K8sGAScheduler: Algoritmo para alocação inteligente de recursos em cluster kubernetes

Thiago Tavares[1,2], **Carlos Santos**[2], **Kleber Cardoso**[1], **Antonio Oliveira-Jr**[1]

[1]Instituto de Informática (INF) – Universidade Federal do Goiás (UFG)

[2]Instituto Federal de Educação, Ciência e Tecnologia do Tocantins (IFTO)

thiagogmta,carlosedu@ifto.edu.br; kleber,antoniojr@ufg.br

Abstract. *The implementation of Kubernetes for application management in cloud and edge environments provides scalability, reliability, and automation. However, cluster management faces challenges such as resource heterogeneity and environment dynamism. Although Kubernetes provides features to address these challenges, optimizing pod allocation is a complex issue that requires advanced solutions. This paper introduces K8sGAScheduler, a Genetic Algorithm-based scheduling technique developed to optimize pod allocation in Kubernetes clusters. This approach considers resource consumption, pod communication, and node capacity constraints, aiming to find an allocation that maximizes cluster efficiency. The results obtained through simulation demonstrate improvements over the Kubernetes default scheduler, offering insights for more efficient resource management in Kubernetes clusters and outlining future research directions in this area*

Resumo. *A implementação do Kubernetes na administração de aplicações em ambientes de nuvem de borda proporciona escalabilidade, confiabilidade e automação. No entanto, o gerenciamento de clusters enfrenta desafios como a heterogeneidade de recursos e a dinamicidade do ambiente. Embora o Kubernetes forneça recursos para lidar com esses desafios, a otimização da alocação de pods é uma questão complexa que requer soluções avançadas. Este artigo introduz o K8sGAScheduler, uma técnica de agendamento baseada em Algoritmo Genético desenvolvida para otimizar a alocação de pods em clusters Kubernetes. Essa abordagem leva em conta o consumo de recursos, a comunicação entre os pods e as restrições de capacidade dos nós, visando encontrar uma alocação que maximize a eficiência do cluster. Os resultados obtidos, por meio de simulação, demonstram melhorias em relação ao scheduler padrão do Kubernetes, oferecendo insights para o gerenciamento mais eficiente de recursos em clusters Kubernetes e delineando para futuras pesquisas nessa área.*

1. Introdução

A introdução do *kubernetes* (k8s) como plataforma de código aberto para orquestração de contêineres revolucionou a forma como aplicações são implantadas e gerenciadas em ambientes de nuvem de borda [Botez et al. 2021]. O k8s oferece diversos benefícios, como escalabilidade, confiabilidade e automação, tornando-se a ferramenta de escolha para muitas empresas [Townend et al. 2019].

Desenvolvido pelo Google e mantido pela *Cloud Native Computing Foundation*, o Kubernetes é um orquestrador de código aberto fundamental para a implantação de aplicativos em contêineres. O k8s simplifica o gerenciamento de aplicativos de microsserviços, oferecendo recursos avançados como balanceamento de carga, escalonamento automático e monitoramento, garantindo a confiabilidade e eficiência na execução dos aplicativos [Burns et al. 2022, Rejiba and Chamanara 2022, Arouk and Nikaein 2020]. No ambiente k8s, cada aplicativo é executado em contêineres encapsulados em pods que são a menor instância da plataforma [Wojciechowski et al. 2021].

O gerenciamento de *clusters* em ambientes de nuvem de borda apresenta desafios específicos, como heterogeneidade de recursos, dinamicidade do ambiente e necessidade de alta disponibilidade [Mfula et al. 2021]. O k8s fornece recursos para lidar com esses desafios, mas otimizar a alocação de cargas de trabalho é um problema complexo que exige soluções avançadas [Chirivella Perez et al. 2018].

A coexistência de cargas de trabalho de lote de serviços sensíveis à latência pode resultar em competição por recursos compartilhados, uma vez que ambos disputam acesso aos mesmos recursos do sistema. Tradicionalmente, essas cargas de trabalho são segregadas e executadas em nós separados no k8s, buscando evitar interferências de desempenho entre elas, o que, porém, frequentemente resulta em subutilização dos recursos disponíveis [Zhang et al. 2021]. O Kubernetes, amplamente adotado para orquestração de contêineres, carece de soluções que considerem pontos de vista dinâmicos de infraestrutura, motivando o desenvolvimento do k8sgascheduler.

O K8sGAScheduler considera métricas de consumo de CPU, memória e relacionamento entre os pods, levando em conta as restrições de capacidade. Ao contrário da abordagem tradicional do kube-scheduler, que trata os pods como entidades independentes e pode resultar na alocação em nós diferentes, o foco é otimizar a colocação dos pods. Ao alocar contêineres em nós com alta utilização de CPU e memória, o sistema pode aproveitar ao máximo os recursos disponíveis sem deixar capacidade ociosa em nós com baixa utilização de CPU [Wojciechowski et al. 2021]. Além disso, ao priorizar a co-locação de pods que se comunicam frequentemente no mesmo nó, reduz-se a latência da comunicação entre os pods, melhorando o desempenho das aplicações no ambiente [Menouer 2021].

Considerando a complexidade de otimização de alocação este estudo busca responder a seguinte questão: Como otimizar o agendamento de recursos em *clusters kubernetes* considerando o consumo de recursos, a comunicação entre os pods e as restrições de capacidades? O objetivo é encontrar uma alocação eficiente de pods que maximize a utilização dos recursos dos nós do cluster, levando em consideração a comunicação entre os pods.

