GreenK8s: Green-aware Scheduling for Sustainable Kubernetes Cluster Management

Yifan Sun*†, Minxian Xu‡§, Adel N. Toosi*†§
*Computing and Information Systems, The University of Melbourne, Melbourne, Australia
yifan.sun.4@student.unimelb.edu.au, adel.toosi@unimelb.edu.au
† Faculty of Information Technology, Monash University, Clayton, Australia
‡Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China
mx.xu@siat.ac.cn

2025 IEEE International Conference on Cluster Computing (CLUSTER) | 979-8-3315-3019-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/CLUSTER59342.2025.11186459

***Abstract*—With the rise of large-scale data centers and increasing demand for energy-efficient operations, there is a growing need to optimize the use of green energy in cloud computing environments. However, current schedulers focus solely on performance, lacking awareness of energy types and opportunities to promote green, low-carbon operations. This paper presents a Green-Aware Scheduling Framework for Kubernetes, named GreenK8s, aimed at minimizing the use of brown energy and maximizing the utilization of renewable energy sources, specifically solar power. Our framework integrates real-time power consumption monitoring with predictive solar energy models to intelligently schedule workloads based on energy availability. The proposed solution incorporates an AI-based solar power prediction model, Pod oversubscription strategies, and a novel scheduler, enabling Kubernetes to dynamically adapt to both the type and availability of green energy. Extensive experiments using the real-world Google Borg dataset and a realistic Kubernetes testbed demonstrate that GreenK8s reduces total energy consumption by up to 39% and increases the average share of green energy in total consumption to 50.65%, compared to state-of-the-art baselines. This work provides a promising approach to improve operational efficiency and sustainability in data centers.**

***Index Terms*—Energy efficient, Cluster architectures, Cloud Computing, Kubernetes, Green energy, Scheduling**

I. Introduction

Data centers, as the core of cloud computing, are major electricity consumers with demand expected to grow from 286 TWh (2016) to 321 TWh by 2030 [1] [2] [3]. They contribute around 0.5% of total U.S. greenhouse gas emissions [4], making their environmental impact increasingly urgent [5]. Integrating renewable energy is essential to mitigate pollution and climate change [6], with companies like Amazon already achieving 100% renewable usage [7]. Renewable-powered data centers [8], energy-aware virtual machine management [9], and sustainable cloud frameworks [10] have become increasingly common.

With the continuous advancement and abstraction of cloud data centers, the concept of containers has been introduced accordingly. Containerization, along with container orchestration tools like Kubernetes (K8s for short) [11], has revolutionized the cloud computing landscape by providing more efficient resource management and deployment solutions [12].

§Corresponding authors.

One of the core components of Kubernetes is its scheduler [13], which is responsible for intelligently assigning containers to the most appropriate physical or virtual machine based on the containers' resource requirements, priorities, quality of service (QoS) guarantees, and other factors [14]. However, the current scheduling is more focused on performance, such as more efficient allocation of hardware, CPU, and memory, when selecting qualified nodes.

While these performance-oriented strategies enhance utilization, they often overlook energy optimization and carbon reduction. This disparity conflicts with the global objectives of combating climate change and establishing effective ecosystem management strategies [15]. We aim to address these challenges by integrating green energy-awareness into the Kubernetes platform. Nevertheless, integrating energy type awareness and reducing carbon emissions adds complexity to scheduling, including the need to monitor real-time energy availability and balance it with QoS requirements. The key research question is "*How can Kubernetes intelligently adapt workloads to dynamically adjust cluster energy consumption based on the availability of green energy, achieving a balance between performance and sustainability?*"

To address this research question, we aim to develop a novel green scheduler within the Kubernetes scheduling framework, named GreenK8s. The new scheduling framework allows the scheduler to have energy source awareness, determine whether the cluster is currently using green energy or traditional brown energy, and schedule the next Pod accordingly, maximizing green energy use and ensuring QoS. To achieve our objectives, our **main contributions** are summarized as:

- **A Green-energy-aware framework**: We develop a Kubernetes scheduler capable of differentiating between green and brown energy sources, enabling real-time energy-aware scheduling.
- **Pod oversubscription strategy**: We design a Pod oversubscription mechanism that consolidates workloads under green energy constraints without compromising the QoS of critical applications by analyzing Pod resource usage and allocation history.
- **Solar power prediction**: We employ a CNN-based solar prediction model, trained on datasets from the Australian Bureau of Meteorology (BOM) [16], to enable proactive

and green-aware scheduling decisions.

- **Implementation & Evaluation**: We deploy and evaluate the scheduler in a realistic Kubernetes cluster with real solar energy input and Google Borg workload, achieving up to 39.34% energy reduction and 50.65% green energy utilization improvement, compared to the state-of-the-art.

II. Related Work

Current scheduling approaches primarily focus on hardware allocation, such as CPU and memory, to optimize performance. For example, Liu et al. [29] and Ding et al. [30] demonstrate how Kubernetes can efficiently use resources and maintain high service quality [31]. While the Kubernetes scheduler has evolved to enhance performance and resource utilization, its awareness of green energy remains limited. Table I highlights some studies that focus on energy efficient scheduling techniques; however, gaps remain in areas such as identifying green and brown energy types, maximizing green energy usage, and ensuring broader applicability within Kubernetes environments.

Early work on energy-aware scheduling primarily aimed to enhance system performance. For example, Zhang et al. [32], Syrigos et al. [20], Wu et al. [24], and Rattihalli et al. [21], Zhang et al. [33] and [18] aimed to improve the overall hardware resource utilization efficiency in cloud environments to reduce costs, save energy, and indirectly lower carbon emissions. However, these studies either did not integrate real-time green energy data, did not distinguish between different types of energy, or relied on simulated experimental tests, which limits their practical applicability. Kaur et al. [17] proposed the idea of green energy-aware scheduling, attempting to schedule tasks to low-carbon or green power nodes to reduce power consumption. Cai et al. [25] further advanced the efforts by attempting to achieve higher performance and energy efficiency in heterogeneous data centers powered by green energy, while reducing overall operational costs and carbon emissions.

Building on earlier efforts, Piontek et al. [19] and Chadha et al. [22] went a step further by attempting to incorporate carbon intensity, carbon factors, and CO_2 awareness from different regions around the world into scheduling considerations. Although these methods help reduce carbon emissions, they lack mechanisms to prioritize green energy in real time. Moreover, scheduling based on regional carbon intensity across data centers is highly complex, and applications may face network instability during transitions between data centers, leading to degraded performance and longer deployment times.

Recent studies have introduced new methods that integrate real-time energy data to optimize resource allocation based on energy availability. Chakraborty et al. [26] target micro-cloud data centers, prioritizing green energy when abundant; Serenari et al. [27] focus on emission-aware scheduling for serverless functions; and Patel et al. [28] employ a stable matching algorithm to align workloads with available green energy in cloud-edge environments. Despite these advancements, they remain platform-specific and do not address Kubernetes clusters within a single data center. Moreover, their experiments rely on simulations and have not been tested in real-world Kubernetes environments.

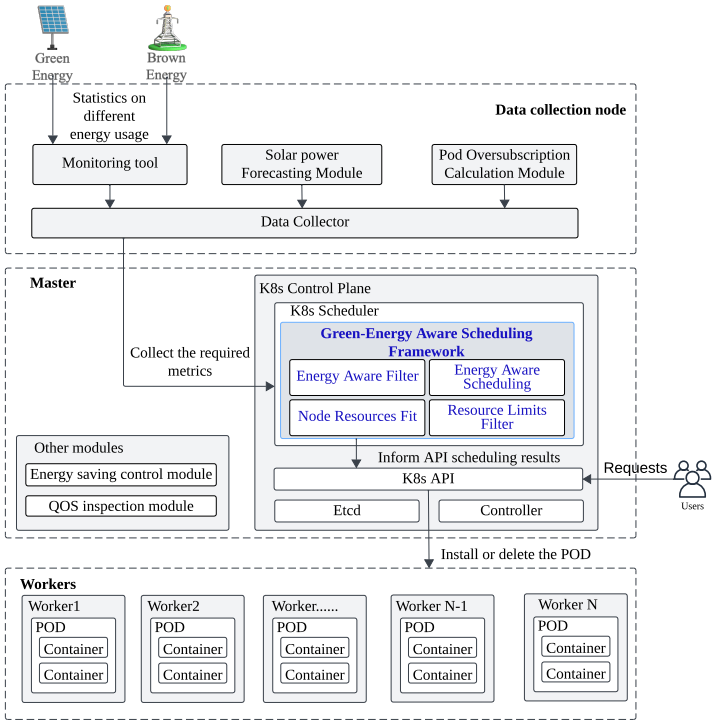


Fig. 1: GreenK8s System Architecture Overview

To address these limitations, our work proposes a green energy-aware scheduling framework specifically for Kubernetes in single data center environments. This framework integrates real-time awareness of green and brown energy, dynamically adjusting workloads to optimize green energy utilization, thereby enhancing Kubernetes' energy efficiency. Our approach maximizes green energy usage and reduces reliance on traditional brown energy, contributing to sustainable container orchestration in Kubernetes environments.

