Carbon-aware Resource Management for Latency-Sensitive Cloud Computing Environments

Tharindu B. Hewage

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

School of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE, AUSTRALIA

May 2025

ORCID: 0000-0001-6348-4602

Carbon-aware Resource Management for Latency-Sensitive Cloud Computing Environments

Tharindu B. Hewage
Principal Supervisor: Prof. Rajkumar Buyya
Co-Supervisors: Dr. Maria Read and Dr. Shashikant Ilager

Abstract

As society deepens its integration with cloud computing for the digitization of services, the proliferation of cloud-based latency-sensitive applications can be seen across various domains, such as the Internet of Things (IoT), Industry 4.0, and applications of the AI boom. Unlike traditional hyper-scale clouds, such applications require application-specific computing environments with tightly coupled application and hardware layers to deliver the stringiest latency performances. Conversely, such tight coupling increasingly limits cloud providers' control over application performance and hardware lifecycle management. Their latency service level objectives (SLOs) require infrastructure to maintain consistent performance through diverse application-specific heterogeneous hardware configurations that often provide lesser optimization opportunities.

Meanwhile, climate crisis-driven initiatives increasingly push cloud providers towards achieving near-term infrastructure carbon efficiency. Already, cloud infrastructures are significant contributors to global carbon emissions; thus, achieving near-term carbon efficiency with increasingly prevailing latency-sensitive cloud computing environments is paramount in meeting the cloud's carbon emission goals. However, the limited provider's control in those environments significantly challenges its potential to achieve carbon efficiency through carbon-aware resource management. In particular, the limited application performance management prevents cloud providers from applying resource management techniques aimed at reducing infrastructure active carbon emissions (i.e., operational carbon), such as integrating low-carbon intermittent renewable energy sources through compromising application performance for carbon efficiency. Further, the limited provider's control over the hardware lifecycle management prevents cloud providers from applying efficient hardware optimization techniques aimed at reducing procured carbon emissions made with associated business activities (i.e., embod-

ied carbon), such as hardware re-purposing to slow down the accumulation of manufacturing carbon emissions from frequent hardware replacements. To this end, uncovering alternative carbon-aware resource management opportunities entangled within tightly coupled application and hardware layers of latency-sensitive cloud computing environments becomes paramount to the cloud's sustainable growth.

This thesis investigates novel algorithms, approaches, and techniques to identify and exploit application-specific resource management opportunities to improve operational and embodied carbon efficiency in latency-sensitive cloud computing environments without compromising application latency SLOs. This thesis advances the state-of-the-art in carbon-aware resource management by making the following key contributions:

1. A comprehensive taxonomy and literature review on the carbon-aware resource management in latency-sensitive cloud computing environments along with a discussion on identified research gaps and potential future research work.

2. A dynamic power budget and a decentralized task scheduling algorithm to alleviate power constraints with renewable energy in geographically distributed IoT Micro-Cloud networks.

3. A framework to exploit application-level fault-tolerance in real-time cloud systems to integrate renewable energy without compromising the system's deterministic application execution.

4. A technique to accommodate low-latency applications in multi-region renewable energy harnessing using server pooling of CPU Simultaneous Multi-Threading (SMT).

5. An aging-aware CPU core management technique for extended amortization of procured embodied carbon in Large Language Model (LLM) inference clusters.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Tharindu B. Hewage, May 2025

Preface

Main Contributions

This thesis research has been carried out in Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne. The main contributions of the thesis are discussed in Chapters 2-6 and are based on the following publications:

- **Tharindu B. Hewage**, Shashikant Ilager, Maria Rodriguez Read and Rajkumar Buyya, "Carbon-aware Resource Management in Latency-Sensitive Cloud Computing Environments: A Taxonomy and Review", *ACM Computing Surveys (CSUR)* [Submitted, May 2025].

- **Tharindu B. Hewage**, Shashikant Ilager, Maria A. Rodriguez, Patricia Arroba, and Rajkumar Buyya, "DEMOTS: A Decentralized Task Scheduling Algorithm for Micro-Clouds with Dynamic Power-Budgets," *Proceedings of the 16th IEEE International Conference on Cloud Computing (CLOUD)*, Pages: 418-427, Chicago, IL, USA, July 2-8, 2023.

- **Tharindu B. Hewage**, Shashikant Ilager, Maria A. Rodriguez, and Rajkumar Buyya, "A Framework for Carbon-aware Real-Time Workload Management in Clouds using Renewables-driven Cores", *IEEE Transactions on Computers*, early access, May 2025.

- **Tharindu B. Hewage**, Shashikant Ilager, Maria A. Rodriguez, and Rajkumar Buyya, "A Technique for Load Shifting Low-latency Applications in Multi-Region Renew-

ables Harvesting via SMT Core Pooling", *IEEE Transactions on Power Systems* [Submitted, May 2025].

- **Tharindu B. Hewage**, Shashikant Ilager, Maria Rodriguez Read, and Rajkumar Buyya, "Aging-aware CPU Core Management for Embodied Carbon Amortization in Cloud LLM Inference", *Proceedings of the 16th ACM International Conference on Future and Sustainable Energy Systems (E-ENERGY)*, Rotterdam, Netherlands, June 17-20, 2025 [Accepted].

Supplementary Contributions

During the Ph.D. candidature, I have also contributed to the following work (this thesis does not claim it as its contributions):

- **Tharindu B. Hewage**, Shashikant Ilager, Maria A. Rodriguez, and Rajkumar Buyya, "CloudSim express: A novel framework for rapid low code simulation of cloud computing environments", *Software: Practice and Experience*, Volume 54, Pages 483-500, March 2024.

Acknowledgements

My PhD journey would not have been possible without the support of many people. First and foremost, my heartfelt thanks to my supervisors, Professor Rajkumar Buyya, Dr. Maria Read, and Dr. Shashikant Ilager. I am grateful for their invaluable support, guidance, and encouragement over those long years. Further, my gratitude goes to my PhD advisory committee chair, Prof. Udaya Parampalli, for the invaluable support towards a timely completion.

My heart also goes to all the past and present members of the qCLOUDS Laboratory at the University of Melbourne. In particular, I thank Dr. Mohammad Goudarzi, Dr. Samodha Pallewatta, Dr. Amanda Jayanetti, Dr. Anupama Mampage, Jie Zhao, Ming Chen, Siddharth Agarwal, Hoa Nguyen, Zhiyu Wang, Duneesha Fernando, Thakshila Imiya Mohottige, Qifan Deng, TianYu Qi, Hootan Zhian, Murtaza Rangwala, Yifan Sun, Prabhjot Singh, Haoyu Bai, Abhishek Sawaika, and Avishka Sandeepa, for the support and our friendship fostered over the years.

I would also like to thank my family, my wonderful mom, Manel, my dad, Cyril, and my loving sisters, Uthpalee and Piyumi, for their unconditional love and support. To all my friends, especially my friendship support group of "Six Thirty", Amila, Pasindu Kadawatha, Supun, Pasindu Hewagamage, Janaka, Kavindu, Janith, Hasitha, and Ramesh. A special appreciation for my loving wife, Nilakshi, for supporting my decision to pursue a PhD, being there through my ups and downs, and the resilience and love shared through all these years.

I acknowledge the University of Melbourne for providing me with the scholarship and resources to pursue my doctoral studies. I would like to sincerely thank the past and present admin staff of the School of Computing and Information Systems for their support. I thank my undergraduate supervisors, Dr Ranga and Dr. Chamira, for their support and grooming me into an individual capable of PhD studies.

Finally, I would like to acknowledge the people of Sri Lanka. Decades ago, a group of brave Sri Lankan decided make education free to every citizen, an equal opportunity to pursue their dreams. As a result of a chain reaction to that, my own PhD dream has become a reality. My heartfelt thanks to each and every, known and unknown, hard-working Sri Lankans for paying for my studies all the way upto tertiary education. This achievement is as much yours as it is mine.

Tharindu B. Hewage
May 2025, Melbourne, Australia

Contents

List of Figures

List of Tables

List of Acronyms

GHG Green House Gases

IT Information Technology

AI Artificial Intelligence

IoT Internet of Things

ML Machine Learning

LLM Large Language Models

SLA Service Level Agreement

SLO Service Level Objectives

DVFS Dynamic Voltage and Frequency Scaling

RAPL Running Average Power Limit

NBTI Negative Bias Temperature Instability

PDU Power Delivery Unit

SMT Simultaneous Multi-threading

CPU Central Processing Unit

GPU Graphics Processing Unit

IaaS Infrastructure-as-a-service

VM Virtual Machines

VNF Virtual Network Functions

WAN Wide Area Network

Chapter 1
Introduction

Cloud computing has become the backbone of modern society with the deepening digitalization of services such as banking, healthcare, e-commerce, gaming, transport, and education [15–18]. Today, hyper-scale cloud providers dominate cloud infrastructures that provide on-demand computation to their users. Over the past fifteen years, the share of data center capacity for hyper-scale cloud providers has grown from 10% to 37% today [19]. Recently, we have seen a surge in expanding hyper-scale cloud infrastructures, where global investments for new data centers have increased by nearly 70% during the past two years [19]. The primary driver behind this trend is the rise of cloud-based latency-sensitive use cases, including generative Artificial Intelligence (AI), autonomous driving, Industry 4.0, and content streaming [13, 15, 17].

Adversely, growing cloud infrastructures incur a detrimental impact on their carbon footprint. Cloud data centers already contribute to nearly 1% of the total global carbon emissions [20]. As data centers expand, they consume increased amounts of energy from carbon-intensive energy sources, which elevates their operational carbon footprint [19]. In India alone, data center capacity is expected to grow from 2 GW today to nearly 5 GW by 2030, while its 74% of coal-based dominance in electricity generation today will likely continue beyond 2030 [19]. Further, the carbon emissions made during associated business activities over the lifecycle of the data center hardware, such as manufacturing, shipping, retiring, and recycling Information Technology (IT) assets, elevate data centers' embodied carbon footprint [21, 22]. Microsoft, a hyper-scale cloud provider, reported that its embodied carbon footprint increased by 30.9% during the past four years [23].

Carbon emissions released into the atmosphere remain for many years, creating a

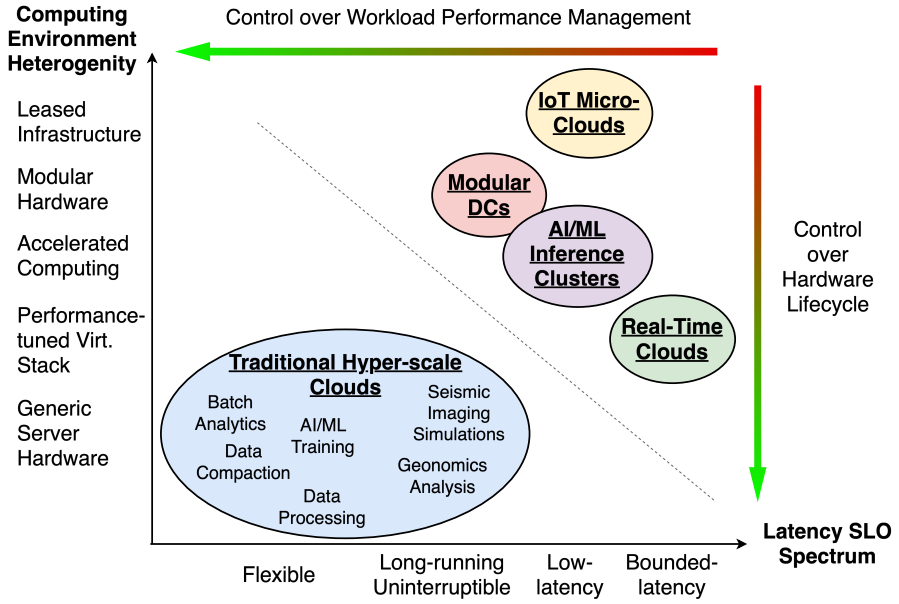


Figure 1.1: Comparison of cloud computing environments for latency SLO.

greenhouse effect that leads to catastrophic effects of global warming, such as environmental events of weather extremes and melting polar caps, leading to economic disruptions and conflicts due to food and water insecurity and rising sea levels [24]. Consequently, environmentally conscious parties constrain cloud providers to limit their data center carbon emissions. One such example is the compliance of cloud providers with the Paris Agreement, requiring data centers to reduce their emissions by 45% between 2020 and 2030 [25]. Today, major hyper-scale cloud providers continue to integrate sustainability measures for both operational and embodied carbon efficiency. For example, Amazon Web Services (AWS) is matching 100% consumed electricity with renewable energy generation, is starting the transition to hydrotreated vegetable oil (HVO) (a fossil fuel alternative that offers up to 90% carbon emission reduction) for backup power generation, and is working with IT hardware manufacturers and data center building construction to utilize low-carbon processes [26]. Similar commitments can be seen with Microsoft Azure [27] and Google Cloud [28].

However, the existing sustainability measures of hyper-scale cloud providers are largely limited to traditional cloud infrastructures facilitating workloads with flexible latency Service Level Objectives (SLOs) on generic server hardware. This is because

flexible latency SLOs provide better opportunities for workload performance management for operational carbon efficiency, and generic server hardware provides better opportunities for hardware lifecycle management for embodied carbon efficiency. As a result, emerging latency-sensitive cloud computing environments having tight latency SLOs and heterogeneous hardware are often overlooked in cloud carbon optimizations. Figure 1.1 compares latency-sensitive cloud computing environments over traditional cloud infrastructures by taking latency SLOs as the x-axis and computing environment heterogeneity as the y-axis. It demonstrates the limited control over workload performance management and hardware lifecycle management present in latency-sensitive cloud computing environments.

The control over workload performance management determines the cloud provider's capacity to optimize operational carbon efficiency through 24/7 integration of low-carbon renewable energy. This is due to the variations in the energy supply of renewable energy sources. Cloud providers depend on controlling the workload performance to match their energy demand with intermittent energy supply variations of renewable energy sources. For instance, suspending/resuming the flexible workloads through time and space allows cloud operator to increase and decrease their energy consumption to match the available energy capacity. Figure 1.1 depicts that latency-sensitive cloud computing environments are often constrained to low-latency and bounded-latency regions in the latency SLO spectrum, thus presenting fewer opportunities at workload performance management. For example, Real-Time Clouds having bounded response times require deterministic computing, which is facilitated by tuning server virtualization stacks for rigid high-performance real-time power profiling, limiting cloud providers' control over fine-grained workload performance management. Similarly, AI or Machine Learning (ML) Inference Clusters serving interactive queries require faster model computations, which are facilitated by performance-constrained cloud servers with Graphics Processing Unit (GPU) accelerators.

The control over hardware lifecycle management determines the cloud provider's capacity to optimize embodied carbon emissions through various management tech-

niques of their IT hardware, such as data center design, resource oversubscription, component re-purposing, component re-using, and hardware life extensions. As shown in Figure 1.1, latency-sensitive cloud computing environments demonstrate the most heterogeneity in their hardware, thus presenting fewer opportunities for hardware lifecycle management. For example, Internet of Things (IoT) Micro-Clouds serving Internet of Things (IoT) workloads for geographically distributed end-users are often facilitated by distributed Micro-Clouds deployed with leased infrastructures, where the cloud provider does not have control over most hardware that is owned by the third-party infrastructure provider. Similarly, Modular data centers (i.e., Modular DCs), where servers are deployed in low-footprint self-contained data center units for rapid deployment and scaling, provide limited design opportunities and limited options to choose alternative low-carbon hardware components due to their resource constraints. As a result, Modular DCs limit the cloud provider's control to optimize its hardware management.

Therefore, to achieve sustainable growth in emerging latency-sensitive cloud computing environments, it is imperative to address its limited control of workload performance management and hardware lifecycle management with the adoption of efficient carbon-aware resource management techniques. Thus, the main focus of this thesis is to study techniques for navigating the complex optimization space of application latency SLO and computing environment heterogeneity of latency-sensitive cloud computing environments. This includes the exploration of resource allocation, workload scheduling, workload execution, energy supply variations, and hardware usage patterns.

We first conduct a comprehensive survey to review existing literature, alongside identified problems, proposed solutions, and the shortcomings, encompassing a broader scope of resource management for both operational and embodied carbon optimization in latency-sensitive cloud computing environments. We then proceed to propose efficient techniques for power management, workload scheduling and execution, and hardware longevity for the carbon efficiency of the computing environments. We demonstrate the superiority of the proposed techniques by extensively evaluating them under both simulation and practical settings.

1.1 Background and Motivations

To better understand the research problem addressed in this thesis, in this section, we present background and motivation on cloud carbon efficiency and latency-sensitive cloud computing environments.

1.1.1 Latency-Sensitive Cloud Computing Environments

Cloud computing provides on-demand compute/storage services following a pay-as-you-go model. In traditional cloud computing, service delivery follows distinct models, such as Infrastructure-as-a-service (IaaS), Platform-as-a-service (PaaS), Software-as-a-service (SaaS), and more recent models such as Function-as-a-service (FaaS). Each service model enables cloud users to bootstrap their application deployment at different levels. In IaaS, cloud users utilize on-demand provisioning and scaling of managed infrastructure resources, such as computing, storage, and networking, to build and deploy their applications. In PaaS, cloud users utilize managed software platforms to develop, run, and maintain their applications. In SaaS, cloud users utilize entire managed application stacks, in which all maintenance, such as software updates and bug fixes, are taken care of. More recently, higher levels of compute abstractions are provided with service models such as FaaS, in which cloud users maintain their application code while its execution and scaling are managed by the cloud provider. The majority of these service models provide different levels of segregation between the application and the infrastructure. In contrast, emerging latency-sensitive cloud computing environments cater to application deployments that are tightly coupled with the intricacies of the infrastructure due to their stringent latency performance constraints.

Figure 1.2 illustrates and compares tight coupling between applications and computing infrastructure in latency-sensitive cloud computing environments over traditional cloud computing service models. As shown, traditional cloud service models focus on delivering ease of use for cloud users. However, latency-sensitive cloud computing environments require application-specific variations of their infrastructure, leading to resource and performance-constrained cloud computing environments. To showcase this, we use three prominent use case examples. In the case of real-time clouds

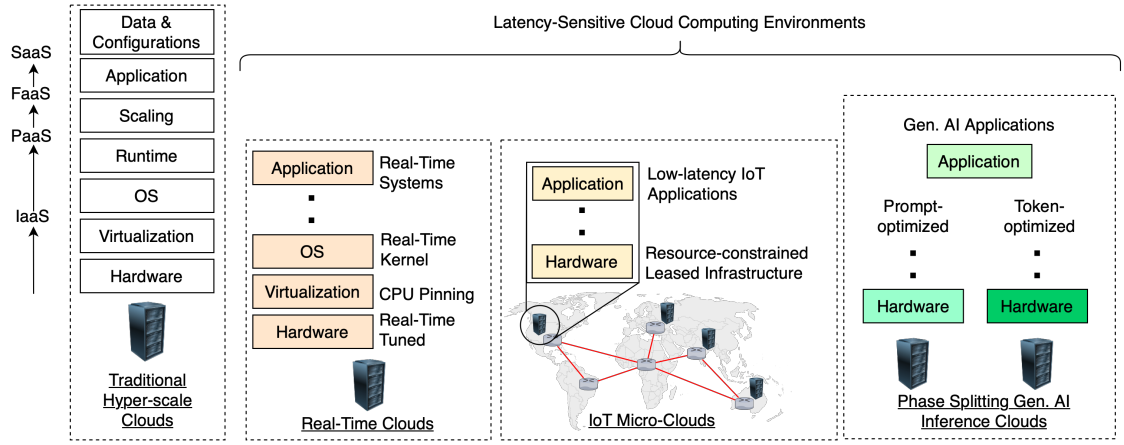


Figure 1.2: Tight coupling between applications and computing infrastructure in latency-sensitive cloud computing environments.

[17], applications require deterministic computing through the cloud virtualization stack to meet bounded-latency responses. Hence, their computing environment consists of performance-tuned hardware, such as disabled power efficiency features in Central Processing Unit (CPU); guaranteed resource allocation through virtualization, such as pinning CPU cores to application Virtual Machines (VM); and prioritized execution of real-time application threads through the operating system's kernel [29]. In the case of the IoT Micro-Clouds [30, 31], applications are required to serve end-users at the network's edge to meet their low-latency responses. As a result, Micro-Clouds often deploy on resource-constrained leased infrastructure from Colocation data center providers. In the case of generative AI inference clouds [14], applications often require specialized server hardware, such as GPU accelerators with phase-specific variants, to meet model computation for low-latency interactive sessions.

Our use case examples demonstrate the complicated intricacies present in latency-sensitive cloud computing environments. As a result, deriving an efficient resource management approach for their carbon efficiency becomes increasingly difficult. Moreover, the majority of existing resource management approaches for carbon efficiency are designed for traditional hyper-scale clouds, which cannot be transferred to latency-sensitive cloud environments, further challenging the problem of carbon-aware resource management. In the next subsection, we focus on carbon efficiency in cloud environ-

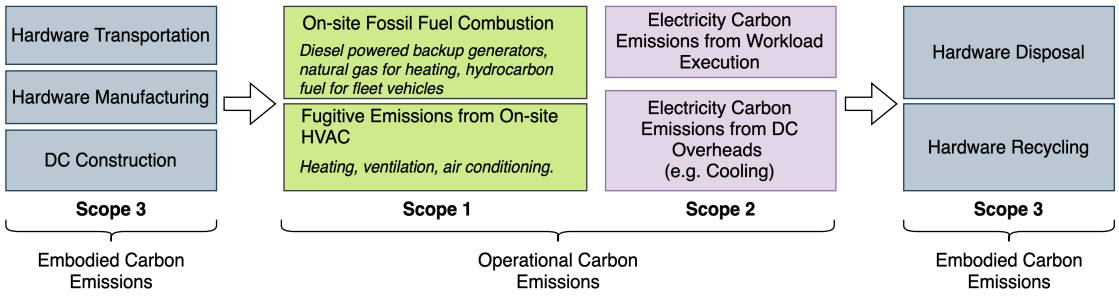


Figure 1.3: Carbon emissions of cloud computing environment lifecycle [1].

ments, highlighting its requirements. Afterward, in section 1.2, we outline concrete challenges that need to be solved to meet those requirements in latency-sensitive cloud computing environments.

1.1.2 Carbon Efficiency in Cloud Computing Environments

In measuring carbon emissions of a cloud computing infrastructure, the Green House Gases (GHG) protocol [32], a global standard formed to manage GHG emissions, outlines three scopes of emissions for data centers to encompass both direct and indirect carbon emissions made during the data center lifecycle. Figure 1.3 illustrates key elements of the data center lifecycle for each scope based on recently published data by Google [1], a hyper-scale cloud provider.

Lifecycle carbon emissions consist of two categories: operational and embodied emissions. Operational carbon emissions of a cloud computing environment relate to scope one and scope two emissions of the GHG protocol. Scope 1 refers to direct emissions made on-site, such as fossil fuel combustion of backup power generators and fleet vehicles and natural gas used for heating. Scope 2 refers to indirect emissions made from electricity consumption. Data centers often consume electricity from power grids where it is located; thus, the carbon intensity of the energy sources used to generate electricity is attributed to the data center's energy demand, predominantly from workload execution and overhead consumptions such as server cooling. Embodied emissions of the cloud computing environment are attributed to indirect carbon emissions made during

Scope 3 activities, such as from DC construction, hardware manufacturing, and installation to hardware disposal and recycling. Carbon emissions made in both categories are measured with $kgCO_2eq$, kilograms of carbon dioxide equivalent.

Carbon efficiency in cloud computing environments refers to reducing emissions across both embodied and operational aspects. In this thesis, we focus on achieving carbon efficiency through effective resource management of the cloud computing environment. In managing operational carbon emissions, cloud resource management is most effective in optimizing Scope 2 emissions due to multiple reasons. Optimizing Scope 1 emissions can involve slow processes, such as transitioning to low-carbon fuel sources. Further, it can be out of the control of the cloud provider, such as fugitive emissions, and could also be critical to data center operations and thus difficult to optimize for, such as powering backup generators. In contrast, Scope 2 emissions can be optimized in the short term by integrating clean energy mixes. Recently, renewable energy sources with clean energy have been penetrating energy grids around the globe [15], presenting cloud operators with opportunities to reduce their operational emissions by matching their electricity demand with clean energy availability. In managing embodied carbon emissions, resource management techniques are most effective in slowing the rate of embodied carbon accumulation and extending the amortization of procured embodied carbon. Alternative approaches to improving the carbon efficiency of business processes, such as data center construction, hardware manufacturing, and transportation, require long-term efforts among multiple stakeholders, thus presenting fewer opportunities to address embodied carbon in the near future.

Therefore, this thesis focuses on aspects of resource management to optimize carbon efficiency in latency-sensitive cloud computing environments for reducing Scope 2 operational carbon emissions through integrating renewable energy mixes of power grids and reducing Scope 3 embodied carbon emissions through slowing down its accumulation and extending amortization of those already procured. In the next section, we discuss key challenges that must be addressed in that.

1.2 Challenges in Carbon-aware Resource Management in Latency-Sensitive Clouds

Latency-sensitive cloud computing environments are increasingly prevailing due to the deep integration of cloud computing with interactive and critical services provided in modern societies. At the same time, maintaining its sustainable growth is paramount in our efforts to address the climate crisis. However, unlike traditional cloud computing, where cloud providers could better optimize for carbon efficiency due to the segregation of infrastructure from applications, tight coupling of latency-sensitive applications with the computing infrastructure, and application-specific diverse patterns, challenges the cloud provider in achieving carbon efficiency with latency-sensitive cloud computing environments. In order to be able to effectively conduct carbon-aware resource management, the provider needs to base its techniques on unique patterns of application execution under different scenarios and hardware configurations.

Carbon-aware resource management spans both operational and embodied carbon optimizations and requires satisfying the unique resource and performance constraints of the computing environment. In operational carbon optimization, providers identify the best approaches to shift electricity consumption to low-carbon renewable energy availability. In embodied carbon optimization, providers identify long-term approaches to reduce future accumulation of embodied carbon and amortization of already procured embodied carbon. We then discuss the resource management challenges associated with the latency-sensitive cloud computing environments:

- **Limited control over workload performance management:** A key challenge in latency-sensitive cloud environments is that applications offer quite limited opportunities to compromise their performance over carbon efficiency. In resource management, exploiting workload performance allows cloud operators to control electricity demand for better utilization of the energy grid's carbon intensity. For instance, traditional cloud servers allow dynamic performance management techniques such as Dynamic Voltage and Frequency Scaling (DVFS) to control server power draw. However, with latency-sensitive applications, DVFS can lead to degraded application performance and violate their Service Level Agreements

(SLAs) of response times. Therefore, the cloud operator needs to explore fine-grained application patterns to uncover opportunities to match the infrastructure energy consumption without degrading latency performance.

- **Supply dynamics of clean energy sources:** Recently, renewable energy sources have been increasingly penetrating the global power grids. However, the majority of such produce electricity with intermittent supply dynamics. This is because intermittent renewable energy sources, such as solar and wind, are relatively easy to provision and operate when compared to stable, clean energy sources, such as nuclear plants. Integrating intermittent renewable energy sources significantly challenges cloud provider in maintaining their workload performance amidst supply variations, especially within performance-constrained environments of latency-sensitive cloud applications.

- **Unpredictable Network Performances:** A significant portion of cloud latency-sensitive applications relies on globally distributed computing environments to deliver improved end-user latency performance. These environments heavily depend on the Wide Area Network (WAN) that connect geographically distributed servers and present opportunities to harness clean energy availability across time of day. However, existing network traffic management in WANs can often lead to unpredictable network performance. In this context, cloud providers are challenged to maintain satisfactory application latency performance over WAN amidst harvesting cross-region clean energy availability.

- **Limited control over hardware lifecycle:** A key characteristic in latency-sensitive cloud environments is that cloud providers often have limited control over their infrastructure hardware. For instance, Micro-Clouds deployed in leased infrastructures limit the cloud provider's role as a renter in infrastructure ownership, thus providing fewer opportunities to optimize embodied carbon emissions. In the case of modular data centers, where servers are deployed in low-footprint self-contained data center units, providers have limited options when choosing alternative hardware components with limited embodied carbon footprints due to the resource constraints of the modular data center. Therefore, cloud providers must

uncover opportunities present in hardware compatibility, hardware performance, and infrastructure design to reduce accumulation and increase amortization of infrastructure embodied carbon through efficient resource management mechanisms, such as hardware re-purposing, hardware re-using, and extending hardware lifetimes.

- **Asymmetric hardware life cycles:** Due to the stringent performance requirements, latency-sensitive cloud servers can depend on hardware accelerators. These server components are less mature in their development when compared to traditional server components; thus, manufacturers release newer hardware generations with improved performance quite frequently. As a result, cloud providers are challenged with a shorter hardware lifecycle for accelerators, which adversely increases the accumulation of the environment's embodied carbon footprint.

- **Premature hardware failures:** Catering for bounded latency performance of certain latency-sensitive cloud environments requires high-performance tuning of hardware to maintain consistent peak server performance. Such that potential violations of response time bounds in Service Level Agreement (SLA) are minimized. However, in the long term, such performance tunings degrade the reliability of the hardware, leading to premature failures and, eventually, shorter hardware refresh life cycles. In this context, cloud providers are challenged by the faster accumulation of embodied carbon from replacement hardware. Therefore, hardware performance tuning must be efficiently optimized to slow down hardware degradation while preserving application latency performance.

1.3 Research Questions and Objectives

There are two major challenges in achieving carbon efficiency in latency-sensitive cloud computing environments. Firstly, the action space of integrating unstable renewable energy sources for latency-sensitive applications must be carefully explored to identify potential resource management solutions that satisfy application-specific latency Service Level Objectives (SLOs) over renewable energy supply dynamics. Secondly, ef-

ficient resource management solutions must be derived to enable cloud providers to optimize their latency-sensitive application-specific hardware for lower embodied carbon, such as achieving prolonged usage of existing hardware, re-purposing hardware to yield a second life, and reducing the dependency on components with higher embodied carbon by design, all while satisfying latency SLOs. The objective of this thesis is to study these challenges in prominent latency-sensitive cloud computing environments at a fine-grained level and propose approaches for all resource management aspects while satisfying performance and resource constraints. In order to meet these objectives, we formulate and address the following research questions.

- Q1. *How to leverage the provisioning flexibility of on-site renewable energy sources to improve power budget constraints of decentralized and distributed IoT Micro-Cloud networks deployed with Colocation data center providers?* Due to congested power grid capacities, Colocation data center providers tend to constrain tenants' subscriptions to the shared power delivery as a mechanism for scaling to cater to the rising demand. Adversely, renting Micro-Clouds is at the risk of workload performance compromises in mitigating power overdraw events. In contrast to power grids, on-site renewable energy generation provides provisioning flexibility to relax power constraints, yet its supply dynamics could present uncertainty in delivering stable power subscriptions. Therefore, it requires efficient management of IoT tasks within the Micro-Cloud network addressing dynamic network performance of WAN, cross-regional clean energy availability, and maintaining the task's latency performance at all times.

- Q2. *How to integrate clean energy with real-time cloud environments of bounded latency without compromising the deterministic nature of its application execution?* Real-time clouds ensure deterministic application performance for end-users to cater to critical services with bounded response times. To do so, they often compromise carbon-efficiency over the computing stack's performance. Although the integration of clean energy is effective in reducing the environment's operational carbon emissions, it demands alternative strategies to uncover mechanisms that only exploit properties of the system where its deterministic nature remains intact.

Therefore, this problem must be addressed carefully, given that intermittency in the supply dynamics of clean energy sources is orthogonal to delivering deterministic performance.

- Q3. *How to efficiently incorporate low-latency applications in load shifting strategies harnessing clean energy availability with multi-region cloud environments?* Low-latency applications are often considered inflexible for load-shifting strategies used in multi-region renewable energy harvesting approaches due to the risks of performance compromises in serving them over WAN with unpredictable network performance. However, given the significant demand for low-latency workloads, it is beneficial for cloud providers to leverage those in load shifting. A potential approach to this problem is limiting scenarios where low-latency applications are being served over WAN through efficient application scheduling. In doing so, the provider must explore their workload mix to dynamically identify scheduling opportunities.

- Q4. *How to sustainably maintain the accumulation of embodied carbon in the rapid expansions of cloud compute clusters catering to low-latency generative AI applications?* The recent surge in AI is predominantly driven by large language model-based generative AI applications. However, it is important to sustain the embodied carbon accumulation from cloud providers expanding their generative AI compute clusters to cater to the rising demand. Several key details must be addressed in this research question. Firstly, the provider must identify key elements contributing to embodied carbon growth and carefully filter for the best carbon optimization opportunities. For example, although frequent upgrades of GPU accelerators that are commonly used in such deployments may contribute to embodied carbon accumulation, it could provide important business value in utilizing rapid performance improvements in newer hardware. Secondly, providers must derive long-term planning to optimize the embodied carbon of the filtered elements. Collectively, identifying the most effective aspect of embodied carbon optimization and utilizing better planning in the long term is vital in solving this research question.

1.4 Thesis Contributions

In addressing research problems discussed in Section 1.3, this thesis makes the following contributions:

1. Presents a taxonomy encompassing aspects of resource management for both operational and embodied carbon efficiency in latency-sensitive cloud computing environments and a detailed analysis of existing related literature using the proposed taxonomy.

2. Investigates a combined solution of dynamic power budget coupled with efficient decentralized task scheduling across geographically distributed regions to alleviate power budget violations while utilizing clean renewable energy availability (addresses the Q1).

    - An approach to realize a dynamic power budget using existing power delivery infrastructure of Colocation data center providers.
    - A dynamic decentralized algorithm to solve the multi-objective problem of low-latency task scheduling for the utilization of the proposed dynamic power budget of clean energy integration under realistic WAN settings.

3. Proposes a framework that drives CPU core availability based on the dynamics of renewable energy sources to exploit fine-grained fault-tolerance present in real-time cloud systems, enabling clean energy harvesting without affecting the real-time system's deterministic nature (addresses the Q2).

    - A core-level VM Execution Model to maintain real-time compute performance for application VMs amidst renewable energy dynamics.
    - A server-level VM Packing Algorithm to effectively utilize CPU core availability across servers to alleviate application execution compromises.
    - A practical testbed built on HP ProLiant servers with Intel Xeon Silver CPUs, running an extended version of the OpenStack [33] cloud resource management middleware to evaluate the proposed framework. Each node hosts a daemon service that wraps the Intel idle state management libraries and a set

of VMs executing the rtEval application [7], a comprehensive load testing tool for measuring realtime latency.

- An open-source release of the implemented framework, accompanied by detailed documentation, provided for the benefit of the research community.

4. Proposes a load shifting technique to limit offloading low-latency applications over WAN during cross-region clean energy harvesting by maintaining a static resource availability with CPU Simultaneous Multi-threading (SMT) (addresses the Q3).

   - An SMT pooling approach to maintain static CPU resources amidst dynamic supply valleys and peaks of renewable energy sources.
   - A VM scheduling algorithm to efficiently manage low-latency applications alongside best-effort applications, limiting offloading scenarios of low-latency applications over WAN.
   - Practical implementation of the proposed SMT pooling approach and VM scheduling algorithm.
   - A practical testbed environment with OpenStack [33] and servers with Intel Hyper-threading technology for evaluating the proposed load-shifting technique.

5. Proposes an aging-aware CPU core management technique for extended amortization of embodied carbon in cloud Large Language Models (LLM) inference clusters that are mostly concentrated on server CPU components (addresses the Q4).

   - An investigation into embodied carbon analysis of inference servers and uncovering CPU utilization patterns that provide better opportunities for carbon optimization.
   - A technique for aging-aware CPU core management to increase CPU life and achieve extended carbon amortization long-term.
   - Extensions to a high-fidelity LLM cluster simulator from Microsoft, a hyperscale cloud provider, to model the role of the CPU and its hardware degrada-

tion over time for evaluating proposed aging-aware CPU core management technique.

- An open-source release of the extended simulator, accompanied by detailed documentation, provided for the benefit of the research community.

1.5 Thesis Organization

The structure of this thesis is shown in Figure 1.4. The rest of this thesis is organized as follows:

- Chapter 2 presents a taxonomy and literature review on the resource management aspects of carbon efficiency in latency-sensitive cloud computing environments. This chapter is derived from:
  - **Tharindu B. Hewage**, Shashikant Ilager, Maria Rodriguez Read and Rajkumar Buyya, "Carbon-aware Resource Management in Latency-Sensitive Cloud Computing Environments: A Taxonomy and Review", *ACM Computing Surveys (CSUR)* [Submitted, May 2025].
- Chapter 3 presents a dynamic decentralized task scheduling algorithm to leverage the provisioning flexibility of renewable energy sources for multi-objective optimization of power overdraw and application latency performance management in distributed Micro-Cloud deployments. This chapter is derived from:
  - **Tharindu B. Hewage**, Shashikant Ilager, Maria A. Rodriguez, Patricia Arroba, and Rajkumar Buyya, "DEMOTS: A Decentralized Task Scheduling Algorithm for Micro-Clouds with Dynamic Power-Budgets," *Proceedings of the 16th IEEE International Conference on Cloud Computing (CLOUD)*, Pages: 418-427, Chicago, IL, USA, July 2-8, 2023.
- Chapter 4 presents a framework to harness clean energy for real-time cloud systems with stringiest latency performance requirements and deterministic application execution. This chapter is derived from:

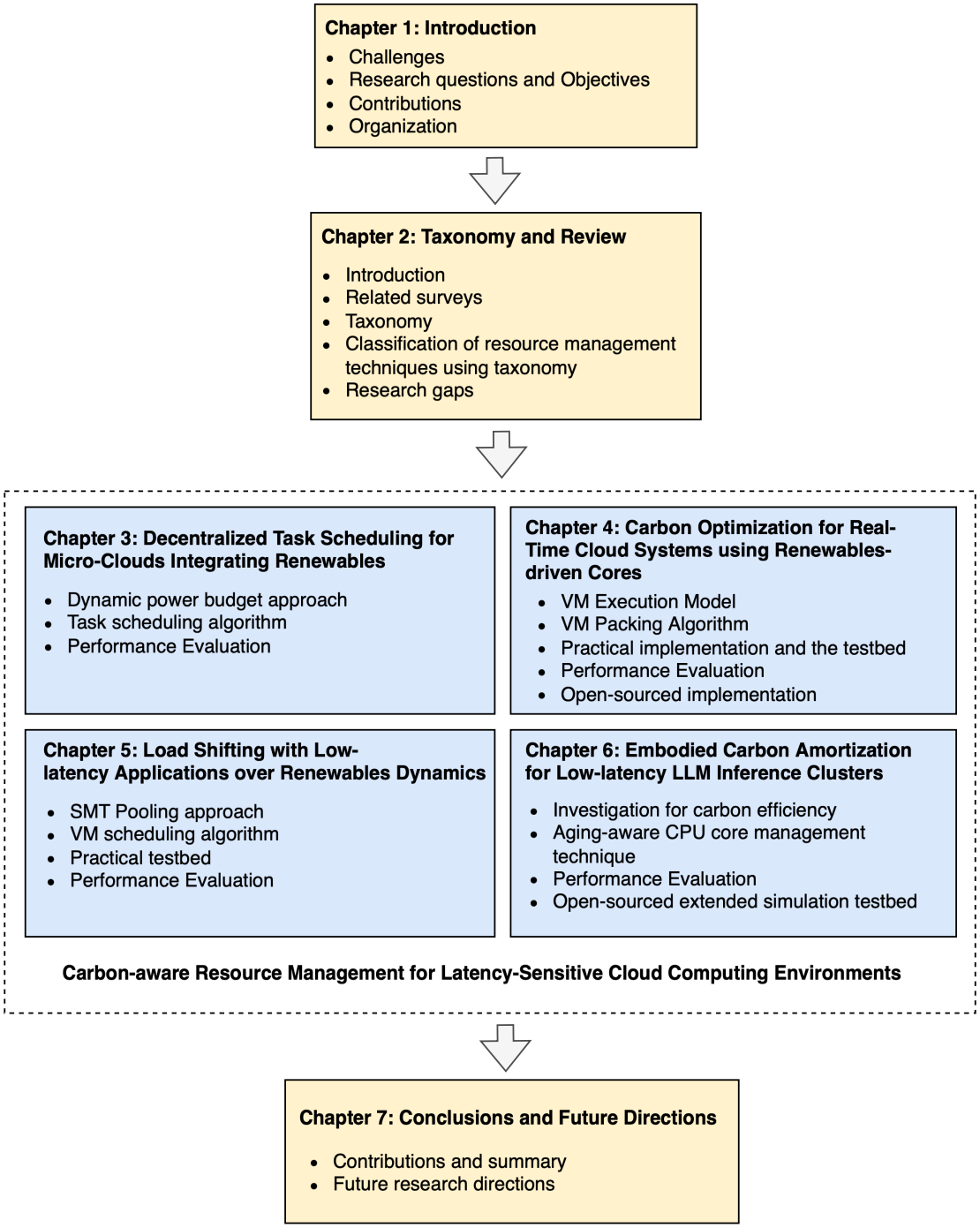


Figure 1.4: Thesis structure.

  - **Tharindu B. Hewage**, Shashikant Ilager, Maria A. Rodriguez, and Rajkumar Buyya, "A Framework for Carbon-aware Real-Time Workload Management in Clouds using Renewables-driven Cores", *IEEE Transactions on Computers*, early access, May 2025.

- Chapter 5 presents a load shifting technique to accommodate low-latency applications in multi-region cloud renewables harvesting via SMT core pooling. This chapter is derived from:

  - **Tharindu B. Hewage**, Shashikant Ilager, Maria A. Rodriguez, and Rajkumar Buyya, "A Technique for Load Shifting Low-latency Applications in Multi-Region Renewables Harvesting via SMT Core Pooling", *IEEE Transactions on Power Systems* [Submitted, May 2025].

- Chapter 6 presents an aging-aware CPU core management technique for extended embodied carbon amortization in cloud-based large language model inference clusters. This chapter is derived from:

  - **Tharindu B. Hewage**, Shashikant Ilager, Maria Rodriguez Read, and Rajkumar Buyya, "Aging-aware CPU Core Management for Embodied Carbon Amortization in Cloud LLM Inference", *Proceedings of the 16th ACM International Conference on Future and Sustainable Energy Systems (E-ENERGY)*, Rotterdam, Netherlands, June 17-20, 2025 [Accepted].

- Chapter 7 concludes the thesis by summarizing its findings and outlining potential future research directions.

Chapter 2

A Taxonomy on Carbon-aware Resource Management in Latency-Sensitive Clouds

This chapter investigates the existing carbon-aware resource management techniques in latency-sensitive cloud computing environments and proposes a taxonomy of elements that influence those techniques under the holistic view of carbon-efficiency over both operational and embodied carbon footprints. We explore workload management, control, environment, and latency SLO aspects in operational carbon management and aspects of the cloud environment's pre-deployment and post-deployment stages in embodied carbon management. After an in-depth literature analysis, we conduct a comprehensive survey and analysis of existing techniques according to the proposed taxonomy. Finally, we identify the gaps in the literature and propose directions for further improvement of carbon efficiency in latency-sensitive cloud computing environments.

2.1 Introduction

The primary difference between carbon-aware resource management for latency-sensitive workloads and carbon optimization in traditional hyper-scale cloud environments is the lack of flexibility in application workloads to compromise performance over carbon efficiency. Cloud carbon optimization involves managing cloud environments, which are complex cyber-physical systems of carbon-intensive stable energy sources, carbon-

This chapter is derived from:

- **Tharindu B. Hewage**, Shashikant Ilager, Maria Rodriguez Read and Rajkumar Buyya, "Carbon-aware Resource Management in Latency-Sensitive Cloud Computing Environments: A Taxonomy and Review", *ACM Computing Surveys (CSUR)* [Submitted, May 2025].

efficient variable availability energy sources, IT assets that guarantee reliable and high performance through carbon-intensive frequent component upgrades, and aging and recycled IT assets with lower carbon intensity with degraded performance and reliability. Carbon-aware resource management refers to the overall aspect of managing the resource requirements of the application workload while optimizing for carbon efficiency, emphasizing harmony between application workload performance and intricacies of cloud environment elements. With the stringiest Service Level Objective (SLOs) of latency-sensitive applications, carbon-aware resource management has fewer opportunities to compromise application workload performance, requiring the exploration of complex application patterns intertwined with deployment architectures to uncover alternative means of carbon optimization while meeting application SLOs. We identify two significant aspects in that, which must be dealt with in a manner suitable for the intricacies of latency-sensitive cloud environments.

