CARBON EFFICIENT KARPENTER

OPTIMIZING KUBERNETES CLUSTER AUTOSCALING FOR CARBON EFFICIENCY

JACOB VALDEMAR ANDREASEN, 201809369

MASTER'S THESIS

January 2024

Supervisor: Christian Fischer Pedersen

AARHUS
UNIVERSITY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CARBON EFFICIENT KARPENTER

Optimizing Kubernetes Cluster Autoscaling for Carbon Efficiency

JACOB VALDEMAR ANDREASEN

Master's Thesis

Master of Science in Computer Engineering

Department of Electrical and Computer Engineering
Aarhus University

January 2024

DEGREE PROGRAMME
Master of Science (MSc) in Engineering (Computer Engineering)
Civilingeniør, cand.polyt. i computerteknologi

SUPERVISOR
Christian Fischer Pedersen

LOCATION
Aarhus

TIME FRAME
September 2023 - January 2024

Climate change is a threat to human well-being and planetary health (very high confidence). There is a rapidly closing window of opportunity to secure a liveable and sustainable future for all (very high confidence). The choices and actions implemented in this decade will have impacts now and for thousands of years (high confidence).

IPCC. Climate Change 2023: Synthesis Report.

PREFACE

This document is styled for printing. Thus, I recommend using two-page view and showing the cover page separately (odd-numbered pages to the right) when viewing this document on a digital device.

This thesis is the culmination of my educational journey at the Department of Electrical and Computer Engineering, Aarhus University. It was written in the autumn of 2023. A year with a disturbing number of broken climate records. The Canadian wildfires began in March and intensified in June, which was the warmest June on record globally. However, June was nothing compared to July, which was not only the warmest July ever, but the warmest month ever recorded in human history! A 17-day heatwave in Greece with temperatures exceeding 45 °C initiated an array of wildfires on the island of Rhodes, leading to the evacuation of 30,000 people. While the European south was deadly dry in July, the European north was immersed in rain. July 2023 was the most rainy July ever recorded in Denmark. Unfortunately, the records don't stop here. August 2023 was the warmest August on record globally. September 2023 was the warmest September on record globally. October 2023 was the warmest October on record globally. November 2023 was the warmest November on record globally. By December, the wildfires in Canada had burned 18.5 million acres of forest. An area four times the size of Denmark. It should come as no surprise by now, that scientists firmly believe that 2023 was the warmest year ever recorded. Yet, 2023 was probably more chill than 2024 is going to be... For this reason, I can hardly imagine any topic being more pressing than global warming and climate change – and that is why I have written this thesis.

* * *

As I type the last words of my thesis, there's a profound sense of mastery over my subject. Yet, I am acutely aware that this sensation is fleeting. The journey of writing this thesis has taught me a valuable lesson: each time I felt I had grasped all there was to know, a newly discovered detail emerged, unveiling a myriad of previously unconsidered questions. Therefore, my current sense of mastery merely feels like an indication that I soon will uncover a new detail, tearing down the assumptions upon which my thesis is built, and, hopefully, replacing them with something better. Now I have learned to learn. Thus I advise you to read swiftly, to read my thesis before it is possibly rendered obsolete by new insights.

Jacob Valdemar Andreasen
Aarhus, December 2023

ACKNOWLEDGMENTS

It is true that no man is an island – and I am no exception.

I am thankful for the guidance provided by my supervisor, Christian Fischer Pedersen, during the project. Our discussions have helped me see things from a new perspective.

I would like to thank Lunar for providing me with Kubernetes clusters in AWS on which I could perform the evaluation of the proposed solution. Without that generous support, this thesis would not have been possible.

I would like to thank David Ekchajzer from Boavizta for reviewing my Pull Requests on GitHub, explaining how BoaviztAPI works, and collaborating with me to integrate my changes into BoaviztAPI. Furthermore, I would like to thank Benjamin Davy for providing feedback on an important Pull Request I made to BoaviztAPI.

I would like to thank Ellis Tarn, Jonathan Innis, Tim Bannister, Jack Francis, and Nick Tran from the Karpenter community for their invaluable review of my design proposal, introducing me to the community, and helping me grasp Karpenter.

Finally, I would like to thank my family, friends, and colleagues for their support during this project.

