National College of Ireland

KubeGreen: A Scheduler for Latency, CPU, and Carbon Optimisation

MSc Research Project
Cloud Computing

Saksham Shailesh Gurbhele
Student ID: 23266724

School of Computing
National College of Ireland

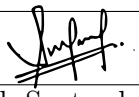
Supervisor: Prof. Diego Lugones

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Saksham Shailesh Gurbhele
Student ID:	23266724
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Prof. Diego Lugones
Submission Due Date:	11/08/2025
Project Title:	KubeGreen: A Scheduler for Latency, CPU, and Carbon Optimisation
Word Count:	6538
Page Count:	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	15th September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

KubeGreen: A Scheduler for Latency, CPU, and Carbon Optimisation

Saksham Shailesh Gurbhele
23266724

Abstract

As adoption of federated Kubernetes for managing applications across distributed environments increases, a challenge emerges: default schedulers lack the intelligence to optime for both performance and environmental sustainability. This research addresses this gap by developing **KubeGreen**, a smart, multi-objective scheduler that reduces latency and carbon emissions. The proposed scheduler is a lightweight, heuristic-based scoring model that evaluates the suitability of each cluster based on three metrics: real-time network latency, carbon intensity, and real-time CPU utilisation.

KubeGreen was implemented and evaluated in a real multi-cluster Kubernetes testbed spanning three different Google Cloud Platform regions. A continuous 24-hour empirical evaluation demonstrated its effectiveness. Compared to static placement strategies, KubeGreen dynamically adapts to fluctuating conditions, achieving an average reduction of **15%** in carbon emissions and **22%** in network latency, while maintaining balanced CPU usage across clusters. This work validates that a practical, metric-driven approach can significantly enhance both the performance and sustainability of cloud native workloads, offering a path toward greener and more efficient multi-cluster orchestration. Future work can explore the integration of additional metrics such as cost or network bandwidth to further refine scheduling decisions.

1 Introduction

The cloud-native technologies changed the paradigm of how modern applications have been deployed and orchestrated to the point where Kubernetes is the industry-standard container orchestration platform Solo.io (2024). With organizations continuing to move to more multi-cloud and hybrid-cloud approachesHorizons (2024), the need to orchestrate an application across not only one cluster, but a distributed, federated cluster environment, is expanding. Central control of geographically distributed clusters is possible with federated Kubernetes, which provides greater resiliency, failover and scale to the clusters. Yet, the default Kubernetes scheduler has a limited scope that focuses on single-cluster deployments and does not take account of inter-cluster optimisation shortcomings. It is also devoid of awareness of service -level agreement (SLA) requirements or sustainability objectives that are very important in federated deployments.

In the meantime, the environmental impact of cloud computing has become a growing concern. Currently, data centers globally accounts about one percent of power consumption and greenhouse-gas emissions and that could increase with the demand in compute.

There is still high power mix dependence in a data center despite cloud providers using renewable energy and bettering the efficiency of infrastructureLavi (2025). Carbon intensity refers to the number of grams of CO_2 produced per kilowatt-hour (gCO_2/kWh) of electricity consumption, which changes significantly over space and time of day depending on the ratio between fossil-fuel and renewable electricity sources EnergyTag (2024). The ability to vary this leads to the possibility of minimising emissions by scheduling data centre workloads in regions with cleaner energy sources to times when there is a surplus of renewable generation Foundation (2022).

The existing schedulers in Kubernetes today cannot add environmental metrics like carbon intensity or electricity mix to the scheduling decision Foundation (2022). Meanwhile, SLA constraints, e.g. response time, throughput or uptime guarantees, are usually managed at higher abstraction levels and are not embedded deep in the core scheduling logic Javed et al. (2023). This absence of SLA-awareness and sustainability integration comes with inefficiencies and operational risk, particularly where latency sensitivity or a mission-critical application is involved. Previous work has mainly concerned optimising resource utilisation or price (e.g., bin packing, node affinity), but little effort has been put to the simultaneous goal of SLA compliance and minimising emissions Javed et al. (2023); Pham et al. (2025).

The main problem to be investigated in this study is the following: is it possible to create a custom workload scheduler on multi-cluster Kubernetes that will reduce the impact on performance that is caused by latencies and reduce carbon emissions at the same time? This relates to designing a scheduler using performance indicators, including low latency and not using overutilized resources as well as the sustainability profiles of the underlying infrastructure. To address this issue a heuristic-based scheduling model, with a multi-objective scoring function, which weighs clusters relative to real-time latency and CPU load was developed.

The research goals will be as follows: explore the existing workload scheduling mechanisms in multi-cluster systems and sustainability-aware computing; develop a smart scheduler that integrates performance-related metrics (latency and CPU load) and carbon intensity data; build a prototype of the scheduler in a real multi-cluster Kubernetes setting based on Google cloud clusters; and evaluate KubeGreen's performance with the region static schedulers(Madrid, Frankfurt and London) with measures such as the amount of CO2 per workload, latency, and CPU utilization. Data related to carbon intensity is sourced from ElectricityMap Corradi (2024), locally cached, to avoid excessive API requests and improve resilience, and updated every hour, and the latency and CPU utilization metrics are offered by workload monitoring based on Prometheus.

There will be definite limitations to the evaluation that must be imposed. The carbon intensity information has been obtained automatically through API calls using ElectricityMap and the requests have been sent at a frequency of one hour to minimize the costs of requesting the API. The data retrieved would be saved locally in an update file which served as a cache to support further scheduling decisions in the same hour. Energy consumption was profiled in the standard formula regarding the CPU consumption and time executing, and the runtime records were gathered through Prometheus. In contrast to the studies based on simulations, to model real network conditions and introduce variability in infrastructure, a multi-cluster Kubernetes environment in the present project was launched on real clusters in three Google Cloud regions (London, Madrid, and Frankfurt).

This study offers a contribution to the emerging cloud green field. This can contribute to a change in future designs of intelligent orchestrators, which address not only the work-

load responsiveness and resource efficiency needs but are also environmentally aligned by providing a feasible way of integrating sustainability and performance-awareness in Kubernetes scheduling layer. Making this work will help reduce operational CO2 emissions, and also enhance the terms under which work is performed in multi-cluster environments to further make stable and more sustainable cloud-native environments.

The remainder of this report is organised as follows. The Abstract provides a concise summary of the problem, approach, results and contributions. The current Introduction sets the context, motivation, research question and contributions. Related Work surveys existing carbon- and performance-aware schedulers and clarifies how our work differs. Methodology details the experimental design, datasets and evaluation metrics. Design Specification formalises the weighted scoring model and system architecture. Implementation explains the prototype integration into Kubernetes Federation. Evaluation presents quantitative results and sensitivity analyses. Finally, the Conclusion summarises findings, discusses limitations and outlines avenues for future work, followed by the References cited throughout the paper.

2 Related Work

2.1 Introduction to Kubernetes Scheduling in Federated Environments

The development of cloud computing has resulted in a more intricate deployment model and other organizations are implementing multi-cloud and federated designs to gain resilience, cost optimisation, and compliance with regulations. Kubernetes has become a de facto for container orchestrators, but the default scheduler is best suited to a homogeneous environment that makes use of only one cluster Tamiru et al. (2021). Such a constraint is especially strong in federated multi-cloud environments where heterogeneity of resources exists across geographical areas with different attributes in terms of costs, characteristics, and energy trends. This literature review analyses existing literature on workload scheduling in Kubernetes in the context of federated multi-cloud environments and considers workload schedulers that take into account multiple goals (both performance ones, e.g., latency targets, and sustainability ones). The systematic review comprises 4 main areas (1) federated Kubernetes orchestration architectures, (2) latency-aware and SLA-compliant scheduling, (3) energy and sustainability-designed scheduling, and (4) machine learning techniques in adaptive scheduling. With such a detailed study I were able to find research gaps in this work and possible directions of future research combining approaches to integrated scheduling to address the usually mutually incompatible demands of performance, compliance and sustainability.

