NANYANG TECHNOLOGICAL UNIVERSITY

Investigation of Carbon-Intelligent Cloud Computing Policy

Chen Yueqi

School of Computer Science and Engineering

2023

NANYANG TECHNOLOGICAL UNIVERSITY

MSAI Master Project MSAI/22/048

Investigation of Carbon-Intelligent Cloud Computing Policy

Submitted by:
Chen Yueqi
under the supervision of
Wen Yonggang

School of Computer Science and Engineering

2023

Statement of Originality

I hereby certify that the work embodied in this report is the result of original research (and/or research survey), is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

[Signature of student in this space]

Chen Yueqi

Date: [25/03/2023]

Supervisor Declaration Statement

I have reviewed the content and presentation style of this report and declare that it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research (and/or research survey) and the writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accordance with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

[Signature of supervisor in this space]

Wen Yonggang

Date: [25/03/2023]

Authorship Attribution Statement

Please select one of the following and strikeout the other, as appropriate:

(A) This report **does not** contain any material from papers published in peer-reviewed journals or from papers accepted at conferences in which I am listed as an author.

(B) ~~This report **contains** material from the following paper(s) published in peer-reviewed journal(s) and/or accepted at conferences in which I am listed as an author.~~

Please provide a list of publications (journal/conference) pertaining to the report in case you choose option (B) above. Clearly state your contribution in the work.

1. Name of all the Authors. Title of the Paper. Name of Journal/Conference. Identifiers for the Journal/Conference, as appropriate. (Year)

   (a) In this paper, my contribution is primarily in ...

   (b) I have also contributed to the research in terms of ...

   (c) My co-authors primarily contributed in

2. Name of all the Authors. Title of the Paper. Name of Journal/Conference. Identifiers for the Journal/Conference, as appropriate. (Year)

   (a) In this paper, my contribution is primarily in ...

   (b) I have also contributed to the research in terms of ...

   (c) My co-authors primarily contributed in ...

[Signature of student in this space]

Chen Yueqi

Date: [25/03/2023]

Contents

Abstract

The rapid growth in data usage and cloud computing has led to an increased demand for data centers, which have a significant environmental impact due to their carbon dioxide (CO2) emissions. This study aims to reduce greenhouse gas emissions associated with multi-cloud task scheduling in data centers by proposing a carbon-aware improved policy that leverages the drift-plus-penalty technique in Lyapunov optimization. The improved policy accounts for transmitting costs and considers real-time carbon intensity monitoring data for workload scheduling and energy use. Results demonstrate that the carbon-aware improved policy offers substantial carbon emissions savings of about 45% ± 3% compared to both the carbon-aware baseline policy and the queue-aware policy while maintaining mean-rate stability of queues. This environmentally friendly and efficient solution for resource allocation and scheduling has significant potential to mitigate the negative impacts of data centers on climate change. However, as a heuristic proposal, further research and improvements are needed to enhance its effectiveness.

Keywords: carbon-aware scheduling policy, data center, Lyapunov Optimization, drift-plus-penalty techinique, carbon savings, mean-rate queue stability.

Acknowledgement

I would like to thank Professor Wen Yonggang for his significant advice, assistance, and knowledge during the duration of our project. I also want to thank Cao Zhiwei, Liu Tianshu, and Wang Ruihang for their insightful feedback and unwavering dedication to this project. Their experience and collaboration have been critical to the project's success, and I am really thankful for the opportunity to work with such esteemed partners.

Acronyms

- ADMM: Alternating Direction Method of Multipliers
- ADMM-DACM: ADMM-based Distributed online Algorithm for Cost Minimization
- AI: Artificial Intelligence
- ARIMA: Autoregressive Integrated Moving Average
- CECM: Carbon-aware Electricity Cost Minimization
- COCA: COst minimization and CArbon neutrality
- CO2: Carbon dioxide
- CPU: Central Processing Unit
- CRAC: Computer Room Air Conditioning
- DAG: Directed Acyclic Graph
- ECOGreen: Electricitycost optimization strategy for Green data center
- ENTSO-E: European Network of Transmission System Operators for Electricity
- ESO: Electricity System Operator
- FS: Flexible Smoothing
- GHG: GreenHouse Gas
- HDD: Hard Disk Drive
- IEA: International Energy Agency
- ISO: Independent System Operator
- IT: Information Technology
- PPA: Power Purchasing Agreement
- PUE: Power Usage Effectivenes
- PV: Photovoltaic
- REC: Renewable Energy Credit
- SCMA: Stochastic Cost Minimization Algorithm
- SPAN: Server Provisioning for cArbon Neutrality

- SSD: Solid-State Drive
- UPS: Uninterruptible Power Supplies
- VM: Virtual Machine

Symbols

- V: a non-negative hyper-parameter used to scale the penalty term;
- M: the number of task types;
- N: the number of clouds;

$\forall t \in \{0,1,2,\dots\}$, $\forall m \in [M]$, $\forall n \in [N]$:

- $a(t)$: $M \times 1$ matrix such that $a(t)[m]$ the number of type-m tasks arrived at edge server at time t;
- $C(t)$: carbon emissions of the whole computing network at time t;
- $\bar{C}$: time-average carbon emissions;
- $C^e(t)$: carbon intensity at the edge server location at time t;
- $C^c(t)$: $N \times 1$ matrix such that $C^c(t)[n]$ is the carbon intensity at cloud n location at time t;
- $d(t)$: $M \times N$ matrix such that $d(t)[m][n]$ is the number of type-m tasks transmitted to cloud n at time t;
- P^c: $N \times 1$ matrix such that $P^c[n]$ is the energy constraint of cloud n;
- $p^c(t)$: $M \times N$ matrix such that $p^c(t)[m][n]$ is the energy consumption of processing a single type-m tasks on cloud n;
- $P^c_{n,total}(t)$: the total energy consumption of cloud n at time t;
- P^e: the energy constraint of edge server;
- $P^e_{process}(t)$: the total energy consumption of edge server for processing tasks at time t;
- $p^e_{process}$: $M \times 1$ matrix such that $p^e_{process}[m]$ is the energy consumption of processing a single type-m tasks by edge server;
- $P^e_{send}(t)$: the total energy consumption of edge server for sending tasks to clouds at time t;
- p^e_{send}: $M \times 1$ matrix such that $p^e_{send}[m]$ is the energy consumption of sending a single type-m tasks to any of the clouds;
- $P^e_{total}(t)$: the total energy consumption of edge server at time t;
- $Q^c(t)$: $M \times N$ matrix where $Q^c(t)[m][n]$ is the queue length of type-m tasks at cloud n;
- $Q^e(t)$: $M \times 1$ matrix where $Q^e(t)[m]$ is the queue length of type-m tasks at edge server;

- $w^c(t)$: $M \times N$ matrix such that $w^c(t)[m][n]$ is the number of type-m tasks processed by cloud n at time t;
- $w^e(t)$: $M \times 1$ matrix such that $w^e(t)[m]$ is the number of type-m tasks processed by edge server at time t.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Data centers are facilities that accommodate a lot of servers and storage systems for managing and storing digital data. By facilitating quick and dependable access to information, data centers in the modern world play a vital role in the operation of enterprises, organizations, and governments. They are in charge of maintaining and storing huge amounts of data that are accessed by billions of people globally.

1.1.1 Data Center Infrastructure

The infrastructure of a data center as shown in Figure 1.1 consists of the physical and technological components that are required to support its operation. This infrastructure can be divided into several categories, including:

- Power Infrastructure: Data centers require a reliable and continuous power supply to keep their servers and other equipment operational. The power infrastructure includes transformers, generators, uninterruptible power supplies (UPS), and electrical distribution systems.
- Cooling Infrastructure: The equipment in data centers generates a significant amount of heat, which can cause damage and reduce efficiency. The cooling infrastructure is designed to regulate the temperature and humidity levels within the data center and typically includes air conditioning units, chillers, and computer room air conditioning (CRAC) units.
- Network Infrastructure: Data centers require extensive networking infrastructure to connect servers and storage devices and allow users to access the data stored in the data center. This infrastructure includes switches, routers, firewalls, load balancers, and other networking equipment.
- Storage Infrastructure: Data centers store vast amounts of data, and the storage infrastructure is designed to provide fast and reliable access to this data. This infrastructure includes storage devices such as hard disk drives, solid-state drives, and tape drives.
- Monitoring and Management Infrastructure: Data centers use sophisticated monitoring and management infrastructure to monitor the performance of servers and equipment, detect and diagnose problems, and manage resources efficiently. This infrastructure includes software tools, monitoring systems, and management consoles.

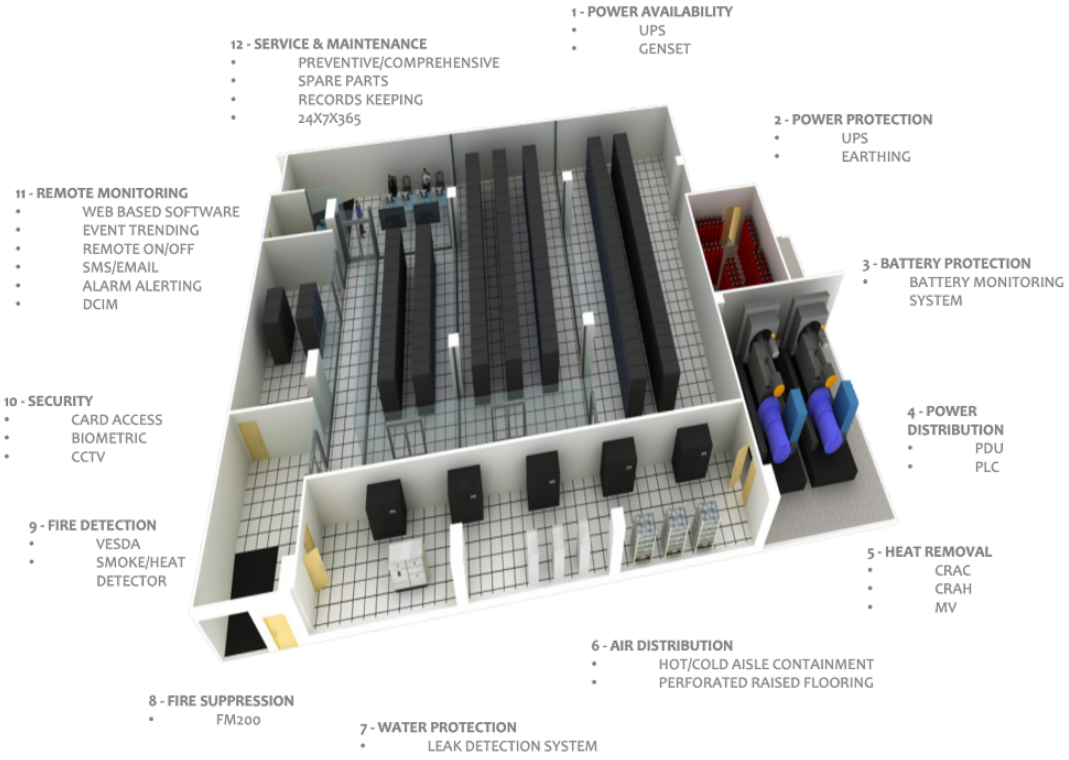

Figure 1.1: Infrastructure of Data Center [1]

- Physical Security Infrastructure: Data centers are designed to be secure facilities, and the physical security infrastructure is essential to ensure the safety of the equipment and data. This infrastructure typically includes access control systems, video surveillance systems, perimeter security, and alarm systems.
- Fire Suppression Infrastructure: The equipment in data centers is at risk of damage or destruction from fires, so the fire suppression infrastructure is designed to extinguish fires quickly without damaging the equipment. This infrastructure typically includes fire detection systems, suppression systems, and extinguishers.

1.1.2 Carbon Footprint of Data Centers

Due to the exponential rise in data usage and cloud computing in recent years, the demands of Data centers have grown in significance. Data centers do, however, also have a sizable negative environmental impact, mostly because of their carbon dioxide (CO2) emissions. To power and cool their servers and equipment, data centers use enormous amounts of electricity. According to International Energy Agency (IEA), data centres and data transmission networks are responsible for nearly 1% of energy-related greenhouse gas (GHG) emissions in 2022. [4] This significant energy consumption causes a considerable amount of CO2 to be released into the atmosphere, which aids in the process of climate change. Data centers are predicted to continue using power at an unprecedented rate; by 2030, estimates in Figure 1.2 suggest that they may account for 3-13% of all electricity used globally [2].

To address the carbon footprint of data centers, various strategies are explored to reduce the energy consumption , including:

- Use Renewable Energy: Data centers can reduce their carbon footprint by using renewable energy sources such as solar, wind, and hydroelectric power. This can be done by

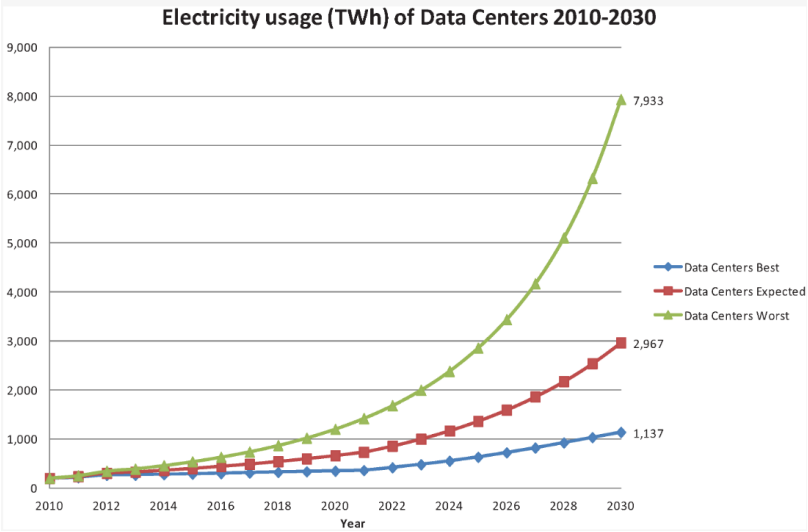

Figure 1.2: Prediction of electricity demand of data centers suggest that they may account for 3-13% of all electricity used globally by 2030 [2]

installing on-site renewable energy systems or by purchasing renewable energy credits from renewable energy providers.

- Efficient Cooling Systems: Cooling systems in data centers can consume a significant amount of energy, and therefore, efficient cooling systems can reduce energy consumption and associated carbon emissions. The use of free cooling techniques, such as economizers and evaporative cooling, can reduce energy consumption significantly.

- Use Efficient Hardware: Data centers can reduce their carbon footprint by using more energy-efficient hardware, including servers, storage systems, and networking equipment. The use of solid-state drives (SSDs), which consume less power than traditional hard disk drives (HDDs), can also contribute to reducing energy consumption.

- Improve Cooling System Design: The design of cooling systems can significantly impact energy consumption and carbon emissions in data centers. Therefore, data center operators should consider using modular cooling systems that allow for scaling, using hot aisle containment, and deploying distributed cooling systems.

- Embrace Cloud Computing: Cloud computing can provide a more energy-efficient alternative to on-premises data centers. Cloud providers invest in energy-efficient hardware, and economies of scale enable them to achieve higher energy efficiency. Additionally, many cloud providers use renewable energy sources to power their data centers.

1.2 Motivation

A special potential for data center decarbonization using multi-cloud task scheduling is offered after investigating the various grid carbon intensities. The data collected by Wiesner et al. [5] is visualized in Figure 1.3, which suggests the characteristics of carbon intensities in different time slots and locations:

- Carbon intensity varies with time and region: The carbon intensity of electricity varies depending on the time and location of energy generation. This is due to the mix of energy sources used to generate electricity, which can include renewable sources like wind and

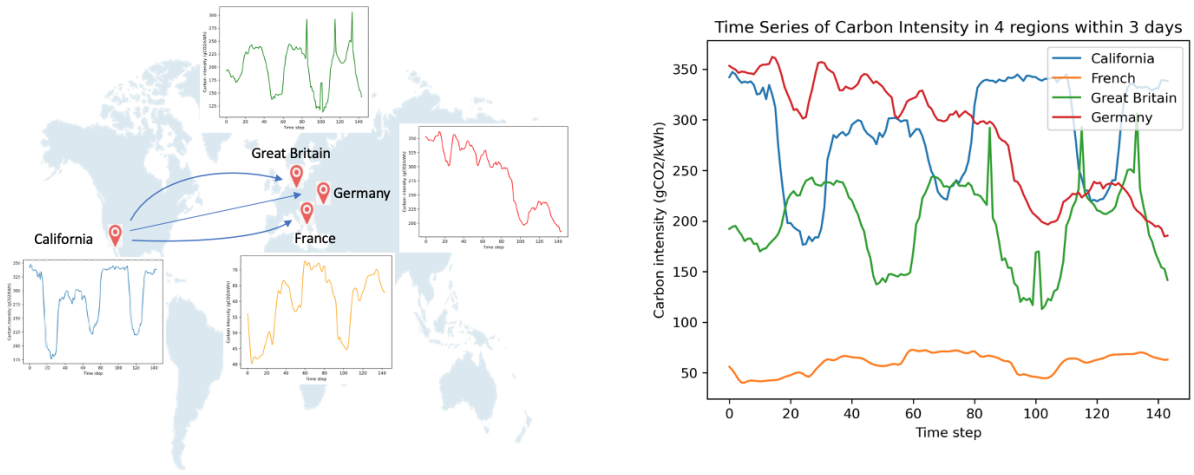

Figure 1.3: Time series of carbon intensity in different locations indicates that (1) carbon intensity varies with time and region; (2) periodic pattern shows in time series of California and Great Britain; (3) opposite periodic patterns are shown in California and Great Britain.

solar, as well as fossil fuel-based sources like coal and natural gas. By taking into account the time and location of energy generation, data centers can schedule their workload to run during periods when the carbon intensity is lower. This can be achieved by leveraging the capabilities of multi-cloud task scheduling, which can move workloads to cloud providers that are operating in regions with lower carbon intensities at specific times.