ABSTRACT

Data centers are responsible for 0.6% of global greenhouse gas emissions. To reduce these emissions, existing research focus mostly on carbon-aware scheduling and scaling of short-running workloads. This thesis introduces a novel method called carbon efficient cluster autoscaling to reduce operational and embodied carbon emissions from all workloads on cloud-based Kubernetes clusters. A carbon efficient cluster autoscaler for Kubernetes is developed by adapting Karpenter, an open-source state-of-the-art cluster autoscaler, to minimize cloud instance carbon emission estimates from BoaviztAPI. The developed Carbon Efficient Karpenter shows a statistically significant median reduction in cloud instance carbon emissions when compared to the original Karpenter. In locations where electricity has low, moderate, and high carbon intensity, the 25th/75th percentile reductions are 26%/45%, 7%/32%, and 4%/5%, respectively. This suggests that carbon efficient cluster autoscaling is a viable method for reducing the carbon footprint of Kubernetes clusters that run on cloud infrastructure. Widespread adoption could incentivize carbon emission optimizations throughout the cloud computing supply chain. An array of research questions emerge from these findings.

RESUMÉ

Datacentre står for 0,6% af de globale drivhusgasudledninger. For at reducere disse udledninger forskes der p.t. i CO_2-bevidst planlægning og skalering af kortlevende applikationer. I dette speciale præsenteres en ny metode til at minimere drifts- og produktionsrelaterede drivhusgasudledninger fra cloudbaserede Kubernetes clustre. Metoden understøtter alle slags applikationer, og betegnes CO_2-effektiv cluster autoskalering. I specialet implementeres CO_2-effektiv cluster autoskalering ved at tilpasse Karpenter, en open source cluster autoscaler, til at minimere BoaviztAPI's estimater af cloud serveres drivhusgasudledninger. Der konstateres en statistisk signifikant reduktion af cloud servernes drivhusgasudledninger (median) ved brug af den CO_2-effektive Karpenter sammenlignet med den originale Karpenter. På lokationer med lav, moderat og høj CO_2e-intensitet i elnettet er 25-/75-percentilen af reduktionen hhv. 26%/45%, 7%/32% og 4%/5%. Det indikerer, at CO_2-effektiv cluster autoskalering er en lovende metode til at reducere drivhusgasudledninger fra cloudbaserede Kubernetes clustre. Udbredt anvendelse kan muligvis tilskynde cloud computing-forsyningskæden til at minimere drivhusgasudledninger. Disse resultater rejser en række nye forskningsspørgsmål.

CONTENTS

1 INTRODUCTION

The conclusions in the latest report from the Intergovernmental Panel on Climate Change are clear. To ensure a livable planet for all, "rapid and far-reaching transitions across all sectors and systems" are needed [1]. The cloud computing industry is physically represented as data centers and is part of the ICT umbrella term. The global energy demand from data centers in 2022 was approximately 240–340 TW h (excluding cryptocurrency mining) [2], which amounts to 1–1.3 % of global electricity demand [2, 3] or 0.3–0.6 % of global carbon emissions[1][2] [4]. Despite a 340 % growth in demand for data center services from 2015 to 2022 [2], global data center electricity use has only grown moderately [2, 5, 6]. This is in part thanks to improvements in IT efficiency and a shift from small to hyperscale data centers [2, 5]. Kamiya and Kvarnström (International Energy Agency) project data center electricity use to stay flat [5], while Andrae estimate it to double or triple by 2030 [7]. Hyperscale datacenter operators like Amazon, Microsoft, Meta, and Google are increasingly investing in corporate renewable energy Power Purchasing Agreements (PPAs) to reduce operational carbon emissions [2, 8]. Google claims to have been carbon neutral since 2017 through its use of PPAs [3], but PPAs has been criticized in various works [9, 10]. For cloud computing to get on track with global net zero CO_2 in 2050, emissions must be cut in half by 2030 [2]. In this regard, reducing energy use and transitioning to renewable energy is a known path to eliminate *some* carbon emissions [8], but it will require new and innovative methods to eliminate *all* carbon emissions from cloud computing [2, 8].

In cloud computing, the preferred approach to run software applications is using containers which encapsulate an application and its dependencies in an isolated environment. Kubernetes, being the most used orchestration platform in the IT industry [11], automates deploying, scaling, and managing containerized applications [11, 12]. Kubernetes schedules containerized applications to a cluster of cloud servers to increase reliability. Since Kubernetes Clusters usually are over-provisioned, cluster autoscalers are often deployed to scale the cluster of cloud servers horizontally and vertically to keep resource utilization at safe and efficient levels. While most schedulers and cluster autoscalers focus on improving utilization to reduce opera-

1 This also includes embodied emissions.

2 Carbon dioxide is the most prominent greenhouse gas, but it is not the only one. A common scale for describing greenhouse gas and carbon dioxide emissions is *carbon dioxide equivalents* (CO_2e). When I say *carbon*, I am referring to carbon dioxide equivalents, which include all greenhouse gases. This is common practice.

tional costs [13–15], new schedulers and autoscalers have recently been proposed to specifically improve the environmental sustainability of cloud computing [16–19]. These sustainable schedulers and autoscalers are termed *carbon efficient* or *carbon aware*. Carbon awareness is about doing more when electricity has a low carbon intensity and less when it has a high carbon intensity. Carbon efficiency on the other hand is a more general term simply defined as emitting the least amount of carbon possible. Contemporary efforts to reduce the environmental impact of cloud computing focus mostly on carbon awareness and are truly commendable. However, as the next two sections explain, many carbon-aware schedulers have limited applicability and regularly overlook a specific kind of carbon emissions that cloud computing necessitates.

