Enabling power-awareness for Kubernetes scheduling through multi-criteria optimization

Mohammed Dhiya Eddine Gouaouri*,
Miloud Bagaa*, Sihem Ouahouah§, Messaoud Ahmed Ouameur * and Adlen Ksentini¶
* Université du Québec à Trois-Rivières, Trois-Rivières, QC, Canada.
Emails: {mohammed.dhyia.eddine.gouaouri, miloud.bagaa, messaoud.ahmed.ouameur}@uqtr.ca
, §Aalto University, Otakaari 24, 02150 Espoo FINLAND (e-mail: sihem.ouahouah@aalto.fi),
¶EURECOM, Campus SophiaTech, France (e-mail: adlen.ksentini@eurecom.fr)

***Abstract*—This paper proposes a new scheduling framework to optimize the placement of cloud workloads submitted online by users within a Kubernetes-orchestrated environment. The proposed method aims to incorporate power awareness during the scheduling process, along with other criteria, such, load balancing, and bin packing. The framework equitably distributes workloads across cluster nodes while also selecting the most resource-efficient node to reduce the number of active nodes and prevent resource fragmentation. Existing strategies often focus on a single criterion, leading to suboptimal and unsatisfactory workload placements. The proposed framework utilizes the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS), a well-known multi-criteria decision analysis algorithm, to account for power consumption and other criteria defined by cloud operators, such as load balancing and bin packing. The algorithm is implemented as a Kubernetes scheduling plugin to rank worker nodes based on these criteria and the submitted workloads. Simulation results demonstrate the effectiveness of the proposed strategy across various scenarios, reducing power consumption by 26.46% and comparable CPU and memory load balancing performance within a large Kubernetes cluster under heavy workloads.**

***Index Terms*—Cloud-Edge Computing Continuum, Scheduling, Multi-Objective Optimization, Kubernetes**

I. Introduction

Cloud computing has rapidly expanded across industries, enabling organizations to leverage scalable, virtualized resources for cost and time efficiency [1]. At its core, cloud computing offers virtualized resources through technologies like Virtual Machines (VMs) and containers, where VMs ensure tenant isolation, and containers allow fast, efficient application deployment [2]. Container technology evolved from early isolation methods such as chroot [3], advancing significantly with Docker in 2013, which simplified application packaging and deployment in isolated environments [4]. As container use grew, tools like Kubernetes emerged to orchestrate large-scale deployments, automating container management and scaling [5].

Today Kubernetes stands as the leading open-source container orchestration platform, valued for its portability and extensibility across various infrastructures. While Kubernetes efficiently handles container deployment and scaling, its current scheduling focuses mainly on resource metrics (e.g., CPU, memory) and may not optimize for energy efficiency [6]. In high-load cloud environments, efficient workload placement is critical for optimizing performance and energy use. Standard scheduling methods, however, focus mainly on resource metrics, often leading to higher power consumption, reduced cluster efficiency, and greater carbon emissions—problems that grow with infrastructure scale. This highlights the need for eco-friendly, energy-aware infrastructure aligned with sustainability goals like those in the European Green Deal [7]. Integrating power-awareness into workload scheduling offers significant advantages across key use cases. In telecommunication, where Kubernetes is used for 5G and 6G deployments, traffic patterns fluctuate, requiring efficient scheduling to balance energy use with service demands. Similarly, in edge computing for IoT devices, energy-efficient scheduling extends device lifespan. Applications in autonomous vehicles, smart city infrastructure, and hyperscale data centers also benefit from power-aware scheduling to optimize energy use and resource allocation. To address the limitations in current scheduling approaches, we propose a novel multi-criteria framework for Kubernetes workload scheduling that simultaneously optimizes load balancing for high availability, bin packing to optimize resource utilization by fitting workloads onto fewer nodes, and power consumption. Implemented as a scalable, customizable Kubernetes plugin, our solution allows users to set Quality of Service (QoS) requirements, integrating seamlessly with other scheduling plugins.

The remainder of this paper is organized as follows: Section II reviews the related work. Section III presents our methodology. Section IV provides evaluation results and analysis, and Section V concludes the paper.