III. System Architecture

This section provides an overview of the green energy-aware scheduler and details the overall system architecture. We focus on a Kubernetes cluster powered by a grid-tied solar energy system. While the system primarily utilizes solar energy, the framework is equally applicable to other types of renewable energy sources. The solar panels generate DC (Direct Current) power, which is then converted into AC (Alternating Current) power by photovoltaic inverters, making it usable by data center hardware.

The system architecture, as shown in Figure 1, consists of three main components: the Master node, K8s worker node group, and the Data collection node.

1) Master node: The Master node acts as a central hub, hosting the green scheduling framework and Kubernetes control plane. In addition to fulfilling the basic functions of

TABLE I: Comparison of Related Work

Work	K8s Support	Energy Source Awareness	Focus	Platform	Energy Data Integration	Green Energy Use	New Scheduler	Energy Adaptive Workload	Real Experiment
Kaur et al. [17]	✓	✓	Reduce energy	IOT		✓	✓		
Menouer [18]	✓		Improve efficiency	K8s			✓		
Piontek et al. [19]	✓		Reduce carbon emissions	K8s			✓	✓	
Syrigos et al. [20]	✓		Reduce power and latency	K8s			✓		✓
Rattihalli et al. [21]	✓		Improve utilization	K8s			✓		✓
Chadha et al. [22]	✓		Reduce carbon emissions	K8s			✓		✓
Jorge-Martinez et al. [23]	✓		Improve utilization	K8s			✓		
Wu et al. [24]			Improve performance	FaaS			✓		✓
Cai et al. [25]		✓	Improve performance	Data Center	✓	✓	✓		✓
Chakraborty et al. [26]		✓	Reduce brown energy use	Cloud	✓	✓	✓	✓	
Serenari et al. [27]		✓	Reduce carbon emissions	OpenWhisk	✓	✓	✓	✓	
Patel et al. [28]		✓	Reduce brown energy use	Cloud-Edge	✓	✓	✓	✓	
GreenK8s	✓	✓	Reduce brown energy use, ensure QoS	K8s	✓	✓	✓	✓	✓

Kubernetes, the Master node also includes additional modules, such as an energy-saving control module and a QoS check module. These are responsible for activating energy-saving measures when green energy is insufficient and for checking the QoS of Pods.

2) K8s worker node group: This node group is responsible for Pod deployment and execution, providing hardware resources for Pods. The group consists of multiple Kubernetes worker nodes, each of which is capable of receiving Pod creation requests. The scheduler selects a target node for Pod deployment by filtering worker nodes based on their available hardware resources.

3) Data collection node: The node provides key system data such as solar power generation and energy consumption of the entire cluster. It includes several modules:

Monitoring tool: Performs real-time monitoring of green energy supply, brown energy usage, and overall system power consumption.

Solar power forecasting module: Provides solar forecasts for future intervals.

Pod oversubscription calculation module: Evaluates the resource usage of Pods based on their historical hardware utilization.

Data collector: Provides an interface that allows scheduling plugins to access the collected data.

IV. GreenK8s System Model

We discretize time into intervals over a finite horizon T, with discrete steps $t \in \{0, 1, \ldots, T\}$, where each t represents a specific time step.

As mentioned earlier, we focus on a Kubernetes cluster powered by a grid-tied solar energy system. When there is sufficient solar energy, any excess unused power is transferred to the power grid, i.e., $export(t) > 0$. Conversely, when solar energy is insufficient, the cluster draws power from the grid, i.e., $import(t) > 0$. Note that $import(t) \times export(t) = 0$ for all t, i.e., at any given time t, either energy is being imported or exported, but not both simultaneously.

In a Kubernetes cluster, the deployment consists of pods and nodes. Each pod Pod_i ($i \in \{1, 2, \ldots, M\}$) is assigned to a specific node Node_j ($j \in \{1, 2, \ldots, N\}$).

A. Cluster Power and Green Solar Power

We assume that all K8s related nodes in the system are connected to an ePDU (Enclosure Power Distribution Unit) that can monitor their power consumption in real time. We use SNMP (Simple Network Management Protocol [34]) to read the power consumption of the servers remotely by the Data collection node. This enables us to accurately calculate the total power usage across all servers in the system. The *total power* (*TP*) consumption of all Kubernetes nodes in the cluster can be calculated by the following formula:

$$TP(t) = \sum_{j=1}^{N} P_j(t), \tag{1}$$

where N represents the number of Kubernetes nodes, and $P_i(t)$ represents the power consumption of the i^{th} Kubernetes node at time t.

The amount of *green power* (*GP*) generated by the system is monitored through the photovoltaic inverter. The inverter records the current (*GP*) data in a 5-minute cycle. We collect these data in watts, and they are readable through a REST API.

B. Power Balance

The real-time power balance of the cluster is the difference between the total power consumption and the amount of green power generated. We define:

$$power\_balance(t) = GP(t) - TP(t), \tag{2}$$

where GP corresponds to *green power*, representing the total instantaneous *green power* generation at time t; TP corresponds to *total power*, representing the total instantaneous power consumption of the cluster at time t.

If $power\_balance(t) > 0$, suggests that green power suffices to support the current workload of the system. If

$power\_balance(t) < 0$, it indicates that green power is insufficient. When there is sufficient green power, the scheduling attempts to allocate the maximum amount of hardware resources requested by the Pod (e.g., `LIMITS` parameter in resource management settings of Pod in K8s) to ensure the highest QoS. It also activates (wakes up) any sleeping worker nodes as required (if any). If there is still excess green power, the system exports the excess power back to the grid to contribute to the grid overall power supply system. When green power is insufficient, the scheduler tries to suspend some Pods deployment or put some worker nodes to sleep to save electricity. If our measures ultimately cannot guarantee power supply, the system automatically imports power from the grid to ensure uninterrupted operation. The balance between solar and grid power ensures a stable and reliable power supply.

C. Brown Energy Usage

The primary goal is to minimize brown energy usage while maximizing QoS within the limits of available green energy. To achieve this, we define our objective functions as follows:

The total energy consumption (TE) and green energy supply (GE) are computed as:

$$TE = \sum_{t=0}^{T} TP(t) \cdot \Delta t, \tag{3}$$

$$GE = \sum_{t=0}^{T} GP(t) \cdot \Delta t, \tag{4}$$

where TE represents the total energy consumption of the cluster over the scheduling period (time horizon); GE represents the total green energy supply over the scheduling period; t represents the t-th time step during the monitoring period; Δt represents the duration of each time step; T represents the total number of time steps in the scheduling period.

The brown energy consumption (BE) is computed as:

$$BE = \sum_{t=0}^{T} \left[TP(t) - GP(t)\right]^{+} \cdot \Delta t, \tag{5}$$

where BE represents the total brown energy consumption over the scheduling period; t represents the t-th time step during the monitoring period; Δt represents the duration of each time step; T represents the total number of time steps in the scheduling period; $[x]^{+}$ ensures that only the positive differences between total power and green power are accounted for, effectively isolating the portion of energy demand supplied by non-renewable (brown) energy. Mathematically, this is represented as $\max(x, 0)$, where any negative values are set to zero.