1.1 CARBON-AWARE AND CARBON EFFICIENT SOFTWARE SYSTEMS

In this section, some carbon-aware and carbon efficient software systems are covered. Each subsection focuses on a specific work, describing its purpose, methodology, and results.

1.1.1 *A Low Carbon Kubernetes Scheduler*

In this conference paper (2019) [16], James and Schien describe the design and implementation of a carbon-aware scheduling policy for Kubernetes. This scheduler leverages information about the real-time carbon intensity of the electric grid in some geographic regions, to select a region with low carbon intensity where software workloads are then scheduled. This *demand shifting* method is called *spatial shifting* [20, 21], and by performing it, the Low Carbon Kubernetes Scheduler can reduce operational carbon emissions from software applications running on Kubernetes.

1.1.2 *Carbon Emission-Aware Job Scheduling for Kubernetes Deployments*

In this paper (Jun. 27, 2023) [17], Piontek *et al.* propose a novel carbon-aware workload scheduling algorithm for Kubernetes that can perform *temporal shifting* on non-critical jobs. Temporal shifting means to shift execution start in time. Note that, in contrast to the previous paper "A Low Carbon Kubernetes Scheduler," this scheduler only works on non-critical jobs. A job in Kubernetes is a type of workload that runs to completion [12]. This is in contrast to a web server for example, which is a continuously running workload that keeps handling requests until it is shut down or crashes. Non-critical means that the job can be postponed for some time, e.g. 24 hours.

The proposed scheduler performs temporal shifting of non-critical jobs based on a Weighted Moving Average model that predicts fu-

ture carbon intensity in Germany's electrical grid based on historical data from ElectricityMaps ([22]) [17]. The scheduler does this while satisfying Service Level Objectives for all jobs.

Like before, this scheduler only considers operational carbon emissions and does not count in the cloud provider's Power Usage Effectiveness (PUE). The scheduler is evaluated in four simulations, where it on average reduces a subset of operational carbon emissions from non-critical jobs by 1.25 % (0.37–2.41 %) [17].

1.1.3 *Let's Wait Awhile: How Temporal Workload Shifting Can Reduce Carbon Emissions in the Cloud*

In this paper (2021) [23], Wiesner *et al.* examine "the potential impact of shifting computational workloads towards times where the energy supply is expected to be less carbon-intensive" [23, p. 260]. First, they create terminology for categorizing and differentiating between workloads: duration (short-running, long-running, or continuously running), execution time (ad hoc or scheduled), and interruptibility (interruptible or non-interruptible) [23]. Next, they analyze the carbon intensity in four regions and find that a weekly seasonal pattern can be observed where the carbon intensity is lower on weekends in some regions. This leads Wiesner *et al.* to suggest that in some regions, very flexible workloads can achieve 20 % reduction in operational carbon by shifting execution to weekends. Another conclusion from this analysis is that there is little potential to reduce emissions by only shifting workloads up to two hours throughout the day. However, for some peak carbon intensity hours (varies between regions) small reductions can be achieved when shifting workloads up to two hours. When considering workloads that can shift up to eight hours, the reduction potential increases, but the amount varies significantly between regions [23].

The simulation framework and datasets from the paper is publicly available [24].

1.1.4 *CarbonScaler: Leveraging Cloud Workload Elasticity for Optimizing Carbon-Efficiency*

In this article (Dec. 12, 2023) [25], Hanafy *et al.* design CarbonScaler, a carbon-aware job autoscaler that minimizes Kubernetes jobs' operational carbon emissions subject to available resource elasticity and temporal flexibility [25]. This is done by scaling available compute resources using a novel algorithm that takes a 24-hour carbon intensity forecast as input. They implement *Carbon AutoScaler* on top of KubeFlow [26] and has published the source code online.

The evaluation of CarbonScaler shows that it achieves a 16 % median reduction in operational carbon emission across [25]. Regions

that exhibit higher diurnal variations in carbon intensity contain a larger carbon reductions potential using this approach. The method only applies to jobs that can be postponed for 24 hours [25].

1.1.5 *Karpenter*

Karpenter is an open-source cluster autoscaler for Kubernetes [27]. Karpenter adds new servers to a Kubernetes Cluster to satisfy the resource requests from unschedulable Pods. Additionally, Karpenter can consolidate servers to reduce resource overhead and lower compute costs. While Karpenter may reduce the carbon footprint of a Kubernetes Cluster compared to one that is manually managed, it is not built to optimize environmental sustainability.

