenciando o consumo de energia, a utilização de recursos, a escalabilidade e o desempenho dos serviços executados no cluster Carrion et al. [2023]; Mondal et al. [2023].

3. Trabalhos relacionados

O escalonamento automático de aplicações containerizadas em Kubernetes tem recebido atenção significativa da comunidade científica nos últimos anos, resultando em uma diversidade de propostas que visam aprimorar o desempenho e a eficiência dos mecanismos padrão de escalonamento. Em uma revisão abrangente, Tran et al. [2022] categorizaram essas abordagens em função da arquitetura das aplicações, do método de escalonamento (horizontal, vertical ou híbrido), do momento da decisão (reativo ou proativo) e das métricas utilizadas como gatilho para o escalonamento. Os autores destacam que, embora as soluções baseadas em aprendizado de máquina e técnicas proativas apresentem maior potencial para lidar com cargas dinâmicas, muitas propostas desconsideram os custos operacionais associados à realocação de *pods* e à reconfiguração do cluster.

Martins et al. [2023] avançaram na direção de modelar o problema de alocação de *pods* como um *Capacitated Facility Location Problem*, resolvido por Programação Linear Inteira Mista. Essa modelagem permite considerar custos fixos de abertura de nós, custos variáveis de alocação de *pods* e penalizações relacionadas ao histórico de falhas dos nós. Embora os resultados experimentais tenham mostrado ganhos significativos em relação ao kube-scheduler padrão, a formulação ignora explicitamente os custos associados à mudança de alocação de *pods* entre slots de tempo consecutivos, o que pode levar a decisões instáveis ou excessivamente sensíveis a pequenas variações no ambiente.

Em paralelo, Li et al. [2014] e Farias et al. [2013] investigaram problemas semelhantes em redes de sensores sem fio compartilhadas. Nesses contextos, os autores destacam a importância de considerar o custo das transições entre estados de alocação para maximizar a eficiência energética e a estabilidade do sistema. Técnicas de *task scheduling* que exploram o compartilhamento de recursos entre múltiplas aplicações demonstraram ser eficazes para reduzir o consumo energético e aumentar o tempo de vida útil da rede, ao evitar realocações desnecessárias.

Além disso, o trabalho de Li et al. [2014] propõe o algoritmo SACHSEN, que equilibra múltiplos critérios de qualidade de serviço e eficiência energética, utilizando heurísticas baseadas em grafos para alocar tarefas em redes heterogêneas. Embora desenvolvido para redes de sensores, o princípio de considerar os custos de transição e a estabilidade das alocações é diretamente aplicável ao contexto de clusters Kubernetes, onde mudanças frequentes de alocação podem resultar em penalizações de desempenho e custo.

Por fim, a literatura aponta que, embora existam avanços importantes na modelagem matemática da alocação de recursos em ambientes distribuídos, poucos trabalhos integram explicitamente as penalizações de realocação em modelos de Programação Inteira Mista aplicados ao Kubernetes. Essa lacuna motiva a proposta deste trabalho, que visa preencher esse espaço ao estender a modelagem existente para capturar dinamicamente os custos de transição, permitindo decisões mais sensíveis ao histórico de alocação.

4. Formulação Proposta

A formulação proposta busca minimizar o custo total de operação de um cluster Kubernetes ao longo de uma sequência de slots de tempo, considerando custos de ativação de nós, alocação

de *Pods*, penalizações relacionadas a históricos de falhas e, principalmente, os custos de transição de *Pods* entre nós. Para exemplificar a nossa metodologia, partimos da formulação do Capacitated Facility Location Problem (CFLP), como já abordado na literatura e no trabalho apresentado em Martins et al. [2023]. No entanto, nosso modelo introduz um aspecto inédito: a penalização explícita pela descontinuidade da alocação anterior de *Pods*, ou seja, um custo adicional associado à migração de *Pods* entre nós em diferentes instantes. Este elemento é central para ambientes dinâmicos como o Kubernetes, nos quais as decisões de realocação podem impactar diretamente na latência, na estabilidade e no consumo de recursos da infraestrutura.

Essa penalização é representada pela última parcela da função objetivo, envolvendo ω e o histórico de alocação $\bar{y}_{ij}$. Ao penalizar mudanças desnecessárias, o modelo busca um equilíbrio entre o custo de abrir novos nós, alocar *Pods* e manter a estabilidade das alocações já estabelecidas.

