





A Multi-Objective Framework for Power-Aware Scheduling in Kubernetes

Mohammed Dhiya Eddine Gouaouri^{*}, Sihem Ouahouah[§], Miloud Bagaa^{*}, Messaoud Ahmed Ouameur^{*},
Adlen Ksentini[¶]

^{*}Department of Electrical and Computer Engineering, Université du Québec à Trois-Rivières, Trois-Rivières, QC, Canada

Emails: {mohammed.dhiya.eddine.gouaouri, miloud.bagaa, messaoud.ahmed.ouameur}@uqtr.ca

[§]Department of Communications and Networking, School of Electrical Engineering, Aalto University, Espoo, Finland

Email: sihem.ouahouah@aalto.fi

[¶]Communication Systems Department, EURECOM, Campus SophiaTech, 06410 Biot, France

Email: adlen.ksentini@eurecom.fr

Abstract—Efficient workload scheduling in Kubernetes is crucial for optimizing energy consumption and resource utilization in large-scale and heterogeneous clusters. However, existing Kubernetes schedulers either ignore power-awareness or rely on simplified, static power models, which limit their effectiveness in managing energy efficiency under dynamic workloads. To address these shortcomings, we present a multi-objective scheduling framework for online Kubernetes pod placement that jointly considers power consumption, resource utilization, and load balancing. The framework follows a two-stage design: (i) a node power-profiling component trains a machine-learning model from real power measurements to predict per-node consumption under varying utilizations; and (ii) an online scheduler uses these predictions within a multi-objective optimization formulation. We implement scheduling optimization using two algorithms, TOPSIS and NSGA-II, adapting them to the Kubernetes context, and also propose a distributed variant of the NSGA-II algorithm that parallelizes fitness evaluation with controlled migration between workers. Experimental results show that the proposed framework outperforms baseline schedulers, achieving a 40% reduction in power consumption and improvements of 74% and 68% in CPU and memory utilization, respectively, while sustaining scalability under high workloads. To the best of our knowledge, this is the first work to integrate learned power models and distributed multi-objective optimization into Kubernetes for power-aware pod scheduling.

Index Terms—Scheduling, Power-Aware Scheduling, Multi-Objective Optimization, Kubernetes, NSGA-II, TOPSIS

I. INTRODUCTION

CLOUD computing has rapidly grown in both industry and academia, as more organizations adopt cloud-based resources and tools to save time and reduce costs [1]. The core aim of cloud computing is to offer virtualized, scalable, and efficient resources to clients. This is made possible through the integration of Virtual Machines (VMs) and container technologies. VMs provide strong isolation, enabling multiple tenants to share the same hardware securely, while containers, defined as lightweight, software-level virtualization technologies, offer fast deployment, efficient management, and enhanced utilization of computing resources [2].

Historically, container technology evolved from early isolation methods such as chroot [3]. In 2008, Linux Containers (LXC) introduced process and file system isolation to Linux systems [4]. The modern era of containerization began with Docker in 2013, which greatly simplified the packaging, deployment, and execution of applications in isolated environments [5]. As container usage grew, managing large-scale container deployments became challenging, leading to the development of container orchestration systems. Kubernetes, open-sourced by Google in 2014, emerged as the leading orchestration tool, providing automated container deployment, scaling, and management. While Docker also introduced Swarm, Kubernetes quickly became the preferred solution due to its robustness and scalability [6].

Kubernetes is now a widely adopted, open-source platform known for its portability and extensibility, making it ideal for diverse infrastructures and applications [7]. It automates container scaling and allocates containers to physical nodes. However, its default scheduling strategy is a rule-based method that primarily focuses on how to effectively fit workloads given different resource types (CPU, memory, disk, etc.), which may not always be optimal [8]. Kubernetes follows a leader-follower architecture, with the master node communicating with worker nodes via the kubelet component, responsible for container lifecycle management in the worker nodes.

In high-load cloud environments, efficient workload placement is critical for optimizing performance and energy use. Standard scheduling methods, however, focus mainly on traditional resource metrics such as CPU and memory, 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 [9]. Integrating power-awareness into workload scheduling offers significant advantages across key use cases. In telecommunications, where Kubernetes is used for 5G and 6G deployments, traffic patterns fluctuate, requiring efficient scheduling to balance energy use

with service demands [10]. 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-objective framework for Kubernetes workload scheduling that simultaneously optimizes three key, and often conflicting, objectives: (i) minimizing active nodes (via workload consolidation) to reduce power consumption and operational costs, (ii) load balancing to distribute workloads evenly across the active nodes for high availability and to prevent performance-degrading hotspots, and (iii) maximizing resource utilization efficiency on the active nodes. Implemented as a scalable and customizable Kubernetes scheduler plugin, our solution allows users to specify Quality of Service (QoS) requirements and is designed to collaborate with existing Kubernetes scheduling plugins, enhancing their functionality.

The main contributions and novelties of the presented study are summarized as follows.

- 1) We introduce a novel two-stage scheduling framework that integrates a machine learning-based power consumption predictor, trained on real measurement data collected through node-level power profiling. The predictive model is subsequently leveraged in the online scheduling stage, enabling energy-aware decision-making during pod placement.
- 2) The scheduler integrates two distinct multi-optimization techniques: TOPSIS and NSGA-II. TOPSIS, a scalarization-based method that transforms the problem into a single objective by combining multiple criteria into a weighted sum. NSGA-II, a metaheuristic genetic algorithm that directly explores the trade-offs between conflicting objectives without scalarization, generating a diverse set of Pareto-optimal solutions. By implementing both techniques, we aim to evaluate and compare their effectiveness and trade-offs in solving multi-objective scheduling problems within Kubernetes, examining differences in solution quality and scalability.
- 3) We also propose an accelerated version of NSGA-II-based scheduler algorithm through parallel computing to avoid the overhead due to large cluster sizes.
- 4) Our solution is implemented as a Kubernetes scheduling plugin that can seamlessly integrate with other plugins. It is highly scalable, allowing users to define additional objectives based on their specific Quality of Service (QoS) requirements.

The remainder of this paper is organized as follows: Section II reviews the related works. Section IV presents a rigorous formulation of the pod scheduling problem and our proposed methodology. Section V presents the details of experiments results and analysis, and Section VI concludes the paper.

II. RELATED WORKS

Several research works have sought to refine the scheduling process of container orchestration systems such as Kubernetes

and its alternatives, aiming to minimize communication latency and improve load balancing. The authors in [11], [17] introduced a Kubernetes scheduler plugin for microservice-based applications, modeled as an acyclic dependency graph, to reduce latency between pods within the same dependency graph. Their heuristic sorts the pods using topological order and schedules them onto nodes based on bandwidth requirements only. The work of [12] presents a framework for network-aware dynamic scheduling and migration of containers on mobile clusters based on Kubernetes, named Kinitos. Similar to Diktyo, this work leverages the microservice dependency graph during scheduling. However, both Diktyo [11] and Kinitos [12] ignore power consumption, in contrast to our work which integrates ML-based power prediction with multi-objective optimization algorithms to jointly optimize load balancing, resource fragmentation, and energy efficiency.

The authors in [18] presented a Docker Swarm container scheduler that employs the TOPSIS multi-objective decision-making algorithm [19], which considered only a limited set of objectives: the number of running containers on a node, CPU availability, and memory space. However, their work did not consider power consumption, a critical factor in cloud and edge infrastructures. In [16], the authors extended their work by incorporating power consumption into the TOPSIS optimization algorithm for Kubernetes. Their method outperformed the default scheduler but relied on simulated power data from CloudSim¹, which limits applicability in real clusters. Moreover, their work did not address the online scheduling problem of pods. In contrast, our work uses real power profiling of worker nodes to train node-specific power consumption models and implements online scheduling as a Kubernetes plugin, ensuring integration with other plugins in practical deployments.

In [20], a linear programming model was proposed to optimize container placement with objectives including host energy consumption, image pulling costs, and workload transition costs. Although the model achieved a 40–50% reduction in total costs compared to Docker Swarm’s Binpack strategy, it simplified workloads into single units rather than multi-dimensional resource vectors. In [21], an ant colony optimization-based scheduling algorithm was integrated into Docker Swarm to improve utilization in non-uniform resource environments. Similarly, [22] applied the Artificial Fish Swarm Algorithm (AFSA) to improve load balancing in Mesos, showing superior performance over Particle Swarm Optimization (PSO).

The work of [13] was among the first to apply a genetic algorithm to container scheduling, using NSGA-II to optimize provisioning, workload balancing, failure rates, and communication overhead. While effective, their approach excluded energy as an optimization objective. Later, [23] extended this line of work with NSGA-III, assigning batches of tasks to heterogeneous nodes under multiple objectives including energy consumption. However, their work neither used real power estimation models nor integrated into Kubernetes. Our

¹An extensible simulation framework that enables modeling, simulation, and experimentation of emerging Cloud computing infrastructures and application services, <https://github.com/Cloudslab/cloudsim>

Table I: Comparison of our work with existing approaches

Reference	Kubernetes	Power-aware scheduling	Power model	Load balancing	Resource efficiency	Optimization method
[11]	✓	✗	✗	✓	✓	Heuristic
[12]	✓	✗	✗	✓	✓	Heuristic
[13]	✓	✗	✗	✓	✓	NSGA-II
[14]	✓	✗	✗	✗	✓	TOPSIS
[15]	✓	✓	Linear	✗	✓	Sparrow Search
[16]	✓	✓	Linear	✗	✓	TOPSIS
Our work	✓	✓	Machine Learning Estimation	✓	✓	TOPSIS & NSGA-II

work differs by combining machine learning-based power prediction with online multi-objective optimization directly embedded in Kubernetes as a production-ready scheduler plugin.

A recent work [15] proposes an energy-aware Kubernetes scheduler that models power from CPU utilization and solves placement with a Sparrow-Search meta-heuristic while considering inter-service communication. In contrast, our scheduler profiles each node in-cluster and trains a machine-learning power predictor, then optimizes a richer multi-objective vector (power, CPU and memory load-balancing, pod balancing, fragmentation) via TOPSIS or NSGA-II within the native scheduler framework. The work of [24] proposes Micro-Ranker, which ranks microservices by inter-service communication and pre-distributes containers across geo-distributed data centers according to rank and a green-energy index, reporting lower energy and latency. Authors of [14] introduce NACS, a network-aware scheduler for edge environments that gathers latency, jitter, and packet-loss metrics and computes entropy-weighted TOPSIS rankings. Unlike NACS, our approach explicitly targets power efficiency through per-node ML-based power modeling and multi-objective optimization rather than relying solely on network performance metrics. Moreover, our scheduler integrates seamlessly with Kubernetes' native plugin framework and supports per-pod online decisions, enabling dynamic, scalable, and power-aware scheduling under varying workloads. Additionally, [25] presents a heterogeneous-resource scheduler that dynamically tunes per-resource weights and applies an enhanced D-TOPSIS across CPU, memory, GPU, network, and disk. Unlike these works, our approach explicitly models node power via learned per-node predictors and jointly optimizes power with load-balancing and fragmentation for online per-pod decisions.

Beyond container scheduling, there exists a line of research on distributed and parallel evolutionary optimization. The work of [26] proposed a cloud-native, multi-population architecture to accelerate metaheuristics by partitioning populations across worker nodes. [27] introduced an asynchronous multi-population metaheuristic framework with message-based migration to balance exploration and efficiency. Similarly, [28] studied Pareto-based optimization for service placement in the computing continuum, leveraging distributed evolutionary strategies to handle large-scale systems. While these works advance distributed evolutionary algorithms in general cloud environments, none of them integrate distributed NSGA-II into Kubernetes for online, pod-level scheduling. Our contribution is therefore novel in bridging distributed evolutionary optimization with practical Kubernetes scheduling through a real,

deployable plugin.

A. Multi-objective decision making and optimization algorithms

Multi-objective decision-making is a critical process for identifying optimal trade-offs across competing objectives in fields such as supply chain management, engineering, telecommunications, and economics [29], with foundational algorithms including the divide-and-conquer approach for Pareto-optimal solutions by [30] and TOPSIS for ranking alternatives by their geometric distance to an ideal solution, introduced by [19]. The enduring relevance and adaptability of TOPSIS are demonstrated by its recent and broad applications, including its combination with fuzzy AHP for green supplier selection in supply chain management [31], adaptive load balancing for fog-cloud resource allocation [32], energy-efficient scheduling for AIoT workloads [33], and priority-based scheduling in hybrid VANETs using an AHP-TOPSIS framework [34]. These diverse applications across dynamic and heterogeneous domains underscore TOPSIS's versatility and reinforce its suitability for integration into complex systems like our proposed power-aware Kubernetes scheduler.