The development and maintenance of Karpenter is currently led by developers employed at Amazon Web Services (AWS). Yet, Karpenter is formally owned by the Cloud Native Computing Foundation (CNCF) which is a sub-foundation under the Linux Foundation [28–30].

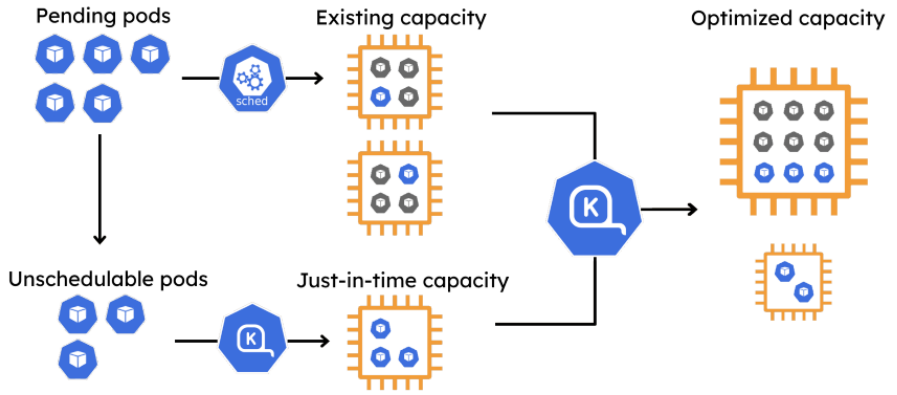

Figure 1: Karpenter observes the aggregate resource requests of unscheduled pods and makes decisions to launch and terminate nodes to minimize scheduling latencies and infrastructure cost. Image and caption licensed under Apache License 2.0 from *Karpenter Documentation* [27].

1.2 RESEARCH OPPORTUNITIES

Most carbon-aware systems only reduce carbon emissions from operational electricity consumption, thereby leaving out embodied carbon emissions [16–18, 20, 23, 25, 31–38]. This is problematic, as some data centers' carbon footprint is dominated by embodied emissions [8, 39]. Minimizing embodied carbon incentivizes the supply chain to choose carbon-efficient suppliers, reduce its own operational emissions, and increase product lifetime [39].

Much of the existing research in this area includes only the carbon impact of servers, excluding the impact of networking infrastructure and object storage. This thesis will align with this, and treat networking infrastructure and object storage as separate topics. Thus focusing only on the carbon impact of servers.

Many carbon-aware systems limit their scope to short-running workloads or jobs, leaving out long-running and continuously running workloads [17, 25, 36]. This is a significant weakness, as short-running workloads may only account for less than 10 % of overall utilization [40, p. 8].

Carbon-aware schedulers adjust the compute resource demand to be high when energy is green and low when energy is dirty. However, the choice of server largely determines how much energy is drawn to meet the resource demand. In the context of Kubernetes, one would use a cluster autoscaler to optimize servers for carbon efficiency.

Although some carbon-aware and carbon efficient systems developed by researchers improve upon existing technologies [16, 17], none are integrated into cluster management tools that Kubernetes Administrators already use. If carbon-aware and carbon efficient systems are to succeed, they must become widely adopted throughout the industry, and thus be easy to enable by Kubernetes administrators. This means that carbon awareness and carbon efficiency ideally should be features in existing systems that Kubernetes Administrators use.

In conclusion, there is an absence of research in systems that (1) optimize both operational and embodied emissions, (2) support all types of workloads, and (3) can easily be enabled by Kubernetes Administrators.

1.3 PROBLEM FORMULATION

How can Karpenter be improved to minimize carbon emissions from Kubernetes clusters that run on cloud infrastructure? Does it make sense to minimize both operational and embodied carbon emissions using Karpenter? How much does this improvement change the aforementioned carbon emissions?

1.4 THESIS STRUCTURE

The purpose of this thesis is to answer the problem formulated in section 1.3. To accomplish this, the thesis is organized as follows. In chapter 1 the problem is introduced and formulated. Chapter 2 provides an examination and critical evaluation of existing research, literature, and methodologies in the field. Chapters 3 to 6 contextualizes the proposed solution by informing the reader about some fundamentals of green software, BoaviztAPI, cloud computing, and Karpenter. In principle, chapters 3 to 6 can be skipped if the reader is familiar with

the respective topics. Nevertheless, it is recommended to read chapters 3 to 6 as the subsequent chapters are difficult to grasp without the knowledge presented herein. A solution is designed and implemented in chapter 7 and evaluated in chapter 8. The findings are discussed in chapter 9, and future research is outlined in chapter 10. Finally, the thesis is concluded in chapter 11. A curious reader might find the elaborations presented in the appendices interesting, but the thesis is assumed to be read without the appendices. The bibliography is conveniently located at the end of the thesis on page 125, making it easy to look up the sources cited throughout the thesis.