D. Quality of Service

To minimize QoS Violation Time (QVT), we define (QVT) as the total QoS violation time over the scheduling period:

$$QVT = \sum_{t=1}^{T} \sum_{j=1}^{N} \mathbb{I}\left(Load_j(t) > C_j\right), \tag{6}$$

where $\text{Load}_j(t)$ is the aggregated CPU load of all Pods running on Node j at time t, C_j is the capacity of Node j, and $\mathbb{I}(\cdot)$ is the indicator function that evaluates to 1 if the condition is true and 0 otherwise.

To minimize the cumulative delay experienced by all pods, either while waiting to be scheduled or for resource availability:

$$TPT = \sum_{t=0}^{T} \sum_{i=1}^{M} \mathbb{I}(\sum_{j=1}^{N} S_{i,j} = 0), \tag{7}$$

where TPT represents the total pending time of all Pods over the scheduling period; The $S_{i,j}(t) \in \{0, 1\}$ indicates whether Pod i is running on Node j at time t ($S_{i,j}(t) = 1$ if scheduled and running; 0 otherwise), and $\mathbb{I}(\cdot)$ is the indicator function.

E. Optimization Problem Formulation

Our objective is to minimize brown energy usage while maximizing QoS and minimizing Pod scheduling delays, within the limits of available green energy.

$$\min\left(\alpha \cdot BE + \beta \cdot QVT + \gamma \cdot TPT\right), \tag{8}$$

where:

- BE: Total brown energy consumption over the scheduling period.
- QVT: Total QoS violation time.
- TPT: Total pending time for Pods.
- α, β, γ: Weight coefficients that determine the relative importance of energy efficiency, QoS, and Pod responsiveness, respectively.

Subject to the following constraints:

$$TP(t) > 0, \quad \forall t \in [0, T] \tag{9}$$

$$GP(t) \geq 0, \quad \forall t \in [0, T] \tag{10}$$

$$import(t) - export(t) = TP(t) - GP(t), \quad \forall t \in [0, T] \tag{11}$$

$$import(t) \times export(t) = 0, \quad \forall t \in [0, T] \tag{12}$$

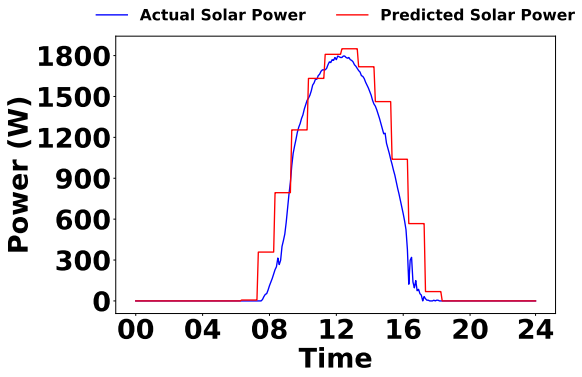
$$\sum_{j=1}^{N} S_{i,j}(t) \leq 1, \quad \forall i \in [1, M],\ \forall t \in [0, T] \tag{13}$$

$$S_{i,j}(t) \in \{0, 1\}, \quad \forall i, j, t \tag{14}$$

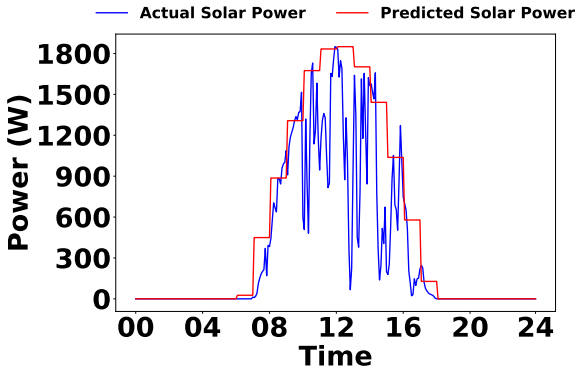
$$\mathbb{I}(Load_j(t) > C_j) = \begin{cases} 1 & \text{if } Load_j(t) > C_j \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

$$\mathbb{I}\left(\sum_{j=1}^{N} S_{i,j}(t) = 0\right) = \begin{cases} 1 & \text{if } Pod_i \text{ is not scheduled at } t \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

Constraint (9) ensures that total power usage at any given time is strictly positive. (10) ensures non-negative green power generation. Since renewable sources like solar energy cannot produce negative energy. (11) and (12) define the power balance logic. (11) ensures that the net power drawn from or supplied to the grid matches the difference between total consumption and green energy production and (12) enforces mutual exclusivity between importing and exporting power,



(a) Sunny day



(b) Cloudy day

Fig. 2: Actual and predicted solar power

i.e., the system cannot import and export simultaneously at any time t. (13) ensures that each Pod is assigned to *at most one* node at any given time. (14) ensures that the scheduling decision variable $S_{i,j}(t)$ is binary, signifying whether a Pod is assigned (1) or not (0) to a node at time t. This is essential for defining the scheduling problem as a mixed-integer optimization model. (15) introduces an indicator function to detect QoS violations. It outputs 1 when the load on a node exceeds its capacity, contributing to the QoS Violation Time (QVT) metric. (16) uses an indicator function to identify Pods that are not scheduled at a given time. If a Pod is not placed on any node, the expression evaluates to 1 and contributes to the Total Pending Time (TPT) metric.

V. METHODOLOGY

Due to the mixed-integer and time-coupled nature of the optimization problem (8), solving it exactly is computationally expensive and infeasible for real-time use. To enable timely and practical scheduling, we design our lightweight heuristic that leverages solar forecasts, system load, and QoS metrics to make efficient, green-aware scheduling decisions. In the following, we present the core components of our proposed scheduling approach.

A. Solar power generation prediction

Since solar power generation is highly variable and heavily influenced by weather conditions, as demonstrated in Figures 2a and 2b, fluctuations in solar intensity can significantly impact the amount of power generated. These rapid variations can destabilize the system. Frequent adjustments to compensate for these fluctuations lead to higher operational costs. Moreover, designing a reactive scheduling system that responds to every change in solar power generation is inefficient, making it impractical to base scheduling decisions solely on real-time green power monitoring. Therefore, we predict near-future trends and changes in solar power supply, enabling the development of proactive scheduling methods that can smooth out power fluctuations, making the system less sensitive to sudden spikes and dips. By forecasting solar power availability in the near future, we can mitigate instability caused by weather conditions, allowing the system to make informed decisions in advance, ensure stability, and reduce unnecessary costs. Therefore, we added a green energy prediction module to our system, which allows the system to be dispatched based on the predicted green energy.

In this work, we use a solar irradiance dataset from the Australian Bureau of Meteorology [16], containing daily solar irradiation data across Australia from 1990 to 2017, to develop a solar power prediction method. We extracted irradiation data for the Clayton area in Melbourne, Victoria, and trained a Convolutional Neural Network (CNN) model to predict hourly solar radiation for any given day. The model uses a sequential architecture with one 1D convolutional layer (64 filters, kernel size 2), dropout, and fully connected layers. It was trained on a normalized dataset using the Adam optimizer with MSE loss, converging within 100 epochs. At the same time, the model leverages a sliding window feature extraction method to capture time series patterns for predicting future radiation values. We trained multiple models using data from 1990 to 2016 as the training set and tested predictions against 2017 data. By normalizing and aligning the peak times of predicted radiation with on-site solar generation data, we get the predicted green power (PGP) , which effectively mitigated supply instability caused by weather changes (Figures 2). Multiple methods were used in subsequent experiments to evaluate the accuracy and effectiveness of the prediction.

B. Pod Oversubscription

Based on the Kubernetes default scheduler source code and official documentation, the current scheduler selects Worker Nodes using the `REQUESTS` and `LIMITS` fields in the Pod YAML file. `REQUESTS` represent guaranteed resources, while `LIMITS` define the maximum resources a Pod can use when excess capacity is available. The scheduler calculates remaining Node resources based on `REQUESTS`. If sufficient resources are available, the Node is considered qualified, and the scheduler may deploy the Pod on it.