2.2 Federated and Multi-Cloud Kubernetes Orchestration

The challenges that federated Kubernetes deployments introduce to workload scheduling are due to the heterogeneity of federated Kubernetes, their geographical distribution and varied resource attributes. A number of schemes are suggested to cover these challenges, however, each has a different perspective on the management of resources and the allocation of work. Tamiru et al. (2021) have proposed mck8s as a dedicated orchestration platform specifically geared towards geo-distributed multi-cluster workloads. This system

reduces an MTFR system since they continuously deploy and scale applications in distributed Kubernetes clustering automatically. The authors showed that there was a considerable reduction in pod provisioning as opposed to the native Kubernetes schedulers. One thing that sets mck8s apart is its dynamic ability to adapt to cluster heterogeneity; thus, it works mainly in the context of federated environments. Nevertheless, there lies a significant weakness in the fact that it explicitly does not accommodate performance measures such as latency constraints or energy efficiency measures when optimizing resource allocations. Along the lines of multi-region scheduling issues, Furnadzhiev et al. (2025) had introduced an extended scheduling scheme involving network-aware plugins. They solution is explicit in everything about interregion latency optimisation using network topology constraints. The mck8s focuses specifically on the placement of batch workloads (the most common form of Apache Spark jobs), while taking the issue of latency into consideration. Heuristics and metaheuristics approaches were used by the authors to overcome the complex tradeoffs regarding the method of data locality with respect to resource utilization. Although, these developments exist, the framework fails to incorporate the sustainability goals to its decision-making process like minimizing the consumption of energy. Most recently, Zambianco et al. (2025) suggested a disruption-aware microservice re-orchestration policies that is designed to be cost-effective in multi-cloud deployments. Their strategy fills the gap in recognition of the dynamism of cloud structures by taking into consideration any form of disruption and its effects on service sustainability. Although optimisation of costs is indirectly linked to sustainability due to efficiency of resources, the system does not have direct means of optimising energy and reducing carbon footprint.

The overview of the work shows that one of the major gaps related to federated Kubernetes orchestration is that most of the current frameworks ensure powerful approaches to resource optimisation and latency management in multi-cluster environments, yet, they do not overlap much with sustainability. Such a compartmental approach does not respond to the rapidly growing need to achieve energy-efficient computing across distributed settings.

2.3 SLA and Latency-Aware Scheduling in Geo-Distributed Systems

Geo-distributed environments are difficult to satisfy SLA requirements, especially the latency criteria because of the network delays associated with regions. To tackle such constraints in Kubernetes deployments, a number of research systems have developed scheduling mechanisms that consider such policies. LAIS reduces end-to-end delays by leaving the choice on whether latency requirements are to be strictly enforced or opportunistically optimised, to the user. Such an approach is suitable for the increasing demands of customizable latency and performance parameters in the cloud implementation. Nevertheless, like LAIS, NeiLatS presents a one-dimensional interest in latency optimisation but does not address energy efficiency and broader sustainability objectives, and is therefore of limited use in environmentally-sensitive deployments. The peculiarity of NeiLatS is providing network stability predictions and expected communication conditions in scheduling decisions. The framework uses an efficacy coefficient technique to trade off in use of multi-resources with latency needs. Although NeiLatS is effective in a dynamic context such as edge computing environments, the solution suffers from the same limitations as other state-of-the-art solutions in that it enforces latency require-

ments independently of energy requirements or pricing limits that are typical of federated multi-cloud deployments. Through their work, it is easy to understand how intelligent scheduling could lessen the response latency and enhance the harnessing of the resources. Li et al. (2023) demonstrated that learning-based methods were able to substantially outperform static scheduling policies, especially in dynamic systems of different workload patterns. Nevertheless, they, like other papers in this segment, do not specify sustainability metrics in the process of optimisation.
An evaluation of these contributions highlights the typical limitation present across the available research to date: although there have been successful methods established to address the phenomenon of latency in geo-distributed Kubernetes set-ups, these initiatives are rather single-objective optimisations that do not take into consideration the energy utilization or carbon usage at all. this gap highlights the need for the development of multi-objective scheduling models, which may meet latency demands and minimize the environmental footprint at the same time.

2.4 Energy and Sustainability-Focused Scheduling

The sustainability concept in the management of cloud infrastructure has become very vital as the data centers are drawing more and more power out of the global electricity supply. In this section, I critically analyse the studies that deal specifically with energy-efficient scheduling within Kubernetes and other container orchestration systems. Syrigos et al. (2023) proposed Energy Efficient and Latency Aware Scheduling (EELAS) framework of the cloud-native ML workloads. EELAS uses heuristic and integer linear programming to smartly map the location of latency-sensitive machine learning inference tasks over fog-edge-cloud continuum infrastructures. The importance of the framework is that it incorporates latency targets and energy optimisation objectives which is one of the only few solutions that deals with both issues at once. Nevertheless, the experimental activity of EELAS is largely restricts to vertical edge systems as opposed to multi-cloud federated systems (horizontally distributed), which restricted the immediate applicability to large-scale Kubernetes systems and systems federated with it. In another light, Zhong and Buyya (2020) proposed an economically effective container orchestration plan under the Kubernetes-based heterogeneous resources infrastructures. Their strategy brings on board autoscaling and intelligent relocating or reallocating of not-fully utilized VM instances so that energy costs can be kept at a minimum reducing the operation costs. In contrast to EELAS, the strategy focuses on the elastic/federated environments and pays attention to the real-world deployment constraints which improves its generality. Nonetheless, this model sacrifices performance objectives or geolatency optimisation in favor of cost-effectiveness, which presents a possible conflict in the setting of strong performance demands. They work on a balance between load distribution and minimum energy consumptions by a new optimisation algorithm. Although their method can be promising in energy reduction, they do not have mechanisms to integrate user-defined SLA requirements into the scheduling decision. Such a restriction highlights the necessity of having scheduling policies that take into account both of these performance measures in a combined manner (latency and availability) and energy consumption in geographically diverse cluster deployments.

This literature review has discussed the latest work on workload scheduling in Kubernetes, considering federated multi-cloud environments, the performance-related criteria

(e.g., latency) and sustainability. Although great strides have been achieved in addressing each of these areas on their own, there continues to be an important shortcoming in solutions that unify these issues into an integrated scheduling system. Formulation of scheduling algorithms capable of hitting performance goals but optimal with respect to sustainability measures is one of the areas that future researchers can look into. These can take advantage of heuristic optimisation algorithms to search for the trade balance that exists in the multi-objective scheduling problem. The pressure on organizations to migrate towards federated multi-cloud deployment and and increasing demands to lessen their environmental impact will further intensify the necessity of this need.

3 Methodology

In this section, the methodology of the research used in the development and evaluation of a smart performance and sustainability-aware scheduler of multi-cluster Kubernetes will be explained. The methodology is a realistic implementation through the use of real infrastructure, metric-based decisions, and empirical evaluation .