The second question in the problem formulation (regarding *scope, i.e. minimize operational and/or embodied carbon*) is answered through analysis in section 4.5. The first question (regarding *how*) is answered in chapter 7 wherein a solution design and implementation is presented. Lastly, the third question (regarding *measurement and quantification of improvement*) is answered in chapter 8 by evaluating the proposed solution through experimentation and analysis.

2 STATE OF THE ART

In this chapter, state-of-the-art research papers and software about carbon-aware and carbon efficient cloud computing are covered. Each section focuses on a specific work, describing its purpose, methodology, and results. The chapter concludes with a summary, clarifying how the present thesis contributes to the field's advancement.

2.1 CARBON-AWARE COMPUTING FOR DATACENTERS

In this paper (Mar. 2023) [36], Radovanović *et al.* introduce Google's global, in-production system for Carbon-Intelligent Compute management, called Carbon-Intelligent Computing System (CICS). CICS is built to reduce grid carbon emissions from Google's data centers' electricity use by temporally shifting flexible jobs. The paper describes the methods and principles behind CICS and evaluates its impact [36].

In this paper, a data center has a scheduler that schedules tasks onto servers in a data center. CICS creates and maintains a Virtual Capacity Curve (VCC) that virtually constrains the data center's capacity to run flexible jobs as illustrated in fig. 2. The VCC is an input to the scheduler. The CICS forecasts day-ahead carbon intensity for the electricity grid for the data center campus, and use this forecast to lower the VCC when carbon intensity is high. The scheduler schedules flexible tasks for timespans where the data center has a high capacity, i.e. when the carbon intensity is low.

The CICS takes multiple requirements and Service Level Objectives into account when calculating the VCC. The VCC depends on the relationship between compute usage and power consumption – which they find to be accurately described using a piecewise linear model.

The CICS is tested on a Google Campus (data center), and shows a 1–2 % reduction of energy use during daily hours where carbon intensity is at its highest. It performs best when there is a high level of flexible jobs and when carbon forecasting is accurate.

An interesting discovery was that the flexible compute conversation condition fails to hold when they allow for large drops in the VCC. This is because interaction with other systems that ensure reliability may cause tasks to spontaneously move execution to a data center with a spatially different location in response to lower VCCs. This can be described as an unanticipated occurrence of spatial shifting, and may increase or decrease carbon emissions.

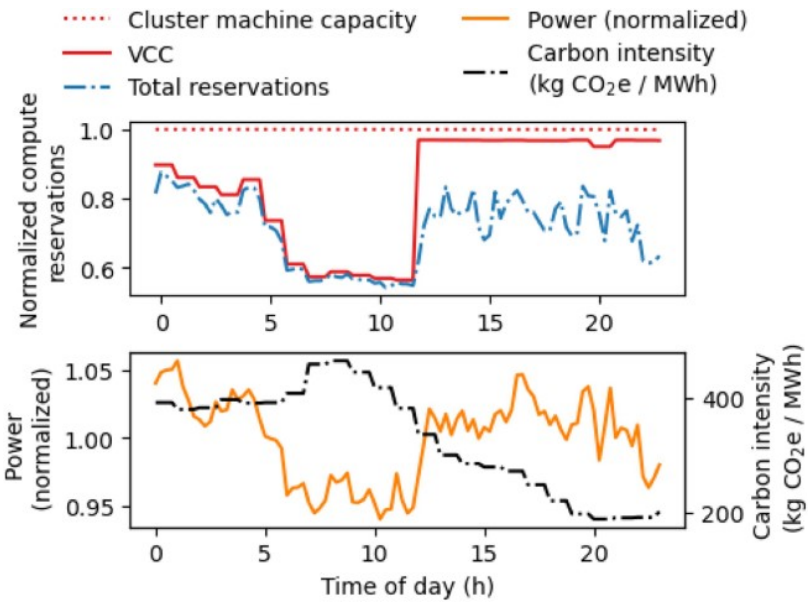

Figure 2: Hourly Virtual Capacity Curve (VCC), compute reservations, cluster power, and carbon intensity in a cluster on a selected day. From [36, Fig. 6].