- Periodic pattern shown in some of the time series of carbon intensity: Some regions, such as California and Great Britain, have periodic patterns in their carbon intensity levels possibly due to seasonal variations, weather patterns, or daily energy consumption patterns. By analyzing the historical carbon intensity data for different regions, data centers can identify these periodic patterns and adjust their workload scheduling accordingly. This can help reduce the overall carbon footprint of the data center by avoiding periods of high carbon intensity.

- Carbon intensities of different countries have different periodic patterns: The periodic patterns in carbon intensity levels can vary depending on the country or region. For instance, the peaks in California coincide with the bottoms in Great Britain, which may be caused by jet lag. Some other factors such as energy mix, climate, and energy consumption patterns may also influence the patterns in different locations. Data centers may benefit from the variation in patterns of carbon intensity across several nations and regions by utilizing multi-cloud job scheduling. By dynamically arranging workloads to execute in areas with lower carbon intensities at specified times, this can assist minimize the data center's total carbon footprint.

1.3 Objectives and Specifications

1.3.1 Objectives

The objective of data center decarbonization in this project is to reduce greenhouse gas emissions associated with multi-cloud task scheduling. By scheduling workloads to run during times of low carbon intensity, data centers can minimize their carbon emissions while still meeting the needs of their users. This involves using real-time carbon intensity monitoring data to make informed decisions about workload scheduling and energy use.

1.3.2 Specifications

This project will provide insights into how data centers can reduce their carbon emissions by leveraging real-time carbon intensity monitoring data to schedule workloads during times of low carbon intensity. It will not cover the use of renewable energy and battery technologies.

1.4 Contribution of the Dissertation

In this dissertation, we show the enhancement of the scheduling policy proposed by Yang and his team in 2022 [6]. Their proposed policy considers the case that the edge server has only one responsibility, which is scheduling tasks to clouds. In this dissertation, the edge server is not only responsible for transmitting tasks but also processing tasks to avoid unnecessary carbon emissions during networking.

1.5 Organisation of the Dissertation

This dissertation is organized into six chapters. Cchapter 1 is the introduction which gives a background and motivation of the project as well as its objectives and specifications. We also declare our contributions to this dissertation. Chapter 2 discusses the existing solutions related to data center decarbonization. Chapter 3 presents the system model used in the project, which includes assumptions, a virtual queueing network model, problem statement, and a carbon-aware task scheduling policy. Charpter 4 describes the design of our experiment, including environment configuration, datasets selection, and baseline models' descriptions. Chapter 5 is a performance evaluation of the proposed policy, which includes evaluation methodology, carbon savings comparison, queues stability, and the trade-off between carbon savings and queue lengths. Chapter 6 concludes the project by summarizing the contributions of the proposed policy and discussing future work such as the utilization of renewable energy, battery management and the guarantee of job deadlines. All the codes are arranged in appendixes.

Chapter 2

Literature Review

The growing need for sustainable data centers has led to numerous studies and models for carbon-aware electricity cost minimization. In this review, we will discuss various studies about their renewable energy type, battery management, and task types as shown in table 2.1:

	Renewable Energy Type	Battery Management	Task types
CECM [7]	wind	No	delay-tolerant
COCA [8]	wind & solar & off-site & RECs	No	delay-sensitive
SPAN [9]	wind & solar & off-site & RECs	No	delay-sensitive
SCMA [10]	wind & solar	charge/discharge thermal storage unit	batch workload
ADMM-DACM [11]	wind	charge/discharge renewable energy	batch workload
ECOGreen [12]	solar	battery life	VM workloads
Smoother [13]	wind & solar	use to smooth renewable power supply	delay-tolerant

Table 2.1: Summary of the renewable energy type, battery management, and task types used in each of the 7 models.

2.1 Utilization of Renewable Energy

The Carbon-Aware Electricity Cost Minimization (CECM) method [7] models a large-scale sustainable data center partially powered by an on-site wind energy plant. This discrete-time system evolves over equal length time slots, with the available on-site wind power modeled as a function of local wind speed. While the study omits on-site solar power, authors declare that the algorithm can still be applied to sustainable data centers with on-site solar plants. Since the available on-site solar power can be modeled as a function of local irradiance and temperature, the total solar power can be calculated for each time slot if the data center has an on-site solar plant. The Cost Minimization and Carbon Neutrality (COCA) [8] and the Server Provisioning for cArbon Neutrality (SPAN) [9] focus on both on-site and off-site renewable energy. On-site renewable energy, including solar and wind, exhibit an intermittent nature due to their dependency on weather conditions. These methods recognize that the most desirable locations for renewable energy generation and data center construction may not coincide, leading to a reliance on off-site renewable energy to achieve carbon neutrality such as Power

purchasing agreements (PPAs) and renewable energy credits (RECs). Different from SPAN, all the analysis in COCA will ignore the off-site renewable energy from the model and use the REC parameter as the desired total energy cap. The Stochastic Cost Minimization Algorithm (SCMA) [10] considers on-site wind generation in New York and Chicago, as well as on-site solar generation in Palo Alto and Houston. While ADMM-based Distributed online Algorithm for Cost Minimization (ADMM-DACM) [11] extracts wind power from four datasets of hourly total production for 29 days from the California ISO as the renewable energy source. The Electricity Cost Optimization Strategy for Green data center (ECOGreen) [12] employs real-world weather data in conjunction with a linear model of a photovoltaic (PV) array to compute solar power quantity. Since the PV module provides energy proportional to the solar irradiance intensity, which primarily depends on the weather. Smoother [13] emphasizes the importance of wind resource characteristics and turbine properties in evaluating the power generated from wind energy sources. They also highlight solar radiation intensity and PV solar cell features as critical factors in determining power generated from solar resources.

2.2 Battery Management

The Stochastic Cost Minimization Algorithm (SCMA) [10] makes assumption that several geo-distributed data centers have their own thermal storage systems. Traditional battery systems, such as uninterruptible power supply (UPS) units, can be expensive and frequent charging and discharging can severely impact their lifetimes. In contrast, thermal storage is more cost-effective, and the authors utilize it to help reduce the cooling costs of data centers. In this paper, thermal energy storage systems employ chilled liquid or ice as a thermal battery, allowing data center operators to run air conditioners at night when rates are lower and use the chilled liquid during the day for cooling purposes. ADMM-based Distributed online Algorithm for Cost Minimization (ADMM-DACM) [11] assumes that each data center has its own battery for energy storage and an on-site renewable energy generation plant for power generation. Batteries store energy from renewable sources when production exceeds the data center's electricity demands and/or from the electricity market when prices are relatively low, yielding long-term benefits. ECOGreen [12] takes into account the health of the battery, considering power loss in battery banks due to aging and charging sequences. Therefore, the researchers control the charging sequences of the battery bank and model the battery aging. The battery model is based on Peukert's law [12] and is designed as a plug-and-play component that can be easily replaced and adapted. Smoother approach [13] formulates the problem of Flexible Smoothing (FS) into a constrained nonlinear programming problem, which determines the optimized charge/discharge scheme for renewable energy. The researchers explore and exploit finite battery capacity, as well as the limited charging/discharging rate, accommodating the demands of real-world situations. They demonstrate the feasibility of using UPS batteries to complement the fluctuation of renewable energy.

2.3 Investigation of Task Types

Since delay tolerant workloads are highly relevant to various Internet services. Carbon-Aware Electricity Cost Minimization (CECM) [7] formulates the electricity cost minimization problem as a time-averaged stochastic programming problem and trades off between electricity cost and the performance of delay tolerant workloads without requiring future information about time-varying system states. While Smoother [13] also employs deferrable workloads, such as batch work, offline tasks, and low-priority jobs, to optimize energy consumption and reduce costs in data centers.

Similarly, Stochastic Cost Minimization Algorithm (SCMA) [10] and ADMM-based Distributed online Algorithm for Cost Minimization (ADMM-DACM) [11] focus on batch workloads that demand significantly larger computational capability. By leveraging the delay-tolerant nature of batch workloads, these methods can enhance renewable energy utilization by postponing their services to periods with abundant renewable sources without violating execution deadlines.

In contrast, Cost Minimization and Carbon Neutrality (COCA) [8] and SPAN (Server Provisioning for cArbon Neutrality) [9] target delay-sensitive interactive workloads, which can constitute over 50% of data center workloads. To measure the overall data center delay performance, they introduce delay cost, which accounts for the delay-induced revenue loss and user dissatisfaction. It is important to note that they disregard the network delay cost between the load distributor and servers. The key difference in delay cost calculation between COCA and SPAN lies in SPAN's introduction of a weighted term to represent the importance of specific job types.

Furthermore, ECOGreen [12] utilizes the autoregressive integrated moving average (ARIMA) prediction model to forecast next-hour traces per VM (virtual machine) based on the CPU and memory utilization data from the previous week, considering the daily periodicity observed in Google Cluster traces. Impressively, the worst-case prediction error remains below 10% for all VMs.

Chapter 3

System Model

In this dissertation, we consider a computing network consisting of one edge server and N cloud computing resources. The edge server, positioned closer to the end-users, is responsible for receiving tasks and making decisions on their processing locations. At each discrete time step t, the edge server must determine whether it should transmit the tasks to one or more of the N available cloud resources for processing or handle the tasks locally within its own computing capabilities.

The role of the edge server in this system is crucial, as it must balance various factors, such as processing latency, resource utilization, energy consumption, and real-time carbon intensity, when deciding where to process tasks. By leveraging the computational resources of the N clouds, the edge server can offload tasks, allowing for more efficient resource allocation and potentially reducing processing times. However, offloading tasks to the clouds might also introduce additional communication overhead and energy consumption due to data transmission.

On the other hand, each cloud resource in the network must also make decisions regarding the number of tasks it should process at time t. This decision-making process is influenced by factors such as current workload, available processing resources, energy consumption, and real-time carbon intensity.

3.1 Assumptions

In this section, we outline the assumptions made in our study to create a foundation for the development and analysis of the proposed model. These assumptions serve as a starting point for our research, allowing us to simplify certain aspects of the problem while focusing on the primary objectives of the study. The following key assumptions have been made:

- **Assumption 1:** Only non-DAG tasks are took into account.
- **Assumption 2:** i.i.d. assumption of task arrival quantity, edge and clouds carbon intensity.
- **Assumption 3:** The proposed policy does not include any past and future information; it is just based on the computing network's current status.

3.2 Virtual Queueing Network Model

This virtual queue network model can aid in capturing the current status of our computer network, which will be useful for analyzing the carbon emissions of our system model. We have all the data we require to compute the carbon emissions for our scheduling policy in the current

state, even without taking into account the past and future knowledge represented by assumptions.

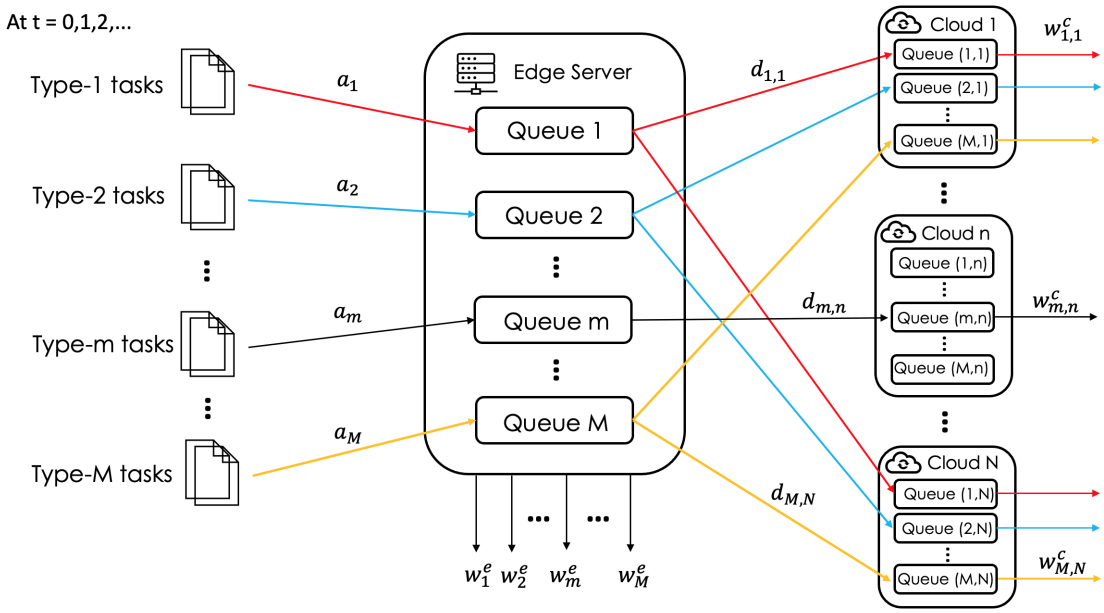

Figure 3.1: Work flow of vitual queueing network: at each time t, a_m number of type-m tasks enter the queue m in edge server, w_m^e or $d_{m,n}$ number of type-m tasks depart from edge server, and queue (m,n) in cloud n receives $d_{m,n}$ type-m tasks while $w_{m,n}^c$ type-m tasks leave cloud n.

Figure 3.1 illustrates the work flow of the time-driven model proposed in this study. The model operates in discrete time steps, denoted by $t \in 0, 1, 2, \ldots$. At each time step, type-m tasks arrive at the edge server, with a quantity of a_m for each task type, where $m \in 1, 2, \ldots, M$.

In the edge server, there is a virtual queue for each task type to manage the incoming tasks. Upon arrival, a type-m task enters its corresponding queue m, where it waits to be dispatched for processing. The edge server is responsible for deciding whether to process the type-m tasks locally or offload them to one of the available cloud resources. This decision is guided by a scheduling policy that computes the number of type-m tasks, $w_m^e \geq 0$, to be processed by the edge server.

If the edge server decides not to process the tasks locally, it will schedule $d_{m,n} \geq 0$ number of type-m tasks to be offloaded to cloud n. Once offloaded, these tasks enter a cloud queue designated as (m, n), where they wait to be processed by the cloud resource. Concurrently, each cloud resource processes $w_{m,n}^c \geq 0$ number of type-m tasks according to its own processing capabilities and the scheduling policy.

By expanding the model's structure in this manner, we can further investigate the dynamics of task processing and resource allocation in the edge-cloud computing network. This detailed view allows us to analyze the performance of scheduling policy and optimize the decision-making process for the edge server and cloud resources. The ultimate goal is to develop a comprehensive framework that efficiently mitigates the carbon emissions while maintain the queue stability in the computing network.

3.3 Problem Statement

In this section, we formulate the problem by considering the dynamics of queues, energy consumptions subject to energy constraints, and carbon emissions in both edge server and clouds.

3.3.1 Dynamics of Virtual Queues

According to the virtual queueing network model, we are able to describe the dynamics of the virtual queue as follows:

- Given that the queue length of type-m task in edge server at time $(t+1)$ and t is denoted as $Q_m^e(t+1)$ and $Q_m^e(t)$. At time t, $d_{m,n}(t)$ is the number of type-m tasks sent to cloud n, $w_m^e(t)$ is the number of type-m tasks processed by edge server, $a_m(t)$ is the number of type-m tasks arrived at edge server. Obviously, $d_{m,n}(t)$ and $w_m^e(t)$ are the number of type-m tasks leaving the edge server, resulting in the subtraction from the length of edge queue m. While $a_m(t)$ is the number of type-m tasks entering edge server, resulting in the addition to the length of edge queue m. Therefore, the dynamic virtual queue of edge server can be defined as:

$$\forall m \in [M],\ Q_m^e(t+1) = \max(Q_m^e(t) - \sum_{n=1}^{N} d_{m,n}(t) - w_m^e(t), 0) + a_m(t) \tag{3.1}$$

- Given that the queue length of type-m task in cloud n at time $(t+1)$ and t is denoted as $Q_{m,n}^c(t+1)$ and $Q_{m,n}^c(t)$. At time t, $w_{m,n}^c(t)$ is the number of type-m tasks processed by cloud n, $d_{m,n}(t)$ is the number of type-m tasks arrived at cloud n. Obviously, $d_{m,n}(t)$ is the number of type-m tasks entering cloud n, resulting in the addition to the length of cloud queue (m,n) while $w_{m,n}^c(t)$ is the number of type-m tasks leaving cloud n, resulting in the subtraction from the length of cloud queue (m,n). Therefore, the dynamic virtual queue of cloud n can be defined as:

$$\forall m \in [M],\ \forall n \in [N],\ Q_{m,n}^c(t+1) = \max(Q_{m,n}^c(t) - w_{m,n}^c(t), 0) + d_{m,n}(t) \tag{3.2}$$

3.3.2 Energy Consumptions

Based on the virtual queueing model, we are able to compute the total energy consumptions of edge server and each cloud respectively:

- The energy consumption of edge server consists of two parts: sending and processing:

$$P_{total}^e(t) = P_{send}^e(t) + P_{process}^e(t)$$

1. **Sending:** Delivering different types of tasks may consume different amounts of energy. Therefore, $p_{m,send}^e$ denotes the amount of energy consumed by sending single type-m task to any of the clouds while $d_{m,n}(t)$ denotes the number of type-m tasks sent to cloud n at time t. Multiplying $p_{m,send}^e$ by $d_{m,n}(t)$ gives us the energy consumption of sending $d_{m,n}(t)$ number of type-m to cloud n at time t. The summation, $\sum_{m=1}^{M}\sum_{n=1}^{N} d_{m,n}(t)p_{m,send}^e$, suggests edge server's total offloading consumption at time t, denoted as $p_{m,send}^e$.