III. PROBLEM FORMULATION

In this section, we formalize the multi-objective pod scheduling problem by presenting a mathematical model that incorporates key optimization goals, such as load balancing, resource utilization, and energy efficiency and provide an NP-hardness analysis to highlight the problem's computational complexity. We then introduce our proposed solution architecture, detailing its modular design and key components, followed by the optimization algorithms developed to yield near-optimal solutions for the defined objectives.

The goal of this work is to enhance Kubernetes scheduling by selecting the most suitable worker node $i \in \mathcal{N}$ to run a pod $j \in \mathcal{M}$ within a multi-objective configuration system, where \mathcal{N} represents the set of worker nodes and \mathcal{M} represents the set of pods. This approach aims to optimize resource load balancing, bin packing to reduce resource fragmentation, and power consumption. These objectives often conflict. For instance, prioritizing load balancing can inadvertently lead to resource wastage or increased power consumption. To address such trade-offs, we introduce a mathematical formulation of the problem as a constrained multi-objective optimization problem. The notations are summarized in Table II.

We define a binary variable $x_{ij} \in \{0, 1\}$ as follows:

$$x_{ij} = \begin{cases} 1, & \text{if pod } j \text{ is scheduled on node } i \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Table II: Summary of symbols and parameters (nodes indexed by i , pods by j).

Symbol	Description
\mathcal{N}	Set of nodes
\mathcal{M}	Set of pods
x_{ij}	Decision variable, 1 if pod j is assigned to node i , else 0
C_j^{cpu}, C_j^{mem}	Available CPU and Memory capacity of node i
d_j^{cpu}, d_j^{mem}	CPU and Memory demand of pod j
R_i^{cpu}	Allocated CPU on node i
R_i^{mem}	Allocated memory on node i
u_i	CPU utilization of node i (used in power model)
P_i	Power consumption of node i
P	Average cluster power
P_{idle}	Idle node power (W)
P_{max}	Max node power (W)
L_{cpu}, L_{mem}, L_p	Cluster load-balancing factors
f_i^{cpu}, f_i^{mem}	Per-node CPU and Memory fragmentation on i
F^{cpu}, F^{mem}	Average cluster CPU and Memory fragmentation
S	Population size for NSGA-II
G_{max}, t	Max generations, tournament size for NSGA-II
$\tilde{I}^{(j)}$	Relaxed probability vector for pod j
$\text{Dir}(\alpha)$	Dirichlet distribution used to sample $\tilde{I}^{(j)}$; α is its parameter
λ	Crossover mixing factor

Let C_j^{cpu} and C_j^{mem} denote the available CPU and memory on node i , respectively.

Let R_i^{cpu} be the allocated CPU on node i defined as:

$$R_i^{cpu} = \sum_{j=1}^M d_j^{cpu} \times x_{ij} \quad (2)$$

Where d_j^{cpu} denotes CPU demands of pod j .

Similarly we define R_i^{mem} to be the allocated memory on node i defined as:

$$R_i^{mem} = \sum_{j=1}^M d_j^{mem} \times x_{ij} \quad (3)$$

Where d_j^{mem} denotes memory demands of pod j .

Next, we define our scheduling objectives that include, resource load balancing, power consumption and resource utilization:

a) Load balancing objectives: Load balancing describes how well the pods are distributed across the worker nodes. An optimal load balancing factor minimizes performance bottlenecks by preventing any single worker node from being overloaded while avoiding under utilization of others.

The load balancing factor is formulated as a coefficient of variation [35] such that a lower value indicates a better distribution of the workload across the nodes of the cluster. In our setup, we explicitly target three distinct load balancing dimensions:

- 1) CPU utilization: Ensures that computational resources are evenly distributed across nodes.
- 2) Memory utilization: Focuses on balancing memory allocation to avoid overloading specific nodes while leaving others underutilized.
- 3) Pod count: Targets the even distribution of the number of pods across all nodes, regardless of resource utilization, to maintain fairness and avoid scheduling bottlenecks.

$$L^{cpu} = \frac{\sigma(R^{cpu})}{\mu(R^{cpu})}, \quad L^{mem} = \frac{\sigma(R^{mem})}{\mu(R^{mem})}, \quad L^{pod} = \frac{\sigma(R^{pod})}{\mu(R^{pod})} \quad (4)$$

where $\sigma(\cdot)$ is the standard deviation function and $\mu(\cdot)$ is the mean function. In addition, R^{cpu} , R^{mem} and R^{pod} are defined as follows:

- R^{cpu} denotes the set of CPU utilization across the cluster:

$$R^{cpu} = \{R_1^{cpu}, R_2^{cpu}, \dots, R_N^{cpu}\}$$

- R^{mem} denotes the set of memory utilization across the cluster:

$$R^{mem} = \{R_1^{mem}, R_2^{mem}, \dots, R_N^{mem}\}$$

- R^{pod} denotes the set of deployed pod counts across the cluster:

$$R^{pod} = \{R_1^{pod}, R_2^{pod}, \dots, R_N^{pod}\}$$

b) Resource fragmentation objectives: Resource fragmentation measures how effectively resources (CPU and memory) are allocated without leaving **unallocatable residuals**. In Kubernetes, pods request fixed amounts of CPU and memory; if the remaining capacity on a node is smaller than typical pod requests, this fragment of resources cannot be used and is effectively wasted. Thus, fragmentation is directly linked to **bin-packing inefficiency**: although aggregate utilization may appear low, scattered small gaps across nodes can still prevent new pods from being scheduled. By minimizing resource fragmentation, we aim to maximize resource utilization, making more space available for incoming pods and enhancing the overall efficiency of the cluster's resource distribution.

We denote by f_i^{cpu} and f_i^{mem} the CPU and memory fragmentation on node i respectively, and they are defined as follows:

$$f_i^{cpu} = \frac{C_i - R_i^{cpu}}{C_i}, \quad f_i^{mem} = \frac{M_i - R_i^{mem}}{M_i} \quad (5)$$

Here, f_i^{cpu} and f_i^{mem} represent the **fraction of capacity left unused** on node i . A high value indicates that significant resources remain idle but may not be practically usable for incoming pods, while a low value indicates tighter packing and better utilization.

Where:

- $R_i^{cpu} \leq C_i \forall i \in [1, N]$
- $R_i^{mem} \leq M_i \forall i \in [1, N]$

F^{cpu} and F^{mem} denote cluster-wise CPU and memory resource fragmentation and they're defined as the average values of nodes' resource fragmentation:

$$F^{cpu} = \frac{1}{N} \sum_{i=1}^N f_i^{cpu}, \quad F^{mem} = \frac{1}{N} \sum_{i=1}^N f_i^{mem} \quad (6)$$

Lower values of F^{cpu} and F^{mem} indicate that resources are more effectively packed and that the cluster is less likely to reject new pods due to unusable fragments. This ensures better scheduling success rates and improved overall efficiency, even under fluctuating and heterogeneous workloads.

c) *Power consumption objective*: Let P_i denote the power consumption of node i , with $P_i : [0, 1] \rightarrow \mathbb{R}$ representing a function that maps the node's CPU resource utilization to its power consumption. The power consumption function is estimated using the CPU utilization level of the node, as detailed IV-A.

Power consumption across the cluster is calculated as follows:

$$P = \frac{1}{N} \sum_{i=1}^N P_i$$

1) *Optimization objective*: We define our objective function \mathcal{F} as follows:

$$\mathcal{F} : \mathbb{R}^{N \times M} \rightarrow \mathbb{R}^{|\mathcal{O}|}$$

$$X \mapsto \begin{pmatrix} P \\ -L^{cpu} \\ -L^{mem} \\ -L^{pod} \\ F^{cpu} \\ F^{mem} \end{pmatrix}$$

Where:

- \mathcal{O} denotes the set of optimization objectives.
- X is the assignment matrix, where each element x_{ij} indicates the assignment of pod j to node i .

The optimization objective is the following:

$$\begin{aligned} &\text{minimize}_X \mathcal{F}(X) \\ &\text{subject to:} \\ &R_i^{cpu} \leq C_i^{cpu}, \quad \forall i \in \{1, 2, \dots, N\} \\ &R_i^{mem} \leq C_i^{mem}, \quad \forall i \in \{1, 2, \dots, N\} \end{aligned} \quad (7)$$

The scheduling problem in (7) thus seeks an assignment of pods to nodes that simultaneously:

- minimizes cluster-wide power consumption,
- improves load balancing across CPU, memory, and pod count (minimizing variance),
- and minimizes CPU and memory fragmentation to maximize bin-packing efficiency.

This formulation ensures that the scheduler balances energy efficiency, fairness, and resource utilization, while respecting the physical capacity constraints of each node.

Theorem 1. *The pod scheduling problem formulated in 7 is NP-hard.*

Proof. To prove the NP-hardness of the problem, we are going to prove that there's a polynomial time reduction from the Multi-dimensional Multi Knapsack Problem (MKP) that's known to be strongly NP-hard problem [36].

The MKP is the multi-dimensional extension of the Knapsack problem where we have multiple knapsacks (bags) instead of just one and each item has more than one dimension. The goal is to maximize the total value of items distributed across multiple knapsacks without exceeding the capacity of any of them across all dimensions.

The MKP problem can be formulated as follows:

$$\text{maximize} \quad \sum_{i=1}^n \sum_{j=1}^m v_j \cdot x_{ij} \quad (8)$$

$$\text{subject to:} \quad \sum_{j=1}^m w_j^{(k)} \cdot x_{ij} \leq C_i^{(k)}, \quad \forall i, \forall k \quad (9)$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \forall j \quad (10)$$

Where:

- n is the number of knapsacks.
- m is the number of items.
- x_{ij} is a binary variable indicating whether item j is placed in knapsack i ($x_{ij} = 1$) or not ($x_{ij} = 0$).
- v_j is the value of item j .
- $w_j^{(k)}$ is the weight of item j across the k -th dimension.
- $C_i^{(k)}$ is the capacity of knapsack i across the k -th dimension.

First, since $v_j > 0$, there must be a p -dimensional vector $u \in \mathbb{R}^p$ such that

$$v_j = ||u||$$

An example of such vector is $u = \begin{pmatrix} v_j \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

Hence, 8 is equivalent to:

$$\text{minimize} \quad \begin{pmatrix} -\sum_{i=1}^n \sum_{j=1}^m v_j \cdot x_{ij} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

subject to the same constraints.

Second, item j can be mapped to a pod j and knapsack i can be mapped to worker node i . The weight of item j across the k -th dimension can be mapped to k -th resource type demands of pod j .

This process of mapping can be done in polynomial time as finding the vector u can be done in polynomial and we have a one-to-one mapping of the constraints.

Therefore the MKP problem can be reduced in polynomial time to the pod scheduling problem which makes the pod scheduling problem an NP-hard problem. \square

IV. METHODOLOGY AND SYSTEM ARCHITECTURE

To solve the scheduling problem formulated in the previous section, we propose a novel Kubernetes system, as illustrated in Figure 1, which outlines the overall system architecture. The system is designed to address multiple objectives, including power consumption optimization, resource utilization efficiency, and load balancing. It can be decomposed into two main components:

- 1) **Power Consumption Estimation**: This component models the power consumption of Kubernetes worker

nodes using node power profiling and machine learning-based estimation. Accurate power modeling serves as a foundation for integrating energy efficiency as one of the optimization objectives.

- 2) **Scheduling Optimization:** This component leverages the power consumption model, along with metrics for resource utilization to optimize the online scheduling process. The scheduler aims to balance competing objectives, such as minimizing energy consumption, maximizing resource utilization, and ensuring even workload distribution.

In the following, we detail the implementation of both the power model estimation and the multi-objective optimization process for scheduling.