2.2 MEASURING IT CARBON FOOTPRINT: WHAT IS THE CURRENT STATUS ACTUALLY?

In this preprint (Jun. 12, 2023) [41], Kennes summarize current methods for measuring software's carbon footprint. First, he divides Greenhouse Gas emissions into four elements: (1) Software Energy Consumption (E_s), (2) Server Overhead Energy Consumption (E_o), (3) Momentaneous Energy Mix (I), and (4) Embodied Carbon Emissions (M). He then splits these elements into two groups: operational carbon emissions (E_s, E_o, I) and embodied emissions (M). The preprint then highlights a formula for calculating the total carbon emissions based on the four elements:

$$TotCarbon = ((E_s + E_o) \cdot I) + M. \tag{1}$$

Equation (1) is the Software Carbon Intensity (SCI) as proposed by The Green Software Foundation without the functional unit (R) [21, 41, 42]:

$$\frac{TotCarbon}{R} = SCI = (E \cdot I) + M \quad \text{per} \quad R. \tag{2}$$

For each of the four elements, the preprint details the current status and possibilities for the purpose of "aligning open-source initiatives, practitioners, and academic research" [41, p. 2].

Current methods to quantify Software Energy Consumption are Power Monitoring Counters, simulation models based on computer architecture, and the Digital Environmental Footprint model from

NAME	UNIT	DESCRIPTION
$TotCarbon$	$kgCO_2e$	Total carbon impact.
E_s	$kW\,h$	Software Energy Consumption.
E_o	$kW\,h$	Server Overhead Energy Consumption.
I	$kgCO_2e/kW\,h$	Momentaneous Energy Mix.
M	$kgCO_2e$	Embodied Carbon Emissions.
R	None	Functional unit. E.g. amount of users or devices.
SCI	$kgCO_2e/R$	Software Carbon Intensity.

Table 1: Units for eqs. (1) and (2).

The Sustainable Digital Infrastructure Alliance. Power Monitoring Counters is often the most accurate, but since it is not always possible to directly measure power in the cloud, Digital Environmental Footprint is a useful estimate [41].

According to Kennes, Power Usage Effectiveness (PUE) is the most used method for reporting the Server Overhead Energy Consumption. However, industry experts and researchers advocate for using additional metrics to nuance the picture. Other relevant metrics are [41]:

- Carbon Usage Effectiveness
- Water Usage Effectiveness
- Energy Re-Usage Factor
- IT Equipment Efficiency
- On-site Energy Matching

The Momentaneous Energy Mix can be estimated using commercial providers of average and real-time data (like ElectricityMaps [22] and Wattime [43]) [41].

Calculating Embodied Carbon can be considered a field of its own, rooted in Life Cycle Assessments [41]. Embodied Carbon involves attributing the emitted carbon related to manufacturing of the resource, to the user of the resource, making the customer responsible for the emissions [41]. The preprint concludes, that in practice it is extremely difficult to get reliable data on Embodied Carbon [41], and therefore in most cases, estimates must be used.

2.3 ON THE PROMISE AND PITFALLS OF OPTIMIZING EMBODIED CARBON

In the aforementioned preprint by Kennes it was assumed (perhaps unknowingly) in eq. (1) that embodied and operational carbon emissions were mathematically compatible and could be combined into a single metric. However, according to Bashir *et al.* that might not be the case.

In this conference paper (Aug. 2, 2023) [39], Bashir *et al.* discuss four pitfalls that should be considered when optimizing for embodied carbon. Before diving into the pitfalls it is important to understand that embodied carbon emissions are "just" someone else's operational carbon emissions. The following example should make this evident.

Let's say that a company $\mathcal{B}$ sells hot water in a bottle and wants to reduce its operational emissions. Instead of warming the water using an electric stove, $\mathcal{B}$ might outsource water heating to their water supplier, company $\mathcal{A}$; so $\mathcal{A}$ now delivers hot water to $\mathcal{B}$. As a result, the carbon emissions associated with water heating has been moved from $\mathcal{B}$'s operational emissions to $\mathcal{A}$'s operational emissions. When $\mathcal{B}$ buys the hot water from $\mathcal{A}$, the carbon emissions associated with water heating are now $\mathcal{B}$'s embodied emissions. If $\mathcal{B}$ sells bottles with hot water to company $\mathcal{C}$, then $\mathcal{B}$'s operational and embodied emissions become the embodied emissions of $\mathcal{C}$. As illustrated, operational emissions become "embodied" in the product they are used for, and when the product is sold, the emissions are inherited by the buyer, who accounts the emission source as "embodied" (in bought products).

The four pitfalls of optimizing embodied carbon, as described by Bashir *et al.*, are:

COMPOUNDING INACCURACY It can be challenging to accurately and verifiably estimate operational carbon emissions. Therefore, when embodied emissions are accumulated through the supply chain, errors and inaccuracies are accumulated. So the estimation inaccuracy for embodied emissions is larger than that for operational emissions.