3.1 Experimental Setup

The workload was put into service in various geographically separate Kubernetes clusters that were setup in London, Frankfurt and Madrid on Google Cloud Platform (GCP). The GKE enabled one cluster to be deployed at a time and background named kubectl contexts to communicate with the test environment. In each cluster Prometheus and Node Exporter was installed to gather real-time information about the CPU usage which was then queried directly by the scheduler. The latencies were measured at scheduler host to Prometheus endpoint of each cluster. The carbon intensity carbon.json data, was stored in a file on the hourly basis of each region using the ElectricityMaps API. The scheduler had considered this three metrics, namely latency, CPU percentage, and carbon intensity to generate a weighted percentage as a score per cluster and decide upon an optimal cluster to place the job.

3.2 Smart Scheduler Design and Metrics

To make scheduling decisions, a custom smart scheduler was developed as an external controller. This scheduler incorporates a multi-objective weighted scoring formula, which evaluates the suitability of each cluster to new workloads. The ruling is made on the basis of three fundamental measures:

- **Network latency** from the client to the cluster (measured externally from the test client)
- **CPU utilization** of the cluster at the time of scheduling (collected in real-time from Prometheus and Node Exporter)
- **Carbon intensity** of the region where the cluster is hosted (retrieved hourly from the ElectricityMap API and stored locally in carbon.json as a cache to reduce API calls and ensure stability)

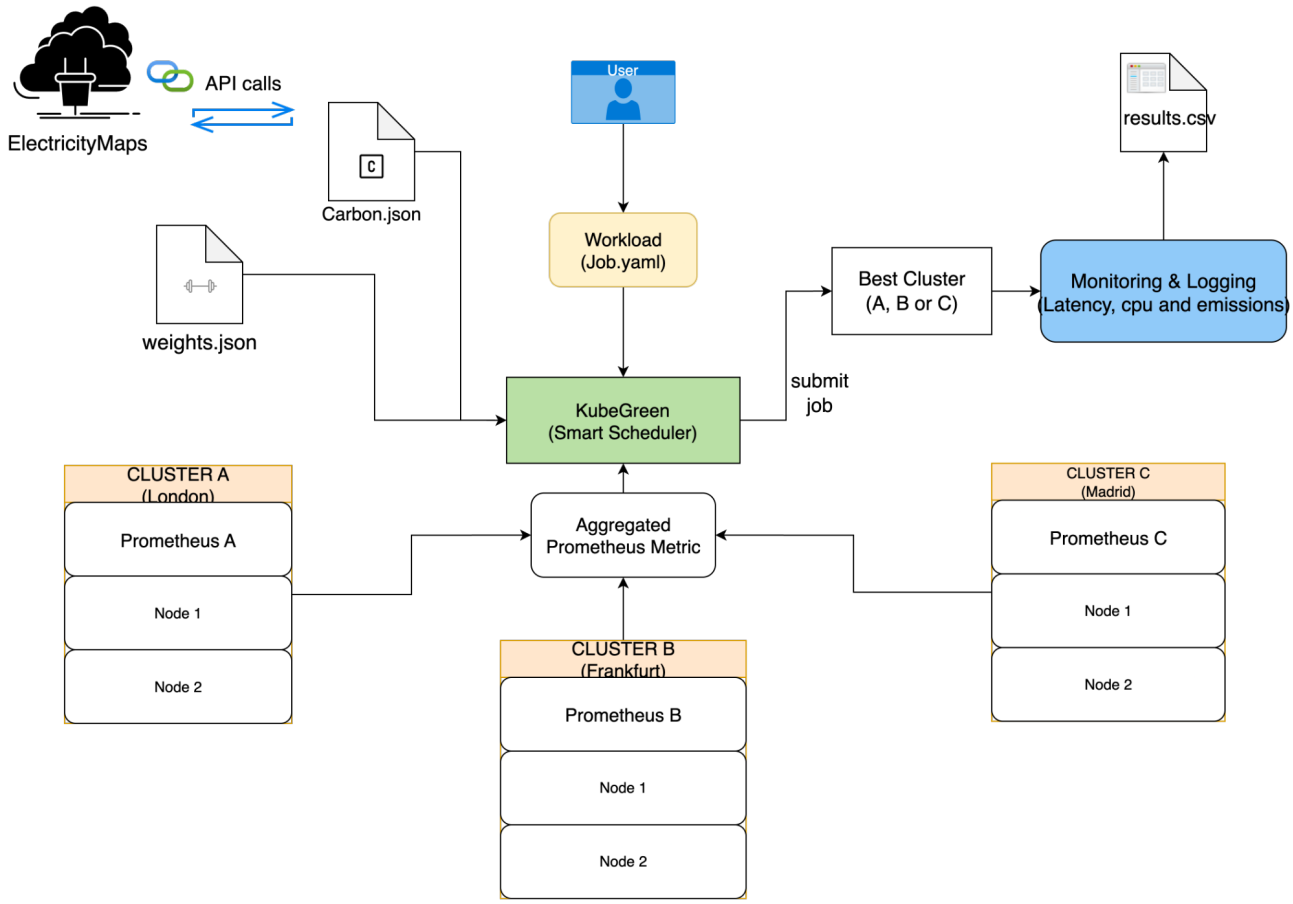


Figure 1: Architectural Diagram of the KubeGreen Scheduler

Unlike other approaches that require frequent external methods to obtain the carbon data, the project makes use of a locally stored `carbon.json` file which is refreshed after each hour with real-time carbon intensity values (in gCO_2/kWh) of each region covered by the clusters. Such a hybrid solution guarantees access to near real-time data on sustainability and prevents the reliability and cost concerns of a live API call during time scheduling. The updated file will present a consistent, reproducible input that the scheduler can utilize in coming up with informed decisions based on the latest environmental conditions.

The final score for each cluster is calculated using raw metric values in the following weighted formula: 1:

$$\text{score} = (\alpha \cdot \text{latency}) + (\beta \cdot \text{carbon}) + (\gamma \cdot \text{cpu}) \tag{1}$$

Where:

- α is the weight for latency — representing the importance of maintaining low latency and responsive performance.
- β is the weight for carbon intensity — prioritizing sustainability in cluster selection.
- γ is the weight for CPU usage — helping avoid overloading clusters or violating performance bounds.

As it is, the weights were summed to 1, as expressed in Equation 2, ensuring a balanced distribution of influence. The workload is placed on the cluster that provides the lowest score. such that there is fair divide of power. The scheduler directs the workload on the cluster that has the lowest composite score.

$$\alpha + \beta + \gamma = 1 \tag{2}$$

3.3 Evaluation and Weight optimisation

The empirical assessment approach was used to ascertain the most optimal combination of weight. There theoretically exists a set of 66 unique arrangements (in 0.1 step variations) of α, β and γ which will meet the constraint in Equation 2. Those that included extreme values (e.g. less than 0.2 or greater than 0.6) were removed in order to avoid unbalanced decision making resulting in a total of 15 combinations that were considered meaningful.

In order to read the trade-offs among these combinations a table 1 of data was created by recording results (latency, CPU, and carbon emissions) of each of the combinations tested. To visualize it each combination had to be labelled according to the $\alpha\beta\gamma$ values (e.g., `0.3-0.2-0.5`). The measures, such as score, latency, CPU usage, and emissions were then summarized through combination and calculated means.