$$P_{send}^e(t) = \sum_{m=1}^{M}\sum_{n=1}^{N} d_{m,n}(t)p_{m,send}^e$$

2. **Processing:** Processing different types of tasks may consume different amounts of energy. Therefore, $p^e_{m,process}$ denotes the amount of energy consumed by processing single type-m task while $w^e_m(t)$ denotes the number of type-m tasks processed by edge server at time t. Multiplying $p^e_{m,process}$ by $w^e_m(t)$ gives us the energy consumption of processing $w^e_m(t)$ number of type-m by edge server at time t. The summation, $\sum_{m=1}^{M} w^e_m(t) p^e_{m,process}$, suggests edge server's total processing consumption at time t, denoted as $P^e_{process}(t)$.

$$P^e_{process}(t) = \sum_{m=1}^{M} w^e_m(t) p^e_{m,process}$$

Sum up the energy consumed by sending and processing, we get the total energy consumption of edge server at time t, denoted as $P^e_{total}(t)$:

$$\begin{aligned} P^e_{total}(t) &= P^e_{send}(t) + P^e_{process}(t) \\ &= \sum_{m=1}^{M} \sum_{n=1}^{N} d_{m,n}(t) p^e_{m,send} + \sum_{m=1}^{M} w^e_m(t) p^e_{m,process} \end{aligned} \tag{3.3}$$

- The energy consumption of each cloud is straightforward. For every cloud $n \in [N]$, the total energy consumption of cloud n, denoted as $P^c_{n,total}(t)$, is the energy consumed by processing tasks. Since we have the amount of energy consumed by processing single type-m task, denoted as $p^c_{m,n}$, and the number of type-m tasks processed by cloud n at time t, denoted as $w^c_{m,n}(t)$. Multiplying $p^c_{m,n}$ by $w^c_{m,n}(t)$ gives us the energy consumption of processing $w^c_{m,n}(t)$ number of type-m by cloud n at time t. The summation, $\sum_{m=1}^{M} w^c_{m,n}(t) p^c_{m,n}$, suggests cloud n's total processing consumption at time t, denoted as $P^c_{n,total}(t)$:

$$\forall n \in [N],\ P^c_{n,total}(t) = \sum_{m=1}^{M} w^c_{m,n}(t) p^c_{m,n} \tag{3.4}$$

3.3.3 Energy Constraints

Given the energy constraints of an edge server and each cloud, denoted by P^e and P^c_n respectively, our goal is to ensure that the total energy consumption of the edge server and each cloud n does not exceed the given energy constraint at a specific time t. It is important to note that these energy constraints will be reset at each time step. These constraints play a crucial role in defining the corresponding spaces of feasible actions: $d_{m,n}(t)$, $w^e_m(t)$, and $w^c_{m,n}(t)$, which represent the task distribution among the edge server and clouds, the workload of the edge server, and the workload of each cloud n, respectively.

$$P^e_{total}(t) \le P^e$$
$$\forall n \in [N],\ P^c_{n,total}(t) \le P^c_n$$

Substitute $P^e_{total}(t)$ and $P^c_{n,total}(t)$ by (3.3) and (3.4), we can see the actions $d_{m,n}(t)$, $w^e_m(t)$, and $w^c_{m,n}(t)$ are subject to the constraints:

$$\sum_{m=1}^{M} \sum_{n=1}^{N} d_{m,n}(t) p^e_{m,send} + \sum_{m=1}^{M} w^e_m(t) p^e_{m,process} \le P^e \tag{3.5}$$

$$\forall n \in [N], \quad \sum_{m=1}^{M} w^c_{m,n}(t) p^c_{m,n} \le P^c_n \tag{3.6}$$

3.3.4 Carbon Emissions

As we know, multiplying energy consumption and carbon intensity returns the carbon emissions. From the previous discussion, we already have the energy consumption; therefore, the only thing we need is the carbon intensities of the areas where edge server and clouds locate. Suppose $C^e(t)$ is the carbon intensity of edge server and $C^c_n(t)$ is the carbon intensity of cloud n at time t. Then $C^e(t) \cdot P^e_{total}(t)$ is the carbon emission of edge server at time t while $\sum_{n=1}^{N} P^c_{n,total}(t) C^c_n(t)$ is the summation of carbon emissions of all clouds. Consequently, the carbon emission of the whole computing network at time t, denoted as $C(t)$, can be shown as the following:

$$C(t) = C^e(t) P^e_{total}(t) + \sum_{n=1}^{N} P^c_{n,total}(t) C^c_n(t) \tag{3.7}$$

Time-Average Carbon Emissions

Given the carbon emissions $C(t)$ at time t, we can define the time-average carbon emissions $\bar{C}$ as the following:

$$\bar{C} = \limsup_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[C(t)]$$

Limit Supremum: Note that "lim sup" is short for "limit supremum". It is used to describe the limiting behavior of a sequence or a function as it approaches a certain point or goes to infinity. Figure 3.2 shows an example of the limits of the supremum and infimum. The x_n sequence is shown in blue. The limits supremum and infimum of x_n, shown as dashed black lines, are approached by the two red curves. The sequence in this instance builds up near the two limits. Between the two, the supremum limit is bigger and the infimum limit is smaller.

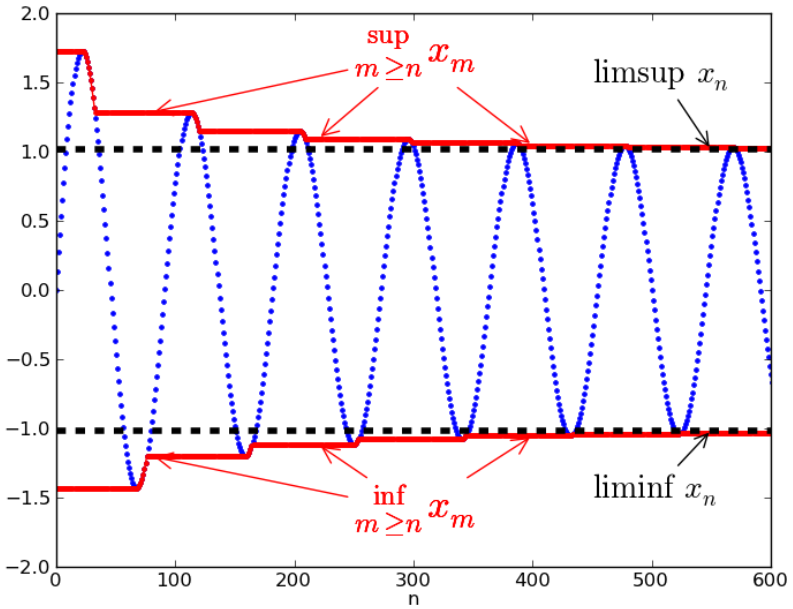

Figure 3.2: Limit infimum and limit supremum describe the limiting behavior of a sequence or a function as it approaches a certain point or goes to infinity [3]

3.3.5 Problem Formulation

Based on the aforementioned system, our primary objective is to design an effective scheduling policy that selects feasible actions at each time t in order to minimize the time-average carbon emissions. The scheduling policy will play a critical role in determining the optimal course of action for sending or processing tasks across the edge server and the various cloud resources. By focusing on reducing the time-average carbon emissions, we aim to achieve a more environmentally sustainable operation while maintaining the performance and efficiency of the system.

In order to achieve the desired objectives, the scheduling policy must efficiently determine the optimal allocation of tasks to the edge server and various clouds while considering the given energy constraints and decision variables. This ensures that the chosen actions are feasible and do not surpass the energy limitations of the edge server and each cloud at any given time t. Although the primary goal is to minimize time-average carbon emissions, it is crucial for the scheduling policy to strike a balance between environmental sustainability, system performance, and efficiency. This approach enables the system to effectively manage the assigned tasks while adhering to the target energy consumption and carbon emission levels.

Furthermore, to prevent task congestion or delays, the scheduling policy should emphasize maintaining queue length stability. This stability is vital for avoiding an excessive accumulation of tasks and ensuring that the system can process incoming tasks in a timely fashion. Queue length stability becomes particularly significant when dealing with variable workloads since fluctuations in task arrival rates may result in increased wait times and potential adverse effects on system performance.

Hence, this problem is stated as: minimize the time-average carbon emissions $\bar{C}$ given that each queue in edge and clouds are stable and the total energy consumptions of edge and clouds are limited by their constraints:

$$
\begin{aligned}
\min \quad & \bar{C} \\
\text{s.t.} \quad & \lim_{T\to\infty} \frac{\mathbb{E}[Q^e_m(T)]}{T} = 0,\ \forall m \in [M], \\
& \lim_{T\to\infty} \frac{\mathbb{E}[Q^c_{m,n}(T)]}{T} = 0,\ \forall m \in [M],\ \forall n \in [N], \\
& P^e_{total}(t) \le P^e, \\
& P^c_{n,total}(t) \le P^c_n,\ \forall n \in [N], \\
& d_{m,n}(t), w^e_m(t), w^c_{m,n}(t) \in \{0,1,2,\dots\},\ \forall m \in [M],\ \forall n \in [N]
\end{aligned}
\tag{3.8}
$$

By carefully considering these factors, the scheduling policy can effectively choose feasible actions at each time t, determining when and where to send or process tasks to achieve the lowest possible time-average carbon emissions while maintaining the queue stability. Ultimately, this will help create a more environmentally friendly system that operates efficiently and sustainably, contributing to a reduction in the overall carbon footprint.

3.4 Problem Reformulation

In last section, we constructed a discrete time system that changes throughout the normalized time slots $t \in \{0,1,2,\dots\}$. We point out that $C(t)$ is a function whose time average $\bar{C}$ should be minimized under time-average constraints and other functions. The drift-plus-penalty approach in Lyapunov optimization can therefore be used to tackle this problem.

Lyapunov Optimization is a powerful mathematical framework that was first introduced by the Russian mathematician Aleksandr Lyapunov in the late 19th century. It is widely used to

design and analyze control algorithms for complex dynamical systems. The drift-plus-penalty methodology is an integral part of Lyapunov optimization, a technique used for designing control policies that optimize the performance of dynamic systems over time. This methodology provides an effective framework for creating scheduling and resource allocation policies that balance system performance, queue length stability, and other objectives, such as minimizing carbon emissions.

3.4.1 Drift-Plus-Penalty Technique in Lyapunov Optimization

The core idea behind the drift-plus-penalty technique in Lyapunov Optimization is to combine the drift term, which represents the change in the Lyapunov function (a scalar measure of the system's current state), with a penalty term that captures the optimization objective, that is minimizing time-average carbon emissions. By minimizing the sum of the drift and penalty terms, the Lyapunov optimization framework aims to strike a balance between maintaining system stability and achieving the desired optimization objective.

Here is a step-by-step introduction to the drift-plus-penalty technique in Lyapunov optimization:

1. **Define the Lyapunov function:** Select a scalar function that captures the system's state, typically represented by the queue lengths. A common choice is the quadratic Lyapunov function, which is the sum of the squares of queue lengths. In this case, we have $Q^e_m(t)$ and $Q^c_{m,n}(t)$, the queue lengths of type-m tasks in edge server and in cloud n respectively. Then we can define the Lyapunov function $L(t)$ as the following:

$$L(t) = \frac{1}{2}\Big\{ \sum_{m=1}^{M} Q^e_m(t)^2 + \sum_{m=1}^{M}\sum_{n=1}^{N} Q^c_{m,n}(t)^2 \Big\} \tag{3.9}$$

2. **Calculate the drift term:** Determine the expected change in the Lyapunov function from time t to $t+1$, given the current state and control decisions. This drift term reflects the system's natural tendency to evolve over time and helps maintain stability. We are able to obtain the drift of Lyapunov function $\Delta(t)$ as:

$$\Delta(t) = L(t+1) - L(t)$$

3. **Define the penalty term:** Introduce a penalty term that represents the optimization objective, which is minimizing the carbon emissions $C(t)$. The penalty term should be designed in a way that encourages the control policy to achieve the desired goal. Recall $C(t)$ in section 3.3.4 is:

$$C(t) = C^e(t)P^e_{total}(t) + \sum_{n=1}^{N} P^c_{n,total}(t)C^c_n(t)$$

4. **Formulate the drift-plus-penalty expression:** Combine the drift and penalty terms to create a single expression that captures both system stability and the optimization objective. Suppose $V \geq 0$ is a weight that used to scale the penalty term, then the drift-plus-penalty expression is defined as:

$$\Delta(t) + V \cdot C(t)$$

5. **Minimize the drift-plus-penalty expression:** Instead of minimizing the drift-plus-penalty expression directly, we will minimize the upper bound of it. Then we are able to design a scheduling policy that minimizes the upper bound of drift-plus-penalty expression at each time step. The details of deriving the upper bound will be shown in section 3.4.2.

3.4.2 Upper Bound of Drift-Plus-Penalty Expression

Suppose the number of type-m tasks arriving at edge server, $a_m(t)$, is upper bounded, then we are able to derive the upper bound of the drift-plus-penalty expression.

According to the previous sections, we have:

$$\Delta(t) + V \cdot C(t) = L(t+1) - L(t) + V \cdot C(t) \tag{3.10}$$

To find the upper bound of the above expression, we start from $L(t+1)$. By equation (3.9), we have:

$$L(t+1) = \frac{1}{2}\Big\{ \sum_{m=1}^{M} Q_m^e(t+1)^2 + \sum_{m=1}^{M}\sum_{n=1}^{N} Q_{m,n}^c(t+1)^2 \Big\}$$
$$2L(t+1) = \sum_{m=1}^{M} Q_m^e(t+1)^2 + \sum_{m=1}^{M}\sum_{n=1}^{N} Q_{m,n}^c(t+1)^2$$

Next, substitute $Q_m^e(t+1)$ and $Q_{m,n}^c(t+1)$ by (3.1) and (3.2) respectively:

$$\begin{aligned} 2L(t+1) = &\sum_{m=1}^{M} \Big\{ \max(Q_m^e(t) - \sum_{n=1}^{N} d_{m,n}(t) - w_m^e(t), 0) + a_m(t) \Big\}^2 \\ &+ \sum_{m=1}^{M}\sum_{n=1}^{N} \Big\{ \max(Q_{m,n}^c(t) - w_{m,n}^c(t), 0) + d_{m,n}(t) \Big\}^2 \end{aligned}$$

Given the inequality $(\max(a-b,0)+c)^2 \le a^2 + b^2 + c^2 + 2a(c-b)$ for $a, b, c \ge 0$, we are able to find the upper bound of $2L(t+1)$ by considering $a = Q_m^e(t)$ or $Q_{m,n}^c(t)$, $b = \sum_{n=1}^{N} d_{m,n}(t) + w_m^e(t)$ or $w_{m,n}^c(t)$, and $c = a_m(t)$ or $d_{m,n}(t)$:

$$\begin{aligned} 2L(t+1) \le &\sum_{m=1}^{M} Q_m^e(t)^2 + \sum_{m=1}^{M} [\sum_{n=1}^{N} d_{m,n}(t) + w_m^e(t)]^2 + \sum_{m=1}^{M} a_m(t)^2 \\ &+ 2\sum_{m=1}^{M} Q_m^e(t)[a_m(t) - \sum_{n=1}^{N} d_{m,n}(t) - w_m^e(t)] \\ &+ \sum_{m=1}^{M}\sum_{n=1}^{N} Q_{m,n}^c(t)^2 + \sum_{m=1}^{M}\sum_{n=1}^{N} w_{m,n}^c(t)^2 + \sum_{m=1}^{M}\sum_{n=1}^{N} d_{m,n}(t)^2 \\ &+ 2\sum_{m=1}^{M}\sum_{n=1}^{N} Q_{m,n}^c(t)[d_{m,n}(t) - w_{m,n}^c(t)] \end{aligned} \tag{3.11}$$

Subtracting $2L(t) = \sum_{m=1}^{M} Q_m^e(t)^2 + \sum_{m=1}^{M}\sum_{n=1}^{N} Q_{m,n}^c(t)^2$ on both sides, we have:

$$\begin{aligned}
2L(t+1) - 2L(t) \leq & \cancel{\sum_{m=1}^{M} Q_m^e(t)^2} + \sum_{m=1}^{M}(\sum_{n=1}^{N} d_{m,n}(t) + w_m^e(t))^2 + \sum_{m=1}^{M} a_m(t)^2 \\
& + 2\sum_{m=1}^{M} Q_m^e(t)(a_m(t) - \sum_{n=1}^{N} d_{m,n}(t) - w_m^e(t)) \\
& + \cancel{\sum_{m=1}^{M}\sum_{n=1}^{N} Q_{m,n}^c(t)^2} + \sum_{m=1}^{M}\sum_{n=1}^{N} w_{m,n}^c(t)^2 + \sum_{m=1}^{M}\sum_{n=1}^{N} d_{m,n}(t)^2 \\
& + 2\sum_{m=1}^{M}\sum_{n=1}^{N} Q_{m,n}^c(t)(d_{m,n}(t) - w_{m,n}^c(t)) \\
& - (\cancel{\sum_{m=1}^{M} Q_m^e(t)^2 + \sum_{m=1}^{M}\sum_{n=1}^{N} Q_{m,n}^c(t)^2})
\end{aligned} \tag{3.12}$$

With the inequality (3.12), we now find the upper bound of the drift-plus-penalty (3.10):

$$\begin{aligned}
\Delta(t) + V \cdot C(t) &= L(t+1) - L(t) + V \cdot C(t) \\
&\leq \frac{1}{2}\sum_{m=1}^{M}(\sum_{n=1}^{N} d_{m,n}(t) + w_m^e(t))^2 + \frac{1}{2}\sum_{m=1}^{M} a_m(t)^2 \\
&\quad + \sum_{m=1}^{M} Q_m^e(t)(a_m(t) - \sum_{n=1}^{N} d_{m,n}(t) - w_m^e(t)) \\
&\quad + \frac{1}{2}\sum_{m=1}^{M}\sum_{n=1}^{N} w_{m,n}^c(t)^2 + \frac{1}{2}\sum_{m=1}^{M}\sum_{n=1}^{N} d_{m,n}(t)^2 \\
&\quad + \sum_{m=1}^{M}\sum_{n=1}^{N} Q_{m,n}^c(t)(d_{m,n}(t) - w_{m,n}^c(t)) \\
&\quad + V \cdot C(t)
\end{aligned} \tag{3.13}$$