Este problema de alocação consiste em um problema NP-Completo e, problemas desse tipo, são caracterizados por sua dificuldade em encontrar soluções eficientes e demandam abordagens heurísticas ou algoritmos aproximados. Embora verificar uma solução proposta possa ser fácil, encontrar essa solução diretamente pode exigir tempo exponencial. Exemplos incluem o problema do caixeiro viajante e o problema da mochila [Farhi et al. 2001]. Como heurística para encontrar alocações de boa qualidade Optou-se por uma abordagem baseada em Algoritmos Genéticos (AG), uma técnica de otimização

inspirada na teoria da evolução biológica. Esses algoritmos são comumente empregados para resolver problemas complexos de otimização, como a alocação eficiente de recursos.

As principais contribuições deste trabalho são: a) Desenvolvimento de um modelo de alocação de pods em nós que considera de forma simultânea os critérios de CPU, memória e taxa de relacionamento entre os pods; b) Implementação de um algoritmo genético personalizado e uma codificação apropriada para representar o problema de alocação; c) Introdução de uma métrica de avaliação da alocação dos pods, que destaca alocações viáveis e penaliza soluções inviáveis, facilitando o processo de busca do AG. Essas contribuições culminaram em uma metodologia de agendamento denominada K8sGAScheduler, cuja avaliação foi realizada por meio de simulação.

O trabalho está estruturado da seguinte forma: na Seção 2, é apresentada a arquitetura do *kubernetes*. Na Seção 3, é detalhada a modelagem do K8sGAScheduler. A Seção 4 apresenta avaliação experimental e os resultados obtidos, enquanto as conclusões são apresentadas na Seção 5.

2. Arquitetura do *kubernetes*

O k8s é dividido em duas partes principais: o *Master Node*, responsável pela gestão global do *cluster*, incluindo a coordenação de tarefas, o gerenciamento de recursos e a comunicação entre os componentes, e o *Worker Node*, onde as cargas de trabalho são executadas. O *Master Node* supervisiona e coordena as operações do *cluster*, enquanto os *Worker Nodes* executam as tarefas atribuídas.

2.1. Arquitetura e Funcionamento do *kubernetes*: Uma Visão em Dez Passoss

A Figura 1 apresenta um diagrama ilustrativo da arquitetura do k8s e resume o seu funcionamento em dez passos. Este diagrama proporciona uma visão dos componentes do k8s e como eles interagem para gerenciar e executar aplicativos em contêineres. Com base na Figura 1, ao solicitar a criação de um pod para uma aplicação em um contêiner (1), a solicitação é recebida e processada pelo kube-api-server que cria um registro no etcd (2) com informações sobre o estado desejado da aplicação. O kube-api-server responde à solicitação (3), mesmo que o pod ainda não tenha sido criado, pois o registro do estado desejado foi feito. A partir daí, os componentes do k8s trabalham para transformar esse estado desejado em um estado real.

O kube-scheduler periodicamente consulta o kube-api-server em busca de tarefas a serem alocadas (4). Quando uma entrada é encontrada, o *scheduler* seleciona os nós que atendem aos requisitos para receber o pod.

Após de selecionar o nó adequado, o kube-scheduler informa ao kube-api-server, que faz um novo registro no etcd (5). Em seguida, o kube-api-server se comunica com o kubelet do nó selecionado para ativar um pod naquele nó (6). O kubelet trabalha em conjunto com o CIR *(Container Runtime Interface)* para criar um pod com o contêiner desejado (7). Posteriormente, o kubelet informa ao kube-api-server sobre a criação do pod (8), e o kube-api-server faz um novo registro no etcd (9).

Dado que a aplicação deve permanecer em funcionamento, é necessário monitorar a infraestrutura para garantir que o estado real corresponda ao estado desejado registrado no início do processo. Para isso, o controller-manager consulta regularmente o kube-api-server para obter informações sobre o cluster (10). O controller-manager gerencia vários