This static, parameter-based scheduling has limitations, leading to hardware underutilization or performance issues. For example, a server with 4 CPUs and 4 GB of memory can

run 4 Pods, each with a REQUESTS of 1 CPU and 1 GB and LIMITS of 2 CPUs and 2GB. If actual usage stays below REQUESTS, resources are underutilized. Conversely, if usage exceeds REQUESTS for all Pods, resource contention arises, causing performance degradation.

To address this, we propose a *Pod Oversubscription Technique*, which leverages a probabilistic factor, denoted as P_{over}. This factor estimates the likelihood of a Pod exceeding its REQUESTS value. The formula is:

$$P_{over} = \frac{(100 - P_{\text{exceed}})}{100}, \tag{17}$$

where P_{exceed} is the smallest percentile where CPU usage surpasses REQUESTS setting. This is calculated using the Pod's historical CPU usage distribution.

Google's Borg dataset [35] includes runtime data, CPU and memory REQUESTS, average and maximum CPU usage, and CPU usage distributions across 11 percentiles: 0% (minimum), 10%, 20%, ..., 100% (maximum). Hence, in our experiments, P_{over} can be conveniently calculated using these values.

C. Scheduler and Algorithm Design

We implement K8s custom plugins to build our GreenK8s scheduler with advanced features.

*1) **Energy-Aware Filter Plugin**:* This plugin selects suitable nodes from the available pool based on predefined criteria. Algorithm 1 outlines the plugin's operation. Upon activation, it initializes the $suitable\_nodes[]$ list and checks the current green power (GP) status (Lines 1–3). If the green power (GP) supply is lower than the cluster's current power TP consumption, the scheduler filters nodes based on the REQUESTS parameter specified in the Pod YAML file to minimize resource usage and conserve power (Lines 3–6).

When green power (GP) is sufficient, the scheduler instead uses the previously calculated P_{over} value for each Pod. Based on the probability distribution of all P_{over} values, two thresholds are defined: p_min (lower bound) and p_max (upper bound). If a Pod's P_{over} exceeds p_max, indicating a high likelihood of exceeding the REQUESTS limit, the scheduler filters nodes based on the LIMITS parameter (Lines 7–10). This approach helps maintain QoS and ensures adequate resources for performance-critical Pods. Conversely, if a Pod's P_{over} falls below p_min, node filtering is based on the REQUESTS parameter, as the Pod is unlikely to exceed its requested resources—thus promoting energy efficiency and maximizing resource utilization.

For Pods with P_{over} values between p_min and p_max, a linear interpolation strategy is used to dynamically adjust their resource requests (Lines 11–15). Specifically, resource requests scale proportionally: as P_{over} approaches p_max, the requests trend toward the LIMITS value; as it nears p_min, they approach the REQUESTS value. This adaptive strategy balances energy savings with performance needs.

More specifically, when the P_{over} lies between p_min and p_max, a step coefficient is calculated to determine the relative position of P_{over} between these two probability boundaries. The adjusted hardware requests are calculated using the following formula:

$$adjusted\_req = REQUESTS + \frac{P_{over} - p\_min}{p\_max - p\_min} \times (LIMITS - REQUESTS). \tag{18}$$

This plugin aims to minimize *BE* (Equation 5) by adjusting Pod resource allocations based on real-time solar power and P_{over} probability. When green energy is insufficient, compressing requests helps avoid excess brown energy usage. When green energy is abundant, resource expansion improves performance and reduces *QVT* (Equation 6).

*2) **NodeResourcesFit plugin**:* The NodeResourcesFit plugin, a default Kubernetes scheduler plugin, scores eligible nodes to select the best one for Pod deployment. By default, it balances Pod loads across nodes. We modified it to increase the weight of the 'CPU allocated' score, so when NodeResourcesFit receives a list of eligible nodes from the Energy-Aware Filter plugin, it prioritizes nodes with higher CPU usage. This enhances hardware resource utilization, optimizes resource packing, and seamlessly integrates with our automatic node sleep and wake-up functions. This adjustment helps consolidate workloads on fewer nodes, enabling more nodes to enter sleep mode, thereby reducing *BE* (Equation 5).

*3) **Energy-Aware Scheduling plugin**:* The plugin serves as the final authority on Pod creation, deciding whether a Pod can be deployed to a node. Algorithm 2 outlines its execution logic. Upon activation, the plugin retrieves green power (GP) and total power (TP) values to evaluate current solar generation and cluster power consumption. If green

Algorithm 1 Energy-Aware Filter Plugin

Input: TP, GP, P_{over}, p_max, p_min, $worker\_node[]$, $pod\_REQUESTS$, pod_LIMITS
Output: List of suitable $suitable\_nodes[]$ to deploy Pod or Pending state

1: **Initialize** $suitable\_nodes[] \leftarrow []$
2: **for** each node $n \in worker\_node[]$ **do**
3: **if** $GP \leq TP$ **or** $P_{over} \leq p\_min$ **then**
4: **if** Meets $pod\_REQUESTS$ **then**
5: **Add** n to $suitable\_nodes[]$
6: **end if**
7: **else if** $P_{over} > p\_max$ **then**
8: **if** Meets pod_LIMITS **then**
9: **Add** n to $suitable\_nodes[]$
10: **end if**
11: **else**
12: $adjReq \leftarrow$ adjustedReq(P_{over}, pod)
13: **if** Meets $adjReq$ **then**
14: **Add** n to $suitable\_nodes[]$
15: **end if**
16: **end if**
17: **end for**
18: **return** $suitable\_nodes[]$ if not empty, otherwise Pod Pending scheduling

Algorithm 2 Energy-Aware Scheduling Plugin

Input: TP, GP, $worker\_node[]$, $current\_time$, $pod\_priority$, $target\_node$, pod **Output:** Permit pod deploy in $target\_node$ or Pending state

1: **if** Sufficient Green Energy ($GP > TP$) **then**
2: **Permit** pod deploy in $target\_node$
3: **else if** High Priority Pod ($pod\_priority \geq 7$) **then**
4: **Permit** pod deploy in $target\_node$
5: **else if** Medium Priority Pod($4 \leq pod\_priority \leq 6$) **then**
6: **if** Low electricity rate time ($current\_time \geq 22PM$ **or** $current\_time \leq 8AM$) **then**
7: **Permit** pod deploy in $target\_node$
8: **else**
9: **Pending** 1 minute, then Restart Scheduling
10: **end if**
11: **else**
12: **Pending** 1 minute, then Restart Scheduling
13: **end if**

power (GP) is sufficient, the plugin approves and deploys the Pod on the pre-selected target node, completing the scheduling round (Lines 1–2).

When green power (GP) is insufficient, deployment decisions are based on $pod\_priority$, a label from 0 to 10 indicating priority. High-priority Pods ($pod\_priority \geq 7$) are deployed regardless of green power availability to maintain critical operations, such as web servers (Lines 3–4). Medium-priority Pods ($4 \leq pod\_priority \leq 6$) are delayed to off-peak electricity supply hours (e.g., 22:00–8:00) if green power is insufficient, as they can tolerate some delay (e.g., daily processing tasks) (Lines 5–7).

If none of these conditions are met, indicating that the Pod belongs to the low-priority category ($pod\_priority \leq 3$), which represents non-essential workloads such as test environments or batch data processing for analytics, the scheduling is paused for 1 minute and retried at regular intervals until the deployment succeeds (Lines 8–12).

This Energy-Aware Scheduling plugin aims to minimize *TPT* (Equation 7) by allowing high-priority Pods to bypass green energy constraints and continue execution. It also reduces *QVT* (Equation 6) by aligning medium-priority Pods with off-peak hours and deferring non-critical Pods, maintaining energy-efficiency while ensuring service quality.

D. The proposed scheduling logic

Algorithm 3 demonstrates the logic behind the proposed scheduler. First, Kube-API receives a Pod creation request and forwards it to the scheduler (Line 1). The scheduler begins by running the *Energy-Aware Filter* Plugin (Algorithm 1), which generates a list of nodes meeting the Pod's requirements (Lines 2–6). This list is passed to the *NodeResourcesFit* plugin, which selects the node with the highest CPU allocation (Lines 7–11). The Pod request and selected node are then sent to the *Energy-Aware Scheduling* plugin (Algorithm 2), which evaluates energy, priority, and low electricity rate time conditions (Lines 12–13). If approved, the process moves to the bind stage, where Kubernetes instructs the target node to create the Pod, completing the scheduling. If conditions are not met, scheduling is delayed for 1 minute before restarting the procedure from the first plugin, repeating until the Pod is successfully bound to a node (Lines 14–16).