A. Power consumption estimation model

The goal of this step is to find the power model that accurately estimates the consumption of power with respect to resource utilization. Since different CPU models exhibit varying power consumption behaviors depending on their utilization, factors like hardware architecture, workload characteristics, and environmental conditions further complicate this relationship. As a result, no universal model exists to map resource utilization directly to power consumption. This complexity makes it challenging to estimate power consumption when scheduling a pod with specific resource requirements on a node. To overcome this, we employ machine learning-based estimation by collecting real-time power consumption data and training a machine learning regression model to predict power usage using CPU utilization as the sole input. Focusing on CPU usage provides a practical and effective method for power prediction, as CPU utilization is both readily available and continuously monitored in Kubernetes environments. While many established works affirm that CPU utilization is the most significant and readily measurable predictor of power consumption, with large-scale studies demonstrating a strong correlation between CPU load and system power draw in data center servers [37], [38], we acknowledge that other factors such as memory access patterns, disk I/O, and network activity can also contribute to total power usage, particularly in heterogeneous workloads. Our current model leverages CPU utilization as the primary feature because of its dominant impact and low overhead for runtime monitoring, which enables efficient, online power estimation [39], [40]. Nevertheless, the proposed framework is designed to be extensible: additional resource-level metrics can be incorporated in future iterations to further refine prediction accuracy in more complex deployment scenarios.

To collect the training data, we deploy *Kepler*² power monitoring tool in the cluster alongside *Prometheus*³. Kepler, continuously extracts power-related metrics from the nodes and stores them in Prometheus as time series data. As illustrated in Fig. 1, (1) whenever a new node joins the cluster

via a join request to the API server, a custom node power profiling controller is triggered (2) to configure and deploy *stress-ng*⁴ pod that stresses the CPU by consuming all the allocated CPU for a specific profiling period. The controller also adds some affinity rules to the pod configuration in order to make sure that this pod gets scheduled onto that specific node (3). Also it cordons the node to make it unschedulable to ensure that no additional workloads interfere with the power measurement, providing a controlled environment for accurate power profiling.

The configuration generated by the controller configures CPU utilization levels between 0% and 100%, with a fine-grained step of 0.5% at each CPU stressing cycle, the next cycle begins whenever the pod terminates. Each utilization level is maintained for one minute to ensure that Kepler power consumption extraction procedure finishes, hence, ensuring accurate and consistent power measurement. The cycle is repeated k number of times and averaged to reduce the variance of the dataset [41]. The chosen step size of 0.5% reflects the continuous nature of CPU usage and allows the model to capture variations in power consumption. Even slight changes in CPU utilization can lead to measurable differences in power consumption, particularly at higher utilization levels, making this granularity necessary for constructing a precise and representative dataset. Kepler continuously collects energy consumption-related metrics of the cluster (4). This data is being exported to Prometheus (5).

After collecting the data, the training procedure starts (6). We experimented with various regression models, including linear regression, random forests, support vector regressors, and neural networks. The experimental results (detailed in section V) demonstrated that *Random Forests* model [42] provides the most accurate power consumption estimates among the tested models. The superiority of this model is its ability to capture very complex patterns in data and also its high generalization capabilities even with limited datasets [43]. The trained model is saved into a model registry (7) that keeps track of the node and its power mode. (8) After a successful model training, the controller is notified to uncoron the node and make it schedulable again (9).

At the end of this stage, we have a model that is able to estimate the power consumption of each node which plays a crucial during the scheduling cycle as it enables building a power-aware scheduler.

B. Scheduling step

In this second step, arriving pods are dynamically assigned to worker nodes using Filter-Score pattern. We developed a new scoring plugin, *PowerAware*, which employs multi-objective optimization to select the most suitable worker node for each pod. The plugin optimizes three competing objectives: resource load balancing (workload efficiency), minimizing resource fragmentation, and reducing power consumption. These goals often conflict; for example, prioritizing load balancing may increase resource wastage or lead to higher power consumption. To address this, the scoring mechanism

²Kepler (Kubernetes-based Efficient Power Level Exporter) is a Prometheus exporter. It uses eBPF to probe CPU performance counters and Linux kernel tracepoints, <https://sustainable-computing.io/>

³Prometheus is a monitoring system & a time series database, <https://prometheus.io/>

⁴CPU stressing tool, <https://github.com/ColinIanKing/stress-ng>

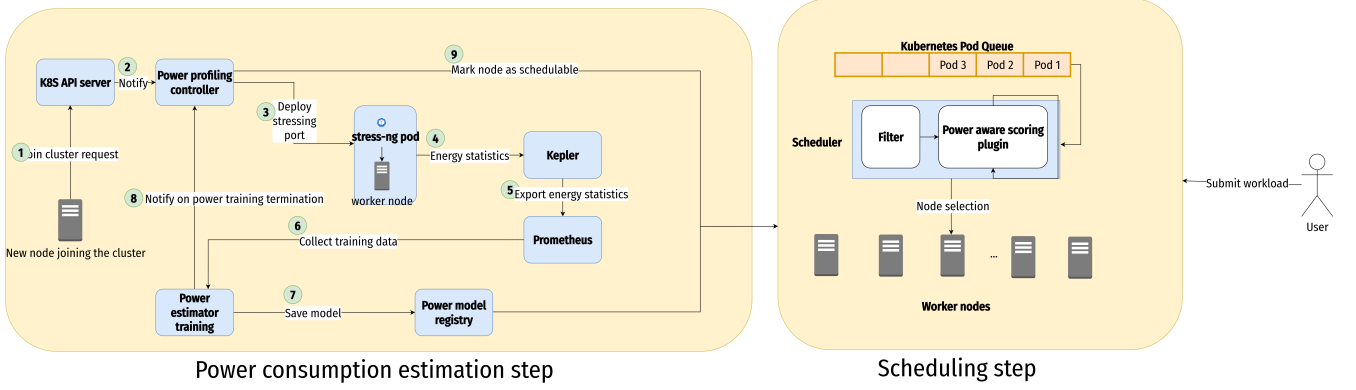


Figure 1: Kubernetes power-aware scheduling framework workflow

balances these objectives using predefined weights, ensuring adaptability to varying system requirements. For this the plugin two optimizers:

- 1) Heuristic optimizer based on TOPSIS algorithm: The algorithm transforms the multi-objective problem into a single-objective optimization using a weighted scalarization technique, ranking nodes based on their proximity to the ideal solution.
- 2) Meta-heuristic-based optimizer (NSGA-II): Which directly tackles the multi-objective problem, employing a genetic algorithm enhanced with custom crossover and mutation operators to efficiently explore and identify Pareto-optimal solutions (solutions where improving one objective would compromise another).

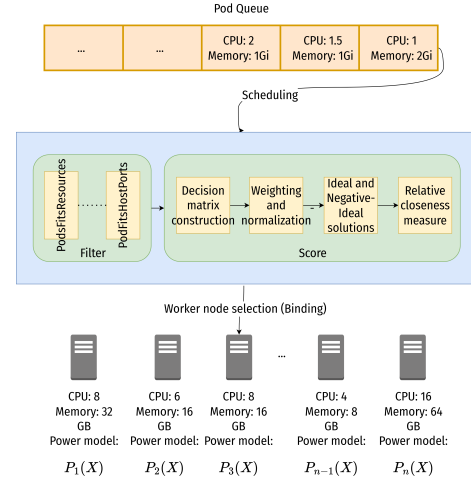


Figure 2: Kubernetes power-aware scheduling workflow with TOPSIS

C. Scheduling optimization using TOPSIS

TOPSIS-based scheduling, as depicted in Fig. 2, operates by evaluating worker nodes based on their CPU and memory usage, as well as their power model, P_i estimated using the procedure described in IV-A. Once a pod is submitted to the queue, the scheduler runs a filtering phase that executes a set of predicates, such as *PodFitsResources* to make sure that the pod can fit the resources available in that node.

The filtering phase outputs a set of candidate nodes \mathcal{N} that enter the scoring phase that executes TOPSIS workflow. Given the set of objectives and a set of preferences or weights for each objective, the framework identifies the optimal node that is closest to the ideal solution which represents a theoretical optimal solution where all objectives achieve their best possible values and furthest from the worst solution, opposite to the ideal solution, where all criteria are at their least desirable values. The weights reflect the relative importance of each objective in the optimization process. It is important to note that the algorithmic steps used in this section follow the classical TOPSIS formulation introduced by [19]. In our work, these formulas are not newly invented but are adapted to the scheduling objective.

a) *Decision matrix construction*: The algorithm begins by constructing a decision matrix whenever a new pod comes, where rows represent the alternative solutions (i.e., the worker nodes), and columns represent the objectives to optimize. Each objective is classified as either a benefit objective, which should be maximized, or a cost objective, which should be minimized. We denote the set of benefit criteria by \mathcal{B} and the set of cost criteria by \mathcal{C} . Formally, the set of objectives \mathcal{O} is defined as: $\mathcal{O} = \mathcal{B} \cup \mathcal{C}$.

Let A be the decision matrix where $a_{i,j}$ represents the value of alternative (i.e., work node) $i \in \mathcal{N}$ for objective $j \in \mathcal{O}$:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1p} \\ a_{21} & a_{22} & \dots & a_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{np} \end{pmatrix}$$

, where

- n : The number of alternatives, which represents the number of worker nodes available for scheduling.
- p : The number of objectives.
- \mathcal{O} : The set of objectives as formulated in III.

b) *Normalization*: The next step is to normalize the columns of the matrix A to make the objectives comparable. The normalized decision matrix R is obtained by:

$$r_{ij} = \frac{a_{i,j}}{\sqrt{\sum_{i=1}^p a_{i,j}^2}}$$

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

c) *Weighted Normalized Decision Matrix*: Each objective 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:

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

d) *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 S^+** : This solution consists of the best values for each objective. For benefit objectives (where higher values are preferable), it takes the maximum value across all alternatives. For cost objectives (where lower values are better), it selects the minimum value:

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

where v_{ij} is the value of the j -th objective for the i -th alternative. This represents an optimal state for all objectives.

- **Negative-Ideal Solution S^-** : This solution is composed of the worst values for each criterion. For benefit objectives, it takes the minimum value across all alternatives, while for cost objectives, it selects the maximum value:

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

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

e) *Distance from the Ideal and Negative-Ideal Solutions*:

For each alternative 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^-** :

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

f) *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.

Algorithm 1 Power-aware scheduling with TOPSIS Optimization

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

foreach $pod \in Q$ **do**

```

    // Create decision matrix for the
    // current pod  $p$  across all candidate
    // nodes
     $A \leftarrow \text{CREATEDECISIONMATRIX}(pod, \mathcal{N})$ 
    // Normalize the decision matrix
     $R \leftarrow \text{NORMALIZE}(A)$ 
    // Apply weights to the normalized
    // matrix
     $V \leftarrow \text{APPLYWEIGHTS}(R, w)$ 
    // Calculate ideal and negative-ideal
    // solutions
     $S^+ \leftarrow \text{IDEALSOLUTION}(V)$ 
     $S^- \leftarrow \text{NEGATIVEIDEALSOLUTION}(V)$ 
    // Compute separation from ideal and
    // negative-ideal solutions
     $D^+ \leftarrow \text{CALCULATEDISTANCE}(V, S^+)$ 
     $D^- \leftarrow \text{CALCULATEDISTANCE}(V, S^-)$ 
    // Determine relative closeness to the
    // ideal solutions
     $C^* \leftarrow \text{CLOSENESS}(D^+, D^-)$ 
    // Find node with highest relative
    // closeness
     $x^* \leftarrow \text{ARGMAX}(C^*)$ 
    // Schedule pod  $p$  on the node with
    // highest relative closeness  $x^*$ 
     $\text{SCHEDULEPOD}(pod, x^*)$ 

```

D. Scheduling optimization using multi-objective genetic algorithms

This section explains how a multi-objective genetic algorithm (NSGA-II) optimizes online scheduling. We first present the solution encoding and custom variation operators, then describe their integration into NSGA-II.