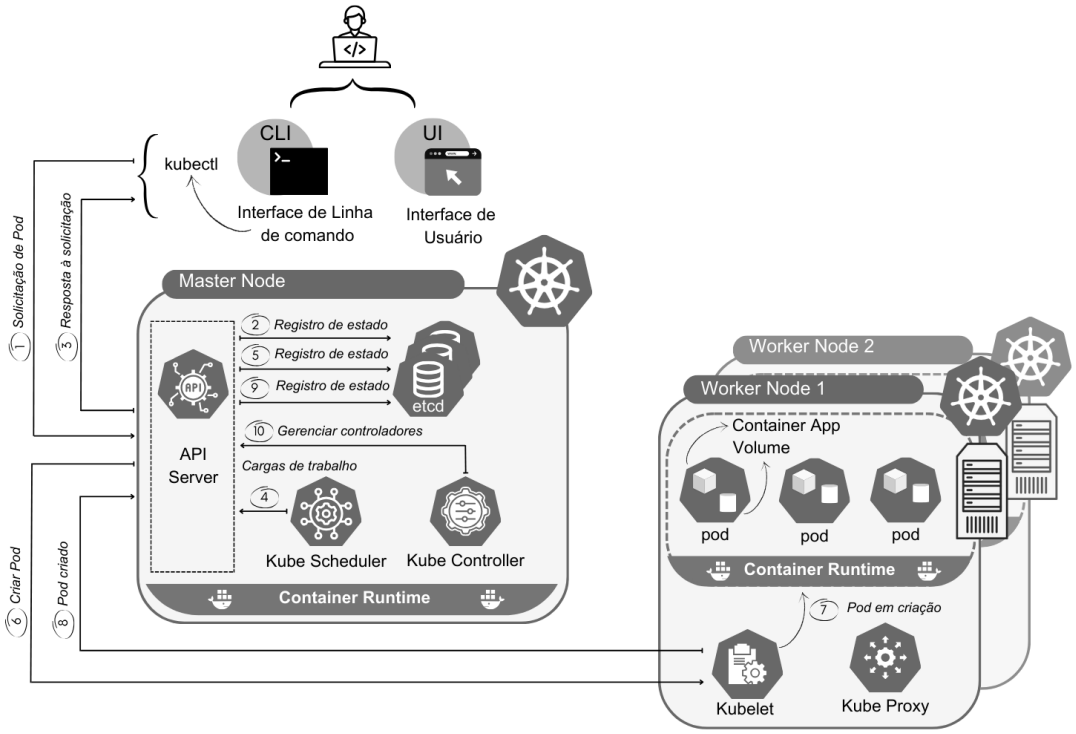

Figura 1. Diagrama da Arquitetura do k8s. Fonte: próprio autor.

controladores, como o replicaSet, que define a quantidade de réplicas de um pod. O controller-manager monitora o cluster para garantir que o estado desejado seja mantido. Se um pod parar, o controller-manager instruirá o kube-api-server para criar um novo pod (10), e o processo será reiniciado.

2.2. *Kubernetes Scheduler*

O kube-scheduler é responsável pela tomada de decisão de alocação de pods nos nós. A alocação é feita através de um processo de filtragem (*filtering*) e pontuação (*scoring*) para determinar o nó mais apropriado para a execução de um pod específico. Inicialmente, o kube-scheduler consulta periodicamente o kube-api-service em busca de pods que ainda não foram alocados em um nó. A filtragem ocorre primeiro, onde são selecionados os nós que atendem aos requisitos mínimos necessários para a execução do pod. Em seguida, o kube-scheduler atribui uma pontuação a cada nó com base em vários critérios. O nó com a maior pontuação é selecionado para a execução do pod. Em casos de empate, um nó é selecionado aleatoriamente.

Embora o agendador padrão seja adequado para muitos casos de uso, existem situações em que um agendador personalizado pode ser mais adequado para atender às necessidades específicas de um aplicativo ou ambiente [Nguyen et al. 2021]. O k8s oferece a flexibilidade de personalizar o processo de agendamento através de *plugins*.

Essa personalização podem ser feita através de: a) um *custom scheduler*, que pode substituir ou trabalhar em conjunto com o *scheduler* padrão; b) um *plugin* para o *scheduler*, que pode ser adicionado ao agendador e requer a recompilação da imagem do contêiner do *scheduler*; ou c) um *scheduler extender*, que pode ser escrito em qualquer

linguagem, configurado em tempo de execução e não requer a recompilação da imagem do contêiner [Wojciechowski et al. 2021]. A terceira abordagem é útil para situações onde as decisões de agendamento precisam ser tomadas com base em recursos que não são diretamente gerenciados pelo agendador padrão [Santos et al. 2019].

A customização do agendador padrão pode aprimorar o desempenho e a eficiência do sistema, mas implica em desafios que exigem amplo conhecimento para implementar soluções eficazes para problemas de agendamento complexos [Rejiba and Chamanara 2022]. A possibilidade de desenvolver um agendador personalizado, integrando métricas não contempladas pelo agendador padrão do *kubernetes*, permite a modelagem e aplicação de novas métricas, como demonstrado pelo K8sGAScheduler na Seção 3.

3. K8sGAScheduler

A modelagem do problema de agendamento dos pods em nós do cluster utiliza como referência o problema das múltiplas mochilas, uma derivação do clássico problema da mochila (*Knapsack problem*). No problema das múltiplas mochilas (*multiple knapsack problem*) o objetivo é selecionar um conjunto de itens que maximize o valor total, respeitando as capacidades individuais de cada mochila [Martello and Toth 1980].

O problema de agendamento do k8s consiste em alocar um conjunto pod, respeitando a capacidade de memória RAM e CPU de cada *Workers node*. Assim, o problema de agendamento do k8s pode ser modelado como problema das múltiplas mochilas, considerando particularidades do problema de agendamento do k8s.

A solução para o problema da mochila requer uma estratégia eficiente de alocação, considerando as características de cada item, como seu valor e tamanho, e encontrando a combinação ideal que otimize o valor total sem exceder as capacidades das mochilas [dell'Amico et al. 2019]. Para encontrar soluções de boa qualidade para o problema das múltiplas mochilas, propomos um Algoritmo Genético.

3.1. Modelagem do Problema de Agendamento proposto

O problema consiste em classificar alocações diferentes para determinar a solução mais eficiente. Para isso, é essencial analisar como cada alocação distribui os pods nos nós e utiliza os recursos disponíveis. A solução ideal é aquela que maximiza a utilização dos recursos em cada nó individualmente e agrupa os pods que mantêm comunicação frequente em um mesmo nó.