Substitute $C(t)$ by equation (3.7), we have:

$$\begin{aligned}
\Delta(t) + V \cdot C(t) \leq & \frac{1}{2}\sum_{m=1}^{M}(\sum_{n=1}^{N} d_{m,n}(t) + w_m^e(t))^2 + \frac{1}{2}\sum_{m=1}^{M} a_m(t)^2 \\
& + \sum_{m=1}^{M} Q_m^e(t)(a_m(t) - \sum_{n=1}^{N} d_{m,n}(t) - w_m^e(t)) \\
& + \frac{1}{2}\sum_{m=1}^{M}\sum_{n=1}^{N} w_{m,n}^c(t)^2 + \frac{1}{2}\sum_{m=1}^{M}\sum_{n=1}^{N} d_{m,n}(t)^2 \\
& + \sum_{m=1}^{M}\sum_{n=1}^{N} Q_{m,n}^c(t)(d_{m,n}(t) - w_{m,n}^c(t)) \\
& + V(C^e(t)P_{total}^e(t) + \sum_{n=1}^{N} P_{n,total}^c(t)C_n^c(t))
\end{aligned} \tag{3.14}$$

Continue Substituting $P^e_{total}(t)$ and $P^c_{n,total}$ by (3.3) and (3.4):

$$\begin{aligned}\Delta(t)+V\cdot C(t) \leq & \frac{1}{2}\sum_{m=1}^{M}(\sum_{n=1}^{N} d_{m,n}(t)+w^e_m(t))^2+\frac{1}{2}\sum_{m=1}^{M} a_m(t)^2 \\ & +\sum_{m=1}^{M} Q^e_m(t)(a_m(t)-\sum_{n=1}^{N} d_{m,n}(t)-w^e_m(t)) \\ & +\frac{1}{2}\sum_{m=1}^{M}\sum_{n=1}^{N} w^c_{m,n}(t)^2+\frac{1}{2}\sum_{m=1}^{M}\sum_{n=1}^{N} d_{m,n}(t)^2 \\ & +\sum_{m=1}^{M}\sum_{n=1}^{N} Q^c_{m,n}(t)(d_{m,n}(t)-w^c_{m,n}(t)) \\ & +V\cdot C^e(t)(\sum_{m=1}^{M}\sum_{n=1}^{N} d_{m,n}(t)p^e_{m,send}+\sum_{m=1}^{M} w^e_m(t)p^e_{m,process}) \\ & +V\sum_{n=1}^{N} C^c_n(t)(\sum_{m=1}^{M} w^c_{m,n}(t)p^c_{m,n})\end{aligned} \tag{3.15}$$

Rearrange all the terms, we have:

$$\begin{aligned}\forall t,\quad \Delta(t)+V\cdot C(t) \leq B & +\sum_{m=1}^{M} Q^e_m(t)a_m(t) \\ & +\sum_{m=1}^{M}\sum_{n=1}^{N}(VC^e(t)p^e_{m,send}+Q^c_{m,n}(t)-Q^e_m(t))d_{m,n}(t) \\ & +\sum_{m=1}^{M}(VC^e(t)p^e_{m,process}-Q^e_m(t))w^e_m(t) \\ & +\sum_{m=1}^{M}\sum_{n=1}^{N}(VC^c_n(t)p^c_{m,n}-Q^c_{m,n}(t))w^c_{m,n}(t),\end{aligned} \tag{3.16}$$

where

$$A=\frac{1}{2}\Big\{\sum_{m=1}^{M}(\sum_{n=1}^{N} d_{m,n}(t)+w^e_m(t))^2+\sum_{m=1}^{M} a_m(t)^2+\sum_{m=1}^{M}\sum_{n=1}^{N} w^c_{m,n}(t)^2+\sum_{m=1}^{M}\sum_{n=1}^{N} d_{m,n}(t)^2\Big\}$$

.

Since $d_{m,n}(t)$, $w^e_m(t)$, and $w^c_{m,n}(t)$ are constrained by (3.5) and (3.6) while $a_m(t)$ is upper bounded, there must exist a constant B such that $A \leq B$. Therefore, inequality (3.16) holds.

3.4.3 Reformulated Problem

As mentioned before, we reformulated problem as minimizing the upper bound of drift-plus-penalty $\Delta(t)+V\cdot C(t)$ instead of minimizing it directly. Given the upper bound in (3.16), we summarize the following:

- The first term B is a constant;
- $Q^e_m(t)$ and $a_m(t)$ in the second term are observations of the edge queue length and the number of arriving tasks at each time step;
- In the remaining terms:

– V is a non-negative hyper-parameter;

– $C^e(t)$ and $C^c_n(t)$ are the observed carbon intensities of edge server and cloud n, where $n \in [N]$, at time step t;

– $p^e_{m,send}$, $p^e_{m,process}$, $p^c_{m,n}$ are the observed energy consumption of sending or processing single type-m task on edge server and cloud n.

– $Q^e_m(t)$ and $Q^c_{m,n}(t)$ are observed queue lengths of edge queue and cloud queues at time t.

– $d_{m,n}(t)$, $w^e_m(t)$, and $w^c_{m,n}(t)$ represent the number of type-m tasks sent to cloud n, processed by edge server, or processed by cloud n respectively. These are actions for our scheduling policy to determine at each time step.

Thus, at each time t, the goal of the scheduling policy is to find feasible actions $d_{m,n}(t)$, $w^e_m(t)$, and $w^c_{m,n}(t)$ that minimize the upper bound based on the observations. Since only the last three terms in the upper bound are directly affected by actions, minimizing the whole upper bound in (3.16) is equivalent to minimize the last three terms. Now the problem can be reformulated as:

$$\begin{aligned}
\min \quad & \sum_{m=1}^{M}\sum_{n=1}^{N}(VC^e(t)p^e_{m,send} + Q^c_{m,n}(t) - Q^e_m(t))d_{m,n}(t) \\
& + \sum_{m=1}^{M}(VC^e(t)p^e_{m,process} - Q^e_m(t))w^e_m(t) \\
& + \sum_{m=1}^{M}\sum_{n=1}^{N}(VC^c_n(t)p^c_{m,n} - Q^c_{m,n}(t))w^c_{m,n}(t) \\
\text{s.t.} \quad & \sum_{m=1}^{M}\sum_{n=1}^{N} d_{m,n}(t)p^e_{m,send} + \sum_{m=1}^{M} w^e_m(t)p^e_{m,process} \le P^e \\
& \sum_{m=1}^{M} w^c_{m,n}(t)p^c_{m,n} \le P^c_n, \quad \forall n \in [N] \\
& d_{m,n}(t), w^e_m(t), w^c_{m,n}(t) \in \{0,1,2,\dots\},\ \forall m \in [M],\ \forall n \in [N]
\end{aligned} \tag{3.17}$$

Note that the problem in (3.17) can be split into $N+1$ independent optimization problems as edge servers and N clouds are subject to distinct and independent constraints.

Firstly, suppose $r_{m,n}(t) = VC^e(t)p^e_{m,send} + Q^c_{m,n}(t) - Q^e_m(t)$ and $h_m(t) = VC^e(t)p^e_{m,process} - Q^e_m(t)$, the optimization problem for edge server can be defined as:

$$\begin{aligned}
\min \quad & \sum_{m=1}^{M}\sum_{n=1}^{N} r_{m,n}(t)d_{m,n}(t) \\
& + \sum_{m=1}^{M}\sum_{n=1}^{N} h_m(t)w^e_m(t) \\
\text{s.t.} \quad & \sum_{m=1}^{M}\sum_{n=1}^{N} d_{m,n}(t)p^e_{m,send} + \sum_{m=1}^{M} w^e_m(t)p^e_{m,process} \le P^e \\
& d_{m,n}(t), w^e_m(t) \in \{0,1,2,\dots\},\ \forall m \in [M],\ \forall n \in [N]
\end{aligned} \tag{3.18}$$

The optimal solution of problem (3.18) requires:

- if $r_{m,n}(t) > 0$, then $d_{m,n}(t) = 0$;

- if $h_m(t) > 0$, then $w^e_m(t) = 0$;
- if $r_{m,n}(t) < h_m(t) < 0$, then $w^e_m(t) = 0$;
- if $h_m(t) < r_{m,n}(t) < 0$, then $d_{m,n}(t) = 0$.

Secondly, let $k_{m,n}(t) = VC^c_n(t)p^c_{m,n} - Q^c_{m,n}(t)$, The optimization problem for each cloud $n \in [N]$ can be defined as:

$$\begin{aligned} \min \quad & \sum_{m=1}^{M} k_{m,n}(t) w^c_{m,n}(t) \\ \text{s.t.} \quad & \sum_{m=1}^{M} w^c_{m,n}(t) p^c_{m,n} \leq P^c_n, \quad \forall n \in [N] \\ & w^c_{m,n}(t) \in \{0,1,2,\dots\}, \ \forall m \in [M], \ \forall n \in [N] \end{aligned} \tag{3.19}$$

The optimal solution of problem (3.19) requires:

- if $k_{m,n}(t) > 0$, then $w^c_{m,n}(t) = 0$.

3.5 Improved Carbon-Aware Task Scheduling Policy

By leveraging the drift-plus-penalty methodology in Lyapunov optimization, it is possible to design scheduling policies that strike a balance between system stability and optimization objectives.

Based on (3.18) and (3.19), we propose a carbon-aware task scheduling policy. This policy can be considered a two phases: one is in edge server and the other is in clouds. The descriptions and details about the two phases in this policy will be shown in the section 3.5.1 and 3.5.2 respectively while the complete policy is shown in section 3.5.3.

Here are the notation details for the later on algorithms:

- V: a non-negative hyper-parameter used to scale the penalty term;
- M: the number of task types;
- N: the number of clouds;

$\forall t \in \{0,1,2,\dots\}, \forall m \in [M], \forall n \in [N]$:

- $a(t)$: $M \times 1$ matrix such that $a(t)[m]$ the number of type-m tasks arrived at edge server at time t;
- $C^e(t)$: carbon intensity at the edge server location at time t;
- $C^c(t)$: $N \times 1$ matrix such that $C^c(t)[n]$ is the carbon intensity at cloud n location at time t;
- $d(t)$: $M \times N$ matrix such that $d(t)[m][n]$ is the number of type-m tasks transmitted to cloud n at time t;
- P^c: $N \times 1$ matrix such that $P^c[n]$ is the energy constraint of cloud n;
- p^c: $M \times N$ matrix such that $p^c[m][n]$ is the energy consumption of processing a single type-m tasks on cloud n;
- P^e: the energy constraint of edge server;

- $p^e_{process}$: $M \times 1$ matrix such that $p^e_{process}[m]$: the energy consumption of processing a single type-m tasks by edge server;
- p^e_{send}: $M \times 1$ matrix such that $p^e_{send}[m]$ is the energy consumption of sending a single type-m tasks to any of the clouds;
- $Q^c(t)$: $M \times N$ matrix where $Q^c(t)[m][n]$ is the queue length of type-m tasks at cloud n;
- $Q^e(t)$: $M \times 1$ matrix where $Q^e(t)[m]$ is the queue length of type-m tasks at edge server;
- $w^c(t)$: $M \times N$ matrix such that $w^c(t)[m][n]$ is the number of type-m tasks processed by cloud n at time t;
- $w^e(t)$: $M \times 1$ matrix such that $w^e(t)[m]$ is the number of type-m tasks processed by edge server at time t;

3.5.1 Phase 1: Carbon-Aware Scheduling Policy For Edge Server

At each time t, tasks will be transmitted or processed by edge server in the manner described below in algorithm 1:

1. For each type of tasks m, find a cloud n' such that $r_{m,n'}(t)$ is the smallest among all clouds N.
2. Sort the task types in the increasing order of $\frac{r_{m,n'}(t)}{p^e_{m,send}}$. That is, task types contribute the smallest $r_{m,n'}(t)$ value to per energy unit will be scheduled first.
3. If $r_{m,n'}(t) < h_m(t) < 0$, then schedule as many as possible type-m tasks to cloud n' given the energy constraint P^e.
4. If $h_m(t) < r_{m,n'}(t) < 0$, then process as many as possible type-m tasks given the energy constraint P^e.
5. Greedily schedule or process tasks by the order in step 2 while following the rules in step 3 and 4 until the remaining P^e is not enough to transmit or process a single task.

3.5.2 Phase 2: Carbon-Aware Scheduling Policy For Clouds

For each cloud $n \in [N]$, tasks will be processed at each time t in the manner described below in algorithm 2:

1. Sort the task types in the increasing order of $\frac{k_{m,n}(t)}{p^c_{m,n}}$. That is, task types contribute the smallest $k_{m,n}(t)$ value to per energy unit will be processed by cloud n first.
2. Cloud n processes as many as possible type-m tasks given the energy constraint $P^c_{m,n}$.
3. Greedily process tasks by the order in step 1 until the remaining $P^c_{m,n}$ is not enough to process a single task.

3.5.3 Complete Carbon-aware Scheduling Policy

Combining algorithm 1 and 2 together, the complete carbon-aware scheduling policy for the whole computing network is shown below in algorithm 3:

Algorithm 1 Improved Carbon-Aware Scheduling Policy For Edge Server

Input: $t, V, M, N, p^e_{send}, p^e_{process}, P^e, a(t), Q^e(t), Q^c(t), C^e(t)$;
Initialization: $d(t) \leftarrow [0]_{M\times N}$, $w^e(t) \leftarrow [0]_{M\times 1}$;
$\forall m$, find n' s.t. $Q^c[m][n']$ is the smallest;
Compute $r(t)[m] \leftarrow V \cdot C^e(t) \cdot p^e_{send}[m] + Q^c(t)[m][n'] - Q^e(t)[m]$, $\forall m$
Compute $h(t)[m] \leftarrow V \cdot C^e(t) \cdot p^e_{process}[m] - Q^e(t)[m]$, $\forall m$;
$order \leftarrow$ sort task types m in the increasing order of $\frac{r(t)[m]}{p^e_{send}[m]}$;
$P \leftarrow P^e$;
for $m \in order$ **do**
  if $\lfloor \frac{P}{p^e_{send}[m]} \rfloor > 0$ and $\lfloor \frac{P}{p^e_{process}[m]} \rfloor > 0$ **then**
    if $h(t)[m] < 0$ **then**
      if $h(t)[m] < r(t)[m]$ **then**
        $w^e(t)[m] \leftarrow \min(Q^e(t)[m], \lfloor \frac{P}{p^e_{process}[m]} \rfloor)$;
        $P \leftarrow P - w^e(t)[m] \cdot p^e_{process}[m]$;
      else
        $d(t)[m][n'] \leftarrow \min(Q^e(t)[m], \lfloor \frac{P}{p^e_{send}[m]} \rfloor)$;
        $P \leftarrow P - d(t)[m][n'] \cdot p^e_{send}[m]$;
      end if
    else
      if $r(t)[m] < 0$ **then**
        $d(t)[m][n'] \leftarrow \min(Q^e(t)[m], \lfloor \frac{P}{p^e_{send}[m]} \rfloor)$;
        $P \leftarrow P - d(t)[m][n'] \cdot p^e_{send}[m]$;
      end if
    end if
  else if $\lfloor \frac{P}{p^e_{send}[m]} \rfloor > 0$ **then**
    $d(t)[m][n'] \leftarrow \min(Q^e(t)[m], \lfloor \frac{P}{p^e_{send}[m]} \rfloor)$;
    $P \leftarrow P - d(t)[m][n'] \cdot p^e_{send}[m]$;
  else if $\lfloor \frac{P}{p^e_{process}[m]} \rfloor > 0$ **then**
    $w^e(t)[m] \leftarrow \min(Q^e(t)[m], \lfloor \frac{P}{p^e_{process}[m]} \rfloor)$;
    $P \leftarrow P - w^e(t)[m] \cdot p^e_{process}[m]$;
  else
    break;
  end if
end for

Algorithm 2 Improved Carbon-Aware Scheduling Policy For Clouds

Input: $t, V, M, N, p^c, P^c, Q^c(t), C^e(t)$;
Initialization: $w^c(t) \leftarrow [0]_{M\times N}$;
for $n \leftarrow 1$ **to** N **do**
  Compute $k(t)[m][n] \leftarrow V \cdot C^c(t)[n] \cdot p^c[m][n] - Q^c(t)[m][n], \forall m$;
  $order \leftarrow$ sort task types m in the increasing order of $\frac{k(t)[m][n]}{p^c[m][n]}$;
  $P \leftarrow P^c[n]$;
  for $m \in order$ **do**
    if $\lfloor \frac{P}{p^c[m][n]} \rfloor > 0$ **then**
      if $k(t)[m][n] < 0$ **then**
        $w^c(t)[m][n] \leftarrow \min(Q^c(t)[m][n], \lfloor \frac{P}{p^c[m][n]} \rfloor$;
        $P \leftarrow P - w^c(t)[m][n] \cdot p^c[m][n]$;
      else
        break;
      end if
    end if
  end for
end for

Algorithm 3 Improved Carbon-Aware Scheduling Policy

Input: $V, M, N, p^e_{send}, p^e_{process}, p^c, P^e, P^c$
for $t \leftarrow 0, 1, 2, \ldots$ **do**
  Observation: $a(t)$, $Q^e(t)$, $Q^c(t)$, $C^e(t)$, and $C^c(t)$;
  $d(t)[m][n]$, $w^e(t)[m]$, $\leftarrow$ Schedule tasks on edge server with algorithm 1;
  $w^c(t)[m][n] \leftarrow$ Schedule tasks on clouds with algorithm 2;
  Update $Q^e(t+1)[m] \leftarrow \max(Q^e(t)[m] - \sum_{n=1}^{N} d(t)[m][n] - w^e(t)[m], 0) + a(t)[m]$ ▷ by eq. (3.1);
  Update $Q^c(t+1)[m][n] \leftarrow \max(Q^c(t)[m][n] - w^c(t)[m][n], 0) + d(t)[m][n]$ ▷ by eq. (3.2)
end for

Chapter 4

Experiment Design

In this section, we will describe the environment configurations, investigate two representative datasets, and introduce two baseline models: the queue-aware model and the original carbon-aware model proposed by Yang and his team [6]. Environment configuration will be discussed in section 4.1. Then we analyze two different datasets in section 4.2 to investigate the situations when edge server and clouds are geographically distributed in different locations. Two baseline models that our proposed model will compare to will be introduced in section 4.3.