Algorithm 3 Overall Scheduling Logic

1: **Start:** Kube-API receives a Pod creation request
2: **procedure** ENERGY-AWARE FILTER PLUGIN
3: **if** success **then return** $suitable\_nodes[]$
4: **else Pending** Pod scheduling
5: **end if**
6: **end procedure**
7: **procedure** NODERESOURCESFIT
8: **if** success **then return** $\max(suitable\_nodes[], \text{CPU usage})$
9: **else Pending** Pod scheduling
10: **end if**
11: **end procedure**
12: **procedure** ENERGY-AWARE SCHEDULING PLUGIN
13: **if** success **then return Deploy** Pod on $target\_node$
14: **else Pending** 1 minute, then Restart Scheduling
15: **end if**
16: **end procedure**

The combined execution of Filter, Fit, and Scheduling plugins acts as a greedy heuristic strategy to find an approximate minimization of the objective function defined in Equation 8.

To conserve brown power and maximize green power usage, our system pauses medium- and low-priority Pods during green power shortages, potentially leaving servers idle. To address this, we developed a script that automatically manages Kubernetes Worker nodes, suspending or waking servers based on green power availability, ensuring power conservation when green power is low. Algorithm 4 summarizes the script's logic. Running continuously, it initializes a `CONSECUTIVE_CHECKS` counter to track instances when power demand exceeds green power (Lines 2–3).

The script retrieves green power and power consumption data, re-enables scheduling on nodes that are healthy but temporarily inactive (uncordons), and compares power availability to demand. If green power is low, `CONSECUTIVE_CHECKS` increments. If the counter reaches `THRESHOLD`, the script evicts workloads (drains) from and powers off the least-utilized node, reducing Pod reallocation while maintaining QoS. The `auto_run` script restarts migrated Pods (Lines 4–9).

If low power condition persists, servers remain turned off until only a sufficient number of servers remain active, supporting high- and medium-priority Pods. When green power is sufficient, the script re-enables scheduling on nodes (uncordons) and, if green power surplus exceeds a threshold (`WAKEUP_THRESHOLD`), wakes up additional servers (Lines 12–14). This threshold ensures enough green power is available to avoid frequent server on-off cycles.

This script ensures that unused compute nodes are turned off when green energy is insufficient, and only reactivated when surplus solar power is available, effectively contributing to the minimization of *BE* (Equation 5).

Algorithm 4 Server Auto Wake-up and Sleep Procedure

Input: `CHECK_INTERVAL`, `THRESHOLD`, `K8s_Wnodes[]`, `SHUTDOWN_ORDER[]`
Start: Initialize `CONSECUTIVE_CHECKS` to 0
while True **do**
  Retrieve GP and TP
  if $GP < TP$ **then**
    Increment `CONSECUTIVE_CHECKS`
    if `CONSECUTIVE_CHECKS` $\geq$ `THRESHOLD` **then**
      Drain and shutdown one node from `SHUTDOWN_ORDER[]`
      Reset `CONSECUTIVE_CHECKS`
    end if
  else
    Reset `CONSECUTIVE_CHECKS` and uncordon any disabled nodes
    if GP exceeds TP + `WAKEUP_THRESHOLD` **then**
      Wake up one **NotReady** or unreachable node in `K8s_Wnodes[]`
    end if
  end if
  Wait for `CHECK_INTERVAL` before next cycle
end while

VI. Performance Evaluations

For performance evaluation, we evaluate three baseline schedulers and three variants of our proposed GreenK8s scheduler under two weather conditions: *sunny* and *cloudy* days. Experiments are conducted on a real-world K8s cluster powered by a grid-tied solar system. We describe the setup and outline the methodology used to assess results.

A. Baselines

1) DD (Default K8s scheduler) [13]: It is the default Kubernetes scheduler that represents the standard scheduling behavior in Kubernetes.

2) KCSS [18]: It uses a multi-criteria decision analysis algorithm to rank candidate nodes according to CPU, memory, disk, energy consumption, etc. and selects the best node to execute the newly submitted container.

3) KEIDS (KEIDS-ES3) [17]: It is a green scheduling algorithm that minimizes carbon emissions and energy usage, while satisfying the container's resource requirements (CPU, memory, runtime) and replica constraints.

B. Proposed Schedulers

1) GSOP (Green scheduler without prediction): This is a variant of our proposed GreenK8s scheduler that makes scheduling decisions based on real-time green power (GP) availability, without using any solar prediction. The Auto Server Wake-up and Sleep script is enabled.

2) GSWP (Green scheduler with prediction): A variant of GreenK8s that schedules based on predicted green power (PGP) data. The Auto Server Wake-up and Sleep script is active. To explore the effect of prediction horizon on performance, we test lead times of 5, 15, 30, and 60 minutes under both sunny and cloudy conditions (8 runs total).

3) GSWPP (Green scheduler with perfect prediction): An oracle version of GreenK8s using actual future solar values instead of CNN-based predictions. It evaluates the impact of forecast errors comparing against GSWP. Experiments use the same four lead times and two weather conditions.

C. Metrics

1) Total energy usage: The system's instantaneous power consumption is recorded every 5 seconds via the *Data Collector* during the experiment and used to calculate the total energy usage. The calculation process is shown in Equation 3. Lower values indicate better performance.

2) Clean energy usage ratio: It is calculated based on the ratio of green energy to the total power consumption. Higher values indicate better performance.

3) Pod QoS violation time percentage: During periods of green energy shortage, Pods are compressed onto fewer nodes. QoS is considered violated when the actual CPU load generated by these running Pods exceeds the total number of vCPUs on a node. We record these overloads time and calculate their percentage relative to the time period during which Pods are compressed onto fewer nodes. A lower percentage indicates better QoS.

4) Pod suspension time: We measure the pending time of each pod before scheduling throughout the testing period and sum these to obtain the total pending time for the experiment. Lower values indicate better.

5) The absolute differences (TAD) in green power (GP) and total power (TP) consumption: The system records green power generation and cluster power consumption. The difference is calculated using the absolute difference between green power and total power consumption at each time point, calculated as follows (Lower values are better):

$$TAD = \sum_{i=1}^{n} |TP_i - GP_i| \cdot \Delta t \tag{19}$$

D. Experimental setup

1) Servers: We selected 9 Dell PowerEdge R710 servers for experiments. One server is used as the master node, seven as worker nodes, and one as the data collection node. Each server has dual Intel E5640 CPUs (8C16T), 48GB RAM, and 340W peak power.

2) CPU Workload: We use the Google Borg dataset [35], which provides CPU usage statistics including request values, average and maximum usage, as well as detailed usage percentiles (from 0% to 100%). To realistically emulate the workload behavior inside each Pod, we use Stress-NG [36] to

generate CPU load. Specifically, each Pod is associated with a cumulative distribution function (CDF) in the dataset. Stress-NG dynamically adjusts CPU pressure based on the Pod's corresponding CDF, ensuring that the runtime CPU utilization follows the statistical usage patterns observed in the Borg traces.

3) Pod Oversubscription Parameter Settings: Our workloads are based on the Google Borg dataset [35]. Considering the P_{over} distribution, the cluster's memory capacity, and initial experiments, we set $p\_max = 0.8$ and $p\_min = 0.2$. p_max higher 0.8 causes deployment failures due to memory limits, while p_min lower 0.2 leads to meaningless workload emulation.

4) Data Collection: The ePDU (Figure 1 monitoring tool) records green energy data and cluster power consumption every 5 seconds. The total power consumption of the master and worker nodes is used as the overall cluster power consumption (*TP*). Note that the data collection node is used solely for recording data; thus, its power consumption is excluded from the results.

5) Green Energy Peak Selection: Our experimental setup includes 18 solar panels with a total capacity of 6.6 kW, exceeding the power consumption of our server cluster. To reduce this discrepancy and ensure experimental validity, we normalized the solar power output. Based on [37], cluster power consumption is typically 64%–75% of the green energy supply. Testing showed that our K8s cluster's peak power consumption is approximately 1300 watts, so we scale the peak green energy generation to 1800 watts.