Na Equação 1, temos o somatório da porcentagem de recursos solicitados por n pods a serem alocados em m nós. O termo S_j representa a porcentagem de ocupação do nó j, onde S_f^s corresponde a procentagem de ocupação da memória e S_f^p a porcentagem de ocupação CPU. O expoente w é o peso da expressão dos valores aumentando o resultado da equação. A quantidade de nós utilizados para a alocação é representado por B_j. E a taxa de relacionamento entre os pods é representada por $T(x)$.

Como restrições temos: A quantidade de CPU K_i^p que um grupo de n pods i demanda não pode ser superior a quantidade de CPU C_j^p disponível em um nó j (Equação 2); A quantidade de memória K_i^s que um grupo de n pods i demanda não pode ser maior que a quantidade de memória C_j^s disponível em um nó j (Equação 3); As variáveis x_{ij}

são variáveis binárias $\{0, 1\}$, indicando se um pod i está (1) ou não (0) alocado em um nó j (Equação 4).

$$\text{Max} \quad f(x) = \frac{1}{\sum_{j=1}^{m} B_j(x)} \left(\sum_{j=1}^{m} S_j^s(x)^w + \sum_{j=1}^{m} S_j^p(x)^w \right) + T(x) \tag{1}$$

$$s.a.$$

$$\sum_{i=1}^{n} K_i^p(x_{ij}) \leq C_j^p, \quad j = 1, ..., m \tag{2}$$

$$\sum_{i=1}^{n} K_i^s \leq C_j^s, \quad j = 1, ..., m \tag{3}$$

$$x_{ij} \in \{0, 1\}, \quad i = 1, ..., n, \quad j = 1, ..., m, \tag{4}$$

As equações de consumo $S_j^s(x)$ e $S_j^p(x)$, Equação 5 e 6, representam, respectivamente, o percentual de consumo de recursos de memória e CPU em cada nó de uma dada alocação. A equação divide o somatório da quantidade de memória de todos os pods alocados no nó pela capacidade total de memória do nó. De forma análoga funciona a Equação 6 para o cálculo de consumo da CPU.

$$S_j^s(x) = \frac{1}{C_j^s} \sum_{i=1}^{n} K_i^s(x_{ij}) \tag{5}$$

$$S_j^p(x) = \frac{1}{C_j^p} \sum_{i=1}^{n} K_i^p(x_{ij}) \tag{6}$$

A função $B_j(x)$ (Equação 7) avalia se um nó foi ou não utilizado na alocação, retornando valor um (1) se $S_j^s \neq 0$ indicando que houve alocação de recursos naquele nó e retorna zero caso contrário. O total de nós ocupados em uma alocação divide o primeiro termo da Equação 1 com objetivo de penalizar alocações que ocupem mais nós.

$$B_j(x) = \begin{cases} 1 & \text{se } S_j^s \neq 0 \\ 0 & \text{c.c.} \end{cases} \tag{7}$$

A Equação 8 representa o custo de comunicação dos pods alocados. A variável t_{ik} é o custo de comunicação entre os pods i e k pertencentes ao conjunto N, sendo N o conjunto de pods no mesmo nó j. A Equação 8 favorece alocações de pods com alta taxa de comunicação no mesmo nó.

$$T(x) = \sum_{j=1}^{n} \sum_{\substack{i,k \in N \\ k > i}} x_{ij} t_{ik} \tag{8}$$

As equações 9 e 10 calculam o excedente de recursos em um nó requeridos por uma alocação. Assim, alocações infactíveis são penalizadas mas não são totalmente descartadas no processo de busca do AG, pois estas alocações, em alguns casos, tornam se alocações de alta qualidade por meio de pequenas alterações [Xiong et al. 2021].

$$I_j^s(x) = \begin{cases} C_j^s - \sum_{i=1}^{n} K_i^s & \text{se } S_j^s > 1 \\ 0 & \text{c.c.} \end{cases} \tag{9}$$

$$I_j^p(x) = \begin{cases} C_j^p - \sum_{i=1}^{n} K_i^p & \text{se } S_j^p > 1 \\ 0 & \text{c.c.} \end{cases} \tag{10}$$

Para resolução do problema, utilizou-se uma abordagem baseada em algoritmos genéticos que consiste em uma técnica de otimização inspirada na teoria da evolução natural. O AG é eficaz na obtenção de soluções para problemas complexos (neste caso problema NP-Completo), exigindo um esforço computacional moderado [Chu and Beasley 1998]. Portanto, o AG é uma proposta adequada para encontrar solução para o problema de alocação de recursos, aqui representado pela Equação 11.

$$\text{Max} f(x) = \frac{\sum_{j=1}^{m} S_j^s(x)^w + \sum_{j=1}^{m} S_j^p(x)^w}{\sum_{j=1}^{m} B_j(x)} - \left(\sum_{j=1}^{m} I_i^s(x) + \sum_{j=1}^{m} I_i^p(x) + \right) + T(x) \tag{11}$$

O AG soluciona um problema de otimização gerando um conjunto de soluções aleatórias (população inicial) e aplicando, nesta ordem, os operadores *crossover*, mutação e seleção. O desempenho do AG depende da codificação da solução e de seus operadores genéticos crossover, mutação e seleção [Xiong et al. 2021].

3.2. Algoritmo Genético

A codificação da solução do problema de alocação de recursos, para que seja possível a aplicação dos operadores genéticos, foi definida da seguinte maneira: dado um cromossomo [0, 1, 0, 2] representa a alocação de 4 pods em 3 nós, e eles estão alocados da seguinte forma: o pod 0 está alocado no nó 0, o pod 1 está alocado no nó 1, o pod 2 está alocado no nó 0 e o pod 3 está alocado no nó 2 conforme ilustra a Figura 2.