- **Operational Carbon Management:** As cloud environments operate, their IT assets consume electricity and produce heat as waste, which requires cooling systems to counter-balance the thermal energy. Energy consumption of IT assets and cooling systems draws power from various energy sources, where the carbon intensity of the source can vary. The direct impact of the cloud environment on carbon emissions of the energy source primarily defines its operational carbon footprint. As the carbon intensity of the energy source reduces, their supply variability increases more often, thus challenging application workload performance. Maintaining required application latency performance while maximizing the integration of low-carbon intensity energy sources requires efficient resource management techniques.

- **Embodied Carbon Management:** Deploying, maintaining, and scaling physical elements of cloud environments incur an indirect carbon cost associated with carbon emissions made during manufacturing, supplying, and recycling IT assets, which are embodied in them. As the demand for latency-sensitive applications increases, cloud environments must scale accordingly while maintaining the performance of IT assets through frequent hardware refreshes, leading to the accu-

mulation of embodied carbon. Providers must employ efficient resource management techniques to slow embodied carbon accumulation while maintaining latency SLOs of application workloads by managing IT assets with heterogeneous reliability and performance characteristics.

Next, we identify the challenges associated with carbon-aware resource management techniques, which are significant in latency-sensitive workloads. We analyze the stated challenges from the perspective of carbon footprint reduction and meeting latency SLOs of end users.

- Diverse latency requirements of application workloads: Latency-sensitive application workloads exhibit varying degrees of latency tolerance in their service level agreements (SLOs). For instance, workloads may allow intermittent latency degradations, which end-user applications can tolerate. However, that may not be possible for time-critical applications, where SLOs expect strict bounds on the response time. Thus, these systems must apply carbon optimizations adapting to the application latency requirements. Further, applications may exhibit different SLOs within their components. Navigating through these complex latency requirements in cloud environments is challenging, where impacting applications SLOs can incur heavy penalties to the cloud operator.

- Variable-availability of carbon-efficient energy: Often, energy sources with lesser carbon intensity are renewable sources with variable availability, such as solar and wind. Compared to stable sources, such as nuclear energy with low carbon intensity, variable availability renewable energy plants can be built and made operational with less cost and time. Additionally, energy capacity availability can spatially vary in geographically distributed cloud environments, which are increasingly prevailing due to the capabilities of low latency application execution near end users. As a result, cloud environments are significantly challenged due to the intermittent availability of low-carbon energy capacities in both space and time to deliver consistent performance to meet application latency SLOs. Not integrating intermittent energy sources could significantly increase the cloud environment's

carbon footprint, whereas the opposite must ensure that application performance is intact.

- Intermittent performance degradations in inter-cloud networks: Cloud environments typically consist of hyper-scale data centers. A typical deployment pattern for latency-sensitive cloud applications is to use networked data centers that are geographically spread. These inter-cloud networks communicate through Wide Area Networks (WAN), which can undergo intermittent traffic congestions, degrading the communication latency. Geo-spread availability of data centers enables cloud operators to redirect workload execution to match the dynamic availability of low-carbon energy capacity during the day. However, WAN traffic congestion challenges the opportunities to do so. Degraded network performance could increase traffic redirection delay, violating application latency SLOs.

- Hard-constraints of resource allocation for latency performance: Specific latency-sensitive applications require deterministic system performance to ensure critical service delivery. For instance, real-time services must meet the hard bounds of their response times. In order to do so, they demand cloud servers to be tuned for high-performance and rigid allocation of computing resources, such as isolated CPU cores for virtual machines and turning off power optimization features of the CPU. As a result, the power management capabilities of the cloud environment are limited, especially when integrating low-carbon renewable energy sources with variable availability. Further, rigid resource allocations make redirecting workloads among sites with better carbon efficiency difficult due to limited placement opportunities satisfying the same placement constraints.

- Resource-constrained cloud environments: In order to satisfy the low-latency serving of application services, some latency-sensitive workloads must be deployed closer to users at the network's edge. These are primarily metropolitan areas where building new data centers can be expensive. A typical deployment pattern is to deploy cloud servers in resource-constrained cloud environments, such as colocated data centers, which allocate specific space and power budget limits to servers. In return, the site's capability to utilize low-carbon energy sources or headroom with

low-carbon servers is limited. For the provider, efficient use of the sites to reduce their carbon impact under resource-constrained environments becomes a complex task.

- Degradation of IT assets: Due to strict performance requirements, low-latency applications may use underlying hardware in an unsustainable manner, leading to premature degradations and carbon-costly frequency hardware replacements. For the cloud operator, alleviating hardware degradation is paramount for both cost and carbon. Since most hardware degradations are slow processes, understanding application patterns that lead to degradations over time and uncovering opportunities to optimize them while maintaining short-term application latency performance can be challenging.

Both aspects of resource management identified above need to be addressed, considering the discussed challenges. Researchers have experimented with various techniques to overcome these challenges and derive better system architectures, workload execution patterns, resource allocation techniques, and resource scheduling techniques. In this chapter, we conduct an in-depth review of the existing literature and identify a classification of the aspects that influence their decisions. In the classification, we discuss inherent challenges and concerns related to those aspects in the context of carbon optimization for latency-sensitive application workloads.

Carbon optimization in cloud environments is determined by the features of the underlying system and the characteristics of the workload latency SLOs. Similarly, our classification comprises key design aspects of the systems, characteristics of the latency-sensitive workloads that the infrastructure is designed for, and the goals of the resource management. Further, using our proposed taxonomy, we summarize existing research on operational and embodied carbon aspects, system architecture designs, and resource scheduling techniques in executing latency-sensitive application workloads. Moreover, we propose ideas for future work to advance resource management in this context. Cloud operators would benefit from our classification by understanding the key focus areas of carbon optimization under latency-sensitive cloud environments and the existing approaches that have already been evaluated. Researchers studying resource

management techniques can refer to existing approaches and future work ideas, which could subsequently form the basis for designing novel techniques. The classification provides a holistic overview of existing and emerging literature for carbon optimization in both embodied and operational carbon aspects.

The rest of the chapter is organized as follows. An overview of the existing surveys and studies on carbon-aware resource management in latency-sensitive cloud environments is provided in Section 2.2. Section 2.3 presents the proposed taxonomy of resource management. Section 2.4 summarizes existing works on carbon-aware resource management in latency-sensitive cloud environments based on the taxonomy. Finally, Section 2.5 discusses the identified gaps in the literature, and Section 2.6 concludes this chapter.

2.2 Related Surveys

Literature surveys investigate existing literature for an area of interest to understand its current state of progress better. In that regard, many surveys study the carbon efficiency aspect of cloud environments. Preliminary surveys characterize carbon efficient cloud computing as green cloud computing emphasizing integrating low-carbon intensity renewable energy sources [34–36]. In that, they study diverse aspects of resource management, such as workload scheduling, power management, and electronic waste management [34–36]. However, they model for generic abstractions of application workloads, such as either batch or interactive workloads, and generic cloud data centers with mostly the same hyper-scale data center system architectures designed for those. In contrast, more recently, a branch of surveys can be seen, which study carbon efficiency in cloud environments that are designed for emerging latency-sensitive workloads [37–44]. In their studies, application latency-specific requirement are studied, such as low-latency streaming applications [37, 38] and Internet of Things (IoT) applications [39–43]. Yet none of these studies consider the holistic view of carbon-aware resource management under latency-sensitive application execution. They consider carbon efficiency as one of many aspects of resource management and often overlook both operational and embodied carbon aspects. Given that power grids continue to decarbonize by

integrating renewable energy sources, embodied carbon is more significant than ever. In this chapter, we address emerging latency-sensitive cloud applications through a holistic study of resource management techniques for both operational and embodied carbon management.

2.3 The Taxonomy

At the topmost level, our taxonomy segregates carbon-aware resource management into the two primary aspects that we identified: operational and embodied carbon management. Operational carbon management comprises workload management, control architecture, system tier, and latency tolerance. Embodied carbon management comprises pre-deployment and post-deployment stages. Figure 2.1 illustrates the proposed taxonomy. In the following sections, we discuss each category in detail, referring to techniques explored in literature so far.

2.3.1 Operational Carbon Management

Most works on carbon efficiency focus on reducing carbon emissions at the operational level of the cloud environment. Here, the foundational approach is to utilize low-carbon-intensive energy sources to power the IT assets. In doing so, resource management techniques are challenged by the variable availability of the renewable energy sources present today, such as solar and wind. Operators opt for variable-availability renewable energy plants due to the lesser cost and deployment times, in contrast to low-carbon-intensive energy plants that provide relatively stable electricity generation. For instance, nuclear and hydro plants capable of providing stable, low-carbon-intensive energy can take many years to finish construction, whereas a solar power plant can be deployed in a few years. Adversely, cloud operators integrating renewable energy sources must match supply dynamics with the data center power load, which is challenging due to timing constraints of latency-sensitive workloads. For that, resource management techniques must address the integrated management of application latency performance and data center power load.

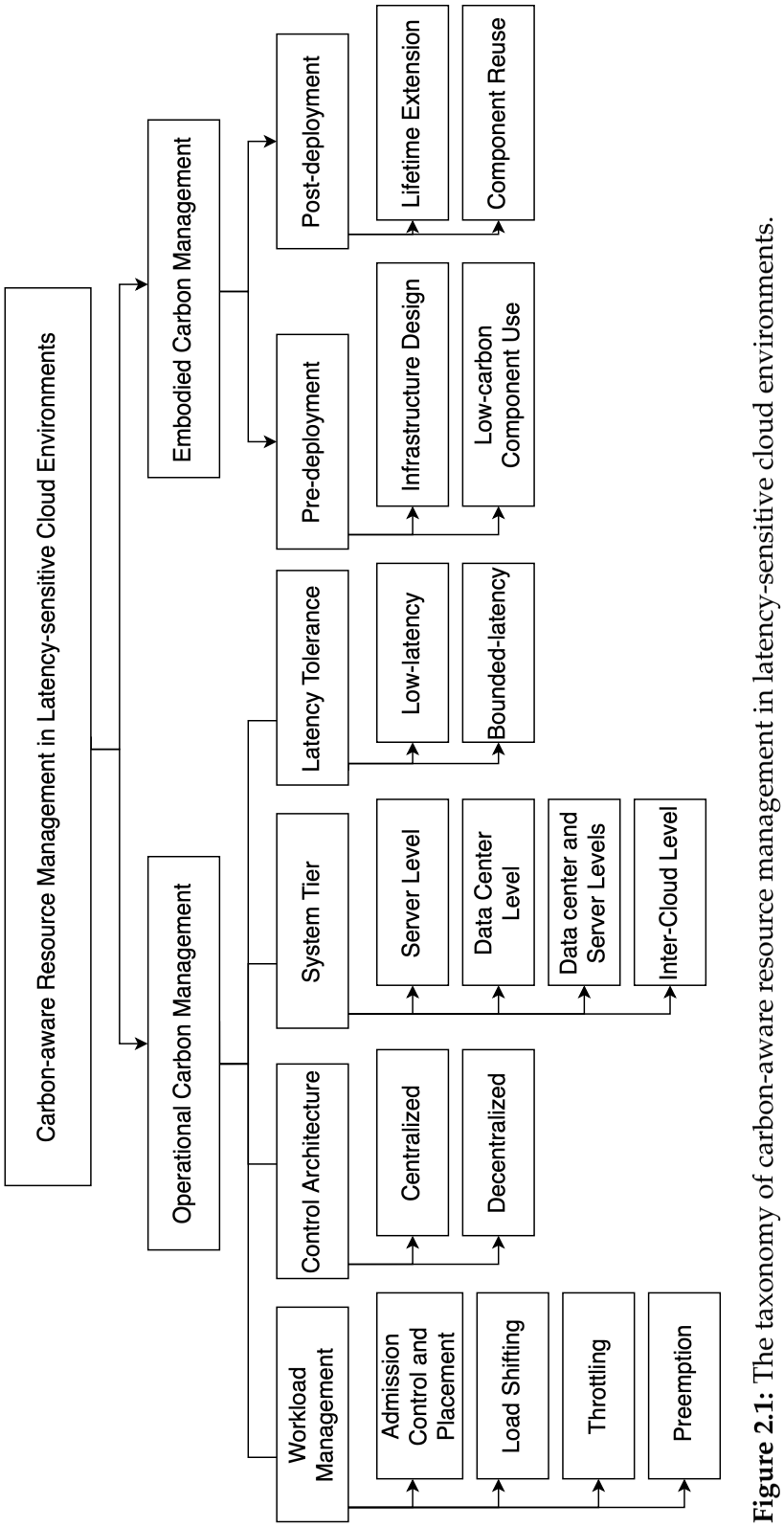


Figure 2.1: The taxonomy of carbon-aware resource management in latency-sensitive cloud environments.

This section identifies and briefly reviews their methods under the key resource management aspects of operational carbon management.

Workload Management

Workload execution yields dynamic power consumption patterns in cloud servers. In clouds, workload execution is carried out in isolated environments from the cloud operator due to privacy concerns, leaving the operator with high-level abstractions of running workloads. In return, the operator must utilize those abstractions to match the server load to the dynamic availability of renewable energy sources. In the following sections, we discuss prominent methods present in the literature in doing that, emphasizing how existing works leverage those for latency-sensitive workloads.

Admission Control and Placement: Distributed system architectures of cloud environments typically consist of multiple data centers or compute clusters that provide heterogeneous performances for latency and carbon efficiency. Therefore, cloud providers have the opportunity to optimize carbon efficiency via the admission control or placement decisions made upon the arrival of workloads. In that, workloads are scheduled to execution sites, improving renewable energy utilization with minimum impact on the latency performance.

Yuan et al. [45] study the problem of scheduling tasks among distributed green data centers to meet their response time constraints. Data centers in their system model integrate on-site renewable energy and periodically relay energy metrics to a centralized task scheduler, such as the electricity price of the power grid and the conversion rate of wind and solar radiation to electricity. Users' tasks arrive at the centralized scheduler and are queued according to the application to which each task belongs. They propose a task scheduling algorithm to split the queued tasks among distributed green data centers to minimize the combined energy cost of both power grid and renewable energy while strictly meeting the task's delay-bounded constraints. In contrast to centralized task arrival, tasks can also arrive in each compute region. Chien et al. [46] design request direction algorithms to minimize the carbon cost of latency-sensitive generative AI inference requests. Their system model consists of multiple compute regions with

different carbon intensities. When a compute region receives a user request, it will be directed to the region with minimum carbon intensity, where the placement decision depends upon the calculated latency of both computation and round trip network time for each compute region. Admission control and placement also fit well with workloads with bounded response times, such as real-time services, because carbon efficiency can be estimated prior to service deployment. Kaur et al. [47] investigate scheduling containers of real-time services in the edge-cloud continuum. They schedule containers among Kubernetes clusters having on-site renewable energy integration. Their scheduling algorithm encompasses carbon footprint minimization and constraints to maintain the real-time performance of services, such as reducing network interference.

Load Shifting: Load shifting involves either suspending/resuming workloads (i.e., shifting in time) or geographically migrating workloads (i.e. shifting in space). Both aim to match the abundance of renewable energy, which varies based on time or location. Load shifting with latency-sensitive workloads primarily leverages shifting in space to avoid latency violation risks of suspending workloads. Shifting in space involves chasing renewable energy availability across geographical locations, in which the network latency performance and scheduling overhead must be efficiently managed.

Sajid et al. [48] proposes connecting geographically distributed data centers with high-performance networks and addressing scheduling overhead with a blockchain-based decentralized approach. Sun et al. [49] leverage space shifting but use it as a last resort to incur a minimum impact on workloads. In return, they achieve average space shifting to 0.015 times per VM per hour, minimizing the associated latency overhead. For workloads that allow a tolerable latency slack, Sukprasert et al. [50] conduct space shifting within a subset of locations that incur latency impact within the tolerable slack. Limiting the use of a subset of locations can reduce overall carbon efficiency. To mitigate that, they demonstrate complementing existing workloads with shiftable batch workloads. Murillo et al. [51] apply exploitation of tolerable latency slack in content delivery networks (CDNs) and show that in CDNs, increasing the latency slack to 60ms can lead to over 60% reduction of carbon emissions. To do so, they complement space shifting with a second form of shifting, called capacity shifting, which moves CDN capacity to greener regions at a coarser time scale of days or weeks. Another opportu-

nity is the workload portability present within applications. Gsteiger et al. [49] exploit space-shifting opportunities within serverless applications. They leverage functions not in the application workflow's critical path and use them to perform space shifting. As a result, end-to-end application latency is left intact. Space-shifting opportunities are also present in specific application load-balancing scenarios. Souza et al. [52] exploit space-shifting opportunities in distributed web services. They conduct dynamic server provisioning and load balancing for dynamic carbon intensities and network latency constraints across geo-distributed regions.

Throttling: Throttling is an in-place technique to dynamically adjust the server performance without shifting the running load. Using throttling, the cloud operator can degrade the servers' performance to reduce its power draw, matching the variable availability of renewable energy. In return, servers take a prolonged time to execute workload instructions, impacting workload service quality such as increasing the response latency in latency-sensitive applications. Therefore, throttling must be carefully orchestrated so that its impact on the application does not violate service level objectives (SLOs).

Li et al. [53] use a data center power delivery architecture that provides access to both renewable energy sources and energy storage and propose a CPU hardware controller called Chameleon. Chameleon switches between two modes of power management in matching renewable energy dynamics: an energy mode that throttles server performance using dynamic voltage and frequency scaling and a performance mode that uses storage energy for power deficiencies. When applied, DVFS degrades CPU performance, affecting running workloads. The authors employ a reinforcement learning-based controlling mechanism to efficiently manage mode switching to mitigate that. Jahanshahi et al. [54] propose a data center power shaping framework that leverages throttling through DVFS to match the dynamics of renewable energy. They co-locate latency-critical workloads and latency-tolerant complementary workloads. Then, CPU cores are grouped into three categories: working, offset, and free. Latency-critical workloads are then pinned to working cores with minimal throttling possibilities, and the remaining cores are allocated for complementary workloads. With that, they leverage data center-level power shaping signals to manage core groups efficiently, balancing throttling and workload performance.

Sharma et al. [55] propose an application-independent power management mechanism called Blink. Blink activates and deactivates servers to yield a duty cycle and, as a result, can control the average power consumption of servers. Applications that can leverage Blink architecture modify their design to adopt that. The duty cycle in Blink throttles servers, which is specified through a blinking policy. Blinking policy balances power management and application performance depending on the application. Experiments with latency-sensitive applications such as Memcached show blinking can be adopted with only a modest throttling overhead. Govindan et al. [56] experiment battery energy storage to utilize renewable energy and use application throttling to sustain battery longevity via lesser frequent discharge cycles. In combining both, they aim to balance between application throttling and premature battery failures. Agarwal et al. [57] propose application throttling through host virtualization to match renewable energy variations. They dynamically change the number of physical cores mapped to virtual machines while keeping the number of virtual cores intact. In return, application interruptions are avoided, yet CPU performance can be throttled. Their approach applies to applications that can tolerate a specific degree of degradation of latency performance. Souza et al. [58] propose a virtualized data center-level energy system to allow applications to control the energy sources according to their workload patterns. In return, applications can better adapt to renewable energy dynamics by exploiting application-specific usage patterns. Here, applications are provided with APIs to throttle their application containers through power capping to control their power draw, which is then exercised to match their latency SLOs better.

Preemption: Isolated execution environments in clouds typically do not reveal underlying infrastructure dynamics to users. For example, a virtual machine's Service Level Objectives (SLOs) are to provide a dedicated and stable physical machine, regardless of underlying resource allocation dynamics. Adversely, cloud operators lose the opportunity to offload infrastructure dynamics to end users where applicable, such as absorbing variable availability of renewable energy. However, recent cloud offerings, such as preemptible VMs [59, 60], attempt to narrow that by providing virtual machines that reflect infrastructure-level dynamics to users. Preemptible VMs unlock the opportunity to relay variable availability of renewable energy to the application layer in specific latency-

sensitive cloud environments, given that the application layer provides fault tolerance. Preemption allows reducing server load upon loss of renewable energy capacity through fault tolerance.

Sun et al. [49] leverage preemptive VMs to delay drawing power from the carbon-intensive grids in geographically distributed networks of modular data centers integrating renewable energy. Their application layer allows users to define fault tolerance by deploying both preemptive and regular VMs. In case of a loss of energy capacity, they first attempt to reduce the data center power draw via VM migrations. If that is insufficient, they opt to shut down preemptive VMs to further reduce the data center power draw rather than sourcing energy from the carbon-intensive power grid. In doing so, an up-time threshold is maintained for preemptive VMs, and VM shutdowns are conducted to adhere to that.

Control Architecture

Integrated management of workloads and variable availability of renewable energy sources requires coordination between different resource management elements such as workloads, data centers and servers, and power delivery. In that regard, there are two prominent control architectures: centralized and decentralized. In this section, we discuss how related works apply their chosen control architecture, adhering to the latency constraints of their cloud environment design.

Centralized: A centralized architecture enables a single point of control, providing an ease of management to the cloud operator. For instance, it allows operators to monitor the cloud environment and execute scheduling logic centrally, yielding its better maintenance, such as extending scheduling logic for future requirements. Further, system elements such as servers and data centers in centralized control are synchronized by design, eliminating the need to perform complex state transfers.

Yuan et al. [45] leverage centralized control to schedule tasks for applications replicated across multiple data centers that integrate on-site renewable energy. They employ per-application queues and execute scheduling logic to optimize green energy while meeting delay-bound constraints. Jahanshahi et al. [54] design PowerMorph: a power

reshaping framework to support frequency regulation requirements of the power grids that can integrate renewable energy sources. PowerMorph reshapes the power consumption of data centers that execute latency-critical applications. A critical challenge in their design is that frequency regulation bids happen every hour, yet the data center should follow the regulation signal every two seconds, where the latter does not provide enough opportunity to perform cluster-level optimization. To overcome that, they offload it to the server level and utilize centralized control at the data center level to conduct regulation provisions.

Sharma et al. [55] propose an application-independent centralized control plane that manages cluster power. Through the APIs, latency-sensitive applications interact with the control plane to regulate their power consumption through opportunities present in specific application patterns. In return, the cluster can integrate renewable energy sources while offloading load adjustment to applications. Govindan et al. [56] introduce energy buffers using batteries to integrate renewable energy in data centers. They use a centralized peak power budget enforcer component to leverage a hybrid approach combining batteries and server throttling. The hybrid approach performs better in dynamically adapting workload service level agreements, such as latency constraints. Sun et al. [49] use a centralized VM scheduler to obtain the global perspective of a modular data center network that integrates on-site renewable energy. It then uses that to find the best fit for VMs to achieve optimized decisions across a mix of modular data centers, priority VMs, and non-priority VMs. In return, the least interruptions are made with priority VMs, such as VMs executing latency-sensitive workloads. Murillo et al. [51] conduct load shifting in content delivery networks deployed on geographically distributed edge data centers that integrate renewable energy. They implement a centralized management component that provides an information service for weather and energy costs, capacity, and demand. As a result, the central component makes informed decisions about spatial load balancing and capacity shifting, considering trade-offs between latency, carbon, and cost. Gsteiger et al. [61] leverage centralized carbon forecasting, pricing, and transmission latency to offload serverless workflows across geo-distributed regions with different carbon intensities. Through centralized management, they identify and deploy serverless functions that have the potential to offload while keeping

latency-sensitive functions intact. Similarly, Souza et al. [52] conduct centralized carbon forecasting, load balancing, and provisioning for distributed web services to minimize emissions while reducing the latency caused by load balancing. Kaur et al. [47] facilitate real-time services across edge nodes integrating on-site renewable energy. In that, a centralized controller runs scheduling logic to schedule applications to meet energy and performance obligations.

Decentralized: In decentralized control, cloud environments achieve improved scalability and autonomy of individual elements. Further, it eliminates single point of failures in the system design. Nevertheless, decentralized control can introduce additional synchronization overheads between individual system elements. As a result, decentralized control is rarely applied in the literature. However, given that the carbon efficiency of renewable energy sources improves as they spread spatially, decentralized control can deliver scalable cloud environments that provide low-latency application performances, while effectively utilizing geographically spread low-carbon intensive energy sources.

Sajid et al. [48] propose a blockchain-based decentralized workload distribution and management model for geo-distributed data centers. It optimizes variability in renewable energy generation as a cost optimization problem and uses a blockchain model to employ a decentralized control architecture. Their work is designed for low-latency networks. Nevertheless, their approach can increase both request scheduling times and processing overheads, affecting application latency performances. To mitigate that, they design their management technique to optimize for both. Li et al. [53] integrate renewable energy in a decentralized manner by allocating individual power budgets to servers and introducing each with a power management microcontroller. In return, each server aims to maximize renewable energy usage individually. Server performance, such as workload latency, is maintained through the power profiles of the microcontroller, which switches between an energy-oriented profile or a performance profile to maintain adequate workload performance while utilizing both renewable energy and battery-stored energy. Agarwal et al. [62] similarly integrate renewable energy. They drive the availability of CPU cores to match renewable energy availability individually at the server level in a decentralized manner. They modify the virtualization layer to efficiently manage dynamic CPU core availability so that virtual machines can continue

executing, eliminating service interruptions.

System Tier

Carbon-aware resource management techniques can be applied at multiple system tiers of the cloud environments. Given the diverse latency SLOs of latency-sensitive applications, each tier offers different optimization opportunities often specific to application dynamics. In this section, we discuss relevant works in the literature that apply operational carbon optimization at various system tiers.

Server Level: Server-level integration of renewable energy commonly involves improving the power delivery architecture of the data center to supply a particular renewable energy allocation to each server. Then, server-level workload management ensures the server load adheres to the energy allocation. As a result, power management can be conducted without migrating workloads between servers, eliminating associated temporary service blackouts that impact application latency performance.

Li et al. [53] leverage server-level energy allocation and introduce a micro-controller in each server to match the energy dynamics by throttling the server performance when needed, given that the workload throughput constraints allow that. Otherwise, battery-stored energy is used to match energy availability. Similarly, Agarwal et al. [62] use a server-level renewable energy budget. They employ dynamic core availability for running workloads to throttle at the server level when needed. In return, workload migration and the associated interruptions are avoided.

Data Center Level: Data center-level integration of renewable energy allows resource management techniques to absorb the variable availability of renewable energy by adjusting the power consumption of servers as a whole. In return, resource management techniques can explore utilization patterns across the data center server to adjust its power consumption while minimizing the application latency performance.

Sharma et al. [55] employ an application-independent power management framework that allows defining policies to engage servers in the data center to yield a duty cycle of activation and deactivation, thus reducing the average power draw of the data center to match renewable energy availability. They demonstrate that specific distributed

latency-sensitive applications integrated into the framework through their APIs can adapt their application patterns to deliver acceptable performances. Govindan et al. [56] apply energy buffers in the data center power delivery with UPS batteries to absorb renewable energy variations. They use server throttling at the data center level to optimize battery life and application performance, such as workload latency. Souza et al. [58] leverage data center-level renewable energy integration to provide virtualized energy systems to applications. In return, individual applications executing in the data center manage grid energy, renewable energy, and batteries through virtualization, employing application-specific carbon budgeting policies that allow meeting their latency constraints.

Data center and Server Levels: Given application characteristics, integrated management at both data center and server levels can yield better integration of renewable energy. In that, server-level integration can enable workload management, avoiding workload migrations, while data center-level management can complement that across the servers.

Jahanshahi et al. [54] design a power reshaping framework for data centers catering to energy capacity signals from the power grid, such as the varying capacity of renewable energy integration. They execute both latency-sensitive and complementary workloads in servers. Due to short intervals of power shaping and long intervals of power biding with the grid, they employ server-level power management via workload throttling and core allocation among latency-sensitive and complementary workloads while conducting data center-level power biding at longer intervals.

Inter-cloud Level: Integrating renewable energy at the inter-cloud level allows renewable energy availability to be utilized in different geographical locations. Here, the challenging aspect is maintaining the application performance of latency-sensitive workloads over the communication overhead of the network that connects geographical locations. In that, resource management techniques aim to tackle the variable availability of renewable energy by directing workloads to locations with sufficient energy availability while adhering to latency constraints.

Yuan et al. [45] conduct spatiotemporal task scheduling across geographically distributed green data centers. Their task scheduling problem applies a delay-bound con-

straint to maintain workload latency performance, and scheduling algorithms are designed to adhere to that. Sajid et al. [48] design their work for high-performance inter-cloud networks. Hence, they optimize application latency performance by minimizing both processing overhead and inter-cloud workload migration delays. Sun et al. [49] co-locate complementary workloads with latency-sensitive workloads in geo-distributed clouds, and aim to reduce the workload migration frequency of latency-sensitive workloads to minimize their latency impact. Sukprasert et al. [50] explore inter-cloud workload execution for latency-sensitive interactive workloads to harness renewable energy. They migrate interactive workloads to greener locations if latency constraints allow it. Murillo et al. [51] apply inter-cloud renewable energy harnessing to content delivery networks. They combine request latency-aware workload shifting with VM capacity shifting while maximizing renewable energy utilization within latency boundaries. Gsteiger et al. [61] utilize inter-cloud renewable energy availability for serverless applications. They segregate functions from serverless application workflows that can be offloaded without increasing end-to-end latency, and use that to utilize energy availability across clouds. Souza et al. [52] harness renewable energy across clouds for distributed web services. They provision resources based on carbon forecasting and conduct load balancing across clouds within the latency constraints. Chien et al. [46] investigate inter-cloud serving of generative AI requests. Directing requests to locations with minimum carbon intensity shows that renewable energy can be utilized without significantly impacting request latency. Kaur et al. [47] leverage a multi-cluster deployment across different locations to execute real-time services. They propose a controller to utilize renewable energy across locations while maintaining adequate application performance.

Latency Tolerance

Application workloads in latency-sensitive cloud environments exhibit various service level objectives (SLOs) in their latency performance. We classify them into two primary categories: Low-latency and Bounded-latency. Low-latency applications can tolerate latency performance degradations to a specific level, where resource management techniques aim to improve their service quality by minimizing such degradations. Bounded-

latency applications, such as real-time applications, have bounded latency responses. For those, resource management techniques must strictly meet the response time boundaries. In this section, we discuss how relevant works achieve that in utilizing the variable-available renewable energy sources.

Low-latency: Low latency applications that tolerate certain levels of latency degradation provide better flexibility in adapting to performance optimization of variable-available renewable energy integrations. As a result, most related works exploit that in two primary aspects: server performance throttling to match renewable energy availability and workload migration across geographical locations for greener energy availability.

Li et al. [53] leverage dynamic throttling of server performance to engage an energy-efficient power profile that harnesses renewable energy. They employ techniques to dynamically switch between a high-performance power profile to minimize the impact of workload latency from server performance throttling. Jahanshahi et al. [54] use Dynamic Voltage Frequency Scaling (DVFS) to throttle CPU core performance, matching renewable energy availability. They minimize latency increases for latency-critical workloads by throttling cores allocated to complementary workloads first. Similarly, Govindan et al. [56] use DVFS-based throttling to optimize between latency performance and longevity of battery energy storage, and Agarwal et al. [62] use throttling through shrinking CPU core availability to exploit latency performance for renewable energy harnessing. Sharma et al. [55] use application-independent blinking of server activation to match renewable energy availability, and offload optimizing latency performance over throttling to the application. A similar approach can be seen with Souza et al. [58], where a virtualized energy system allows applications to manage to throttle their containers according to workload patterns while minimizing the latency performance impact over variable availability of renewable energy.

Besides server performance throttling, many works exploit low latency for renewable harnessing across geographical locations. Sajid et al. [48] exploits latency flexibilities in workloads with intermittent interruptions of workload availability to shift and execute them in greener data centers. They propose an optimized scheduling approach to minimize the interruption durations. A similar workload migration approach is used by Sun et al. [49] for modular data centers. They prioritize migrating VMs with fewer

VM states to reduce the impact of migration overhead on latency performance. Murillo et al. [51] explore adjusting response latency time in content delivery networks to better harness renewable energy across edge data centers. They show that increasing latency within safe limits can reduce carbon emissions up to 35.5%. Gsteiger et al. [61] leverage low latency in serverless applications for carbon optimization by segregating functions in application workflows, and offloading those with lesser latency impact to greener locations. Souza et al. [52] increase request latency in distributed web services for renewable energy harvesting across locations by combining load balancing and resource provisioning. Chien et al. [46] apply renewable energy harnessing across geo-spread locations to serve generative AI requests, and show low-latency characteristics of requests allow operating within safe limits.

Bounded-latency: Exploiting applications with bounded latency constraints for renewable energy integration is less common. This is due to the rigid nature of latency upper bounds, which do not provide much room for resource management techniques to absorb renewable energy variations.

Yuan et al. [45] leverage a scheduling algorithm that strictly guarantees the task's delay-bound constraints while maximizing the usage of renewable energy across distributed green data centers. Kaur et al. [47] explore real-time services with well-defined response time boundaries for harvesting renewable energy across Kubernetes clusters with on-site renewable energy integration. They propose a controller to optimize green energy utilization and performance impacts from application inferences in container scheduling.

2.3.2 Embodied Carbon Management

Management of embodied carbon is broad, spanning across the lifecycle of IT assets. We classify those into two primary stages: Pre-deployment and Post-deployment. We then discuss related resource management techniques in planning, designing, managing, and recycling IT assets in cloud environments, referring to the primary stages that we identified, while narrowing our focus for those considering the impact to application latency performance.

Pre-deployment

Since embodied carbon management is conducted through the management of IT assets, the pre-deployment stage of cloud environments approaches that at the infrastructure design. A well-designed infrastructure for carbon efficiency reduces the need to optimize again once the infrastructure is available to execute workloads. Here, we focus on two primary aspects: infrastructure design and low-carbon component use, emphasizing the latency performance of running workloads as a design goal.

Infrastructure Design: Cloud environments can exhibit heterogeneous designs depending on the workload performance requirements. Each design can offer specific opportunities to improve embodied carbon efficiency. Moreover, specific tools in estimating embodied carbon footprints can lead to better designs. Finally, techniques that maintain adequate application latency performance with a reduced hardware footprint can also reduce the embodied carbon footprint.

Sun et al. [49] investigate geographically distributed data center networks capable of serving low-latency workloads. They use modular data centers that integrate renewable energy sources (rMDC) in their design. rMDC incurs a lesser embodied carbon footprint than traditional data centers. Their work enables colocating rMDC with more stable energy sources, reducing the number of servers required to utilize peak energy spikes, thus further reducing the embodied carbon. Ji et al. [63] design a carbon estimation tool for servers with accelerators, such as GPU and FPGA, that serves latency-sensitive workloads. They advance accurate embodied carbon estimation and enable operators to derive improved carbon efficiency in their designs. Tannu et al. [64] explore embodied carbon intensity in storage mediums. Their work provides insightful guidance on selecting carbon-efficient storage mediums like HDD or SSD. SSDs provide better performance for workload latency than HDDs, yet their embodied carbon efficiency can differ. Therefore, carbon insights help operators make design decisions to balance workload performance and embodied carbon efficiency. Gupta et al. [65] aim to shrink the overall hardware footprint of the cloud environment using resource oversubscription, reducing underutilization and embodied carbon footprint. They design a dynamic system to observe resource underutilization and present a dynamic resource leasing platform. Through that, their system can execute latency-sensitive workloads with capacity avail-

ability service level agreements (SLOs) over the oversubscribed infrastructure. Results show that infrastructure shrinking as much as 25% is achievable.

Low-carbon Component Use: Opting for components with lower embodied carbon footprints enables operators to optimize the carbon efficiency of their cloud environment design. However, the associated performance bottlenecks must be identified, and suitable techniques must be employed to mitigate those.

Zhong et al. [66] propose a tiered memory system that reduces embodied carbon footprint. Instead of local DRAM, they use a hardware-managed tiered memory system for low-carbon CXL-based memory. Although the embodied carbon footprint of CXL is lower, it can incur higher latencies than local DRAM. They introduce a software stack to manage that. The combined hardware-managed tiering system and the software stack provide memory performance closer to local DRAM.

Post-deployment

Once deployed, a cloud environment's embodied carbon footprint can be optimized by managing installed IT assets. Most works aim to extend an asset's operating life or reuse older components. At the post-deployment stage, the embodied carbon footprint is already acquired. Thus, extending the asset's lifetime or reusing older components allows operators to amortize the acquired carbon further. However, using aged components can increase component failure risks in the cloud environment. Resource management techniques must mitigate that by exploring opportunities in hardware-software settings while maintaining adequate workload latency performances.

Lifetime Extension: Resource management techniques that optimize embodied carbon footprint through the asset's lifetime extension exploit resource usage patterns specific to the hardware in focus. They manipulate resource usage patterns, and through that, they improve asset longevity. The resulting lifetime extension allows for further amortization of the asset's embodied carbon footprint.

Zhao et al. [67] aims to reduce uneven core wear-off in multi-core CPUs from executing workloads with core affinity, such as real-time workloads requiring deterministic performance. In return, premature failures of specific cores can be avoided for extended

CPU life. They provide a performance metric independent of CPU micro-architectural characteristics to measure uneven CPU core usage, which can be used at the cloud resource management layer. Leveraging the proposed metric, the workload scheduler can shift workloads between CPU cores based on the CPU stress. Wang et al. [68] employ an aging-aware workload scheduler to maintain workload performance among servers with heterogeneous aging characteristics. They identify older servers that could maintain sufficient performance under specific conditions, such as during low load times, and use that knowledge in the workload scheduler. Tannu et al. [64] reduce SSD wear-off in storage server fleets. They employ dynamic data redirection to locations with lower write intensity, evens out SSD writing across the server fleet, reducing their aging rate and allowing extended embodied carbon amortization. Similarly, Gupta et al. [69] reduce the SSD aging rate by increasing over-provisioning in their cloud environment setting to reduce the write amplification factor. McAllister et al. [70] exploit premature failure risks in flash-based cache. Instead of a legacy logical block addressable device interface, they propose fairyWREN, a flash cache designed for write read erase interface (WREN). WREN allows application control over data placement and garbage collection, which fairyWREN uses to reduce writes via caching policies. fairyWREN has a better read latency at peak load, improving the latency-sensitive performance of workloads and extending flash storage lifetime for improved embodied carbon amortization.

Component Reuse: Used components can be employed in data centers to reduce acquired embodied carbon in newer components, given that the workload performance, such as latency requirements, is maintained. Used components have already amortized their initial embodied carbon footprint to a certain degree; thus, using them reduces the overall embodied carbon footprint of the cloud environment.

Wang et al. [71] propose a framework to evaluate carbon savings at scale with used components, such as used memory and SSD. It enables providers to evaluate the performance at scale with used components, such as tail latency and low load latency. In return, operators can make an informed decision on using used components in their cloud environment. Tannu et al. [64] propose re-purposing used flash devices. They focus on Multi-level cell (MLC) devices, which store multiple bits within each cell, thus providing higher capacities. However, MLC can rapidly wear out compared to single-cell de-

vices (SLC). The work proposes a strategy to transform used MLC into low-capacity SLC devices, enabling a second life to amortize embodied carbon. Chien et al. [46] explore embodied carbon optimization for the geo-distributed serving of low-latency generative AI requests, where used servers provide headroom in each location. The work evaluates the amount of headroom needed for effective carbon improvements. Gupta et al. [69] explore reusing general-purpose hardware instead of employing specialized accelerators. Their work shows a promising balance between component reuse and workload performance.

2.4 Classification of Resource Management Techniques Using Taxonomy

Table 2.1 and 2.2 review key works on carbon-aware resource management in latency-sensitive cloud environments related to the proposed taxonomy, where Table 2.1 focus on operational carbon management and Table 2.2 focus on embodied carbon management. The works we present here propose novel resource management techniques exploring one or more resource management aspects that we have identified.

Work	Workload Management			Control	Environment			Latency SLO	
	Load Matching	**Granularity**	**Mixed Criticality**		**System Tier**	**Topology**	**Application**	**Tolerance**	**Optimize**
[49]	Load Shifting, Preemption	VM	✓	Centralized	Inter-Cloud	Networked Clouds	Modular DCs	Low-latency	Downtime, Criticality
[51]	Load Shifting	Server Load	-	Centralized	Inter-Cloud	Networked Clouds	CDNs	Low-latency	Response Latency
[61]	Load Shifting	Function	✓	Centralized	Inter-Cloud	Networked Clouds	Serverless	Low-latency	End-to-end Latency
[52]	Load Shifting	Request	-	Centralized	Inter-Cloud	Networked Clouds	Distributed Web Services	Low-latency	Response Latency

[58]	Throttling	VM, Container	-	Centralized	Data Center level, Server-level	Server Fleets	Virtualized or Containerized Apps	Low-latency	Response Latency
[46]	Admission Control and Placement	Request	-	-	Inter-Cloud	Networked Clouds	Gen. AI Serving	Low-latency	Response Latency
[54]	Throttling	Task	✓	Centralized	Data Center level, Server-level	Server Fleets	-	Low-latency	Response Latency
[48]	Load Shifting	Request	-	Decentralized	Inter-Cloud	Networked Clouds	-	Low-latency	Migration Latency
[47]	Admission Control and Placement	Container	-	Centralized	Inter-Cloud	Networked Clouds	Real-Time Services	Bounded-latency	Response Latency

[45]	Admission Control and Placement	Task	-	Centralized	Inter-Cloud	Networked Clouds	Green DCs	Bounded-latency	Response Latency
[53]	Throttling	Server Load	✓	Decentralized	Server level	Server Fleets	Throughput Servers	Low-latency	Response Latency
[55]	Throttling	Server Load	-	Centralized	Data Center level	Server Fleets	Interrupt-tolerable Apps	Low-latency	Response Latency
[56]	Throttling	Server Load	-	Centralized	Data Center level	Server Fleets	-	Low-latency	Response Latency
[62]	Throttling	VM	-	Decentralized	Server level	Server Fleets	Virtualized Apps	Low-latency	Response Latency

Table 2.1: Classification of resource management techniques of operational carbon reduction.

Work	Approach	Pre-deployment		Post-deployment		Optimize	
		Infra. Design	**Low-carbon Components**	**Extended Life**	**Component Reuse**	**Hardware Aspect**	**App Latency**
[49]	Designing, Planning	Energy Stability	Modular DC	-	-	Data Center Footprint	Network Delay
[63]	Carbon Estimation	Carbon vs Performance	-	-	-	Accelerators	Accelerated Computing
[65]	Over - subscription	Size Reduction	-	-	-	Utilization	Capacity-availability SLOs
[64]	Component Selection, Workload Scheduling	Carbon vs Performance	-	SSD Aging	Re-purpose Flash Devices	Storage Medium, Write Intensity	Storage Access
[66]	Component Selection	-	CXL Memory	-	-	Memory	Memory Access

[67]	Workload Scheduling	-	-	CPU Aging	-	CPU Wear-off	Scheduling Overhead
[68]	Workload Scheduling	-	-	Server Aging	-	Servers	Age vs Performance
[69]	Over Provisioning	-	-	SSD Aging	General-purpose H/W for Accelerators	Write Intensity, Accelerators	Storage Access, General-purpose H/W vs Performance
[70]	Workload Scheduling	-	-	Flash Device Aging	-	Storage Write Intensity	Storage Access
[71]	Carbon Estimation	-	-	-	Used Components	Reusing at Scale	Age vs Performance
[46]	Workload Scheduling	-	-	-	Servers	Data Center Headroom	Age vs Performance

Table 2.2: Classification of resource management techniques for embodied carbon reduction.

2.5 Research Gaps

The in-depth review we conducted for carbon-aware resource management in latency-sensitive cloud computing environments highlights open problems with great potential for exploration. This section discusses those areas in detail, alongside the broader categories we have identified, for operational and embodied carbon efficiency. In return, we lay the groundwork for research and development work for the future.