A multi-objective genetic algorithm maintains a *population* of candidate solutions and iteratively improves them across generations via three main operators: *selection*, *crossover*, and *mutation*.

a) *Encoding*.: A solution (individual) for pod j is a vector $I^{(j)} \in \{0, 1\}^{|\mathcal{N}|}$ where $x_{ij} = 1$ if pod j is assigned to node i and 0 otherwise; thus $\sum_{i=1}^{|\mathcal{N}|} x_{ij} = 1$:

$$I^{(j)} = [x_{1j}, x_{2j}, \dots, x_{|\mathcal{N}|j}].$$

To simplify crossover and mutation while preserving feasibility, we relax $I^{(j)}$ to a probability vector $I^{(j)} \in \Delta^{|\mathcal{N}|-1}$ on the simplex $\Delta^{|\mathcal{N}|-1} = \{\mathbf{z} \in \mathbb{R}_{\geq 0}^{|\mathcal{N}|} : \sum_{i=1}^{|\mathcal{N}|} z_i = 1\}$, where \tilde{x}_{ij} denotes the probability of assigning pod j to node i .

b) *Population generation (sampling strategy)*.: Because individuals live on the simplex, we initialize the population P_0 by sampling from a Dirichlet distribution:

$$P_0 = \{\tilde{I}_1^{(j)}, \tilde{I}_2^{(j)}, \dots, \tilde{I}_S^{(j)}\}, \quad \tilde{I}_s^{(j)} \sim \text{Dir}(\boldsymbol{\alpha}), \quad s \in \{1, \dots, S\},$$

where S is the population size and $\text{Dir}(\boldsymbol{\alpha})$ is the Dirichlet distribution with parameter $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_K)$, $\alpha_i > 0$. Sampling from $\text{Dir}(\boldsymbol{\alpha})$ guarantees *feasible* individuals at initialization (non-negativity and unit-sum) without projection or repair. Moreover, $\boldsymbol{\alpha}$ is a hyperparameter used to steer diversity and sparsity: $\alpha_i = 1$ yields uniform exploration; $\alpha_i < 1$ encourages sparse (few-node) allocations; $\alpha_i > 1$ favors well-spread allocations.

After evaluation, we perform the reverse relaxation operation as given by Eq. 11 to get the index k at which the entry is 1. The index k is used as a row index of the decision matrix to get the value of the solution. Algorithm 2 summarizes this step.

$$x_{kj} = 1 \text{ where } k = \arg \max(\tilde{I}^{(j)}), \text{ and } x_{ij} = 0 \quad \forall i \neq k \quad (11)$$

c) *Selection strategy*: For the selection strategy, we use *Tournament Selection*, where a random subset of individuals is selected, and the individual with the highest fitness from that subset is chosen as a parent. This process is repeated to select pairs of parents for the next generation. The tournament size t is typically set to 2 or 3. The procedure is as follows:

- Randomly select t individuals from the population.
- The individual with the highest fitness among these t individuals is selected as a parent.
- Repeat the process for the second parent.

The motivation behind choosing this method is that it ensures diversity in the population while maintaining a focus on high-fitness individuals for reproduction.

d) *Crossover strategy*: For crossover, our strategy involves generating offspring from two parents by randomly averaging the probabilities of the two parents.

Let the two parents be represented as:

$$p_k = [\tilde{x}_{1j}, \tilde{x}_{2j}, \dots, \tilde{x}_{|\mathcal{N}|j}], \quad p_{k'} = [\tilde{x}'_{1j}, \tilde{x}'_{2j}, \dots, \tilde{x}'_{|\mathcal{N}|j}]$$

For each gene i , the offspring c is generated as convex combination of the genes of its parents, as follows:

$$c_{kk'} = \begin{bmatrix} \lambda \tilde{x}_{1j} + (1 - \lambda) \tilde{x}'_{1j} \\ \lambda \tilde{x}_{2j} + (1 - \lambda) \tilde{x}'_{2j} \\ \vdots \\ \lambda \tilde{x}_{|\mathcal{N}|j} + (1 - \lambda) \tilde{x}'_{|\mathcal{N}|j} \end{bmatrix} \quad (12)$$

Where:

- $c_{kk'}$ is the resulting child after crossover operation between parents p_k and $p_{k'}$.
- λ is a mixing factor that represents the contribution of each parents in child's genes. The operation is illustrated in Fig. 3.

Because each individual is a probability vector on the simplex, taking a convex combination of two parents with a mixing factor λ keeps the offspring feasible (non-negative entries that sum to one) without any repair. It is also lightweight and tunable: sampling $\lambda \in [0, 1]$ balances exploration and exploitation while avoiding extra normalization steps, which is important for online scheduling latency.

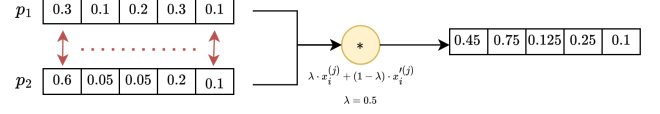


Figure 3: Crossover operation

e) *Mutation strategy*: The chosen mutation strategy is a swap mutation, which modifies individuals by randomly swapping two elements in their probability vectors.

For each individual $\tilde{I}^{(j)}$ in the population, two indices, i_1 and i_2 , are randomly chosen from the probability vector, and the values at these indices are swapped:

$$\tilde{I}^{(j)}[i_1] \leftrightarrow \tilde{I}^{(j)}[i_2]$$

Fig 4. depicts a visual diagram for the proposed mutation operation.

The proposed mutation maintains population diversity by altering the order of genes in each individual's probability vector, without changing its overall distribution. Although the resulting population may not immediately yield high-quality solutions, iterating this process ultimately guides the population toward the Pareto front.

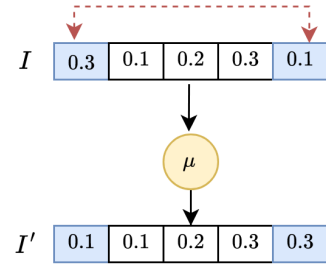


Figure 4: Mutation operation

f) *Fitness function*: The fitness function describes how good a solution is. For a solution $\tilde{I}^{(j)}$, we perform the reverse relaxation operation to obtain $I^{(j)}$ as given by Eq. 11.

Algorithm 2 Fitness function

Input: Decision matrix of pod j , $X^{(j)}$, Individual $\tilde{I}^{(j)}$

Output: Objective value v

$k \leftarrow \arg \max(\tilde{I}^{(j)})$

$v \leftarrow X^{(j)}[k, :]$

return v

g) *Non-Dominated Sorting Genetic Algorithm II*:

NSGA-II is a widely used genetic algorithm for multi-objective optimization, leveraging non-dominated sorting, crowding distance, and elitism to guide candidate solutions toward an optimal Pareto front.

Algorithm 3 NSGA-II-based scheduling

Input: Set of nodes \mathcal{N} , Set of pods \mathcal{M} , Population size S , Maximum number of generations G_{\max}

```

for  $p \in \mathcal{M}$  do
   $A^{(p)} \leftarrow \text{CREATEDECISIONMATRIX}(p, \mathcal{N})$ 
   $P_0 \leftarrow \text{INITIALIZEPOPULATION}(S)$  ①
   $\text{EVALUATEPOPULATION}(P_0, A^{(p)})$  ②
   $t \leftarrow 1$ 
  while  $t \leq G_{\max}$  do
     $F_t \leftarrow \text{NONDOMINATEDSORTING}(P_t)$  ③
     $D_t \leftarrow \text{CROWDINGDISTANCE}(F_t)$  ④
     $P_{\text{parents},t} \leftarrow \text{PARENTSELECTION}(F_t, D_t)$  ⑤
     $C_t \leftarrow \text{CROSSOVERANDMUTATION}(P_{\text{parents},t})$  ⑥
     $C_t \leftarrow \text{EVALUATEPOPULATION}(C_t, A^{(p)})$  ⑦
     $R_t \leftarrow P_t \cup C_t$  ⑧
     $B_t \leftarrow \text{NONDOMINATEDSORTING}(R_t)$ 
     $P_{t+1} \leftarrow \text{SELECTNEXTGENERATION}(B_t, N)$ 
     $t \leftarrow t + 1$ 
  // Select node from Pareto-front
   $x^* \leftarrow \text{SELECT}(P_{t+1})$ 
   $\text{SCHEDULEPOD}(p, x^*)$ 

```

Non-dominated sorting groups solutions into groups called fronts, each front contains those solutions that are dominated only by the previous front. The crowding distance is used to maintain diversity within the population, as less crowded regions are more preferred over crowded regions. NSGA-II is employed to optimize the objective function \mathcal{F} in an online manner as the pods arrive. The NSGA-II algorithm described in Alg. 3 operates through a series of steps to reach the optimal Pareto front solutions in our multi-objective optimization.

First, the algorithm begins by generating an initial population ① which involves randomly assigning probabilities for scheduling pods to nodes using a Dirichlet distribution. Each individual represents a probability distribution where entries signify the likelihood of each pod being assigned to a specific node. Next, the population is evaluated ②, in which the objective values for each individual such as power consumption, resource fragmentation, and load balancing using the decision matrix $A^{(p)}$ that contains the value of each possible assignment. Then, non-dominated sorting step ③ ranks the population P_t based on Pareto dominance, dividing it into different non-dominated fronts. This ensures that individuals with the lowest number of dominations are ranked higher. To maintain diversity within these fronts, the crowding distance of the resulting population F_t ④ is calculated, which assigns a crowding distance value to each solution, favoring those located in less crowded regions of the solution space. Once the population has been ranked and distances computed, parent selection using the binary tournament selection method is invoked to select parents for crossover, prioritizing individuals based on their dominance and crowding distance ⑤. The selected parents undergo crossover and mutation using our custom strategies ⑥. These offspring are evaluated ⑦, and the parent and offspring populations are combined using ⑧.

The combined population is sorted again, and finally, the top individuals are selected based on their Pareto rank and crowding distance ⑨, ensuring the best solutions survive to the next generation. This process repeats until the maximum number of generations is reached, resulting in a set of Pareto-optimal solutions, representing balanced trade-offs between the competing objectives. We select a node x^* from the Pareto-front that was found and schedule the pod p to it.

h) Exploration–Exploitation and Selection Pressure: In the proposed NSGA-II-based optimizer, exploration is promoted through random initialization of the population using a Dirichlet distribution and the stochastic nature of crossover and mutation operators, which generate diverse candidate assignments. Exploitation is achieved through selection pressure, enforced by binary tournament selection, non-dominated sorting, and crowding distance ranking, which iteratively guide the population toward the Pareto front. This moderate selection pressure ensures that high-quality solutions are favored while maintaining diversity to prevent premature convergence. In contrast, the TOPSIS-based optimizer is primarily deterministic and focuses on exploitation, selecting the node with the highest proximity to the ideal solution based on the weighted objective matrix. While this ensures consistent convergence toward a single optimal solution under the given weights, it limits exploration of alternative trade-offs between objectives.

i) Time Complexity Analysis: We analyze worst-case time complexity per scheduling decision (i.e., per pod). Let \mathcal{N} be the set of nodes and let $\mathcal{N}_{\text{cand}} \subseteq \mathcal{N}$ be the subset remaining after the filtering step. Let \mathcal{O} be the set of objectives and \mathcal{S} the population set. Constructing the decision matrix over $\mathcal{N}_{\text{cand}}$, column-wise normalization, weighting, computing ideal/negative-ideal points, and evaluating Euclidean distances and relative closeness each take $\mathcal{O}(|\mathcal{N}_{\text{cand}}| |\mathcal{O}|)$, yielding an overall $\mathcal{O}(|\mathcal{N}_{\text{cand}}| |\mathcal{O}|)$.

For the *NSGA-II-based scheduler*, each generation performs fast non-dominated sorting in $\mathcal{O}(|\mathcal{O}| |\mathcal{S}|^2)$, crowding-distance assignment in $\mathcal{O}(|\mathcal{O}| |\mathcal{S}| \log |\mathcal{S}|)$, and selection, crossover, mutation in $\mathcal{O}(|\mathcal{S}|)$. Thus the total per-pod complexity is $\mathcal{O}(G |\mathcal{O}| |\mathcal{S}|^2)$.

Consequently, the TOPSIS-based scheduler is linear in both $|\mathcal{N}_{\text{cand}}|$ and $|\mathcal{O}|$, making it suitable for real-time placement, whereas NSGA-II is quadratic in $|\mathcal{S}|$ and aims for higher-quality Pareto trade-offs and benefits from parallelism for large $|\mathcal{S}|$.

V. EXPERIMENTATION

In this section, we are going to describe the experimentation setup and results for our proposed work. We first present the evaluation results of the power estimation machine learning model. Then, we present the results of the proposed multi-objective scheduling algorithms and compare them to the default scheduler under different scenarios.