Figura 2. Representação da distribuição de pods em nós no cluster k8s, conforme definido pelo vetor de alocação (cromossomo)

A primeira etapa do AG consiste em inicializar a população, que é um conjunto de indivíduos (soluções candidatas) aleatórios. Posteriormente, cada indivíduo tem sua aptidão avaliada de acordo com a Equação 11.

Os indivíduos com maior aptidão são selecionados (operador seleção) para o processo de cruzamento (*crossover*) para criar novos indivíduos (filhos) a partir das características dos pais e gerar uma nova população. Há ainda o processo de mutação, que adiciona uma camada de exploração, permitindo modificações abruptas e aleatórias nos cromossomos para evitar estagnação e explorar regiões inexploradas do espaço de busca.

A avaliação da aptidão dos filhos gerados fornece uma base para a seleção dos sobreviventes, onde os cromossomos mais aptos são preservados para a próxima geração. Esse ciclo de *crossover*, mutação, avaliação e seleção é repetido por várias gerações, permitindo que a população evolua e se adapte ao longo do tempo.

O problema de alocação de recursos possui soluções distintas com o mesmo valor objetivo. Na subseção 3.3, exemplificamos essa característica e como a Equação 11 distingue uma solução de boa qualidade das demais.

3.3. Representação da avaliação da aptidão

Dadas n alocações, surge a necessidade de classificar qual delas apresenta a melhor aptidão. Nesse sentido, foram formuladas quatro alocações: x_1, x_2, x_3 e x_4, para avaliar a qualidade de suas aptidões. Para este exemplo, considere-se: quatro alocações diferentes; a avaliação baseada apenas no consumo de CPU; e a ausência de uma taxa de relacionamento entre os pods a ser considerada. O objetivo principal é alocar os pods usando o menor número possível de nós e atender as retrições do problema, visando otimizar a utilização de recursos para futuras alocações e evitar a necessidade de realocação.

As Figuras 3 e 4 representam as alocações x_1 e x_2 que, em termos gerais, possuem a mesma proporção final de alocação considerando o somatório da porcentagem de recursos utilizados. A introdução do peso w atua de forma a distinguir as soluções e premiar aquela que otimize a utilização dos nós. Ao considerar $S_j^p(x)^w$ com $w = 2$, obtemos $f(x_1) = 0.25$ e $f(x_2) = 0.2916$, indicando que a alocação x_2 é melhor em termos da utilização dos recursos dos nós.

Comparativamente, de acordo com nosso objetivo, a solução x_3 (Figura 5) apresenta uma alocação melhor que a proposta por x_2 (Figura 4). A importância da divisão do total de CPU utilizado (S_j^p) por B_j reside no fato de que a qualidade da solução é inversamente proporcional ao total de nós utilizados na alocação, assim uma alocação mais eficiente é aquela com menos nós sendo necessários para atender à demanda por recursos. Nesse aspecto a alocação proposta por x_3 possui um valor objetivo de $f(x_3) = 0.5625$ indicando maior eficiência que a solução x_2.

A Figura 6 apresenta o cenário de x_4 onde ocorre uma infactibilidade que é penalizado pelo seu excedente (vide equações 9 e 10), retornando $f(x_4) = 0.4850$. A equação de infactibilidade tem como objetivo verificar se uma alocação é infactível e qualificar o quão inadequada é aquela alocação.

É importante destacar que, mesmo que uma alocação seja infactível, essa pode contribuir no processo evolutivo para se chegar ao ótimo global. Observa-se que $f(x_4)$ é inferior a $f(x_3)$ pois x_3 é uma alocação melhor do que x_4. Porém, $f(x_4)$ é maior $f(x_1)$ e $f(x_2)$, isso se deve ao fato de que a alocação x_4, ainda que infactível, está mais próxima do ótimo que as alocações x_1 e x_2. Isso evidencia o fato de que uma alocação, mesmo infactível, se faz importante dentro do processo evolutivo para que seja alcançado o ótimo