6) Solar data, experiment dates and duration: We randomly selected solar energy data of sunny days (31, Aug, 2024) and cloudy days (2, Sep, 2024) from the historical data of PV inverters for the experiment. Each experiment spanned a full 24-hour period, from 00:00 to 24:00. The experiment was accelerated by a factor of 4, simulating 24 hours in 6 hours. Data from 7:00 to 18:00, when solar energy is available and fluctuating, are visualized for effective comparisons, while calculations cover the entire 24-hour period. Each baseline described earlier was tested on both sunny and cloudy weather conditions, with more than 40 experiments performed in total.

E. Experimental Results

*1) **Prediction Model Accuracy Results**:* To determine the optimal sliding window size that minimizes prediction errors, we conducted multiple experiments and trained several models. The sliding window length was varied from 7 to 60 days, with results shown in Table II. Accuracy first drops and then improves as the window increases. The lowest Mean Squared Error (MSE) occurs at an 11-day window, while the lowest Mean Absolute Error (MAE) is observed at 12 and 14 days. We selected the 11-day window based on its minimal MSE, as this metric heavily penalizes large deviations, which is critical for solar radiation prediction and energy management.

*2) **Green scheduler with prediction comparison**:* The GSWP baseline uses predicted green power for scheduling decisions. To determine the optimal prediction interval, we tested forecast data for 5 minutes, 15 minutes, 30 minutes, and 1 hour ahead. As shown in Table III, the 15-minute-ahead forecast consistently achieves the highest green power share, solar utilization, and the lowest absolute difference, effectively aligning green power supply with consumption. We further calculate the difference in Green Energy Share, Brown Energy Share, Solar Utilization and TAD between GSWP and GSWPP (Table IV) to show the potential inaccuracies in these predictions. It can be observed that the overall TAD errors remain within a very small range, and the maximum deviation in other utilization-related metrics is only 7.8%. For subsequent baseline comparisons, we adopt the 15-minute-ahead forecast setting for GSWP.

TABLE II: MSE and MAE values for different sliding window days for the prediction method

Sliding Window Days	MSE	MAE
7	0.007617	0.045368
10	0.007283	0.041937
11	**0.007160**	0.042079
12	0.007392	**0.041434**
14	0.007374	0.041437
18	0.007763	0.043667
21	0.007763	0.043667
28	0.008234	0.044479
60	0.008623	0.045227

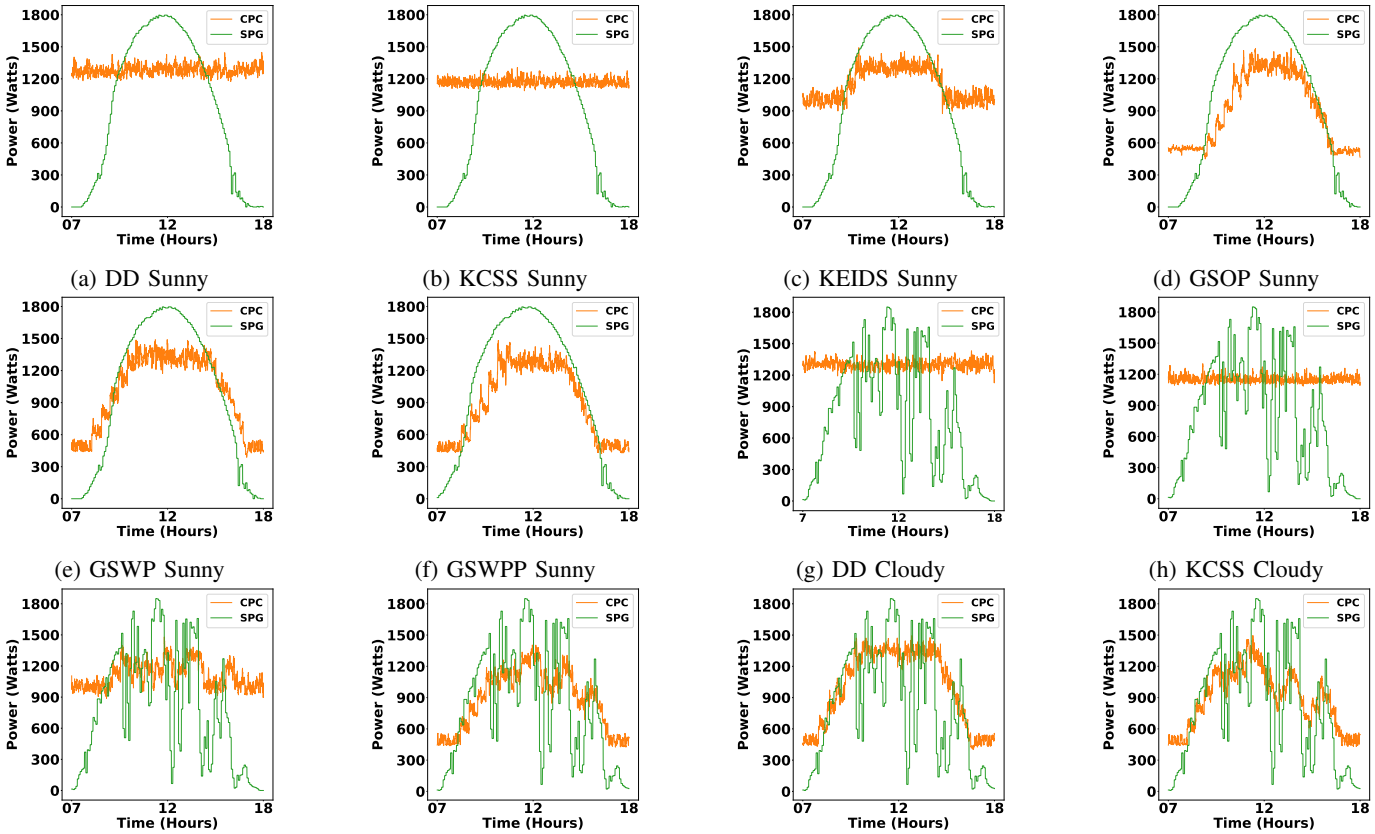
TABLE III: Comparison of energy consumption and solar energy utilization rate and TAD of GSWP at different forecast time intervals

Weather	Early Time	Green Energy Share (%)	Brown Energy Share (%)	Solar Utilization (%)	TAD (kWh)
Sunny	5 Min	53.16	46.84	84.79	2.2322
Sunny	**15 Min**	**54.12**	**45.88**	**84.94**	**2.1611**
Sunny	30 Min	53.97	46.03	84.70	2.1731
Sunny	1 Hour	51.91	48.09	82.79	2.3314
Cloudy	5 Min	46.27	53.73	88.87	2.3597
Cloudy	**15 Min**	**47.18**	**52.82**	**90.64**	**2.2876**
Cloudy	30 Min	46.62	53.38	89.53	2.3327
Cloudy	1 Hour	46.25	53.75	88.83	2.3616

TABLE IV: Differences between GSWP and GSWPP in Energy Shares, Solar Utilization, and TAD

Weather	Early Time	Δ Green Energy Share (%)	Δ Brown Energy Share (%)	Δ Solar Utilization (%)	Δ TAD (kWh)
Sunny	5 Min	0.33	-0.33	5.10	-0.0379
Sunny	15 Min	1.27	-1.27	7.81	-0.1113
Sunny	30 Min	1.72	-1.72	6.61	-0.1425
Sunny	1 Hour	0.67	-0.67	5.81	-0.0587
Cloudy	5 Min	-0.84	0.84	3.74	0.0797
Cloudy	15 Min	0.16	-0.16	6.84	0.0043
Cloudy	30 Min	-0.30	0.30	5.25	0.0404
Cloudy	1 Hour	0.94	-0.94	7.25	-0.1814

*3) **Evaluation of Scheduler Performance on Sunny and Cloudy Days**:* Figure 3 presents detailed power consumption