2.5.1 Cloud Environment Characteristics

Emerging low-latency applications such as the Internet of Things (IoT) execute in cloud deployments that rent space and power with distributed resource-constrained environments such as colocation data centers. As tenants, clouds subscribe to limited power budgets from the colocation provider, which may integrate on-site renewable energy in its shared power delivery to improve carbon efficiency. As a result, tenant clouds are challenged with better utilization of renewable energy through shared power delivery, not just locally but across distributed deployments of similar clouds that provide varying renewable energy availabilities depending on the time of the day and the location.

Most low-carbon intensive renewable energy integration solutions for cloud environments use data center-level allocation of renewable energy. Although that allows management of workloads across servers or even between clouds to match the renewable energy capacity intermittencies, server-level or core-level renewable energy allocations can benefit applications with strict latency SLOs preventing workload migrations. However, dynamic resource usage in clouds may allocate workloads unevenly across servers, which may yield underutilization of renewable energy with server or core-level energy allocations. Exploiting opportunities in resource management to improve such is an interesting open problem.

Increasingly popular generative AI-based applications demand low-latency responses, which require cloud servers with accelerators for faster model computation. Although existing works segregating generative AI model computation workflows explore better energy efficiency through older servers, exploring embodied carbon impact in that has not been thoroughly exploited. For instance, defining latency SLOs for generative

AI model inference requests can better inform the request scheduler to balance performance and carbon efficiency among older and newer generation servers, allowing cloud environments to deploy older generation servers or used servers to reduce their embodied carbon footprint.

2.5.2 Application Characteristics and Latency SLOs

Cloud providers increasingly offer infrastructure-as-a-service (IaaS) solutions that reveal application criticality to the cloud provider, such as evictable virtual machines. Leveraging such with application fault-tolerance for renewable energy intermittency is still an open challenge. For example, IaaS can be utilized to deploy application-specific middleware solutions serving latency-sensitive workloads that can tolerate intermittent VM failures. In that, exploiting application-specific latency service level objectives (SLOs) to integrate renewable energy with VM evictability as a load-matching technique requires novel resource management techniques.

Many renewable energy integration techniques for latency-sensitive cloud environments leverage low-latency applications. However, as the cloud paradigm continues to penetrate a wide range of use cases, applications with bounded latency SLOs, such as real-time services becomes significant in the operational carbon footprint of cloud computing. Therefore, exploring potential opportunities to absorb the variable availability of renewable energy sources with bounded latency applications becomes paramount in reducing that.

Another aspect of bounded latency applications is that they often require servers tuned to high-performance power profiles, which can yield server components to stress over longer periods. As a result, server components can undergo premature failures, forcing the cloud provider to engage in frequent server replacements and increasing the cloud environment's embodied carbon footprint. In this context, sustainable usage of cloud resources while maintaining an adequate server performance to meet application latency SLOs becomes critical in managing bounded latency applications.

Opportunities in server longevity for embodied carbon reduction can be seen in increasingly prevailing generative AI inference clusters, which offload most of the compu-

tation stress to accelerators such as GPUs. As a result, resource usage patterns in such clusters can lead to underutilization of resources such as CPUs. Improving resource management techniques to identify underutilization patterns and leverage that to reduce computation stress in server components can improve their longevity, leading to amortizing embodied carbon over the improved lifetime.

2.5.3 Resource Management Approaches

Due to low-latency performance in executing applications at the network's edge, geographically distributed cloud environments are becoming increasingly popular. Nevertheless, carbon optimization in those deployments today mostly leverages centralized control, mainly for the ease of maintenance and management, yielding bottlenecks such as limited scaling capabilities. As a solution, decentralized control can be implemented. However, decentralized resource management requires efficient synchronization between clouds with minimum latency overhead. Exploring application-specific opportunities, such as executing with mixed latency SLOs of both low-latency and bounded components, can lead to better implementation of decentralized control.

Computing models such as serverless computing allow the resource management layer to make granular management decisions. For instance, instead of application-level scheduling, the resource management layer can schedule functions that may collectively conduct an application workflow. As a result, additional opportunities are available to make granular changes to the server power draw, which can match intermittent renewable energy availability. Exploiting that for various renewable energy allocation methods, such as server-level or core-level allocations, and various latency SLOs is an interesting research problem.

Another aspect is the management of thermal load on server components to improve its longevity. Inefficient resource usage patterns could stress server components over time, creating thermal hotspots and leading to premature component failures. For instance, uneven CPU stress of infrastructure tasks such as virtualization and workloads tasks have been identified to create thermal hotspots in CPU cores. Further exploring it for bounded latency applications that may incur similar stress due to their dedicated

allocation of CPU cores and exploiting scheduling those with infrastructure tasks and low-stress application tasks to improve CPU longevity for servers executing mixed latency SLO applications can yield better management of their embodied carbon footprint.

2.6 Summary

In this chapter, we presented a detailed review of the aspect of carbon-aware resource management, focusing on latency-sensitive cloud computing environments. We proposed a taxonomy for a holistic view of carbon-aware resource management in both operational and embodied carbon efficiency. We discussed both aspects of operational and embodied carbon management and analyzed existing works using the taxonomy. Our taxonomy presents an in-depth view of both operational and embodied carbon aspects for cloud operators to identify carbon optimization opportunities to meet their short-term and long-term carbon efficiency goals. Further, it provides the groundwork for researchers to understand existing works in carbon-aware resource management to investigate and build upon their work. Finally, we provide a gap analysis highlighting the identified challenges, emphasizing the great potential for future work.

This thesis explores and addresses some of the identified research gaps. Beyond that, we further outline potential new research directions in the last chapter.

Chapter 3

Decentralized Task Scheduling for Micro-Clouds Integrating Renewables

Emerging latency-critical Internet of Things (IoT) applications increasingly utilize geographically distributed lightweight Micro-Clouds to alleviate network overheads from Wide Area Network (WAN) traffic congestion. These Micro-Clouds often lease infrastructure resources from Colocation data center providers to reduce deployment costs. This chapter focuses on power budget constraints imposed by Colocation providers on the tenant Micro-Clouds, emphasizing utilizing the provisioning flexibility of renewable energy sources to mitigate the application latency performance impact involved. We propose a dynamic power budget for Micro-Clouds using on-site renewables and a decentralized task scheduling algorithm to effectively utilize that across geographical locations. The proposed approach aims to reduce the Colocation provider's power constraint violations by scheduling over dynamic WAN performance and renewable energy availability. We implement and evaluate our approach in a simulated environment. The proposed approach, when compared with state-of-the-art scheduling techniques, can reduce power overdraw impact up to 19%, task latency increase impact up to 47%, and task schedule time impact up to 49%.

3.1 Introduction

The growing use of latency-critical applications in Micro-Clouds, driven by the Internet of Things (IoT), is expected to result in substantial energy demand. Micro-Clouds decentralize traditional hyper-scale clouds by dispersing computational capacity over a

This chapter is derived from:

- **Tharindu B. Hewage**, Shashikant Ilager, Maria A. Rodriguez, Patricia Arroba, and Rajkumar Buyya, "DEMOTS: A Decentralized Task Scheduling Algorithm for Micro-Clouds with Dynamic Power-Budgets," *Proceedings of the 16th IEEE International Conference on Cloud Computing (CLOUD)*, Pages: 418-427, Chicago, IL, USA, July 2-8, 2023.

large number of lightweight data centers deployed at the network's edge [30, 31]. This approach exploits lower communication latency, making them ideal for IoT applications. Compared to traditional hyper-scale clouds, Micro-Clouds have a smaller energy footprint, resulting in smaller power ratings (less than three orders of magnitude than a traditional hyper-scale cloud [30]). However, due to the predicted fast growth rate, Micro-Clouds are expected to consume a similar amount of energy as traditional hyper-scale clouds by 2028 [72, 73].

Due to their lightweight nature, Micro-Clouds offer flexibility in cost-effective deployments. In this regard, deploying Micro-Clouds as tenants in multi-tenant Colocation data centers (hereby used as Colocation data centers) is widely adopted. For example, the data center operator Vapor IO plans to build thousands of edge Colocation data centers, which are a type of Colocation data centers hosting latency-critical IoT workloads [74]. In a Colocation data center, the data center operator manages the data center facility and physical power/cooling infrastructures, and leases them to service providers (i.e. tenants) as a shared space. The service providers deploy Micro-Clouds on the leased resources. In this approach they only have to manage physical servers, thereby significantly reducing maintenance costs [74].

Keeping up with the rapid growth of Micro-Clouds is expensive for data center operators due to significant costs in building new data centers [75]. Because of that, data center operators exploit utilizing existing resources. Typically, additional physical space is already available in data centers. Therefore, the data center operators focus on employing server-level recovery mechanisms to oversubscribe its power infrastructure by provisioning additional servers. This technique is known as power oversubscription and it is a common approach used by data centers [74, 75].

In Colocation data centers, the data center operator cannot employ power oversubscription via server-level recovery mechanisms. This is because the Colocation data center provider does not have control over the servers that are managed by tenants. To overcome this challenge, the data center operator implements a reduced *soft power budget* in the tenant's subscribed power. The difference between the *soft power budget* and the subscribed power capacity is then used to provision additional servers [74]. For example, in the case of a Micro-Cloud deployment as a tenant in a Colocation data center,

the Micro-Cloud operates under a reduced *soft power budget*, such as 90% of the originally subscribed power. The remaining 10% of the power capacity contributes to the provision of additional servers.

However, operating under a *soft power budget* can result in Micro-Clouds having to resort to extreme power reduction measures such as power capping and workload throttling. Especially, when the rare power peaks exceed the available power budget (i.e. *power overdraw events*) [76]. This can have a detrimental impact on the performance of latency-critical workloads. Therefore, balancing between the power oversubscription techniques and the optimal workload performance is an ongoing challenge [75].

In this regard, our work aims to improve workload performance by minimizing *power overdraw events*. As a result, the need to employ extreme power recovery techniques is also minimized. One way to achieve this is increasing the *soft power budget* by combining additional energy. But drawing this additional energy from the power grid is not feasible because the power grids feeding energy to Colocation data centers are already stressed [77]. On the other hand, Colocation data centers already procure onsite renewable energy [78]. Therefore, drawing additional energy from on-site renewable sources is a viable option. The next challenge in this approach is combining additional energy with the *soft power budget* in a cost-effective manner. Such that expensive upgrades in the tenant power delivery system are minimized. When we consider an existing tenant in a Colocation data center, this tenant subscribes to a certain amount of power. This subscribed power is delivered to the Micro-Cloud via a Power Delivery Unit (PDU) with a sufficient power capacity. When *soft power budget* is implemented, a portion of the PDU power capacity is left unused (e.g., for *soft power budget* equivalent to 90% of the subscribed power, a 10% in the PDU capacity is left unused). We identify that this underutilized PDU capacity can be used to combine additional energy with the *soft power budget*, thus avoiding expensive PDU upgrades. Therefore, we propose the addition of renewable power to the *soft power budget*, thereby creating a *dynamic power budget* for Micro-Clouds. The *dynamic power budget* increases the *soft power budget*, thereby minimising *power overdraw events* and the need for extreme power reduction measures. This improves workload performance under power oversubscription techniques being applied. An example of this approach is illustrated in Figure 3.1.

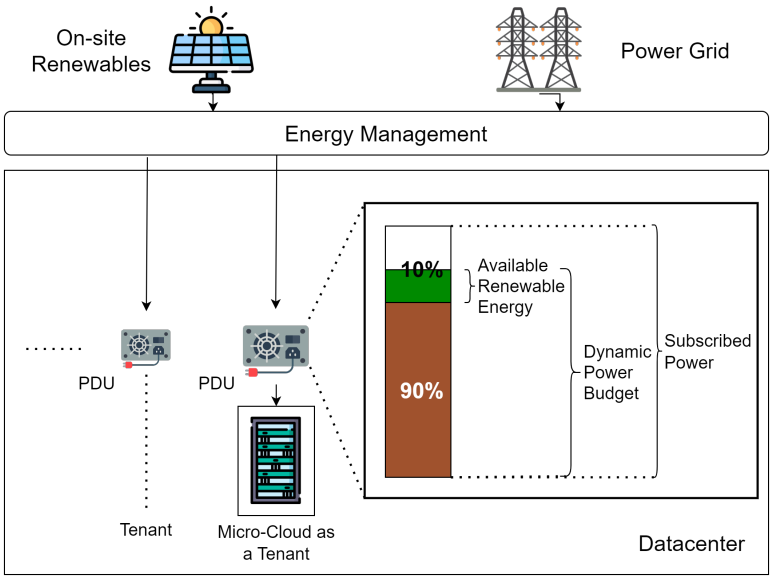


Figure 3.1: Dynamic power budget for the Micro-Clouds.

Furthermore, the *dynamic power budget* can be implemented across a geographically distributed network of Micro-Clouds deployed with Colocation data centers. In such a scenario, excess power in the *dynamic power budget* is highly likely to be available in one or more Micro-Clouds at a given time due to the intermittent nature of the renewable energy across different time zones. This motivates us to avoid employing extreme power reduction measures entirely, by offloading tasks to another Micro-Cloud with available power in its *dynamic power budget*. Concretely, if a Micro-Cloud is about to undergo a *power overdraw event*, its energy consumption can be reduced below its power budget by offloading a portion of its tasks to other Micro-Clouds that have available power in their *dynamic power budgets*. The feasibility of this approach is shown in Fig. 3.2 in which, the *dynamic power budget* is compared across three time zones. We observe that when a *power overdrawn event* occurs at one location, one or more of the other locations have available power in their *dynamic power budget*.

When tasks are offloaded across Micro-Clouds, it is carried out via dynamic task scheduling. Dynamic task scheduling allows for real-time adaptation to changes in the system, albeit at the cost of additional computational overhead, which can be minimized through lightweight computation approaches. Moreover, it is crucial to perform dy-

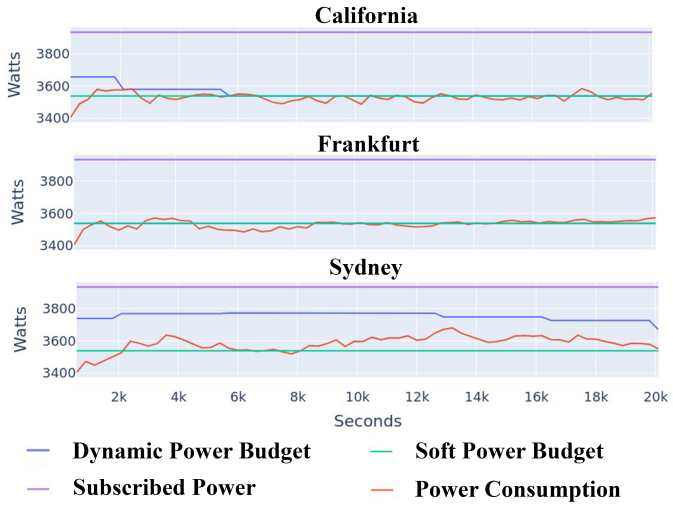


Figure 3.2: Micro-Clouds with dynamic power budgets across different time zones (*Captured against Azure workload traces [2] and solar energy via the PVGIS tool [3]* over 5.5 hours of workload execution).

namic task scheduling in a decentralized manner. Otherwise, substantial round-trip communication times can lead to delayed scheduling decisions. A few studies such as Multi-Criteria Optimal Placement (MCOP) [79] and lightweight service placement heuristic [80] have explored decentralized dynamic task scheduling in Micro-Cloud networks. They focus on multi-criteria optimization, such as node availability and the number of connections. But they do not consider the potential bottlenecks in inter-Micro-Cloud communication. Micro-Clouds are usually interconnected via Wide Area Networks (WAN). WAN can undergo severe traffic congestion, in which its tail latency can be worsened up to 2.5x [81]. Since Micro-Clouds frequently communicate with each other during decentralized task scheduling, dynamic WAN traffic congestion becomes a severe bottleneck to it. Additionally, these studies do not consider the energy optimization of Micro-Clouds. Consequently, we address these bottlenecks taken into account during scheduling decisions.

We propose an approach to realize a *dynamic power budget* in a Micro-Cloud, and a novel task scheduling algorithm called DEMOTS: Decentralized Multi-criteria Optimization Task Scheduling to harness *dynamic power budget* across a network of Micro-Clouds. DEMOTS utilizes a decentralized approach to schedule tasks while tolerating

dynamic WAN traffic congestion. DEMOTS outperforms state-of-the-art scheduling algorithms by up to 19% gain in reducing power overdraw impact, up to 47% gain in reducing task latency increase impact, and up to 49% gain in reducing task schedule time impact, across various tuning levels. In summary, the key contributions of our work are:

- An approach to realize a *dynamic power budget* for Micro-Clouds deployed in Colocation data centers.
- A system model and performance metrics to measure the impact of *power overdraw events* on Micro-Clouds, and the impact of dynamic task scheduling on latency-critical IoT tasks.
- A formal definition of decentralized task scheduling with *dynamic power budgets*, and formulation of the multi-objective problem.
- A novel dynamic decentralized task scheduling algorithm (DEMOTS) to solve the multi-objective problem by utilizing the *dynamic power budget* under realistic WAN traffic congestions.
- Extensive experiments and analysis of results comparing with the state-of-the-art algorithms demonstrating the superiority of DEMOTS.

The rest of the chapter is organized as follows. In Section 3.2 we discuss the related literature. Section 3.3 provides the system model and its essential components. In Section 3.4 we present our proposed DEMOTS algorithm. Section 3.5 describes the performance evaluation and experimental results. Finally, Section 3.6 summarises the chapter.

3.2 Related Work

Most existing approaches for dynamic task scheduling in multi-cloud networks, which are distributed geographically, focus on centralized renewable energy harnessing. Yuan et al. [45] focus on scheduling delay-tolerant applications in a centralized manner, while

Table 3.1: A comparison of related task scheduling algorithms.

Work	Geo-Distributed	Dynamic	Decentralized	WAN Traffic	Task Latency
Tarneberg et al. [73]	✓				
Sharma and Rao [82]	✓	✓			
Zhao et al. [72]	✓	✓			
Yuan et al. [45]	✓	✓			✓
Sajid et al. [48]	✓	✓	✓		
Selimi et al. [80]	✓	✓	✓		
Panadero et al. [79]	✓	✓	✓		
Our Proposed	✓	✓	✓	✓	✓

strictly meeting delay-bounded constraints, and taking into account the spatial and temporal variations of both grid and renewable energy. Sharma and Rao [82] propose a centralized scheduler that aims to optimize the percentage of renewable energy used. They identify a trade-off between average task waiting time and the percentage of renewable energy and suggest a method to improve the renewable energy percentage in geographically distributed multi-clouds. Zhao et al. [72] present a more recent approach that uses centralized deep reinforcement learning to utilize renewable energy in geographically distributed multi-clouds. Similarly, Tarneberg et al. [73] use a dynamic application placement technique to achieve the optimal placement of applications in mobile multi-cloud networks. However, this approach is not purely decentralized, and it is not capable of performing prioritized scheduling of specific criteria such as latency or power.

In contrast with centralized approaches, some recent works have explored decentralized approaches. Sajid et al. [48] use blockchain technology to implement a decentralized scheduling mechanism for energy management across geographically distributed multi-clouds. The use of renewable energy is managed through the blockchain network. Selimi et al. [80] employ a lightweight service placement heuristic that schedules tasks

in community networks with dynamic network bandwidth and node availability. However, it requires the decision of a clustering parameter for optimum performance, which can be difficult in growing Micro-Clouds. To address this issue, Panadero et al. [79] propose the Multi-criteria Optimum Placement (MCOP) algorithm, which automatically handles the clustering issue. However, none of the approaches described above takes into account the impact on latency-critical tasks, as well as the impact of realistic WAN traffic congestion levels during their scheduling decisions.

Table 3.1 summarizes the related work compared to the proposed approach. The majority of the reviewed approaches use centralized scheduling, which becomes a bottleneck for geographically distributed Micro-Cloud networks due to increased communication delays over WAN. Decentralized scheduling approaches provide an advantage, but none of them considers realistic WAN traffic congestion levels, energy optimizations, and the impact of scheduling decisions on the reliability of latency-critical IoT applications. The proposed approach aims to address these identified gaps.

3.3 System Model and Problem Formulation

3.3.1 System Model

The proposed system architecture model, shown in Figure 3.3, is a geographically distributed, decentralized, and homogeneous network of Micro-Clouds. The network latency between Micro-Clouds can change due to WAN traffic congestion. Each Micro-Cloud leases power infrastructure and physical space from a Colocation data center provider. The provider employs the proposed approach to provide a *dynamic power budget* for Micro-Clouds. Users submit latency-critical IoT tasks dynamically to the closest Micro-Cloud. A decentralized task scheduler in each Micro-Cloud dynamically schedules tasks among the Micro-Clouds to avoid *power overdraw events*. The following subsections describe different models and performance metrics we use.

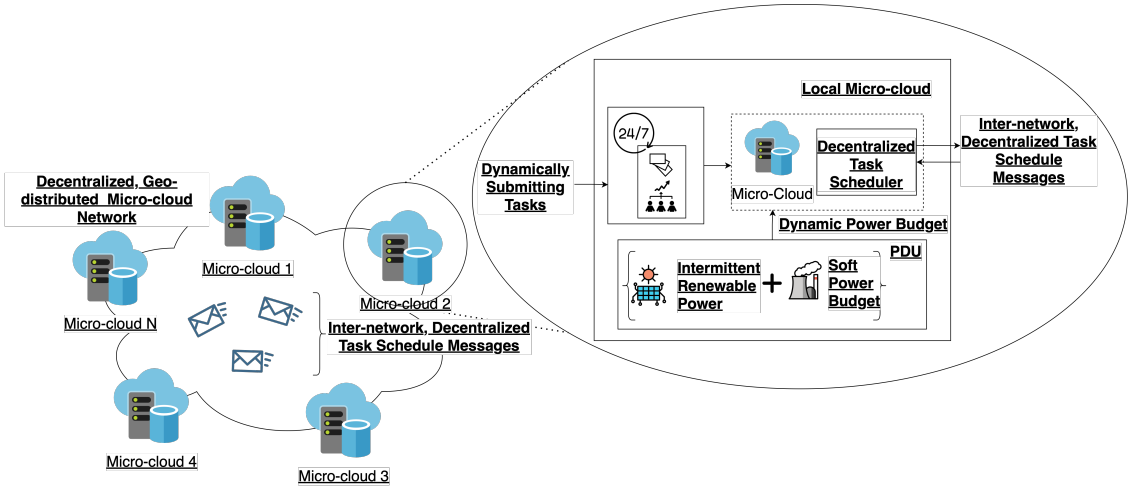


Figure 3.3: System architecture of the geographically distributed network of Micro-Clouds.

Workload Model

We use a location-aware, utilization-based workload model where the workloads are submitted dynamically to Micro-Clouds based on the local time and following a daily trend. The set of tasks that were submitted and executed by all the Micro-Clouds during the time period $t = 0$ to $t = T^{MAX}$ are denoted by Ω,

$$\Omega = \{t_1, ..., t_M\}$$

where M is the total number of tasks.

Power Consumption Model

Micro-Cloud power consumption is primarily determined by the power consumption of computing elements (i.e. power consumption of the Information Technology (IT) systems), and power overhead not used for computing (mostly used for cooling systems) [83]. Using the Power Usage Effectiveness (PUE) [84] metric, a linear relationship can be derived between the IT power and the overhead power [85].

$$P_{overhead} = (\text{PUE} - 1) \times P_{IT} \tag{3.1}$$

where P_{IT} is the IT power, $P_{overhead}$ is the power overhead, and PUE is a constant value. Due to this, we model power consumption just considering P_{IT}. For each Micro-Cloud P_{IT} is introduced as $Pc_i(t)$ (dynamic IT power consumption of i^{th} Micro-Cloud) using a host power consumption model. Our model is based on the CPU utilization level, as this resource represents the main contribution to the host power consumption [86].

$$Pc_i(t) = \sum_{j}^{hosts \in i^{th} \text{Micro-Cloud}} Pw_j(U_j(t)) \tag{3.2}$$

where $Pc_i(t)$ is the power consumption of the i^{th} Micro-Cloud and for the j^{th} host in that Micro-Cloud, the Pw_j is the power consumption model, and $U_j(t)$ is the CPU utilization at t^{th} time.

Dynamic Power Budget Model

To model the *dynamic power budget*, we combine the *soft power budget* (P_{spb_i}) and the intermittent renewable power provided by the Colocation data center provider ($P_{rp_i}(t)$).

$$Pb_i(t) = P_{spb_i} + P_{rp_i}(t) \tag{3.3}$$

where $Pb_i(t)$ is the *dynamic power budget* of the i^{th} Micro-Cloud at the time t.

Power Overdraw Model

We model the power overdraw as the amount of the power consumption exceeding the $Pb_i(t)$ at t^{th} time.

$$P_{od_i}(t) = \begin{cases} Pc_i(t) - Pb_i(t) & \textit{if } Pc_i(t) > Pb_i(t) \\ 0 & \textit{otherwise} \end{cases} \tag{3.4}$$

where $P_{od_i}(t)$ is the power overdraw amount at the time t.

Performance Metrics

The following performance metrics are calculated for all Micro-Clouds in the network during the time period $t = 0$ to $t = T^{MAX}$.

- *Power Overdraw Impact (P_{OI}):* Measures magnitude and the time spent during *power overdraw events*.
$$P_{OI} = \sum_{i=1}^{N} \int_{t=0}^{t=T^{MAX}} P_{od_i}(t)\, dt \tag{3.5}$$

- *Task Schedule Time Impact (T_{SI}):* Measures the amount of time that tasks were blacked out during the dynamic scheduling.
$$T_{SI} = \sum_{t_k \in \Omega} T_{ST_{t_k}} \tag{3.6}$$
where $T_{ST_{t_k}}$ is the time spent by the task t_k during inter-Micro-Clouds scheduling.

- *Task Latency Increase Impact (L_I):* Measures the increased end-user communication latency and the amount of time spent with that.
$$L_I = \sum_{t_k \in \Omega} \sum_{i \in N} L_{t_{ki}} \times T_{t_{ki}} \tag{3.7}$$
where $L_{t_{ki}}$ is the latency increase between the originating Micro-Cloud of the task t_k, and the i^{th} Micro-Cloud. $T_{t_{ki}}$ is the time it spent in the i^{th} Micro-Cloud.

3.3.2 Problem formulation

The overall objective of our problem is to minimize Power Overdraw Impact (P_{OI}), but in doing so, also try to minimize both Task Schedule Time Impact (T_{SI}) and Task Latency Increase Impact (L_I).

3.4 DEMOTS - Decentralized Multi-criteria Optimization Task Scheduling

We present our proposed DEMOTS approach, which is a decentralized algorithm that dynamically schedules tasks and adapts to traffic congestion levels in the inter-Micro-Cloud network. It runs in each Micro-Cloud.

DEMOTS continuously monitor for potential *power overdraw events* of the Micro-Cloud via the Power Overdraw Model ($P_{od_i}(t)$) defined in eq. 3.4. It tries to offload a batch of low-priority tasks to bring down the power consumption of the Micro-Cloud. For each task in the batch, it broadcasts a task offload request across the WAN and waits for responses within a fixed window of time using a time-out. Based on the responses received, the lexicographic method is used to filter and select a destination Micro-Cloud for each task. The selection is based on three optimization criteria designed to minimize Power Overdraw Impact (P_{OI}), Task Schedule Time Impact (T_{SI}), and Task Latency Increase Impact (L_I). Finally, the tasks are offloaded to the destination Micro-Clouds.

Overall, DEMOTS propose three novel optimization criteria to minimize P_{OI}, T_{SI}, L_I, and a novel decentralized approach to offload tasks over WAN with dynamic traffic congestion levels.

3.4.1 Optimization Criteria

The following three optimization criteria are proposed for selecting a destination Micro-Cloud to offload the task t_k.

Minimize P_{OI}

Prioritizing Micro-Clouds with sufficiently available $Pb_i(t)$ would minimize the P_{OI}. We propose the Power Reliability (Pr_i) metric to measure the sufficiently available $Pb_i(t)$.

$$Pr_i(t) = P([Pb_i(t) - Pc_i(t)] > {Pc_i}^{MAX}{}_{t_k}) \tag{3.8}$$

Where ${Pc_i}^{MAX}{}_{t_k}$ is the maximum amount of power that the task t_k would consume at the i^{th} Micro-Cloud. We then estimate $Pc_i(t)$ with its rolling average value, denoted as

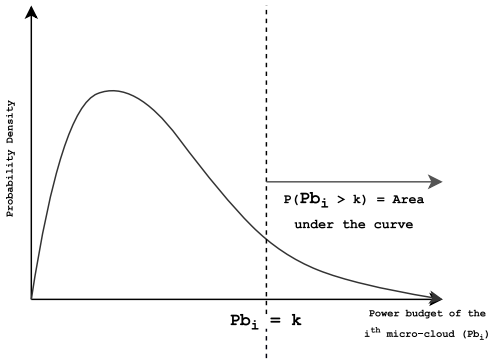


Figure 3.4: Calculating power reliability (Pr_i) of the i^{th} candidate Micro-Cloud.

Pc_{avg_i}.

$$Pr_i(t) = P([Pb_i(t) - Pc_{avg_i}] > {Pc_i}^{MAX}{}_{t_k})$$

$$Pr_i(t) = P(Pb_i(t) > ({Pc_i}^{MAX}{}_{t_k} + Pc_{avg_i}))$$

Define $k = {Pc_i}^{MAX}{}_{t_k} + Pc_{avg_i}$. Then,

$$Pr_i = P(Pb_i(t) > k)$$

where k is a positive constant. This simplifies Pr_i to the cumulative sum of the probability density function (PDF) of the Micro-Cloud's $Pb_i(t)$ as illustrated in Fig. 3.4. We estimate this PDF using historical data for each Micro-Cloud to calculate the Pr_i. Thus, the first optimization criterion is to select the candidate Micro-Cloud maximizing Pr_i.

$$\textbf{Criterion 1} : \arg\max_i(Pr_i) \tag{3.9}$$

Minimize T_{SI}

Prioritizing Micro-Clouds with sufficient processing capability would ensure immediate task deployment, thus optimizing the T_{SI}. We propose the Redundant Processing Capacity metric of the i^{th} candidate Micro-Cloud (R_{pc_i}) to measure sufficient processing

Table 3.2: Description of symbols for DEMOTS task scheduling algorithm.

Symbol	Description
N	Number of Micro-Clouds in the inter Micro-Clouds network.
T^{MAX}	Time duration.
Ω	Set of tasks that were executed during T^{MAX}.
γ	Sensitivity of the dynamic task schedule initiation. A higher value leads to aggressive task offloading.
θ	Batch size of the low-priority tasks. A higher value initiates a larger number of task offload requests.

capability.

$$R_{pc_i} = \frac{C_{PE_i}(t) - T_{PE_{t_k}}}{T_{PE_{t_k}}} \tag{3.10}$$

In which,

- $C_{PE_i}(t) =$ *Available processing elements count in the* i^{th} *candidate Micro-Cloud at time t*
- $T_{PE_{t_k}} =$ *Minimum required processing elements for task* t_k

Thus, the second optimization criterion is to select the candidate Micro-Cloud maximizing R_{pc_i}.

$$\textbf{Criterion 2} : \arg\max_i(R_{pc_i}) \tag{3.11}$$

Minimize L_I

Prioritizing Micro-Clouds having the minimum increase in end-user communication latency would optimize the L_I. Thus, the third optimization criterion is to select the candidate Micro-Cloud minimizing the increase in end-user communication latency ($L_{increase_i}$).

$$\textbf{Criterion 3} : \arg\min_i(L_{increase_i}) \tag{3.12}$$

Algorithm 1 Initiate.

Input: $Pb_i(t), Pc_i(t)$
Output: Task Schedules

1: **while** each-refresh-interval **do**
2: $P_{ratio}(t) \leftarrow \frac{Pb_i(t) - Pc_i(t)}{Pb_i(t)}$
3: **if** $P_{ratio}(t) \leq \gamma$ **then**
4: $T_{lp}(t) \leftarrow getLowPriorityTasks(\theta)$
5: **while** $t_k \leftarrow T_{lp}(t)$ **do**
6: $broadcast(t_k)$
7: $scheduleDispatch(t_k)$

Algorithm 2 Respond.

Input: Rq_{t_k}: *Offload request for* t_k
Output: Response

1: $L_{increase_i} \leftarrow getLatency(Rq_{t_k})$
2: $C_{PE_i} \leftarrow getPCM(Rq_{t_k})$
3: $P_{r_i} \leftarrow getPR(Rq_{t_k})$
4: $replyBack(L_{increase_i}, C_{PE_i}, P_{r_i})$

3.4.2 Decentralized Tasks Offloading over WAN

DEMOTS tasks offloading has three stages; `Initiate`: Broadcasts task offload requests of low-priority tasks, `Respond`: Responds to the task offload requests, and `Dispatch`: Selects the destination Micro-Cloud using three optimization criteria and offloads the task.

- `Initiate`: A Micro-Cloud periodically calculates its available power percentage relative to the Dynamic Power Budget ($Pb_i(t)$). If the available power percentage is below the threshold set by the tuning parameter $\gamma \in \{0, 1\}$, the Micro-Cloud selects a batch of low-priority tasks, where the batch size is set by the tuning parameter $\theta \in \{0, 1\}$. Since the scheduler does not have information about the priority level of the tasks, resource usage is used as an estimator of the priority, where high resource usage means high priority. For each task in the batch, the Micro-Cloud broadcasts a task offload request over the WAN, and a timeout is set to wait for responses. Due to WAN traffic congestion, obtaining responses from all available Micro-Clouds within a reasonable time cannot be guaranteed, thereby the time-

out ensures a reasonable reaction time to *power overdraw events*. Upon meeting the time-out, a `Dispatch` stage is scheduled for each task offload request. This process is outlined in Algorithm 1.

- `Respond`: Upon receiving a task offload request from another Micro-Cloud, a Micro-Cloud responds with the necessary information that is needed to compute three optimization criteria for the offloading task. This process is outlined in Algorithm 2, in which the processing capacity metric $C_{PE_i}(t)$ is calculated by the sub-routine *getPCM*, and Power Reliability (Pr_i) is calculated by the sub-routine *getPR*.

- `Dispatch`: When the time-out is reached for a task offload request, the destination Micro-Cloud is chosen by evaluating three criteria using the lexicographic method. The most significant criterion is Criterion 1, which ensures that Dynamic Power Budget ($Pb_i(t)$) is available. A subset of responses is filtered based on this criterion. Then, this subset is further filtered based on Criteria 2 and 3 to isolate a single response, which is selected as the destination Micro-Cloud. The task is then offloaded to this destination Micro-Cloud.

3.5 Performance Evaluation

In this section, we describe our experimental setup and demonstrate the effectiveness of DEMOTS algorithm by analyzing the results and comparing them with baselines.

3.5.1 Experimental Setup

We evaluate our proposed solution in a simulated environment using the CloudSim toolkit [87]. We use real-world workload traces for generating application workload. We also configure other relevant parameters from real-world traces and models such as network latency data, power models, and solar energy traces.

We extend the Datacenter class of CloudSim toolkit to implement a power budget-aware Micro-Cloud component and further extend the Host class to implement power-

aware hosts. Each host utilizes an extended PowerModel class to model the physical server that we use. We implement new Power Source classes and embed them into the extended Datacenter class to manage renewable energy through solar energy traces. The utilization of each VM and its associated workload is implemented using the extended Vm and Cloudlet classes of the CloudSim toolkit. To manage resource utilization data in the workload trace, we employ an extended Cloudlet Scheduler class. The inter-Micro-Cloud network is managed using the default implementation of the Network Topology class in the CloudSim toolkit, and network latency data is handled accordingly.

Micro-Cloud Design: We design a Micro-Cloud as a tenant residing in a Colocation data center which occupies a full dedicated server rack [74]. We select the rack size as 42U, based on modern mixed-energy Micro-Clouds designs [88]. We consider a rack that has 21 Fujitsu RX300 S6 XeonE5620 servers, each occupying 2U. We use a power model developed specifically for the selected server type [89]. The peak power consumption of this server rack is 3.9 kW.

***Dynamic Power Budget* Implementation:** We implement our proposed *dynamic power budget* approach for the Micro-Cloud. We set the *soft power budget* imposed by the data center operator at 90% of the Micro-Cloud's subscribed power (which is 3.9 kW, the peak power consumption in our Micro-Cloud design). Therefore, we provision a solar panel providing 265 W peak power [88], to accommodate the remaining 10% of that. The solar energy is simulated using the real traces obtained from the European Commission's Photovoltaic Geographic Information System (PVGIS) [3], per each geographical location.

Dynamic Workload Submission: We utilize the Microsoft Azure public VM workload traces from the year 2019 [2]. It represents one of the most recent publicly available production VM workload traces [90], and exhibits a dynamic workload submission trend. We shift that trend based on the local time zone, such that the workload submission trend is location-aware.

Experiment Scenario: We employ a geographically distributed Micro-Clouds network. Each of the Micro-Cloud in the network is deployed in a Colocation data center as a tenant. The Colocation data centers are located in dispersed geographical locations, such that they are in different time zones from each other. The geographical locations

are configured based on AWS data center regions. While AWS regions are hyper-scale clouds, we assume our Micro-Clouds exist in similar locations inside colocated facilities to reflect the realistic scenarios, such as the presence of Micro-Clouds across time zones and geographically closer to application users. We use nine different AWS regions connected over a WAN, and the real average inter-region communication latency values [91]. In order to simulate the effects of WAN traffic congestion, we scale WAN communication latency from its average values, up to the worst-case upper bound, which is 2.5x [81].

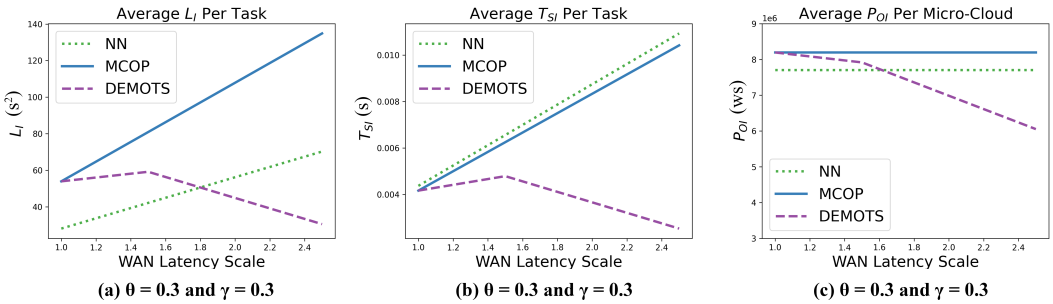
3.5.2 Baseline Algorithms

We consider approaches suitable for Micro-Cloud networks and avoid comparisons with centralized scheduling approaches that can lead to significantly delayed scheduling decisions due to WAN traffic congestion. Therefore, we compare our DEMOTS approach with the following two decentralized scheduling methods.

- **Nearest Neighbour (NN)**: A heuristic that schedules tasks to the nearest available Micro-Cloud, in terms of the latency [92].
- **MCOP**: A dynamic decentralized task scheduling algorithm for Micro-Cloud networks [79]. MCOP is the state-of-the-art decentralized task scheduling algorithm for Micro-Clouds that provides faster execution based on its lightweight heuristic approach.

3.5.3 Results and Analysis

We carried out 24-hour-long experiments for different parameter configurations. The γ value determines between a reactive ($\gamma = 0$) and a proactive ($\gamma > 0$) approach towards managing *power overdraw events*. Our aim is to avoid these events entirely, thus we set γ at 0.3 (i.e., 30% of the available power triggers task offloading). The θ controls the batch size of tasks offloading. We observed scheduler sensitivity towards θ across a range of values ($\theta \in \{0.3, 0.4, 0.5, 0.6\}$). Each scenario was executed across worsening WAN traffic congestion by up-scaling communication latency. In all configurations, we set



(a) θ = 0.3 and γ = 0.3 **(b) θ = 0.3 and γ = 0.3** **(c) θ = 0.3 and γ = 0.3**

Figure 3.5: Performance metrics comparison of the task scheduling among Micro-Clouds for $\theta = 0.3$.

DEMOTS timeout, such that when the WAN latency values are at the average, DEMOTS is able to receive responses from all the Micro-Clouds in the network before offloading a task.

Fig. 3.5 shows scheduler performances for $\theta = 0.3$ in Task Latency Increase Impact (L_I), Task Schedule Time Impact (T_{SI}) and Power Overdraw Impact (P_{OI}). As WAN latency increases, both MCOP and NN show linear trends for L_I and T_{SI} (Fig. 3.5-a and Fig. 3.5-b), and a constant trend in P_{OI} (Fig. 3.5-c). The reason behind both these trends is that MCOP and NN are not aware of the changes in WAN latency, thus performing the same scheduling decisions. Based on Equations 3.6 and 3.7, if the scheduling decisions remain the same, the L_I and T_{SI} change linearly with the increasing WAN latency, whereas based on Equation 3.5 the P_{OI} stays the same. In contrast, DEMOTS reacts to changing WAN latency and outperforms DEMOTS across all three metrics (Fig. 3.5-a, Fig. 3.5-b, and Fig. 3.5-c). This is justified as DEMOTS uses a timeout-based waiting approach, in which the worsening WAN latency forces it to change its scheduling decisions. DEMOTS achieve the same performance as MCOP when WAN latency is at its average. However, as WAN latency increases, DEMOTS take a different scheduling approach from MCOP, which converges in better performances.

Fig. 3.6 shows the comparison of the total number of task reschedules by schedulers, which is preferred to be minimized for latency-critical IoT tasks. Because for such tasks, reliability is critical, and an increased number of reschedules reduces task reliability. In that regard, NN shows significantly worst task reliability. NN's lowest number of task reschedules is achieved at $\theta = 0.3$, in which both MCOP and DEMOTS show

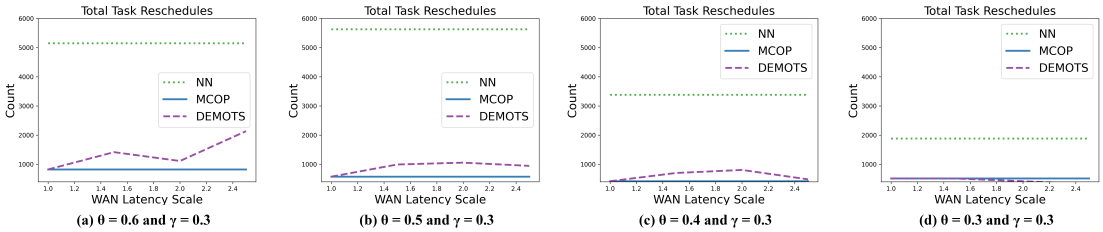


Figure 3.6: Total number of task reschedules of the task scheduling among Micro-Clouds against θ.

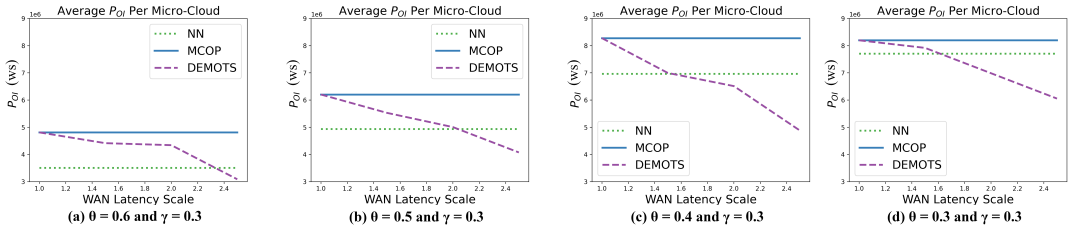


Figure 3.7: Power overdrawn impact performance of the task scheduling among Micro-Clouds against θ.

significantly lower numbers of task rescheduled (Fig. 3.6-d). This trend continues for increasing θ (Fig. 3.6-c to Fig. 3.6-a). In comparison, DEMOTS and MCOP seem to be in a similar range, with DEMOTS having a slight increase over MCOP. The θ determines the task batch size, thus overall, decreasing θ results in a reduced number of task reschedules across all schedulers, as shown from Fig. 3.6-a to Fig. 3.6-d.