Because the data uses different units and scales (the latency measured in milliseconds and emissions measured in grams), it was not efficient to visually compare the raw data. Hence Min-Max Normalization was applied only for visualization using the formula in Equation 3:

$$X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \tag{3}$$

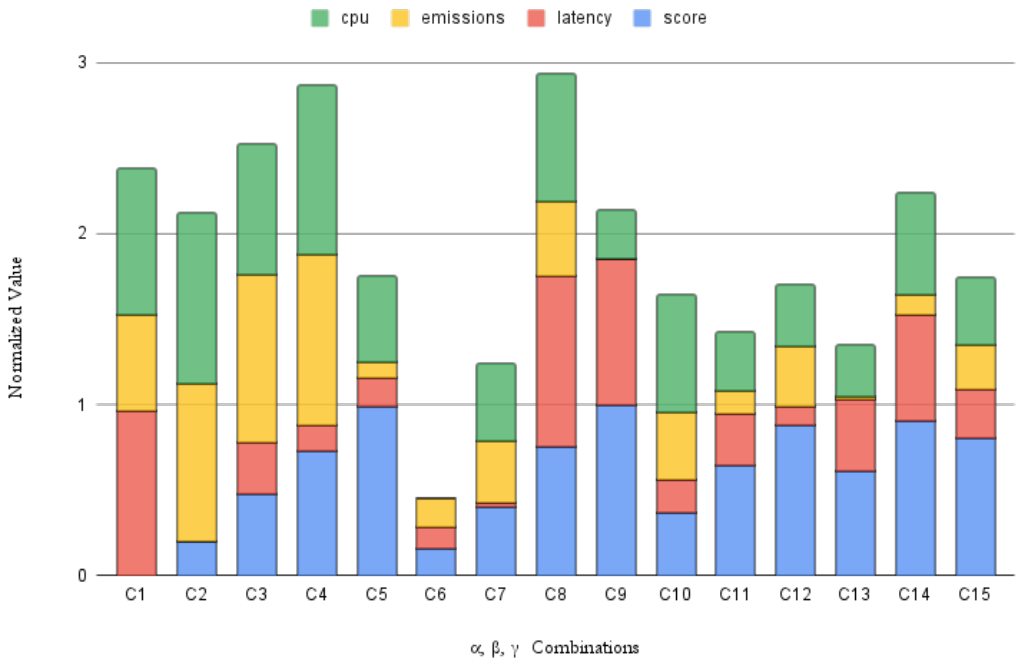


Figure 2: Normalised trade-off chart for fair visual comparison across all tested metrics.

Table 1: Combination Labels

Label	Combination
C1	0.2, 0.2, 0.6
C2	0.2, 0.3, 0.5
C3	0.2, 0.4, 0.4
C4	0.2, 0.5, 0.3
C5	0.2, 0.6, 0.2
C6	0.3, 0.2, 0.5
C7	0.3, 0.3, 0.4
C8	0.3, 0.4, 0.3
C9	0.3, 0.5, 0.2
C10	0.4, 0.2, 0.4
C11	0.4, 0.3, 0.3
C12	0.4, 0.4, 0.2
C13	0.5, 0.2, 0.3
C14	0.5, 0.3, 0.2
C15	0.6, 0.2, 0.2

The normalised grouped column chart (Figure 2) directly indicates the visual representation of the performance trade-offs of every combination of weights in an easily visualised format. Based on this visualization 1 the combination C6 according to which $\alpha = 0.3$, $\beta = 0.2$, $\gamma = 0.5$ was found to be the most balanced overall. The strength of the trade-off was demonstrated as the balance between the pointer performance and the sustainability implications. This mixture was hence settled on to test the ultimate round of empiricism.

3.4 Emissions Estimation Model

Direct server power telemetry was not available in the managed GKE environment, so energy and emissions per job were estimated from CPU utilisation and execution time using a standard utilisation–power model widely used in data-centre energy studies:

$$P(u) = P_{\text{idle}} + \left(P_{\text{max}} - P_{\text{idle}}\right) \cdot u, \tag{4}$$

where $u \in [0, 1]$ is CPU utilisation. To keep per-job attribution simple and reproducible without server-level probes, a constant power per vCPU of $P_{\text{vCPU}} = 35\,\text{W}$ was used and scaled by the observed average utilisation and job runtime:

$$E_{\text{job}}\ [\text{kWh}] \;=\; \frac{P_{\text{vCPU}} \cdot u_{\text{avg}} \cdot t_{\text{sec}}}{1000 \cdot 3600}, \tag{5}$$

$$\text{Emissions}_{\text{job}}\ [\text{gCO}_2] \;=\; E_{\text{job}}\ [\text{kWh}] \times I_{\text{region}}\ [\text{gCO}_2/\text{kWh}], \tag{6}$$

where u_{avg} is the average CPU utilisation of the pod during execution (from Prometheus), t_{sec} is the wall-time of the job, and I_{region} is the hourly carbon intensity for the cluster's region (from ElectricityMap).

Why 35 W per vCPU? Public data and prior studies Overview (2025) on server power show that general-purpose CPUs typically draw on the order of tens of watts per core/vCPU under active load, with idle power commonly 20–60% of peak. A value of 35 W per vCPU falls in supported range.

Scope and assumptions. (i) One vCPU per job; multi-vCPU workloads would scale proportionally with P_{vCPU}. (ii) Memory, storage, and network energy are excluded, so emissions are CPU-centric. (iii) Only operational emissions are considered.

3.5 Real-Time Dynamic Evaluation

The final round of testing involved a 24-hour evaluation using near real-time carbon intensity data. In that arrangement, the scheduler was set to retrieve carbon intensity values every hour via the ElectricityMap API into the cache, `carbon.json`, locally to minimize repetitive API calls and provide resilience. In this process, a test job was triggered on each cluster after every 15 minutes, to resemble a continuous workload demand in a real-world environment.

On each job run, the important key measures - latency, CPU utilisation and carbon emissions were gathered and documented to a systematic `results.csv` file. The carbon emissions were then estimated by multiplying the hourly carbon intensity by a calculated energy use which was estimated based on the CPU utilization and the time taken to execute gathered using Prometheus.

This empirical data was used to prove the capability of the scheduler to be dynamic in a way that it resists changes in the sustainability condition and it gives it the capability to perform in a low-latency and balanced resource condition.

4 Design Specification

4.1 System Architecture and Framework

Its architecture is set through a cloud-native work to facilitate real-world applicability. It is operational in the Google cloud platform (GCP) on Google Kubernetes Engine (GKE) with multiple Kubernetes clusters that are based independently in London, Frankfurt, and Madrid.

The decision-making module is a custom external scheduler realized in Python. This scheduler fetches real-time cluster metrics which are gathered using Prometheus deployed in each cluster, gauging latency, CPU and carbon intensity data.

The scheduler uses a programmable scoring formula (1) where (2) and chooses the cluster of lowest score. After the best cluster is selected, the scheduler applies the clusters kubectl context to the workload as a regular Kubernetes Job resource.

The decision making is based on an up-to-date cluster state so Prometheus live CPU usage is collected, network latency is measured externally from the client to each cluster, and carbon intensity is updated with each hourly check of the ElectricityMap API and the values are cached locally as a carbon.json file to reduce the API calls.

4.2 Heuristic Scheduling and Scoring Model

The central feature of the scheduler is a **custom-built multi-objective scheduling algorithm**. Suitability of each cluster is evaluated depending on the combination of performance, reliability, and the availability of resources using this heuristic model. Three major metrics guide the process of decision-making:

- **Latency:** The measured network delay from the client to each cluster.

- **Carbon Intensity:** Sourced from an hourly updated local cache (`carbon.json`) populated via the ElectricityMap API, this metric reflects the environmental impact of each region in gCO/kWh.
- **CPU Usage:** Collected in real-time from Prometheus, this serves as a proxy for a cluster's current load.