A. Power model experimentation

We collect a training dataset of a Kubernetes node using the approach discussed in section IV-A. The node is equipped with 4 vCPUs Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz

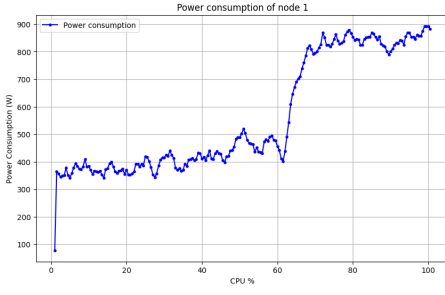


Figure 5: Node 1 power consumption

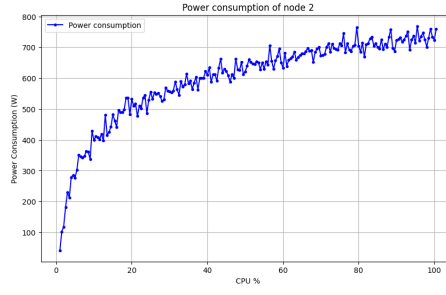


Figure 6: Node 2 power consumption

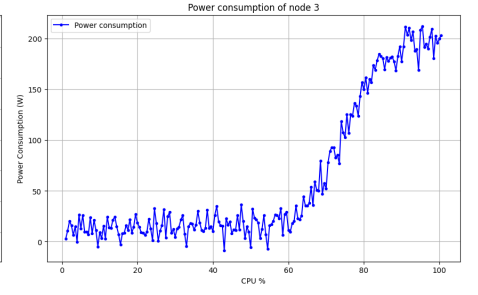


Figure 7: Node 3 power consumption

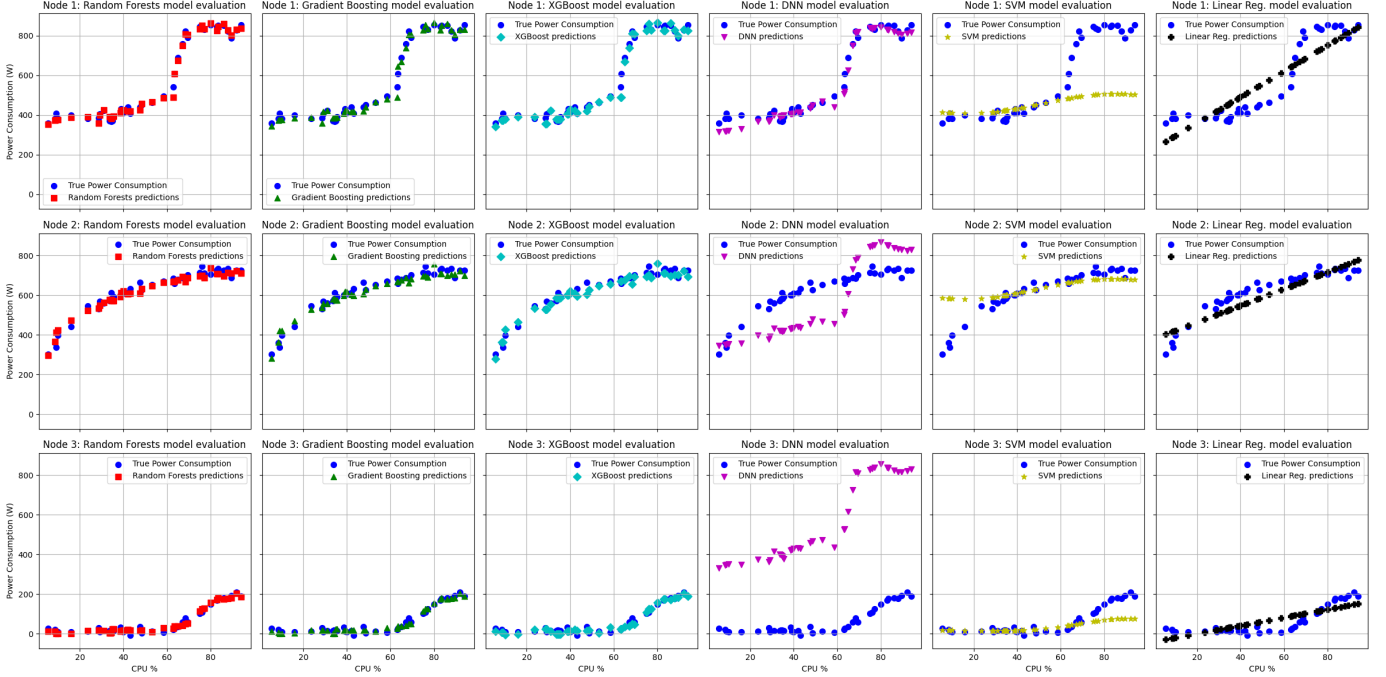


Figure 8: Model comparison

and 8GB of RAM. Figure 5 depicts power consumption of this node with respect to CPU utilization level. To further test robustness of the machine learning model that we developed, we create two synthetic power consumption datasets based on logarithmic [44] and exponential [45] power consumption models with some added noise which are presented in figures 6 and 7.

The training dataset for each node consisted of approximately 200 samples, generated by varying CPU utilization from 0% to 100% in steps of 0.5%. The dataset was split using an 80/20 ratio for training and testing, respectively. The hyperparameters for the evaluated machine learning models were as follows:

- Random Forest: 100 estimators, no maximum depth limit, and a fixed random state for reproducibility.
- Gradient Boosting & XGBoost: 100 estimators with a learning rate of 0.1.
- Neural Network (DNN): A multi-layer perceptron with two hidden layers (64 and 32 neurons, ReLU activation), trained for 100 epochs using the Adam optimizer.
- Support Vector Regressor (SVR): Default Radial Basis Function (RBF) kernel.

- Linear Regression: No specific hyperparameters.

We experimented with various machine learning models and evaluated them using the Mean Squared Error (MSE), as it is well-suited for regression problems. The MSE measures the average squared difference between predicted and actual values, with lower MSE values indicating better performance of the model. Table III summarizes the MSE results for various machine learning models across all nodes. The results clearly show that Random Forests outperformed all other models, achieving the lowest MSE values for all nodes.

Figure 8 provides a visualization of the prediction results of the various models on the test dataset for each node. It can be seen that tree-based ensemble models, such as Random Forests and Gradient-boosting methods perfectly fit the testing dataset with superior performance of the Random Forests model.

These results highlight the strength of Random Forests in capturing complex patterns across all nodes, owing to its ensemble structure, which combines multiple decision trees for robust predictions. Meanwhile, models like Linear Regression and Neural Networks struggled, likely due to their limitations in handling non-linearities and the high data demands of Neural Networks. Therefore, Random Forests model will be

Table III: Mean Squared Error (MSE) of models across nodes.

	Random Forest	Gradient Boosting	XGB	Neural Nets (DNN)	SVR	Linear Reg.
Node 1	32.499	72.205	85.111	174.553	9146.598	13 979.716
Node 2	538.690	647.410	555.771	821.216	6916.651	2548.945
Node 3	152.927	154.189	186.683	1093.259	2587.034	1449.659

Table IV: Node specifications and assigned power models.

Node	Tier	Power Efficiency	vCPUs	RAM (GB)	P_{idle} (W)	P_{max} (W)	Power Model
Node 1	High-Tier	Efficient	32	128	50	200	Exponential: $P(u) = P_{idle} + (P_{max} - P_{idle})u^2$
Node 2	High-Tier	Inefficient	32	128	70	250	Linear: $P(u) = P_{idle} + (P_{max} - P_{idle})u$
Node 3	Mid-Tier	Efficient	16	32	40	150	Exponential: $P(u) = P_{idle} + (P_{max} - P_{idle})u^2$
Node 4	Mid-Tier	Efficient	16	32	45	160	Exponential: $P(u) = P_{idle} + (P_{max} - P_{idle})u^2$
Node 5	Mid-Tier	Inefficient	16	32	50	170	Linear: $P(u) = P_{idle} + (P_{max} - P_{idle})u$
Node 6	Mid-Tier	Inefficient	16	32	55	180	Linear: $P(u) = P_{idle} + (P_{max} - P_{idle})u$
Node 7	Low-Tier	Efficient	4	16	20	80	Logistic: $P(u) = P_{idle} + (P_{max} - P_{idle})(1 + e^{-u})^{-1}$
Node 8	Low-Tier	Efficient	4	16	25	90	Logistic: $P(u) = P_{idle} + (P_{max} - P_{idle})(1 + e^{-u})^{-1}$
Node 9	Low-Tier	Inefficient	4	16	30	100	Linear: $P(u) = P_{idle} + (P_{max} - P_{idle})u$
Node 10	Low-Tier	Inefficient	4	16	35	110	Linear: $P(u) = P_{idle} + (P_{max} - P_{idle})u$

used to estimate power consumption given a utilization level in our proposed power-aware scheduler.

a) *Discussion:* Across all nodes, Random Forests emerged as the most accurate model, consistently achieving the lowest MSE. Ensemble methods, including Random Forests, Gradient Boosting, and XGBoost, demonstrated strong generalization capabilities, making them well-suited for this type of regression task. Neural Networks showed competitive performance, effectively modeling complex patterns in the data, although their accuracy was slightly lower than that of the ensemble methods. The results suggest that Neural Networks could benefit from additional hyperparameter tuning or a larger dataset to fully realize their potential. In contrast, Support Vector Machines and Linear Regression were less effective, as they are less suited for tasks with limited data and the latter can't handle non linear dependencies.

These findings prove the importance of selecting models that balance accuracy, complexity, and adaptability to the dataset's characteristics. Random Forests and other ensemble methods stand out as highly suitable choices for power estimation tasks in heterogeneous environments, while Neural Networks offer a promising alternative, particularly in scenarios where their strengths can be maximized.

B. Scheduler experimentation setup

Our scheduler is presented as a Kubernetes plugin. The plugin has been developed with Go language⁵ using the official *scheduler-plugins*⁶ project.

Table IV outlines the specifications of the Kubernetes cluster nodes used in our experimentation. The cluster is composed of 10 nodes, categorized into three tiers: high-tier, mid-tier, and low-tier, reflecting a diverse range of computational capacities and power efficiency. This diversity is introduced to emulate realistic cloud environments, where clusters often consist of heterogeneous nodes with varying resource capabilities and energy efficiency. Below, we explain the rationale behind the selection of node specifications:

a) Tier categorization :

- High-Tier Nodes: These nodes represent high-performance machines commonly found in production environments for handling compute-intensive tasks. One node is configured as power-efficient, reflecting modern energy-optimized hardware, while the other is less efficient to simulate legacy systems. These nodes have 32 vCPUs and 128 GB of RAM, ensuring sufficient computational power for demanding workloads.
- Mid-Tier Nodes: These nodes represent the majority of machines in a typical cluster, balancing computational power and energy consumption. The configurations (16 vCPUs, 32 GB of RAM) represent a typical middle ground between high-performance and low-resource nodes. To capture variability, two nodes are power-efficient, while two are not.
- Low-Tier Nodes: These nodes represent edge devices or older hardware with limited resources (4 vCPUs, 16 GB of RAM). They are included to study how scheduling algorithms handle constrained nodes and to evaluate energy efficiency in resource-scarce environments.

b) *Power model:* The cluster includes nodes classified as either *efficient* or *inefficient*, within each tier, reflecting differences in hardware architectures and power management technologies. Efficient nodes represent modern systems equipped with enhanced power-saving features, while inefficient nodes simulate legacy hardware that consumes more energy for similar workloads. The power consumption values for each node are generated using the machine learning model trained after profiling procedure described earlier in section IV-A.

c) *Diversity to mitigate bias:* The heterogeneity of the cluster is designed to capture any potential biases in the scheduling algorithms and to ensure a fair comparison among schedulers. By including nodes with varying resource capabilities and power efficiency levels, we evaluate how well each scheduler adapts to a wide range of conditions. This setup provides a robust platform to assess performance, fairness, and adaptability across diverse scheduling scenarios. We set up three experimental scenarios. In the first scenario, we submit a small workload of 50 pods, each with varying computational

⁵<https://go.dev/>

⁶<https://github.com/kubernetes-sigs/scheduler-plugins>

requirements, to establish a baseline for efficiency comparison. In the second scenario, we increase the workload up to 100 pods. Finally, we submit a heavy workload of 500 pods with varied requirements to assess the system's robustness under higher demand. Additionally, we model the stochastic nature of pod arrivals in real systems as a Poisson process with a constant rate $\lambda = 10$ of 10 pods per time unit.