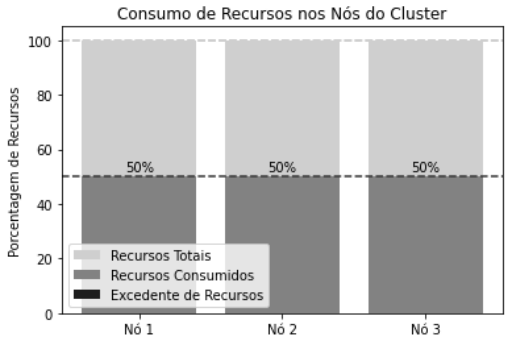

Figura 3. Alocação homogênea inicial, $f(x_1) = 0.2500$

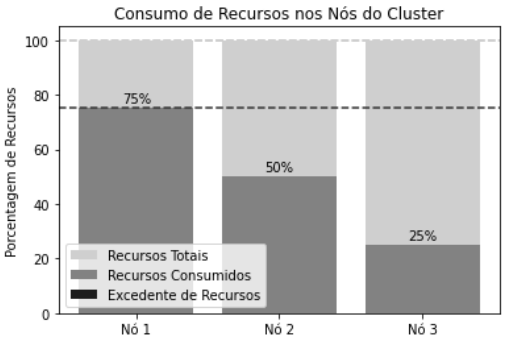

Figura 4. Alocação promissora, $f(x_2) = 0.2916$

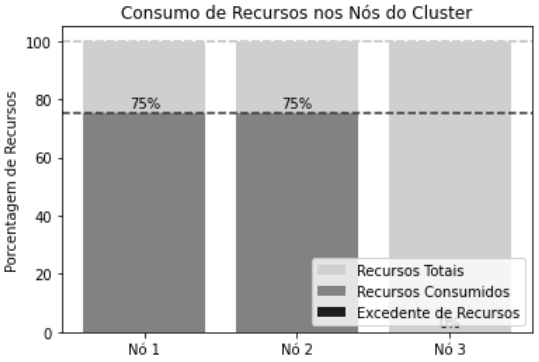

Figura 5. Alocação de alta qualidade, $f(x_3) = 0.5625$**.**

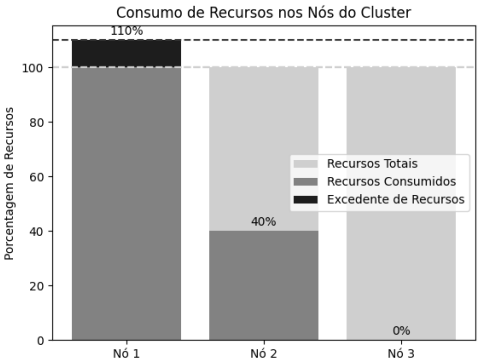

Figura 6. Alocação infactível penalizada, $f(x_4) = 0.4850$

global. Na Seção 4, são apresentados experimentos realizados para validar o modelo e o AG proposto.

4. Avaliação experimental

Para avaliar a eficiência do K8sGAScheduler, foram implantados pods com *Nginx* em três cenários diferentes em um ambiente *Kubernetes* hospedado na plataforma *Digital Ocean*. Durante a avaliação, a distribuição dos pods no cluster foi observada e os dados de alocação de recursos foram registrados para uma análise posterior. Simulações foram realizadas para analisar os resultados da alocação proposta pelo AG, configurado com parâmetros de recursos que refletiam as condições do ambiente de nuvem.

Os objetivos do experimento incluem estabelecer parâmetros adequados para uma comparação significativa entre os resultados dos diferentes cenários e validar a hipótese de que o método de agendamento do K8sGAScheduler supera o método padrão de agendamento do Kubernetes em eficácia na alocação de recursos para lotes de pods nos nós do *cluster*.

4.1. Configurações do Algoritmo Genético

O AG desenvolvido foi implementado em Python. Para todos os cenários de avaliação, as variáveis de configuração do AG foram: a) Tamanho da população: 100 indivíduos; b)

Número de gerações: 100 (o critério de parada); c) Probabilidade de cruzamento: 80%; d) Probabilidade de mutação: 20%.

O tamanho da população e o número de gerações são parâmetros importantes e tem impacto direto na eficiência do AG. Um tamanho de população maior permite que o AG explore um espaço de busca maior, o que aumenta as chances de encontrar uma solução satisfatória. Um número maior de gerações permite que o AG tenha mais tempo para explorar o espaço de busca, o que aumenta as chances de encontrar uma solução satisfatória. No entanto, um tamanho de população e gerações grande pode levar a um aumento no tempo de execução do AG [Roeva et al. 2013].

O algoritmo desenvolvido para validação desse trabalho buscou o equilíbrio entre o espaço de busca e o tempo de execução. A partir da análise da evolução da aptidão ao longo das gerações, observou-se que o algoritmo converge para um valor constante antes da centésima geração o que levou a definição de 100 gerações para estes experimentos.

4.2. Cenários de Avaliação

Para o cenário de avaliação foi utilizado um cluster na *Digital Ocean* com três *worker nodes*, cada *node* com 2 vCPUs, 2GB de RAM e 60GB de disco. Para implementação do algoritmo genético foi utilizado a linguagem python e seu código fonte está disponível publicamente no GitHub[1].

Para simular diferentes capacidades de alocação, foram projetados três cenários diferentes intitulados: Baixa demanda, Média demanda e Alta demanda, com alocação de 20, 25 e 30 pods, respectivamente.

1. Baixa demanda - 20 pods:
    - 20 pods com CPU de 50 milicores e memória de 64MB.
2. Média demanda - 25 pods:
    - 15 pods com CPU de 50 milicores e memória de 64MB.
    - 10 pods com CPU de 100 milicores e memória de 128MB.
3. Alta demanda - 30 pods:
    - 20 pods com CPU de 50 milicores e memória de 64MB.
    - 10 pods com CPU de 100 milicores e memória de 128MB.

4.3. Estudo de Caso

No contexto deste estudo, a avaliação da eficiência do K8sGAScheduler baseia-se na maximização da aptidão, refletindo a qualidade das alocações de recursos no *cluster*.

Inicialmente foi testado o cenário do cluster na *Digital Ocean*. Para avaliar o comportamento do *scheduler* padrão foi feito o *deployment* de três cenários conforme descrito na Seção 4.2.

O primeiro *deploy*, do cenário de baixa demanda, registrou a seguinte alocação [0,1,1,1,0,2,0,2,2,1,1,2,0,0,1,0,0,2,2,2]. Na sequência outros dois *deployments* foram executados conforme os cenários de média demanda e alta demanda, retornando respectivamente as alocações: [0,2,1,2,1,2,2,1,0,1,1,0,2,0,0,1,0,0,1,2,2,0,1,0,2] e [1,2,2,0,2,1,0,0,1,2,1,1,0,2,0,2,2,1,0,0,0,2,0,1,2,2,1,1,1,0]. As aptidões resultantes de cada cenário foram registradas na Tabela 1.