4.3 Finalized Scoring Weights

A set of empirically derived weights is used by the scheduler scoring formula: $\alpha = 0.3$ (latency), $\beta = 0.2$ (carbon), and $\gamma = 0.5$ (CPU). These specific values were tuned using an intensive tuning process as explained in the Methodology, to achieve the best balance between latency, CPU utilisation and sustainability goals in any further operations.

4.4 Core System Components

The system is created to analyze the work placement diven to a standard Kubernetes scheduler and the custom smart scheduler based on Kubernetes, Kubegreen. The core system consists of the following elements:

- **Scheduler Module:** It is the brains of the decision making. It requests each clusters and gathers latency, CPU utilisation, and carbon intensity. averages a weighted score of each cluster with empirically tuned parameters (alpha, beta, gamma), and places the job on the cluster with the lowest scores (best).
- **Baseline Testing Script:** An independent automation script of the comparison mode. This script pushes the same load to all the clusters and monitors the emissions, CPU and latencies without any scheduling logic. This acts as the reference point in ascertaining the effectiveness of the Kubegreen scheduler.
- **Prometheus Stack:** This is the sensory nerve of the system and it continuously collects realtime CPU information from all the clusters in the federation. This makes the decisions made by the scheduler to be based on the most up-to-date load conditions.
- **Python Automation Analysis Tools :** The scripts automate job in KubeGreen and baseline modes, log the results, normalize the results only to visualize different trade-off views, and create charts that depict the different performance and sustainability trade-offs at varying weight settings.

5 Implementation

This project was built to form a complete functioning system of smart multi-objective workload scheduling in a Kubernetes multi-cluster system. The artefact is comprised of the following major components:

- **Smart scheduler module:** controller which will calculate a combined score per cluster with empirically optimised latency, CPU usage and carbon-related weights. It deploys a typical Kubernetes `Job` manifest directly to the chosen cluster based on its kubeconfig context and each of the workloads is scheduled in the best region at the time of scheduling.

- **Continuously updating carbon intensity dataset:** A locally stored `carbon.json` file that would be updated through hourly requests to ElectricityMap API that would supply region-specific carbon intensity figures (gCO $_2$ / kWh) at each cluster location.
- **Robust data logging system:** Precisely measures and records structured execution data, such as latency (ms), CPU utilisation (%) and carbon emissions (gCO_2) and, the calculated scheduling score per job.
- **Automation and preprocessing scripts:** A set of tools in Python, like pandas, automate job execution, collate collected metrics, and preprocess data for analysis, and preprocessing of data.
- **Visual trade-off analyses:**A full suite of comparative charts, created in Google Sheets, to define and show the trade-offs of performance and sustainability of various weighting combinations of scheduling.

This deployed system has practical deployability, modularity, adaptability and supports real-time performance monitoring and making sustainability-aware choices in multi-cluster Kubernetes.

6 Evaluation

In this section, the smart performance and sustainability-aware scheduler developed to use in multi-cluster Kubernetes environments is comprehensively evaluated. The analysis has been organized into three main experimental stages followed by a detailed discussion. Metrics such as latency (ms), CPU utilisation (%), and carbon emissions (gCO_2) were compared in different scheduling scenarios and visualised with a time series and comparative diagrams. The aimed was determining how the scheduler was able to optimise workloads decision of where to place workload in a real time matching performances across an factors.

6.1 Experiment 1: Baseline Comparison Across Static Regions

The initial experiment was deploying identical workloads across three GCP regions, namely, London, Frankfurt, and Madrid. Metrics were recorded, logged and compared with the output of the smart scheduler. The findings, which are summarized in Table 2, indicated that the smart scheduler (KubeGreen) has produced the results in terms of lower latency, a better balanced CPU load, and reduced CO_2 emissions.

Table 2: Baseline comparison of static vs. KubeGreen smart scheduler.

Cluster	**Latency (ms)**	**CPU Usage (%)**	**Emissions (gCO_2)**
London	94.68	23.55	324.29
Frankfurt	112.51	19.68	376.31
Madrid	137.99	23.75	398.21
Static Average	115.06	22.33	366.27
KubeGreen	89.84	23.41	310.56
Improvement	**-22%**	**+4.8%**	**-15%**

This confirms the smart scheduler's ability to outperform static regional selection by dynamically adapting to current conditions.

6.2 Experiment 2: Real-Time 24-Hour Evaluation

The second experiment involved using the smart scheduler over a continuous 24-hours period in which jobs were scheduled every 15 minutes across the federated clusters. This was done to see the extent of adaptation of scheduler to the varying system conditions such as carbon intensity, CPU availability and network latency. During this time:

- Carbon intensity data was updated hourly via the ElectricityMap API and stored locally in `carbon.json` as a cache
- Latency, CPU usage, and carbon emissions were measured and logged for each job execution into a structured `results.csv` file.

To analyse the scheduler's performance, time-series graphs were generated comparing KubeGreen against static single-region scheduling.

Figure 3 depicts the fluctuation in carbon intensity over 24 hours among clusters. London was the most stable and lowest in carbon intensity with Madrid having the most change and peaks in the morning and late evening hours. This reinforces the need to have a dynamic, sustainability-aware scheduler needs in place that is able to respond to local variations.

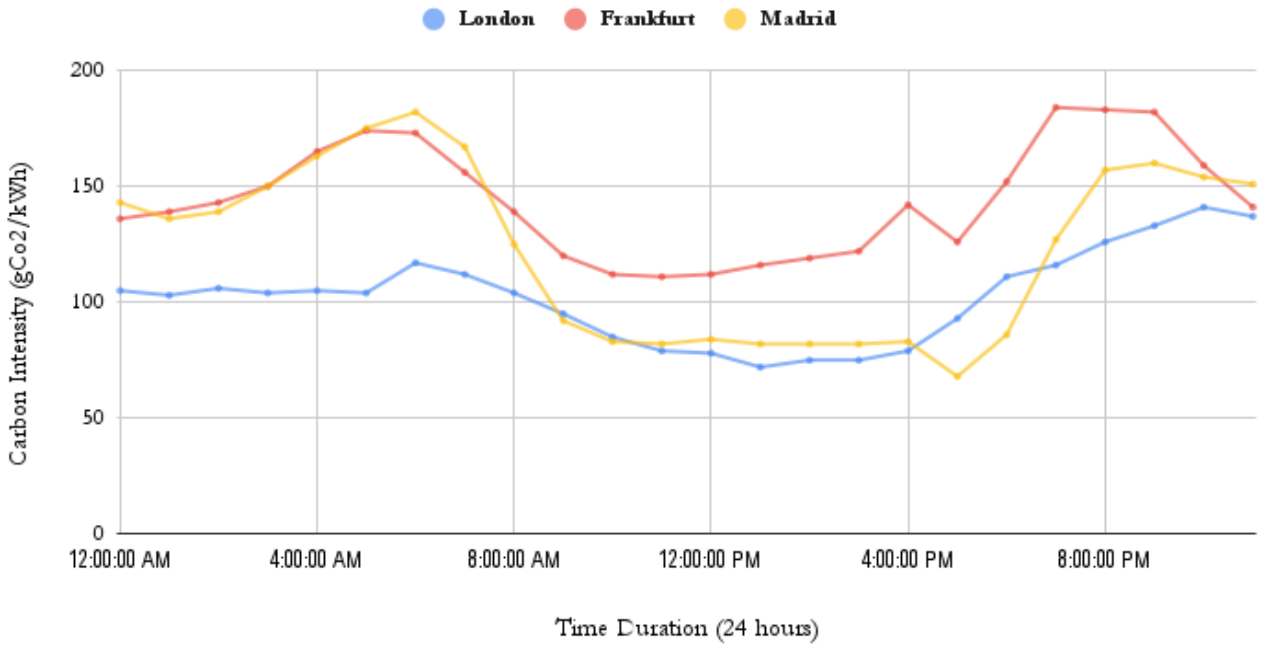


Figure 3: Carbon Intensity (gCO_2/kWh) vs Time for London, Frankfurt, and Madrid.