A seguir, detalhamos o modelo matemático proposto:

Modelo de Alocação com Penalidade:

$$\min \sum_{i \in I} \left(\alpha_i x_i + (\beta_i + \gamma_i) \sum_{j \in J} y_{ij} \right) + \sum_{i \in I} \sum_{j \in J_0: \bar{y}_{ij}=1} (1 - y_{ij}) \omega \quad (1)$$

$$\sum_{j \in J} u_j^k y_{ij} \leq U_i^k x_i, \quad \forall i \in I, \forall k \quad (2)$$

$$\sum_{i \in I} y_{ij} = 1, \quad \forall j \in J \quad (3)$$

$$y_{ij} \in \{0, 1\}, \quad \forall i \in I, \forall j \in J \quad (4)$$

$$x_i \in \{0, 1\}, \quad \forall i \in I \quad (5)$$

O modelo proposto busca minimizar o custo total da alocação de *Pods* em nós do cluster, levando em consideração quatro componentes principais na função objetivo. O primeiro é o custo fixo de ativar cada nó, representado por $\alpha_i x_i$, seguido do custo de alocar *Pods* nesses nós, dado por $\beta_i y_{ij}$. Além disso, o modelo considera a penalização associada ao histórico de falhas dos nós, expressa por $\gamma_i y_{ij}$. Por fim, e como principal diferencial em relação a modelos anteriores, é incorporada uma penalização adicional pelo desligamento da alocação anterior, ou seja, quando um *Pod* deixa de ser alocado no mesmo nó onde já estava previamente posicionado. Esse último termo, $(1 - y_{ij})\omega$, busca desencorajar migrações desnecessárias, equilibrando o custo-benefício de realocar um *Pod* em relação à estabilidade da solução.

As restrições do modelo garantem a viabilidade técnica da alocação. A restrição de capacidade limita o consumo de recursos de cada tipo k em cada nó i , assegurando que os *Pods* alocados não ultrapassem a capacidade disponível, desde que o nó esteja ativado. Já a restrição de alocação única impõe que cada *Pod* seja alocado em exatamente um nó, evitando alocações múltiplas ou conflitantes. As últimas duas restrições definem o domínio das variáveis, garantindo que as decisões de abrir um nó (x_i) e alocar um *Pod* (y_{ij}) sejam binárias, representando a natureza discreta das escolhas do sistema. Dessa forma, o modelo formaliza o problema de alocação considerando tanto a eficiência operacional quanto a continuidade das decisões ao longo do tempo.

Parâmetros e Variáveis

- I : Conjunto de nós do cluster, indexado por i .

- J : Conjunto de *pods*, indexado por j .
- $J_0 \subseteq J$: Conjunto de *pods* previamente alocados em uma solução anterior.
- $\bar{y}_{ij} \in \{0, 1\}$: 1 se o *pod* j estava alocado no nó i na solução anterior, 0 caso contrário.
- u_j^k : Demanda do *pod* j em relação ao recurso k .
- U_i^k : Capacidade do nó i para o recurso k .
- α_i : Custo de ativação do nó i .
- β_i : Custo de alocação de um *pod* no nó i .
- γ_i : Penalização associada a falhas do nó i .
- ω : Penalização por descontinuar a alocação anterior.
- $x_i \in \{0, 1\}$: Variável binária que indica se o nó i está ativado.
- $y_{ij} \in \{0, 1\}$: Variável binária que indica se o *pod* j é alocado no nó i .

5. Experimentos Computacionais

Nesta seção, apresentamos a avaliação experimental realizada para comparar o comportamento do algoritmo de escalonamento padrão do Kubernetes, conhecido como Kube-scheduler, com a abordagem de otimização proposta neste trabalho, baseada em Programação Inteira.

5.1. Ambiente Computacional

Os experimentos foram conduzidos em um ambiente computacional composto por um computador pessoal equipado com processador Apple M2 Max, 32 GB de memória RAM e sistema operacional macOS Sonoma 14.4. A implementação da metodologia proposta, bem como da abordagem baseada no escalonador padrão do Kubernetes, foi realizada em Python 3.11. Para a resolução da formulação matemática proposta, utilizou-se o resolvidor Gurobi versão 11.0.0, acessado por meio de sua API para Python. Todos os scripts desenvolvidos, assim como os dados gerados durante os experimentos, estão disponíveis publicamente em um repositório no GitHub para fins de reprodutibilidade.