II. Background and Related works

In the literature, resource scheduling is part of the broader field of resource allocation and management, where the goal is to assign tasks or jobs to a set of machines or workers to optimize resource utilization or job completion time.

This problem has been shown to be NP-hard, meaning no algorithm can solve it in polynomial time unless $P = NP$ [8]. As a result, approximation algorithms are often used to obtain fast and satisfactory solutions.

Cloud orchestration systems like Kubernetes use a scheduling component to assign workloads to machines. Many studies have aimed to improve this component with objectives such as energy efficiency, load balancing, and task prioritization. For instance, [9] developed a Kubernetes scheduler plugin to minimize latency between dependent workloads by modeling them as an acyclic dependency graph. Similarly, [10] introduced a topology-aware scheduler for IoT services at the edge, using a cluster graph to account for network infrastructure.

[11] proposed a Docker Swarm scheduler using the TOPSIS multi-criteria decision-making algorithm [12], focusing on container count, CPU, and memory. However, their approach overlooks power consumption, an important factor in cloud environments. Later, [13] extended this by including power consumption in a Kubernetes scheduler with TOPSIS. Although effective, their solution uses simulated power data and lacks compatibility with other Kubernetes plugins. In contrast, our approach supports real power estimation by profiling worker nodes and offers a scheduling algorithm as a plugin, enabling integration with other Kubernetes plugins for broader use cases.

A. Multi-criteria decision making algorithms

Multi-criteria decision-making (MCDM) is a critical process used in fields like supply chain management, engineering, and economics to support optimal decision-making [14]. A foundational method in this area is the *Kung algorithm* by [15], designed for multi-objective optimization. This algorithm identifies Pareto-optimal solutions, where enhancing one objective would compromise another. It uses a divide-and-conquer strategy to find these optimal points by splitting the solution set into smaller subsets, determining Pareto fronts for each, and merging them—a technique widely used in economics and engineering.

Another significant contribution is TOPSIS, introduced by [12]. This method uses a decision matrix for alternative solutions, applying normalization and weighting to find the option closest to an ideal solution and farthest from a negative ideal. Further details on TOPSIS are discussed in section III.

Our work advances existing efforts by incorporating power awareness directly into the Kubernetes scheduling process through a multi-objective optimization framework. Our approach uniquely combines load balancing for workload spreadness to enhance cluster high availability with bin packing strategies that maximize resource utilization, minimize node usage, and significantly reduce overall power consumption.

III. Methodology

The goal of this work is to enhance Kubernetes scheduling by selecting the most suitable worker node to run a pod within a multi-criteria configuration system. This approach aims to optimize resource load balancing, bin packing, and power consumption. These goals often conflict; for example, prioritizing load balancing may lead to resource wastage or increased power consumption. To address this, we propose a new scheduling framework based on TOPSIS for multi-criteria optimization. The choice of this algorithm for this purpose was due to its extensive use for fast search of good solutions within a discrete multi-criteria optimization problem [16].

The scheduling framework, as depicted in Fig. 1, operates by evaluating worker nodes based on their CPU and memory usage, as well as their power model, P_i.

Once a pod is submitted to the queue Q, the scheduler runs a filtering phase that executes some predicates, such as *PodFitsResources* to make sure that the pod can fit the resources available in that node.

This phase outputs a set of candidate nodes $\mathcal{N}$ that proceed to the scoring phase, which executes the workflow described in III-A. Given the set of criteria (objectives) and their associated weights reflecting the relative importance of each criterion, the framework identifies the optimal node. This node is the one closest to the ideal solution, where all criteria achieve their best possible values, and furthest from the worst solution, where all criteria are at their least desirable values. Additionally, the power model of each node is determined through a power profiling step, which maps CPU utilization to the estimated power consumption of the node if selected to host the pod. While several factors may influence power consumption, CPU utilization has been observed to be the most significant factor [17].