4.1 Environment Configuration

Inn this section, we introduce the environment configuration, including M, N, P^e, P^c_n, p^e_{send}, $p^e_{process}$, and $p^c_{m,n}$ for our simulation studies below:

- The computing network in our study consists of 1 edge server and $N = 5$ clouds;
- Suppose all clouds are identical;
- The energy constraint of edge server P^e is 30000 kW·h;
- The energy constraint of each cloud n P^c_n is 15000 kW·h, where $n \in \{1,2,3,4,5\}$ in this case;
- Suppose there are $M = 5$ types of tasks and the types refer to different AI training models on ImageNet [6]. The transmission and computation consumptions (p^e_{send}, $p^e_{process}$, and $p^c_{m,n}$) are shown in table 4.1:

Type (m)	Model	p^e_{send}	$p^e_{process}$ / $p^c_{m,n}$
1	ResNet50	3.45	74
2	InceptionV3	3.45	97
3	DenseNet121	3.45	54
4	SqueezeNet	3.45	16
5	MobileNetV2	3.45	5.8

Table 4.1: The transmission and computation consumptions of different types of tasks. Assume that edge server and clouds have the same computation consumptions and all clouds are identical.

4.2 Investigation of Carbon Intensity Datasets

The two distinct datasets discussed in this section are able to simulate two different scenarios for the location distribution of edge server and clouds. This configuration enables a closer examination of location factors. The second scenario involves edge servers and clouds distributed across different countries. Understanding these two distinct scenarios can help inform more sustainable and efficient strategies for the deployment of edge servers and cloud infrastructure on both local and global scales. Be aware that we only utilized the ESO dataset for this experiment, and we seek to further investigate our model with the second dataset in the future.

4.2.1 Dataset #1: National Grid ESO - Regional Carbon Intensity Forecast

Regional carbon intensity forecasts dataset generated by the National Grid Electricity System Operator (ESO) forecasts carbon intensity across 14 different regions in the Great British using machine learning models and metered generation [14]. This dataset contains detailed data on the regional carbon intensity, a measure of how much CO2 emissions are produced per kilowatt hour of electricity consumed, ranging from near real-time to forecasts up to two days ahead. Key features of the dataset include:

- **Geographic scope:** This dataset covers all regions within the Great British, providing a comprehensive overview of the country's carbon intensity of electricity landscape.
- **Temporal granularity:** This dataset includes carbon intensity forecasts every 30 minutes, allowing for detailed analysis of fluctuations and trends.

Figure 4.1 shows partial of the dataset as an example:

datetime	North Scotland	South Scotland	North West England	North East England	Yorkshire	North Wales and Merseyside	South Wales	West Midlands	East Midlands	East England	South West England	South England	London	South East England
2022-01-01 00:00:00	0.0	28.0	16.0	31.0	68.0	63.0	98.0	39.0	95.0	87.0	71.0	131.0	99.0	113.0
2022-01-01 00:30:00	0.0	28.0	16.0	33.0	67.0	72.0	97.0	44.0	95.0	89.0	75.0	126.0	98.0	112.0
2022-01-01 01:00:00	0.0	26.0	16.0	31.0	67.0	92.0	106.0	54.0	91.0	81.0	76.0	123.0	94.0	120.0
2022-01-01 01:30:00	0.0	26.0	17.0	31.0	66.0	84.0	97.0	50.0	85.0	31.0	68.0	119.0	60.0	101.0
2022-01-01 02:00:00	0.0	26.0	20.0	31.0	72.0	93.0	112.0	56.0	97.0	35.0	81.0	140.0	75.0	123.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 4.1: Partial of regional carbon intensity forecasts dataset includes 14 regions in Great Britain and records the carbon intensity of each region every 30 minutes.

In the analysis of the time series of regional carbon intensity data as shown in Figure 4.2, each time step represents a 30-minute interval, providing a granular view of how carbon intensity varies with both time and region. Throughout these fluctuations, East Midlands consistently exhibits the highest carbon intensity, making it a key area of focus for potential improvements in emissions reduction. Conversely, North East England consistently demonstrates the lowest carbon intensity for the majority of the time, which could be attributed to a higher proportion of low-carbon energy sources in its generation mix.

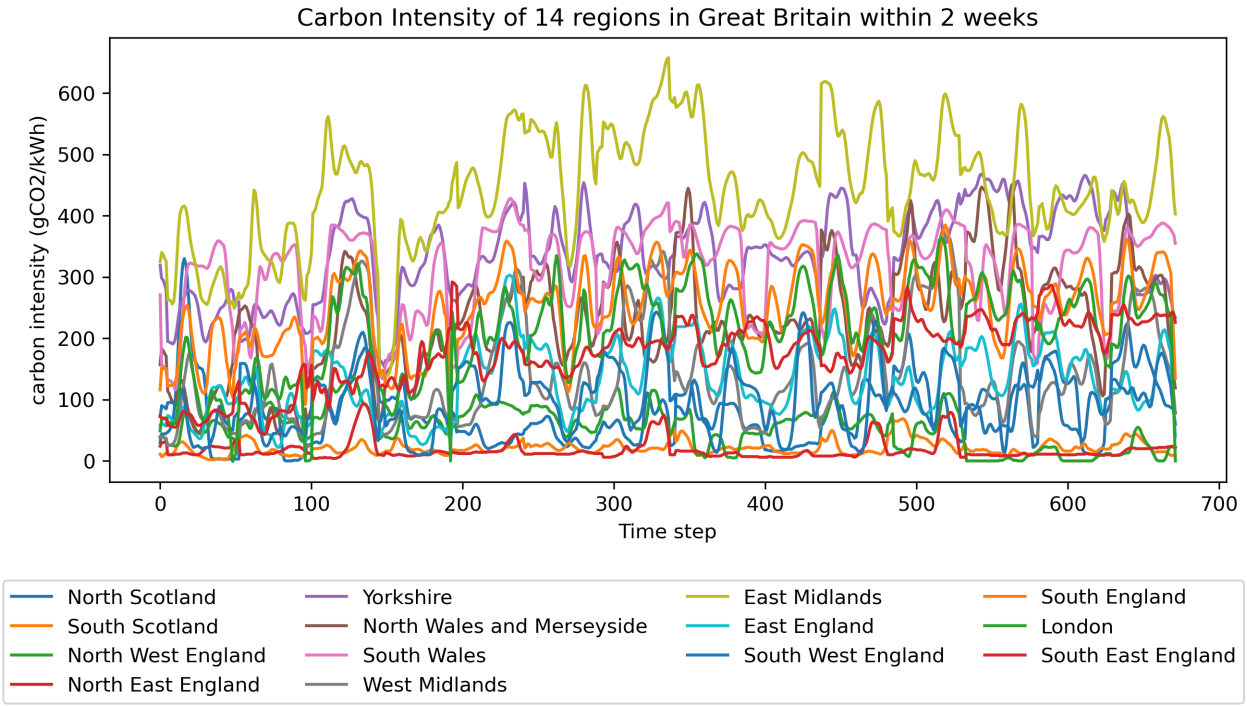

Figure 4.2: Biweekly time series of carbon intensity in 14 regions in Great Britain shows that East Midlands consistently exhibits the highest carbon intensity while North East England consistently demonstrates the lowest carbon intensity for the majority of the time.

To study the relation between time and carbon intensity in more detail, we randomly pick one week's carbon intensity and aggregate the carbon intensity every 7.5 hours. Then plot the time series of the selected data as shown in Figure 4.3. As we can see, in most regions, the time series data reveal a similar periodic pattern, reflecting the fluctuating demand for electricity. Notably, carbon intensity tends to be relatively low during midnight hours of the day, from 00:00 to 07:30. This can be attributed to lower electricity demand during these hours. Consequently, these patterns verifies the feasibility of the carbon-aware scheduling policy.

4.2.2 Dataset #2: ENTSO-E Transparency Platform and California ISO

This dataset was created by Wiesner and his team, as documented in their 2021 publication [15]. The researchers made use of energy production and carbon intensity datasets available on the ENTSO-E Transparency Platform, as well as those from the California Independent System Operator (ISO) to generate this dataset. With such information, they developed a script that computed the carbon intensity for four distinct regions: Germany, Great Britain, France, and California. The dataset encompasses the entire year of 2020, with an additional buffer of 10 days on either end. Note that the data is also recored every 30 minutes in this dataset.

Figure 4.4 shows an example of the dataset:

Based on Figure 4.5, the analysis of carbon intensity across the four regions: Germany, Great Britain, France, and California, reveals some interesting observations. France consistently demonstrates the lowest carbon intensity compared to other 3 regions, likely attributable to the country's significant reliance on nuclear power for electricity generation according to International Energy Agency (IEA) [16]. California exhibits a strong periodic pattern in its carbon intensity while the pattern in Germany is less clear. Great Britain and California seems have the opposite periodic pattern at most time.

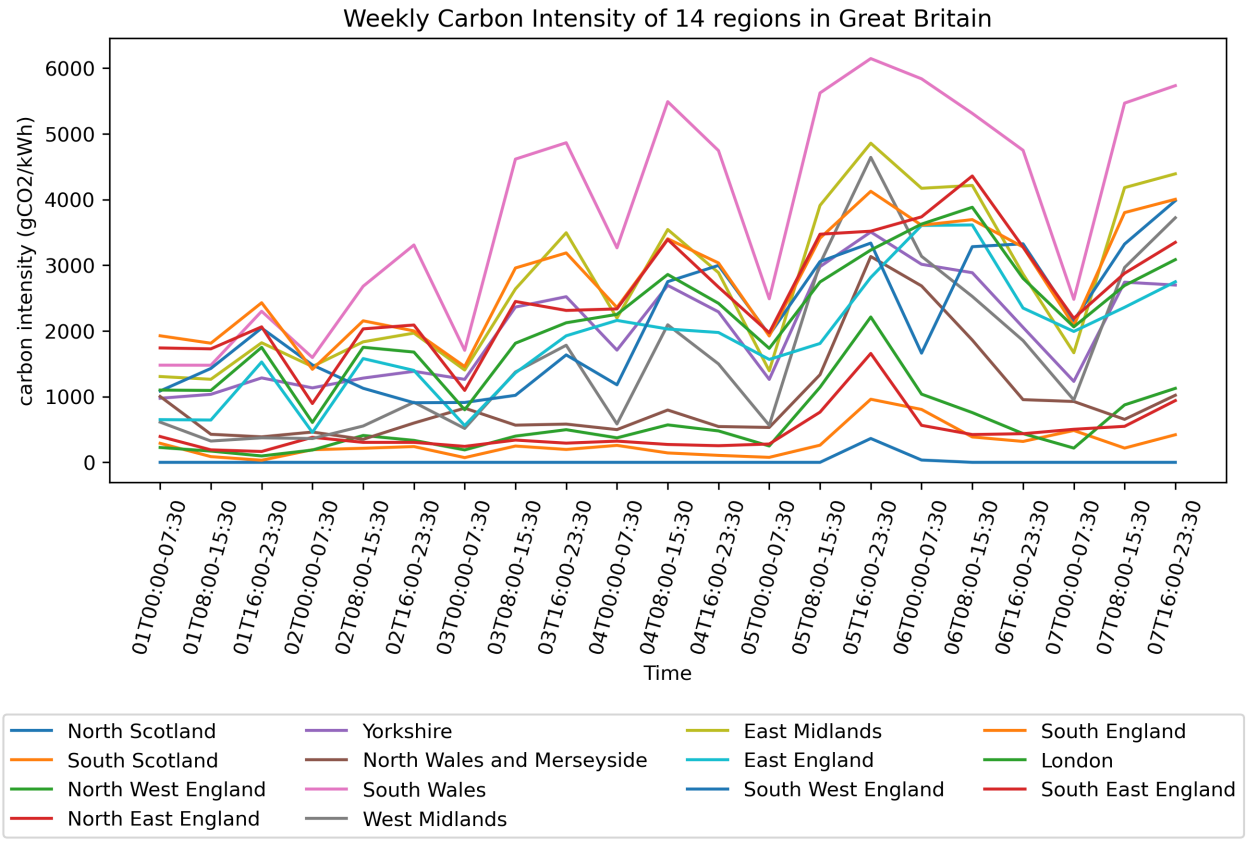

Figure 4.3: Weekly carbon intensity aggregated per 7.5h of 14 regions in Great Britain shows that the time series data has a similar periodic pattern in most regions and carbon intensity is low during 00:00 to 07:30.

Time	cal_ci	fr_ci	gb_ci	ger_ci
2019-12-20 00:00:00	349.734382	42.341238	120.995467	218.290031
2019-12-20 00:30:00	347.238271	42.101656	135.088696	217.494426
2019-12-20 01:00:00	346.218074	41.997517	141.357058	216.761357
2019-12-20 01:30:00	346.432492	42.016643	144.691745	215.191938
2019-12-20 02:00:00	342.529566	42.146853	141.128537	212.392223
...	...	...	...	...

Figure 4.4: Partial of the dataset #2 includes 4 countries and records the carbon intensity of each region every 30 minutes.

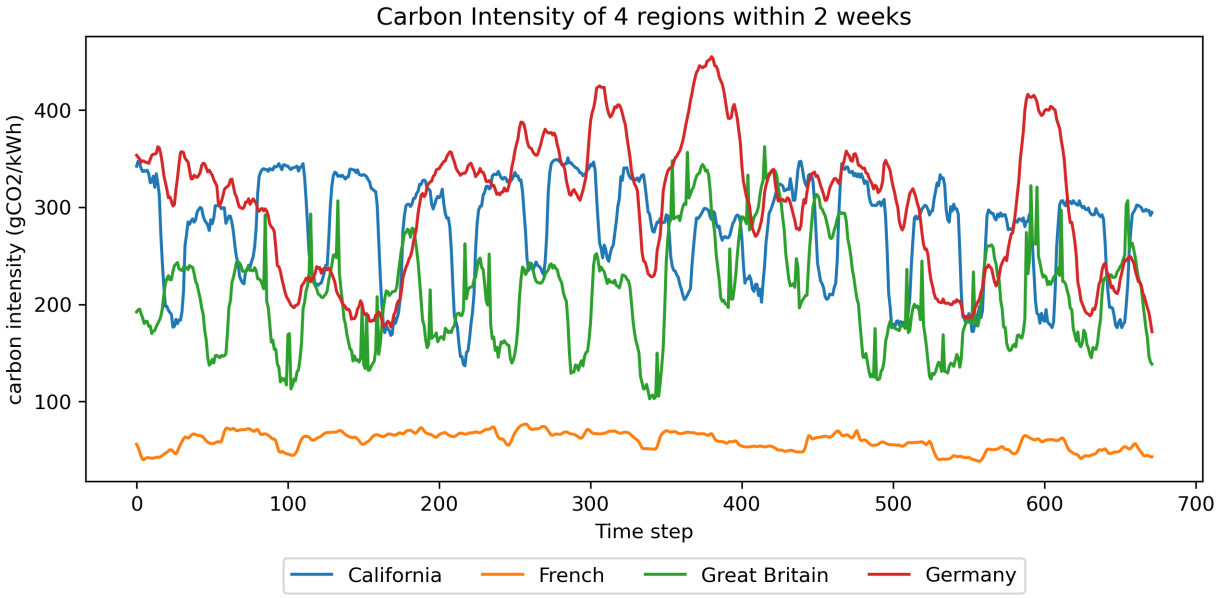

Figure 4.5: Biweekly time series of carbon intensity of 4 countries indicates that (1) France consistently has the lowest carbon intensity; (2) the pattern of Germany is not clear; (3) California has a strong periodic pattern that is opposite to the pattern of Great Britain.