To enable a custom scheduler in Kubernetes, we 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 1.

```

1 kind: KubeSchedulerConfiguration
2 apiVersion: kubescheduler.config.k8s.io/v1
3 clientConnection:
4   kubeconfig: kubeconfig.yaml
5 profiles:
6   - schedulerName: power-aware-scheduler
7     plugins:
8       multiPoint:
9         enabled:
10          - name: PowerAware
11          pluginConfig:
12            - name: PowerAware
13              args:
14                kind: PowerAwareArgs
15                apiVersion: kubescheduler.config.k8s.io/v1
16                optimizer: Topsis
17                powerCriteria:
18                  weight: 0.5
19                  type: Cost
20                cpuLBCriteria:
21                  weight: 0.1
22                  type: Cost
23                memLBCriteria:
24                  weight: 0.1
25                  type: Cost
26                podLBCriteria:
27                  weight: 0.1
28                  type: Cost
29                cpuFragmentationCriteria:
30                  weight: 0.1
31                  type: Cost
32                memFragmentationCriteria:
33                  weight: 0.1
34                  type: Cost

```

Listing 1: Kubernetes Scheduler Configuration for TOPSIS-based Power-Aware Scheduling

The configuration profile presented in listing 1 activates the custom *PowerAware* plugin and defines the criteria used for scheduling decisions. Each criterion maps to an objective and comprises two attributes: *weight* and *type*. The *weight* attribute, employed during the weighting phase of the TOPSIS decision matrix and determines the relative importance of each criterion, with power consumption assigned a high priority in this testing scenario. Meanwhile, the *type* attribute specifies whether a criterion is a *Cost*, which the scheduler aims to minimize, or a *Benefit*, which the scheduler seeks to maximize.

```

1 kind: KubeSchedulerConfiguration
2 apiVersion: kubescheduler.config.k8s.io/v1
3 clientConnection:
4   kubeconfig: kubeconfig.yaml
5 profiles:
6   - schedulerName: power-aware-scheduler
7     plugins:
8       multiPoint:
9         enabled:
10          - name: PowerAware
11          pluginConfig:
12            - name: PowerAware
13              args:
14                kind: PowerAwareArgs
15                apiVersion: kubescheduler.config.k8s.io/v1
16                optimizer: Nsga2
17                populationSize: 100
18                numGenerations: 20
19                powerCriteria:
20                  type: Cost
21                cpuLBCriteria:
22                  type: Cost
23                memLBCriteria:
24                  type: Cost
25                podLBCriteria:
26                  type: Cost
27                cpuFragmentationCriteria:
28                  type: Cost
29                memFragmentationCriteria:
30                  type: Cost

```

Listing 2: Kubernetes Scheduler Configuration for NSGA-II-based Power-Aware Scheduling

Indeed, the criteria defined for our plugin map to the dimensions of our objective function defined in 7:

- *powerCriteria*: A cost criterion, is assigned the highest weight, indicating the priority to minimize power consumption during scheduling.
- *cpuLBCriteria*: It is a cost criterion that encourages the scheduler to select nodes so that to maximize CPU load balancing (as load balancing factor is formulated as coefficient of variation, so a lower value indicates better load balancing).
- *memLBCriteria*: Similar to *cpuLBCriteria* but for memory.
- *podLBCriteria*: Similar to *cpuLBCriteria* and *memLBCriteria* but for pod counts as it encourages the scheduler to evenly distribute the pods across the cluster in terms of number of pods per nodes.
- *cpuFragmentationCriteria* and *memFragmentationCriteria* are defined as cost criteria to promote bin packing, with the goal of optimizing resource utilization by reducing resource fragmentation.

The optimizer used during the scheduling phase can be configured by setting the *optimizer* field in the scheduler configuration. Selecting *Topsis* enables TOPSIS-based optimization, prioritizing solutions based on their proximity to ideal and nadir solutions. Alternatively, setting the field to *Nsga2* activates NSGA-II-based optimization, as shown in listing 2 that sets the population size and the number of generation of the genetic algorithm.

d) *Scheduling Algorithm Hyperparameters*: The proposed schedulers were configured with the following parameters:

- TOPSIS-Based scheduler: The weight vector for the objectives was set to prioritize energy efficiency: $w = [0.5, 0.1, 0.1, 0.1, 0.1, 0.1]$ for

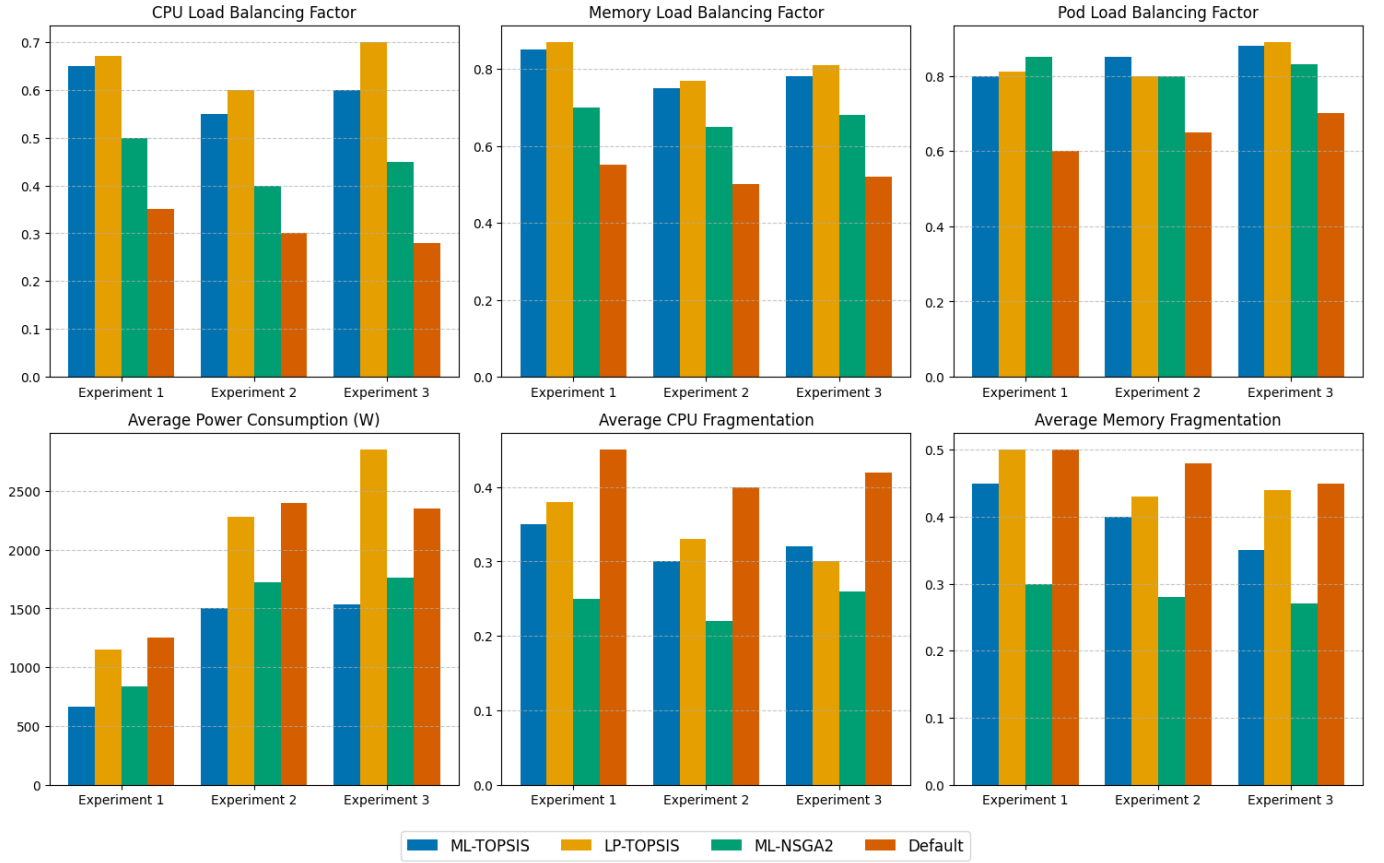


Figure 9: Performance comparison between different schedulers

$[P, L^{cpu}, L^{mem}, L^{pod}, F^{cpu}, F^{mem}]$ respectively. These weights are policy parameters; operators can adjust w to reflect cost, fairness, or performance priorities without changing the scheduler implementation.

- NSGA-II-based scheduler: The parameters were selected via a small grid search to balance Pareto-set quality and online per-pod scheduling latency. We swept population size $\{64, 100, 128\}$, generations $\{10, 15, 20, 30\}$, tournament size $\{2, 3, 4\}$, crossover probability $\{0.8, 1.0\}$, and mutation probability $\{0.05, 0.1, 0.2\}$, and chose the configuration that minimizes inverted generational distance. The final setting is: population size $S = 100$, number of generations $G_{\max} = 20$, tournament size = 2, crossover probability = 1.0, mutation probability = 0.1. The mixing factor λ used in Eq. 12 is drawn uniformly at random $\lambda \sim \mathcal{U}(0, 1)$ and α parameter of the Dirichlet distribution is set to 1.

C. Scheduler experimentation results

This section presents an in-depth analysis of the comparative performance of our proposed ML-TOPSIS-based and ML-NSGA-II-based scheduler algorithms that use a node-specific power estimation model using machine learning against baseline schedulers such as the default scheduler and the scheduler proposed in [16] that incorporates a linear power model for the worker nodes based on CPU utilization level that we will name the LP-TOPSIS scheduler (LP for linear power), as

shown in Fig. 9. The analysis covers the primary metrics: CPU load balancing factor, memory load balancing factor, pod load balancing factor, average power consumption (W), average CPU fragmentation and average memory fragmentation in different experiments.

a) Resource load balancing optimization analysis:

The results demonstrate that the default scheduler consistently outperforms the other scheduler in the CPU load balancing objective, reflecting its inherent design to optimize load distribution. Our NSGA-II-based scheduler (ML-NSGA-II), while effective, achieves slightly worse CPU load balancing, indicating a trade-off with its multi-objective optimization approach. However, the TOPSIS-based scheduler (ML-TOPSIS) exhibits poor performance, which can be due to the low weight attributed to this objective. a similar analysis can be made about memory and pod count load balancing objective where the default scheduler outperforms the other algorithms. The LP-TOPSIS scheduler achieves results similar to our ML-TOPSIS-based scheduler on this objective.

b) Power consumption optimization analysis: In terms of average power consumption, our ML-TOPSIS-based scheduler consistently gives the most energy-efficient results across all three experiments, achieving a remarkable 66% reduction in power consumption. This result can be attributed to the higher weight assigned to the power consumption objective in the scheduler profile, prioritizing energy efficiency during the scheduling process.

The ML-NSGA-II-based scheduler also demonstrated sig-

nificant energy savings, achieving a 40% reduction in power consumption. Although slightly less effective than ML-TOPSIS in terms of absolute energy reduction, ML-NSGA-II maintains a relatively small average gap of 15.43% from the best-performing algorithm (ML-TOPSIS) across all experiments. This result underscores NSGA-II’s ability to balance energy savings with other competing objectives, offering a robust trade-off strategy in multi-objective optimization scenarios and also proves the ability of the node-specific power model in estimating the true power consumption of worker nodes.

The LP-TOPSIS-based scheduler exhibits reduction in power consumption but not as good as our solution, and that is because the linear power model, which does not reflect the actual power consumption of the worker nodes.

In contrast, the default scheduler, while optimized for general load balancing, exhibits higher power consumption, particularly under high workload conditions. This is a direct result of its lack of explicit energy-aware mechanisms, which limits its ability to minimize power usage effectively.

c) Resource fragmentation optimization analysis:

In terms of average CPU and memory fragmentation, the results indicate distinct differences among the schedulers. The ML-NSGA-II-based scheduler performs best in minimizing both CPU and memory fragmentation across all experiments, improving cluster resource utilization with improvement 74% for CPU resources and 68% for memory. Our ML-TOPSIS-based scheduler also shows good performance compared to the default scheduler with 31% improvement for CPU resources and 20% for memory, though it does not match the performance of ML-NSGA-II. The default scheduler, shows higher average fragmentation, indicating that its allocation strategies may lead to more free and unusable resources. The resource fragmentation measure of the LP-TOPSIS scheduler indicates performance close but slightly worse than our ML-TOPSIS-based scheduler.