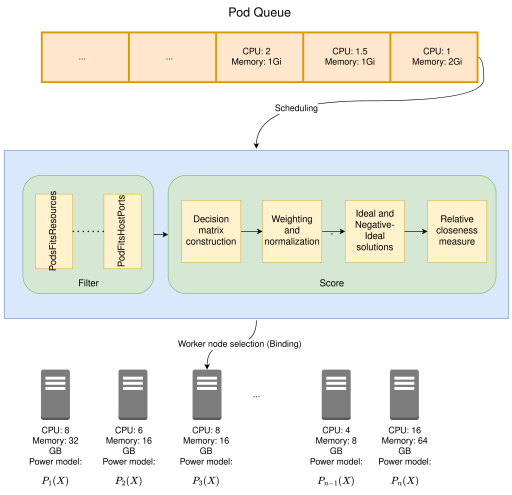

Fig. 1: Kubernetes power-aware scheduling workflow

A. Scoring workflow with TOPSIS

1) Decision matrix construction: The algorithm begins by constructing a decision matrix, where rows represent the alternative solutions (i.e., the worker nodes) and columns represent the criteria to optimize. Each criterion is classified as either a benefit criterion (to be maximized) or a cost criterion (to be minimized). The set of benefit criteria is denoted by $\mathcal{B}$, and the set of cost criteria is denoted by $\mathcal{C}$.

Let X be the decision matrix where x_{ij} represents the value of alternative $i \in \mathcal{N}$ for criterion $j \in \mathcal{O}$:

$$X = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{pmatrix}$$

where:

- m: The number of alternatives (worker nodes).
- n: The number of criteria.
- $\mathcal{O}$: The set of criteria, including the number of pods, CPU utilization, memory consumption, and power consumption.

2) Normalization: The next step is to normalize the columns of the matrix to make the criteria comparable. The normalized decision matrix R is obtained by:

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} x_{ij}^2}}$$

$$R = \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ r_{21} & r_{22} & \dots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m1} & r_{m2} & \dots & r_{mn} \end{pmatrix}$$

3) Weighted Normalized Decision Matrix: Each criterion is assigned a weight, w_j, based on its relative importance defined by an expert. The weighted normalized value v_{ij} is calculated as:

$$v_{ij} = w_j \times r_{ij}$$

Where $w_j \in [0, 1]$ and $\sum_{j=1}^{n} w_j = 1$.

Thus, the weighted normalized decision matrix V becomes:

$$V = \begin{pmatrix} v_{11} & v_{12} & \dots & v_{1n} \\ v_{21} & v_{22} & \dots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{m1} & v_{m2} & \dots & v_{mn} \end{pmatrix}$$

4) Determine the Ideal and Negative-Ideal Solutions: In the TOPSIS method, the ideal and negative-ideal solutions serve as benchmarks, representing the best and worst possible outcomes for each criterion. These solutions are used to rank alternatives based on how close they are to the ideal and how far they are from the negative-ideal.

- **Ideal Solution** A^+: This solution consists of the best values for each criterion. For benefit criteria (where higher values are preferable), it takes the maximum value across all alternatives. For cost criteria (where lower values are better), it selects the minimum value:

  $$A^+ = (\max(v_{ij}) \,|\, j \in \mathcal{B}, \min(v_{ij}) \,|\, j \in \mathcal{C})$$

  where v_{ij} is the value of the j-th criterion for the i-th alternative. This represents an optimal state for all criteria.
- **Negative-Ideal Solution** A^-: This solution is composed of the worst values for each criterion. For benefit criteria, it takes the minimum value across all alternatives, while for cost criteria, it selects the maximum value:

  $$A^- = (\min(v_{ij}) \,|\, j \in \mathcal{B}, \max(v_{ij}) \,|\, j \in \mathcal{C})$$

  The negative-ideal solution represents the least favorable outcome for all criteria.

5) Distance from the Ideal and Negative-Ideal Solutions: For each solution i, the Euclidean distance from the ideal and negative-ideal solutions is calculated as follows:

- **Distance from the Ideal Solution** D_i^+:

  $$D_i^+ = \sqrt{\sum_{j=1}^{n} (v_{ij} - A_j^+)^2}$$

- **Distance from the Negative-Ideal Solution** D_i^-:

  $$D_i^- = \sqrt{\sum_{j=1}^{n} (v_{ij} - A_j^-)^2}$$

6) Relative Closeness to the Ideal Solution: The relative closeness of alternative i to the ideal solution is given by:

$$C_i^* = \frac{D_i^-}{D_i^+ + D_i^-}$$

where $0 \leq C_i^* \leq 1$. The closer C_i^* is to 1, the better the alternative.