[1]Disponível em: `https://github.com/LABORA-INF-UFG/paper-k8sgascheduler`

Tabela 1. Resumo das aptidões registradas para os *deployments* dos cenários de baixa, média e alta demanda no *cluster* da *Digital Ocean*

Benchmark	Baixa demanda	Média demanda	Alta demanda
Aptidão	0.0715	0.2194	0.2850

Para avaliar os resultados do K8sGASchedule foram executadas dois conjuntos de avaliação sendo: no primeiro ciclo de avaliação foi definido uma taxa de relacionamento de 0% (Tabela 2) para matriz de relacionamentos, representando a ausência de comunicação entre os pods para tomada de decisão de alocação. No segundo ciclo, essa taxa foi aumentada para 10% (Tabela 3), refletindo o preenchimento aleatório de 10% dos valores na matriz simétrica, representando o custo de comunicação entre os pods para a alocação. Cada um dos ciclos de avaliação foram executados cinquenta vezes utilizando o algoritmo K8sGAScheduler. Para cada ciclo de avaliação foi registrado: a melhor aptidão, a média global das aptidões, a mediana e o desvio padrão.

Tabela 2. Resultados do K8sGAScheduler para taxa de relacionamento de 0%

Benchmark	Baixa demanda	Média demanda	Alta demanda
Melhor Aptidão	0.6406	0.8272	0.8681
Média	0.3727	0.6124	0.5829
Mediana	0.2626	0.6350	0.5114
Desvio Padrão	0.2131	0.1896	0.1528

Ao analisar os resultados obtidos nas avaliações realizadas no *cluster* da *Digital Ocean*, notamos uma clara distinção na eficácia das alocações entre o *scheduler* padrão (Tabela 1) e o algoritmo K8sGAScheduler (Tabela 2). Na Tabela 1, os resultados alcançados pelo *scheduler* padrão foram, respectivamente, 0.0715, 0.2194 e 0.2850 para cada cenário. Enquanto isso, o K8sGAScheduler registrou aptidões de 0.6406, 0.8272 e 0.8681 respectivamente, para os mesmos cenários apresentando melhores alocações.

A alocação obtida pelo AG para o cenário de baixa demanda foi [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] comparando essa alocação com a do *Digital Ocean* verificamos que a nossa alocação ocupa apenas um nó, ressaltando que o modelo representa adequadamente nosso objetivo e que o AG é capaz de alcançar os resultados esperado. Para o cenário de média demanda a alocação apresentada foi [2,2,1,2,2,2,2,2,2,2,2,1,2,1,2,2,2,2,2,2,2,2,2,2,2] e para o cenário de alta demanda a alocação informada foi [0,0,2,2,0,2,2,2,2,2,2,0,2,2,2,2,2,0,0,2,2,2,2,2,0,2,2,2,2,2,2,2,2].

Além disso, ao observarmos os resultados das Tabelas 3 onde foi considerada uma taxa de relacionamento de 10%, o K8sGAScheduler foi capaz de manter sua eficiência de alocação. Por exemplo, para o cenário de média demanda com 10% de taxa de relacionamento obtivemos a alocação [1,1,1,1,1,1,1,2,1,2,1,1,2,1,1,1,1,1,1,1,1,1,2,1], que utiliza dois nós dos três disponíveis. Os pods 2 e 3 possuem taxa de relacionamento de 0.83 [2] e estão no mesmo nó, portanto o AG é capaz encontrar soluções que aloca pods com alta taxa de relacionamento no mesmo nó. Isso indica que o K8sGAScheduler é capaz

[2]Algoritmo media_demanda_tr_10.py disponível no GitHub linha 97

de considerar não apenas os recursos individuais dos pods, mas também suas interações, para otimizar a alocação de recursos no cluster.

Tabela 3. Resultados do K8sGAScheduler para taxa de relacionamento de 10%

Benchmark	Baixa demanda	Média demanda	Alta demanda
Melhor Aptidão	9.2406	11.6322	26.9056
Média	8.1718	9.9456	19.9561
Mediana	8.2465	0.6350	19.8525
Desvio Padrão	0.6360	0.9500	2.1639

Os resultados mostram que o K8sGAScheduler apresenta alocações eficientes, com aptidões superiores ao scheduler padrão em nossas avaliações. No entanto, há espaço para melhorias e refinamentos adicionais. Observa-se que o desvio padrão aumenta mediante a dificuldade de alocação (Tabela 3) indicando gradativa dispersão dos valores em relação a média. Planeja-se realizar ajustes no AG, buscando não apenas maximizar a aptidão das alocações, mas também reduzir o desvio padrão, tornando a média das alocações mais consistente.

5. Conclusão

Neste artigo, apresentamos a proposta de um modelo para otimizar a alocação de recursos em *clusters* Kubernetes, intitulado K8sGAScheduler. Observamos que o K8sGAScheduler demonstrou capacidade de fornecer aptidões significativamente mais altas em comparação com o *scheduler* padrão. Essa melhoria na alocação de recursos é particularmente relevante em ambientes de computação em nuvem, onde a otimização dos recursos disponíveis pode impactar diretamente no desempenho e na eficiência dos sistemas.