D. Running time comparison

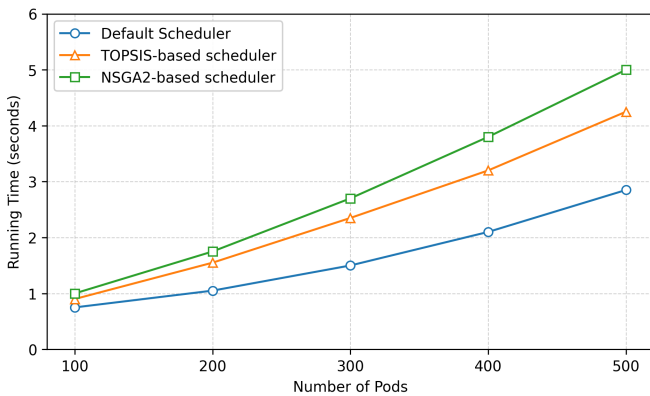


Figure 10: Running time comparison between different algorithms

Fig. 10 compares the running time of each algorithm with respect to the number of pods on the cluster of 10 nodes defined in Table IV. At the lower end of the workload size (100 to 200 pods), all three schedulers show minimal differences in

running times. The Default Scheduler demonstrates the fastest performance, with scheduling time under of 1 second. At this scale, the computational overhead of both NSGA-II and TOPSIS is negligible, with their performance only slightly trailing behind the default scheduler. As the number of pods increases beyond 200, more pronounced differences in running time become evident. The Default Scheduler consistently achieves the lowest running time, confirming its minimal overhead and efficiency. Between the two proposed schedulers, TOPSIS exhibits lower running times than NSGA-II across all larger workloads, but with minimal gap, demonstrating its computational advantage. This efficiency stems from its deterministic scoring mechanism, which requires fewer iterations than population-based evolutionary algorithms. In contrast, NSGA-II, while effective for multi-objective optimization, incurs higher computational overhead as the workload scales, clearly due to its iterative population evaluations and non-dominated sorting operations. Overall, the results highlight that while the Default Scheduler remains the most computationally efficient, TOPSIS offers a favorable trade-off between runtime and optimization performance, achieving slightly faster execution than NSGA-II across all workloads. However, this comes at the expense of solution diversity, as TOPSIS produces a single ranked solution, whereas NSGA-II explores a broader set of Pareto-optimal solutions, offering more flexibility for multi-objective decision making.

E. Cluster-wise scalability analysis

To address the overhead associated with increasing cluster size, we developed a distributed variant of our NSGA-II-based scheduler. In this design, the global population of candidate solutions is partitioned into equal-sized sub-populations, each of which is assigned to a dedicated worker thread. Every worker independently executes the evolutionary operator: selection, crossover, and mutation on its local sub-population. This approach enables parallel fitness evaluation, thereby significantly reducing execution time compared to the centralized version. To avoid premature convergence within isolated sub-populations, we incorporated a migration mechanism. At predefined intervals, a small percentage of elite individuals (e.g., the top $r = 10\%$) are exchanged among sub-populations following a ring topology. This controlled migration maintains genetic diversity across workers, ensuring that the distributed algorithm continues to explore the search space effectively while preserving solution quality. Synchronization is handled at the end of each generation to allow workers to exchange individuals without incurring excessive communication overhead. The workflow of the distributed NSGA-II is summarized in Algorithm 4, and show in Fig. 11.

We evaluated the execution time of the distributed NSGA-II using 4 worker threads for cluster sizes ranging from 100 to 500 nodes, while maintaining a fixed workload of 100 pods. The results, shown in Fig. 12, demonstrate that the distributed version of the genetic algorithm achieves an **average overhead reduction of 52.8%** compared to the centralized implementation. Importantly, this improvement was achieved with only a negligible decrease in solution quality

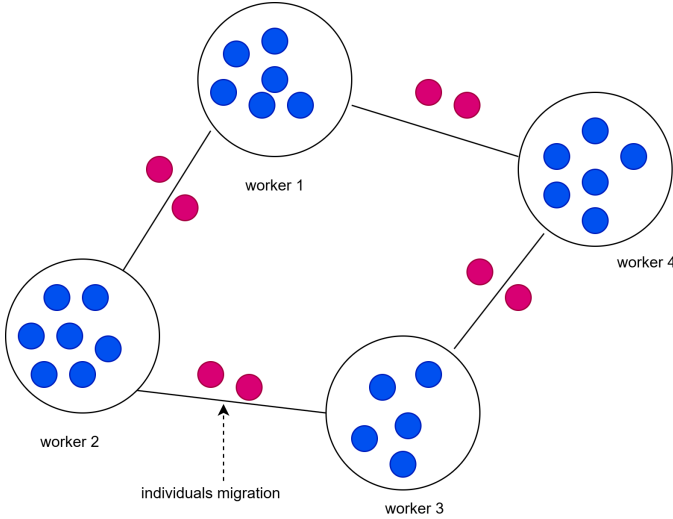


Figure 11: Distributed NSGA-II execution

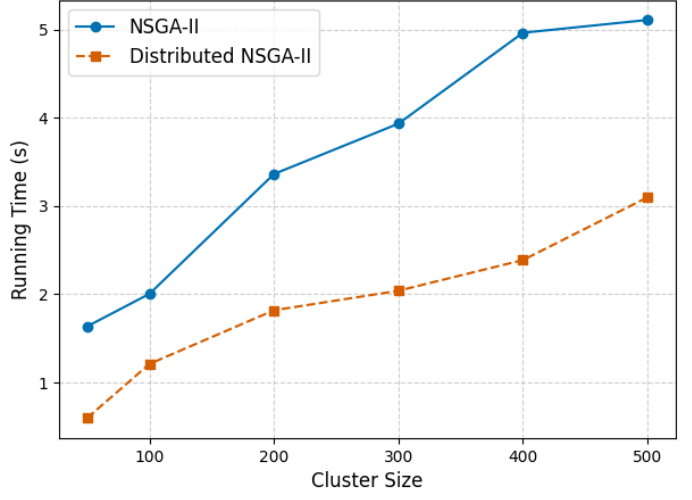


Figure 12: Cluster-wise scalability analysis

(within 2–3% of the centralized approach), confirming that our distributed strategy scales effectively without sacrificing accuracy.

Algorithm 4 Distributed NSGA-II with migration

Input: Population size S , number of thread workers k , max generations G , migration period M , elite rate r

Output: Final Pareto front

```

1 Initialize population  $Pop$  of size  $S$ 
2 Partition  $Pop$  into  $k$  sub-populations  $\{Pop_1, Pop_2, \dots, Pop_k\}$ 
3 for  $g \leftarrow 1$  to  $G_{max}$  do
4   for  $i \leftarrow 1$  to  $k$  in parallel do
5     Evaluate fitness of individuals in  $Pop_i$ 
6     Apply selection, crossover, and mutation to form  $Pop'_i$ 
7      $Pop_i \leftarrow Pop'_i$ 
8   if  $g \bmod mig = 0$  then
9     // Migration every  $mig$  generations
10    for  $i \leftarrow 1$  to  $k$  do
11      Select top  $r\%$  elite individuals  $E_i$  from  $Pop_i$ 
12      for  $i \leftarrow 1$  to  $k$  do
13        Send  $E_i$  to worker  $(i \bmod k) + 1$ 
14        Receive  $E_{i-1}$  from worker  $((i+k-2) \bmod k) + 1$ 
15        Insert received elites  $E_{i-1}$  into  $Pop_i$  and trim to  $|Pop_i|=S/k$ 
16 Merge all  $Pop_i$  and perform non-dominated sorting to obtain the Pareto front
  
```

F. Pareto-front analysis

In this section, we study and demonstrate the effectiveness of the proposed algorithms in identifying Pareto-optimal points within the pod scheduling optimization problem. In multi-objective optimization, the Pareto front represents the set of solutions where no objective can be improved without

compromising at least one other. Algorithm 5 outlines a naive and brute-force search method for determining the set of Pareto-optimal points that constitute the Pareto frontier.

Algorithm 5 Pareto Front Search

Input: Decision matrix $\mathcal{X} \in \mathbb{R}^{N \times |\mathcal{O}|}$, where N is the number of solutions and $|\mathcal{O}|$ is the number of objectives.

Output: Set of Pareto-optimal solutions \mathcal{P} .

```

16 Initialize:  $\mathcal{P} \leftarrow \emptyset$ 
17 foreach solution  $y \in \mathcal{X}$  do
18   dominated  $\leftarrow$  False
19   foreach solution  $y_p \in \mathcal{X}, y \neq y_p$  do
20     if  $\forall i, y_p[i] \leq y[i] \wedge \exists i, y_p[i] < y[i]$  then
21       dominated  $\leftarrow$  True
22       Break
23   end
24 end
25 if dominated = False then
26    $\mathcal{P} \leftarrow \mathcal{P} \cup \{y\}$ 
27 end
28 end
29 return  $\mathcal{P}$ 
  
```

We created 1000 nodes in a simulation environment using the *Kubernetes-scheduler-simulator*⁷. Three pods are deployed, and the output solutions by TOPSIS and NSGA-II algorithms are observed. The reason behind the high number of simulated nodes is to have a clear visualization of Pareto-optimal points and demonstrate the ability of the proposed algorithms to find these solutions.

Figures 13, 14, 15 present visualizations of the search space and the results of the TOPSIS-based and NSGA-II-based pod scheduling algorithms. To enable visualization, the dimensionality of the objectives was reduced. Load balancing metrics (CPU, memory, and pod count) were aggregated using their l^2 -norm, as well as for resource fragmentation metrics (CPU and memory). This method preserves the relative contributions

⁷<https://github.com/kubernetes-sigs/kube-scheduler-simulator>

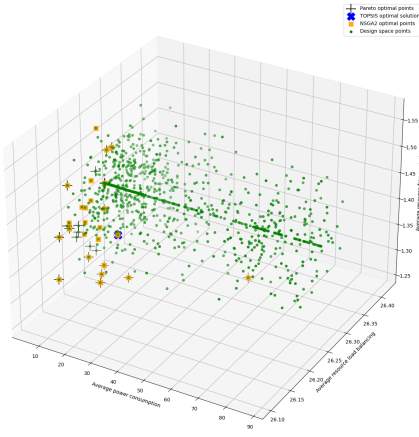


Figure 13: Pareto-front analysis for Pod 1

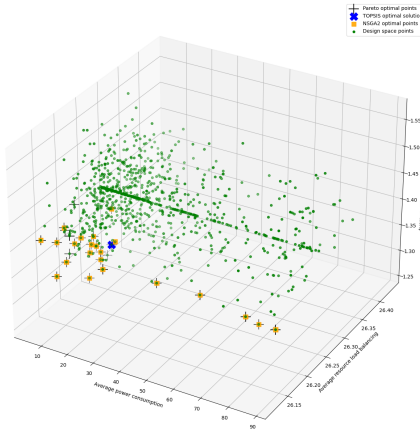


Figure 14: Pareto-front analysis for Pod 2

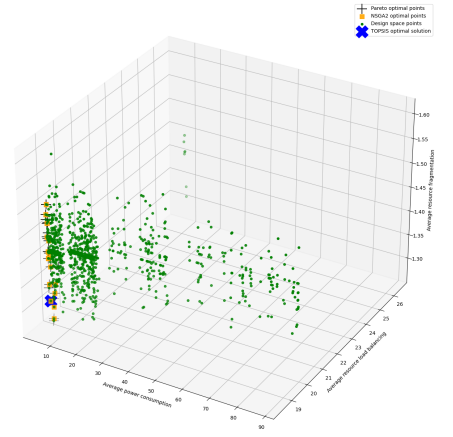


Figure 15: Pareto-front analysis for Pod 3

of individual metrics while simplifying the visualization to a 3-dimensional space, capturing power consumption, load balancing, and resource fragmentation.

a) Analysis: We observe that the TOPSIS-based scheduler produces a single, well-balanced solution that reflects the predefined weights assigned to each objective, making it particularly suitable for scenarios where priorities are well known and a deterministic outcome is preferred. In contrast, NSGA-II generates a diverse set of Pareto-optimal solutions, offering a broad spectrum of trade-off options for decision-makers. For instance, as shown in the figures, NSGA-II finds solutions that achieve lower power consumption at the expense of slightly higher fragmentation, as well as configurations with more balanced resource utilization but moderate power usage.