Fig. 3.7 shows scheduler sensitivity towards θ in optimizing Power Overdraw Impact (P_{OI}). In general, decreasing θ results in lowering P_{OI} performance (i.e., higher values for P_{OI}) for all schedulers as seen from Fig. 3.7-a to Fig. 3.7-d. NN has the worst sensitivity, with P_{OI} having a comparatively fast growth rate. In contrast, both MCOP and DEMOTS show better sensitivity. Moreover, DEMOTS outperform other schedulers by converging to better P_{OI} across the WAN latency scale, despite the decreasing θ.

Table 3.3: Performance comparison of schedulers.

θ	**Scheduling Algorithm**	**Mean Value across the WAN Latency Scale (1-2.5x)**							
		Avg. P_{OI} per Micro -Cloud ($\times 10^6$ws)	**Gain over MCOP**	**Avg. L_I per task (s * s)**	**Gain over MCOP**	**Avg. T_{SI} per task ($\times 10^{-3}$s)**	**Gain over MCOP**	**Total Task Re-schedules**	**Gain over MCOP**
0.3	NN	7.70	6.0%	51.52	47.91% /	8.0	-4.92%	1883	-267.77%
	MCOP	8.19	-	98.89	-	7.6	-	512	-
	DEMOTS	7.38	9.85%	47.88	51.58%	3.8	49.86%	448	12.5%
0.4	NN	6.96	15.85%	34.75	37.41%	9.6	-123.35%	3384	-711.51%
	MCOP	8.27	-	55.52	-	4.3	-	417	-
	DEMOTS	6.66	19.46%	37.50	32.45%	4.0	5.41%	600	-44.06%
0.5	NN	4.93	20.46%	37.78	40.80%	1.22	-101.75%	5629	-882.37%
	MCOP	6.20	-	63.83	-	6.06	-	573	-
	DEMOTS	5.20	16.12%	38.53	39.63%	5.4	9.70%	890	-55.41%
0.6	NN	3.50	27.06%	34.42	33.84%	1.0	-56.18%	5153	-528.41%
	MCOP	4.80	-	52.04	-	6.0	-	820	-
	DEMOTS	4.16	13.45%	36.86	29.15%	6.0	3.96%	1369	-67.0%

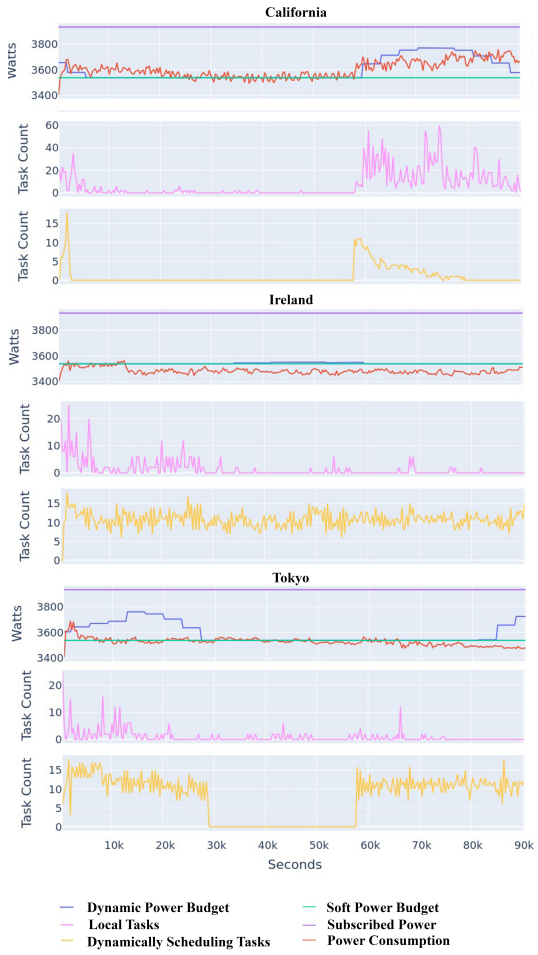


Figure 3.8: DEMOTS: harnessing dynamic power budget (*Collected data from 24 Hours of workload execution*).

To summarize, a comparison of average performance metrics over the WAN latency scale is depicted in Table 3.3. While having better Power Overdraw Impact (P_{OI}) and Task Latency Increase Impact (L_I) gains over MCOP (6% to 27%, and 33% to 47% respectively), NN lags behind in Task Schedule Time Impact (T_{SI}) (-4% to -123%). Most importantly, NN needs to perform a higher number of task reschedules over MCOP (-267% to -882%), which significantly decreases task reliability. In contrast, DEMOTS performs much better in the number of tasks reschedules over MCOP (-67% to 12.5%). Therefore, both MCOP and DEMOTS surpass NN in scheduling latency-critical IoT tasks demanding the highest level of reliability. Moreover, DEMOTS outperforms MCOP by 36% to

47% in P_{OI}, 9% to 19% in L_I, and 3% to 49% in T_{SI}. Therefore, in the overall scheduling problem, DEMOTS is able to harness *dynamic power budget* to reduce *power overdraw events* by dynamically scheduling latency-critical IoT tasks over a WAN with dynamic traffic congestion.

Fig. 3.8 showcase DEMOTS harnessing *dynamic power budget* across different time zones. In which, we observe the collective decentralized task offloading of DEMOTS successfully shares the workload based on available power. Concretely, the Micro-Cloud in California handles workloads near its power capacity (i.e., power consumption is around the *soft power budget*), until the observing time reaches 60k (seconds). Throughout this period of time, DEMOTS do not offload tasks to California. Afterwards, the zone receives excess *dynamic power budget*, in which California starts receiving tasks from other Micro-Clouds executing DEMOTS to utilize the excess power. The same behaviour can be seen in parallel for both Ireland and Tokyo.

3.6 Summary

In this chapter, we explored latency-critical Internet of Things (IoT) applications deployed on distributed Micro-Clouds that are leasing resources from power-constrained Colocation data centers. To scale its capacity amidst power constraints, Colocation data center providers impose a *soft power budgets* on its leasing Micro-Clouds. In return, Micro-Clouds are forced to manage potential *power overdraw events* with extreme power overdraw recovery techniques, increasing the probability of compromising the latency performance of the IoT tasks. We proposed an approach to advance the *soft power budget* to a *dynamic power budget*, leveraging the provisioning flexibility of renewables to increase the data center energy capacity via on-site renewable energy sources. To effectively harness *dynamic power budgets* across geographically distributed Micro-Clouds connected over a Wide Area Network (WAN), we proposed DEMOTS: A dynamic decentralized task scheduling algorithm. DEMOTS can jointly optimize task latency performance and *dynamic power budgets* over unstable network overheads of WAN traffic congestion. Our extensive simulation-based evaluation experiments showed that the proposed DEMOTS outperformed the state-of-the-art decentralized scheduling algo-

rithm by up to 19% reduction in Power Overdraw Impact, up to 47% reduction in Task Latency Increase Impact, and up to 49% reduction in Task Schedule Time Impact.

This chapter presented a dynamic decentralized task scheduling algorithm that exploits the provisioning flexibility of renewable energy sources to reduce power overdraw events in power-constrained Micro-Clouds executing latency-critical IoT applications. In the next chapter, we study leveraging renewable energy to reduce the operational carbon footprint of cloud computing environments facilitating bounded-latency applications, in particular, resource management for the operational carbon efficiency of real-time cloud applications.

Chapter 4

Carbon Optimization for Real-Time Cloud Systems using Renewables-driven Cores

Operational carbon optimization in cloud platforms is often limited to temporally flexible workloads that can be executed when and where renewable energy is available. In this chapter, we focus on temporally inflexible real-time workloads instead and present a framework to utilize renewable energy in real-time cloud systems using renewables-driven cores. Our framework proposes a VM Execution Model to ensure workload VMs are intact from core availability dynamics. Furthermore, using renewables-driven cores, it introduces the Green Cores concept to convert the utilization of renewables as a server packing attribute and derive a novel VM Packing Algorithm to efficiently harvest renewable energy across servers. We practically evaluate our framework by implementing it with OpenStack. To conduct long-running experiments, we use a large-scale simulator using real-world trace data. Our results demonstrate that the proposed framework outperforms state-of-the-art baselines by a 6.52× reduction in real-time latency variance and a joint 79.64% *increase in renewable energy harvest with a* 34.83% *reduction in VM interruptions.*

4.1 Introduction

Due to the environmental concerns of Greenhouse gas (GHG) emissions, electrical grids continue to integrate low-emission renewable energy sources. In 2022, the share of renewables in total electricity generation was 39% and is projected to be 91% by 2035 [93].

This chapter is derived from:

- **Tharindu B. Hewage**, Shashikant Ilager, Maria A. Rodriguez, and Rajkumar Buyya, "A Framework for Carbon-aware Real-Time Workload Management in Clouds using Renewables-driven Cores", *IEEE Transactions on Computers*, early access, May 2025.

However, a significant portion of renewable energy sources, such as solar and wind, exhibit variable-availability (intermittent) supply dynamics [15]. Between 2022 and 2035, energy reports project the share of solar and wind renewables in total generation to rise from 12% to 58% [93].

Data centers develop various *load matching* strategies to match workload execution over renewable energy supply dynamics. Amongst them, *load shifting* is commonly practised [94–97]. *Load shifting* uses workloads with temporal flexibility to suspend/resume their execution. For example, Google's delay-tolerant workloads, such as machine learning, data compaction, and data processing, tolerate delays as long as their work gets completed within 24 hours [96]. Workloads execute in periods when renewable energy capacity is higher, resulting in reduced GHG emissions. However, *load shifting* falls short when applied to real-time workloads with strict response time boundaries [98]. Real-time workloads cannot tolerate the delays inherent in *load shifting*.

Nevertheless, the growing prevalence of real-time cloud applications, such as autonomous vehicles, industrial automation [13], and railway control systems [17] expects to account for nearly 30% of the world data by 2025 [99]. As a result, cloud operators will eventually have to incorporate growing real-time workloads in intermittent renewable energy integration. In this context, one must find an alternative load matching strategy to *load shifting* for delay-intolerant real-time workloads. Existing solutions, such as applying CPU-wide low power profile to match renewable energy supply [100], often result in increased latency, making them unsuitable for real-time applications. Moreover, techniques like Harvest Virtual Machines (HVMs), which allow uninterrupted execution of workloads with reduced resources [6], can still degrade performance and fail to meet real-time constraints.

Given these challenges, there is a need for an efficient strategy to integrate renewable energy into real-time cloud systems (Real-Time Clouds). To this end, we propose a framework to harvest renewable energy in Real-Time Clouds. We use Renewables-driven cores to integrate renewable energy for servers. It dynamically switches the power profiles of each CPU core between a real-time power profile and a low power profile to match renewable energy intermittency. Then, our framework applies a twofold solution to utilize this dynamic core availability. First, we develop a **VM Execution**

Model to guarantee that real-time virtual machines (VMs) occupy cores at the real-time power profile. Our model adopts renewable energy fluctuations by conducting criticality-aware VM evictions as needed. Secondly, we develop a **VM Packing Algorithm** to optimize the use of available cores across servers. It reduces the likelihood of VM evictions while maximizing renewable energy utilization (renewable energy harvest). Our algorithm frames renewable energy management as a VM placement optimization problem by introducing the concept of **Green Cores**. Green Cores presents each server as an inventory of two virtual CPU core types: Green and Regular. Green cores quantify renewable energy usage, whereas Regular cores quantify core usage that does not increase risks of VM eviction incidents. Using Green Cores, we achieve a computationally inexpensive VM packing algorithm, which is required to handle VM throughput at scale [96].

We implement our framework in OpenStack [33] as openstack-gc. We combine OpenStacks control plane with an on-node daemon service. The daemon service implements Renewables-driven cores in the server using per-core sleep states. openstack-gc control plane then communicates with the daemon service to orchestrate our VM Execution Model and VM Packing Algorithm. We evaluate our framework at the core-level using VMs running RTEval, a program from the Real-Time Linux project to measure real-time performance [7]. We evaluate our framework at the server-level using a 14-day VM arrival trace from Azure [8]. We use two testbeds: an experimental openstack-gc cloud deployed on an HPE ProLiant server with a 12-core Intel Xeon CPU, and a large-scale simulation testbed. We make the following contributions in designing, implementing, and evaluating our framework.

- We propose a core-level **VM Execution Model** to utilize renewable energy without degrading real-time latency performance in VMs. We leverage criticality-aware VM evictions for that.
- We propose a server-level **VM Packing Algorithm** to reduce VM eviction incidents over renewable energy utilization.
- We implement a prototype of our framework in OpenStack, detailing its design and demonstrating its practicality.

- We evaluate our approach against multiple baselines. Our results show: i) 6.52× reduction in coefficient of variation of real-time latency in VMs over the existing workload temporal flexibility-based VM execution model, and ii) a joint optimization of 79.64% reduction in VM eviction incidents and 34.83% increase in utilized renewable energy over state-of-the-art packing algorithms [101].

The rest of the chapter is organized as follows: Section 4.2 discusses related work. Section 4.3 provides the background and motivation for our problem with a use case study. Section 4.4 details our system model and problem formulation. Section 4.6 outlines the design of our proposed framework. Section 4.7 describes the implementation of openstack-gc. Section 4.8 presents the performance evaluation of our framework, and finally, Section 4.9 summarises the chapter.

4.2 Related Work

Load matching with renewables-driven cores: Common load matching techniques for intermittent renewable energy, such as geographical load balancing, workload migration, admission control, and capacity planning [95–97], depend on either suspending/resuming or migrating flexible workloads. In contrast, Renewables-driven cores avoids both by performing load matching with dynamic core availability. SolarCore [102] and Chameleon [53] use per-core Dynamic Voltage and Frequency Scaling (DVFS) and Power Gating to implement Renewables-driven cores. In their work, workloads utilizing power-adjusted cores can undergo performance degradation, thus better suited for throughput workloads with flexible deadlines. PowerMorph [54] improves this via core grouping, hosting critical and best-effort workloads and power adjustments isolated to core groups. However, workload core affinity can dynamically change during load matching, unfavourable for time-critical workloads such as real-time compute [29]. Slackshed [6] implement Renewables-driven cores for virtual machine (VM) execution. They achieve uninterrupted VM execution at the expense of dynamic CPU allocation, thus better suited for throughput workloads with flexible time constraints.

In contrast, our work preserves workload time boundaries over Renewables-driven cores and leverages criticality-aware VM evictions within safe limits of the application

Table 4.1: Comparison of relevant work with our proposed framework for carbon-optimization in real-time clouds.

Work	Rnw-driven Cores	Critical Workloads	Real-Time	VM Mgt.	Criticality-aware Packing	Rnw. Harvest
SolarCore (2011) [102]	✓					✓
Chameleon (2013) [53]	✓					✓
Kumbhare et. al (2021) [101]		✓		✓	✓	
PowerMorph (2022) [54]	✓	✓				✓
Slackshed (2023) [6]	✓			✓		✓
Our Proposed	✓	✓	✓	✓	✓	✓

layer.

VM packing algorithms: VM packing is a widely studied research problem. Most existing works focus on variants of bin packing algorithms to improve resource utilization at scale [103, 104], yet consider servers as static inventories. Opposed to that, Kumbhare et al. [101] explore an inventory where servers oversubscribe power delivery, yielding a dynamically changing inventory capacity. They propose a criticality-aware packing algorithm to co-pack critical and best-effort components, reducing workload impact. However, in doing so, they do not consider renewable energy harvesting opportunities.

In contrast, our work achieves joint optimization of workload impact and harvesting in dynamic inventories.

4.3 Background, Motivation, and Use case

This section provides background on real-time workloads in clouds (Real-Time Clouds) and the application of Renewables-driven cores. It then motivates our contributions

with a use case study. Finally, it outlines the key takeaways.

4.3.1 Real-Time Clouds

Real-Time Clouds deploy cloud-based real-time applications, such as industry 4.0 use cases [105], transport use cases [17], and software-defined networks [4]. A key requirement in real-time computing is to produce computation results in a bounded time [105]. Clouds achieve that by tuning the entire virtualization stack to reduce latency in executing application instructions [17, 29, 106, 107], such as setting each CPU core to a consistent high-performance power profile and pinning each virtual machine (VM) core to a dedicated physical core, resulting rigid VM placement constraints while delivering deterministic performance. Further, real-time cloud systems employ an application-specific middleware layer over VMs to provide fault-tolerance in VM failures [4, 17].

4.3.2 Renewables-driven Cores

Renewables-driven cores is a load-matching technique that dynamically adjusts per-core power draw in CPU to match server load for renewable energy dynamics [53, 102]. Unlike the CPU-wide throttling techniques [101], it narrows power optimization to the core level. However, such per-core power dynamics must be efficiently utilized, adhering to application performance requirements using a suitable workload execution model. Existing works that utilize Renewables-driven cores in clouds use Harvest Virtual Machine (HVM) as the workload execution model [108], which dynamically shares available CPU cores among the VM cores.

4.3.3 Motivation

We outline the motivation behind our proposed framework, specifically focusing on the rationale for selecting Renewables-driven cores as the load-matching technique for Real-Time Clouds. Then, we discuss the lack of static compute allocation in the existing VM execution solution for Renewables-driven cores and how it can impact real-time VMs. Following this, we present our approach to addressing this limitation by exploiting the

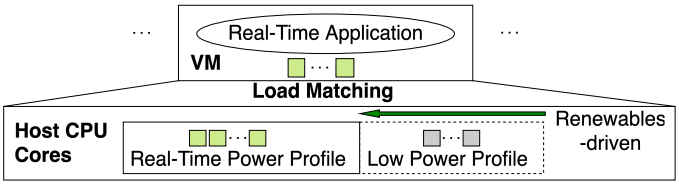


Figure 4.1: Renewables-driven cores in Real-Time Clouds.

presence of mixed-criticality in Real-Time Clouds, composed of VMs hosting critical components and best-effort components, to implement criticality-aware VM evictions.

How do Renewables-driven cores align with the needs of Real-Time Clouds? Renewable energy integration using Renewables-driven cores as the load-matching technique enables avoiding both suspend/resume and CPU-wide throttling of workloads. Figure 4.1 illustrates a scenario where a set of cores reside in the low power profile. However, the remaining cores reside in the real-time power profile. They provide the opportunity to serve real-time VMs amidst renewable energy fluctuations.

Why is a static compute allocation important to real-time VMs?: To apply Renewables-driven cores in Real-Time Clouds, we need a workload execution model to match dynamic core availability for VMs. In this regard, the existing solution is Harvest VMs (HVMs) [6]. HVMs approach is to change the number of physical cores (pCPUs) in the VM but preserve the number of virtual cores (vCPUs). It allows continued execution of the VM amidst dynamic core availability. However, for real-time VMs, this dynamic compute allocation can introduce performance degradation. For instance, if the number of pCPUs is less than that of vCPUs, vCPUs oversubscribe physical cores, leading to scheduling delays, which must be avoided with real-time compute [29]. Therefore, insufficient pCPU allocations can lead to undesirable real-time latency spikes in the VMs. Maintaining a static compute allocation is important to avoid such scenarios. In Section 4.8.2, we practically show the VM's real-time performance degradation when the number of pCPUs falls below vCPUs.

Opportunities in mixed-criticality within Real-Time Clouds to provide static compute allocation for VMs over Renewables-driven cores: An alternative to the HVM approach of continuing the execution of VMs with insufficient pCPUs is to evict the VMs. Existing works show opportunities for this in Real-Time Clouds via the mixed-criticality

Table 4.2: Mixed-criticality use cases in 5G network slicing. [13].

Scenario	Reliability	Criticality
Autonomous Driving	99.999%	Critical
Industrial Machinery	99.999%	Critical
4K/8K HD Video	-	Best-effort
Mass Gathering	-	Best-effort

of real-time systems [4, 17, 109]. Firstly, they model real-time system components as either critical or best-effort. Then, an application-specific middleware layer exploits this mixed-criticality to provide fault tolerance for component failures. In real-time cloud systems, application-specific middleware layers use reconfiguration policies to recover the system upon VM failures [4]. In this context, there is an opportunity to conduct VM evictions in Real-Time Clouds safely. Since a VM eviction is a well-defined failure event, the application-specific middleware layer can tolerate it through reconfiguration. More importantly, we can guarantee a static compute allocation for VMs with a fixed allocation of CPU cores and conduct criticality-aware VM eviction if cores are insufficient instead of continued VM execution with degraded performance.

4.3.4 Usecase

To further motivate our approach, we experiment with a real-time cloud use case of 5G Network Slicing via Virtual Network Functions (VNF) [13]. As the application-specific middleware layer, we employ the production-grade VNF management and orchestration middleware, OSM MANO [4]. We map VNFs to critical and best-effort components based on the service quality level of their network slice. Table 4.2 denotes an example where the criticality of four different 5G scenarios is interpreted based on service reliability. Figure 4.2 illustrates our study. We connect the MANO deployment with a real-time tuned two-node OpenStack deployment as the real-time cloud. We use the auto-heal feature of MANO as the reconfiguration policy [4]. Each VNF is deployed as a VM in OpenStack with two virtual CPU cores. We use 50% Renewables-driven cores in servers at 100% initial renewable energy capacity. Once the deployment stabilizes, we drop that

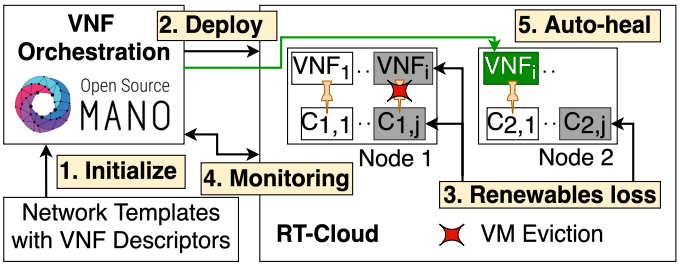


Figure 4.2: Use case: load matching with evictions via application-level reconfiguration of NFV Management Middleware MANO's auto-healing over VM failures [4].

Table 4.3: Comparison of the VM packing inventory when load matching with the 5G network slicing prototype over different packing strategies.

Packing	Before		After		Evictions
	Node 1	Node 2	Node 1	Node 2	
Tightly	0/8	8/8	4/4	4/4	2
Spread	4/8	4/8	4/4	4/4	0

to 0%, reducing server core count by half and evicting VMs to load match. We repeat the experiment for two VM scheduling approaches in OpenStack.

Table 4.3 denotes our observations. Firstly, upon the loss of renewable energy capacity, MANO reconfigured the available cores through auto-healing. Secondly, OpenStack initiates VM evictions. Thus, knowledge of VM criticality is beneficial in reducing the impact of eviction. For example, it permits evicting best-effort components prior to critical components. Thirdly, the VM packing strategy can change the number of VM evictions. Of the two packing approaches used in our use case, the Tightly approach triggered two eviction incidents, whereas the spreading approach yielded none.

4.3.5 Key Takeaways

From our motivations and the use case experiment, we identify the following key takeaways:

1. Renewables-driven cores enable integrating renewable energy in Real-Time Clouds.

In that,

(a) A static compute allocation for VMs ensures their real-time performance.

(b) Criticality-aware VM evictions enable maintaining static compute allocations over Renewables-driven cores.

2. The server-level VM packing strategy can influence the likelihood of VM eviction incidents.

Motivated by the above, we design our framework to apply Renewables-driven cores in Real-Time Clouds. It addresses point 1 using a VM Execution Model and point 2 using a VM Packing Algorithm. It advances cloud renewable energy integration by providing deterministic computing amidst the supply intermittency of renewable energy. It enables carbon-efficient computing with time-critical real-time cloud workloads, which is otherwise considered inflexible for carbon optimization [96].

4.4 System Model and Problem Formulation

4.4.1 System Model

Our system model is shown in Figure 4.3. In that, each server receives dedicated allocations of grid and renewable energy capacities through a mixed power delivery system. Allocations are even across all servers. We use homogeneous servers for simplicity, but the model can be adapted for heterogeneous servers by allocating power capacities proportionately. Each server monitors the dynamic availability of its allocated renewable capacity for *load matching*. We model renewable energy as intermittent and carbon-free and grid energy as static and carbon-intensive. An application-specific middleware layer manages each VM. It can tolerate VM eviction incidents. At arrival, VMs provide their criticality to the cloud control plane as either critical or best-effort. A VM packing algorithm then places VMs in the server inventory.

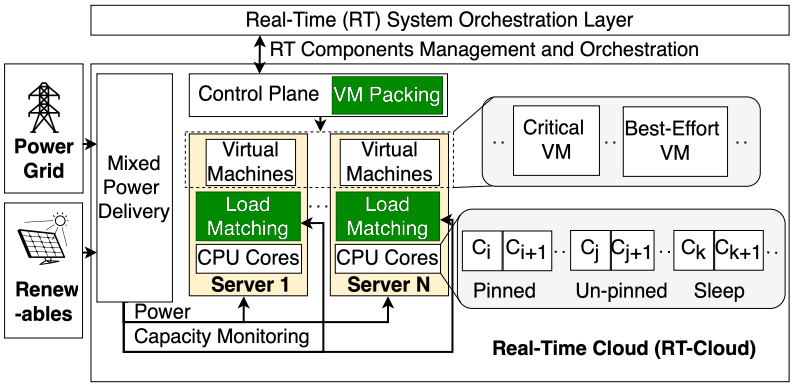


Figure 4.3: A high-level system model of the proposed carbon-aware real-time cloud with contributions highlighted in green.

4.4.2 Problem Formulation

Server power modeling: Data center power modeling outlines key load elements, such as information and technology (IT), cooling, and internal power conditioning system [110]. IT load corresponds to server power consumption of active VM execution and idle power draw, which can be modeled encompassing the power consumption of server components such as CPU, Memory, GPU and HDD [111, 112]. Our focus in this chapter is integrating intermittent renewable energy with the IT load for CPU-dominant real-time workloads [17, 29]. In that regard, server power can be estimated using a linear function (f) of CPU power ($P_{CPU}(t)$) [112] with over 90% accuracy. Based on that, we derive a CPU utilization-aware linear multi-piece server power model for the server power usage of the power distribution unit (PDU). First, we state server power at time t ($P_S(t)$) as,

$$P_S(t) = f(P_{CPU}(t))$$

In multi-core CPUs, the cumulative sum of core power becomes a close upper bound of $P_{CPU}(t)$ [113]. Based on that, we derive an upper-bound to server power and use that as an estimate to $P_{CPU}(t)$. Therefore, for a server with N number of cores,

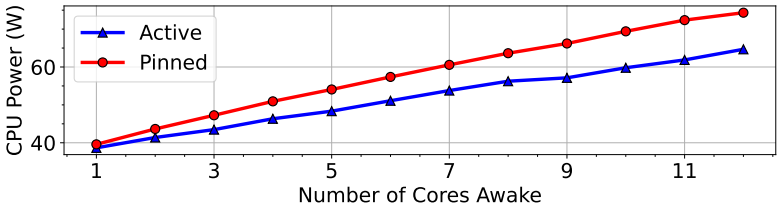


Figure 4.4: CPU power as cores awake in Renewables-driven cores of Intel Xeon Silver CPU with core power states: i) *Sleep* ≡ sleep state of C6, ii) *Active* ≡ sleep state of *POLL*, and iii) *Pinned* ≡ pinned with 100% utilization.

.

$$P_S(t) \simeq f(\sum_{i=1}^{N} P_{CORE_i}(t)) \tag{4.1}$$

where $P_{CORE_i}(t)$ is the power consumption of i^{th} core at time t. Commodity servers often consist of homogeneous cores. Thus, we apply the same model here. Next, we model power states for the three distinct states of $P_{CORE_i}(t)$. When a core is unused, its power state is either,

- *Active* ≡ $P_{CORE_i}(t) = P_{ACT}$ (an idle core)
- *Sleep* ≡ $P_{CORE_i}(t) = P_{SLP}$ (a core in the low power profile)

In contrast, a core pinned to a VM exhibits a power state of $P_{CORE_i}(t) = F(U_{CORE_i}(t))$. Where $U_{CORE_i}(t)$ is the utilization of the core at time t, and F is a linear function [113]. Dynamics of $U_{CORE_i}(t)$ depends on the VM workload, which is a black box to the cloud operator [101]. Therefore, in packing problems, a representative utilization statistic is commonly estimated based on historical data [101, 112]. Based on this, we use U_{RT} to estimate $U_{CORE_i}(t)$. Cloud operator sets the exact U_{RT} value using deployment-specific data. As a result, the pinned power state of a core becomes,
Pinned ≡ $P_{CORE_i}(t) = P_{PIN}$, where $P_{PIN} = F(U_{RT})$

We verify our core power model with an Intel Xeon Silver CPU with 12 cores. For that, we use $U_{RT} = 100\%$. We wake up cores from 1 to 12 and plot CPU package power obtained through Intels Running Average Power Limit (RAPL) interface. We conduct the same experiment for both *Active* and *Pinned* scenarios. Figure 4.4 illustrates our results. The constant slopes of the linear graphs verify P_{PIN} and P_{ACT}. The remaining

power state is P_{SLP} and it contributes to both the graphs, thus the same linear trends also verify P_{SLP}.

We then apply the core power model in Equation 4.1 and derive the following model to estimate server power using core counts as variables.

$$\begin{aligned} P_S(t) \simeq f(m(t) \times P_{PIN} + l(t) \times P_{SLP} \\ +(N - m(t) - l(t)) \times P_{ACT}) \end{aligned} \tag{4.2}$$

where at time t, $m(t)$ is the pinned core count and $l(t)$ is the sleeping core count.

Renewable energy harvest: Electricity generated from replenishing renewable energy sources (i.e., renewables) emits significantly lower amounts of carbon when compared to grid energy, which often relies on fossil fuel-based energy generation. However, most renewable energy sources rely on intermittent natural resources that vary depending on the time of the day and geographical location, such as solar and wind [15]. In return, renewables yield intermittent power capacities compared to stable grid power.

In our model, renewable energy harvesting denotes maximizing the utilization of the intermittent power capacity of renewables. We use a heterogeneous server power allocation of dedicated renewable and grid energy capacities. In doing so, each server is guaranteed a specific amount of stable power capacity, allowing the resource management layer to utilize that in maintaining the stringiest service level objectives (SLOs) of real-time VMs. The portion of the renewable energy capacity is set by the data center operator, depending on the fault tolerance levels of the data center power delivery. In Section 4.8.1, we provide an example of deciding that with our large-scale testbed design. The dynamics of the renewable capacity availability depends on the volume pattern of the renewable energy source. Figure 4.13a illustrates an example of that with our 14-day VM packing experiments. We assume renewable energy does not incur additional costs besides supply dynamics.

As a result of the heterogeneous server power allocation, harvesting renewables in our system model must be done at the server level. When server power meets the grid capacity (P_{GRID}), we denote $P_S(t) = P_{GRID}$. Renewable energy harvesting begins when $P_S(t) > P_{GRID}$. Therefore, for an arbitrary time period ΔT, we denote harvested renew-

able energy ($E_{RW}(\Delta T)$) of the server as,

$$E_{RW}(\Delta T) = \int_{\Delta T} \{u(P_S(t) - P_{GRID}) \times (P_S(t) - P_{GRID})\}\, dt \tag{4.3}$$

where u is the unit step function.

Service quality: We model service quality with real-time latency performance of VMs and the number of VM eviction incidents. In our system model, an application-specific middleware layer provides fault tolerance over VM evictions. Therefore, we prefer minimizing VM evictions as an objective at the resource management layer. Meanwhile, real-time latency performance impacts application business logic. We prefer a bounded latency performance for that.

Problem formulation: We formulate our problem as follows: Given an arbitrary ΔT period, maximize renewable energy harvesting while preserving service quality. Thus our **objective** is:

$$\text{Maximize} \quad E_{RW}(\Delta T) \quad \text{and} \quad \text{Minimize} \quad n$$

where $E_{RW}(\Delta T)$ is the harvested renewable energy derived in Equation 4.3, and n is the number of VM eviction incidents. The objective function should satisfy the following **constraints**:

$$\bar{l}_i \leq \bar{l}_{\max_i} \quad \text{and} \quad \sigma_i \leq \sigma_{\max_i} \quad \text{for} \quad vm_i \in S_{vm}$$

S_{vm} is the virtual machines executed during ΔT and $\bar{l}_i$ and σ_i are the mean and variance of real-time latency, respectively. $\bar{l}_{\max_i}$ and $\sigma_{\max_i}$ are deployment-specific upper bounds.

4.5 Green Cores: Convert Renewables Utilization into a Packing Attribute

In this section, we introduce the concept of Green Cores, which converts the utilization of renewables as a server packing attribute, and outline its core idea, formulation, and boundaries. Green Cores enables us to design a framework in Section 4.6 to efficiently harvest renewables in Real-Time Clouds. In Section 4.8, we show the superiority of that over existing VM management approaches.

4.5.1 High-level idea of Green Cores

The core idea behind Green Cores is to identify the actual harvest of renewables, which is not reflected in Renewables-driven cores. Although Renewables-driven cores increases the available CPU cores matching that of renewables capacity, its increased core count is not an accurate signal for renewables harvest. Instead, a combination of core availability and their utilization with VMs encompass the server utilization of renewable energy capacity. Existing works addressing similar problems integrate feedback signals from power systems to identify server power draw and conduct VM packing accordingly [101] with the added complexity of integrating power domain and VM packing. Instead, we calculate a server inventory of Green Cores using the characteristics of Renewables-driven cores and our power allocation, which derives a server inventory of green and regular virtual core types. Unlike Renewables-driven cores, the number of green cores utilized maps to renewables harvest.

4.5.2 Comparison between Renewables-driven cores and Green Cores

Although both Renewables-driven cores and Green Cores may sound similar, they are distinct concepts. Renewables-driven cores is a physical notation that refers to the availability of additional cores corresponding to renewable energy dynamics. However, utilization of those additional cores does not necessarily utilize available renewables capacity. In contrast, the Green Cores is a virtual notation that converts the utilization of renewables into a packing attribute, such that utilizing a green core map to renewables

harvest.

4.5.3 Formulation of the Green Cores server inventory

In a server, we denote the number of cores that remain in a constant real-time power profile as R where $0 \leq R \leq N$, such that the size of Renewables-driven cores at time t ($l(t)$) is $0 \leq l(t) \leq N - R$. We choose a value for R such that when R cores are at a *Pinned* power state and $l(t)$ is $N - R$, the server power draw meets the grid capacity. Using our server power model in Equation 4.1 we derive,

$$P_S(t) \simeq f(R \times P_{PIN} + (N - R) \times P_{SLP}) = P_{GRID} \tag{4.4}$$

where $m(t) = R$, $l(t) = N - R$, and P_{GRID} is the grid capacity.

We derive an equation for the amount of renewable energy harvested by subtracting Equation 4.4 from Equation 4.1.

$$\begin{aligned} P_S(t) - P_{GRID} = f(&m(t) \times P_{PIN} + l(t) \times P_{SLP} \\ &+ (N - m(t) - l(t)) \times P_{ACT} \\ &- R \times P_{PIN} - (N - R) \times P_{SLP}) \end{aligned} \tag{4.5}$$

We then model $m(t)$ with R as $m(t) = R + g(t)$ where $g(t)$ is an arbitrary function. Substituting this model in Equation 4.5 yields:

$$\begin{aligned} P_S(t) - P_{GRID} = &f(g(t) \times (P_{PIN} - P_{ACT}) \\ &+ (P_{ACT} - P_{SLP}) \times ((N - R) - l(t))) \end{aligned}$$

Here, we denote the leakage power ($L(t)$): power drawn by Renewables-driven cores at the *Active* state as $L(t) = (P_{ACT} - P_{SLP}) \times ((N - R) - l(t))$.

$$P_S(t) - P_{GRID} = f(g(t) \times (P_{PIN} - P_{ACT}) + L(t)) \tag{4.6}$$

Then, substituting Equation 4.6 in Equation 4.3, we estimate renewable energy harvest for an arbitrary time period δt where $g(t) \geq 0$,

$$E_{RW}(\delta t) = \int_{\delta t} f(g(t) \times (P_{PIN} - P_{ACT}) + L(t))\, dt$$

where f is the linear function to map CPU power into server power, in which a positive input yields a positive power value. Here, when δt is small enough to match the measurement interval of the system, the $E_{RW}(\delta t)$ can be stated as,

$$E_{RW}(\delta t) \simeq f(g(\delta t) \times (P_{PIN} - P_{ACT}) + L(\delta t)) \times \delta t$$

where both $g(\delta t)$ and $L(\delta t)$ are values measured for the δt interval. The design of our Renewables-driven cores ensures that $L(\delta t)$ is independent from workload execution. In contrast, the value of $g(\delta t)$ depends on the packing decisions of the clouds control plane. Therefore, if $g(\delta t)$ is changed with different packing algorithms,

$$E_{RW}(\delta t) \propto g(\delta t) \tag{4.7}$$

Equation 4.7 states that if the packing algorithm positively increases $g(\delta t)$, the renewable energy harvest increases. However, a positive $g(\delta t)$ also means that $m(t) > R$, implying VMs are pinned to Renewables-driven cores, thus increasing the eviction possibilities.

Based on the calculation, We derive the server inventory of Green Cores. Green Cores presents a server with CPU cores of two types: Green and Regular. For Green cores, active cores ($C_{G_{active}}$) are calculated with $(N - R) - l(t)$ and used cores ($C_{G_{used}}$) are calculated with $g(t)$ if $g(t) \geq 0$ (otherwise is set to 0). For Regular cores, active ($C_{R_{active}}$), and used ($C_{R_{used}}$) cores are calculated with R, and $m(t)$ if $m(t) < R$ (otherwise is set to R), respectively. Calculations of Green cores quantify the usage of renewable energy capacity, and calculations of Regular cores quantify the usage of cores that do not increase the risks of VM eviction incidents.

This is illustrated in Figure 4.5 with a side-by-side comparison between power domain and the server inventory of Green Cores. In this scenario, the number of pinned cores ($m(t)$) increases from t_1 to t_5. As a result, server power draw ($P_S(t)$) increases.

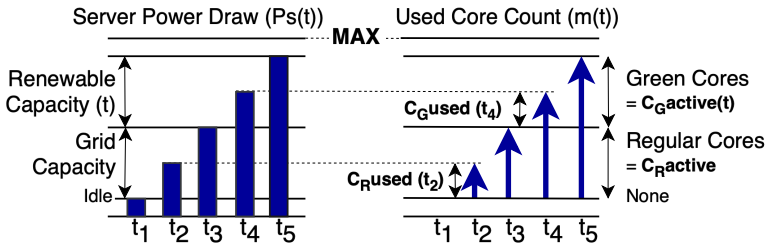


Figure 4.5: Comparison of server power draw vs proposed server inventory of Green Cores.

Until t_3, server power draw is less than the grid capacity, where $C_{G_{used}} = 0$ and $C_{R_{used}}$ is proportionate to the utilized energy capacity. During this period, there are no risks of VM eviction incidents. Beyond t_3, the risk of VM eviction incidents increases as the server power draw utilizes renewable energy capacity, where $C_{G_{used}}$ is proportionate to the utilized energy capacity and $C_{R_{used}}$ is capped at R.

4.5.4 Boundaries of the Green Cores server inventory

The derivation of Green Cores server inventory is tightly coupled with Renewables-driven cores and the power model in our system model. As a result, it relies on the accuracy of estimation techniques we used in Section 4.4.2. Nevertheless, the core idea of Green Cores can be applied to similar contexts by adjusting the inventory calculation for their specific system models.

4.6 Design

In this section, we outline the design of our framework. It combines a core-level VM Execution Model with a server-level VM Packing Algorithm. The VM Execution Model guarantees a static compute allocation for VMs at the core-level amidst the intermittency of Renewables-driven cores. To do so, it exploits the mixed-criticality of Real-Time Clouds and conducts criticality-aware VM evictions. In order to reduce the severity of that, we pack VMs at the server-level to optimize the number of best-effort and critical VM types provided to each server while maximizing the per-server utilization of

renewable energy capacity. We do that with our server-level VM Packing Algorithm.

4.6.1 Design of the Core-level VM Execution Model

We select a subset of cores and apply Renewables-driven cores to them. At 100% renewable energy capacity, we set all server cores to the real-time power profile. At 0% renewable energy capacity, we put all cores in the subset to a low power profile. For in-between, we set the real-time power profile to a partial amount of cores in the subset and set the low power profile for the rest. In this case, the number of cores in the real-time power profile is proportionate to available renewable energy capacity. For example, at 50% renewable energy capacity, half of the cores in the subset are set to the real-time power profile. In our approach, Renewables-driven cores dynamics depend solely on the renewable energy intermittency and are independent of the workload execution dynamics.

We pin VM cores to server cores set to the real-time power profile at the VM deployment and do not change it for the duration of the VM lifetime. If the number of such server cores is insufficient to serve running VMs, we perform a minimum amount of criticality-aware VM evictions. We evict best-effort VMs first and critical VMs as a last resort. Our model guarantees a static compute allocation for a VMs lifetime. The VM eviction events trigger well-defined VM failure events at the application-specific middleware layer, allowing it to recover through reconfiguration. In the next section, we design a server-level VM Packing Algorithm to reduce the possibility of such VM eviction events.

4.6.2 Design of the Server-level VM Packing Algorithm

Possibilities of VM eviction incidents with our VM Execution Model increases when the number of VMs packed in a server begin renewable energy harvesting (see Equation 4.3). Therefore, optimizing VM eviction incidents must be conducted jointly with optimizing the renewable energy harvest. We convert that joint optimization task into a VM packing optimization problem using the Green Cores server inventory we introduced in Section 4.5. We then design a server-level VM packing algorithm to address that.

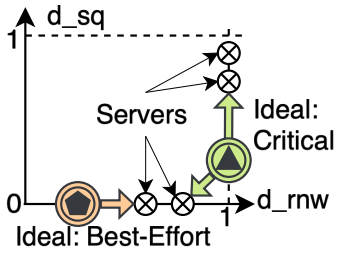


Figure 4.6: Joint optimization of renewables harvest and VM eviction incidents via the Euclidean space of Green Cores server inventory.

Algorithm 3 Proposed VM Packing Algorithm.

Input: V, S, τ_1, τ_2
Output: Placement Sorted Servers

function GETPLACEMENTPREFERENCES(V: VM, S: Candidate servers, τ_1: Ideal point for critical VMs, τ_2: Ideal point for best-effort VMs)
  $\epsilon \leftarrow GetCriticality(V)$
  $\tau \leftarrow GetIdealPoint(\epsilon, \tau_1, \tau_2)$
  for all $s_i \in S$ **do**
    $d_{sq_i} \leftarrow GetRNW(s_i)$
    $d_{rnw_i} \leftarrow GetSQ(s_i)$
    $d_i \leftarrow GetDistance(d_{sq_i}, d_{rnw_i}, \tau)$
    $s_i.score \leftarrow 1 - d_i$
  return $getSorted(S)$
function GETIDEALPOINT(ϵ, τ_1, τ_2)
  return τ_1 if ϵ is critical else τ_2
function GETSQ(s_i)
  $C_{R_{active}}(t), C_{R_{used}}(t) \leftarrow s_i$
  return $\frac{|C_{R_{active}}(t) - C_{R_{used}}(t)|}{C_{R_{active}}(t)}$
function GETRNW(s_i)
  $C_{G_{active}}(t), C_{G_{used}}(t) \leftarrow s_i$
  return $\frac{|C_{G_{active}(t)} - C_{G_{used}}(t)|}{C_{G_{active}}(t)}$
function GETDISTANCE($d_{sq_i}, d_{rnw_i}, \tau$)
  $d_{sq_\tau}, d_{rnw_\tau} \leftarrow \tau$
  $distance \leftarrow \frac{\sqrt{(d_{sq_\tau} - d_{sq_i})^2 + (d_{rnw_\tau} - d_{rnw_i})^2}}{\sqrt{2}}$
  return $distance$

Algorithm 3 outlines the proposed VM packing algorithm. It takes the VM creation request (V) and the set of candidate servers (S) as inputs. It then provides a sorted list

of candidate servers in the order of placement preference as the output. Additionally, it takes two other input parameters, called ideal points, each for critical VMs (τ_1) and best-effort VMs (τ_2). Our intuition behind the packing algorithm stems from the representation of a server in Green Cores. In that, a server is presented with two attributes: green cores and regular cores. Our algorithm uses those attributes to represent a server in a two-dimensional Euclidean feature space. Figure 4.6 illustrates that. For an arbitrary time t, we calculate a two-dimensional feature vector $\equiv (d_{rnw}, d_{sq})$ to represent a server in this space. We denote the axis d_{rnw} to quantify the opportunity to harvest renewables. We denote the axis d_{sq} to quantify the opportunity to deploy VMs with a minimum probability for an eviction incident. Firstly, we get the criticality of the V (ϵ), which is either critical or best-effort (line 2). Then, we filter the corresponding ideal point (τ) for ϵ (line 3). Afterward, we iterate through each server in S and calculate its feature vector. For the i^{th} server, we calculate the value for d_{rnw} as d_{sq_i} with $GetRNW$ subroutine (line 5), and we calculate the value for d_{sq} as d_{rnw_i} with $GetSQ$ subroutine (line 6). Using both, we calculate the Euclidean distance between the feature vector and the τ using the *GetDistance* subroutine (line 7) and derive the preference score from that as closest being the higher (line 8). Using the calculated preference score, we sort the S and provide it as the output (line 9).