O modelo proposto (Equação 11) foi capaz de conduzir o AG para alocações de boa qualidade evidenciando a distinção entre soluções de baixa, média e alta qualidade, assim como, permitindo que soluções infactíveis façam parte da população minimizando a convergência precoce do AG.

No entanto, é importante reconhecer algumas limitações deste estudo. Embora nos concentremos nos critérios de CPU e memória, juntamente com o relacionamento entre os pods, outras métricas importantes, como utilização de disco e consumo energético, podem ser incorporadas ao modelo para torná-lo mais abrangente. Além disso, a introdução de um peso para ampliar a penalização de alocações infactíveis pode melhorar ainda mais a eficácia do modelo. Para os próximos passos, planejamos: a) aplicar o modelo em ambientes de produção para avaliar seu desempenho em cenários reais; b) realizar ajustes no algoritmo genético buscando reduzir o desvio padrão; c) inserir novas métricas ao modelo como utilização de disco e consumo energético; d) avaliar a eficácia na inserção de um peso para a penalização de alocações infactíveis, buscando aprimorar sua capacidade de otimização de recursos.

As contribuições deste estudo para a área de conhecimento incluem a demonstração da eficácia do modelo em conjunto com a heurística de algoritmo genético para otimização da alocação de recursos em ambientes de *cluster*, bem como a identificação de áreas para futuras pesquisas e desenvolvimentos.

Em resumo, este estudo fornece *insights* de que o K8sGAScheduler pode oferecer vantagens significativas em termos de alocação de recursos em ambientes de computação em nuvem. Essas descobertas têm implicações importantes para a prática, teoria e pesquisa futura nesta área em constante evolução.

Referências

Arouk, O. and Nikaein, N. (2020). 5g cloud-native: Network management automation. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–2.

Botez, R., Costa-Requena, J., Ivanciu, I.-A., Strautiu, V., and Dobrota, V. (2021). Sdn-based network slicing mechanism for a scalable 4g/5g core network: A kubernetes approach. *Sensors*, 21(11).

Burns, B., Beda, J., Hightower, K., and Evenson, L. (2022). *Kubernetes: up and running: dive into the future of infrastructure*. O'Reilly Media, Beijing Boston, third edition edition.

Chirivella Perez, E., Alcaraz Calero, J., Wang, Q., and Gutiérrez-Aguado, J. (2018). Orchestration architecture for automatic deployment of 5g services from bare metal in mobile edge computing infrastructure. *Wireless Communications and Mobile Computing*, 2018.

Chu, P. and Beasley, J. (1998). *Journal of Heuristics*, 4(1):63–86.

dell'Amico, M., Delorme, M., Iori, M., and Martello, S. (2019). Mathematical models and decomposition methods for the multiple knapsack problem. *Eur. J. Oper. Res.*, 274:886–899.

Farhi, E., Goldstone, J., Gutmann, S., Lapan, J., Lundgren, A., and Preda, D. (2001). A quantum adiabatic evolution algorithm applied to random instances of an np-complete problem. *Science*, 292(5516):472–475.

Martello, S. and Toth, P. (1980). Solution of the zero-one multiple knapsack problem. *European Journal of Operational Research*, 4(4):276–283. Combinational Optimization.

Menouer, T. (2021). Kcss: Kubernetes container scheduling strategy. *The Journal of Supercomputing*.

Mfula, H., Ylä-Jääski, A., and Nurminen, J. (2021). Seamless kubernetes cluster management in multi-cloud and edge 5g applications. In *International Conference on High Performance Computing Simulation (HPCS 2020)*. International Conference on High Performance Computing amp; Simulation, HPCS ; Conference date: 22-03-2021 Through 27-03-2021.

Nguyen, H. T., Do, T. V., and Rotter, C. (2021). Scaling upf instances in 5g/6g core with deep reinforcement learning. *IEEE Access*, PP:1–1.

Rejiba, Z. and Chamanara, J. (2022). Custom scheduling in kubernetes: A survey on common problems and solution approaches. *ACM Comput. Surv.*, 55(7).

Roeva, O., Fidanova, S., and Paprzycki, M. (2013). Influence of the population size on the genetic algorithm performance in case of cultivation process modelling. In *2013 Federated Conference on Computer Science and Information Systems*, pages 371–376.

Santos, J., Wauters, T., Volckaert, B., and De Turck, F. (2019). Towards network-aware resource provisioning in kubernetes for fog computing applications. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 351–359.

Townend, P., Clement, S., Burdett, D., Yang, R., Shaw, J., Slater, B., and Xu, J. (2019). Invited paper: Improving data center efficiency through holistic scheduling in kubernetes. In *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 156–15610.

Wojciechowski, , Opasiak, K., Latusek, J., Wereski, M., Morales, V., Kim, T., and Hong, M. (2021). Netmarks: Network metrics-aware kubernetes scheduler powered by service mesh. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–9.

Xiong, J., Chen, B., He, Z., Guan, W., and Chen, Y. (2021). Optimal design of community shuttles with an adaptive-operator-selection-based genetic algorithm. *Transportation Research Part C: Emerging Technologies*, 126:103109.

Zhang, X., Li, L., Wang, Y., Chen, E., and Shou, L. (2021). Zeus: Improving resource efficiency via workload colocation for massive kubernetes clusters. *IEEE Access*, 9:105192–105204.