5.2. Implementação e Descrição dos Experimentos

Foi desenvolvido um conjunto fixo de 40 instâncias artificiais para possibilitar a replicação e a validação sistemática dos experimentos. Cada instância corresponde a uma sequência temporal de 20 unidades de tempo, nas quais novas demandas de alocação de *pods* são progressivamente introduzidas no sistema. Essa construção reflete a evolução do estado do cluster, exigindo a cada instante uma nova decisão de alocação baseada no histórico anterior, condição necessária para aplicar a penalização por descontinuidade na alocação de *pods* entre nós.

As instâncias foram geradas variando o número de nós ($N \in 10, 20, 50, 100, 200$) e de *pods* ($P \in 50, 100, 200, 500, 1000, 5000, 10000$), de forma a contemplar diferentes níveis de escala e complexidade. Para cada combinação, testaram-se dois métodos de escalonamento: (i) um que reproduz o comportamento do escalonador padrão do Kubernetes, que distribui os *pods* aleatoriamente entre os nós disponíveis; e (ii) a formulação proposta em Programação Inteira Mista, modelada em Python e resolvida com o solver Gurobi.

Os parâmetros das instâncias foram definidos aleatoriamente dentro de faixas preestabelecidas para garantir diversidade de cenários. As capacidades dos nós (U_i) foram sorteadas entre

$m/(n + 1)$ e $2n$, sendo m o número de *pods* e n o número de nós. As demandas dos *pods* (u_j) variaram entre 1 e 10 unidades. Os custos de abertura (α_i) e de alocação (β_i) dos nós foram sorteados entre 1 e $4n$, enquanto o custo de penalização por histórico de erros (γ_i) foi fixado em zero, simulando um ambiente ideal sem falhas.

Cada configuração foi executada 10 vezes de forma independente, sempre sobre as mesmas instâncias, para assegurar consistência estatística na análise. No modelo de Programação Inteira, o Gurobi foi configurado com tolerância de subótimo (MIPGap) de 1%, equilibrando tempo de execução e qualidade da solução. Registraram-se o tempo médio e os custos obtidos por cada método, visando posterior comparação de desempenho.

5.3. Resultados

O principal objetivo dos experimentos foi comparar o custo total das soluções e o tempo de execução entre a abordagem proposta e o escalonador nativo do Kubernetes. Os resultados mostraram que, em praticamente todos os cenários avaliados, a metodologia baseada em otimização obteve soluções significativamente melhores em termos de custo, frequentemente uma ordem de grandeza inferior ao custo das soluções encontradas pelo escalonador aleatório. Esse ganho reflete uma alocação mais eficiente dos *pods*, potencializando a utilização da infraestrutura física do cluster.

Contudo, observou-se que o custo da solução nem sempre diminui proporcionalmente ao aumento do número de nós para uma mesma quantidade de *pods*. Esse comportamento está relacionado ao sorteio aleatório dos parâmetros das instâncias, que pode gerar cenários em que os novos nós apresentam alta penalização ou baixa capacidade, o que dificulta a exploração dessas novas opções.