Process of VM Packing: A server inventory is empty at first, where its feature vector maps to $\equiv (1,1)$. As VMs get packed, their regular core usage increases. Thus, the feature vector moves vertically towards $\equiv (1,0)$. Once all regular cores are used, its green core usage increases; thus, the feature vector moves horizontally towards $\equiv (0,0)$. This behavior allows the packing algorithm to decide its server preference depending on VM criticality. Figure 4.6 illustrates a scenario of ideal point placement. In that, the ideal point for critical VMs is placed between $(1,1)$ and $(1,0)$, such that critical VMs prefer servers with available cores supported by stable grid energy (refer Section 4.5.3) to reduce the eviction risks. In contrast, the ideal point for best-effort VMs is between $(0,0)$ and $(1,0)$, such that best-effort VMs prefer servers that draw power beyond the grid allocation to maximize renewables harvest. Tunable ideal points in our algorithm enable the cloud operator to adjust for deployment-specific performances [105]. In Section 4.8, we show its superiority in the sensitivity analysis of our large-scale VM packing testbed.

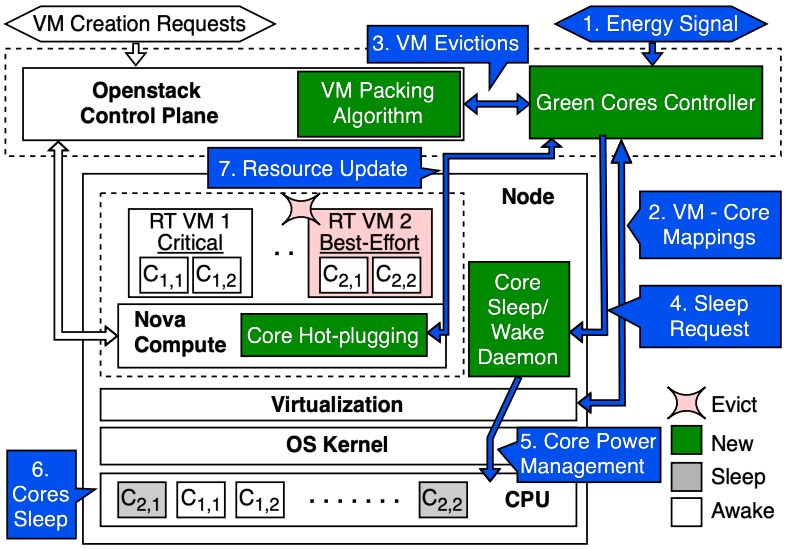


Figure 4.7: OpenStack-GC system architecture with highlighted server load matching workflow.

4.7 Implementation

In this section, we outline openstack-gc: implementation of our framework in OpenStack.

Implementation of Openstack-GC: Figure 4.7 illustrates the system architecture of openstack-gc. We highlight newly added OpenStack extensions in green. We deploy an on-node daemon service to realize Renewables-driven cores. We introduce a Green Cores Controller at the control plane to orchestrate the proposed VM Execution Model. We implement the proposed VM Packing Algorithm as a VM scheduling algorithm in OpenStack.

Renewables-driven cores: We implement an on-node daemon service in Golang to control per-core power profiles. By making an API call to the daemon service, the openstack-gc control plane can specify the number of cores to put into a specific power profile. If the real-time power profile is requested, the daemon service sets the requested number of cores into a high-performance state. If the low power profile is requested, the daemon service sets the requested cores into a deep sleep state. The daemon service wraps the Intel Power Optimization Library[1] and overrides the kernel management of the sleep state and operating frequency of each core to achieve this.

[1] https://github.com/intel/power-optimization-library.git

VM Execution Model: We orchestrate our VM Execution Model using the load matching workflows of openstack-gc. First, we enable the dedicated cores feature in OpenStack to pin each VM core to a dedicated server core, resulting in a static core allocation for each deployed VM. Then, our load-matching workflows of openstack-gc take place. Suppose an increased energy capacity signal arrives to openstack-gc. In that case, the Green Cores Controller calculates and notifies on-node daemon services to set the required number of cores from the low power profile to the real-time power profile. If a decreased energy capacity signal is provided to openstack-gc, the Green Cores Controller pings APIs of the virtualization layer (openstack-gc uses Libvirt[2]) in each node to obtain mappings of VM cores to server cores. Then, the Green Cores Controller calculates and triggers criticality-aware VM evictions by blocking API calls to the OpenStack. Upon completion, the required cores are put to the low power profile using on-node daemon services. Figure 4.7 illustrates the workflow for the decreased energy capacity. In both cases, our modified Nova Compute, OpenStacks on-node compute service, periodically polls the Green Cores Controller to obtain cores at the low power profile. Afterwards, Nova Compute signals the control place to omit cores from VM scheduling in the low power profile.

VM Packing Algorithm: We modify the OpenStack scheduler service to poll the Green Cores Controller and obtain server inventory attributes of Green Cores for all server nodes. To provide that, the Green Cores Controller pings virtualization layers of servers to obtain core usage information and calculates server inventory attributes of Green Cores. Our implementation of the proposed VM Packing Algorithm as a VM scheduling algorithm in OpenStack consumes obtained Green Cores server attributes to make VM placement decisions.

4.8 Performance Evaluation

We evaluate our core-level VM Execution Model and the server-level VM Packing Algorithm using a multi-node openstack-gc prototype deployment. Further, we evaluate its efficacy at scale using long-running production VM traces over a large-scale simulation

[2] `https://www.libvirt.org`

Table 4.4: Openstack-GC prototype: node specifications.

Attribute	**Description**
Server Model	ProLiant DL380 Gen10
CPU	Intel(R) Xeon(R) Silver 4214
Physical Cores	12
Hyper Threading	Disabled
Renewables-driven Cores	6
Real-time power profile	C-state = *POLL* at 2699 MHz
Low power profile	C-state = C6

Table 4.5: Openstack-GC prototype: VM specifications.

Attribute	**Description**
Resources	CPU: 6 Cores, RAM: 6GB
OS	CentOS 7
Kernel	Linux 3.10.0 + CERNs Real-Time patches
System Load	Load test of RTEval [7]
Latency Monitoring	Cyclictest [114]

testbed.

4.8.1 Experimental Setup

Openstack-GC prototype experiments: We deploy a prototype two-node openstack-gc cloud on HPE ProLiant servers with 12-core Intel Xeon Silver CPUs. Table 4.4 outlines its node specifications.

Workload: Table 4.5 outlines the VM specifications for our real-time workloads. We use CentOS 7 VMs with CERNs real-time kernel patches [115] applied. We run the RTEval tool from the Linux foundation project, Real-Time Linux [7], to emulate a system load. Alongside the load, RTEval continuously measures the VM kernels real-time per-

formance via the Cyclictest tool [114]. Further, we synthesize 30-minute traces for VM arrivals and renewables dynamics from Microsoft Azure's VM packing trace [8] and ELIA solar data [116].

Baseline: We use Harvest Virtual Machines (HVM): the existing VM execution model over Renewables-driven cores [6] to evaluate advancements of our VM Execution Model. We use Openstack's default VM packing implementation in its nova scheduling service [33] to evaluate advancements of our VM Packing Algorithm.

Metrics: We use Intels Running Average Power Limit (RAPL) [5] interface to capture CPU metrics in the server every 0.5 seconds. We collect i) core residencies at the C6 sleep state and ii) core operating frequency in MHz. Further, we use server power estimation using the linear power model of CPU power that is shown over 90% accuracy [112]. For that, we collect PkgWatt metric in RAPL (power consumption of the CPU socket [117]) as the server power metric (Figure 4.4 illustrate the verification of CPU power estimation with RAPL for our system model). Inside VMs, we measure real-time performance with the latency to wake up a real-time thread using the Cyclictest tool [114]. For VM packing performance, we use the Eviction Incidents to count the number of eviction incidents of best-effort and critical VMs. We use the Normalized Lifetime (nLT) to measure the severity of an eviction incident. For each evicted VM, we normalize its lifetime from the original lifetime in the trace. A lower nLT value implies increased severity. We use scheduling overhead time to measure the same with packing algorithms. For that, we analyze logs from OpenStack to identify the scheduling duration for VM creation requests.

Trace-driven simulations at scale: We use 8K+ servers, each with 40 CPU cores, to match the realistic similar values in Microsoft's Azure's cloud zones [103]. Existing fallback mechanisms of Azure's data center power delivery suggest that a 12% power overdraw is manageable [101]. To operate within that, we add four cores in each server and use them as Renewables-driven cores.

Workload: We use the full 14-day Azure VM packing trace [8], which contains request arrivals, resource requirements, lifetime on Azure, and criticality. We use renewable dynamics from ELIA solar data [116]. We normalize and scale the renewable dynamics trace, so that the maximum renewable energy capacity can wake all Renewables-driven

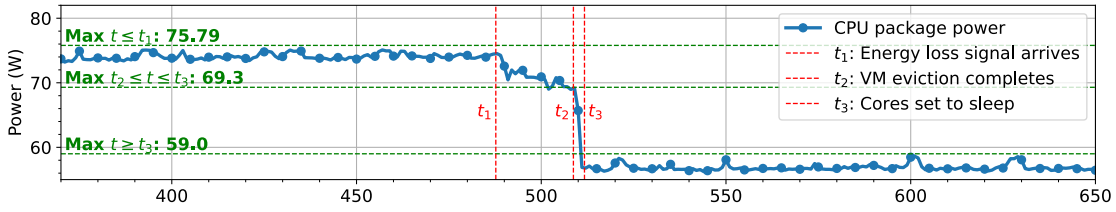


Figure 4.8: CPU package power of Intel's RAPL interface [5] during load matching of OpenStack-GC prototype. The proposed VM Execution Model conduct server load matching: $t < t_1$: server with two 6-core VMs, $t = t_1$: energy loss triggers a VM eviction, $t = t_2$: VM eviction completes, and $t = t_3$: unpinned cores enter deep sleep.

cores in a server.

Baselines: To evaluate the proposed VM packing algorithm, we use two comparison baselines. The Best-Fit packing (best-fit) is a commonly used packing approach in production clouds [103, 118] that packs VMs tightly in servers. We use it to evaluate our advancements over a commonly used VM packing approach. The Criticality-Aware packing (crt-aware) is a packing approach that reduces VM throttling incidents in power over-subscribed data centers [101]. Similar to our problem context, it leverages VM criticality to reduce VM performance impact incidents incurred from server load exceeding available power capacity. We use it to evaluate our advancements over the state-of-the-art.

Metrics: In addition to Eviction Incidents and Normalized Lifetime (nLT), we use the Harvested Renewables to measure utilization of renewable energy capacity. Using derivations of Green Cores, we calculate it as $\equiv \int_0^T C_{G_{used}}(t)dt$ for a period of T.

4.8.2 Evaluation of Core-level VM Execution Model

We evaluate the proposed VM Execution Model's ability to maintain the server load to match available renewable energy capacity and its impact on the real-time performance of VMs. Firstly, we signal openstack-gc prototype deployment with a 100% energy capacity to wake all Renewables-driven cores in the server. Then, we pin all cores by deploying two 6-core VMs. In both VMs, we run the RTEval program for the duration of the experiment to emulate a peak load. Then, we signal a 0% energy capacity.

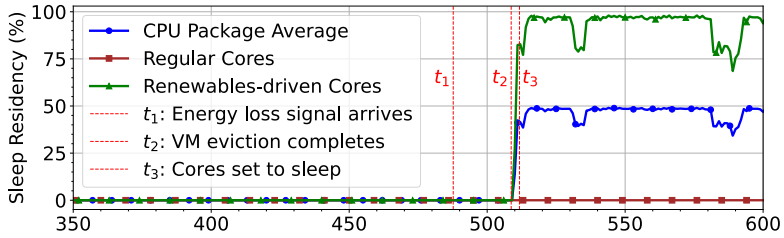
Figure 4.8 shows the CPU package power observed throughout. We collect it via

Intel's Running Average Power Limit (RAPL) [5] interface. The CPU package draws up to 75.79W at peak load with a relatively constant trend. t_1 denotes the arrival of the energy loss signal for 0% renewable energy capacity. openstack-gc's response shows a two-stage power reduction; t_1 - t_2 and t_2 - t_3. The former shows the power reduction from evicting one of the VMs to unpin six cores. Latter shows the power reduction from putting unpinned cores to deep sleep. After t_3, CPU package power does not exceed 59W. It translates to a 22% reduction of the peak power draw. With the linear model of CPU power to server power [96], our openstack-gc deployment shows a reduction of 22% of the server peak power in matching 100% to 0% loss of renewable energy capacity.

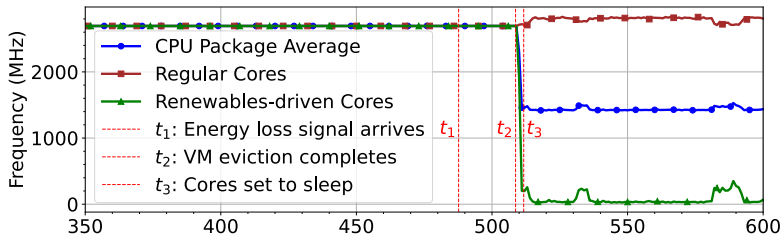
Figure 4.9 shows CPU core power characteristics. We capture operating frequency and C6 deep sleep state residency (i.e. in a given measurement period, the percentage that the core resided in the sleep state) of cores. We average it for the six cores that enter deep sleep state after t_3 (Renewables-driven cores), for the six cores that continue to operate, and for overall. Until t_2, openstack-gc maintains a constant 2700 MHz operating frequency of cores with 0% residency in deep sleep, showing cores operating at the real-time power profile. Afterwards, Renewable-driven cores mostly reside in a deep sleep with 0 MHz operating frequency, showing their low power profile. It shows that openstack-gc only changes power profiles after the VM eviction completion at t_2. Throughout the lifetimes of VMs, VM cores are allocated with physical cores in the real-time power profile.

The spikes in maintaining the low power profile show the characteristics of controlling core power through Intel's Power Optimization library that we use in openstack-gc. Our prototype also has an overhead of running external services, including the OpenStack control plane services in the same server, which could be attributed to the spikes shown. Despite that, the server power draw shows a 59W upper bound in figure 4.8, showing openstack-gc can maintain a constant power reduction over the intermittent spikes in the low power profile.

We then evaluate the superiority of our VM Execution Model's static core allocation shown in the load matching experiments by comparing that with baseline HVM's approach of dynamic physical core allocation. HVMs approach is to change the number of physical cores (pCPUs) allocated to the VM while preserving the number of virtual



(a) CPU core C6 deep sleep state residency.



(b) CPU core operating frequency.

Figure 4.9: CPU core power of Intel's RAPL [5] in server load matching of Openstack-GC prototype.

cores (vCPUs). To evaluate its workload impact on real-time computing, we monitor the real-time performance of an HVM over the dynamic allocation of pCPUs. Our experimental HVM consists of 2 vCPUs. We set pCPUs to the real-time power profile. We then execute the RTEval [7] program inside the HVM to measure the real-time performance. We dynamically adjust the mapping of pCPUs through our virtualization management, libvirt's APIs[3].

The results are illustrated in Figure 4.10. When the number of pCPUs $\geq$ to the number of vCPUs, the HVM sustains a consistent real-time latency performance, having both mean and mean absolute deviation statistics consistent for all three cases of pCPUs ≥ 2. The opposite shows increased latency variance inside the HVM. Compared to pCPUs $= 2-$which maps to the performance of our VM Execution Model due to its static core allocation$-$the case of pCPUs $= 1$ increases the mean latency by 30% alongside a 7.32$\times$ increase of the mean absolute deviation of latency. In contrast, our VM Execution Model can incur VM evictions. In the next experiment, we evaluate its impact on real-time application performance.

[3] `https://www.libvirt.org`

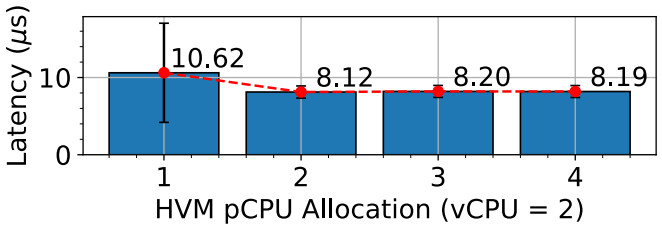


Figure 4.10: Comparison of real-time latency performance over pCPU allocations for 2-core HVM [6].

Next, we evaluate the impact of the proposed VM Execution Model at the application layer. We use the Harvest VM (HVM) as a comparison baseline, the existing VM execution model over Renewables-driven cores[6]. We use an experimental deployment of OSM MANO [4], a Virtual Network Functions (VNF) orchestration and management application layer, to match a real-time application layer having both critical and best-effort components (see Table 4.2). In public clouds, server utilization is around 60%, and the packing density (i.e. utilization of servers running at least one VM) is around 85% [103]. To match that, we use two 12-core servers and tightly pack one server with two 6-core VMs while the other is left unused. MANO is a generic orchestration layer where the exact time-bound requirements depend on the use case. Therefore, for VMs, we measure the real-time latency of the VM kernel for a consistent real-time latency performance independent of the use case. To match application-level reconfiguration over component failures, we enable MANO's auto-heal feature, which reconfigures itself via VM redeployment. HVM executes VMs under resource variations. To match it's worst-case, we set the dynamics of Renewables-driven cores to sleep five cores in both servers, such that HVM executes a VM with 6 VM cores allocated to one physical core. In contrast, the proposed approach evicts one of the VMs, triggering MANO to redeploy it in the unused server. We obtain the time taken for reconfiguration via MANO's event logs.

Figure 4.11 shows the real-time latency performance of the affected VM in both the proposed and HVM approaches. In both methods, the remaining VM continues executing under the same core allocations without any performance impact since the server's physical core count is sufficient. Until the server core sleep event at time axis $= 300$, the affected VM in both approaches shows the same mean real-time latency. Afterwards, the core count reduces. With HVM, the affected VM's mean real-time latency increases from

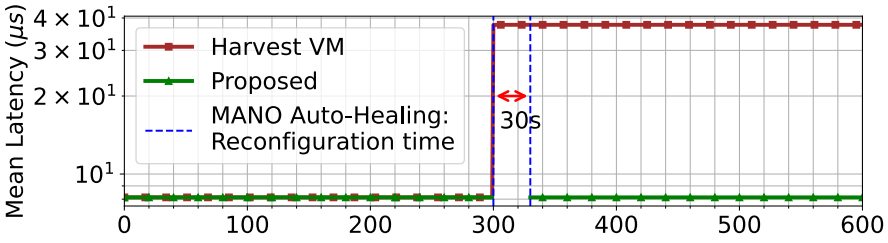


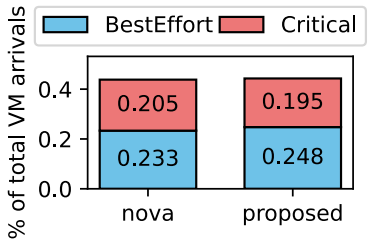
Figure 4.11: Real-time performance comparison for different VM execution models with OSM MANO [4] as the application layer. Mean real-time latency of RTEval [7] in affected VMs is plotted over the experiment duration. The proposed model evicts VMs and HVM reduces allocated physical cores.

8.13μs to 37.65μs. When comparing the coefficient of variation of the VM's real-time latency, it increases by 6.52×. With the proposed approach, the affected VM undergoes a 30-second service unavailability. However, when it resumes afterwards, the VM retains the same real-time latency performance. The results show that, unlike the existing temporal flexibility-based approach, the proposed VM Execution Model maintains intact real-time latency performance. In doing so, it incurs brief service unavailability from VM evictions as a trade-off. In the next section, we evaluate the role of our proposed VM packing algorithm in reducing the impact of that on the application layer.

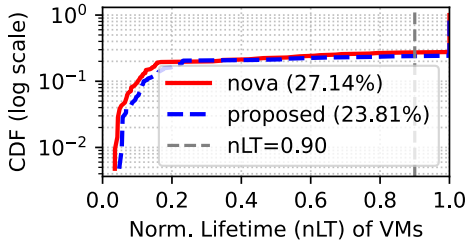
4.8.3 Evaluation of Server-level VM Packing Algorithm

We first evaluate the practical aspects of our VM Packing Algorithm with the openstack-gc prototype over the default OpenStack. We focus on the scheduling overhead of our implementation and the impact of service quality on renewables dynamics. Then, we replay long-running VM arrivals and renewable dynamics to evaluate renewables harvest and long-term service quality impact with the large-scale simulation testbed.

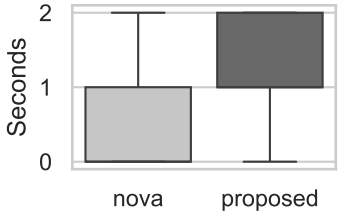
We write a Python client to read VM arrivals in the Azure trace and make VM creation requests to openstack-gc deployment in real time. For each VM request, it spawns a lifecycle management thread, which then waits in real-time and makes the VM creation request. Afterward, it periodically polls the deployment to check the deployed VM status. Management thread completes if the VM has prematurely deleted, which is then marked as an eviction incident, or the VM has lived to the lifetime provided in



(a) Number of VM eviction incidents.



(b) Distribution of normalized lifetimes (nLT) of VMs with nLT CDF value $\leq$ 90%.

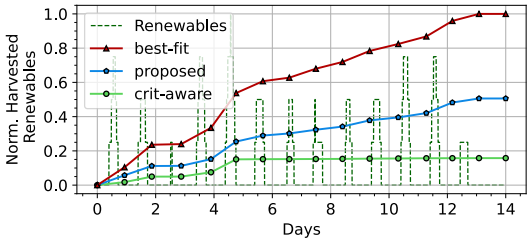


(c) Distribution of scheduling overhead in VM deployment.

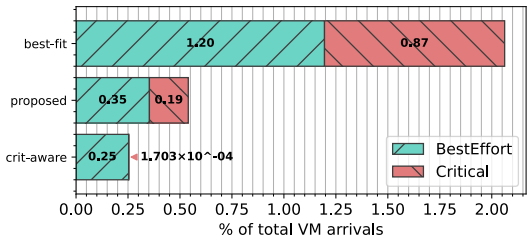
Figure 4.12: Performance of the proposed VM Packing Algorithm in VM packing experiments of openstack-gc prototype.

the trace data, which then is deleted via a request made to the deployment. In parallel, a separate client emulates renewables dynamics by reading the trace data. It emulates a single peak renewables dynamics matching 24-hour solar availability by switching Renewables-driven cores through openstack-gc APIs.

Figure 4.12 illustrates our results. Our proposed algorithm outperforms OpenStack nova regarding the severity of eviction incidents. Eviction incident counts in Figure 4.12a show our proposed algorithm reduces critical VM percentage from 0.205 to 0.195 while leveraging that with best-effort VM evictions. In Figure 4.12b, our proposed algorithm reduces CDF value for the 90% of normalized lifetime of VMs from 27.14% to 23.81%. In return, Figure 4.12c illustrates the distribution of scheduling overhead in VM requests. Note that OpenStack logs that we leverage for that have a granularity of seconds. In measuring the overhead values, we disable the synchronization overhead of openstack-gc for OpenStack nova. In return, results demonstrate the impact of additional scheduling overhead in openstack-gc, where the overhead distribution con-



(a) Accumulation of harvested renewable energy capacity.



(b) The number of VM eviction incidents.

Figure 4.13: Joint optimization of renewables harvest and VM eviction incidents of the proposed VM Packing Algorithm for the 14-day Azure VM trace [8].

centrates to 2 seconds from 1 second. Increased scheduling overhead can delay the VM deployment, resulting in lesser optimized packing decisions. For example, servers may increase critical VM eviction incidents with renewable dynamics due to the lack of best-effort VMs. Most of the scheduling overhead is attributed to the synchronization implementation of openstack-gc prototype, where the controller polls each server to collect information in calculating the server inventory of Green Cores. With that, the overhead shown in the results can increase with the deployment size. However, apart from that, the remaining algorithm implementation does not significantly increase the scheduling overhead. We use OpenStack's default filter scheduler and integrate our algorithm with its existing iteration of servers, avoiding additional re-iterations. Nevertheless, our synchronization implementation in the openstack-gc prototype can be improved by applying scheduling optimization techniques. For instance, Azure's production VM scheduler, Protean [103], addresses a similar scaling problem in VM packing by implementing an optimistic concurrency model. Collectively, the results of VM packing with openstack-gc prototype highlight its potential in improving VM eviction severity and opportunities to improve its scheduling overhead for production deployments. Next, we evaluate the proposed VM packing algorithm at scale for renewables harvesting over long-running experiments.

For that, we use a large-scale simulation test bed to evaluate our framework at the data center scale. In the test bed, we first implement Renewables-driven cores with a trace of renewable energy dynamics and then implement the proposed VM Execution

Model. We expose the test bed to a 14-day Azure VM workload arrival trace. We use our proposed VM packing algorithm and comparison baselines to determine VM allocations across servers.

We first tune our algorithm by observing its performance over short-running experiments. We aim to jointly optimize the reduction of VM eviction incidents and increase renewable energy harvest. Once tuned, we conduct 14-day packing experiments. Figure 4.13 shows (a) harvested renewable energy and (b) the number of VM evictions. Values for the former are normalized among the comparison baselines, and values for the latter are expressed as a percentage of the total number of VM requests.

Both baselines show their inability to conduct joint optimization. The best-fit algorithm is most effective in harnessing renewable energy yet evicts over 2% of VM requests. It incurs the highest amount of critical VM evictions among the three algorithms. The crit-aware, on the other hand, shows the most effectiveness in reducing VM eviction incidents with 0.25% of total VMs evicted with a 1.703×10^{-4}% of critical VM evictions, the lowest amongst three algorithms. However, it shows the least harvested renewable energy with an 80% reduction from the best-fit algorithm. Our proposed algorithm shows a joint optimization, a 34.83% increase over crit-aware in harvested renewable energy and a 79.64% reduction of VM eviction incidents compared to best-fit. Our algorithm reaches 50% of the renewable energy harvest performance of best-fit with a 26.09% VM eviction incidents of best-fit, showing its joint optimization characteristic to favour lesser eviction incidents. Figure 4.14 shows distributions of a normalized lifetime (nLT) of evicted VMs. The proposed VM packing algorithm surpasses best-fit and approaches crit-aware with the CDF value for nLT $\leq$ 90%.

Sensitivity analysis: We conduct a sensitivity analysis of our algorithms hyper-parameters. In the proposed packing algorithm, we represent each server using a 2-dimensional feature vector $\equiv (d_{rnw}, d_{sq})$: d_{rnw} quantifies renewable energy usage and d_{sq} quantifies the possibility of VM eviction incidents. Parameters of our algorithm are two instances of this vector (called ideal points), one for each critical and best-effort VM type. For initial values, we set critical VM ideal point to $(1, 0.5)$ such that those VMs prefer servers with the potential to reduce VM eviction incidents, and best-effort ideal point to $(0.2, 0.0)$ such that those VMs prefer servers with the potential to harvest renewable energy.

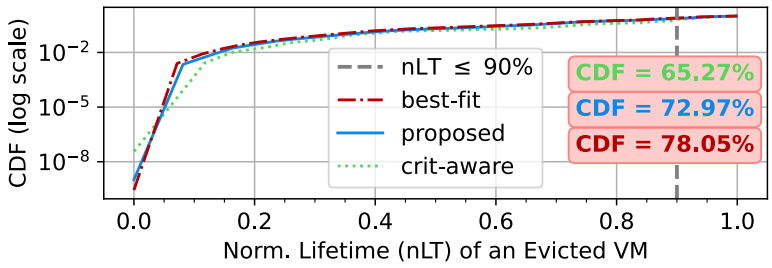
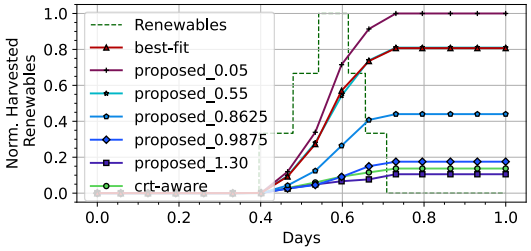
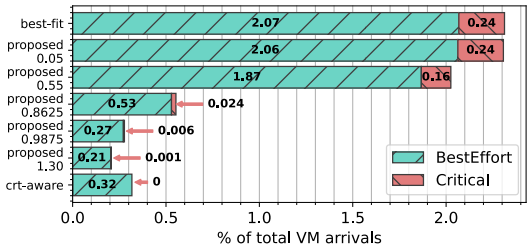


Figure 4.14: Distribution of normalized lifetime (nLT) of evicted VMs during the 14-day VM packing experiment.



(a) Accumulation of harvested renewable energy capacity.



(b) The number of VM eviction incidents.

Figure 4.15: Sensitivity analysis performance of the proposed VM packing algorithm. 24 hours experiments are conducted as the distance between ideal point parameters change.

In subsequent experiments, we move critical ideal point closer to the other and conduct 24-hour packing experiments in each step. Figure 4.15 illustrates our results. As ideal points move closer, our proposed algorithm favours increasing renewable energy harvest, surpassing the leading baseline best-fit at a distance of 0.05. Although this behaviour compromises eviction incidents, the number of incidents is still less than that of the best-fit. In contrast, as ideal points deviate, our proposed algorithm favours decreasing eviction incidents, surpassing the leading baseline crt-aware at distances of 0.9875 and 1.30.

4.8.4 Discussion

Our evaluations show the potential of our framework to manage server power using Renewables-driven cores. Per-core application of low power profile demonstrates our

framework can reduce the server power to match supply variations of renewable energy. CPU power metrics shown during that indicate that if a core pins to a VM, that core's power profile transition will not occur. Even if unused cores are insufficient, the framework evicts the VM first before changing the power profile. As a result, our framework guarantees a static compute allocation throughout a VM's lifetime. Evaluation of the real-time latency performance of VM kernels shows that the static compute allocation provided in our framework significantly outperforms existing workload temporal-flexibility-based VM execution solutions.

Our approach shows two trade-offs. Firstly, a sustained core power profile until the completion of VM evictions requires redundancies in the data center power delivery to support the short periods of server power overdraws. However, existing cloud data centers can support similar requirements [101]. Therefore, our framework fits into existing data center designs. Secondly, the static compute allocation requires VM evictions if enough unused cores are unavailable to match the energy supply. However, an application-specific middleware layer in clouds manages real-time VMs, which provides fault tolerance over VM evictions [4, 17]. Therefore, VM evictions in our framework do not incur application-level failures for real-time workloads. Moreover, large-scale packing experiments show that our framework can reduce the number of VM eviction incidents by utilizing core availability across the servers, jointly optimizing that with the utilization of renewable energy capacity. Our eviction-based approach exploits findings of a previous study showing that cloud applications prefer VM evictions over continued VM execution with performance degradation [101].

Performance of our framework improves with the presence of best-effort VMs. Therefore, cloud operators need to tune our algorithm according to the workload variations. Sensitivity analysis of our framework's packing algorithm parameters shows the ability to support that (see Section 4.8.3). The algorithm can be tuned to favour renewable energy harvesting for a deployment that expects an increased number of best-effort VMs. Otherwise, it can be tuned down to reduce the number of VM eviction incidents. Our framework design expects server utilization levels in typical data centers, where a slack of unused capacity is available [103]. It allows the real-time application layer to reconfigure in the events of VM evictions. If the data center utilization levels are much higher,

the application layer may be unable to do so. In such cases, the algorithm tuning must be adjusted to reduce eviction incidents.

4.9 Summary

In this chapter, we explored how to integrate intermittent renewable energy sources with deterministic real-time cloud systems. To this end, we proposed a framework consisting of a VM Execution Model and a VM Packing Algorithm, which guarantees real-time power profiles for workload VMs and jointly optimizes renewable energy harvesting and application service quality. We practically implemented the proposed framework as openstack-gc by extending OpenStack with an on-node per-core CPU sleep management daemon and a controller at the control plane. We evaluated the framework's power management and VM packing by experimenting on a two-node openstack-gc prototype. Furthermore, we conducted 14-day long-running experiments to capture the framework's performance in optimizing renewables harvesting and VM interruptions. As evidenced by our experiments, the proposed framework demonstrated its superiority in real-time workload management by reducing the coefficient of variation of real-time latency in VMs by 6.52× over the existing workload temporal-flexibility-based solution. Additionally, it showcased the safe energy harvesting capability with a joint 79.64% reduction of VM eviction incidents and 34.83% increase of harvested renewable energy over state-of-the-art baselines.

This chapter utilized renewables-driven cores to optimize carbon for temporally inflexible real-time cloud systems. The key idea behind renewables-driven cores is to manage server performance for available energy capacity while isolating performance degradations to specific CPU cores, such that the remaining cores can operate at the nominal performance. We did not explore the core-level performance isolation with renewable energy dynamics beyond the real-time workloads. In the next chapter, we study core-level performance isolation with low-latency applications, another type of temporal inflexible workload that is prominent in public clouds. We explore opportunities present in latency Service Level Objectives of low-latency applications through core deep idling to accommodate them with multi-region renewable energy harvesting.

Chapter 5

Load Shifting for Low-latency Applications with SMT Core Pooling

Low-latency applications in clouds are often overlooked in multi-region renewable energy harvesting approaches. Cloud regions are typically interconnected via inefficient Wide Area Networks (WANs); thus, shifting and serving low-latency workloads over WANs can impact application latency Service Level Objectives (SLOs). In this chapter, we focus on containing low-latency applications within their local region to avoid serving over WAN and propose a load-shifting technique based on core-level server power management of Simultaneous Multi-threading (SMT) server pooling. Using a hardware-software co-design approach, our technique maintains a static set of logical cores amidst renewable energy dynamics and efficiently utilizes that for co-scheduling low-latency and best-effort applications. We practically implement our approach with OpenStack and Intel Hyper-threading technology of Intel CPUs and evaluate against Azue VM traces. Our results demonstrate that in comparison to the state-of-the-art baseline, our proposed technique achieves an 80% reduction in offloading low-latency VMs and a 43.81% reduction in coefficient of variation of p90 end-user latency while having a worst-case latency compromise of 11.97% due to SMT cores.

5.1 Introduction

Cloud data center fleets are often spatially distributed and powered by sources integrated with intermittent renewables, emitting varying amounts of CO_2 per kilowatt-hour (kWh) [96]. As a result, carbon optimization in cloud computing infrastructures

This chapter is derived from:

- **Tharindu B. Hewage**, Shashikant Ilager, Maria A. Rodriguez, and Rajkumar Buyya, "A Technique for Load Shifting Low-latency Applications in Multi-Region Renewables Harvesting via SMT Core Pooling", *IEEE Transactions on Power Systems* [Submitted, May 2025].

predominantly relies on load shifting—shifting workloads across data center locations via cloud Virtual Machine (VM) scheduling [95–97, 119, 120]—to chase availability of renewables for workload execution. Adversely, network traffic among spatially distributed data centers can hop through Wide Area Networks (WAN) with high latency variances [11], resulting in unpredictable end-user latency performances in VMs. For instance, when aggregated network traffic of VMs reaches edge routers of WAN, traffic management at the IP layer that is unaware of the aggregation can split them into tunnels with different latency performances [11]. To avoid potential application performance degradation from such unpredictable latency performances, cloud providers often limit load shifting to best-effort workloads with flexible latency requirements, such as scientific simulations, batch processing, and machine learning training [96]. To this end, this chapter explores the opportunities in accommodating low-latency workloads for cloud load shifting, emphasizing minimizing the high latency variance that VMs experience during load shifting.

Use cases of cloud low-latency computing are emerging across various industries, including healthcare, factories, automotive, and aviation [105]. According to recent forecasts, they are predicted to accommodate nearly 30% of the world's data in the near term [99]. Today, low-latency computing plays a vital role in content delivery networks, streaming applications, and applications of recent AI-boom, such as serving generative AI models. Figure 5.2 illustrates a spectrum of such applications available in the cloud according to their latency Service Level Objectives (SLOs). Preserving latency SLOs is imperative to maintain application service quality, whereas accommodating the growing low-latency applications in load shifting is equally important for the sustainable growth of clouds. The key challenge is the workload impact from cloud network overheads. At renewables supply valleys, the data center's workload must be migrated to match the server's reduced resource capacity. As migrations often move workloads across WANs, workloads are then susceptible to WAN's high latency variances. Existing techniques for load shifting either compromise latency performance with potential server performance throttling [54] or do not consider opportunities in limiting workload migration within the data center.

To address these gaps, we propose a technique that aims to contain low-latency ap-

plications within the local cloud region. We leverage a hardware-software co-design approach. At the hardware level, we tackle the server resource capacity reduction problem in renewable supply valleys. We apply an application-independent CPU core-level power management mechanism with two heterogeneous server pools. CPUs in server pools are physically the same, yet one enables simultaneous multi-threading (SMT) to double the available server logical cores through hardware multi-threading. During supply valleys, half of the CPU cores are set to deep idle and restored at the peaks. In return, we maintain a static set of logical cores across the server pools. At supply valleys, servers of the SMT pool exhibit the static set, whereas the servers of the non-SMT pool exhibit the static set at supply peaks. At the software level, we dynamically chase the static set of logical cores across the server pools for low-latency workloads. In our technique, load shifting for low-latency workloads is mostly conducted within the local cloud region using its fast network fabric, eliminating the communication overheads of WAN. Further, executing low-latency workloads in the SMT pool incurs only a minimum performance overhead since SMT cores use hardware multi-threading, which provides better performance.

We implement our technique with OpenStack and core-level power management with CPU idle states. To evaluate, we use an experimental cloud region with SMT server pooling and a local network fabric. We use an HP ProLiant server for each pool with a 12-core Intel Xeon silver CPU. We enable SMT through Intel Hyper-threading technology. We use VM arrival data from Azure's workload traces and renewable dynamics from ELIA solar data. We measure low-latency performance inside VMs by running the Cyclictest tool. The **key contributions** of our work are as follows:

(1) Propose a new technique for localized load shifting of low-latency workloads to integrate renewable energy, avoiding the latency compromises of workload shifting over WANs across geographical regions.
(2) Implement the proposed technique in real cloud settings and conduct detailed experiments using production VM and renewable energy data.
(3) Evaluate the proposed technique against state-of-the-art baselines, focusing on maintaining low-latency performance. Our results show an 80% reduction in offloading low-latency VMs, a 43.81% reduction in coefficient of variation of p90 end-user latency, and

Table 5.1: Comparison of relevant works with our proposed technique for load shifting with low-latency applications.

Work	Multi-cloud Carbon Optimization	Load Shifting	Uninterruptible Execution	Low-latency Applications	Static Resource Capacity
Radovanovic'23 [96]	✓	✓			
Carbonscaler'23 [121]	✓	✓	✓		
Zheng'20 [97]	✓	✓	✓		
CDN-Shifter'24 [122]	✓	✓	✓	✓	
Our Proposed	✓	✓	✓	✓	✓

an 11.97% performance compromise due to SMT cores.

The rest of the chapter is organized as follows: Section 5.2 discusses related work. Section 5.3 provides the background and motivation for our problem. Section 5.4 details our system model and problem formulation. Section 5.5 presents our technique to accommodate low-latency applications for multi-region renewables harvesting using SMT core pooling. Section 5.6 outlines our implementation of the proposed technique. Section 5.7 presents the performance evaluation of the proposed technique, and finally, Section 5.8 summarises the chapter.

5.2 Related Work

Integrating renewables in clouds: Many previous studies have explored integrating renewable energy into cloud data center operation. They commonly exploit load shifting techniques in both space and time, such as geographical load balancing [96, 121, 122], VM migrations [95–97, 119, 120], and workload admissions and capacity planing [96]. Most techniques are intended for flexible workloads that can tolerate delayed execution and responses. For instance, carbon optimization at Google [96] leverages internal workloads that can withstand delays as long as 24 hours, such as machine learning, data processing, and simulation. They employ virtual capacity curves across their geographi-

cally dispersed data centers and employ temporal shifting to perform carbon-optimized computing. Some works explore shifting workloads across locations over WAN, considering the impact of WAN's dynamic traffic congestion towards the workloads [123]. However, they do not focus on low-latency applications and only focus migration cost calculations for the energy cost of the transfer. In contrast, we explore geographical load shifting to accommodate low-latency applications and provide a technique to better manage server power over intermittent renewable energy while maintaining service quality of low latency applications.

VM scheduling of dynamic inventories: Cloud VM scheduling for energy management is a widely researched problem [103, 104]. Many aim to optimize VM scheduling to increase energy efficiency for static inventories, where server resources remain static and server power is managed through the workload management [124]. However, recent works explore VM scheduling for dynamic inventories where available server resources are adjusted to tolerate the changes in the data center power delivery. For instance, Microsoft increase utilization of their clouds via power oversubscription, where server are provisioned to oversubscribe the power capacity [101] and power overdraw events are managed by throttling the server compute capabilities. They leverage a dynamic VM scheduling technique to reduce the workload impact, considering VM priority levels. However, workload throttling in their approach can lead to non-deterministic application performances. In our work, we employ a dynamic inventory to absorb renewable intermittencies and conduct VM scheduling to leverage the idea of chasing static resources within the datacenter for low-latency applications, providing better stability in application performance.

5.3 Background and Motivation

In this section, we provide background on load shifting for the utilization of renewable energy across geographically distributed cloud regions, introducing its dynamic server resource scaling problem and challenges in accommodating low-latency applications. We then detail our motivations for solving those.

Dynamic server resource scaling problem in utilizing intermittent renewable energy

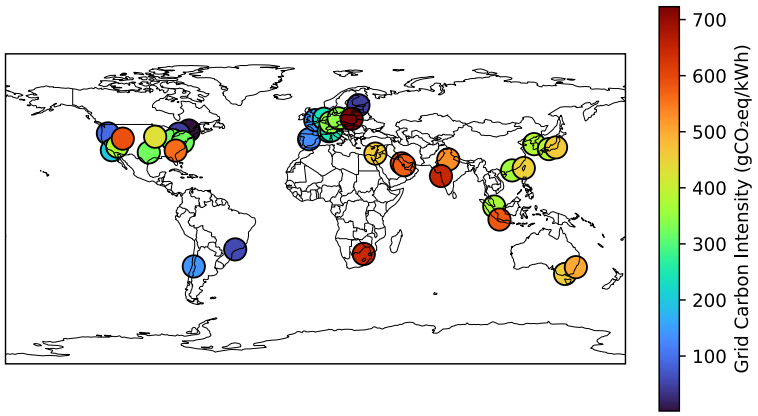


Figure 5.1: Average operational gross carbon emissions per unit of energy from the power grid across Google cloud regions [9].

availability across cloud regions: Cloud providers provision data centers across geographical regions around the globe to cater to various requirements, such as end-user latency and data privacy. [9]. Those regions are powered by electricity grids with varying carbon intensities in their supply, primarily due to the integration of intermittent renewable energy sources such as solar and wind [121]. Recently, driven by 24/7 zero emission goals [125], hyper-scale cloud providers increasingly utilize low-carbon computing opportunities across their cloud regions [96]. Figure 5.1 illustrates grid carbon intensity per unit of energy across Google cloud regions. In that, executing workloads with lower carbon intensity is beneficial in reducing cloud providers' carbon footprint. In doing so, cloud platforms model carbon intensity as a resource for optimization. Since the carbon intensity of cloud regions predominately depends on the supply dynamics of renewable energy sources, it can be modeled as a dynamic server resource scaling problem, where server resource capacity scales up or down depending on the supply dynamics of clean energy. Recent examples of such can be seen in virtual capacity caurves [96] and workload scaling for carbon-optimization [121]. The key approach in addressing dynamic server resource scaling is load shifting, where the execution of workloads is either shifted in time or space across cloud regions.