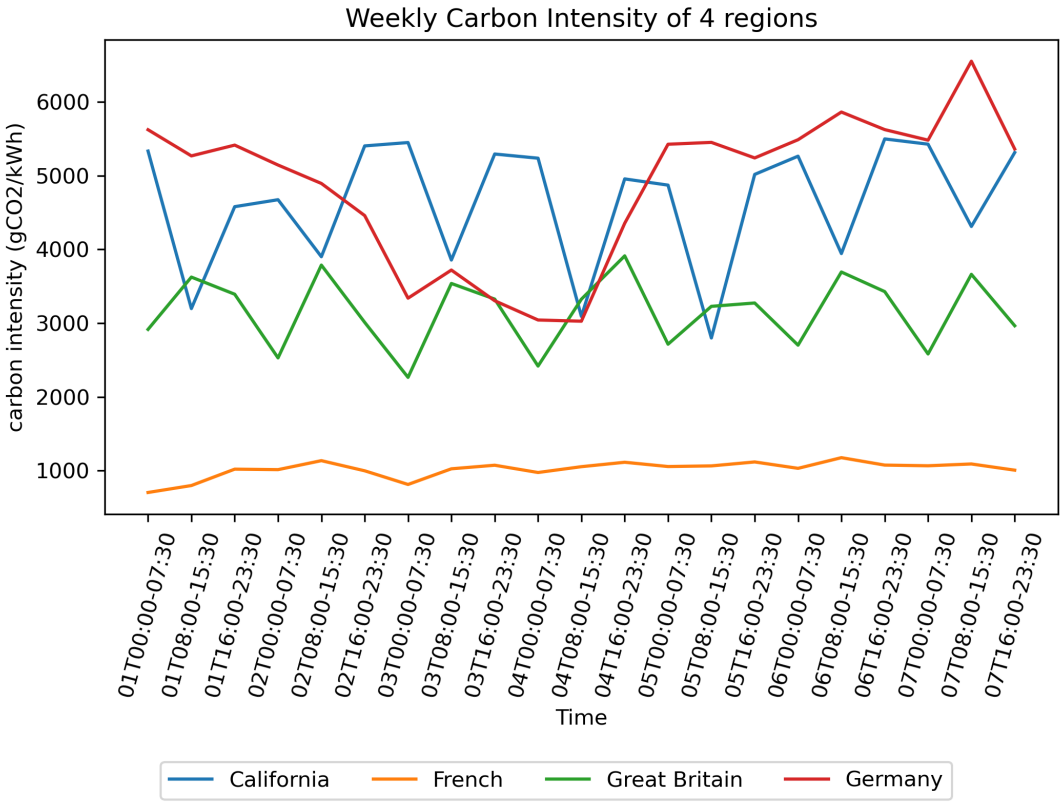

Figure 4.6: Weekly carbon intensity aggregated every 7.5 hours of 4 countires verifies (1) the lowest carbon intensity of Great Britain occurs during 00:00-07:30; (2) California and Great Britain have opposite periodic pattern possibly due to time zone difference.

Similarly, we randomly pick one week's carbon intensity and aggregate the carbon intensity every 7.5 hours. Then plot the time series of the selected data as shown in Figure 4.6. Firstly, the pattern of Great Britain also shows that the lowest carbon intensity mostly occurs in the period of 00:00-07:30, which verifies our observation in the ESO dataset. Secondly, the opposite periodic patterns of carbon intensity in California and Great Britain shows more clearly in this figure. For example, the lowest carbon intensity in California is during 08:00-15:30 which is the period that has the highest carbon intensity in Great Britain and vice versa. This phenomenon may be attributed to time zone differences. Since California and Great Britain have an 8-hour difference between them. As a result, peak and off-peak electricity demand hours occur at different times in the two regions, causing a shift in the carbon intensity patterns when compared side-by-side.

4.3 Baseline Models

In this section, we will go into detail about the baseline models for evaluating our proposed solution. A thorough understanding of the baseline models is essential for contextualizing the performance evaluation and highlighting the unique contributions of our proposed model. Section 4.3.1 illustrates the queue-aware model while section 4.3.2 focuses on the primary carbon-aware model proposed by Yang [6].

4.3.1 Queue-Aware Model

The main idea of the queue-aware model is to greedily transmit tasks in the longest edge queue to the shortest cloud queue and process tasks in the longest queue in each cloud first. The details of the queue-aware scheduling policy for this model is shown in algorithm 4.

4.3.2 Carbon-Aware Baseline Model

This model is proposed by Yang and his team [6]. As a heuristic approach, their model treats edge servers as a component that schedules and transmits tasks only. However, it may inadvertently contribute to increased transmission costs under certain circumstances. In particular, when transmission costs are exceptionally high, it might prove more efficient and cost-effective to process tasks directly on the edge server. This consideration underscores the need for a more adaptive model that can recognize and respond to such situations, minimizing unnecessary transmission expenses while maintaining the network stability. Since this model is the fundamental base of our proposed model (the improved carbon-aware model), they have similar policies. Recall that our improved carbon-aware policy has two phases. Only the first phase of these two models are different and we provide the details about first phase of this baseline model below in algorithm 5 for your reference.

Algorithm 4 Queue-Aware Scheduling Policy

Input: $V, M, N, p^e_{send}, p^c, P^e, P^c$
for $t \leftarrow 0,1,2,\ldots$ **do**
  Observation: $a(t)$, $Q^e(t)$, $Q^c(t)$, $C^e(t)$, and $C^c(t)$;
  Set $d(t) \leftarrow [0]_{M\times N}$ and $w^c(t) \leftarrow [0]_{M\times 1}$
  order $\leftarrow$ task types m in the decreasing order of $Q^e(t)[m]$;
  $P \leftarrow P^e$;
  for $m \in order$ **do**
    Find $n' \in [N]$ such that $Q^c(t)[m][n']$ is the smallest;
    if $\lfloor \frac{P}{p^e_{send}[m]} \rfloor > 0$ **then**
      $d(t)[m][n'] \leftarrow \min(Q^e(t)[m], \lfloor \frac{P}{p^e_{send}[m]} \rfloor$;
      $P \leftarrow P - d(t)[m][n'] \cdot p^e_{send}[m]$;
    else
      break;
    end if
  end for
  for $n \leftarrow 1$ **to** N **do**
    order $\leftarrow$ task types m in the decreasing order of $Q^c(t)[m][n]$;
    for $m \in order$ **do**
      if $\lfloor \frac{P}{p^c[m][n]} \rfloor > 0$ **then**
        $w^c(t)[m][n] \leftarrow \min(Q^c(t)[m][n], \lfloor \frac{P}{p^c[m][n]} \rfloor$;
        $P \leftarrow P - w^c(t)[m][n'] \cdot p^c[m][n]$;
      else
        break;
      end if
    end for
  end for
  Update $Q^e(t+1)[m] \leftarrow \max(Q^e(t)[m] - \sum_{n=1}^{N} d(t)[m][n], 0) + a(t)[m]$ ▷ by eq. (3.1);
  Update $Q^c(t+1)[m][n] \leftarrow \max(Q^c(t)[m][n] - w^c(t)[m][n], 0) + d(t)[m][n]$ ▷ by eq. (3.2)
end for

Algorithm 5 Baseline Carbon-Aware Scheduling Policy For Edge Server

Input: $t, V, M, N, p^e_{send}, P^e, a(t), Q^e(t), Q^c(t), C^e(t)$;
Initialization: $d(t) \leftarrow [0]_{M\times N}$, $w^e(t) \leftarrow [0]_{M\times 1}$;
$\forall m$, find n' s.t. $Q^c[m][n']$ is the smallest;
Compute $r(t)[m] \leftarrow V \cdot C^e(t) \cdot p^e_{send}[m] + Q^c(t)[m][n'] - Q^e(t)[m], \forall m$
$order \leftarrow$ sort task types m in the increasing order of $\frac{r(t)[m]}{p^e_{send}[m]}$;
$P \leftarrow P^e$;
for $m \in order$ **do**
  if $\lfloor \frac{P}{p^e_{send}[m]} \rfloor > 0$ **then**
    if $r(t)[m] < 0$ **then**
      $d(t)[m][n] \leftarrow \min(Q^e(t)[m], \lfloor \frac{P}{p^e_{send}[m]} \rfloor)$;
      $P \leftarrow P - d(t)[m][n] \cdot p^e_{send}[m]$;
    else
      break;
    end if
  end if
end for

Chapter 5

Performance Evaluation

After the carefully designed experimental setup in last chapter, we are able to present a comprehensive performance evaluation of the proposed model in this chapter. By comparing the proposed model with the baseline models, we will highlight its strengths and identify any potential areas for improvement. The evaluation metrics will be described in section 5.1, providing valuable insights into the model's real-world applicability and potential for adoption in network computing environments.

5.1 Evaluation Methodology

To assess the performance of our proposed model, we will employ a comprehensive evaluation methodology that takes into account 3 aspects of performance. Primarily, we will investigate the trade-off between carbon emissions and queue lengths by tuning the hyper-parameter V in section 5.2. After finding the best value of V, we will analyze the carbon savings by comparing our proposed model with two baseline models (queue-aware model and carbon-aware baseline model) in section 5.3. This comparison will be conducted on both monthly and yearly basis, allowing us to obtain a clearer understanding of the long-term environmental impact of our model. Additionally, to ensure the network stability, we will examine the mean-rate stability of queues in both edge server and clouds across all models in section 5.4. By analyzing these factors, we aim to provide an overall evaluation of our proposed model, highlighting its potential for minimizing carbon emissions while maintaining the stability of the computing network.

5.2 Trade-off between Carbon Savings and Queue Lengths

Tuning the hyper-parameter V shows the trade-off between carbon savings and queue lengths. Figure 5.1 shows (a) the monthly carbon emissions and (b) the queue lengths at edge server for our improved carbon-aware model with different settings of V. As V gets larger, the carbon emissions of the computing network reduce while more tasks are accumulated in the edge queue. For example, although the carbon emissions at $V = 0.5$ is the lowest, the corresponding queue at edge server is the longest. We notice that $V = 0.005$ is the best choice among the 4 values since the emissions significantly reduces compared to $V = 0.0005$ while the edge queue slightly increases.

This phenomenon can be explained by the drift-plus-penalty expression: $\Delta(t) + V \cdot C(t)$, where $\Delta(t)$ is the Lypunov function that used to measure the congestion of queues and $C(t)$ measures the carbon emissions. Increasing V indicates weight more on carbon emissions; that is, the goal leans towards reducing more carbon emissions allowing more congested queues and vice versa.

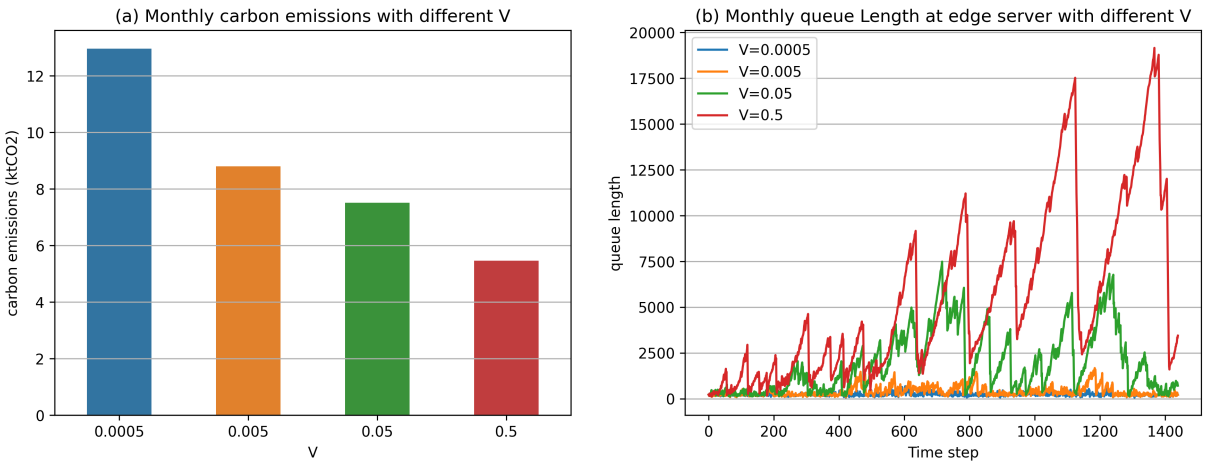

Figure 5.1: (a) monthly carbon emissions with $V = 0.0005, 0.005, 0.05, 0.5$ respectively; (b) monthly queue lengths at edge server with $V = 0.0005, 0.005, 0.05, 0.5$ respectively.

5.3 Carbon Savings Comparison

As explained in section 5.2, $V = 0.005$ is our best setting. Thus, both the carbon-aware baseline and improved models will set V to be 0.005 in later sections. Now, we first look at the time series of carbon emissions within 3 days as shown in Figure 5.2. Carbon-aware improved policy as green line has the least carbon emissions at majority of time. Although the carbon emissions of carbon-aware baseline policy exceeds that of the queue-aware policy at some time steps, the line of the former shown in orange falls below the blue line for the latter by looking at the whole picture.

According to the monthly and yearly carbon emissions in Figure 5.3, the carbon-aware improved policy demonstrates remarkable carbon emission savings in both monthly and yearly comparisons. Within a month, this policy saves 45% more carbon emissions than the carbon-aware baseline policy and an impressive 47% more than the queue-aware policy. The carbon-aware baseline policy, in comparison, saves only 4% more emissions than the queue-aware policy within the same timeframe. When examining yearly results, the carbon-aware improved policy continues to outperform, reducing emissions by about 43% compared to the carbon-aware baseline policy and 46% compared to the queue-aware policy. Meanwhile, the carbon-aware baseline policy achieves a 6% reduction in emissions compared to the queue-aware policy over the year. Interestingly, the monthly carbon emissions are approximately equal to the yearly ones, indicating that the average monthly carbon emissions remain relatively consistent throughout the year.

5.4 Queues Mean-Rate Stability

Given that the mean-rate stability of queues is defined as:

$$\lim_{t\to\infty} \frac{\mathbb{E}[Q(t)]}{t} = 0,$$

where $Q(t)$ should be replaced with $Q_m^e(t)$ or $Q_{m,n}^c(T)$, representing the queues in edge server and in clouds respectively. This definition implies that as time t goes to infinity, the time-average queue lengths at time t should converge to 0.

Regarding the time-average queue lengths $\mathbb{E}[Q(t)]$, Figure 5.4 shows that all curves exhibit fluctuations before eventually converging to 0, verifying the mean-rate stability of all the queues. For all policies, edge queues reach convergence faster than cloud queues. Among all, the carbon-aware baseline policy has the slowest convergence rate. One notable observation is that Cloud 1 has the largest number of tasks in its queue compared to other cloud queues at the beginning, which can be attributed to the cold start of the scheduling polices. In the initial iteration of our algorithms, when identifying the cloud with the shortest queue for each task type, all the queues in all clouds have lengths of zero. Consequently, this leads to the scheduling of all task types to the first cloud, resulting in an increased number of tasks in Cloud 1's queue at the start point.

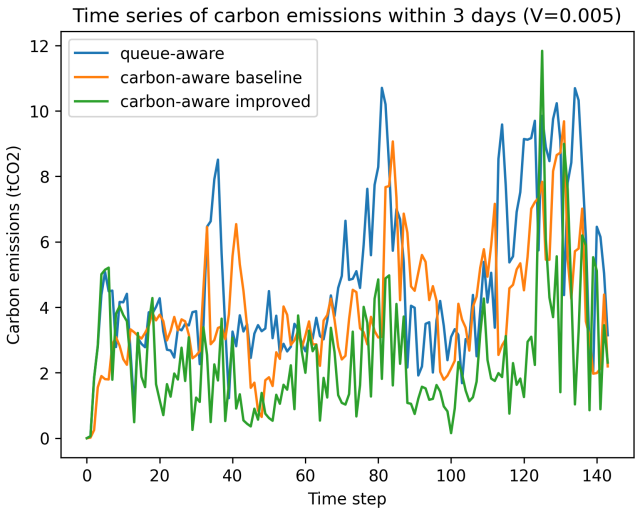

Figure 5.2: Time series of carbon emissions based on 3 policies (queue-aware, carbon-aware baseline, and carbon-aware improved) within 3 days. Note that two carbon-aware policies have $V = 0.005$.

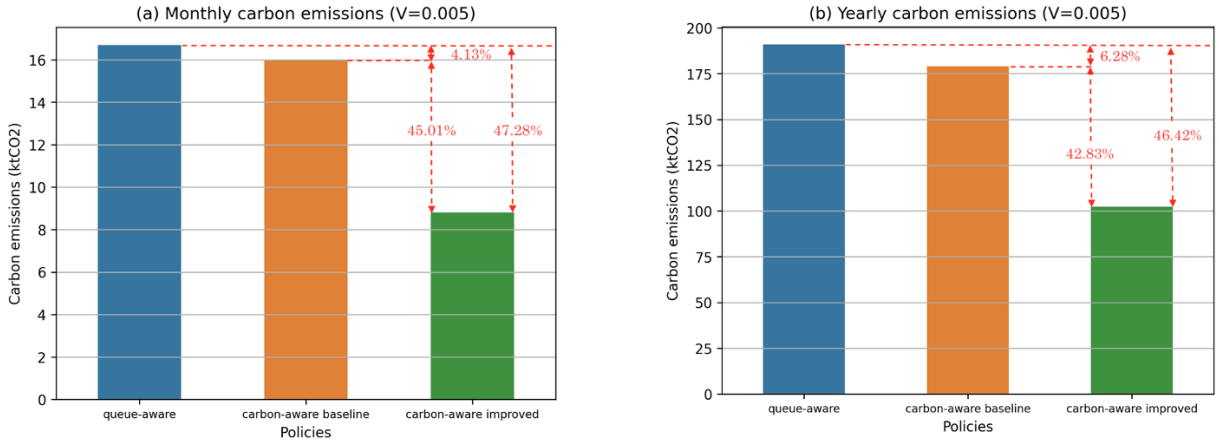

Figure 5.3: (a) monthly carbon emissions based on 3 policies; (b) yearly carbon emissions based on 3 policies. In both graphs, three policies are queue-aware, carbon-aware baseline, and carbon-aware improved while two carbon-aware policies have $V = 0.005$.