The pod is scheduled on the node with the highest relative closeness. Algorithm 1 summarizes these steps.

IV. Experimentation

A. Simulation setup

To validate our proposal that aims to develop a Kubernetes scheduler plugin, we conducted a series of experiments using the *kube-scheduler-simulator*[1] framework. A tool that replicates the Kubernetes scheduling process in a controlled environment. This simulator allows us to test custom scheduling algorithms with various configurations and supports large-scale deployments with thousands of

[1] https://github.com/kubernetes-sigs/kube-scheduler-simulator

Algorithm 1: Power-aware scheduling with TOPSIS Optimization

Input: Queue of arriving pods Q, Set of worker nodes $\mathcal{N}$, Criteria weights vector w

foreach $p \in Q$ **do**
- $\mathrm{X} \leftarrow \texttt{createDecisionMatrix}(p, \mathcal{N})$;
- $\mathrm{R} \leftarrow \texttt{normalize}(\mathrm{X})$;
- $\mathrm{V} \leftarrow \texttt{applyWeights}(\mathrm{R}, w)$;
- $\mathrm{A}^+ \leftarrow \texttt{idealSolution}(\mathrm{V})$;
- $\mathrm{A}^- \leftarrow \texttt{negativeIdealSolution}(\mathrm{V})$;
- $D^+ \leftarrow \texttt{calculateDistance}(\mathrm{V}, \mathrm{A}^+)$;
- $D^- \leftarrow \texttt{calculateDistance}(\mathrm{V}, \mathrm{A}^-)$;
- $C^* \leftarrow \texttt{closeness}(D^+, D^-)$;
- $x^* \leftarrow \texttt{argmax}(C^*)$;
- $\texttt{schedulePod}(p, x^*)$;

nodes and pods. It includes visualization tools and logs to analyze scheduling decisions and performance, making it ideal for experimenting with and improving scheduling strategies.

The worker nodes specifications such as allocatable CPUs and available memory are generated randomly in such a way to emulate real cluster setup in which we have small, medium and large sized nodes. The following table describes CPU and memory ranges for different flavors:

TABLE I: CPU and Memory Specifications for Different Node Flavors

Node Flavor	**CPU Range (Cores)**	**Memory Range (GiB)**
Small	[2, 4]	[8, 16]
Medium	[8, 16]	[32. 64]
Large	[32, 64]	[128, 256]

The power consumption for nodes is generated using an exponential model used in [18].

$$P(x) = k_0 + k_1 \cdot \exp^{-k_2 \cdot x} \tag{1}$$

Where x is CPU utilization, k_0 represents the baseline power consumption of the node, k_1 determines the maximum additional power consumption as CPU usage increases, and k_2 controls the rate at which power consumption rises with utilization, influencing how quickly it saturates.

We conducted three experiments , as presented in table II: (1) a small cluster of 10 nodes with 100 pods to establish a baseline, (2) a mid-sized cluster of 30 nodes with 100 pods to evaluate scalability, and (3) the same mid-sized cluster under a heavy workload of 500 pods to assess robustness under high demand.

To enable a custom scheduler in Kubernetes scheduling framework, we just need to create a new scheduler profile and provide the configurations needed by our plugin. Our scheduler profile named *power-aware-scheduler* is presented in listing IV-A.