This diversity highlights NSGA-II's strength in exploring the solution space more extensively, which is particularly valuable when system administrators need flexibility to adapt to dynamic workloads or varying energy constraints. On the other hand, TOPSIS's single solution provides faster and simpler decision-making when such flexibility is not required. Overall, this analysis underscores the complementary nature of the two approaches: TOPSIS excels in quick, weight-driven selection, while NSGA-II enables deeper exploration of the design space for multi-objective trade-offs.

VI. CONCLUSION

In this paper, we presented a two-stage, power-aware Kubernetes scheduling framework that couples node power profiling with a machine-learning power estimator trained on real measurements. The scheduler plugin integrates two multi-objective strategies: a TOPSIS selector that aggregates normalized criteria into a single score, and an NSGA-II evolutionary optimizer that jointly considers energy efficiency, load balancing, and resource fragmentation. A distributed NSGA-II variant partitions the population across workers to parallelize fitness evaluation and maintain diversity via lightweight migration. In evaluation, both approaches outperform the default Kubernetes scheduler. TOPSIS offers fast decisions and strong power savings but is limited by fixed weights and scalability trade-offs; NSGA-II yields more balanced solutions with a favorable quality-time trade-off, especially in the distributed

setting. The framework is modular and readily extensible to additional objectives and plugins.

Future work includes: (i) enriching the power model with other factors such as memory utilization, I/O, and network usage, (ii) consider online rescheduling and migration driven by power predictions (iii) topology and latency awareness via application graphs across the cloud-edge-IoT continuum. We will also broaden evaluation to per-pod scheduling latency, high-arrival throughput, long-run stability, and adaptive weighting for responsiveness under diverse workloads.

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-ic 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] Z. Wan, D. Lo, X. Xia, and L. Cai, "Practical and effective sandboxing for linux containers," *Empirical Software Engineering*, vol. 24, pp. 4034–4070, 2019.
- [5] 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>
- [6] M. Sureshkumar and P. Rajesh, "Optimizing the docker container usage based on load scheduling," in *2017 2nd International Conference on Computing and Communications Technologies (ICCT)*. IEEE, 2017, pp. 165–168.
- [7] S. Hoque, M. S. De Brito, A. Willner, O. Keil, and T. Magedanz, "Towards container orchestration in fog computing infrastructures," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. IEEE, 2017, pp. 294–299.
- [8] 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.
- [9] "The european green deal," https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/european-green-deal_en, 2024, accessed: 2024.
- [10] A. Boudi, M. Bagaa, P. Pöyhönen, T. Taleb, and H. Flinck, "Ai-based resource management in beyond 5g cloud native environment," *IEEE Network*, vol. 35, no. 2, pp. 128–135, 2021.
- [11] J. Santos, C. Wang, T. Wauters, and F. De Turck, "Diktyo: Network-aware scheduling in container-based clouds," *IEEE Transactions on Network and Service Management*, vol. 20, no. 4, pp. 4461–4477, 2023.

- [12] T. Tsokov and H. Kostadinov, "Kinitos: Dynamic network-aware scheduling and descheduling in kubernetes clusters with mobile nodes," *Journal of Network and Computer Applications*, vol. 238, p. 104157, 2025.
- [13] C. Guerrero, I. Lera, and C. Juiz, "Genetic algorithm for multi-objective optimization of container allocation in cloud architecture," *Journal of Grid Computing*, vol. 16, no. 1, pp. 113–135, 2018.
- [14] Y. Qiao, J. Xiong, and Y. Zhao, "Network-aware container scheduling in edge computing," *Cluster Computing*, vol. 28, no. 2, p. 78, 2025.
- [15] W. Rao and H. Li, "Energy-aware scheduling algorithm for microservices in kubernetes clouds," *Journal of Grid Computing*, vol. 23, no. 1, p. 2, 2025.
- [16] T. Menouer, "Kcsc: Kubernetes container scheduling strategy," *The Journal of Supercomputing*, vol. 77, no. 5, pp. 4267–4293, May 2021.
- [17] 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 Softwareization (NetSoft)*. IEEE, Jun 2019, pp. 351–359.
- [18] 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.
- [19] C. Hwang and K. Yoon, *Multiple Attribute Decision Making: Methods and Applications*. New York: Springer-Verlag, 1981.
- [20] D. Zhang, B.-H. Yan, Z. Feng, C. Zhang, and Y.-X. Wang, "Container oriented job scheduling using linear programming model," in *2017 3rd International Conference on Information Management (ICIM)*. IEEE, 2017, pp. 174–180.
- [21] C. Kaewkasi and K. Chuenmuneewong, "Improvement of container scheduling for docker using ant colony optimization," in *2017 9th international conference on knowledge and smart technology (KST)*. IEEE, 2017, pp. 254–259.
- [22] Y. Li, J. Zhang, W. Zhang, and Q. Liu, "Cluster resource adjustment based on an improved artificial fish swarm algorithm in mesos," in *2016 IEEE 13th International Conference on Signal Processing (ICSP)*. IEEE, 2016, pp. 1843–1847.
- [23] M. Imdoukh, I. Ahmad, and M. Alfaiakawi, "Optimizing scheduling decisions of container management tool using many-objective genetic algorithm," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 5, p. e5536, 2020.
- [24] A. Saboor, A. K. Mahmood, A. H. Omar, M. F. Hassan, S. N. M. Shah, and A. Ahmadian, "Enabling rank-based distribution of microservices among containers for green cloud computing environment," *Peer-to-Peer Networking and Applications*, vol. 15, no. 1, pp. 77–91, 2022.
- [25] J. Cai, H. Zeng, F. Liu, and J. Chen, "Intelligent dynamic multi-dimensional heterogeneous resource scheduling optimization strategy based on kubernetes," *Mathematics*, vol. 13, no. 8, p. 1342, 2025.
- [26] M. Ivanović, D. Talia, V. Riccio, and J. Kolodziej, "Cloud-native frameworks for distributed evolutionary computation: A survey," *Future Generation Computer Systems*, vol. 128, pp. 199–220, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167739X21003896>
- [27] M. García-Valdez, A. Tonda, and C. A. Coello Coello, "Asynchronous multi-population metaheuristics for distributed optimization," *Symmetry*, vol. 15, no. 2, p. 467, 2023. [Online]. Available: <https://www.mdpi.com/2073-8994/15/2/467>
- [28] P. Di Cicco, A. Cicio, F. Marozzo, and D. Talia, "Pareto-based multi-objective optimization for service placement across the computing continuum," in *International Conference on Network and Service Management (CNSM)*. IFIP/IEEE, 2024. [Online]. Available: <https://dl.ifip.org/db/conf/cnsm/cnsm2024/1571045691.pdf>
- [29] 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.
- [30] 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.
- [31] J. Joseph, R. S. Kadadevaramath, B. L. Shankar, S.-Y. Chen, and A. Kadadevaramath, "An integrated fuzzy multi-criteria approach for evaluating sustainable suppliers in supply chains: Ahp, topsis, and vikor," *International Journal of Environmental Sciences*, pp. 173–183, 2025.
- [32] A. A. Gad-Elrab *et al.*, "Adaptive multi-criteria-based load balancing technique for resource allocation in fog-cloud environments," *arXiv*, 2024.
- [33] P. Pradeep and E. Al-Masri, "Energy-optimized scheduling for aiot workloads using topsis," *arXiv*, 2025.
- [34] B. Bhabani and J. Mahapatro, "Performance evaluation of priority-based scheduling in hybrid vanets for different criteria weights using ahp-ahp and ahp-topsis," *IETE Journal of Research*, vol. 70, no. 6, pp. 5759–5770, 2023.
- [35] 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.
- [36] J. Puchinger, G. R. Raidl, and U. Pferschy, "The multidimensional knapsack problem: Structure and algorithms," *INFORMS Journal on Computing*, vol. 22, no. 2, pp. 250–265, 2010.
- [37] Y. Zhang and N. Ansari, "Power consumption estimation in data centers using machine learning techniques," *IEEE Access*, vol. 5, pp. 1470–1479, 2017.
- [38] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.
- [39] Y. Li, A. Zhou, X. Ma, and S. Wang, "Profit-aware edge server placement," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 55–67, 2021.
- [40] S. Chen, J. Chen, and H. Li, "Joint optimization of upf placement and traffic routing for 5g core network user plane," *Computer Communications*, vol. 216, pp. 86–94, 2024.
- [41] S. X. Xie, D. Liao, and V. M. Chinchilli, "Measurement error reduction using weighted average method for repeated measurements from heterogeneous instruments," *Environmetrics: The official journal of the International Environmetrics Society*, vol. 12, no. 8, pp. 785–790, 2001.
- [42] G. Biau and E. Scornet, "A random forest guided tour," *Test*, vol. 25, pp. 197–227, 2016.
- [43] R. Iranzad and X. Liu, "A review of random forest-based feature selection methods for data science education and applications," *International Journal of Data Science and Analytics*, pp. 1–15, 2024.
- [44] A. W. Lewis, S. Ghosh, and N.-F. Tzeng, "Run-time energy consumption estimation based on workload in server systems," *HotPower*, vol. 8, pp. 17–21, 2008.
- [45] K. De Vogeleer, G. Memmi, P. Jouvelot, and F. Coelho, "The energy/frequency convexity rule: Modeling and experimental validation on mobile devices," in *Parallel Processing and Applied Mathematics: 10th International Conference, PPAM 2013, Warsaw, Poland, September 8-11, 2013, Revised Selected Papers, Part I 10*. Springer, 2014, pp. 793–803.

VII. BIOGRAPHY



Mohammed Dhiya Eddine Gouaouri Mohammed Dhiya Eddine Gouaouri is a doctoral student at Université du Québec à Trois-Rivières (UQTR), QC, Canada. He received his M.Sc and Engineering degrees from the Higher National School of Computer Science (ESI), Algeria, in 2023. His research focuses on cloud-edge collaboration optimization and machine learning.



Miloud Bagaa (Senior Member, IEEE) received his engineering, master's, and Ph.D. degrees from the University of Science and Technology Houari Boumediene, Algiers, Algeria, in 2005, 2008, and 2014, respectively. From 2009 to 2015, he was a researcher at the Research Center on Scientific and Technical Information, Algiers. From 2015 to 2020, he held postdoctoral and senior researcher positions at the Norwegian University of Science and Technology and Aalto University. He is currently a professor in the Department of Electrical and Computer Engineering, Université du Québec à Trois-Rivières (UQTR), QC, Canada.



Sihem Ouahouah Sihem Ouahouah received the Engineering degree from the University of Science and Technology Houari Boumediene, Bab Ez-zouar, Algeria, in 2005, and the master's degree in computer science from Ecole Nationale Supérieure d'Informatique, Oued Smar, Algeria, in 2015. She is currently pursuing the Doctoral degree with Aalto University, Espoo, Finland. Her research interests include unmanned aerial vehicles, machine learning, and the Internet of Things.



Messaoud Ahmed Ouameur Messaoud Ahmed Ouameur received his bachelor's degree in electrical engineering from the National Institute of Electronics and Electrical Engineering (INELEC), Boumerdes, Algeria, in 1998. He earned an M.B.A. from the Graduate School of International Studies, Ajou University, South Korea, in 2000, and his master's and Ph.D. degrees (Hons.) in electrical engineering from Université du Québec à Trois-Rivières (UQTR), QC, Canada, in 2002 and 2006, respectively. He has extensive experience in software-

defined radio systems and embedded real-time systems, with research interests in distributed massive MIMO, reconfigurable intelligent surfaces, and embedded deep learning systems for communication and IoT.



Adlen Ksentini Adlen Ksentini (Senior Member, IEEE) is a professor in the Communication Systems Department at EURECOM. He leads the Network Softwarization group, focusing on activities related to network softwarization, 5G/6G, and edge computing. His research interests include network softwarization and cloudification, with an emphasis on network virtualization, Software-Defined Networking (SDN). He has participated in several H2020 and Horizon Europe projects on 5G and beyond, such as 5G!Pagoda, 5GTransformer, 6GBricks, 6G-Intense,

Sunrise-6G, and AC3. He serves as the technical manager for 6G-Intense and AC3, focusing on zero-touch management of 6G resources and applications, as well as the Cloud Edge Continuum, respectively. His interests extend to system and architectural challenges as well as algorithmic problems related to these topics, utilizing tools such as Markov Chains, optimization algorithms, and machine learning (ML). He is also a member of the OAI board of directors, where he oversees OAI 5G Core Network and O-RAN management (O1 and E2) for OAI RAN activities.