(a) DD Sunny (b) KCSS Sunny (c) KEIDS Sunny (d) GSOP Sunny

(e) GSWP Sunny (f) GSWPP Sunny (g) DD Cloudy (h) KCSS Cloudy

(i) KEIDS Cloudy (j) GSOP Cloudy (k) GSWP Cloudy (l) GSWPP Cloudy

Fig. 3: Cluster Total Power Consumption (CPC) vs Solar Power Generation(SPG) on Sunny and Cloudy Days

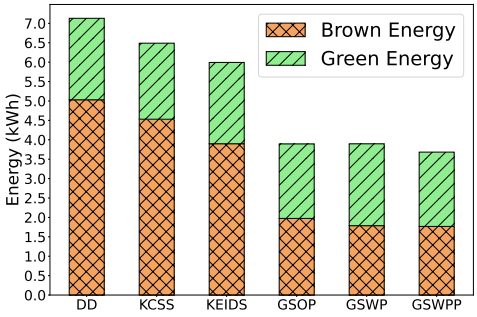


Fig. 4: All methods' energy consumption on sunny days

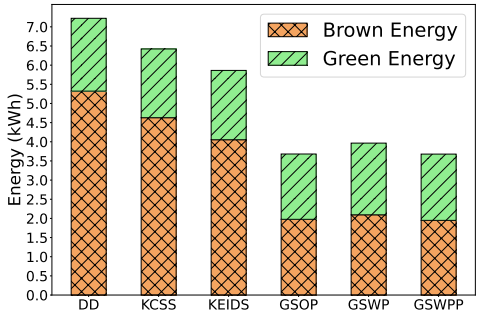


Fig. 5: All methods' energy consumption on cloudy days

of each baselines and our scheduler. Figure 4 and Figure 5 show that no matter it was sunny day (S) or cloudy day (C), our GSWP scheduler significantly reduces power consumption compared to the DD, achieving 45.37% on sunny days and 45.10% on cloudy days. At the same time, although KCSS and KEIDS also achieve certain energy savings compared to DD, our GSWP scheduler introduces a Pod Oversubscription strategy, which enables the system to pack more Pods onto fewer nodes when green energy is limited, allowing more compute nodes to be turned off to save energy. In addition, GSWP incorporates priority scheduling, ensuring that critical service Pods continue to run without interruption, while less critical workloads are deferred. This policy enables fine-grained control over resource usage and helps to enable energy-saving measures such as server hibernation. Therefore, compared with the KCSS and KEIDS benchmarks, GSWP reduces power consumption by 40% and 35.00% on sunny days, and by 38.27% and 32.39% on cloudy days, respectively. Compared with DD, KCSS, and KEIDS as a whole, the average reduction is 39.34%.

Compared to GSOP and GSWPP, GSWP improves Solar Utilization (%) by 4.82%, 2.12% on sunny days and 0.89%,

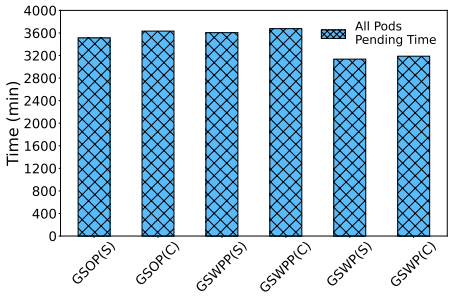


Fig. 6: Total pending time of all Pods

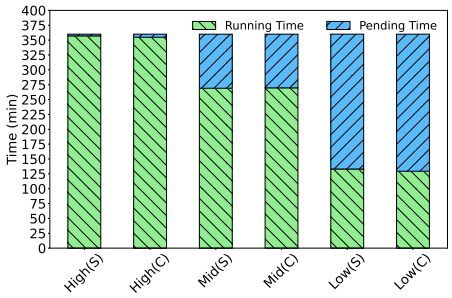


Fig. 7: Avg. pending time of diff. priorities

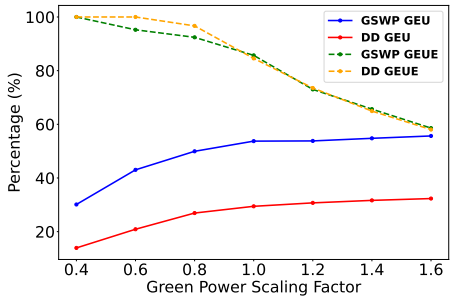


Fig. 8: Diff. Solar Power factor result

0.17% on cloudy days, with an average increase of 2%. Additionally, GSWP achieves the highest solar energy utilization rate and the second-lowest Total Absolute Difference (TAD), only behind GSWPP (which assumes perfect prediction). This is because the predicted solar power data produced by our CNN model is smoother and less volatile than the actual values used in GSWPP, especially during cloudy periods with sharp fluctuations. While GSWPP assumes access to perfect future data (i.e., actual solar values 15 minutes ahead), those real values still suffer from abrupt changes due to cloud cover. In contrast, our prediction model generalizes these variations, making the solar power signal more stable and thus helping the scheduler make steadier decisions.

*4) **Pod QoS violation percentage and Pod pending time**:* In this section, we present QoS-related metrics including Pod QoS violation percentage and Pod suspension time. Since DD, KCSS and KEIDS baselines do not encounter QoS issues, we focus on comparing GSOP and GSWP baselines, as well as variations in prediction times within the GSWP baseline.

Figure 6 shows that, over a 24-hour experiment, our GSWP method's total pod pending time is significantly lower than GSOP and GSWPP. The average reduction was 888.5 minutes. This reduction is attributed to leveraging predicted solar power data. It helps smooth solar power curves, avoids instability during sharp fluctuations, and thereby decreases pending time, particularly on cloudy days. Additionally, we observed that with 15-minute-ahead predictions, the QoS violation rates drop to 20.34% on cloudy days, compared to 17.41% on sunny days, with overall low violation levels.

At the same time, we calculated the waiting time of Pods of different priorities in the 24-hour experiment in the GSWP method, as shown in Figure 7, and the results were consistent with expectations. High-priority Pods have the shortest pending time, with an average waiting time of 4 minutes under different weather conditions. Medium-priority Pods have a moderate suspension time, with an average of 90.6 minutes. Due to green energy constraints, low-priority Pods have the longest suspension time, with an average of 230 minutes.

*5) **Evaluating Scheduler Performance Under Varying Solar Energy Supply Scales**:* To evaluate scheduler adaptability under different green energy conditions, we scaled solar power generation using factors (0.4 to 1.6) under sunny weather, simulating under- and over-provisioning scenarios. Figure 8 shows that as the scaling factor increases, GSWP's GEU (green energy share in total energy) steadily improves, peaking at 55.66% with a factor of 1.6. Under low supply (factors 0.4, 0.6), GSWP achieves near-perfect GEUE (green energy efficiency), at 100% and 95.23%, demonstrating strong utilization of limited solar power. In contrast, the Default scheduler maintains 100% efficiency but contributes only 13.93% and 20.89% to total energy, respectively. When scaling exceeds 1.0, GSWP's GEUE drops gradually (e.g., 58.59% at 1.6), yet remains more stable than Default (58.04%). These results demonstrate GSWP's robustness in adapting to both scarce and surplus green energy scenarios.

VII. Conclusions and Future Work

In this paper, we proposed GreenK8s, a green scheduler for Kubernetes that significantly reduces power consumption compared to the default Kubernetes scheduler while maintaining high QoS levels. Our approach minimizes brown energy use and QoS violations in a grid-tied Kubernetes cluster, prioritizing workloads with different importance levels. GreenK8s achieves an average energy savings of 39.34% and increases the average share of green energy in total consumption to 50.65%, demonstrating adaptability across varying weather conditions. By leveraging solar power forecasts, the scheduler improves solar energy utilization compared to non-predictive methods, optimizing renewable energy use and reducing brown power reliance in the cluster. Future work includes extending GreenK8s to large-scale cluster deployments for deeper scalability analysis, alongside handling forecast inaccuracies and developing a reinforcement learning scheduler to adapt to dynamic energy and workload conditions.

Code Availability: The source code is publicly available at https://github.com/disnetlab/GreenKubeScheduler.