Challenges in accommodating low-latency applications for load shifting: Low-latency

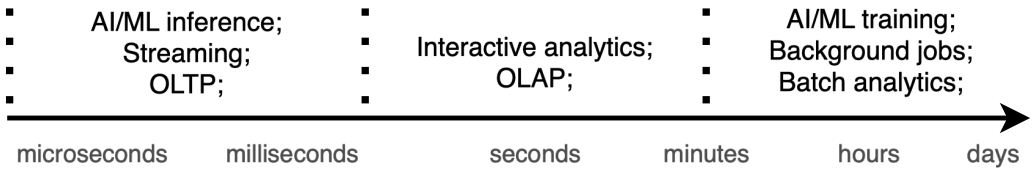


Figure 5.2: Low-latency applications in cloud data centers [10].

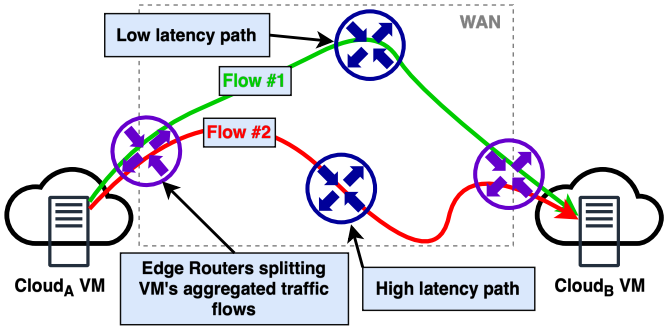


Figure 5.3: Traffic management at IP layer splitting aggregated traffic flows of cloud VMs when communicating over WAN [11].

applications require maintaining an upper bound in their end-user communication latency. Figure 5.2 illustrates a spectrum of end-user latency for various low-latency applications in the cloud. Such applications are increasingly prevalent [99]; thus, their impact on carbon-aware cloud computing is growing. However, low-latency applications are not flexible in load shifting, primarily due to the complexities in maintaining their rigid performance constraints amidst server resource down-scaling of renewable energy supply valleys. These applications cannot tolerate longer service unavailability in shifting across time, and more importantly, they cannot tolerate shifting across space (i.e., shifting among cloud regions) due to unpredictability in network performance connecting the cloud regions. Figure 5.3 illustrates an example of that. Geographically distributed cloud regions are connected using wide area networks (WAN). Each cloud region connects to the WAN using an edge router and passes its network traffic flows to the WAN's traffic management. Recent studies show that WAN's traffic management at the IP layer is unable to distinguish aggregated traffic flows of a cloud virtual machine; thus, an aggregated traffic flow can get split at the edge router, routed through different paths with various latency performances before arriving at the destination [11]. The resulting ap-

plication end-user latency is unpredictable and unfavorable to application users [101]. Further, increased latency performance may violate application service level agreements (SLAs), which can incur costly penalties to the cloud provider.

Motivations: Therefore, to accommodate low-latency applications in load shifting, we identify server resource down-scaling at renewable energy supply valleys as its major bottleneck. Since load shifting must offload workload outside the local cloud region to match scaled-down server resource capacity, we **hypothesize that maintaining a static resource capacity amidst renewable supply valleys for low-latency applications would enable load shifting to retain those within the local cloud regions**, avoiding costly communication overheads of WANs. In this context, we outline the following research questions to draw our motivations for this chapter.

o1*: Can we manage static server resource capacity amidst renewables valleys to yield an infrastructure favorable for low-latency applications?*

o2: *For a technique designed to achieve o1, what compromises are made, and more importantly, can the compromises operate within the service level agreements (SLAs) of low-latency applications?*

In order to address *o1* and *o2*, we propose a load-shifting technique to accommodate low-latency applications in carbon-aware cloud computing. Section 5.5 details its inner workings, Section 5.6 describes our implementation of the proposed technique, and Section 5.7 evaluates and discusses its performance over state-of-the-art baselines.

5.4 System Model and Problem Formulation

Figure 5.4 illustrates our system model. We consider a multi-region cloud deployment. Each region integrates intermittent renewable energy in the data center power delivery. As a result, data center servers in each cloud region undergo dynamic energy capacity availability. In return, server compute capacity dynamically scales up and down. We model that with the dynamic server resource scaling model. Workload schedulers in cloud regions match their workload execution with the server resource availability via load shifting both within the data center and data centers across regions via VM offloading over a Wide Area Network (WAN). Regions receive workloads as VM arrivals,

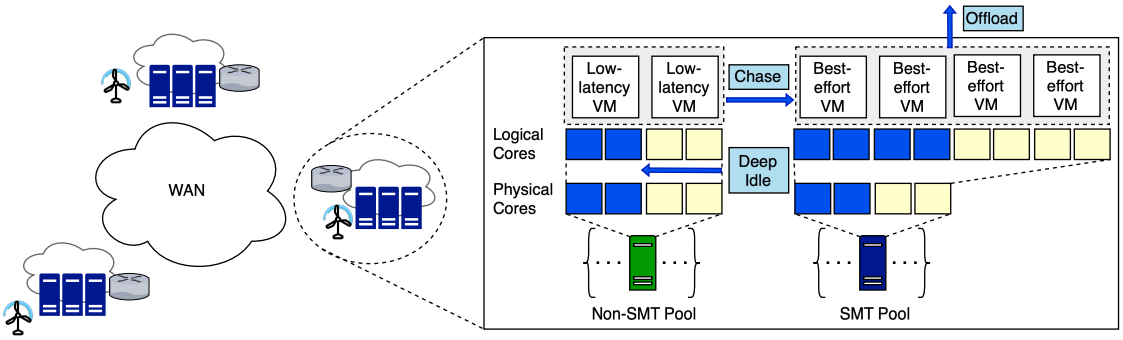


Figure 5.4: System architecture of the proposed load shifting technique.

which are either VMs of low-latency applications or VMs of best-effort applications. The WAN that connects regions offer network routes with varying latency performance, which we model with the latency performance model. Using the derived models, we then formulate our problem.

Server dynamic resource scaling model: We model server resource scaling from the dynamic availability of renewable energy capacity through availability of CPU cores. Given the energy capacity, server power is managed by enabling a sufficient amount of CPU cores. The same technique has been commonly used as a core-level server power management mechanism [6, 102].

A data center in our model integrates a mixed power delivery of both carbon-intensive stable power sources, such as fossil fuel-based energy generation, and low-carbon-intensive intermittent renewable energy sources, such as solar and wind. Stable sources provide baseline energy capacity for the data center, whereas intermittent sources provide peaks and valleys of low-carbon-intensive energy availability. Data center power consumption is typically a combination of information and technology components (IT) such as servers, cooling, and internal power conditioning systems [110]. Servers in our system model facilitate CPU-intensive low-latency and best-effort workloads. Therefore, majority of dynamic power draw variations corresponds to CPU power. For such systems, prior works show server power can be estimates as a linear function of CPU power with 90% accuracy [112]. Moreover, for multi-core CPUs, the cumulative sum of core power becomes a close upper bound to the CPU power [113]. Combining both, we use the following model to estimate server power.

$$P_S(t) = f(\sum_{i=1}^{N} P_{CORE_i}(t)) \tag{5.1}$$

where at time $t = t$, $P_S(t)$ is the server power, N is the number of homogeneous cores, $P_{CORE_i}(t)$ is the power consumption of the i^{th} core, and f is a linear function.

We set server power capacity of the stable grid energy sources (P_{grid}), such that grid capacity is sufficient to support both non-IT power load and half of the peak IT power load.

$$P_{grid} = P_{S_{peak}}/2$$

We set power capacity from the renewable sources ($P_{rnw}(t)$) as a two level power signal, where once sufficient capacity is available, renewable sources are able to provide power to support peak server load.

$$P_{rnw}(t) = \begin{cases} P_{S_{peak}}/2, & \text{t = High renewables} \\ 0, & \text{t = Low renewables} \end{cases}$$

With that, we define our dynamic server capacity scaling model ($P_{cap}(t)$) as,

$$P_{cap}(t) = P_{grid} + P_{rnw}(t)$$

WAN latency performance model: We model WAN latency performance based on the traffic management characteristics of WANs connecting multi-region cloud deployments [11]. In that, communication latency across WAN is unpredictable, thus changes over-time. With that, we model WAN latency performance as follows.

For the duration of ΔT, the set of executed low-latency VMs (S_{VM}) is,

$$S_{VM} = \{VM_1, ..., VM_m\}$$

For S_{VM}, we calculate WAN latency performance ($Lat(S_{VM})$) as,

$$Lat(S_{VM}) = \frac{1}{\sum_i^m \int_t^{\Delta T} L_i(t)\, dt}$$

where $L_i(t)$ is the end-user latency of the i^{th} VM at time t.

Problem formulation: We formulate our problem as follows. Given an arbitrary time period ΔT, maximize end-user latency performance of VMs while meeting the dynamic capacity scaling of servers.

$$\text{Maximize} \quad Lat(S_{VM}) \quad \text{and} \quad \forall S \in \text{Servers}, P_S(t) \leq P_{cap}(t)$$

5.5 Load Shifting for Low-latency Applications with SMT Core Pooling

In this section, we detail the design of our load shifting technique. Our technique addresses the research questions identified in Section 5.3 to accommodate low-latency applications in cloud load shifting. In Section 5.7 we showcase its superiority in delivering improved latency performance for low-latency applications.

Figure 5.4 illustrates the system architecture of the hardware-software co-design of the proposed technique. At the hardware level, we leverage heterogeneous server pools of SMT cores to maintain a static CPU capacity amidst renewable supply dynamics. We employ core-level power management to meet the dynamic resource scaling of servers for renewable supply variations. In return, we maintain a static set of logical cores cores across the server pools. We then exploit the static set at the software layer. Using a novel VM scheduling algorithm, we efficiently manage low-latency application VMs for the static set, lowering their possibilities of shifting over the WAN. Our approach involve three steps: 1) Core-level power management to meet server resource scaling, 2) SMT pooling to yield static resource availability, and 3) novel VM scheduling algorithm to utilize SMT pooling for low-latency VMs. Collectively, we maximize retaining low-latency VMs within the local region by exploiting the hardware multi-threading of SMT.

5.5.1 Hardware-level logical core management

Server power management: To cater to dynamic server resource scaling for the supply dynamics of renewable energy, we employ per-core deep idling [126] to manage the

server power draw. At t = Low Renewables, we set half of the server cores into the deep sleep, such that the peak power draw of the server meets $P_{cap}(t)$.

$$P_S(t = \text{Low Renewables}) \leq f(\sum_{i=1}^{N/2} P_{CORE_{peak}})$$

where $P_{CORE_{peak}}$ is the peak power draw of each homogeneous core. Opposed to that, we awake all cores to better utilize $P_{cap}(t)$ at t = High Renewables.

$$P_S(t = \text{High Renewables}) \leq f(\sum_{i=1}^{N} P_{CORE_{peak}})$$

We then leverage CPU Simultaneous Multi-threading (SMT) to exploit our server power management to yield a static set of CPU resources.

SMT pooling: Simultaneous Multi-threading (SMT) in CPUs doubles the available number of logical cores [127] while providing better performance due to its hardware multi-threading nature. To exploit our server power management with SMT, we maintain two server pools where the servers in one of the pools enable SMT (i.e., SMT pool). The number of logical cores in servers of the SMT pool is double the amount of its physical CPU cores. In combination with our server power management, which deep idles half of the physical cores at renewable supply valleys, we realize a static set of logical cores across the server pools, regardless of the renewable supply state. At supply valleys, servers in the SMT pool provide the static set of logical cores, whereas, at supply peaks, servers in the non-SMT pool provide that instead. An example of that is illustrated in Figure 5.4. In that example, N is set to 4. At supply peaks, the non-SMT pool exhibits four logical cores, and the SMT pool exhibits eight logical cores. At supply valleys, the SMT pool exhibits four logical cores. Collectively, a static set of four logical cores is present in the local region at all times, which is a significant advantage over traditional server power management, where the renewable energy integration often down scale resource capacity of each server.

The primary reason for not enabling SMT in both pools is that core oversubscription methods such as SMT can still impact the low-latency performance of VMs [29]. In our system architecture, the static set of logical cores only rely on SMT cores during energy supply valleys, thus the latency performance impact is further reduced. In the next

Algorithm 4 Proposed VM Scheduling Algorithm for SMT Pooling

Input: v: Incoming VM request,
S_{smt}: Set of servers in the SMT pool,
$S_{\text{non-smt}}$: Set of servers in the non-SMT pool,
$P(t) \in \{peak, valley\}$: Renewable supply state at time t,
$P(t^-) \in \{peak, valley\}$: Renewable supply state at time t^-
Output: Scheduling decisions for incoming VM placement and server pool management

```
1: function HANDLEVMPLACEMENT(v, P(t))
2:     if ISLOWLATENCY(v) then
3:         if P(t) = peak then
4:             PLACE(v, S_non-smt)
5:         else
6:             NOTADMIT(v)
7:     else
8:         if P(t) = peak then
9:             PLACE(v, S_smt)
10:        else
11:            PLACE(v, S_smt ∪ S_non-smt)
12: function HANDLEPOWERTRANSITION(P(t⁻), P(t))
13:    if P(t⁻) = peak and P(t) = valley then
14:        OFFLOADBESTEFFORTVMS(S_smt)
15:        LIVEMIGRATE(v ∈ low-latency, S_non-smt, S_smt)
16:    else if P(t⁻) = valley and P(t) = peak then
17:        OFFLOADBESTEFFORTVMS(S_non-smt)
18:        LIVEMIGRATE(v ∈ low-latency, S_smt, S_non-smt)
19: while SYSTEMISRUNNING do
20:    if ISPOWERTRANSITION(t⁻, t) then
21:        HANDLEPOWERTRANSITION(P(t⁻), P(t))
22:    for all v arriving do
23:        HANDLEVMPLACEMENT(v, P(t))
```

subsection, we efficiently utilize the static set of logical cores for low-latency applications via a novel VM scheduling algorithm.

5.5.2 Software-level VM scheduling algorithm

In this section, we propose a software-level VM scheduling algorithm to utilize the hardware-level logical cores provided by SMT pooling. Algorithm 4 outlines our VM

scheduling algorithm. It takes incoming VM placement requests and the state of the renewable energy supply (i.e., peak or valley) as inputs. It then determines VM placement decisions and handles server pool management for renewable energy dynamics.

The pseudo-code for VM placement is outlined in the subroutine HANDLEVMPLACEMENT (line 1). It first identifies the criticality of the VM as either low-latency or best-effort (line 2). For low-latency VM placement requests at peak renewable energy supply, requests are admitted to the cloud region for placement and deployed in the non-SMT pool. Since CPU cores of the non-SMT pool do not have the performance overhead of having hardware multi-threading, the placement decision aims for maximum CPU performance. Conversely, requests will not be admitted to the cloud region if the renewable energy supply is at a valley (line 5). At that stage, available cores in the non-SMT pool are reserved to restore the performance of already deployed low-latency VMs at renewable energy peaks. We provide inner details of the server pool management in the next paragraph. In the case of best-effort VM placement requests (line 7), we admit and deploy those regardless of the renewable energy supply state, with the exception of limiting their deployment to the SMT pool during supply peaks (line 9).

The pseudo-code for server pool management is outlined in the subroutine HANDLEPOWERTRANSITION (line 12). It takes two input parameters: the state of the renewable energy supply at times t and immediately before (t^-). In case of a transition from supply peak to a valley, we first offload best-effort VMs in the SMT pool from the cloud region (line 14) and internally live migrate low-latency VMs to the SMT pool from the non-SMT pool. Here, offloaded VMs are handled similarly to standard load shifting across cloud regions, which places those VMs outside the local cloud region. Our approach exploits the best-effort nature of VMs to relax constraints in offloading. In the case of a transition from supply valley to a peak, we first offload best-effort VMs from the non-SMT pool and live migrate VMs from the SMT pool to the non-SMT pool (line 16). Overall, through live migrations, server pool management maintains deployed low-latency VMs inside the data center. Migrations cause minimum disruption to the application execution, and most importantly, end-user latency performance is kept intact by keeping VMs in the same cloud region throughout. We make room for the low-latency VMs via offloading best-effort VMs, aiming for a minimum impact from geographical

load shifting towards the application service quality.

Our algorithm's collective management of VM placement and server pool management is designed to improve the service quality of the cloud region's already admitted low-latency VMs. It improves the utilization of SMT pooling and the server power management's approach of halving the number of available physical cores, guaranteeing a fixed resource capacity for low-latency VMs through their live migrations and offloading best-effort VMs.

5.6 Implementation

We implement our proposed technique in a real experimental cloud environment. We implement core-level power management of servers via the CPU idle states feature [128] and SMT pooling using the same feature present in server CPUs [129], both for Intel CPUs. We implement our proposed VM scheduling algorithm at the cloud resource management layer using OpenStack [118].

First, we implement core-level power management of the server by deploying a daemon service in each server [130]. The daemon service wraps the Intel power optimization library that provides low-level API control of the CPU C-States in each CPU core. We then expose high-level RESTful APIs to the deep idle half of the CPU cores. As a result, upon receiving a core deep idle request, the daemon service overrides the default kernel's behavior of CPU idle states and maintains half of the CPU cores at the deepest C-State. Conversely, a wake request will restore the deep idle state. We write the daemon service in golang. Secondly, we select the set of servers for the SMT pool and enable CPU hyper-threading through their BIOS settings. Finally, we deploy OpenStack to manage the servers. We modify its default server filter in the VM scheduling workflow to omit specific server pools for certain VM types (see Algorithm 4). We assume VM placement requests that are unsuccessful in finding a scheduling decision will be offloaded from the cloud region. Further, we implement a Golang controller in the OpenStack control plane for server pool management. It is driven by the events from the renewable energy supply, for which we expose a high-level API. Upon triggering, it conducts live migrations between servers and emulates VM offloading by conducting

VM evictions through OpenStack APIs (see Algorithm 4).

5.7 Performance Evaluation

In this section, we evaluate the performance of our proposed technique. We outline our experimental design and setup, compare our results with the state-of-the-art baseline, and analyze them in detail.

5.7.1 Experimental design and setup

We conduct evaluation experiments in a real prototype multi-node cloud region. We model the multi-region characteristics for our experiments using real measured data from production cloud regions [11]. In our prototype cloud, we allocate an identical HP ProLiant server to each SMT and non-SMT pool. Each server has an Intel Xeon CPU with 12 physical cores. For the SMT pool, we enable Intel Hyper-threading through its bios settings. Both servers are part of a research server cluster in a private network and share a fast network fabric. We install OpenStack in both servers, marking one as the control plane. We deploy our Golang daemon service on both servers and the Golang controller on the server marked for OpenStack's control plane.

Baselines:

We compare our proposed technique with state-of-the-art ***Space-Shifting***. Space-shifting is the predominantly used load shifting technique [96, 122]. It aims to manage the cloud region's power draw by shifting its flexible workloads across space to other cloud regions based on energy availability. We use space-shifting to compare the superiority of our proposed technique in accommodating low-latency workloads.

Workloads:

We use Microsoft Azure's VM packing data [8] to expose our testbed to realistic VM requests in production clouds. We scale Azure's data to match our experimental de-

ployment by sampling and synthesizing VM arrival traces using the data. For renewable energy supply dynamics, we use solar dynamics data from the ELIA dataset [116]. Further, we use real latency variation data measured for cloud regions [11] in our experimental setup to evaluate VM latency impact from shifting.

Metrics:

To measure the latency performance, we use two latency metrics. We use ***application latency performance*** to measure the performance impact inside the low-latency VM. We use ***end-user latency*** to measure the impact of shifting VMs across the WAN. To measure application latency performance, rather than measuring the latency performance of specific cloud applications, we monitor the latency performance of the VM's guest operating system. For that, we use the cyclictest tool [114]. For end-user latency, we use WAN latency data. For that, we measure latency for the duration of the VM's lifetime provided by the trace data. Using a statistical model we build from WAN latency data, we sample a latency value for each time step and get the aggregated value. To measure the scheduling performance, we monitor the number of ***Retained***, ***Offloaded***, and ***Not Admitted*** VMs in the cloud region. Retained VMs complete their lifetime inside the cloud region, offloaded VMs are interrupted at mid-life to get shifted over WAN, and not-admitted VMs are diverted to a different cloud region at admittance.

5.7.2 Results and Analysis

We evaluate the performance of our proposed technique using a two-fold approach. Firstly, we evaluate the performance impact of executing VMs on the heterogeneous server pools. Secondly, we evaluate the performance impact of various placement decisions made due to the load-shifting approach.

Application latency performance with SMT pooling: Due to SMT pooling, VMs in our technique can be deployed on either SMT or non-SMT cores. To evaluate application latency performance, we measure the latency performance of the guest operating system for various core allocation configurations. Since SMT cores share CPU components, our experiment is designed to measure the performance impact as resource contention in

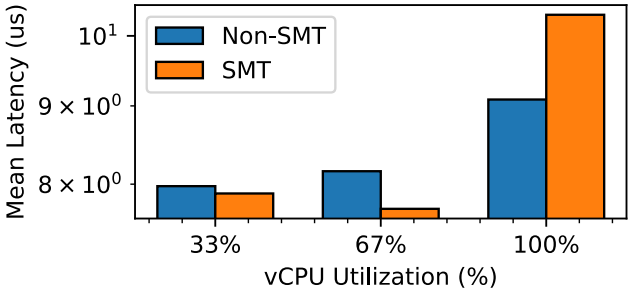
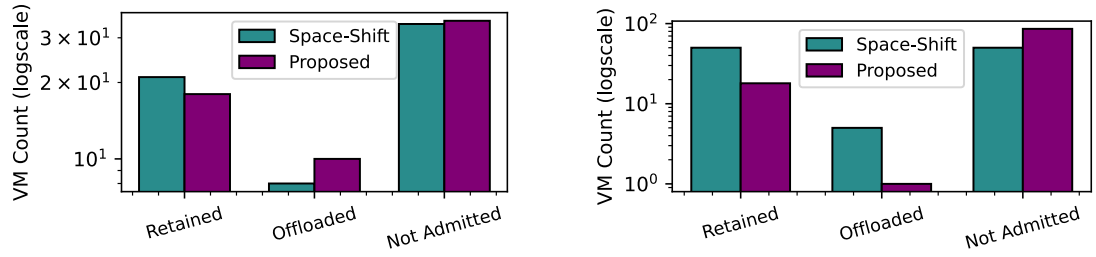


Figure 5.5: Comparison of mean application latency performance of heterogeneous server pools as resource contention increases.

the CPU increases. We first allocate VMs to occupy a portion of the CPU cores available in a server of each pool. Then, we execute a system load in the VM to emulate a utilized application and measure the latency performance of the guest operating system. We then repeat the experiment for different allocated CPU core portions until it reaches 100%. Our experiment measures the low-latency application impact of deploying a VM in a server for different packing levels across the server pools. Figure 5.5 illustrates the results. It shows the mean latency value measured for the guest operating system over the utilization of the server's virtual CPU cores (vCPU) across both SMT and non-SMT pools.

The results show that the impact of application latency from deploying a VM on a specific server pool significantly depends on CPU utilization. For both 33% and 67% of the vCPU utilization, the mean latency performance of VMs on SMT and Non-SMT pools is around 8 microseconds. However, as utilization increases to 100%, there is a significant increase in the VM mean latency, where packing on SMT cores increases from 8 to 10 microseconds. Packing on non-SMT cores increases from 8 to only about 9 microseconds. In comparison, packing on SMT cores increases the mean latency by 11.97%. In this context, the application latency impact of placing a VM across heterogeneous server pools depends on the service quality of the application. For instance, server pooling will primarily impact an application sensitive to 1-microsecond performance degradation. In contrast, low-latency applications that can tolerate larger latency penalties will not exhibit degradation in their service quality.



(a) Comparison for Best-effort VMs.

(b) Comparison for Low-latency VMs.

Figure 5.6: Comparison of VM scheduling performance in the experimental cloud region.

Scheduling impact of low-latency VMs: In this experiment, we measure the scheduling impact of retaining deployed VMs inside the cloud region during its lifetime, offloading VMs in mid-life to other cloud regions, and VM admittance to the cloud region. We replay VM request arrivals of the Azure trace and 24-hour renewable supply change dynamics. We define a threshold in the renewable energy supply to define peaks and valleys. Afterward, we conduct the same experiment for our proposed technique and the baseline. Figure 5.6a and 5.6b illustrates the results for each best-effort and low-latency VM types. It shows the number of VMs in each category of the x-axis.

The results show that the proposed technique significantly surpasses Space-Shift in reduced offloaded events for low-latency VMs with an 80% reduction. In comparison, it increases the best-effort VM offloading. In admitting VM requests for deployment, the proposed technique is closely the same, with a slightly decreasing number of not-admitting events for both best-effort and low-latency VM types. The Space-Shift approach performs better in retaining both low-latency and best-effort VM types. Collectively, the behavior of the proposed technique shows its superiority in managing low-latency VMs. This is because prior studies show that even with relatively degraded performance, users favor a deterministic application performance [101]. In that regard, offloading events incur the most disruption to low-latency applications since offloading can introduce latency overheads from the wide area network's performance. The proposed technique shows the minimum offloading events and compromises in not admitting VMs. Although not admitting does shift the VM across WAN, end-user service

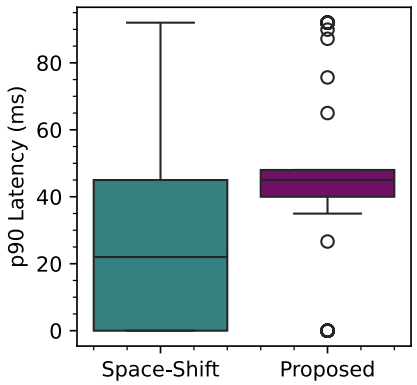


Figure 5.7: Comparison of p90 end-user latency distribution of Low-latency VMs.

quality would indicate the degraded performance since deployment rather than showing mid-life.

End-user latency impact of low-latency VMs: We measure the impact of end-user latency in our scheduling impact experiments to evaluate the impact of shifting VMs across WAN. Using probabilistic models, we generate data from real inter-cloud-region latency over WAN. We sample a distribution of latency values for each VM for the lifetime and calculate its p90 value. WAN latency does not impact retained VMs, which complete their entire lifetime inside the local cloud region. However, WAN latency does impact the not-admitted VMs and has a partial impact on offloaded VMs. Figure 5.7 illustrates the results. It compares the distributions of p90 latencies of VMs.

The results show the superiority of the proposed technique in reducing the p90 latency variance. Although there are several outliers, the latency performance of VMs with the proposed technique primarily concentrates around 40 milliseconds with a coefficient of variation of 0.62. In contrast, with Space-Shift, the p90 latency value disperses around 20 milliseconds with a coefficient of variation of 1.10. The proposed technique reduces the coefficient of variation of p90 latency by 43.81%. Therefore, the results indicate that VMs managed with the Space-Shift technique will most likely deliver unpredictable end-user performance compared to the proposed approach, which is unfavorable for the service quality [101].

5.8 Summary

In this chapter we explored how maintaining static resources amidst renewable energy supply dynamics enable accommodating temporal inflexible low-latency applications for multi-region renewables harvesting. To this end we proposed a technique to employ a hardware-software co-design for maintaining a static set of logical cores in CPUs. Firstly, at the hardware level, we conducted core-level server power management by deep idling and awakening half of the physical CPU cores to match renewable valleys and peaks, respectively. We employed two server pools, with one enabling Simultaneous Multi-threading (SMT) in the CPU, doubling the available logic cores. As a result, we realize a static set of logical cores that is always present in one of the SMT or non-SMT server pools. Secondly, at the software level, we leveraged a novel VM scheduling algorithm to efficiently utilize the static logical core set for low-latency application VMs. We implemented our technique with OpenStack for Inter CPUs and evaluated using Azure VM traces on a practical two-node prototype. As evidenced by our results, the proposed technique surpassed state-of-the-art baseline with an 80% reduction in offloading low-latency VMs, 43.81% reduction in coefficient of variation of end-user latency, and worst-case latency compromise of 11.97% due to SMT cores.

So far, thesis chapters explored techniques to optimize the operational carbon footprint of prominent latency-sensitive cloud computing environments. In the next chapter, we study how to optimize the infrastructure embodied carbon footprint. In particular, we focus on rapidly growing Large Language Model (LLM) inference clusters. LLM inference clusters exhibit significant embodied carbon accumulation rates due to their shorter hardware refresh lifecycle. We explore opportunities to extend the amortization of procured embodied carbon in LLM inference clusters by slowing the aging effects of their hardware components through efficient resource management.

Chapter 6

Embodied Carbon Amortization for Low-latency LLM Inference Clusters

Broad adoption of Large Language Models (LLM) demands rapid expansions of cloud LLM inference clusters, leading to the accumulation of embodied carbon that mostly concentrates on the inference server CPU. In this chapter, we study the amortization of CPU embodied carbon in inference clusters and propose an aging-aware CPU core management technique to further amortize embodied carbon over an extended lifespan. We uncover CPU underutilization patterns in LLM inference and exploit those using core deep idling to slow down CPU aging effects. We conduct extensive simulation-based experiments using real-world trace data with an extended simulator from a public cloud provider. When compared with state-of-the-art baselines, our results demonstrate an estimated 37.67% reduction in yearly embodied carbon emissions through p99 performance of managing CPU aging effects, a 77% reduction in CPU underutilization, and less than 10% impact on the inference service quality.

6.1 Introduction

The proliferation of applications driven by cloud-based generative Large Language Model (LLM) inference is seen across diverse domains, such as conversational agents [16], education [18], and coding assistance [131]. As the popularity of such applications scales their user base to billions [132], cloud service providers continue to expand LLM in-

This chapter is derived from:

- **Tharindu B. Hewage**, Shashikant Ilager, Maria Rodriguez Read, and Rajkumar Buyya, "Aging-aware CPU Core Management for Embodied Carbon Amortization in Cloud LLM Inference", *Proceedings of the 16th ACM International Conference on Future and Sustainable Energy Systems (E-ENERGY)*, Rotterdam, Netherlands, June 17-20, 2025 [Accepted].

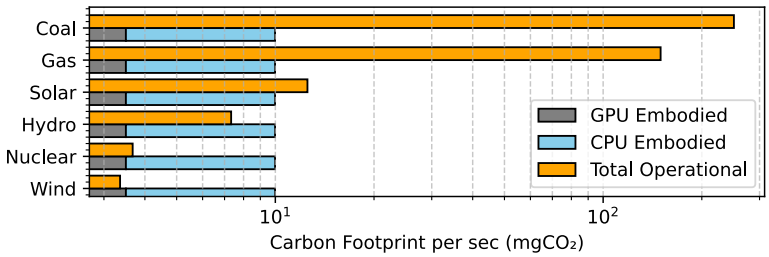


Figure 6.1: Carbon footprint of A100x4 GPU server running per second inference application when powered by energy sources with different carbon intensity [12].

ference clusters towards the GigaWatt scale to support the growing demand [133]. Recently, Meta announced its plans to build a brand new data center targeting AI workloads [134], and xAI plans to expand their AI cluster from 100K to 1 million GPUs [135].

LLM Inference clusters deploy and serve pre-trained LLMs [14]. In inference, a user request (i.e. prompt query) is split into a set of input tokens. Input tokens are then fed to the model (i.e. forward pass) to generate the first output token and an intermediate context called KV-cache. KV-cache and the first token are then fed for the second forward pass, and the process is repeated until a stopping condition is met [14]. In return, a single request can incur many forward passes. To reduce the latency in that, clusters employ parallel model computation via GPU accelerators [136]. Due to memory constraints, each server typically utilizes several GPUs to support modern LLMs with billions of parameters [137]. In return, LLM inference becomes both memory and compute-intensive [138]. When serving LLMs at scale, inference clusters employ many inference optimization techniques to better utilize underlying resources, such as phase splitting [14] and iteration-level scheduling [139]. As a result, LLM inference at scale results in a complex set of CPU tasks (i.e. inference tasks), such as facilitating steps of optimization techniques [14, 139], request scheduling [138], and tokenization [12].

Growth of LLM inference clusters also increases cloud infrastructure carbon footprint [15, 133, 134]. It combines two aspects: direct carbon emissions owing to energy sources (i.e. operational), and indirect carbon emissions results from business activities such as manufacturing, shipping and recycling IT assets (i.e. embodied) [21, 22]. Today, as cloud service providers invest in renewable energy generation to meet net-zero emission goals [134, 140], renewable energy sources with lesser carbon intensity [141]

continue to penetrate power grids [15], diminishing the effect of operational carbon over the embodied. Microsoft, a hyper-scale cloud provider reported that over the past four years, its operational carbon was reduced by 6.3 percent while embodied increased by 30.9 percent [23]. In LLM inference clusters, most of its embodied carbon accounts for CPU components, including the die and mainboard [12]. Therefore, optimizing CPU embodied becomes paramount for sustainable growth of LLM inference clusters. Figure 6.1 illustrates that. With lesser carbon intensive renewable energy sources, CPU embodied becomes the dominant carbon aspect in inference servers.

We study the problem of optimizing CPU embodied in LLM inference clusters. Inference clusters amortize CPU embodied over it's lifetime. Therefore, extending CPU life further amortize its embodied carbon. CPU life extensions are typically achieved through extending its hardware refresh cycle [69]. CPU hardware refresh cycle replaces CPUs with newer hardware generations. Its aim is to gain performance-per-watt improvements [69] and avoid reliability risks of silicon aging [69, 142, 143]. However, CPU performance gains in that are minimal for inference clusters. This is because CPU tasks in inference clusters carry-out GPU-accelerated LLM inference and these inference tasks mostly benefits from single core performance, which has plateaued in recent years [144]. As a result, the sole aim from maintaining a standard hardware refresh cycle is to avoid silicon aging. In this context, extending CPU hardware refresh cycle requires efficient management of CPU to delay silicon aging effects. It is worth noting that this is not the case for GPU, for which the embodied carbon footprint is smaller and performance gains of newer hardware generation is significant [12]. Many works exploring silicon aging management in CPU employ efficient aging-aware workload management [67, 142, 145, 146]. They leverage task scheduling among CPU cores to even-out core aging and in return, slow down the aging rate of the overall CPU [67, 142, 143, 145]. However, leveraging the opportunities present in CPU usage patterns of cloud LLM inference is yet to be explored.

To this end, we propose an aging-aware CPU core management technique to extend the CPU life in inference clusters. In return, cluster embodied carbon is further amortized over the increased lifespan. We design our technique for CPU usage patterns in cloud llm inference. Using production inference traces, we uncover that LLM inference

clusters mostly underutilize CPU cores with occasional usage bursts. To exploit that, we design a dynamic **working set** of cores where the cores in the set remain active while others deep idle [128]. We then design online algorithms that (1) identify and adjust the *working set* based on usage bursts, and (2) assign inference tasks inside the *working set* to even-out aging across cores. Collectively, our approach achieves age-halting and reduced underutilization of cores. The *working set* however, can lead inference tasks to oversubscribe the CPU if not scaled in-time. The online algorithms we propose are also designed to mitigate that.

We implement our approach by extending splitwise-sim, a high-fidelity LLM cluster simulator from Microsoft [14]. We use production LLM inference traces generated with data collected from LLM inference services in Azure [14] and use state-of-the-art CPU core management techniques as baselines. Results for our experimental cluster show; estimated 37.67% reduction in yearly embodied carbon emissions through p99 performance of managing CPU aging effects and reduction of CPU core underutilization by 77%, all while maintaining CPU oversubscription below 10%. The **key contributions** of our work are as follows:

1. An investigation into the role of the CPU in state-of-art LLM inference clusters and uncovering CPU underutilization patterns using production traces.

2. A new technique for age-aware CPU core management using dynamic age-halting of deep idling CPU cores is proposed.

3. Implement the proposed technique in a simulated environment and conduct extensive experiments using production inference traces.

4. An evaluation of our proposed technique against state-of-the-art CPU core management baselines, focusing on its efficiency in managing CPU core aging, reducing yearly embodied carbon, and controlling task-related CPU oversubscription.

The rest of the chapter is organized as follows. In Section 6.2 we discuss the related literature. Section 6.3 provides background and motivation. Section 6.4 provides the system model and its essential components, and the problem formulation. In Section 6.5 we present our proposed aging-aware CPU core management technique. Section

Table 6.1: Comparison of relevant works with our proposed technique for embodied carbon optimization through CPU aging management.

Work	Even-Out Core Aging	Process Variation Aware	Avoid CPU Profiling	Dynamic Age-halting
Facelift08 [143]	✓	✓		
Hyat15 [142]	✓	✓		
Tamer'21 [147]	✓			
Shoulao23 [145]	✓	✓		
Zhao23 [67]	✓		✓	
Our Proposed	✓	✓	✓	✓

6.6 outline our implementation in the simulated environment and implication on implementing our proposed technique in practice. Section 6.7 describes the performance evaluation and experimental results. Finally, Section 6.8 summarises the chapter.

6.2 Related Work

The environmental impact of embodied carbon in growing LLM inference clusters has caught attention in recent years [12, 15, 140, 148, 149]. As an early research area, these works model embodied carbon in LLM inference and advocate for potential directions to reduce that [12, 22, 149]. Further, some outline CPU GPU asymmetric optimization opportunities of heterogeneous energy, performance, and inference application patterns [12, 22], and accounting carbon footprint for a given inference request on specific hardware settings [150]. In contrast, to actively optimize embodied carbon, some works propose system-level techniques. These include controlling LLM token generation [151] and disaggregation of specific compute onto older hardware [152]. Building on studies of embodied carbon optimization in CPU GPU asymmetric lifetimes, we explore system-level solutions for fine-grain CPU-aging management by leveraging request-level patterns in cloud LLM inference clusters.

A plethora of works investigate mitigating CPU aging effects through workload management [67, 142, 143, 145, 147]. Their predominant approach is even-outing tasks across the cores to reduce uneven aging. These include utilizing CPU profiling [142,

143, 145, 147], and addressing manufacturing process variations in CPU [142, 143, 145]. However, not many consider the efficacy of the proposed techniques in cloud settings. For example, conducting CPU profiling in clouds with large server fleets is difficult. Nevertheless, few recent works consider cloud-efficient techniques, such as workload management at the resource management level to reduce severe exercising of specific cores [67]. In addition to even-outing aging, age halting is an efficient approach to slow down CPU aging. Age halting has been used in the literature to leverage dark silicon in CPU for age management [142]. However, their age-halting is static since the age-halting adjustments are only done after a relatively longer epoch. In contrast, we study age management for cloud LLM inference with even-outing aging and dynamic age-halting [153].

6.3 Background and Motivation

In this section, we provide background on embodied carbon amortization in cloud servers and optimizing it by extending CPU life. We then outline our investigations on applying that in cloud LLM inference clusters. We highlight key takeaways from where we draw our motivations for this chapter.

6.3.1 Background: Embodied Carbon Amortization in Cloud Servers through CPU Age Management

In managing the carbon footprint of cloud data centers, Green House Gas (GHG) protocol, a global standard formed to manage GHG emissions [32] defines three scopes. Scopes 1 and 2 represent the operational carbon, typically owing to the carbon intensity of the data center's energy sources. Scope 3 represents embodied carbon: carbon emissions that indirectly result from manufacturing and shipping of servers and other IT assets that have already been built and installed in data center [21]. Unlike operational carbon, which can be optimized by adopting less carbon-intensive energy sources, embodied carbon needs to be amortized over the asset's lifespan. Here, amortization is a way to account for embodied carbon. For example, if a server with a 4-year operational

lifetime causes 1000 $kgCO_2eq$ of Scope 3 emissions, then amortization accounts for a 250 $kgCO_2eq$ of embodied carbon emissions per year.

Recent studies of embodied carbon optimization outline three tenets of environmental design: reduce, reuse, and recycle [69]. Out of that, this chapter focuses on recycling, more specifically enabling a second life of the CPU by improving its reliability to extend the lifetime [69]. Primary reason for CPU reliability degradation is the silicon aging of its transistors beyond the rated life [143]. Many works studying silicon aging in CPU [142, 143, 147] model that with **Negative Bias Temperature Instability (NBTI)**. NBTI is an aging mechanism that affects PMOS transistors in CPU [146]. It is caused by the stress of workload execution. During workload execution, transistors in the CPU continue to switch, applying stress on transistors and releasing them back. When stress is applied, NBTI shifts the transistor's threshold voltage (ΔV_{th}) but leaves a residual shift when the stress is removed. That incurs a slight increase in the ΔV_{th}, accumulating over time. As a result, critical path delay in the circuit increases, reducing the maximum operating frequency of the CPU (i.e., **CPU Aging**). Since NBTI aging results from workload execution, workload management techniques can mitigate that for improved CPU reliability [142, 143, 145, 147]. In return, embodied carbon is further amortized through a second life of the CPU.

6.3.2 Motivation: Impact of CPUs on Embodied Carbon in LLM Inference Clusters

Servers in LLM inference clusters serve generative AI requests via low-latency model computation through GPU accelerators. Therefore, most of the inference server's thermal design power (TDP) comes from the GPU. In return, GPU dominates the server's operational carbon footprint [12]. Nevertheless, the electrical grids powering inference clusters continue to integrate low-emission renewable energy sources [93]. As a result, the operational carbon intensity of inference servers continue to diminish, whilst embodied carbon accounting for the majority of the server's carbon footprint (see Figure 6.1).

The embodied carbon of an inference server consists of two categories: CPU embodied and GPU embodied. GPU embodied is attributed to a combination of components

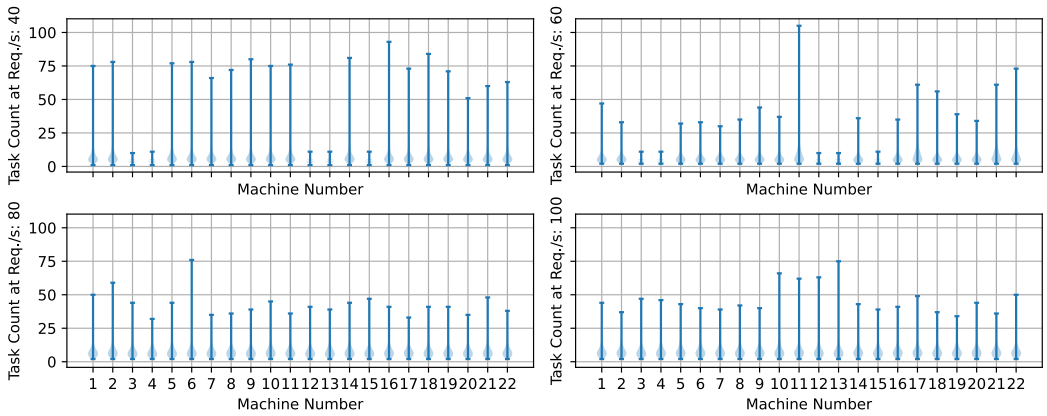


Figure 6.2: Distributions of running inference tasks in an LLM inference cluster of 22 H100 machines.

such as SoC, PCB, Heatsink, etc [12]. GPU is packaged into an independent hardware unit and installed using a standard interface such as PCIe. In return, it is loosely coupled from the server and straightforward to replace [154]. In contrast, the CPU is installed as a tightly coupled component with many other associated components, such as the CPU chassis, mainboard, cooling, etc [12]. In between, CPU components have strict compatibility requirements, making them increasingly complex to replace individually. For instance, the CPU die must be compatible with the mainboard's socket type and its chipset generation [155]. In cloud platforms, replacing individual CPU components has become increasingly unsustainable. Firstly, maintaining a continuous supply of spare components with fine-grained compatibility requirements is difficult to procure due to the poor availability of repair parts [156]. Secondly, recent technology trends in data centers, such as utilizing liquid cooling, significantly increase the time and effort required to repair servers at the component level [157]. Therefore, upon a CPU component failure, cloud providers often replace the CPU and the associated components as a whole [158]. In this context, the CPU embodied of an inference server is attributed to the embodied carbon footprint of the CPU and its associated components [12].