DOUBLE-COUNTING CARBON When embodied carbon is reduced, the supply chain can also claim reductions in their operational or embodied carbon. That means multiple companies can claim the same carbon reductions. The aggregate carbon reductions in a supply chain are therefore much larger than the actual reductions.

EMBODIED > OPERATIONAL Most technology companies have very long and complex, interconnected supply chains. Because of that, their embodied emissions will almost always be much larger than their operational emissions. However, that does not

imply that reducing embodied emissions is more important than reducing operational emissions, because our operational emissions are somebody else's embodied emissions.

COMBINING METRICS Embodied and operational carbon is measured on multiple scales because of double-counting embodied emissions. Therefore, the two are incompatible and cannot be combined in a single metric in a meaningful way.

Note that, this does not mean that it is unimportant to reduce embodied carbon. Although embodied emissions are difficult to account for, optimizing embodied emissions still motivates the supply chain to become more carbon-efficient and avoids some emissions.

Bashir *et al.* highlight that it would be extremely effective to introduce a global carbon tax, resulting in the alignment of economic and sustainability incentives. However, in the absence of such carbon tax, minimizing embodied carbon can provide an incentive (although much weaker) for carbon optimizations throughout the economy [39].

2.4 A TESTBED FOR CARBON-AWARE APPLICATIONS AND SYSTEMS

In this preprint (Jun. 16, 2023) [31], Wiesner *et al.* attacks the problem that almost all new carbon-aware approaches are evaluated on self-implemented testbeds, leading to low comparability of approaches. To solve this, Wiesner *et al.* envision and design a co-simulation testbed for carbon-aware applications and systems. A prototype of the testbed called Vessim is developed and open-sourced [44]. Vessim is still under heavy development.

2.5 READY FOR RAIN? A VIEW FROM SPEC RESEARCH ON THE FUTURE OF CLOUD METRICS

In this report (2016) [45], Herbst *et al.* propose metrics for benchmarking features in cloud systems. The elasticity metrics they specify are useful for evaluating autoscalers: under/over-provisioning accuracy, under/over-provisioning timeshare, and jitter. These metrics have been used in many peer-reviewed research articles [46–50].

2.6 DIGITAL & ENVIRONMENT: HOW TO EVALUATE SERVER MANUFACTURING FOOTPRINT, BEYOND GREENHOUSE GAS EMISSIONS?

In this blog post (Nov. 26, 2021) [51], the Boavizta Working Group capture and describe a journey they have been on. A journey to define a formula that can accurately estimate the environmental impact of component manufacturing for data centers (embodied carbon). The

blog post is divided into three iterations that build on each other to improve the formula, showing the rationale leading to the final formula.

The Boavizta Working Group start by highlighting that most literature discussing the environmental impact of data centers focus on the environmental impact of energy consumption. Cloud infrastructure providers only report direct (scope 1) and indirect (scope 2) greenhouse gas emissions linked to energy consumption, and exclude supply-chain emissions (scope 3) [51]. The Boavizta Working Group cites "Chasing Carbon" [8] which confirms the importance of indirect emissions from manufacturing of servers [51].

2.6.1 *First Iteration: Identifying Impacting Variables*

In the first iteration, the Boavizta Working Group analyze data sheets from manufacturers called Product Carbon Footprint or Product Environmental Footprint. Few data sheets on product footprints are available, resulting use of data from only selected servers from Dell and HP. Each Product Carbon Footprint or Product Environmental Footprint publication formalizes the data in different ways. Greenhouse gas emissions related to manufacturing are not directly reported, and thus they had to use a percentage and multiply it with the reported total.

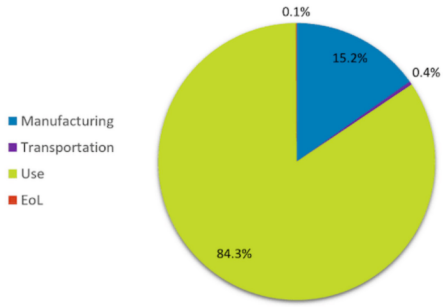

Figure 3: Breakdown of Dell R740 (a specific Dell server) lifecycle carbon footprint in Europe [51, 52]. EoL means End of Life.

Using this approach results in the first formula, which is very simple [51]:

$$server_{manuf_{gwp}} = server_{total_{gwp}} \cdot server_{manuf_{ratiogwp(\%)}}. \tag{3}$$

When analyzing the manufacturer's data they found that the breakdown between embodied emissions (manufacturing) and operational emissions (use) was approximately 20 % embodied emissions and 80 % operational emissions in Europe, and 45 % embodied and 55 % operational in France. This confirmed the importance of measuring embodied carbon in countries with low-carbon electricity like France.