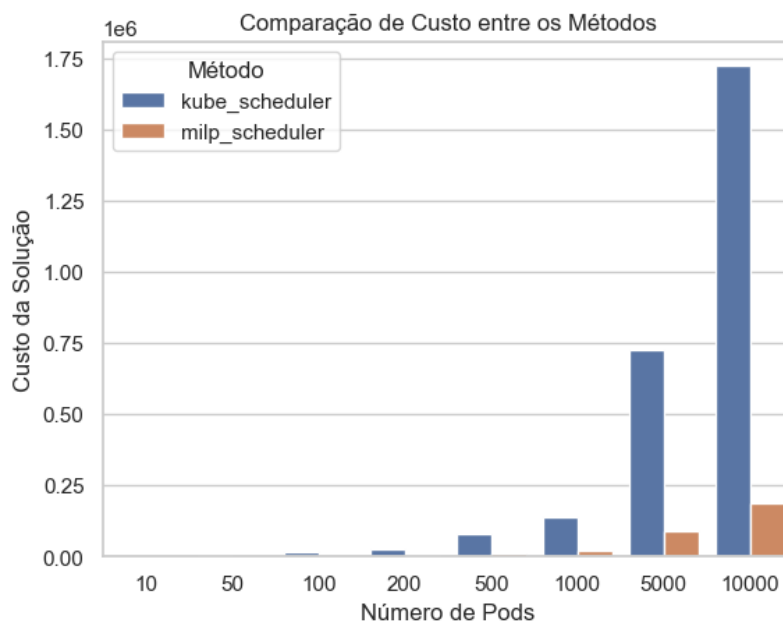


Figura 1: Comparação do Custo entre os Métodos

Na Figura 1, observa-se que o *milp_scheduler* apresenta um desempenho significativamente superior em relação ao custo da solução em todos os cenários avaliados. Esse comportamento é consistente ao longo de diferentes tamanhos de instância, desde 50 até 10.000 *pods*. O ganho é especialmente evidente nas instâncias maiores, onde o *kube_scheduler* apresenta custos crescentes em escala muito superior ao *milp_scheduler*. Isso demonstra que o modelo matemático proposto é

capaz de encontrar alocações mais econômicas, reduzindo o custo total da infraestrutura utilizada, especialmente em cenários de maior escala.

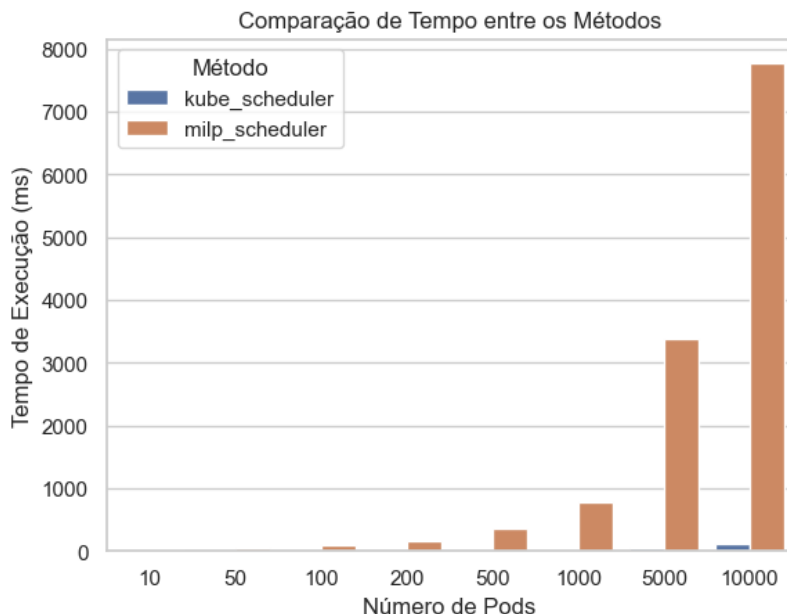


Figura 2: Comparação do Tempo de Execução entre os Métodos

Por outro lado, a Figura 2 revela o custo computacional em termos de tempo de execução para cada método. Enquanto o *kube_scheduler* apresenta tempos praticamente constantes e muito baixos, mesmo para instâncias maiores, o *milp_scheduler* mostra crescimento exponencial do tempo de execução à medida que o número de *pods* aumenta, chegando a ultrapassar 7.500 ms para instâncias com 10.000 *pods*. Esse comportamento evidencia o já esperado trade-off entre qualidade da solução e esforço computacional. Embora o *milp_scheduler* traga soluções com custos substancialmente menores, seu tempo de execução se torna significativamente maior, o que pode ser um fator limitante para aplicações que exigem decisões em tempo quase real.

De forma geral, os resultados confirmam que a solução baseada em otimização é vantajosa em termos de eficiência da alocação, mas sua aplicabilidade prática deve considerar o tempo disponível para o processo decisório, principalmente em cenários de grande escala.