Refined carbon modeling studies in LLM inference clusters show a substantial impact from CPU embodied towards cluster's embodied carbon footprint [12]. In an Azure T4 inference server, GPU embodied calculates to only 41.8 kgCO2eq emissions for its total lifetime. However, CPU embodied accounts for 278.3 kgCO2eq emissions, whilst

54% of that attributes to CPU die, CPU chassis, and the mainboard alone [12]. Therefore, cloud providers are challenged with lowering the rate of accumulation of CPU embodied and achieving extended amortization of procured CPU embodied.

Inference clusters acquire CPU embodied faster than it can amortize through its hardware refresh life cycle. A hardware refresh cycle aims to gain performance improvements of newer CPU hardware generations and avoid reliability concerns of aging CPU [69]. Recent studies show that LLM inference clusters may not gain significant performance benefits from newer CPU hardware generations. CPUs in inference clusters execute tasks (i.e., inference tasks) facilitating the inference workflows, such as phase splitting, request scheduling, batching, and tokenization [12, 14]. These typically benefit from the single-core performance, yet the yearly single-core performance of newer CPU hardware generations has been mostly the same [144]. That leaves avoiding reliability concerns of CPU aging as the sole benefit of the CPU hardware refresh cycle.

It's important to note that the reliability concerns of aging are significant in the CPU, whereas its associated components, such as the mainboard, pose minimal risks. For instance, long-term failure analysis of servers shows mainboard failure rates have dropped over time [158]. Further, many associated components, such as storage and memory, already implement error correction mechanisms for aging hardware, such as Error Correction Codes (ECC) [159]. In contrast, erroneous computations of unreliable aging CPUs are extremely difficult to correct through preventive mechanisms [159] and have become increasingly frequent in hyper-scale cluster deployments [159, 160].

As discussed in Section 6.3.1, an efficient aging-aware workload management technique can mitigate CPU reliability concerns, which also translates to an extension to the CPU hardware refresh cycle, allowing the cluster to further amortize its CPU embodied.

Takeaways: *Extended amortization of CPU embodied significantly reduces the carbon footprint of inference clusters. However, it is constrained by the reliability concerns of CPU aging. Hence, there is an opportunity for an effective aging-aware CPU management technique to optimize that.*

CPU Utilization Patterns in LLM Inference Clusters: To design an efficient aging-aware CPU management technique, it is important to understand CPU utilization patterns in cloud LLM inference. For that, we monitor and analyze CPU utilization patterns

in a high-fidelity simulated LLM inference cluster environment that infer real workload traces.

We use a LLM cluster simulator from Microsoft's [14] and extend it to model CPU in cloud LLM inference. We then replay production inference traces from Azure and observe cluster CPU utilization. In that, we allocate each CPU task to a dedicated core. Our cluster settings closely match that in production. In Section 6.6, We discuss our experimental setup in detail. Figure 6.2 illustrates our results. Each subplot maps to a different throughput level and shows the distribution of concurrent inference tasks executed in each cluster machine. The x-axis denotes the machine number, and the y-axis denotes the inference task count. We make two key observations in that.

- ***O1:*** *Cores are mostly underutilized, as indicated by the lower mean values in the violin plots.*

- ***O2:*** *There are occasional bursts of running tasks as indicated by the maximum values of the violin plots, which justifies having CPUs with higher core counts.*

In our simulation, our key finding is that underutilized CPU cores are available to the machine operating system to schedule system tasks apart from the inference serving platform. Therefore, these cores can actively execute system tasks in a time-shared manner [153]. In return, all cores can actively execute instructions regardless of being allocated to an inference task, thus continuing to age due to the transistor stress of workload execution. In this context, we identify an opportunity to halt aging in the underutilized cores. We ***hypothesize to reduce the available cores to match the number of running tasks of the inference platform***. In return, we can put the remaining cores to deep idle, which turns off the clock and power gate the CPU cores [153, 161], stopping the transistor switching and halting the core aging. However, doing so can introduce a new set of challenges. As shown in Figure 6.2, the number of concurrent inference tasks can dynamically change. Therefore, limiting the available cores can lead inference tasks to oversubscribe the CPU unless the number of available cores is scaled in time. Moreover, a reduced set of available cores can introduce core affinity, which can increase failure risks of individual CPU cores due to uneven core aging [67].

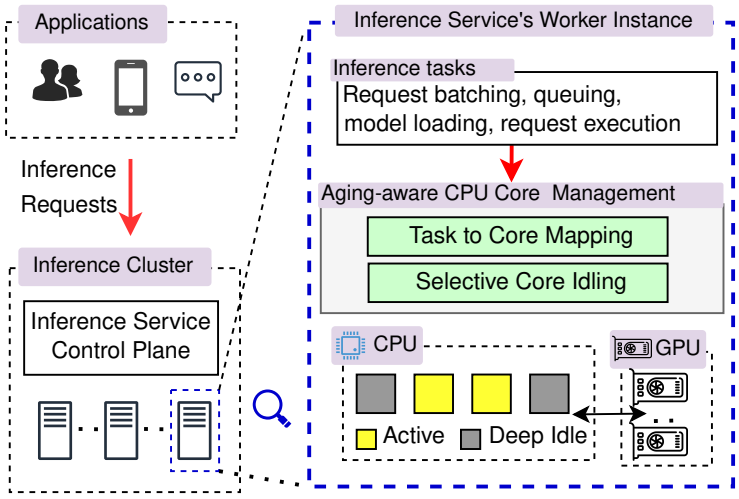


Figure 6.3: High-Level system diagram of aging-aware CPU core management in LLM inference clusters.

Takeaways: *Underutilized cores in cloud LLM inference provide the opportunity to halt CPU aging through deep idling unused cores. However, it presents new challenges, including timely switching of core idle states to reduce CPU oversubscription, and efficient use of the available cores to avoid uneven core aging.*

6.4 System Model and Problem Formulation

This section presents the system model, its components, and the formulation of the problem.

6.4.1 System Model

Figure 6.3 provides a high-level view of our system model. We model a high-performance LLM inference cluster deployment with inference-optimized servers. Our deployment matches similar production cluster deployments where cluster resources, such as servers with GPU accelerators, are designed for LLM inference [14]. Firstly, the inference requests from end-user applications reach the cluster's inference service. Each request is then scheduled to servers by the cluster-level scheduler. Inference servers run virtualized worker instances, which conduct request batching, queuing, model loading,

and finally execute the request leveraging an inference backend, such as vLLM [162]. Inference backend efficiently utilizes CPU and GPU resources and may leverage high-bandwidth InfiniBand interconnections between GPUs, such as sharing intermediate KV-cache in phase splitting [14]. Our system architecture matches that of production deployments, such as the NVIDIA Triton server inference architecture on Kubernetes [162].

In return, the server-level worker instance of the inference service executes many CPU tasks (i.e., inference tasks). For each inference task, we allocate a dedicated CPU core. If the cores are insufficient, we assume that inference tasks oversubscribe the CPU. We introduce a new component to conduct aging-aware CPU core management. It oversees assigning cores to inference tasks and controlling the idle states of CPU cores. A CPU core in our system model can switch between either active or deep idle states [128]. Being in the active state gradually ages the CPU cores. In contrast, deep idling halts cores from aging. However, cores that deep idle become unavailable for inference task execution. Cores in the active state are available for task execution, yet allocating a task accelerates core aging. In Section 6.4.2, we provide in-depth details about the silicon aging behavior with our aging model. CPU cores in the host are isolated and pinned to worker instance CPU cores, such that task mapping and idle state changes directly reflect on the physical core. Overall, the combination of inference task execution and deep idling control the CPU aging rate, where a lower value further amortizes its embodied carbon through the increased lifespan [67].

6.4.2 Aging Model

We model aging of CPU cores due to execution of system and inference tasks. In that regard, Negative-bias Temperature Instability (NBTI) is a major aging mechanism [142, 146] for CPU. In this work, we model NBTI-induced core aging. Similar to previous works [146], we use a reaction-diffusion based aging model to calculate CPU core frequency degradation due to NBTI-induced aging.

$$f(t) = f_0 \times (1 - \frac{\Delta V_{th}}{V_{dd} - V_{th}}) \tag{6.1}$$

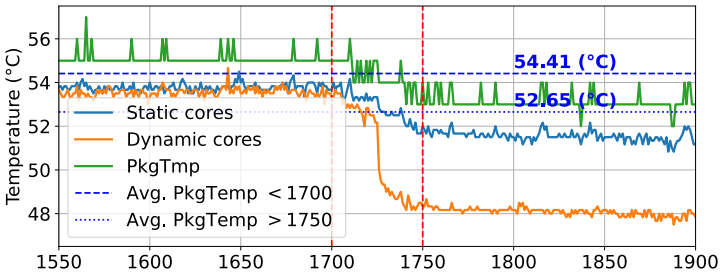


Figure 6.4: Changes in Operating temperature when 6 out of 12 cores set to deep idle in an Intel Xeon CPU. If awake, cores are 100% utilized.

where $f(t)$ is the frequency at time t and f_0 is the initial frequency of the core. Previous works show that f_0 can deviate from the nominal value due to variations in the manufacturing process [142, 163]. To accommodate that, we use the following model to calculate f_0.

$$f_0 = K' \min_{k,l \in S_{CP}} \left(\frac{1}{p_{kl}} \right).$$

where K' is a technology-dependent constant, and S_{CP} represents the sections of the core containing critical paths. In order to calculate f_0, we first divide the chip area into an $N_{\text{chip}} \times N_{\text{chip}}$ grid and assume that critical paths are contained entirely within the grid cells. Then, we assign each grid cell with a gaussian random variable (p_{kl}). In order to calculate the spatial correlation between the random variables, we use the following formula [163].

$$\rho_{ij,kl} = e^{-\alpha \sqrt{(i-k)^2+(j-l)^2}} \quad \forall\, i,j,k,l \in [1, N_{\text{chip}}].$$

where α decides how quickly spatial correlations die out. For our experiments, we set N_{chip} to 10 and K' to 1. Then, we set the mean of random variables by solving for a scenario where if a core does not exhibit process variation, f_0 should equal the nominal value. We set the remaining parameters similar to the calculation of a previous work [163].

After f_0, we calculate ΔV_{th} in the Equation 6.1, which is the shift in the threshold voltage. CPU cores in our system can undergo time intervals in different idle states. In order to calculate ΔV_{th} in those, we calculate ΔV_{th} using the following recursive equation

Table 6.2: Temperature modelling for different core states.

Idle-state	**C-state [128]**	**Inference Task**	**Temperature** ($^{\circ}C$)
Active	C0	Allocated	54
Active	C0	Unallocated	51.08
Deep Idle	C6	N/A	48

[164].

$$\Delta V_{\text{th}}(t_p) = ADF_p \left[\left(\frac{\Delta V_{\text{th}}(t_{p-1})}{ADF_p} \right)^{\frac{1}{n}} + \tau_p \right]^n$$

where $V_{\text{th}}(t_p)$ is the value at p^{th} time interval, and τ_p is the length of the p^{th} time interval. ADF is a time-independent factor for each time interval, which we calculate using the following equation.

$$ADF(T, V_{\text{dd}}, Y) = K \cdot \exp\Big(-\frac{E_0}{k_B T}\Big) \cdot \exp\Big(\frac{B\, V_{\text{dd}}}{t_{\text{ox}}\, k_B T}\Big) \cdot Y^n \tag{6.2}$$

where Y is the stress from the executing task. We assume each task in our system incur the worst case by setting it to 1.0. T is the operating temperature of the CPU core. In order to create a realistic temperature model, we conduct an experiment by running a high utilization task in a server-grade CPU and switching its cores between active and deep idle states. During the experiment, we monitor the changes in core temperatures. Figure 6.4 illustrates our observations. Table 6.2 denotes the temperature model we derive from that. We set the rest of the parameters in equation 6.2 as follows. K is a fitting parameter. To calculate its value, we use CPU aging data from a previous work, which states that for 22nm CPU technology, the worst-case frequency reduction due to aging for a lifetime of 10-years can reach 30% [146]. We set values of our model to match this scenario and solve the ΔV_{th} equation to find the value for K. All parameters that were not explicitly mentioned are set similar to a previous work matching for the 22nm CPU technology [146].

6.4.3 Embodied Carbon Model

Our carbon modeling is based on our findings in the section 6.3.2. We model the embodied carbon of the CPU and its associated components in LLM inference servers to accurately measure the carbon amortization optimizations of our proposed technique. We use recent fine-grained carbon modeling [12], encompassing different configurations and components of multi-GPU inference servers, such as the peripheral components for cooling and power delivery. With that, the yearly CPU embodied carbon footprint of an LLM inference server ($CF^{\text{emb,cpu}}$) is modeled as,

$$CF^{\text{emb,cpu}} = \frac{1}{LT}\left(N_r K_r + \sum_k CF_k\right)$$

where $k \in \{\text{CPU Die, CPU Chassis, Mainboard, DRAM, PDN, etc.}\}$, CF_k is the carbon footprint of associated component k, LT is the lifetime in years, and $N_r K_r$ is the packing carbon footprint. Inner details of CF_k calculations are based on a recent prior work [12].

In section 6.7, we use our yearly CPU embodied carbon model to quantify the resulting carbon savings of the proposed technique. In that, we use concrete values of CPU embodied carbon footprint calculated using our carbon model for the Azure T4 inference server with 16GB GDDR6 denoted by the VM flavor Standard_NC4as_T4_v3 [12, 165].

6.4.4 Problem Formulation

This chapter considers a cluster of LLM inference servers managed by an inference service. It serves inference requests arriving from cloud users.

At the server level, the GPU-accelerated inference requests results in a dynamic number of concurrent CPU tasks (i.e., inference tasks). They are handled by a multi-core CPU. Due to manufacturing process variations, each CPU core exhibits a maximum frequency value that deviates from the nominal value, degrading over time with workload execution due to core aging. Depending on the task allocation and management of core idle states, the rate of frequency degradation among the cores can differ. Over time, the multi-core CPU exhibits a distribution of degraded frequencies among its cores, increas-

ing their failure risks [67]. In addition, the service quality of serving inference tasks can be impacted if active cores are insufficient to facilitate the running inference tasks. In this context, we formulate our problem as follows.

Our multi-core CPU contains an N number of CPU cores. We denote the number of deep idling cores at time t with $N_{idle}(t)$, the number of executing tasks with $T(t)$, and the frequency of the core i with $f_{core_i}(t)$. For the duration of ΔT, the following equation calculates the reduction of core frequency due to core aging ($f_{red_i}(\Delta T)$) for the i^{th} core.

$$f_{red_i}(\Delta T) = f_{\text{core},i}(t) - f_{\text{core},i}(t + \Delta T) \quad i \in N$$

Similarly, the following equation calculates the variance in the CPU core frequency distribution ($f_{\text{var}}(\Delta T)$).

$$f_{\text{var}}(\Delta T) = \text{Var}(F) \quad \text{where } F = \left\{ f_{\text{core},i}\big(t + \Delta T\big) : i \in N \right\}$$

Finally, the following equation quantifies service quality impact from inference serving due to core deep idling ($T_{oversub}(\Delta T)$).

$$T_{oversub}(\Delta T) = \int_{t}^{t+\Delta T} u(T(t) - (N - N_{idle}(t))) \times (T(t) - (N - N_{idle}(t)))$$

Where u is the unit step function. The goal of our problem is to extend amortizing CPU embodied carbon through increasing its operating lifespan by mitigating CPU aging effects. For that, both per-core and uneven aging effects across cores must be reduced. Moreover, the impact on the service quality of inference tasks must also be reduced. With that, we state our problem,

$$\begin{aligned} \text{Min.} \quad & f_{red_i}(\Delta T) \quad \forall i \\ \text{Min.} \quad & f_{\text{var}}(\Delta T) \\ \text{Min.} \quad & T_{oversub}(\Delta T) \end{aligned}$$

Algorithm 5 Proposed Task-to-Core Mapping Algorithm.

Input:
cpu_cores: *working set* of cores.
`task`: Inference task assigned (or `None` if no task)
`last_idle_durations`: Recent idle durations.
Output: The selected core to run the inference task.

```
selected_core ← None
selected_idle_score ← 0.0
for all core in cpu_cores do
    if core.task ≠ None then
        continue
    idle_score ← ∑(core.last_idle_durations)
    if (selected_core = None) or (idle_score > selected_idle_score) then
        selected_core ← core
        selected_idle_score ← idle_score
return selected_core
```

6.5 Embodied Carbon Amortization through CPU Aging Management

In this section, we provide the design and inner workings of our proposed aging-aware CPU core management technique. Our design is based on our findings in Section 6.3.2. We evaluate the performance of the proposed technique in Section 6.7, showcasing its potential to reduce yearly embodied carbon emissions of inference clusters.

Figure 6.3 illustrates the design of the proposed technique. We optimize CPU core aging at the server level to extend CPU lifetime, matching the CPU GPU asymmetric lifetime. To implement our proposed *aging-aware core management*, we introduce two main mechanisms: *(1) Task to Core Mapping*, and *(2) Selective Core Idling*. *Task to Core Mapping* runs an algorithm to decide the mapping of each inference task to a CPU core. It aims to mitigate uneven aging among available CPU cores. Whereas *Selective Core Idling* runs an algorithm to determine a *working set* of cores to match current inference throughput. It halts aging of cores in the non-working set by setting them to deep idle. Apart from age halting, it further complements uneven core aging by selecting cores to deep idle in an aging-aware manner.

Together, the proposed technique reduces overall aging rate of the CPU in two aspects. Both *Task-to-Core Mapping* and *Selective Core Idling* **even-out aging** across cores,

preventing early effects of premature aging in specific cores. *Selective Core Idling* **halts aging** in cores, when the inference throughput provide opportunities to do so. It delays aging effects in cores.

6.5.1 Task-to-Core Mapping

The primary goal of the *Task-to-Core Mapping* is to reduce the age variance among CPU cores. To achieve that, it distributes the stress of inference tasks favoring lesser-aged cores. As a result, older cores age slower, delaying overall aging effects.

Algorithm 5 outlines the proposed algorithm for *Task-to-Core Mapping*. It takes the set of active cores (i.e., *working set*) as the input and selects a core to run an inference task. Therefore, each new inference task executes the algorithm 5 once. To reduce the execution time in that, we design the algorithm to leverage an age estimation approach for its selection logic, rather than obtaining CPU micro-architectural attributes to calculate an accurate value. To achieve that, each core in the input *working set* provides two additional attributes: task assigned status, and its idle history. Using a core's idle history, we calculate an estimation for it's age. We maintain a core's last eight idle durations, similar to that of the Linux governor algorithm [128]. At execution, we create placeholders for both the selected core and its idle score (line 1). Then, we iteratively evaluate each core in the *working set*. We calculate an idle score for each core that has not been assigned a task yet (line 6). Idle score accumulate all idle durations in the provided history. Here, the insight is that if a core mostly remained idle, its aging rate is lower than that of a less idle core. We then use the idle score to conduct a relative comparison among the cores to filter the core with the most idle score (line 7). As a result, the core with the least aging estimation is selected to execute the next inference task.

Overall, the algorithm estimates the age of each core using a rolling idle duration window and distributes the stress of executing inference tasks in a least-aged-first manner. In return, the aging effects of cores take a prolonged time to appear, i.e., slowing down the CPU aging rate. Our age estimation approach avoid the overhead in calculating an accurate aging value. Since *Task-to-Core Mapping* is executed quite frequently in the cloud environments, a minimum execution overhead ensures reduced latency im-

Algorithm 6 Proposed Selective Core Idling Algorithm.

Input:
cpu_cores: List of available cores,
oversub_tasks: Number of CPU oversubscribing tasks,
Output: Adjusted core idle states: deep idle or active.

$N \leftarrow get\_total\_core\_count(\text{cpu\_cores})$
$\text{active\_cores} \leftarrow get\_active\_core\_count(\text{cpu\_cores})$
$\text{normal\_tasks} \leftarrow get\_assigned\_task\_count(\text{cpu\_cores})$
$C_{SLP_t} \leftarrow N - \text{active\_cores}$
$T_t \leftarrow \text{normal\_tasks} + \text{oversub\_tasks}$
$T_t \leftarrow \min(N, T_t)$
$e_t \leftarrow \left(N - C_{SLP_t} - T_t\right)$
$e_{t\_prd} \leftarrow e_t$
$e_{t\_prd} \leftarrow \frac{e_{t\_prd}}{N}$
if $e_{t\_prd} \geq 0$ **then**
    $F(e_{t\_prd}) \leftarrow \tan\left(0.785 \cdot e_{t\_prd}\right)$
else
    $F(e_{t\_prd}) \leftarrow \arctan\left(1.55 \cdot e_{t\_prd}\right)$
$e_{t\_corr} \leftarrow N \times F(e_{t\_prd})$
$e_{t\_corr} \leftarrow \text{int}(e_{t\_corr})$
$\delta_{\text{cores}} \leftarrow |e_{t\_corr}|$
if $e_{t\_corr} > 0$ **then**
    $put\_cores\_idle(\delta_{\text{cores}}, \text{cpu\_cores})$
else if $e_{t\_corr} < 0$ **then**
    $put\_cores\_active(\delta_{\text{cores}}, \text{cpu\_cores})$

pact on inference request serving.

6.5.2 Selective Core Idling

In addition to the aging rate reduction of *Task-to-Core Mapping*, we provide a core-level optimization mechanism to halt aging, called *Selective Core Idling*. It contributes to the overall aging reduction in the CPU by dynamically halting core aging, whenever the inference task execution is able to tolerate that.

The main idea behind *Selective Core Idling* is to leverage the unused CPU cores in cloud LLM inference for deep idling (see Section 6.3.2). We do that by dynamically adjusting the size of the *working set* of cores, and using the remaining cores for deep idling. The key challenge here is to match the size of the *working set* to the number

of running inference tasks. If the *working set* is smaller then inference tasks begin to oversubscribe the CPU, whereas a larger *working set* leave a portion of unused cores in the active state which otherwise would have been utilized for deep idling. To address that, we design an algorithm for *Selective Core Idling*. The main part of our algorithm is a module called a reaction function. The reaction function decides the algorithm's sensitivity on adjusting the size of the *working set*. We periodically execute the *Selective Core Idling* algorithm to adjust the *working set* to match the inference throughput.

Algorithm 6 outlines the proposed algorithm for *Selective Core Idling*. It takes two inputs: the set of available cores, and the number of inference tasks that are oversubscribing the CPU. Once executed, it adjusts the *working set* by selectively setting the idle states of available cores to either deep idle or active. Firstly, the algorithm process the set of available cores to obtain the number of total cores, number of cores that are in the active state, and the number of inference tasks that are allocated with a dedicated core (line 1). Using them, the algorithm calculates the number of cores that are currently deep idling (line 4), as well as the total number of tasks. Here, we cap the total number of tasks at the total number of CPU cores (line 6). This is done to obtain a normalized error term (e_{t_prd}), which we are calculating next (line 9). The error term indicates the severity of CPU oversubscription of the inference tasks. We then use the error term as the input to the part of our algorithm which carry out the reaction function (line 10 to line 13). The output of the reaction function is then scaled back (line 14) and used to determine the cores to set either active or deep idle. When putting cores to deep idle, we do that in the order of most aged first. When putting cores to active, we do it in the order of least aged first. That way, we complement even-out core aging of *Task to Core Mapping* when deep idling the cores.

Reaction Function: The reaction function is designed to be independent from the number of CPU cores. It takes the normalized error term (e_{t_prd}) in Algorithm 6 (line 9) as the input. It returns a normalized output between -1 and $+1$. Here a positive output indicates CPU underutilization, requiring to set cores from active to deep idle. Whereas a negative output indicates CPU oversubscription, requiring to set cores from deep idle to active. In the inference cluster, CPU oversubscription impact inference latency of user requests, which is a short term effect requiring immediate action. Whereas CPU un-

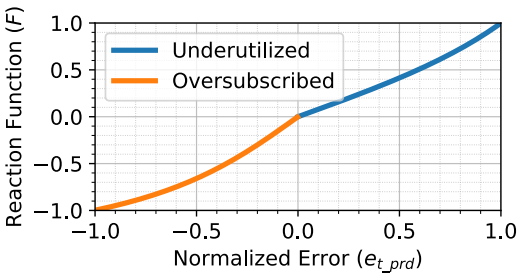


Figure 6.5: Behavior of the piecewise Reaction Function (F) for utilization of the CPU.

derutilization leads to aging of cores, which is a long-term effect since aging is a slow process. To balance this trade-off, we design the reaction function to react slower for CPU underutilization and faster for the CPU oversubscription. Figure 6.5 illustrates the behavior of our reaction function. Algorithm 6 denotes the equation and the values we used for that (line 10 to line 13).

6.6 Implementation

We implement our proposed technique in a simulated environment. We use splitwise-sim [14], an event-driven, high-fidelity LLM cluster simulator from Microsoft. It employs Splitwise, a state-of-the-art phase splitting LLM serving technique [14]. Compared to vanilla request level scheduling, Splitwise increases CPU load due to the facilitation of additional tasks of phase splitting. As a result, our simulation environment enables exposing our technique to realistic CPU stress levels present in state-of-the-art cloud LLM inference clusters.

First, we extend the simulator to model the inference tasks that run on the CPU. Table 6.3 outlines the tasks we modeled. We model the CPU load of the executor component that facilitates the inference workflow, worker instance tasks that handle memory and iterative-level scheduling, and the tasks of interconnects. For that, we merge each class function with APIs of a new processor subclass we implemented for the CPU. Inside the CPU class, we manage the state of CPU cores. Using the models we outlined in Section 6.4, we maintain core temperatures, idle states, and the shift in threshold voltage. Each

Table 6.3: Tasks modeled as inference tasks in the extended splitwise-sim [14] simulator.

Task Name	Class/Function
finish_flow	Executor.finish_flow
finish_request	Executor.finish_request
finish_task	Executor.finish_task
submit	Executor.submit
submit_chain	Executor.submit_chain
submit_flow	Executor.submit_flow
submit_task	Executor.submit_task
alloc_memory	Instance.alloc_memory
free_memory	Instance.free_memory
start_iteration	ORCAInstance.start_iteration
flow_completion	Link.flow_completion

class function call outlined in Table 6.3 invokes `assign_core_to_cpu_task` API of the CPU class. It then provide inputs and invoke the Algorithm 5. In return, we determined a CPU core to cater the function call. We update the idle states and temperature of the core to match the task execution. Further, we update the shift in the threshold voltage and the resulting operating frequency of the core. The execution time of the calling function in the simulator is then adjusted according to the operating frequency. In parallel to that, we periodically invoke the `adjust_sleeping_cores` API of the CPU class to conduct *Selective Core Idling*. It retrieves the system state and execute the Algorithm 6. In return, the algorithm set the idle state of a number of CPU cores to either active or deep idle. Since the periodic execution of the algorithm 6 does not add an overhead to inference request latency, we use it as an opportunity to accurately calculate degraded core frequency due to aging. We assume that data is provided by the core-level aging sensors [142] with an additional overhead.

In practice, the performances of implementations of our proposed technique could depend on the underlying CPU hardware. For instance, the selective core idling in our

proposed technique is supported across CPU vendors via the CPU idle states feature [166]. Since their specific hardware implementations could differ [153, 167], CPU age halting would be more effective depending on the number of CPU components get deactivated at the deepest sleep state [153]. In return, deeper sleep states introduce longer transition latencies [166]. Although those typically stay at the microsecond-scale [153], depending on the inference task throughput, CPU oversubscription may increase due to longer wake times of idle cores. Practical implementations could also inherit certain operational latency overheads. Although selective core idling and task-to-core mapping mechanisms do not incur coordination latency in between due to their independent execution, they can exhibit latency overhead within their workflows. For instance, the implementation of the task-to-core mapping mechanism could introduce a latency overhead in algorithm execution. We reduce the impact of that through low-complexity algorithm design. Implementation of selective core idling could also introduce a communication latency overhead when retrieving data from core-level aging sensors. We reduce the impact of that by executing its algorithm periodically rather than executing it for each inference task scheduling event.

6.7 Performance Evaluation

In this section, we evaluate the performance of our proposed CPU core management for amortizing embodied carbon in LLM clusters. We provide our experimental design and setup, and compare our results with state-of-the-art baselines and analyze them in detail.

6.7.1 Experiment Design and Setup

We conduct evaluation experiments in the simulated environment that we modeled and implemented with an LLM cluster simulator from Microsoft. We describe inner details of our implementation in Section 6.6. We model a cluster of 22 GPU-optimized Nvidia H100 machines with 5 prompt instances and 17 token instances of phase splitting [14]. Our cluster design is an iso-throughput, power-optimized cluster design for cloud LLM

inference [14]. We use it to create a realistic cloud inference cluster. Each server in the cluster runs a worker instance. For the worker instance CPU counts, we use CPU core counts of 40 and 80 to match public VM offerings in Azure for Nvidia H100 machines [168].

Baselines:

As discussed in section 6.3.1, our problem context requires extended embodied carbon amortization through an efficient CPU age management technique. However, not many works focus on CPU age management in the cloud settings for LLM inference. In that regard, we use two state-of-the-art baselines. We use ***linux*** to compare the performance of our technique with state-of-the-art inference serving systems [14], and we use ***least-aged*** to compare the performance of our technique with state-of-the-art CPU age management techniques designed for cloud settings that cater CPU tasks similar to inference tasks used in our system model. We further detail our baselines as follows:
linux: It represents executing the servers with the task to CPU core allocation of Linux LLM inference servers. To implement that, we use CPU data from a LLM inference server, captured while executing inference requests [169]. Using the data, we build a probabilistic model to generate inference task to CPU core mappings.
least-aged *[67]:* It is an aging-aware task-serving idea proposed for cloud servers. Unlike most works, *least-aged* proposes the idea of assigning the tasks away from aged cores using executed work as an aging estimate, without requiring frequent CPU profiling. Although *least-aged* was designed for cloud CPU tasks in general, the task characteristics it uses for aging apply to the inference tasks that we model.

Workloads:

We use LLM inference traces generated by Microsoft using real Azure inference data [14]. Each request in the trace is characterized with the number of input tokens and the number of output tokes generated. It does not provide the actual query that was used in the public cloud environment due to privacy requirements. For our performance metrics that we outline next, the actual query does not make an impact, rather the execution

times resulted from processing input and output tokens.

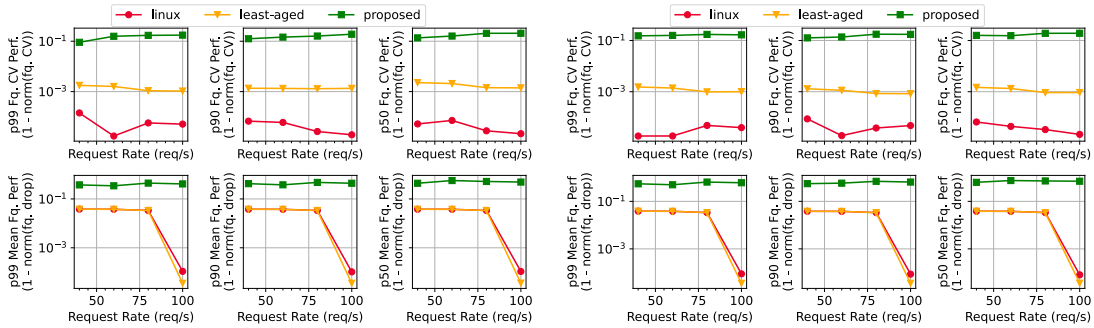
Metrics:

To measure the CPU aging effects, we use the coefficient of variation (CV) of the distribution of frequencies among CPU cores in each inference server after the experiment. We then calculate the percentile values of that across the cluster. The resulting **frequency CVs** reflect how well the technique could even out the aging effects among the cores in the cluster machines. To measure the application impact, we calculate the distribution of the number of idle CPU cores in inference servers during the experiments. The resulting data reflect the impact of **CPU oversubscription** across the cluster servers.

6.7.2 Results and Analysis

We carry out the performance evaluation as follows. We sample a set of initial core frequencies for each inference server CPU according to the process variation model described in Section 6.4.2. We then replay LLM inference traces on the cluster. We conduct repeated experiments for different throughput levels of LLM inference traces, for each baseline, and for our proposed technique. At the end of the experiments, we calculate the degradation of initial CPU core frequencies through our metrics to evaluate how each baseline and our proposed technique managed cores across the cluster to reduce CPU aging effects and to minimize the application impact. We further estimate the reduction of yearly embodied carbon emissions of the cluster, resulting from the CPU aging management.

Management of core aging effects: Based on our aging model described in Section 6.4.2, the initial frequency of each CPU core deviates from the nominal value due to process variation. Due to the execution of inference tasks, initial frequencies degrade over time, resulting a frequency distribution across CPU cores. Figure 6.6 illustrates the results of that. Its subplot, figure 6.6a illustrates the performance of managing the coefficient of variation (CV) of the core frequency distribution. The performance value decreases when frequency CV increases, and vice versa. It also illustrates the performance of managing the mean frequency degradation. The performance value in that decreases when



(a) Comparison of 40 VM cores

(b) Comparison of 80 VM cores

Figure 6.6: Comparison of managing aging effects in CPU.

the mean frequency degradation increases, and vice versa. Both performances are for the VM with 40 cores. Figure 6.6b illustrates the same for the VM core count of 80. All plots share the x axis, which is the throughput of inference traces.

The results show that in both VM core types, the reduction of frequency variance among cores with *least-aged* is better than the *linux*. It shows the effectiveness of assigning inference tasks away from the aged cores to improve aging imbalance across cores in *least-aged*. However, the proposed technique significantly outperforms both baselines in that. It showcase the superiority of core age even-out behavior across the both *Task to core mapping* and *Selective Core Idling* mechanisms in the proposed technique. The results of managing mean frequency degradation performance shows that both baselines exhibit quite similar performances. In both Figure 6.6a and 6.6b, frequency performance values for baselines show the same for request rates from 40 to 80, and deviates slights at request rate of 100. Whereas our proposed technique consistently surpasses both across all evaluated request rates. It shows the effectiveness of the age halting behavior of the proposed technique. Frequency performance, which is the aging effect of our aging model, shows sustained performance further in-time than the baselines. The discussed performance patterns are consistent across changing VM core sizes.

Reduction of yearly CPU-embodied carbon emissions: Delayed CPU aging effects allow cloud operators to extend CPU lifespan by increasing the hardware refresh lifecycle. We apply the same with our results of managing core aging effects. We take the hardware refresh cycle of a typical linux-based LLM inference server as 3 years [12] and its

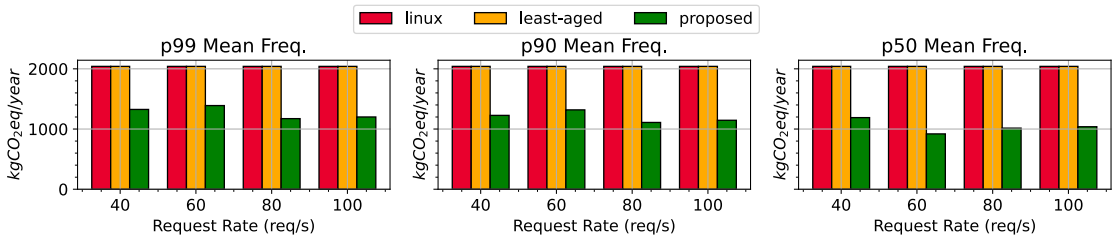
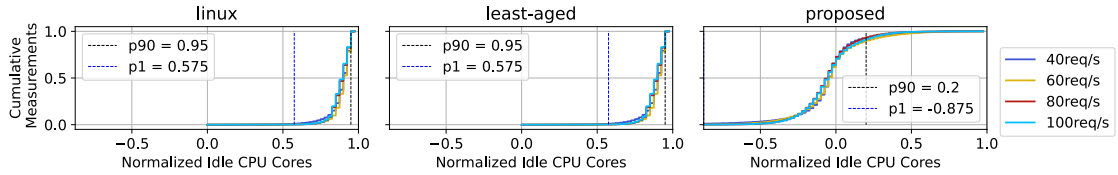


Figure 6.7: Comparison of estimated yearly CPU embodied carbon reduction in the cluster through management of CPU aging effects.

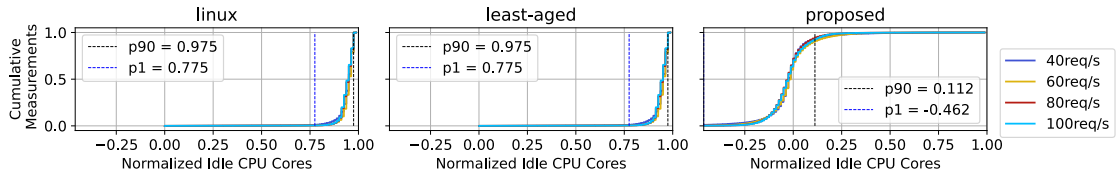
CPU embodied carbon during this lifespan as 278.3 $kgCO_2eq$ [12]. We then compare the reduction of the mean core frequency of other techniques to *linux* and estimate an increase in lifecycle extension using a linear model. Using the carbon and lifespan expansion data, we calculate yearly embodied carbon emissions for baselines and the proposed technique. Figure 6.7 presents our results. It shows yearly CPU-embodied carbon emissions of the cluster for the age management performance of different throughput levels.

The results show that yearly CPU-embodied carbon savings with the *least-aged* is minimal when compared to *linux*. This is due to the similar performance of the mean frequency degradation that we discussed previously. In contrast, a cluster managed with our proposed technique shows significant carbon savings in CPU embodied. When estimated with p99 mean frequency performance, our proposed method reduces yearly CPU embodied emissions in our experimental inference cluster by 37.67%. It further increases to 49.01% for p50 mean frequency performance. The observed results are due to superiority of mean frequency performance of our proposed technique. It showcase achieving CPU embodied carbon reduction through effective age management. Its important to note that advantage for carbon reductions with *least-aged* over *linux* may improve with the experiment duration. However, goal of our experiments is to evaluate advantage of our proposed technique, which we show within the evaluated experiments.

Application impact of aging-aware core management: Figure 6.8 illustrates the results of idle cores availability in the cluster servers during inference task execution. The x-axis in all figures shows normalized idle CPU cores, in which a positive value indicates core underutilization and a negative value indicates core oversubscription, whereas, y-axis



(a) Comparison for 40 VM cores



(b) Comparison for 80 VM cores

Figure 6.8: Comparison of utilization of available cores for running tasks. X-axis denotes normalized idle CPU cores (a negative indicates CPU oversubscription and a positive indicates CPU underutilization).

to denote the measurement distribution.

The results show that both baselines do not incur core oversubscription but underutilize cores, yielding positive values of idle CPU cores. p1 to p90 percentiles in both baselines reside closer to 1.0, with a higher VM count increasing the closeness. In contrast, the proposed technique outperforms both baselines in CPU underutilization. Its p90 percentile is at least 77.8% better in both VM core counts. However, its p1 percentile being negative indicates that the proposed technique does result in CPU oversubscription. The severity of that improves with the VM core count, showing a smaller P1 value. We observe consistent idle core distributions across different inference throughout rates. Additionally, results show that p1 of the proposed technique is at least less than -0.1, which means the proposed technique maintain the CPU oversubscription below 10%.

In summary, we observe core aging even-out behavior of our proposed technique surpassing baselines in reducing frequency CVs. Alongside, age halting behavior in our technique showcases its superiority in delaying mean frequency degradation. Both frequency CV and mean frequency performance are metrics of CPU aging effects in our system model. We then estimate yearly CPU embodied emissions in the experimental inference cluster, based on the mean frequency performance. Results highlight efficacy of our proposed method to reduce CPU embodied through managing its aging effects.

In return, our proposed method show CPU oversubscription, which can impact service quality of the inference tasks. Yet, results show that our proposed technique is able to maintain its severity.

6.8 Summary

In this chapter, we proposed an aging-aware CPU core management technique, showcasing its potential to extend cloud LLM inference cluster embodied carbon amortization through increasing CPU lifespan. Exploiting CPU underutilization patterns that we uncovered, the proposed technique not only even-out silicon aging across cores but also harnesses the opportunities of age halting using core deep idling. Our empirical simulations demonstrated the superiority of the proposed technique over existing methods with an estimated 37.67% reduction in yearly embodied carbon emissions through p99 performance of managing CPU aging effects, a 77% reduction in CPU underutilization, and less than 10% impact to the inference service quality. Our technique enables LLM inference to reduce embodied carbon through the CPU and improve performance with the GPU via the CPU GPU's asymmetric lifetime.

Chapter 7
Conclusions and Future Directions

This chapter concludes the thesis by summarizing its works and key contributions. Further, it highlights key future directions to continue advancing carbon efficiency in latency-sensitive cloud computing environments.

7.1 Summary of Contributions

The deeper integration of cloud computing with modern society has led to the adoption of cloud-based serving of latency-sensitive services. Today, various paradigms, such as the Internet of Things (IoT), leverage cloud-based deployments to deliver low-latency application response times. Further, time-bounded real-time applications of Industry 4.0, transport, and healthcare have started their transition to the cloud. Recently, the low-latency application use cases of the Artificial Intelligence (AI) boom heavily rely on the cloud for their deployments. Most of these latency-sensitive use cases rely on the cloud due to their cost-effectiveness in delivering on-demand, managed infrastructure wrapped in a pay-as-you-go model. As a result, latency-sensitive cloud computing today creates significant business value for hyper-scale cloud providers. In delivering latency service level objectives (SLO), cloud providers maintain heterogeneous infrastructures for latency-sensitive use cases that are quite different from traditional centralized hyper-scale data centers. A key characteristic in such deployments is that their application and hardware layers are often tightly coupled and often applications-specific. Such an intertwined nature of the application and the hardware layers provide lesser control in managing application performance and hardware lifecycle. Unlike traditional clouds which often have segregated application and hardware layers, these infrastructures pro-

vide limited opportunities for the cloud provider to manage its resources efficiently.

As a result, achieving carbon efficiency in latency-sensitive cloud computing environments has been increasingly difficult. Today, the majority of the focus on carbon-aware resource management in clouds favors traditional hyper-scale cloud infrastructure. Application management in those environments provides much flexibility in compromising the performance over carbon efficiency. Carbon optimization in clouds involves both operational and embodied carbon optimizations. Operational carbon is commonly optimized through the integration of variable-available renewable energy sources, in which application performance compromises to account for energy supply intermittency. Embodied carbon is typically optimized through efficient management of the hardware lifecycle, such as component re-purposing and recycling, in which the cloud provider's control over the hardware lifecycle is paramount. Therefore, the sustainable growth of latency-sensitive cloud computing environments requires uncovering opportunities among the tight coupling of application and hardware layers in application-specific cloud deployments. In this thesis, we investigated dynamic approaches for handling several key resource management processes, including scheduling, provisioning, power delivery, and hardware management, to the satisfaction of the application latency performance and carbon efficiency of both operational and embodied aspects.

Chapter 1 introduced the increasingly prevailing latency-sensitive cloud computing environments and their unique resource management challenges due to the limited provider control. Next, the adverse impact of those challenges on the infrastructure carbon footprint was presented, discussing the detrimental impact of climate impact from cloud carbon emissions and highlighting the challenges related to achieving carbon efficiency in latency-sensitive cloud computing environments. Further, it presented the identified research questions and summarized the thesis contributions.

Chapter 2 investigated the existing carbon-aware resource management techniques in latency-sensitive cloud computing environments, encompassing both operational and embodied carbon optimizations. Next, a detailed taxonomy of the resource management aspects for both operational and embodied carbon optimizations was presented. Further, relevant recent literature was reviewed according to the taxonomy, and com-

prehensive discussions on the identified key research gaps were presented.

Chapter 3 proposed an approach to realize a dynamic power budget and a decentralized task scheduling algorithm for power-constrained distributed Micro-Cloud deployments using on-site renewable energy integration. The proposed dynamic power budget locally relaxes Micro-Cloud power constraints using the provisioning flexibility of on-site renewable energy sources, without introducing significant hardware changes through the existing underutilization opportunities present in the power delivery system. The proposed decentralized task scheduling algorithm utilizes the dynamic power budget across the multi-region Micro-Clouds to deliver jointly optimized relaxation of power constraints and application low-latency performances.