TABLE II: Cluster and workload sizes for each experiment

Experiment	Number of Nodes	Number of Pods
Experiment 1	10	100
Experiment 2	30	100
Experiment 3	30	500

```
...
profiles:
    schedulerName: power aware scheduler
    plugins:
      multiPoint:
        enabled:
          name: PowerAware
        pluginConfig:
            name: PowerAware
            args:
              kind: PowerAwareArgs
              apiVersion: kubescheduler.config.k8s.
    io/v1
              podLoadBalancingCriteria:
                weight: 0.2
                type: "Cost"
              cpuCriteria:
                weight: 0.2
                type: "Cost"
              memCriteria:
                weight: 0.2
                type: "Cost"
              powerCriteria:
                weight: 0.4
                type: "Cost"
```

Listing 1: Kubernetes Scheduler Configuration

The configuration profile shown in Listing IV-A enables the custom *PowerAware* plugin and specifies the set of criteria for scheduling decisions. Each criterion includes two attributes: *weight* and *type*. The *weight* attribute is used in the weighting phase of the decision matrix within the TOPSIS workflow, determining the relative importance of each criterion. In this case, a high importance is given to power consumption. The *type* attribute indicates whether the criterion is a *Cost*, which the scheduler aims to minimize, or a *Benefit*, which the scheduler seeks to maximize.

Indeed, the criteria defined for our plugin are as follows:

- *podLoadBalancingCriteria* is a cost criterion that encourages the scheduler to select nodes with fewer pods, improving load balancing across the cluster. If the user wants to pack more pods within a node, they would change its type to *Benefit*.
- *cpuCriteria* and *memCriteria* are defined as cost criteria to promote bin packing, aiming to optimize resource utilization by concentrating workloads onto fewer nodes.
- *powerCriteria*, also a cost criterion, is assigned the highest weight, indicating the priority to minimize power consumption during scheduling.

This balanced approach ensures efficient resource allocation while reducing overall energy consumption in the cluster.

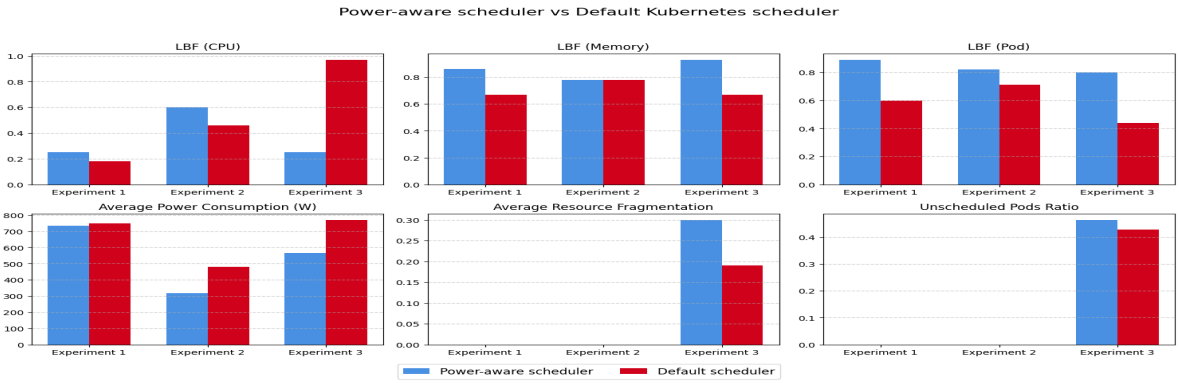

Fig. 2: Comparison of Power-aware scheduler vs Default scheduler across experiments for various metrics.

B. Evaluation and results

To evaluate the effectiveness of our scheduling algorithm and the power integration in the scheduler, we define the following key metrics:

a) Load Balancing Factor: Load balancing is an important metric to optimize, as it describes how well the pods are distributed across the machines. An optimal load balancing factor minimizes performance bottlenecks by preventing any single machine from being overloaded while avoiding under utilization of others.

We define the resource-wise *Load Balancing Factor* with respect to a resource r as:

$$LBF_r = \frac{\sigma(\mathcal{R})}{\mu(\mathcal{R})}$$

Where:

- r: A resource type, where $r \in \{\text{cpu}, \text{memory}\}$.
- $\mathcal{R}$: The set of resource usages of resource type r over cluster nodes.
- σ: The standard deviation of the set of resource usages $\mathcal{R}$ of resource r.
- μ: The mean of the set $\mathcal{R}$.