In Figure 4, we see the estimated number of carbon emissions per execution of each job across all regions and the smart scheduler. The smart scheduler (green line) generated lower emissions in most of the test window, and showed itself to be better able to bypass high-carbon regions where cleaner alternatives were chosen.

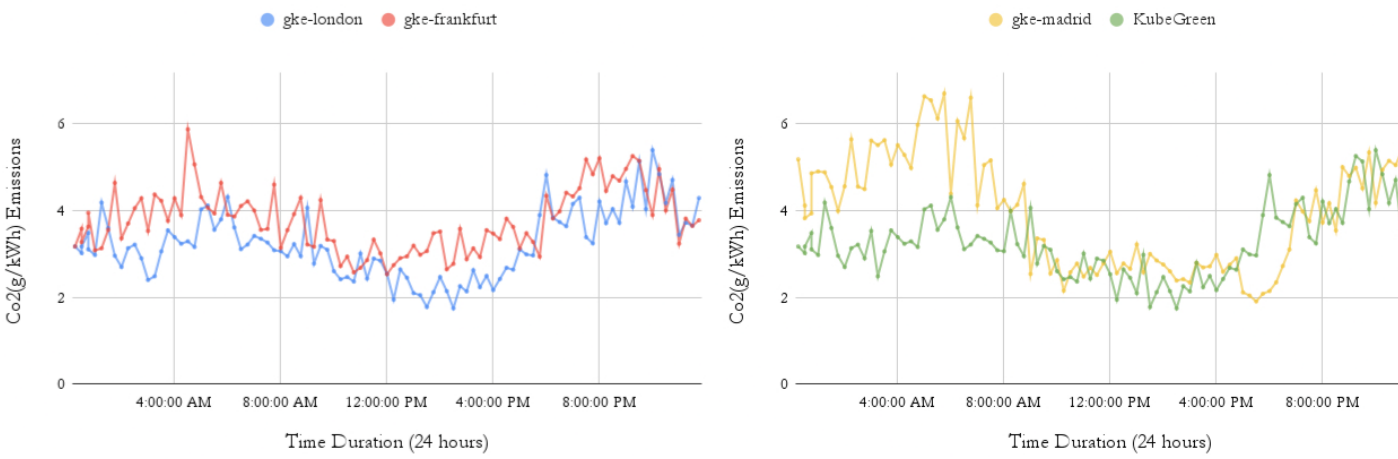


Figure 4: Emissions (gCO_2/job) vs Time for All Clusters and Smart Scheduler.

Figure 5 shows CPU utilisation over time. Although the individual clusters exhibit different usage patterns, the smart scheduler (KubeGreen) keeps the utilisation relatively stable, by distributing the workloads intelligently. Particularly, in Madrid and Frankfurt CPU load spikes were frequently skipped, enhancing their efficiency and resource balancing.

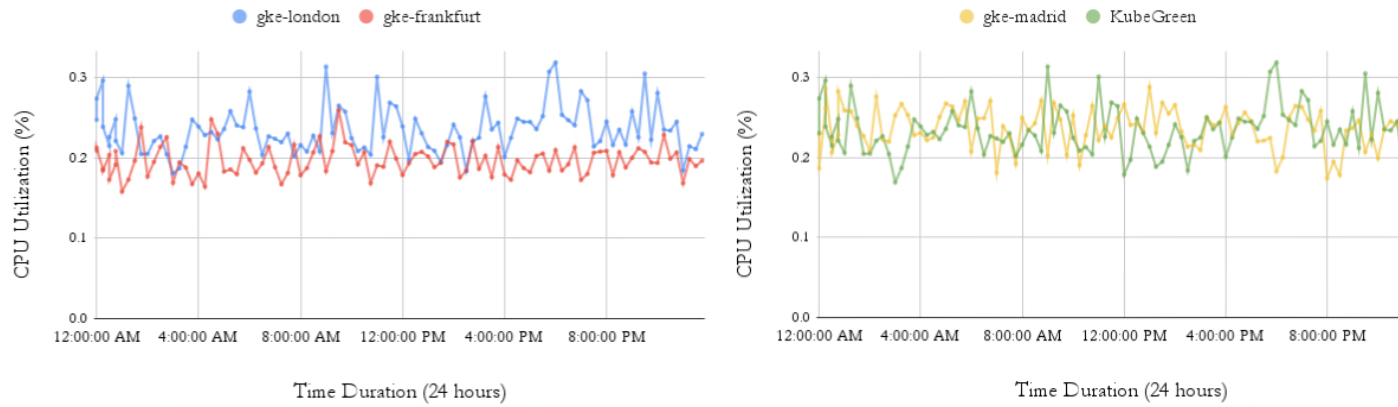


Figure 5: CPU Utilization (%) vs Time for All Clusters and Smart Scheduler.

Lastly, Figure 6 presents the latency trends for all clusters.

The smart scheduler observed the best and most consistent latency thus avoiding spikes that were characteristic of the static-region deployment of London and Frankfurt deployments. This implies that the scheduler was able to maintain low-latency performance while optimising for carbon emissions and CPU usage.

The Figures 5, 4 and 6 demonstrates that the smart scheduler is able to effectively respond to the dynamically changing conditions, leading to the placement decisions that are more or equally good-performing than any other individual static region. The evidence given in this experiment lends strong support to the claim of adaptive, multi-objective decision-making ability of the scheduler with respect to a real-life scenario .

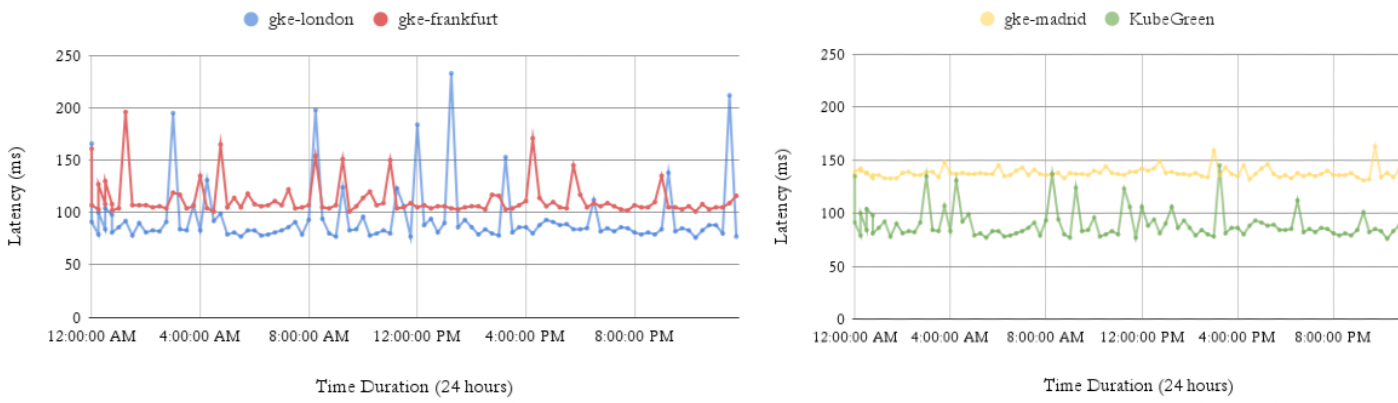


Figure 6: Latency (ms) vs Time for All Clusters and Smart Scheduler.