6. Conclusão e Trabalhos Futuros

Neste trabalho, propusemos uma nova formulação matemática de Programação Inteira para o problema de alocação de *pods* em clusters Kubernetes, incorporando explicitamente o custo de descontinuidade da alocação anterior. Essa característica permite ao modelo penalizar migrações desnecessárias de *pods*, promovendo soluções que equilibram o custo de uso da infraestrutura com a estabilidade das alocações ao longo do tempo. A formulação foi avaliada computacionalmente em diferentes cenários e comparada com o comportamento do escalonador nativo do Kubernetes. Os resultados indicaram que a abordagem baseada em otimização é capaz de reduzir significativamente o custo total da solução, potencializando uma utilização mais eficiente da infraestrutura computacional.

Por outro lado, também foi observado que o tempo de execução do método proposto cresce exponencialmente com o aumento do número de *pods*, o que pode limitar sua aplicabilidade em situações que demandam decisões em tempo muito curto. Esse trade-off entre qualidade da

solução e esforço computacional evidencia que a escolha do método deve considerar o contexto operacional, especialmente em ambientes com grande variabilidade de carga.

Como trabalhos futuros, pretende-se explorar estratégias para reduzir o tempo de execução da solução proposta, como o uso de heurísticas baseadas no modelo exato para gerar soluções aproximadas em tempos mais curtos. Além disso, pretende-se estender o modelo para considerar cenários estocásticos. Outra linha promissora é a integração de técnicas de aprendizado de máquina para estimar custos e antecipar decisões, potencialmente combinando a robustez do modelo matemático com a velocidade de métodos preditivos. Por fim, pretende-se validar o modelo em um ambiente Kubernetes real, utilizando aplicações reais e cargas variáveis para avaliar o impacto prático da abordagem proposta em operações de larga escala.

Referências

- Ahmad, I., AlFailakawi, M. G., AlMutawa, A., e Alsalman, L. (2022). Container scheduling techniques: A survey and assessment. *Journal of King Saud University – Computer and Information Sciences*.
- Carrion, C., Tiberti, W., et al. (2023). Latency-aware kubernetes scheduling for microservices orchestration at the edge. In *2023 IEEE Conference on Network Softwarization (NetSoft)*.
- Carrion, M. C. (2022). Kubernetes scheduling: Taxonomy, ongoing issues and challenges. *ACM Computing Surveys*.
- de Farias, C. M., Pirmez, L., Delicato, F. C., Li, W., Zomaya, A. Y., e de Souza, J. N. (2013). A scheduling algorithm for shared sensor and actuator networks. In *2013 International Conference on Information Networking (ICOIN)*.
- Furnadzhiev, R. et al. (2025). Efficient orchestration of distributed workloads in multi-region kubernetes cluster. *Computers*.
- Koubàa, M. et al. (2024). A multi-objective approach for optimizing virtual machine placement. *Digital*.
- Li, W., Delicato, F. C., Pires, P. F., Lee, Y. C., Zomaya, A. Y., Miceli, C., e Pirmez, L. (2014). Efficient allocation of resources in multiple heterogeneous wireless sensor networks. *Journal of Parallel and Distributed Computing*.
- Martins, C., Moura, P. N., Lucena, S. C., e Campos, C. A. V. (2023). Escalonamento de containers em clusters kubernetes usando uma abordagem de programação linear inteira mista. In *Simpósio Brasileiro de Pesquisa Operacional*.
- Mondal, S. K. et al. (2023). Toward optimal load prediction and customizable autoscaling in kubernetes. *Mathematics*.
- Rejiba, Z. e Chamanara, J. (2022). Custom scheduling in kubernetes: A survey on common problems and solution approaches. *ACM Computing Surveys*.
- Senjab, K., Abbas, S., Ahmed, N., e Khan, A. U. R. (2023). A survey of kubernetes scheduling algorithms. *Journal of Cloud Computing*.
- Shahryari, S. et al. (2022). Copam: Cost-aware vm placement and migration for mobile edge computing. *Computer Communications*.

- Tran, M.-N., Vu, D.-D., e Kim, Y. (2022). A survey of autoscaling in kubernetes. In *2022 13th International Conference on Ubiquitous and Future Networks (ICUFN)*.
- Zhang, D., Yan, B., Feng, Z., Zhang, C., e Wang, Y.-X. (2017). Container oriented job scheduling using linear programming model. In *2017 3rd International Conference on Information Management (ICIM)*.