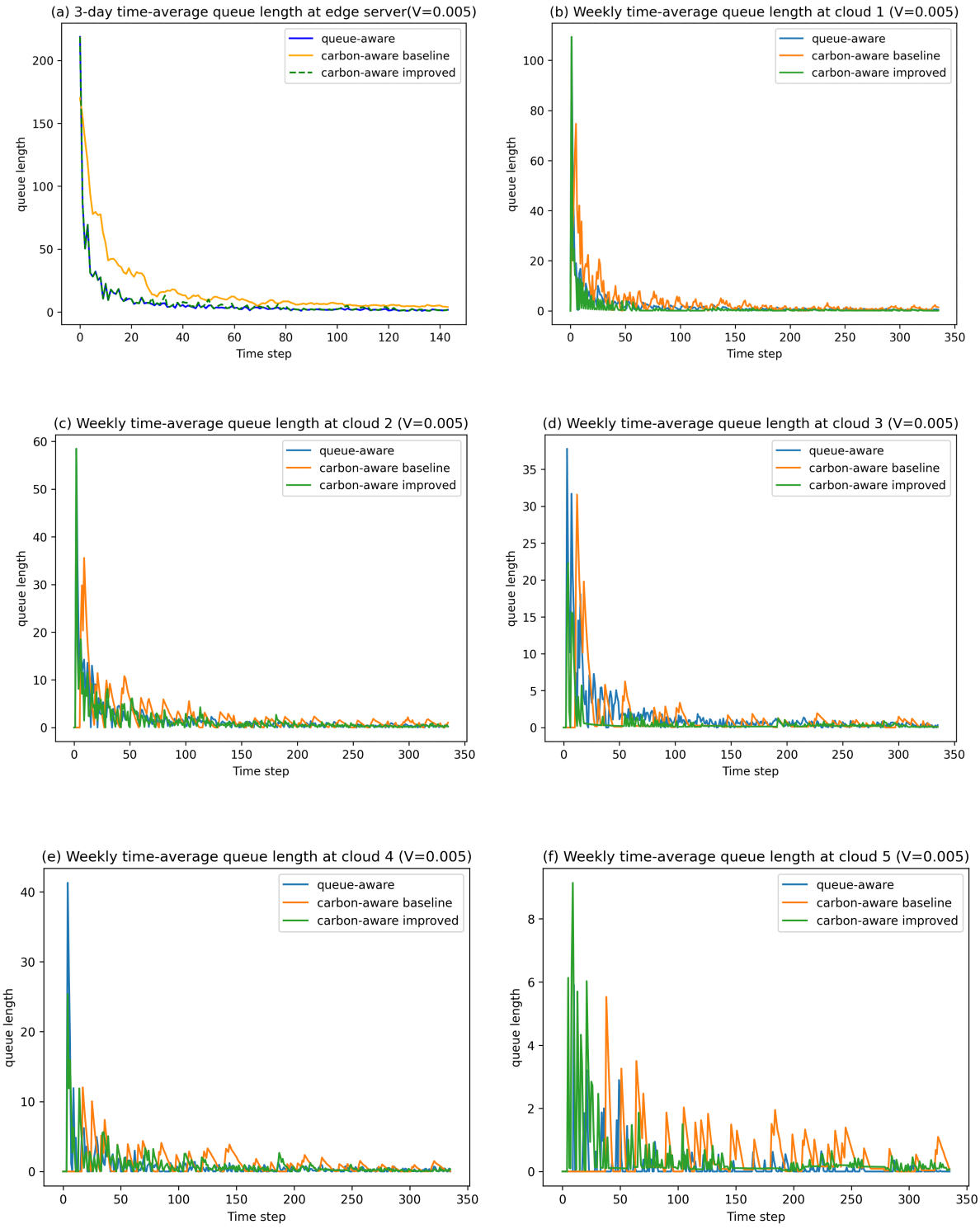

Figure 5.4: (a) Time-average edge queue length within 3 days based on 3 policies; (b)-(f) Weekly time-average cloud queue length for each of the 5 clouds respectively based on 3 policies. In all graphs, three policies are queue-aware policy, carbon-aware baseline policy, and carbon-aware improved policy and two carbon-aware policies have $V = 0.005$.

Chapter 6

Conclusion and Future Works

In this final chapter, we bring together the various threads of our research and reflect upon the implications of our findings. The primary goal of this study is to advance our understanding of the green computing network and to establish a foundation for our future work. The following sections will conclude our finding (section 6.1), followed by detailed discussions on the possible future works (section 6.2), emphasizing the importance of continued efforts to our research.

6.1 Conclusion

In conclusion, leveraging the drift-plus-penalty technique in Lyapunov optimization for our carbon-aware improved policy offers significant advantages in terms of carbon emissions reduction and queue stability in a computing network. This approach effectively balances the competing objectives of minimizing carbon emission and adhering to energy constraints and mean rate stability of queues, resulting in an environmentally friendly and efficient solution for resource allocation and scheduling.

The key difference between the carbon-aware improved policy and the carbon-aware baseline policy is that the improved policy takes into account the transmitting cost of the edge server, while the baseline policy does not. More specifically, the improved policy will decide to process the tasks at edge server when the transmitting cost is too high while the edge server in baseline policy will still deliver tasks to clouds. In addition, the queue-aware scheduling policy is another approach that focuses on managing the length of the queues, aiming to balance the load of all queues. However, this policy does not prioritize carbon emissions reduction, which makes it less environmentally friendly compared to the other two carbon-aware policies. These designs result in the improved policy's superior performance, with substantial monthly and yearly carbon emissions savings about $45\% \pm 3\%$ compared to both the carbon-aware baseline policy and the queue-aware policy. Besides, the time-average queues length of all policies eventually converges to 0, verifying the mean-rate stability of all queues.

Overall, implementing the carbon-aware improved policy with the drift-plus-penalty technique in Lyapunov optimization leads to substantial environmental benefits while maintaining a stable and efficient computing network. However, this is still a heuristic proposal and more future works are required for further improvements. We will discuss the possible future works in the next sections.

6.2 Future Works

In the pursuit of further advancements in the carbon-aware computing network, several future works can be explored to enhance the efficiency and sustainability of the system. We will

discuss the additional task scheduling when renewable energy is available in section 6.2.1, the battery management to avoid peak power consumption in section 6.2.2, and the deadline guaranteed job postponement in section 6.2.3.

6.2.1 Renewable Energy Utilization

Our carbon-conscious scheduling policy can be significantly improved by incorporating renewable energy sources. The main objective of this enhancement is to allocate more tasks during periods when renewable energy is readily accessible. To achieve this, we need to analyze the traces of different renewable energy sources, such as solar, wind, hydro, geothermal, and biomass. Each of these resources exhibits unique characteristics based on temporal and spatial conditions.

For instance, solar energy is more abundant during daylight hours or in areas with intense sunlight. If we choose to solely rely on solar energy, our tasks will be predominantly scheduled during daytime, which is opposite to the current policy. According to the distinct features of each renewable energy source, we need to determine which energy or combination of energies to employ, and decide on either a combined or alternate usage of these resources.

Gaining a thorough understanding of the attributes of each renewable energy resource will aid in designing a more effective and environmentally friendly scheduling policy. By tailoring our approach to the specific strengths and availability of these renewable resources, we can minimize our carbon footprint while maintaining efficient task completion.

6.2.2 Battery Management

Battery management is critical to green computing networks, as scheduling policies can use it to avoid peak power consumption periods. The primary objectives of battery management include optimizing energy usage, minimizing carbon emissions, and prolonging battery life. Firstly, the management should prioritize charging and discharging batteries in a manner that reduces carbon emissions. This can be achieved by leveraging renewable energy sources and minimizing reliance on high-carbon energy during battery charging. Secondly, to minimize the carbon footprint of the network, battery management need to enable aligning task scheduling with periods of low carbon intensity. This involves charging batteries when renewable energy sources are more readily available and discharging them during times when the electricity grid relies on high-carbon sources. By intelligently adjusting task scheduling based on the availability of cleaner energy, the network can operate more sustainably and reduce its overall environmental impact. Last but not least, the management should determine the ideal percentage of stored electricity that can be used without causing significant wear and tear to extend the lifespan of batteries. By carefully monitoring and managing battery usage, it's possible to balance performance and sustainability, ultimately leading to longer-lasting batteries.

6.2.3 Deadline-Guaranteed Job Postponement

Deadline-Guaranteed Job Postponement is a scheduling strategy used in green computing networks to optimize energy consumption and reduce carbon emissions without compromising the completion of tasks within their specified deadlines. This scheduling policy will continuously evaluate the state of the network, including the progress of ongoing tasks, energy availability, and the approaching deadlines of postponed tasks. It dynamically adjusts the execution plan to ensure that all tasks are completed within their specified deadlines while maintaining optimal energy efficiency.

Each task in the network has an associated duration and deadline, which must be met to ensure the successful completion of tasks. The scheduling policy takes into account these durations and deadlines when deciding which tasks can be postponed and for how long. With deadline and duration constraints, tasks within the network are considered to be urgent if they are estimated to be completed just within their deadlines. Then tasks will be assigned different priority levels based on their urgency. Higher-priority tasks are executed as soon as possible, while lower-priority tasks can be considered for postponement.

Deadline-guaranteed job postponement also helps to balance the computational load across the network. By postponing certain tasks, the network can distribute workloads more evenly, preventing overloading of individual nodes and improving overall performance and energy efficiency.

Such scheduling policy will monitor the energy consumption of the network and the availability of renewable energy sources. It attempts to align the execution of postponed tasks with periods of low carbon intensity or when renewable energy is more readily available. This reduces the overall carbon footprint of the network.

Bibliography

[1] Critical Group of Companies. data center, 2022.

[2] Anders S. G. Andrae and Tomas Edler. On global electricity usage of communication technology: Trends to 2030. *Challenges*, 6(1):117–157, 2015.

[3] Wikipedia Contributors. Limit inferior and limit superior, Feb 2023.

[4] George Kamiya. Data centres and data transmission networks – analysis - iea, 2022.

[5] Philipp Wiesner, Ilja Behnke, Dominik Scheinert, Kordian Gontarska, and Lauritz Thamsen. Let's wait awhile: How temporal workload shifting can reduce carbon emissions in the cloud. In *Middleware'21: 22nd International Middleware Conference*. ACM, 2021.

[6] Chien-Sheng Yang, Chien-Chun Huang-Fu, and I-Kang Fu. Carbon-neutralized task scheduling for green computing networks. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 4824–4829. IEEE, 2022.

[7] Hui Dou, Yong Qi, Wei Wei, and Houbing Song. Carbon-aware electricity cost minimization for sustainable data centers. *IEEE Transactions on Sustainable Computing*, 2(2):211–223, 2017.

[8] Shaolei Ren and Yuxiong He. Coca: Online distributed resource management for cost minimization and carbon neutrality in data centers. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2013.

[9] ASM Hasan Mahmud and Shaolei Ren. Dynamic server provisioning for carbon-neutral data centers. In *2013 42nd International Conference on Parallel Processing*, pages 536–541. IEEE, 2013.

[10] Yuanxiong Guo, Yanmin Gong, Yuguang Fang, Pramod P Khargonekar, and Xiaojun Geng. Energy and network aware workload management for sustainable data centers with thermal storage. *IEEE Transactions on Parallel and Distributed Systems*, 25(8):2030–2042, 2013.

[11] Guanglin Zhang, Shun Zhang, Wenqian Zhang, Zhirong Shen, and Lin Wang. Distributed energy management for multiple data centers with renewable resources and energy storages. *IEEE Transactions on Cloud Computing*, 10(4):2469–2480, 2020.

[12] Ali Pahlevan, Marina Zapater, Ayse K Coskun, and David Atienza. Ecogreen: Electricity cost optimization for green datacenters in emerging power markets. *IEEE Transactions on Sustainable Computing*, 6(2):289–305, 2020.

[13] Xinxin Liu, Yu Hua, Xue Liu, Ling Yang, and Yuanyuan Sun. Design and implementation of smooth renewable power in cloud data centers. *IEEE Transactions on Cloud Computing*, 2021.

[14] Datopian. Eso data portal: Regional carbon intensity forecast - dataset— national grid electricity system operator, 2020.

[15] Philipp Wiesner, Ilja Behnke, Dominik Scheinert, Kordian Gontarska, and Lauritz Thamsen. Let's wait awhile: How temporal workload shifting can reduce carbon emissions in the cloud. In *Middleware'21: 22nd International Middleware Conference*. ACM, 2021.

[16] IEA. France 2021 – analysis - iea, 2021.

Appendix A - Dataset Analysis

6.3 Dataset #1: National Grid ESO - Regional Carbon Intensity Forecast

```
# Import packages
import numpy as np
import math
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import matplotlib.dates as dates
import seaborn as sns

# Load dataset #1
CI = pd.read_csv('regional_carbon_intensity.csv',index_col='_id')
CI['datetime'] = pd.to_datetime(CI['datetime'], format="%Y/%m/%d %H:%M")
regions = ['North Scotland','South Scotland','North West England','North East England',
           'Yorkshire','North Wales and Merseyside','South Wales','West Midlands',
           'East Midlands','East England','South West England','South England','London','South East England']

# January 2022
CI202201 = CI.iloc[57650:57650+48*30]
CI202201

# Bi-weekly time series
x = range(672)
plt.figure(figsize=(10,4),dpi=300)
y = CI.loc[0:672, regions]
plt.plot(x, y)
plt.legend(regions,bbox_to_anchor=(1.025, -0.2), ncol=4)
plt.title("Carbon Intensity of 14 regions in Great Britain within 2 weeks")
plt.xlabel('Time step')
plt.ylabel('carbon intensity (gCO2/kWh)')
plt.show()

# Aggregated carbon intensity per 7.5h
plt.figure(figsize=(10,4), dpi=300)
plt.plot(range(21), CI202201_7.groupby(np.arange(len(CI202201_7)) // 16).sum())
plt.legend(regions, bbox_to_anchor=(1.025, -0.5), ncol=4)
t = ['00:00-07:30','08:00-15:30','16:00-23:30']
labels = []
day = 1
while day<=7:
    for i in t:
        if day < 10:
            s ='0'+str(day)+'T'+i
        else:
            s =str(day)+'T'+i
        labels.append(s)
    day += 1
```

```
47
48 plt.xticks(range(21),labels,rotation=75)
49 plt.title("Weekly Carbon Intensity of 14 regions in Great Britain")
50 plt.xlabel('Time')
51 plt.ylabel('carbon intensity (gCO2/kWh)')
52 plt.show()
```

Listing 6.1: Dataset #1 analysis

6.4 Dataset #2: ENTSO-E Transparency Platform and California ISO

```
1  # Import packages
2  import numpy as np
3  import math
4  import pandas as pd
5  import matplotlib.pyplot as plt
6  from datetime import datetime
7  import matplotlib.dates as dates
8
9  # Load dataset #2
10 cal_ci = pd.read_csv('lets-wait-awhile/data/cal_ci.csv', index_col='Time')
11 fr_ci = pd.read_csv('lets-wait-awhile/data/fr_ci.csv', index_col='Time')
12 gb_ci = pd.read_csv('lets-wait-awhile/data/gb_ci.csv', index_col='Time')
13 ger_ci = pd.read_csv('lets-wait-awhile/data/ger_ci.csv', index_col='Time')
14 CI = pd.concat([cal_ci, fr_ci, gb_ci, ger_ci], axis=1, join="inner")
15 CI.columns=['cal_ci', 'fr_ci', 'gb_ci', 'ger_ci']
16 CI.dropna(inplace=True)
17 CI['Time']=CI.index
18 CI['Time'] = pd.to_datetime(CI['Time'], format="%Y/%m/%d %H:%M")
19 cols = CI.columns.tolist()
20 cols = cols[-1:] + cols[:-1]
21 CI = CI[cols]
22
23 # Carbon intensity in 2020
24 CI2020 = CI.iloc[576:576+48*366]
25 CI2020
26
27 # Bi-weekly time series
28 x = range(672)
29 plt.figure(figsize=(10,4),dpi=300)
30 y = CI.loc['2020-01-01 00:00:00':'2020-01-14 23:30:00',['cal_ci','fr_ci','gb_ci','
       ger_ci']]
31 plt.plot(x, y)
32 plt.legend(['California','French','Great Britain','Germany'],bbox_to_anchor=(0.85,
       -0.15),ncol=4)
33 plt.title("Carbon Intensity of 4 regions within 2 weeks")
34 plt.xlabel('Time step')
35 plt.ylabel('carbon intensity (gCO2/kWh)')
36 plt.show()
37
38 # Aggregated carbon intenisty per 7.5h
39 plt.figure(figsize=(8,4),dpi=300)
40 plt.plot(range(21), CI202002_7.groupby(np.arange(len(CI202002_7)) // 16).sum())
41 plt.legend(['California','French','Great Britain','Germany'],
42           bbox_to_anchor=(0.9, -0.5),ncol=4)
43 t = ['00:00-07:30','08:00-15:30','16:00-23:30']
44 labels = []
45 day = 1
46 while day<=7:
47     for i in t:
48         if day < 10:
```

```
            s ='0'+str(day)+'T'+i
        else:
            s =str(day)+'T'+i
        labels.append(s)
    day += 1

plt.xticks(range(21),labels,rotation=75)
plt.title("Weekly Carbon Intensity of 4 regions")
plt.xlabel('Time')
plt.ylabel('carbon intensity (gCO2/kWh)')
plt.show()
```

Listing 6.2: Dataset #2 analysis

Appendix B - Policies

```
import numpy as np
import math
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import matplotlib.dates as dates
import seaborn as sns

# load carbon intensity dataset
CI = pd.read_csv('regional_carbon_intensity.csv',index_col='_id')
CI['datetime'] = pd.to_datetime(CI['datetime'], format="%Y/%m/%d %H:%M")
```