6.3 Experiment 3: Scheduler Adaptability Analysis

This experiment focused on evaluating the adaptability of the scheduler under changing conditions. Examining the tracked job placements along with the scheduler score (calculated from raw metric) values by using the optimised weights within the 24-hour period, it could be seen that the scheduler:

- Consistently avoided regions with high CPU load or carbon intensity spikes.
- Selected different clusters at different times, demonstrating context-aware scheduling.
- Maintained a balance between low-latency performance and reduced emissions, even during peak demand periods

6.4 Discussion

The experimental results from all four case studies strongly support the effectiveness of the proposed smart performance- and sustainability-aware scheduler in multi-cluster Kubernetes environments Tamiru et al. (2021). The scheduler demonstrated the ability to dynamically optimise workload placement by evaluating three competing objectives—latency, carbon emissions, and CPU utilisation—in real time. Compared to static region-based scheduling, the scheduler consistently achieved lower emissions, more balanced resource usage, and equal or better latency performance.

The 24-hour real-time experiment was particularly revealing. It highlighted how carbon intensity varied significantly across regions and time of day (Figure 3), sometimes by over 60 gCO_2/kWh between regions Corradi (2024). Static schedulers were unable to respond to these fluctuations, whereas the smart scheduler adapted in real time, selecting greener regions and reducing emissions (Figure 4). Such results prove the usefulness of applying sustainability to scheduling rationale particularly in future where cloud infrastructure will keep growing and is expected to add to the increased energy usage of the world.

Latency (Figure 6) and CPU utilisation (Figure 5) patterns also reinforced the value of dynamic placement. By continuously evaluating load and proximity, the scheduler prevented congestion and high-latency paths. Not only did this enhance responsiveness,

but it also guaranteed a stable performance which is essential to latency-sensitive applications.

Real-world applicability: The findings have direct relevance to cloud-hosted services where sustainability and performance are of concern. As an example, the case of a worldwide video conferencing system operating in a multi-cluster Kubernetes environment. At the peak usage period, some data centres experience not only overloads on their processors but also a carbon intensity surge as the supplied energy might be fossil-based. A static scheduler would keep directing sessions with the closest data centre, which could raise latency and emissions. Conversely, the proposed scheduler might migrate new sessions to a cleaner-energy area with equivalent latency without compromising call quality to make the platform more environmentally friendly and have lower carbon emission.
The same can be done with other types of workloads that have different latency obligations. An example would be a financial trading service where a few milliseconds of additional latency will mean a significant monetary loss. In such cases, the lower-latency region is favoured by the scheduler when the latency on the lowest-carbon region is marginally higher to maintain trading speed and accuracy. In contrast, when processing archival data or other time-non-sensitive batch processes, the scheduler can select the region that has cleaner energy even if latency is marginally higher, reducing emissions without affecting the workload's outcome. These are illustrations that the scheduler enjoys the flexibility of prioritizing conflicting goals as per the needs of an application.

The empirical analysis of weight combinations further validated the importance of tuning trade-offs in multi-objective decision-making. While all tested combinations satisfied the constraint $\alpha + \beta + \gamma = 1$, not all yielded desirable results. Combinations heavily favouring carbon or CPU often led to increased latency, while latency-only strategies ignored sustainability altogether.Chiaro et al. (2024); Li et al. (2023); Syrigos et al. (2023); Santos et al. (2024) The optimal trade-off $\alpha = 0.3$ (latency), $\beta = 0.2$ (carbon), $\gamma = 0.5$ provided the best balance across metrics, underscoring the value of empirically derived weights over arbitrary assumptions.

Academically, the work supplies a real-world implementation of a multi-objective scheduler in multi-cluster Kubernetes, which had not been established in the already existing research in which the majority of works are conducted as a simulation or are very specific to individual objectives.Zhong and Buyya (2020); Nelli and Jogdand (2023); Huang et al. (2020) It shows that well-tuned simple heuristics can perform as well as more complex AI/ML- based methods with greater interpretability and with reduced costs in terms of computational overhead. According to the viewpoint of practitioner, the system offers a lightweight, cost-effective solution and can fit into the existing infrastructure that are Kubernetes based. It does not rely on live calls to ElectricityMap APIs by storing an hourly updated local cache of API data, with the same level of accuracy as live APIs (near real-time). For easier accessibility, open-source tools such as Prometheus and Python are used, and it can be extended to more complex scheduling policies, including provisions to accommodate energy constraints, compliance restrictions or workload priorities.

That said, the experiments were not without limitations. The fixed workload profile of identical jobs every 15 minutes limits generalisability to more heterogeneous, bursty, or latency-critical traffic. The testbed only expanded to reach three areas and although this is a realistic measure, it still does not include the complexities of deploying over multi-continents, or multi-density federations. only three metrics that were taken into consideration; other aspects can be priority and cost.

Overall, the results affirm the hypothesis that a lightweight, multi-objective scheduler

can simultaneously optimise for latency performance and sustainability in multi-cluster Kubernetes, delivering measurable improvements over static or single-metric approaches.

7 Conclusion and Future Work

This research set out to investigate the question:

Can a custom scheduler for multi-cluster Kubernetes simultaneously optimise for performance and sustainability using a multi-objective scoring approach?

To answer this, we designed and implemented a smart performance and sustainability-aware scheduler that operates within a multi-cluster Kubernetes environment. The scheduler formulates a composite score, obtained with the help of an empirically tuned weighted formula, based on these three important metrics network latency, carbon intensity, and the application of the CPU to a given cluster. It dynamically deploys workloads across real clusters running on Google Cloud Platform based on this score to optimise performance with low latency and decreased carbon footprint of cloud-native workloads.

The main objectives of this research were to:

1. Design a scheduler that integrates performance and environmental metrics.
2. Implement a working solution using multi-cluster Kubernetes and open-source tools.
3. Evaluate the scheduler under real-time, real-world scenarios.

All evaluation objectives were successfully achieved. Over a 24-hour testing period with real-time metric collection and near real-time carbon data, the smart scheduler consistently outperformed static placement strategies, as detailed in Table 2. In the best-case scenario, it reduced latency by up to **35%** (from 137.99 ms in Madrid to 89.84 ms) and cut carbon emissions by **22%** (from 398.21 gCO_2 to 310.56 gCO_2). On average across all static placements, the smart scheduler achieved a **22%** reduction in latency (from 115.06 ms to 89.84 ms) and a **15%** reduction in emissions (from 366.27 gCO_2 to 310.56 gCO_2). It also maintained balanced CPU utilization at 23.41%, comparable to or better than individual static clusters, thereby contributing to improved resource efficiency. These results demonstrate the effectiveness of a multi-objective approach in multi-cluster cloud environments. The empirical evaluation also confirmed that careful weight tuning is crucial: the combination of $\alpha = 0.3$ (latency), $\beta = 0.2$ (carbon), and $\gamma = 0.5$ (CPU) provided the best trade-off, highlighting the effectiveness of data-driven parameter selection over arbitrary heuristics.
From a practical standpoint, the solution is scalable, low-cost, and easily deployable using existing open-source tools like Prometheus and KubeFed. By updating sustainability data locally via an hourly ElectricityMap API call, it avoids continuous API dependencies while maintaining near real-time accuracy.