The LBF_r is formulated as a coefficient of variation [19], which means that a lower value of LBF_r indicates a better distribution of the workload across the nodes of the cluster.

We also define the pod count-wise load balancing factor as follows:

$$LBF_{\text{pod}} = \frac{\sigma(\mathcal{P})}{\mu(\mathcal{P})}$$

Where $\mathcal{P}$ is the set of pod counts across the cluster nodes.

b) Resource Fragmentation Factor: Resource fragmentation refers to the inefficient use of resources, where small, scattered portions of unused resources are unlikely to be allocated, increasing the number of pods that cannot be scheduled.

For a node i, we define the unused resources vector u as follows:

$$u = \begin{bmatrix} \text{Unused CPU} \\ \text{Unused Memory} \end{bmatrix}$$

The total resources vector v is defined as:

$$v = \begin{bmatrix} \text{Total CPU} \\ \text{Total Memory} \end{bmatrix}$$

The resource fragmentation factor for machine i is defined as:

$$RF_i = \frac{||u||_2}{||v||_2} \times unschedulable\ pods\ ratio$$

Where

$$unschedulable\ pods\ ratio = \frac{unschedulable\ pods\ count}{Total\ number\ of\ pods}$$

The average resource fragmentation factor over all cluster nodes is:

$$\text{RF} = \frac{\sum_{i=1}^{m} RF_i}{m}$$

A lower value of RF indicates a lower resource fragmentation, hence a better resource utilization.

C. Results analysis

Fig. 2 presents the evaluation results of our power-aware scheduler with TOPSIS algorithm compared to the default Kubernetes scheduler across three experiments to evaluate the robustness of our proposed approach. In the first experiment, the CPU, memory, and pod count balancing factors for our power-aware scheduler were slightly higher than those of the default scheduler, yet still indicate effective resource distribution. Additionally, our scheduler achieved a modest reduction in average power consumption, with 735.68 W compared to 750.19 W for the default scheduler. For the second experiment and with a larger node count, our scheduler achieves a balanced CPU load with an LBF of 0.60, a slightly worse than the default's 0.46. However, it significantly reduces power consumption, achieving

317.55 W compared to the default's 480.69 W, which is roughly 34% power reduction. Both schedulers showed stable results in handling resource fragmentation and the ratio of unscheduled pods. In the last experiment, under a high workload, our scheduler shows a slightly higher unscheduled pod ratio and resource fragmentation than the default scheduler. Our scheduler achieves an effective CPU load balancing of 0.25, which is significantly lower than the default scheduler's 0.97. This improvement may partly result from the higher number of unscheduled pods, suggesting that further analysis is needed to confirm whether this indicates a clear advantage in CPU load balancing.

D. Discussion

Our experiments compared the power-aware scheduler with the default Kubernetes scheduler across varying workloads and cluster sizes, yielding several insights:

- **Baseline Efficiency**: In a small cluster, it offered stable resource distribution and a slight (2%) reduction in power consumption.
- **Significant Power Savings**: In a mid-sized cluster, power consumption decreased by 34%, demonstrating its effectiveness in energy-sensitive environments.
- **High-Load Resilience**: Under heavy workloads, it maintained balanced CPU utilization, reducing power usage by 26.46%, though with minor increases in resource fragmentation and unscheduled pods.

While the default scheduler remains slightly more optimized for resource usage, our power-aware scheduler demonstrates considerable potential for reducing carbon footprint and promoting sustainable computing within Kubernetes clusters, balancing resource efficiency and availability.

V. CONCLUSION

In this paper, we introduced a novel Kubernetes scheduling framework that incorporates power consumption awareness in pod placement. Implemented as a Kubernetes scheduler plugin, our framework uses the TOPSIS algorithm to balance multiple criteria: pod load balancing, bin packing for efficient resource use, and node power consumption.

Our experiments show that this approach effectively reduces cluster power consumption by 26.46% under high workloads compared to the default Kubernetes scheduler. Although some trade-offs in load balancing were observed, operators can adjust criteria weights flexibly to prioritize objectives as needed.