Chapter 4 proposed a framework to integrate renewable energy with real-time cloud systems without compromising its deterministic nature of application execution. The two-fold framework presented in this work utilized renewable-driven cores as a power management technique over renewable energy dynamics and implemented a VM execution model to ensure deterministic application execution exploiting fault tolerance provided in real-time cloud systems. Next, it introduced the concept of Green Cores to translate energy dynamics into packing inventory attributes and leveraged that to propose an efficient server-level VM packing algorithm to jointly optimize for renewable energy harvesting and to reduce impact from fault-tolerance exploitation. This work also entailed an open-sourced implementation of the proposed framework.

Chapter 5 proposed a load shifting technique to accommodate low-latency applications in multi-region renewable energy harvesting in cloud platforms. The proposed technique was designed around local retention of low-latency applications, to which simultaneous multi-threading (SMT) pooling was used. An efficient task scheduling algorithm was designed to leverage server pools of SMT to maintain a static set of compute resources inside the local cloud region for low-latency applications and to co-locate them with best-effort applications, shifting the workload efficiently across the cloud regions.

Chapter 6 proposed an aging-aware CPU core management technique for Large Language Model (LLM) inference cloud deployments to reduce its embodied carbon impact. By investigating CPU components in inference servers, this work uncovered its CPU underutilization patterns to develop a task scheduling technique coupled with dynamic

core deep idling to slow down the CPU aging effects and even out the individual aging of cores. The proposed technique aims to improve CPU underutilization in the short term and prolonged CPU usage in the long term, allowing cloud providers to further amortize the cluster's embodied carbon over the extended lifetime. This work also presented an open-sourced implementation of an extended simulator from a production cloud provider to effectively evaluate CPU aging effects.

Collectively, these chapters proposed multiple algorithms, power delivery management techniques, and effective hardware management approaches for carbon-aware resource management in latency-sensitive cloud environments, which is a timely contribution to the state-of-the-art. The outcomes of this study emphasize the effectiveness of carbon optimization in the broader view of both operational and embodied aspects, highlighting the potential for the overall sustainable growth of latency-sensitive cloud infrastructures.

7.2 Future Research Directions

Based on the research works presented in this thesis, we propose potential future directions for carbon-aware resource management in latency-sensitive cloud computing environments.

7.2.1 Carbon Fault Tolerance

The research works of this thesis often faced challenges in absorbing the energy supply variations of renewable energy sources to deliver application latency performance, in which, we exploited existing fault-tolerance mechanisms presented in the application layer. In contrast, application management can significantly benefit from first-party fault tolerance features defined for carbon efficiency. In that, abstractions of energy dynamics are presented to cloud users in the form of predictive infrastructure unreliability patterns. Cloud users are then invited to implement fault tolerance for such dynamics. The carbon fault tolerance we propose closely resembles spot instances present in cloud offerings today, in which infrastructure underutilization is presented to cloud users with an incentive of reduced usage cost. Given the tightening carbon policies enforced upon

cloud providers, similar incentives can be offered to cloud users for rapid adoption of carbon fault tolerance. Further, limiting the carbon footprint is a key aspect of business growth today. Thus, the carbon fault tolerance adoption incentive may not be monetary but rather be presented as a carbon incentive. In implementing carbon fault tolerance, cloud providers unlock new dimensions in latency-sensitive application layer flexibility for carbon optimization, which can be effectively integrated into its resource management aspects.

7.2.2 Renewables-powered CPU Cores

Driving CPU core availability with renewable energy supply dynamics is an effective method for server power management over intermittent renewable energy integrations. However, it requires efficient management of CPU cores at the cloud virtualization layer. In contrast, combined optimization of CPU architecture and cloud resource management can be advanced for renewables-powered CPU cores, where CPU core sets are physically powered with renewable energy. Analogous to *big.LITTLE* CPU core architectures present today, in which a particular set of cores are designed for low-powered operations, a renewables-powered CPU core set can be reserved for application execution with performance dynamics of energy supply. However, these cores can be operated at peak performance profiles at energy supply peaks rather than being limited to low-powered execution such as in *big.LITTLE*. In return, cloud resource management presents better application management opportunities. Renewables-powered CPU cores can be utilized to execute components of the application layer that provide throttling opportunities, whereas the critical components are executed with the rest of the cores. Further, the performance patterns of such cores can be matched with application performance patterns, uncovering efficient utilization opportunities for renewable energy without affecting application latency performance.

7.2.3 Application Component-level Latency SLOs

Latency-sensitive cloud applications typically define application-wide latency service level objectives (SLOs), leading to inefficient carbon efficiency in the infrastructure that caters to tight latency performance requirements. However, modern applications of-

ten consist of many components, and those components may exhibit different latency SLOs individually. Defining such fine-grained component-level SLOs enables cloud providers to efficiently manage their resources for better carbon efficiency. For instance, the VM execution model introduced in Chapter 4 of this thesis can significantly benefit from component-level latency SLOs in scheduling VMs over renewables-driven cores, enabling deeper utilization of renewable energy with lesser application performance impacts.

7.2.4 Thermal Management for Hardware Degradation

This thesis investigated silicon aging in Chapter 6 towards hardware degradation, which enabled the optimization of embodied carbon in the computing cluster. In that, temperature variations played a significant role. Similarly, the thermal load of the computing hardware can play an important role in hardware degradation. Latency-sensitive cloud environments often involve hardware tuned to high performance, which results in higher thermal loads. In our experiments carried out during this thesis, we observed that CPU power management techniques, such as deep idling, can be effective in dynamic control of the thermal load. Similar techniques can be utilized for other hardware components, such as GPU-specific techniques, to allow the resource management layer to observe the thermal load and optimize that for application latency performance and hardware longevity, enabling improved cloud infrastructure embodied carbon emissions.

7.2.5 Sustainable Performance Profiles

In specific latency-sensitive cloud computing environments, such as real-time cloud systems, hardware must be tuned to specific performance profiles. These involve maintaining peak hardware performance at all times, increasing the possibility of premature hardware failures through component degradation. In contrast, enabling the cloud provider to either dynamically or periodically opt for sustainable performance profiles of reduced performance allow to deliver a balance between hardware degradation and application performance. Cloud providers can exploit specific application usage pat-

terns and manage the performance profile switching at the resource management layer to increase hardware residency time with sustainable performance profiles, which eventually contributes to slowing down the accumulation of embodied carbon emissions through better hardware longevity. Furthermore, dynamic management of sustainable performance profiles has the potential to be leveraged to optimize supply-demand interactions with the energy grid. In that, the characteristics of the power grid can be fed into the cloud infrastructure as an additional input signal for performance profile management at the resource management layer.

7.3 Final Remarks

Latency-sensitive cloud computing environments have emerged as a growing subset of cloud infrastructures for various application use cases, including the Internet of Things, Industry 4.0, and the adoption of Artificial Intelligence-based cloud applications. In delivering the stingiest latency performances, these environments exhibit intertwined application and hardware layers, limiting the cloud provider's control over its application performance management and hardware lifecycle. Adversely, that limits the opportunity to achieve operational and embodied carbon optimization through compromising application performance, such as integrating intermittent renewable energy and optimizing hardware lifecycle. With the growing climate crisis concerns, improving carbon efficiency in latency-sensitive cloud computing environments is critical. In this thesis, we explored the tight coupling of application and hardware layers in application-specific latency-sensitive cloud computing environments to improve its carbon efficiency through efficient resource management. The algorithms, models, power delivery architectures, and hardware management approaches presented in this thesis achieve better carbon efficiency of both operational and embodied aspects while maintaining adequate application latency performances. Furthermore, the research outcomes of this thesis identify opportunities to continue advancing carbon efficiency in latency-sensitive cloud computing environments.

Bibliography

[1] I. Schneider, H. Xu, S. Benecke, D. Patterson, K. Huang, P. Ranganathan, and C. Elsworth, "Life-cycle emissions of ai hardware: A cradle-to-grave approach and generational trends," 2025.

[2] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in Proceedings of the 26th Symposium on Operating Systems Principles, 2017, pp. 153–167.

[3] J. R. C. European Commission, "Photovoltaic geographical information system," Available at https://re.jrc.ec.europa.eu/pvg_tools/en/#DR. (2022).

[4] ETSI. (2020) European telecommunications standards institute's osm mano autohealing. [Online]. Available: https://osm.etsi.org/docs/user-guide/v15/05-osm-usage.html#autohealing

[5] Intel, Intel 64 and IA-32 Architectures Software Developers Manual. Intel, 2016.

[6] A. Agarwal, S. Noghabi, I. Goiri, S. Seshan, and A. Badam, "Unlocking unallocated cloud capacity for long, uninterruptible workloads," in Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23), 2023, pp. 457–478.

[7] RTEval, "Rteval," 2023. [Online]. Available: https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/rteval

[8] Azure. (2020) Azure trace for packing 2020. [Online]. Available: https://github.com/Azure/AzurePublicDataset/blob/master/AzureTracesForPacking2020.md

[9] Google, "Carbon free energy for google cloud regions," 2025. [Online]. Available: https://cloud.google.com/sustainability/region-carbon

[10] T. Anderson, A. Belay, M. Chowdhury, A. Cidon, and I. Zhang, "Treehouse: A case for carbon-aware datacenter software," SIGENERGY Energy Inform. Rev., pp. 64–70, 2023.

[11] C. Miao, Z. Zhong, Y. Xiao, F. Yang, S. Zhang, Y. Jiang, Z. Bai, C. Lu, J. Geng, Z. He, Y. Wang, X. Zou, and C. Yang, "Megate: Extending wan traffic engineering to millions of endpoints in virtualized cloud," in In Proceedings of the ACM SIGCOMM 2024 Conference, 2024, pp. 103–116.

[12] Y. L. Li, O. Graif, and U. Gupta, "Towards carbon-efficient llm life cycle," in Proceedings of the 3rd Workshop on Sustainable Computer Systems, 2024.

[13] Q. Zhang, F. Liu, and C. Zeng, "Adaptive interference-aware vnf placement for service-customized 5g network slices," in Proceedings of the IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, 2019, pp. 2449–2457.

[14] P. Patel, E. Choukse, C. Zhang, A. Shah, I. n. Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative llm inference using phase splitting," in 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA), 2024, pp. 118–132.

[15] R. Bianchini, C. Belady, and A. Sivasubramaniam, "Datacenter power and energy management: past, present, and future," IEEE Micro, pp. 1–9, 2024, early access.

[16] OpenAI. (2022) Introducing chatgpt. [Online]. Available: https://openai.com/index/chatgpt/

[17] G. Gala, G. Fohler, P. Tummeltshammer, S. Resch, and R. Hametner, "Rt-cloud: Virtualization technologies and cloud computing for railway use-case," in Proceedings of the 24th IEEE International Symposium on Real-Time Distributed Computing (ISORC), 2021, pp. 105–113.

[18] J. Bowne. (2024) Using large language models in learning and teaching. [Online]. Available: https://biomedicalsciences.unimelb.edu.au/study/dlh/assets/documents/large-language-models-in-education/llms-in-education

[19] IEA. (2025) International energy agency's energy and ai report. [Online]. Available: https://www.iea.org/reports/energy-and-ai

[20] ——. (2023) International energy agency's data centres and data transmission networks. [Online]. Available: https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks

[21] M. Corporation. (2022) The role of embodied carbon in cloud emissions. [Online]. Available: https://go.microsoft.com/fwlink/p/?linkid=2233506

[22] X. Zhang, Y. Yang, and D. Wang, "Spatial-temporal embodied carbon models for the embodied carbon accounting of computer systems," in Proceedings of the 15th ACM International Conference on Future and Sustainable Energy Systems, 2024, pp. 464–471.

[23] S. Beatty. (2024) Microsoft builds first datacenters with wood to slash carbon emissions. [Online]. Available: https://news.microsoft.com/source/features/sustainability/microsoft-builds-first-datacenters-with-wood-to-slash-carbon-emissions/

[24] U. Nations. (2020) The climate crisis - a race we can win. [Online]. Available: https://www.un.org/sites/un2.un.org/files/2020/01/un75_climate_crisis.pdf

[25] P. Patel, T. Gregersen, and T. Anderson, "An agile pathway towards carbon-aware clouds," SIGENERGY Energy Inform. Rev., vol. 4, pp. 10–17, Sep. 2024.

[26] AWS. (2025) Sustainability in the cloud. [Online]. Available: https://sustainability.aboutamazon.com/products-services/aws-cloud

[27] M. Nakagawa. (2024) Sustainable by design: Advancing the sustainability of ai. [Online]. Available: https://blogs.microsoft.com/blog/2024/04/02/sustainable-by-design-advancing-the-sustainability-of-ai/

[28] Google. (2024) Environmental report. [Online]. Available: https://sustainability.google/reports/google-2024-environmental-report/

[29] Openstack. (2021) Real time. [Online]. Available: https://docs.openstack.org/nova/2023.2/admin/real-time.html

[30] Q. Pei, S. Chen, Q. Zhang, X. Zhu, F. Liu, Z. Jia, Y. Wang, and Y. Yuan, "Cooledge: Hotspot-relievable warm water cooling for energy-efficient edge datacenters," in Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2022, pp. 814–829.

[31] M. Xu, Z. Fu, X. Ma, L. Zhang, Y. Li, F. Qian, S. Wang, K. Li, J. Yang, and X. Liu, "From cloud to edge: A first look at public edge platforms," in Proceedings of the 21st ACM Internet Measurement Conference, 2021, pp. 37–53.

[32] G. Protocol. (2024) Greenhouse gas protocol. [Online]. Available: https://ghgprotocol.org/about-us

[33] Openstack. (2024) The most widely deployed open source cloud software in the world. [Online]. Available: https://www.openstack.org

[34] F. Kong and X. Liu, "A survey on green-energy-aware power management for datacenters," ACM Comput. Surv., vol. 47, no. 2, 2014.

[35] L.-D. Radu, "Green cloud computing: A literature survey," Symmetry, vol. 9, no. 12, 2017.

[36] J. Shuja, A. Gani, S. Shamshirband, R. W. Ahmad, and K. Bilal, "Sustainable cloud data centers: A survey of enabling techniques and technologies," Renewable and Sustainable Energy Reviews, vol. 62, pp. 195–214, 2016.

[37] R. Farahani, Z. Azimi, C. Timmerer, and R. Prodan, "Towards ai-assisted sustainable adaptive video streaming systems: Tutorial and survey," arXiv preprint arXiv:2406.02302, 2024.

[38] X. Liu and R. Buyya, "Resource management and scheduling in distributed stream processing systems: A taxonomy, review, and future directions," ACM Comput. Surv., vol. 53, no. 3, May 2020.

[39] P. Souza, T. Ferreto, and R. Calheiros, "Maintenance operations on cloud, edge, and iot environments: Taxonomy, survey, and research challenges," ACM Comput. Surv., vol. 56, no. 10, Jun. 2024.

[40] C. Barrios and M. Kumar, "Service caching and computation reuse strategies at the edge: A survey," ACM Comput. Surv., vol. 56, no. 2, 2023.

[41] R. Jeyaraj, A. Balasubramaniam, A. K. M.A., N. Guizani, and A. Paul, "Resource management in cloud and cloud-influenced technologies for internet of things applications," ACM Comput. Surv., vol. 55, 2023.

[42] M. Goudarzi, M. Palaniswami, and R. Buyya, "Scheduling iot applications in edge and fog computing environments: A taxonomy and future directions," ACM Comput. Surv., vol. 55, 2022.

[43] F. A. Salaht, F. Desprez, and A. Lebre, "An overview of service placement problem in fog and edge computing," ACM Comput. Surv., vol. 53, no. 3, 2020.

[44] R. Queiroz, T. Cruz, J. Mendes, P. Sousa, and P. Simões, "Container-based virtualization for real-time industrial systemsa systematic review," ACM Comput. Surv., vol. 56, no. 3, Oct. 2023.

[45] H. Yuan, J. Bi, and M. Zhou, "Spatiotemporal task scheduling for heterogeneous delay-tolerant applications in distributed green data centers," IEEE Transactions on Automation Science and Engineering, vol. 16, pp. 1686–1697, 2019.

[46] A. A. Chien, L. Lin, H. Nguyen, V. Rao, T. Sharma, and R. Wijayawardana, "Reducing the carbon impact of generative ai inference (today and in 2035)," in Proceedings of the 2nd Workshop on Sustainable Computer Systems, 2023.

[47] K. Kaur, S. Garg, G. Kaddoum, S. H. Ahmed, and M. Atiquzzaman, "Keids: Kubernetes-based energy and interference driven scheduler for industrial iot in edge-cloud ecosystem," IEEE Internet of Things Journal, vol. 7, no. 5, pp. 4228–4237, 2020.

[48] S. Sajid, M. Jawad, K. Hamid, M. U. Khan, S. M. Ali, A. Abbas, and S. U. Khan, "Blockchain-based decentralized workload and energy management of geo-distributed data centers," Sustainable Computing: Informatics and Systems, vol. 29, pp. 100–461, 2021.

[49] J. Sun, Z. Gong, A. Agarwal, S. Noghabi, R. Chandra, M. Snir, and J. Huang, "Exploring the efficiency of renewable energy-based modular data centers at scale," in Proceedings of the 2024 ACM Symposium on Cloud Computing, 2024, pp. 552–569.

[50] T. Sukprasert, A. Souza, N. Bashir, D. Irwin, and P. Shenoy, "On the limitations of carbon-aware temporal and spatial workload shifting in the cloud," in Proceedings of the Nineteenth European Conference on Computer Systems, 2024, pp. 924–941.

[51] J. Murillo, W. A. Hanafy, D. Irwin, R. Sitaraman, and P. Shenoy, "Cdn-shifter: Leveraging spatial workload shifting to decarbonize content delivery networks," in Proceedings of the 2024 ACM Symposium on Cloud Computing, 2024, pp. 505–521.

[52] A. Souza, S. Jasoria, B. Chakrabarty, A. Bridgwater, A. Lundberg, F. Skogh, A. Ali-Eldin, D. Irwin, and P. Shenoy, "Casper: Carbon-aware scheduling and provisioning for distributed web services," in Proceedings of the 14th International Green and Sustainable Computing Conference, 2024, pp. 67–73.

[53] C. Li, X. Li, R. Wang, T. Li, N. Goswami, and D. Qian, "Chameleon: Adapting throughput server to time-varying green power budget using online learning," in Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED), 2013, pp. 100–105.

[54] A. Jahanshahi, N. Yu, and D. Wong, "Powermorph: Qos-aware server power reshaping for data center regulation service," ACM Trans. Archit. Code Optim., vol. 19, 2022.

[55] N. Sharma, S. Barker, D. Irwin, and P. Shenoy, "Blink: managing server clusters on intermittent power," in Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems, 2011, pp. 185–198.

[56] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar, "Benefits and limitations

of tapping into stored energy for datacenters," SIGARCH Comput. Archit. News, vol. 39, pp. 341–352, Jun. 2011.

[57] P. Ambati, . Goiri, F. Vieira Frujeri, A. Gun, K. Wang, B. Dolan, B. Corell, S. Pasupuleti, T. Moscibroda, S. Elnikety, M. Fontoura, and R. Bianchini, "Providing slos for resource-harvesting vms in cloud platforms," in Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI), 2020.

[58] A. Souza, N. Bashir, J. Murillo, W. Hanafy, Q. Liang, D. Irwin, and P. Shenoy, "Ecovisor: A virtual energy system for carbon-efficient applications," in Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, 2023, pp. 252–265.

[59] Google. (2022) Rethinking your vm strategy with spot vms. [Online]. Available: https://cloud.google.com/blog/topics/cost-management/rethinking-your-vm-strategy-spot-vms

[60] V. Shandilya. (2020) Announcing the general availability of azure spot virtual machines. [Online]. Available: https://azure.microsoft.com/en-us/blog/announcing-the-general-availability-of-azure-spot-virtual-machines/

[61] V. U. Gsteiger, P. H. D. Long, Y. J. Sun, P. Javanrood, and M. Shahrad, "Caribou: Fine-grained geospatial shifting of serverless applications for sustainability," in Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles, 2024, pp. 403–420.

[62] A. Agarwal, S. Noghabi, I. Goiri, S. Seshan, and A. Badam, "Unlocking unallocated cloud capacity for long, uninterruptible workloads," in Proceedings of 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23), 2023, pp. 457–478.

[63] S. Ji, Z. Yang, X. Chen, S. Cahoon, J. Hu, Y. Shi, A. K. Jones, and P. Zhou, "Scarif: Towards carbon modeling of cloud servers with accelerators," in Proceedings of the 2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2024, pp. 496–501.

[64] S. Tannu and P. J. Nair, "The dirty secret of ssds: Embodied carbon," SIGENERGY Energy Inform. Rev., vol. 3, pp. 4–9, 2023.

[65] N. Gupta, I. Narayanan, S. Handa, S. Chakraborti, P. Thapar, B. Shan, A. Rao, Y. Liu, P. Wang, Y. Wu, Q. Gao, C. C.-C. Cheng, S. You, L. Huang, J. Fan, K. Yu, K. Lin, T. Mu, P. Malani, H. Wang, T. Lu, and P. Zhang, "Dynamic idle resource leasing to safely oversubscribe capacity at meta," in Proceedings of the 2024 ACM Symposium on Cloud Computing, 2024, pp. 792–810.

[66] Y. Zhong, D. S. Berger, C. Waldspurger, R. Wee, I. Agarwal, R. Agarwal, F. Hady, K. Kumar, M. D. Hill, M. Chowdhury, and A. Cidon, "Managing memory tiers with CXL in virtualized environments," in Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24), 2024, pp. 37–56.

[67] J. Zhao, K. Lim, T. Anderson, and N. Enright Jerger, "The case of unsustainable cpu affinity," in Proceedings of the 2nd Workshop on Sustainable Computer Systems, 2023.

[68] J. Wang, U. Gupta, and A. Sriraman, "Peeling back the carbon curtain: Carbon optimization challenges in cloud computing," in Proceedings of the 2nd Workshop on Sustainable Computer Systems, 2023.

[69] U. Gupta, M. Elgamal, G. Hills, G.-Y. Wei, H.-H. S. Lee, D. Brooks, and C.-J. Wu, "Act: designing sustainable computer systems with an architectural carbon modeling tool," in Proceedings of the 49th Annual International Symposium on Computer Architecture, 2022, pp. 784–799.

[70] S. McAllister, Y. S. Wang, B. Berg, D. S. Berger, G. Amvrosiadis, N. Beckmann, and G. R. Ganger, "FairyWREN: A sustainable cache for emerging Write-Read-Erase flash interfaces," in Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24), 2024, pp. 745–764.

[71] J. Wang, D. S. Berger, F. Kazhamiaka, C. Irvene, C. Zhang, E. Choukse, K. Frost, R. Fonseca, B. Warrier, C. Bansal, J. Stern, R. Bianchini, and A. Sriraman, "De-

signing cloud servers for lower carbon," in Proceedings of the 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA), 2024, pp. 452–470.

[72] J. Zhao, M. A. Rodrĩguez, and R. Buyya, "A deep reinforcement learning approach to resource management in hybrid clouds harnessing renewable energy and task scheduling," in Proceedings of 2021 IEEE 14th International Conference on Cloud Computing (CLOUD), 2021, pp. 240–249.

[73] W. Trneberg, A. Mehta, E. Wadbro, J. Tordsson, J. Eker, M. Kihl, and E. Elmroth, "Dynamic application placement in the mobile cloud network," Future Generation Computer Systems, vol. 70, pp. 163–177, 2017.

[74] M. A. Islam and S. Ren, "Ohm's law in data centers: A voltage side channel for timing power attacks," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 146–162.

[75] A. G. Kumbhare, R. Azimi, I. Manousakis, A. Bonde, F. Frujeri, N. Mahalingam, P. A. Misra, S. A. Javadi, B. Schroeder, M. Fontoura, and R. Bianchini, "Prediction-Based power oversubscription in cloud platforms," in Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC 21), 2021, pp. 473–487.

[76] X. Fu, X. Wang, and C. Lefurgy, "How much power oversubscription is safe and allowed in data centers," in Proceedings of the 8th ACM International Conference on Autonomic Computing, 2011, pp. 21–30.

[77] Y. Muhammad, R. Khan, M. A. Z. Raja, F. Ullah, N. I. Chaudhary, and Y. He, "Solution of optimal reactive power dispatch with facts devices: A survey," Energy Reports, vol. 6, pp. 2211–2229, 2020.

[78] P. Huang, B. Copertaro, X. Zhang, J. Shen, I. Lfgren, M. Rnnelid, J. Fahlen, D. Andersson, and M. Svanfeldt, "A review of data centers as prosumers in district energy systems: Renewable energy integration and waste heat reuse for district heating," Applied Energy, vol. 258, p. 114109, 2020.

[79] J. Panadero, M. Selimi, L. Calvet, J. M. Marqus, and F. Freitag, "A two-stage multi-criteria optimization method for service placement in decentralized edge micro-clouds," Future Generation Computer Systems, vol. 121, pp. 90–105, 2021.

[80] M. Selimi, L. Cerdà-Alabern, F. Freitag, L. Veiga, A. Sathiaseelan, and J. Crowcroft, "A lightweight service placement approach for community network micro-clouds," Journal of Grid Computing, vol. 17, pp. 169–189, 2019.

[81] A. Saeed, V. Gupta, P. Goyal, M. Sharif, R. Pan, M. Ammar, E. Zegura, K. Jang, M. Alizadeh, A. Kabbani, and A. Vahdat, "Annulus: A dual congestion control loop for datacenter and wan traffic aggregates," in Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, 2020, pp. 735–749.

[82] D. Sharma and S. Rao, "Scheduling computing loads for improved utilization of solar energy," Sustainable Computing: Informatics and Systems, vol. 32, p. 100592, 2021.

[83] M. Acton, P. Bertoldi, J. Booth, L. Newcombe, A. Rouyer, and R. Tozer, "2018 best practice guidelines for the eu code of conduct on data centre energy efficiency," Publications Office of the European Union, Luxembourg, Tech. Report. EUR 29103 EN, 2018, 2017.

[84] Z. Zhou, "Greenedge: Greening edge datacenters with energy-harvesting iot devices," in Proceedings of the 27th IEEE International Conference on Network Protocols (ICNP), 2019, pp. 1–6.

[85] S. Prez, P. Arroba, and J. M. Moya, "Energy-conscious optimization of edge computing through deep reinforcement learning and two-phase immersion cooling," Future Generation Computer Systems, vol. 125, pp. 891–907, 2021.

[86] S. Ilager, K. Ramamohanarao, and R. Buyya, "Etas: Energy and thermal-aware dynamic virtual machine consolidation in cloud data center with proactive hotspot

mitigation," Concurrency and Computation: Practice and Experience, vol. 31, p. e5221, 2019.

[87] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software: Practice and Experience, vol. 41, pp. 23–50, 2011.

[88] R. Brännvall, M. Siltala, J. Gustafsson, J. Sarkinen, M. Vesterlund, and J. Summers, "Edge: Microgrid data center with mixed energy storage," in Proceedings of the Eleventh ACM International Conference on Future Energy Systems, 2020, pp. 466–473.

[89] P. Arroba, J. M. Moya, J. L. Ayala, and R. Buyya, "Dynamic voltage and frequency scaling-aware dynamic consolidation of virtual machines for energy efficient cloud data centers," Concurrency and Computation: Practice and Experience, vol. 29, p. e4067, 2017.

[90] V. K. Jayakumar, S. Arbat, I. K. Kim, and W. Wang, "Cloudbruno: A low-overhead online workload prediction framework for cloud computing," in Proceedings of the 2022 IEEE International Conference on Cloud Engineering (IC2E), 2022, pp. 188–198.

[91] A. Atrey, G. Van Seghbroeck, H. Mora, F. De Turck, and B. Volckaert, "Spech: A scalable framework for data placement of data-intensive services in geo-distributed clouds," Journal of Network and Computer Applications, vol. 142, pp. 1–14, 2019.

[92] H. Ma, Z. Zhou, and X. Chen, "Leveraging the power of prediction: Predictive service placement for latency-sensitive mobile edge computing," IEEE Transactions on Wireless Communications, vol. 19, pp. 6454–6468, 2020.

[93] IEA. (2023) International energy agency's report on low-emissions sources of electricity. [Online]. Available: https://www.iea.org/reports/low-emissions-sources-of-electricity

[94] T. Sukprasert, A. Souza, N. Bashir, D. Irwin, and P. Shenoy, "On the limitations of carbon-aware temporal and spatial workload shifting in the cloud," in Proceedings of the Nineteenth European Conference on Computer Systems, 2024, pp. 924–941.

[95] E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. Koomey, "Recalibrating global data center energy-use estimates," Science, vol. 367, no. 6481, pp. 984–986, 2020.

[96] A. Radovanovi, R. Koningstein, I. Schneider, B. Chen, A. Duarte, B. Roy, D. Xiao, M. Haridasan, P. Hung, N. Care, S. Talukdar, E. Mullen, K. Smith, M. Cottman, and W. Cirne, "Carbon-aware computing for datacenters," IEEE Transactions on Power Systems, vol. 38, no. 2, pp. 1270–1280, 2023.

[97] J. Zheng, A. A. Chien, and S. Suh, "Mitigating curtailment and carbon emissions through load migration between data centers," Joule, vol. 4, no. 10, pp. 2208–2222, 2020.

[98] E. Barbieri. (2023) What is real-time linux? part i. [Online]. Available: https://ubuntu.com/blog/what-is-real-time-linux-i

[99] J. R. David Reinsel, John Gantz, "The digitization of the world from edge to core," 2018. [Online]. Available: https://www.readkong.com/page/the-digitization-of-the-world-from-edge-to-core-8666239

[100] L. Piga, I. Narayanan, A. Sundarrajan, M. Skach, Q. Deng, B. Maity, M. Chakkaravarthy, A. Huang, A. Dhanotia, and P. Malani, "Expanding datacenter capacity with dvfs boosting: A safe and scalable deployment experience," in Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1, 2024, pp. 150–165.

[101] A. G. Kumbhare, R. Azimi, I. Manousakis, A. Bonde, F. Frujeri, N. Mahalingam, P. A. Misra, S. A. Javadi, B. Schroeder, M. Fontoura, and R. Bianchini, "Prediction-Based power oversubscription in cloud platforms," in Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC 21), 2021, pp. 473–487.

[102] C. Li, W. Zhang, C.-B. Cho, and T. Li, "Solarcore: Solar energy driven multi-core architecture power management," in Proceedings of the 17th IEEE International Symposium on High Performance Computer Architecture, 2011, pp. 205–216.

[103] O. Hadary, L. Marshall, I. Menache, A. Pan, E. E. Greeff, D. Dion, S. Dorminey, S. Joshi, Y. Chen, M. Russinovich, and T. Moscibroda, "Protean: VM allocation service at scale," in Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), 2020, pp. 845–861.

[104] T. Baker, B. Aldawsari, M. Asim, H. Tawfik, Z. Maamar, and R. Buyya, "Cloud-senergy: A bin-packing based multi-cloud service broker for energy efficient composition and execution of data-intensive applications," Sustainable Computing: Informatics and Systems, vol. 19, pp. 242–252, 2018.

[105] Ubuntu, "A ctos guide to real-time linux," 2023. [Online]. Available: https://ubuntu.com/engage/cto-guide-real-time-kernel

[106] Intel. (2024) Overview of intel time coordinated computing (tcc) tools - measurement library. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/real-time-systems-measurement-library.html

[107] ——. (2016) Nfv performance optimization for virtualized customer premises equipment. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/nfv-performance-optimization-for-vcpe.html

[108] P. Ambati, I. Goiri, F. Frujeri, A. Gun, K. Wang, B. Dolan, B. Corell, S. Pasupuleti, T. Moscibroda, S. Elnikety, M. Fontoura, and R. Bianchini, "Providing SLOs for Resource-Harvesting VMs in cloud platforms," in Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), 2020, pp. 735–751.

[109] G. Durrieu, G. Fohler, G. Gala, S. Girbal, D. Gracia Pérez, E. Noulard, C. Pagetti, and S. Pérez, "DREAMS about reconfiguration and adaptation in avionics," in Proceedings of the ERTS 2016, 2016, pp. 48–57.

[110] K. M. U. Ahmed, M. H. J. Bollen, and M. Alvarez, "A review of data centers energy consumption and reliability modeling," IEEE Access, vol. 9, pp. 152 536–152 563, 2021.

[111] S. Qi, D. Milojicic, C. Bash, and S. Pasricha, "Shield: Sustainable hybrid evolutionary learning framework for carbon, wastewater, and energy-aware data center management," in Proceedings of the 14th International Green and Sustainable Computing Conference, 2024, pp. 56–62.

[112] A. Radovanovic, B. Chen, S. Talukdar, B. Roy, A. Duarte, and M. Shahbazi, "Power modeling for effective datacenter planning and compute management," IEEE Transactions on Smart Grid, vol. 13, no. 2, pp. 1611–1621, 2022.

[113] R. Basmadjian and H. de Meer, "Evaluating and modeling power consumption of multi-core processors," in Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet, 2012.

[114] Linux. (2023) Linux foundation projects: Real-time linux tools: Cyclictest. [Online]. Available: https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest/start

[115] CERN, "Rt (realtime) @ cern," 2024. [Online]. Available: https://linux.web.cern.ch/rt/

[116] ELIA. (2024) Elia group open data platform. [Online]. Available: https://www.elia.be/en/grid-data/open-data

[117] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "Rapl in action: Experiences in using rapl for power measurements," ACM Trans. Model. Perform. Eval. Comput. Syst., vol. 3, no. 2, 2018.

[118] OpenStack. (2023) Compute schedulers. [Online]. Available: https://docs.openstack.org/nova/2023.2/admin/scheduling.html

[119] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. Andrew, "Greening geographical load balancing," SIGMETRICS Perform. Eval. Rev., vol. 39, no. 1, pp. 193–204, 2011.

[120] A. James and D. Schien, "A low carbon kubernetes scheduler," in Proceedings of the 6th International Conference on ICT for Sustainability, ICT4S 2019, Lappeenranta, Finland, June 10-14, 2019, vol. 2382, 2019.

[121] W. A. Hanafy, Q. Liang, N. Bashir, D. Irwin, and P. Shenoy, "Carbonscaler: Leveraging cloud workload elasticity for optimizing carbon-efficiency," Proc. ACM Meas. Anal. Comput. Syst., 2023.

[122] J. Murillo, W. A. Hanafy, D. Irwin, R. Sitaraman, and P. Shenoy, "Cdn-shifter: Leveraging spatial workload shifting to decarbonize content delivery networks," in Proceedings of the 2024 ACM Symposium on Cloud Computing, 2024, pp. 505–521.

[123] Y. Guo and G. Porter, "Carbon-aware inter-datacenter workload scheduling and placement," in Poster Session of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI'23), 2023.

[124] M. Liu and X. Tang, "Dynamic bin packing with predictions," Proc. ACM Meas. Anal. Comput. Syst., vol. 6, 2022.

[125] Google, "24/7 by 2030: Realizing a carbon-free future," 2020. [Online]. Available: https://www.gstatic.com/gumdrop/sustainability/247-carbon-free-energy.pdf

[126] R. J. Wysocki, "Cpu idle time management," 2018. [Online]. Available: https://docs.kernel.org/admin-guide/pm/cpuidle.html

[127] Openstack, "Cpu topologies," 2023. [Online]. Available: https://docs.openstack.org/nova/2023.2/admin/cpu-topologies.html

[128] I. Corporation, "Cpu idle time management," 2018. [Online]. Available: https://www.kernel.org/doc/html/v5.4/admin-guide/pm/cpuidle.html

[129] ——, "Intel hyper-threading technology," 2025. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html

[130] T. B. Hewage, "core-power-mgt," 2025. [Online]. Available: https://github.com/tharindu-b-hewage/core-power-mgt

[131] GitHub. (2021) Introducing github copilot: your ai pair programmer. [Online]. Available: https://github.blog/news-insights/product-news/introducing-github-copilot-ai-pair-programmer/

[132] T. F. Times. (2024) Openai targets 1bn users in next phase of growth. [Online]. Available: https://www.ft.com/content/e91cb018-873c-4388-84c0-46e9f82146b4

[133] Crusoe. (2024) Crusoe to build initial 200 mw ai data center with plans to expand at 1.2 gw lancium clean campus. [Online]. Available: https://crusoe.ai/newsroom/crusoe-200mw-ai-data-center/

[134] L. Office of the governor. (2024) Landry announces meta selects north louisiana as site of $10 billion artificial intelligence optimized data center. [Online]. Available: https://gov.louisiana.gov/news/4697

[135] Reuters. (2024) Musk's xai plans massive expansion of ai supercomputer in memphis. [Online]. Available: https://www.reuters.com/technology/artificial-intelligence/musks-xai-plans-massive-expansion-ai-supercomputer-memphis-2024-12-04/

[136] R. Merritt. (2023) Why gpus are great for ai. [Online]. Available: https://blogs.nvidia.com/blog/why-gpus-are-great-for-ai/

[137] P. Schmid, O. Sanseviero, P. Cuenca, and L. Tunstall. (2023) Llama 2 is here - get it on hugging face. [Online]. Available: https://huggingface.co/blog/llama2

[138] S. Verma and N. Vaidya. (2023) Mastering llm techniques: Inference optimization. [Online]. Available: https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/

[139] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for Transformer-Based generative models," in Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), 2022, pp. 521–538.

[140] Crusoe. (2023) How together and crusoe are reducing the carbon impact of generative ai. [Online]. Available: https://crusoe.ai/blog/crusoe-together-reducing-carbon-impact-of-generative-ai/

[141] D. Maji, N. Bashir, D. Irwin, P. Shenoy, and R. K. Sitaraman, "Untangling carbon-free energy attribution and carbon intensity estimation for carbon-aware computing," in Proceedings of the 15th ACM International Conference on Future and Sustainable Energy Systems, 2024, pp. 580–588.

[142] D. Gnad, M. Shafique, F. Kriebel, S. Rehman, D. Sun, and J. Henkel, "Hayat: harnessing dark silicon and variability for aging deceleration and balancing," in Proceedings of the 52nd Annual Design Automation Conference, 2015.

[143] A. Tiwari and J. Torrellas, "Facelift: Hiding and slowing down aging in multicores," in 2008 41st IEEE/ACM International Symposium on Microarchitecture, 2008, pp. 129–140.

[144] A. Tomlinson and G. Porter, "Something old, something new: Extending the life of cpus in datacenters," SIGENERGY Energy Inform. Rev., vol. 3, no. 3, pp. 59–63, 2023.

[145] A. F. Lorenzon, G. Korol, M. Brandalero, and A. C. S. Beck, "Harnessing the effects of process variability to mitigate aging in cloud servers," in Proceedings of the 2023 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2023, pp. 1–6.

[146] M. Ansari, S. Safari, A. Yeganeh-Khaksar, R. Siyadatzadeh, P. Gohari-Nazari, H. Khdr, M. Shafique, J. Henkel, and A. Ejlali, "Atlas: Aging-aware task replication for multicore safety-critical systems," in Proceedings of the 29th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2023, pp. 223–234.

[147] F. S. Saadatmand, N. Rohbani, F. Baharvand, and H. Farbeh, "Tamer: an adaptive task allocation method for aging reduction in multi-core embedded real-time systems," The Journal of Supercomputing, vol. 77, pp. 1939–1957, 2021.

[148] A. Faiz, S. Kaneda, R. Wang, R. C. Osi, P. Sharma, F. Chen, and L. Jiang, "LLMCarbon: Modeling the end-to-end carbon footprint of large language models," in Proceedings of the Twelfth International Conference on Learning Representations, 2024.

[149] S. Nguyen, B. Zhou, S. Liu, and Y. Ding, "Towards sustainable large language model serving," in Proceedings of the 3rd Workshop on Sustainable Computer Systems, 2024.

[150] Z. Fu, F. Chen, S. Zhou, H. Li, and L. Jiang, "Llmco2: Advancing accurate carbon footprint prediction for llm inferences," arXiv preprint arXiv:2410.02950, 2024.

[151] B. Li, Y. Jiang, V. Gadepally, and D. Tiwari, "Sprout: Green generative ai with carbon-efficient llm inference," in Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, 2024, pp. 21 799–21 813.

[152] T. Shi, Y. Wu, S. Liu, and Y. Ding, "Greenllm: Disaggregating large language model serving on heterogeneous gpus for lower carbon emissions," arXiv preprint arXiv:2412.20322, 2024.

[153] J. H. Yahya, H. Volos, D. B. Bartolini, G. Antoniou, J. S. Kim, Z. Wang, K. Kalaitzidis, T. Rollet, Z. Chen, Y. Geng, O. Mutlu, and Y. Sazeides, "Agilewatts: An energy-efficient cpu core idle-state architecture for latency-sensitive server applications," in Proceedings of the 55th IEEE/ACM International Symposium on Microarchitecture (MICRO), 2022, pp. 835–850.

[154] Lenovo. (2025) Thinksystem sr650 maintenance manual. [Online]. Available: https://pubs.lenovo.com/sr650/sr650_maintenance_manual.pdf

[155] Intel. (2024) How to find compatible motherboards for the intel xeon processor family? [Online]. Available: https://www.intel.com/content/www/us/en/support/articles/000057630/processors.html

[156] D. S. Berger, F. Kazhamiaka, E. Choukse, . Goiri, C. Irvene, P. Misra, A. Kumbhare, R. Fonseca, and R. Bianchini, "Research avenues towards net-zero cloud platforms," in NetZero 2023: 1st Workshop on NetZero Carbon Computing, February 2023.

[157] J. Lyu, M. You, C. Irvene, M. Jung, T. Narmore, J. Shapiro, L. Marshall, S. Samal, I. Manousakis, L. Hsu, P. Subbarayalu, A. Raniwala, B. Warrier, R. Bianchini, B. Schroeder, and D. S. Berger, "Hyrax: Fail-in-Place server operation in cloud platforms," in Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23), 2023, pp. 287–304.

[158] F. Shoji, S. Matsui, M. Okamoto, F. Sueyasu, T. Tsukamoto, A. Uno, and K. Yamamoto, "Long term failure analysis of 10 peta-scale supercomputer," HPC in Asia Poster, ISC, 2015.

[159] H. D. Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar, "Silent data corruptions at scale," CoRR, vol. abs/2102.11245, 2021.

[160] P. H. Hochschild, P. Turner, J. C. Mogul, R. Govindaraju, P. Ranganathan, D. E. Culler, and A. Vahdat, "Cores that don't count," in Proceedings of the Workshop on Hot Topics in Operating Systems, 2021, pp. 9–16.

[161] J. H. Yahya, J. S. Kim, A. G. Yalk, J. Park, E. Rotem, Y. Sazeides, and O. Mutlu, "Darkgates: A hybrid power-gating architecture to mitigate the performance impact of dark-silicon in high performance processors," in Proceedings of the 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2022, pp. 1170–1183.

[162] AWS. (2025) Deploying multiple large language models with nvidia triton server and vllm. [Online]. Available: https://awslabs.github.io/data-on-eks/docs/gen-ai/inference/GPUs/vLLM-NVIDIATritonServer

[163] B. Raghunathan, Y. Turakhia, S. Garg, and D. Marculescu, "Cherry-picking: Exploiting process variations in dark-silicon homogeneous chip multi-processors,"

in Proceedings of the 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013, pp. 39–44.

[164] I. Moghaddasi, A. Fouman, M. E. Salehi, and M. Kargahi, "Instruction-level nbti stress estimation and its application in runtime aging prediction for embedded processors," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, pp. 1427–1437, 2019.

[165] Azure. (2024) Ncast4_v3 sizes series. [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/gpu-accelerated/ncast4v3-series?tabs=sizebasic

[166] R. J. Wysocki. (2018) Cpu idle time management. [Online]. Available: https://docs.kernel.org/admin-guide/pm/cpuidle.html

[167] K. Mayo, S. Rajasekaran, I. Pasichnyk, and M. Senizaiz. (2018) High performance computing (hpc) tuning guide. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/tuning-guides/58479_amd-epyc-9005-tg-hpc.pdf

[168] Azure. (2024) Ncads h100 v5-series. [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-machines/ncads-h100-v5

[169] G. Wilkins, S. Keshav, and R. Mortier, "Hybrid heterogeneous clusters can lower the energy consumption of llm inference workloads," in Proceedings of the 15th ACM International Conference on Future and Sustainable Energy Systems, 2024, pp. 506–513.