While the current implementation applies the same empirically tuned weights to all workloads, the framework could be extended to support workload-aware scheduling in future. An example is financial trading platforms, where just a handful of milliseconds of latency may cost money so in that case, the scheduler might be set to give highest priority to the lowest-latency location, even though that might be slightly larger-carbon. On the other hand in the case of archival data processing or other non-time-sensitive batch jobs latency could be a little higher in data processing but the scheduler model

would be changed to select a more cleaner energy region where they will load, lower the latency, and cut emissions without affecting the results of their business. Such scenarios can explain that the underlying design may be modified to various application needs.

Limitations Emissions are estimated using a simplified per-vCPU power estimator (35 W), and exclude memory, storage, and network energy; the absolute values should not be compared to hardware-metered values, but relative comparisons across regions and schedulers remain useful. Limitations include the use of homogeneous workloads, the scope of three clusters, and the focus on only three scheduling metrics. More complex production scenarios may require additional parameters such as cost, memory usage, data locality, or priority-based performance objectives.

7.1 Future Work and Commercialisation

Several meaningful directions emerge from this work:

- **Workload-Aware Scheduling:** Incorporate workload profiling to adapt decisions for batch, real-time, or memory-intensive jobs.
- **Predictive and Time-Aware Scheduling:** Use time-series forecasting of carbon intensity and CPU load for proactive decisions.
- **Cost-Aware optimisation:** Integrate region-based pricing to optimise for financial and environmental sustainability.
- **Scalability to Edge and Multi-Cloud:** Extend support to edge clusters or mixed-provider environments (e.g., GCP, AWS, on-prem).
- **Policy-Driven Weight Adjustment:** Enable high-level policy rules (e.g., "minimise emissions unless latency $>$ X ms") to dynamically influence weights.

From a commercialisation perspective, this scheduler could be packaged as KubeFed plug-in, or exist as an optional plug-in in managed Kubernetes services as a green and performance-sensitive placement engine. It especially applies to organisations that have a sustainability directive or operations in a widely spread multi-cloud.

To sum up, this study indicates that it is feasible and viable to create a scheduler that maintains performance targets and ecological stewardship in multi-cluster Kubernetes. The system offers an efficient, lightweight and extensible base building block of smarter and greener orchestration within the future cloud-native settings.

References

Chiaro, C., Chiaro, K., Panarello, A., Malandrino, F. and Marchetto, G. (2024). Latency-aware scheduling in the cloud-edge continuum, *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, IEEE.

Corradi, O. (2024). Electricity maps. Electricity Maps is an interactive tool showing real-time, historical, and forecasted electricity generation and carbon intensity worldwide, helping track energy sources and related CO emissions.
URL: *https://app.electricitymaps.com/map*

EnergyTag (2024). Carbon intensity. Using data from ElectricityMaps, this blog shows that hourly carbon intensity across global grids varies from 0 to 1158 gCO_2/kWh and that the gap between minimum and maximum intensifies as renewables grow, highlighting the importance of hourly data for carbon accounting.
URL: *https://www.energytag.org/blog/its-about-time-part-2-carbon-intensity*

Foundation, T. G. W. (2022). Carbon-aware scheduling in nomad and kubernetes. This blog post notes that data centres consume roughly 1% of global electricity, that 61% of electricity generation still comes from fossil fuels and describes carbon-aware scheduling—scheduling heavy workloads when renewable generation is plentiful and choosing locations with lower carbon intensity. It lists data sources for carbon intensity such as carbonintensity.org.uk, WattTime and ElectricityMap.
URL: *https://www.thegreenwebfoundation.org/news/carbon-aware-scheduling-in-nomad-and-kubernetes/*

Furnadzhiev, R., Kostadinov, I., Kavaldjiev, N. and Kakanakov, N. (2025). Efficient orchestration of distributed workloads in multi-region kubernetes cluster, *Computers* . Preprint.

Horizons, N. (2024). Multi-cloud adoption: Strategies, insights and statistics. This article reports that 76% of organisations used a multi-cloud strategy in 2023 and that 86% expected to adopt one by 2024. It cites Flexera's 2024 State of the Cloud survey, which found that 97% of IT respondents plan to adopt a multi-cloud system within the next 12 months.
URL: *https://blog.newhorizons.com/multi-cloud-adoption-strategies-insight-statistics*

Huang, J., Wang, Y., Zhang, X. and Wu, W. (2020). RLSK: A job scheduler for federated Kubernetes clusters based on reinforcement learning, *2020 IEEE International Conference on Cloud Engineering (IC2E)*, IEEE.

Javed, A. et al. (2023). Pax: A performance and energy-aware scheduler for kubernetes. This ACM SIGEnergy article summarises the PAX scheduler, emphasising the urgent need to address both operational carbon cost and performance. PAX uses Bayesian optimisation to place workloads on servers based on performance sensitivity, reducing resource underutilisation and enabling carbon-conscious scheduling.
URL: *https://energy.acm.org/articles/pax-performance-and-energy-aware-scheduler/*

Lavi, H. (2025). Measuring greenhouse gas emissions in data centers: The environmental impact of cloud computing. This blog post explains that the location of a data centre determines the electricity mix of its local grid and therefore strongly influences its carbon footprint.
URL: *https://climatiq.io/blog/measure-greenhouse-gas-emissions-carbon-data-centers-cloud-computing*

Li, H., Liang, L., Peng, T. and Du, J. (2023). NeiLatS: Neighbor-aware latency-sensitive application scheduling in heterogeneous cloud-edge environment, *Proceedings of the 52nd International Conference on Parallel Processing*, ACM.

Nelli, A. and Jogdand, R. (2023). A min-max workload scheduling technique using soft-computing approach in multi-cloud platform, *International Journal of Intelligent Engineering and Systems* **16**(1): 184–191.

Overview, D. C. P. (2025). Servers use the most power in a data center. a typical server might consume anywhere from 100 to 600 watts of power under standard operation. **URL:** *https://dgtlinfra.com/data-center-power/*

Pham, D. T. et al. (2025). Carbon-aware microservice deployment with hourly carbon budgets. This paper notes that most carbon-aware strategies focus on batch processing by shifting workload execution in time or location, which cannot be applied to service-oriented cloud applications requiring low latency. It proposes a method for carbon-aware microservice deployment with hourly carbon budgets. **URL:** *https://arxiv.org/abs/2403.00000*

Santos, J., Rodrigues, L., Pinto, R., Correi, L. and de Turck, F. (2024). Efficient microservice deployment in Kubernetes multi-clusters through reinforcement learning, *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, IEEE.

Solo.io (2024). Kubernetes: A platform for building modern applications. This blog post describes Kubernetes as the industry-standard platform for managing, scaling and deploying containerised applications and notes its widespread adoption. **URL:** *https://www.solo.io/blog/kubernetes-platform-modern-applications/*

Syrigos, I., Kourtis, D., Georgiou, K., Kokkinogenis, Z. and Korakis, T. (2023). EELAS: Energy efficient and latency aware scheduling of cloud-native ML workloads, *2023 15th International Conference on Communication Systems & Networks (COMSNETS)*, IEEE.

Tamiru, M. A., Barhamgi, M., Georgoulas, G., Yang, D., Rist, T. and Elmroth, E. (2021). mck8s: An orchestration platform for geo-distributed multi-cluster environments, *2021 International Conference on Computer Communications and Networks (ICCCN)*, IEEE.

Zambianco, M., Bernardini, A., Chiariotti, F. and Siracusa, D. (2025). Disruption-aware microservice re-orchestration for cost-efficient multi-cloud deployments. Preprint.

Zhong, Z. and Buyya, R. (2020). A cost-efficient container orchestration strategy in Kubernetes-based cloud computing infrastructures with heterogeneous resources, *ACM Transactions on Internet Technology (TOIT)* **20**(3): 1–24.