Listing 6.3: Import packages and load dataset

```
def queue_aware(M, N, Pe, Pc, pe, pc,T):
    list_ce = []
    edge_queue = []
    qca = []
    qcb = []
    qcc = []
    qcd = []
    qce = []

    np.random.seed(10)

    Qe = np.zeros(M,dtype=np.int64)
    Qc = np.zeros((M,N),dtype=np.int64)


    for t in range(57650,57650+48*T):
        # carbon emission at time t
        ce = 0

        d = np.zeros((M,N),dtype=np.int64) # number of task-m arrive cloud n
        w = np.zeros((M,N),dtype=np.int64) # nubmer of task-m depart from cloud n

        # number of task-m arrives at edge server
        a = np.random.randint(0,400,size=M,dtype=np.int64)

        # carbon intensity of
        Ce = CI_edge.iloc[t] # edge server
        Cc = CI_clouds.iloc[t] # cloud n

        # edge server
        order = np.argsort(Qe).tolist()[::-1] # sort edge queues from the longest
        n1 = np.argmin(Qc, axis=1) # clouds with shortest queue
        P = Pe
        for m in order:
            if math.floor(P/pe[m]) > 0:
                d[m][n1[m]] = min(Qe[m], math.floor(P/pe[m]))
                P -= d[m][n1[m]] * pe[m] # math.floor(P/pe[m]) * pe[m]
```

```
                ce += d[m][n1[m]]*pe[m]*Ce # carbon emission = energy consumption *
    intensity
            else:
                break

        # clouds
        for n in range(N):
            P = Pc[n]
            order = Qc.transpose().argsort().tolist()[::-1][n]

            for m in order:
                if math.floor(P/pc[m]) > 0:
                    w[m][n] = min(Qc[m][n], math.floor(P/pc[m]))
                    P -= w[m][n]*pc[m]
                    ce += w[m][n]*pc[m]*Cc.iloc[n]
                else:
                    break

        list_ce.append(ce/1000000)
        # update Qe and Qc
        Qe -= np.sum(d,axis=1)
        Qe[Qe<0] = 0
        Qe += a

        edge_queue.append(Qe.mean()/(t-57649))

        Qc -= w
        Qc[Qc<0] = 0
        Qc += d

        qc = np.mean(Qc,axis=0)
        list_qc = [qca, qcb, qcc, qcd, qce]
        for i,a in enumerate(list_qc):
            a.append(qc[i]/(t-57649))
    return list_ce, edge_queue, qca, qcb, qcc, qcd, qce
```

Listing 6.4: Queue-aware policy

```
def carbon_aware_baseline(V, M, N, Pe, Pc, pe, pc, T):
    list_ce = []
    edge_queue = []
    qca = []
    qcb = []
    qcc = []
    qcd = []
    qce = []

    np.random.seed(0)

    Qe = np.zeros(M,dtype=np.int64)
    Qc = np.zeros((M,N),dtype=np.int64)

    for t in range(57650,57650+48*T):
        ce = 0

        d = np.zeros((M,N),dtype=np.int64) # number of task-m arrive cloud n
        w = np.zeros((M,N),dtype=np.int64) # nubmer of task-m depart from cloud n

        # number of task-m arrives at edge server
        a = np.random.randint(0,400,size=M,dtype=np.int64)

        # carbon intensity of
        Ce = CI_edge.iloc[t] # edge server
        Cc = CI_clouds.iloc[t] # cloud n
```

```
27
28          # edge server
29          n1 = np.argmin(Qc, axis=1)
30
31          order = (np.min(Qc, axis=1)-Qe)/pe
32          order = order.argsort().tolist()
33
34          P = Pe
35          for m in order:
36              if math.floor(P/pe[m]) > 0:
37                  if V*Ce*pe[m] + Qc[m][n1[m]] - Qe[m] < 0:
38                      d[m][n1[m]] += min(Qe[m], math.floor(P/pe[m]))
39                      P -= math.floor(P/pe[m]) * pe[m] # d[m][n1[m]] * pe[m]
40                      ce += d[m][n1[m]]*pe[m]*Ce # carbon emission = energy
    consumption * intensity
41                  else:
42                      break
43
44          # clouds
45          for n in range(N):
46              P = Pc[n]
47              order = Qc[:,n]/pc
48              order = order.argsort().tolist()[::-1]
49
50              for m in order:
51                  if math.floor(P/pc[m]) > 0:
52                      if V*Cc.iloc[n]*pc[m] - Qc[m][n] < 0:
53                          w[m][n] += min(Qc[m][n], math.floor(P/pc[m]))
54                          P -= w[m][n]*pc[m]
55                          ce += w[m][n]*pc[m]*Cc.iloc[n]
56                      else:
57                          break
58
59          list_ce.append(ce/1000000)
60          # update Qe and Qc
61          Qe -= np.sum(d,axis=1)
62          Qe[Qe<0] = 0
63          Qe += a
64
65          edge_queue.append(Qe.mean()/(t-57649))
66
67          Qc -= w
68          Qc[Qc<0] = 0
69          Qc += d
70
71          qc = np.mean(Qc,axis=0)
72          list_qc = [qca, qcb, qcc, qcd, qce]
73          for i,a in enumerate(list_qc):
74              a.append(qc[i]/(t-57649))
75      return list_ce, edge_queue, qca, qcb, qcc, qcd, qce
```

Listing 6.5: Carbon-aware baseline policy

```
1  def carbon_aware_improved(V, M, N, Pe, Pc, pe_send, pe_process, pc, T):
2      list_ce = []
3      edge_queue = []
4      qca = []
5      qcb = []
6      qcc = []
7      qcd = []
8      qce = []
9
10     np.random.seed(10)
11
```

```
    Qe = np.zeros(M,dtype=np.int64)
    Qc = np.zeros((M,N),dtype=np.int64)

    for t in range(57650,57650+48*T):
        ce = 0

        d = np.zeros((M,N),dtype=np.int64) # number of task-m arrive cloud n
        we = np.zeros(M,dtype=np.int64) # nubmer of task-m depart from edge server
        wc = np.zeros((M,N),dtype=np.int64) # nubmer of task-m depart from cloud n

        # number of task-m arrives at edge server
        a = np.random.randint(0,400,size=M,dtype=np.int64)

        # carbon intensity of
        Ce = CI_edge.iloc[t] # edge server
        Cc = CI_clouds.iloc[t] # cloud n

        # edge server
        n1 = np.argmin(Qc, axis=1)
        r = V*Ce*pe_send + np.min(Qc, axis=1) - Qe
        h = V*Ce*pe_process - Qe

        order = r/pe_send
        order = order.argsort().tolist()


        P = Pe
        for m in order:
            if (math.floor(P/pe_send[m])>0) and (math.floor(P/pe_process[m])>0):
                if h[m]<0:
                    if h[m] < r[m]:
                        we[m] = min(Qe[m], math.floor(P/pe_process[m]))
                        P -= we[m]*pe_process[m]
                        ce += we[m]*pe_process[m]*Ce
                    else:
                        d[m][n1[m]] = min(Qe[m], math.floor(P/pe_send[m]))
                        P -= d[m][n1[m]] * pe_send[m]# math.floor(P/pe[m]) * pe[m]
                        ce += d[m][n1[m]]*pe_send[m]*Ce
                else:
                    if r[m]<0:
                        d[m][n1[m]] = min(Qe[m], math.floor(P/pe_send[m]))
                        P -= d[m][n1[m]] * pe_send[m]# math.floor(P/pe[m]) * pe[m]
                        ce += d[m][n1[m]]*pe_send[m]*Ce
            elif math.floor(P/pe_send[m])>0:
                d[m][n1[m]] = min(Qe[m], math.floor(P/pe_send[m]))
                P -= d[m][n1[m]] * pe_send[m]# math.floor(P/pe[m]) * pe[m]
                ce += d[m][n1[m]]*pe_send[m]*Ce
            elif math.floor(P/pe_process[m])>0:
                we[m] = min(Qe[m], math.floor(P/pe_process[m]))
                P -= we[m]*pe_process[m]
                ce += we[m]*pe_process[m]*Ce
            else:
                break


        # clouds
        for n in range(N):
            P = Pc[n]
            order = Qc[:,n]/pc
            order = order.argsort().tolist()[::-1]

            for m in order:
                if math.floor(P/pc[m]) > 0:
```

```
                    if V*Cc.iloc[n]*pc[m] - Qc[m][n] < 0:
                        wc[m][n] = min(Qc[m][n], math.floor(P/pc[m]))
                        P -= wc[m][n]*pc[m]
                        ce += wc[m][n]*pc[m]*Cc.iloc[n]
                    else:
                        break

        list_ce.append(ce/1000000)
        # update Qe and Qc
        Qe = Qe - np.sum(d,axis=1) - we
        Qe[Qe<0] = 0
        Qe += a

        edge_queue.append(Qe.mean()/(t-57649))

        Qc -= wc
        Qc[Qc<0] = 0
        Qc += d
        qc = np.mean(Qc,axis=0)
        list_qc = [qca, qcb, qcc, qcd, qce]
        for i,a in enumerate(list_qc):
            a.append(qc[i]/(t-57649))
    return list_ce, edge_queue, qca, qcb, qcc, qcd, qce
```

Listing 6.6: Carbon-aware improved policy

```
CI_edge = CI['North West England']
CI_clouds = CI[['South East England', 'North Wales and Merseyside', 'South Wales','
    West Midlands','East Midlands']]

def V_tuning():
    M = 5 # number of tasks
    N = 5 # number of clouds
    T = 30 # time range

    # kWh, energy constraints of
    Pe = 30000 # edge server
    Pc = np.full(N, 15000) # each cloud

    # kWh, energy consumption
    pe_send = np.full(M, 3.45)
    pe_process = np.full(M, [74, 97, 54, 16, 5.8])
    pc = np.full(M, [74, 97, 54, 16, 5.8]) # assume clouds are homogeneous

    x = CI.iloc[57650:57650+48*T]['datetime']
    ce1, equeue1, _, _, _, _, _ = carbon_aware_improved(0.0005, M, N, Pe, Pc,
    pe_send, pe_process, pc, T)
    ce2, equeue2, _, _, _, _, _ = carbon_aware_improved(0.005, M, N, Pe, Pc,
    pe_send, pe_process, pc, T)
    ce3, equeue3, _, _, _, _, _ = carbon_aware_improved(0.05, M, N, Pe, Pc, pe_send
    , pe_process, pc, T)
    ce4, equeue4, _, _, _, _, _ = carbon_aware_improved(0.5, M, N, Pe, Pc, pe_send,
     pe_process, pc, T)

#     print('Carbon Savings:')
#     print((np.array(ce0).sum()-np.array(ce1).sum())/np.array(ce0).sum())

    # bar graph for carbon savings
    d_bar = {'x':['0.0005','0.005','0.05','0.5'],
             'y':[np.array(ce1).sum()/1000, np.array(ce2).sum()/1000,
              np.array(ce3).sum()/1000, np.array(ce4).sum()/1000]}
    df_bar = pd.DataFrame(d_bar)

    plt.figure(dpi=300)
```

```
    plt.grid(axis='y')
    sns.barplot(data=df_bar,x='x',y='y',width=0.5)
    plt.xlabel("V")
    plt.ylabel("carbon emissions (ktCO2)")
    #plt.ylim((0,3000))
    plt.title("(a) Monthly carbon emissions with different V")
    plt.show()


    # edge queue
    d2={'x':x,'qe1':equeue1, 'qe2':equeue2,'qe3':equeue3, 'qe4':equeue4}
    df2 = pd.DataFrame(d2)

    plt.figure(dpi=300)
    plt.grid(axis='y')
    plt.plot(range(48*30), df2[['qe1','qe2', 'qe3','qe4']])
    plt.legend(['V=0.0005','V=0.005','V=0.05','V=0.5'])
    plt.xlabel('Time step')
    plt.ylabel('queue length')
    plt.title("(b) Monthly queue Length at edge server with different V")
    plt.show()
```

Listing 6.7: Hyper-parameter (V) tuning

```
CI_edge = CI['North West England']
CI_clouds = CI[['South East England', 'North Wales and Merseyside', 'South Wales','
    West Midlands','East Midlands']]

def main():
    V = 0.005 # penalty term V is a non-negative number
    M = 5 # number of tasks
    N = 5 # number of clouds
    T = 30 # time range

    # kWh, energy constraints of
    Pe = 30000 # edge server
    Pc = np.full(N, 15000) # each cloud

    # kWh, energy consumption
    pe_send = np.full(M, 3.45)
    pe_process = np.full(M, [74, 97, 54, 16, 5.8])
    pc = np.full(M, [74, 97, 54, 16, 5.8]) # assume clouds are homogeneous

    x = CI.iloc[57650:57650+48*T]['datetime']
    ce0, equeue0, qca0, qcb0, qcc0, qcd0, qce0 = queue_aware(M, N, Pe, Pc, pe_send,
     pc, T)
    ce1, equeue1, qca1, qcb1, qcc1, qcd1, qce1 = carbon_aware_baseline(V, M, N, Pe,
     Pc, pe_send, pc, T)
    ce2, equeue2, qca2, qcb2, qcc2, qcd2, qce2 = carbon_aware_improved(V, M, N, Pe,
     Pc, pe_send, pe_process, pc, T)

    print('Monthly carbon Savings (queue-baseline):')
    print((np.array(ce0).sum()-np.array(ce1).sum())/np.array(ce0).sum())
    print('Monthly carbon Savings (Baseline-Improved):')
    print((np.array(ce1).sum()-np.array(ce2).sum())/np.array(ce1).sum())
    print('Monthly carbon Savings (queue-Improved):')
    print((np.array(ce0).sum()-np.array(ce2).sum())/np.array(ce0).sum())

    # Monthly
    d_bar = {'x':['queue-aware', 'carbon-aware baseline', 'carbon-aware improved'],
             'y':[np.array(ce0).sum()/1000, np.array(ce1).sum()/1000, np.array(ce2)
    .sum()/1000]}
    df_bar = pd.DataFrame(d_bar)
    # Monthly bar graph for carbon savings
```

```
    plt.figure(dpi=300)
    plt.grid(axis='y')
    sns.barplot(data=d_bar,x='x',y='y',width=0.5)
    plt.xticks(fontsize=8)
    plt.xlabel("Policies")
    plt.ylabel("Carbon emissions (ktCO2)")
    plt.title("(a) Monthly carbon emissions (V=0.005)")
    plt.show()

    # yearly
    x = CI.iloc[57650:57650+48*357]['datetime']
    ce_year0, _, _, _, _, _, _ = queue_aware(M, N, Pe, Pc, pe_send, pc, 357)
    ce_year1, _, _, _, _, _, _= carbon_aware_baseline(V, M, N, Pe, Pc, pe_send, pc,
     357)
    ce_year2, _, _, _, _, _, _ = carbon_aware_improved(V, M, N, Pe, Pc, pe_send,
    pe_process, pc, 357)

    print('Yearly carbon Savings (queue-baseline):')
    print((np.array(ce_year0).sum()-np.array(ce_year1).sum())/np.array(ce_year0).
    sum())
    print('Yearly carbon Savings (baseline-improved):')
    print((np.array(ce_year1).sum()-np.array(ce_year2).sum())/np.array(ce_year1).
    sum())
    print('Yearly carbon Savings (queue-improved):')
    print((np.array(ce_year0).sum()-np.array(ce_year2).sum())/np.array(ce_year0).
    sum())

    # Yearly bar graph for carbon savings
    d_bar2 = {'x':['queue-aware', 'carbon-aware baseline', 'carbon-aware improved'
    ],
              'y':[np.array(ce_year0).sum()/1000, np.array(ce_year1).sum()/1000, np
    .array(ce_year2).sum()/1000]}
    df_bar2 = pd.DataFrame(d_bar2)
    plt.figure(dpi=300)
    plt.grid(axis='y')
    sns.barplot(data=df_bar2,x='x',y='y',width=0.5)
    plt.xticks(fontsize=8)
    plt.xlabel("Policies")
    plt.ylabel("Carbon emissions (ktCO2)")
    plt.title("(b) Yearly carbon emissions (V=0.005)")
    plt.show()

    # carbon emissions
    d={'x':range(48*3),'ce0':ce0[0:48*3], 'ce1':ce1[0:48*3], 'ce2':ce2[0:48*3]}
    df = pd.DataFrame(d)

    plt.figure(dpi=300)
    plt.plot(df['x'], df[['ce0','ce1','ce2']])
    plt.legend(['queue-aware','carbon-aware baseline','carbon-aware improved'])
    plt.xlabel('Time step')
    plt.ylabel('Carbon emissions (tCO2)')
    plt.title("Time series of carbon emissions within 3 days (V=0.005)")
    plt.show()

    # edge queue
    #d2={'x':range(3*48),'qe0':equeue0[0:48*3], 'qe1':equeue1[0:48*3], 'qe2':
    equeue2[0:48*3]}
    d2={'qe0':equeue0[0:48*3], 'qe1':equeue1[0:48*3], 'qe2':equeue2[0:48*3]}
    df2 = pd.DataFrame(d2)

#     plt.figure(dpi=300)
#     plt.grid(axis='y')

```

```
    cycler = plt.cycler(linestyle=['-', '-', '--'],color=['blue', 'orange', 'green'])
    fig, ax = plt.subplots(dpi=300)
    plt.grid(axis='y')
    ax.set_prop_cycle(cycler)
    df2.plot(ax=ax)

    #plt.plot(df2['x'], df2[['qe0','qe1','qe2']])
    plt.legend(['queue-aware','carbon-aware baseline','carbon-aware improved'])
    plt.xlabel('Time step')
    plt.ylabel('queue length')
    plt.title("(a) 3-day time-average queue length at edge server(V=0.005)")
    plt.show()

    # cloud queue
    list_qc = [[qca0, qca1, qca2],
               [qcb0, qcb1, qcb2],
               [qcc0, qcc1, qcc2],
               [qcd0, qcd1, qcd2],
               [qce0, qce1, qce2]]
    labels = ["b",'c','d','e','f']
    for i,a in enumerate(list_qc):
        d3={'x':range(48*7),'qc0':a[0][0:48*7], 'qc1':a[1][0:48*7], 'qc2':a[2][0:48*7]}
        df3 = pd.DataFrame(d3)

        plt.figure(dpi=300)
        plt.plot(df3['x'], df3[['qc0','qc1','qc2']])
        plt.legend(['queue-aware','carbon-aware baseline','carbon-aware improved'])
        plt.xlabel('Time step')
        plt.ylabel('queue length')
        plt.title(f'({labels[i]}) Weekly time-average queue length at cloud {i+1} (V=0.005)')
        plt.show()
```

Listing 6.8: Main function