VIII. Acknowledgements

This work was partially supported by Monash University and the University of Melbourne, whose support is gratefully acknowledged. Minxian Xu is also supported by Guangdong Basic and Applied Basic Research Foundation (No. 2024A1515010251, 2023B1515130002), Key Research and Development and Technology Transfer Program of Inner Mongolia Autonomous Region (2025YFHH0110) and Shenzhen Basic Research Program under grant JCYJ20240813155810014.

REFERENCES

[1] A. Katal, S. Dahiya, and T. Choudhury, "Energy efficiency in cloud computing data centers: a survey on software technologies," *Cluster Computing*, vol. 26, no. 3, pp. 1845–1875, 2023.

[2] J. Shuja, S. A. Madani, K. Bilal, K. Hayat, S. U. Khan, and S. Sarwar, "Energy-efficient data centers," *Computing*, vol. 94, no. 12, pp. 973–994, 2012.

[3] M. Koot and F. Wijnhoven, "Usage impact on data center electricity needs: A system dynamic forecasting model," *Applied Energy*, 2021.

[4] M. A. B. Siddik, A. Shehabi, and L. Marston, "The environmental footprint of data centers in the united states," *Environmental Research Letters*, vol. 16, no. 6, p. 064017, 2021.

[5] H. Rong, H. Zhang, S. Xiao, C. Li, and C. Hu, "Optimizing energy consumption for data centers," *Renewable and Sustainable Energy Reviews*, vol. 58, pp. 674–691, 2016.

[6] S. Ghazanfari-Rad and S. Ebneyousef, "A survey of renewable energy approaches in cloud data centers," in *2023 8th International Conference on Technology and Energy Management (ICTEM)*, 2023, pp. 1–7.

[7] Amazon, "Amazon meets 100% renewable energy goal seven years early," https://www.aboutamazon.com.au/news/sustainability/amazon-meets-100-renewable-energy-goal-seven-years-early, 2024, accessed: 2024-07-18.

[8] Y. Liang, P. Li, W. Su, W. Li, and W. Xu, "Development of green data center by configuring photovoltaic power generation and compressed air energy storage systems," *Energy*, vol. 292, p. 130516, 2024.

[9] A. Tarafdar, M. Debnath, S. Khatua, and R. K. Das, "Energy and makespan aware scheduling of deadline sensitive tasks in the cloud environment," *Journal of Grid Computing*, vol. 19, pp. 1–25, 2021.

[10] 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.

[11] Kubernetes, "Kubernetes overview documentation," https://kubernetes.io/docs/concepts/overview/, 2024, accessed: 2024-09-14.

[12] H. Saito, H.-C. C. Lee, and C.-Y. Wu, *DevOps with Kubernetes: accelerating software delivery with container orchestrators*. Packt Publishing Ltd, 2019.

[13] Kubernetes, "Kube scheduler documentation," https://kubernetes.io/docs/reference/command-line-tools-reference/kube-scheduler/, 2024, accessed: 2024-09-14.

[14] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Comput. Surv.*, vol. 55, no. 7, Dec. 2022. [Online]. Available: https://doi.org/10.1145/3539606

[15] E. Nicholson, K. E. Watermeyer, J. A. Rowland, C. Sato, S. L. Stevenson, A. Andrade, T. Brooks, N. Burgess, S.-T. Cheng, H. Grantham, S. Hill, D. Keith, M. Maron, D. Metzke, N. Murray, C. R. Nelson, D. Obura, A. Plumptre, A. Skowno, and J. Watson, "Scientific foundations for an ecosystem goal, milestones and indicators for the post-2020 global biodiversity framework," *Nature Ecology Evolution*, vol. 5, pp. 1338 – 1349, 2021.

[16] Australian Bureau of Meteorology, "Australian hourly solar irradiance gridded data," http://www.bom.gov.au/climate/how/IDCJAD0111.shtml, 2022, accessed: 2024-09-14.

[17] K. Kaur, S. Garg, G. Kaddoum, S. H. Ahmed, and M. Atiquzzaman, "Keids: Kubernetes-based energy and interference driven scheduler for industrial iot in edge-cloud ecosystem," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4228–4237, 2019.

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

[19] T. Piontek, K. Haghshenas, and M. Aiello, "Carbon emission-aware job scheduling for kubernetes deployments," *The Journal of Supercomputing*, vol. 80, no. 1, pp. 549–569, 2024.

[20] I. Syrigos, D. Kefalas, N. Makris, and T. Korakis, "Eelas: Energy efficient and latency aware scheduling of cloud-native ml workloads," in *2023 15th International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 2023, pp. 819–824.

[21] G. Rattihalli, N. Hogade, A. Dhakal, E. Frachtenberg, R. P. H. Enriquez, P. Bruel, A. Mishra, and D. Milojicic, "Fine-grained heterogeneous execution framework with energy aware scheduling," in *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*. IEEE, 2023, pp. 35–44.

[22] M. Chadha, T. Subramanian, E. Arima, M. Gerndt, M. Schulz, and O. Abboud, "Greencourier: Carbon-aware scheduling for serverless functions," in *Proceedings of the 9th International Workshop on Serverless Computing*, 2023, pp. 18–23.

[23] D. Jorge-Martinez, S. A. Butt, E. M. Onyema, C. Chakraborty, Q. Shaheen, E. De-La-Hoz-Franco, and P. Ariza-Colpas, "Artificial intelligence-based kubernetes container for scheduling nodes of energy composition," *International Journal of System Assurance Engineering and Management*, pp. 1–9, 2021.

[24] Z. Wu, Y. Deng, Y. Zhou, J. Li, and S. Pang, "Faasbatch: Enhancing the efficiency of serverless computing by batching and expanding functions," in *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, 2023, pp. 372–382.

[25] H. Cai, Q. Cao, H. Jiang, and Q. Wang, "Greenhetero: Adaptive power allocation for heterogeneous green datacenters," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, 2021, pp. 160–170.

[26] T. Chakraborty, A. N. Toosi, and C. Kopp, "Elastic power utilization in sustainable micro cloud data centers," *IEEE Transactions on Sustainable Computing*, vol. 8, no. 3, pp. 465–478, 2023.

[27] J. Serenari, S. Sreekumar, K. Zhao, S. Sarkar, and S. Lee, " GreenWhisk: Emission-Aware Computing for Serverless Platform ," pp. 44–54, Sep. 2024. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/IC2E61754.2024.00012

[28] Y. S. Patel and P. Townend, "A stable matching approach to energy efficient and sustainable serverless scheduling for the green cloud continuum," in *2024 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 2024, pp. 25–35.

[29] J. Liu, S. Wang, A. Zhou, J. Xu, and F. Yang, "Sla-driven container consolidation with usage prediction for green cloud computing," *Frontiers of Computer Science*, vol. 14, pp. 42–52, 2020.

[30] Z. Ding, S. Wang, and C. Jiang, "Kubernetes-oriented microservice placement with dynamic resource allocation," *IEEE Transactions on Cloud Computing*, vol. 11, no. 2, pp. 1777–1793, 2022.

[31] C. Song, M. Xu, K. Ye, H. Wu, S. S. Gill, R. Buyya, and C. Xu, "Chainsformer: A chain latency-aware resource provisioning approach for microservices cluster," in *International Conference on Service-Oriented Computing*. Springer, 2023, pp. 197–211.

[32] X. Zhang, L. Li, Y. Wang, E. Chen, and L. Shou, "Zeus: Improving resource efficiency via workload colocation for massive kubernetes clusters," *IEEE Access*, vol. 9, pp. 105 192–105 204, 2021.

[33] X. Zhang, X. Wang, Y. Liu, and Z. Deng, "An efficient scheduling strategy for containers based on kubernetes," in *International Conference on Collaborative Computing: Networking, Applications and Worksharing*. Springer, 2022, pp. 326–342.

[34] SNMP Research International, Inc., "Simple network management protocol (snmp)," 2024, accessed: 2025-04-24. [Online]. Available: https://www.snmp.com/protocol/

[35] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[36] C. I. King, "stress-ng: a tool to stress a computer system in various selectable ways," https://github.com/ColinIanKing/stress-ng, 2024, accessed: 2024-04-25.

[37] R. Bianchini, Í. Goiri, K. Le, and T. D. Nguyen, "Parasol: A solar-powered μdatacenter," 2012.