The framework is highly configurable, supporting scalability across multiple criteria and nodes while allowing seamless integration with other scheduling plugins. This provides a flexible, energy-efficient scheduling solution for Kubernetes clusters in diverse operational settings. Future works will consider dynamic weight assignment depending on the workload behaviour and priorities in real-time.

REFERENCES

[1] A. K. Roy, "Enhancing cloud cost efficiency: A predictive ml approach for optimized resource allocation based on infrastructure usage and workload behavior," *International journal of science and research*, 2023.

[2] A. K and P. G., "Container-to-fog service integration using the dis-lc algorithm," *International Journal of Computer Network and Information Security*, 2024.

[3] N. Bhaia, L.-H. Hung, R. Cordingly, and W. Lloyd, "Understanding container isolation: An investigation of performance implications of container runtimes," in *Proceedings of the 9th International Workshop on Container Technologies and Container Clouds*, 2023, pp. 7–12.

[4] J. M. Ramavat and K. S. Patel, "Harmonizing heterogeneous hosts: A strategic framework for docker container placement optimization," *International Journal of Engineering Trends and Technology*, vol. 72, no. 7, pp. 58–68, 2024. [Online]. Available: https://doi.org/10.14445/22315381/IJETT-V72I7P106

[5] M. Sureshkumar and P. Rajesh, "Optimizing the docker container usage based on load scheduling," in *2017 2nd International Conference on Computing and Communications Technologies (ICCCT)*. IEEE, 2017, pp. 165–168.

[6] T. Goethals, F. De Turck, and B. Volckaert, "Extending kubernetes clusters to low-resource edge devices using virtual kubelets," *IEEE Transactions on Cloud Computing*, vol. 10, no. 4, pp. 2623–2636, 2020.

[7] "The european green deal," https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/european-green-deal˙en, 2024, accessed: 2024.

[8] J. Y.-T. Leung, Ed., *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Boca Raton, FL: CRC Press, 2004.

[9] J. Santos, T. Wauters, B. Volckaert, and F. D. Turck, "Towards network-aware resource provisioning in kubernetes for fog computing applications," in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, Jun 2019, pp. 351–359.

[10] S. Nastic, T. Pusztai, A. Morichetta, V. C. Pujol, S. Dustdar, D. Vii, and Y. Xiong, "Polaris scheduler: Edge sensitive and slo aware workload scheduling in cloud-edge-iot clusters," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 2021, pp. 206–216.

[11] T. Menouer and P. Darmon, "New scheduling strategy based on multi-criteria decision algorithm," in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, Feb 2019, pp. 101–107.

[12] C. Hwang and K. Yoon, *Multiple Attribute Decision Making: Methods and Applications*. New York: Springer-Verlag, 1981.

[13] T. Menouer, "Kcss: Kubernetes container scheduling strategy," *The Journal of Supercomputing*, vol. 77, no. 5, pp. 4267–4293, May 2021.

[14] S. K. Sahoo and S. S. Goswami, "A comprehensive review of multiple criteria decision-making (mcdm) methods: advancements, applications, and future directions," *Decision Making Advances*, vol. 1, no. 1, pp. 25–48, 2023.

[15] H.-T. Kung, F. Luccio, and F. P. Preparata, "On finding the maxima of a set of vectors," in *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*. ACM, 1975, pp. 37–46.

[16] M. S. Kumar, A. Tomar, and P. K. Jana, "Multi-objective workflow scheduling scheme: a multi-criteria decision making approach," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 12, pp. 10 789–10 808, 2021.

[17] H. Ahvar, A.-C. Orgerie, and A. Lebre, "Estimating energy consumption of cloud, fog and edge computing infrastructures," *IEEE Transactions on Sustainable Computing*, vol. 7, no. 2, pp. 277–288, 2022. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02083080

[18] S. C. IO, "Peaks: Design overview," https://github.com/sustainable-computing-io/peaks/blob/main/docs/design/overview.md, accessed: 2024-10-24.

[19] F. Bertolino, S. Columbu, M. Manca, and M. Musio, "Comparison of two coefficients of variation: a new bayesian approach," *Communications in Statistics-Simulation and Computation*, pp. 1–14, 2023.