They also found that the margins of error displayed by the manufacturers on overall carbon impact were very high at approximately 100 %. This suggests that an iterative approach to estimate embodied carbon is viable, starting with approximating component impacts and using more precise factors when they become available [51].

Based on these observations, the Boavizta Working Group arrive at a new formula for rack servers that has a 4 % error [51]:

$$\begin{aligned} server_{manuf_{gwp}}[\mathrm{kgCO_2e}] =& 850\,\mathrm{kgCO_2e} \\ &+ CPU_{units} \cdot 150\,\mathrm{kgCO_2e/unit} \\ &+ ram_{size} \cdot 3\,\mathrm{kgCO_2e/GB}. \end{aligned} \tag{4}$$

This formula is only based on data provided by Dell and HP which is not representative for the configurations of servers running in hyperscale data centers [51].

2.6.2 *Second Iteration: Defining Emission Factors per Component*

To increase the accuracy, Boavizta searched for studies and data allowing them to evaluate the environmental footprint of semiconductor manufacturing [51]. The "component" approach led them to a study called *Life-Cycle Assessment of Semiconductors* by Boyd wherein she states that "carbon emissions associated with the manufacture of high-density electronic components account for the majority of the emissions associated with server manufacturing" [53]. Other studies that the second iteration relied on includes *Product Carbon Footprints* [54], *The Environmental Footprint of Logic CMOS Technologies* [55], and *Life Cycle Assessment of Dell R740* [56]. These publications were analyzed to assess the impact of each component and identify semiconductors with a significant impact on the carbon footprint [51].

They compared two similar machines using data from the same manufacturer (Dell) but from two different studies: *Product Carbon Footprint of PowerEdge R740* and *Life Cycle Assessment of Dell R740*. The result from the studies was disturbingly different (45 % uncertainty). Based on this, the authors conclude that given the current data (2021), they had to accept a significant margin of error in the carbon impacts published by manufacturers [51]. Because of this, it seemed acceptable to use *somewhat* arbitrary emission factors, and so eq. (4) was updated to eq. (5):

$$\begin{aligned} server_{manuf_{gwp}}[\mathrm{kgCO_2e}] =& 900\,\mathrm{kgCO_2e} \\ &+ CPU_{units} \cdot 100\,\mathrm{kgCO_2e/unit} \\ &+ RAM_{size} \cdot \frac{150\,\mathrm{kgCO_2e}}{128\,\mathrm{GB}} \\ &+ SSD_{size} \cdot 100\,\mathrm{kgCO_2e/GB} \\ &+ HDD_{size} \cdot 50\,\mathrm{kgCO_2e/GB} \\ &+ GPU_{units} \cdot 150\,\mathrm{kgCO_2e/unit} \end{aligned} \tag{5}$$

Although eq. (5) is more detailed, it still does not integrate environmental criteria other than the Global Warming Potential, and the emission factors are highly inaccurate [51].

2.6.3 *Third Iteration: Multi-Criteria Impact Calculation*

The focus of the third iteration is the manufacturing of semiconductors and multi-criteria impact analysis.

Semiconductors are the basic element in integrated circuits, and high-density semiconductors are the main source to greenhouse gas emissions from server manufacturing [53]. In particular, *wafer* manufacturing and chip assembly is a large contributor to this. See fig. 4 for an overview of the lifecycle of an integrated circuit.

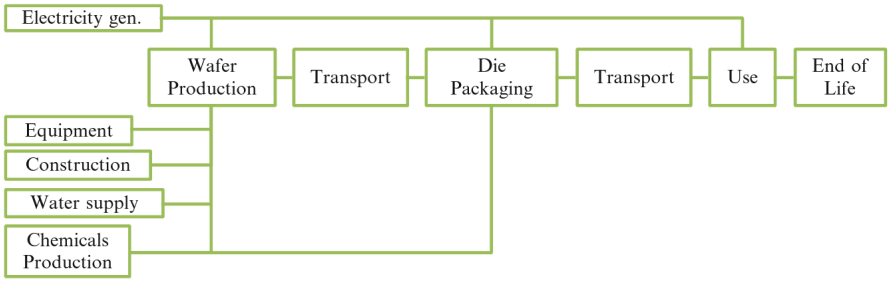

Figure 4: Life-cycle stages of an integrated circuit [53, Fig. 2.1].

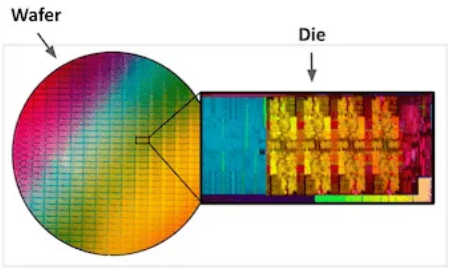

Figure 5: Wafer and die [51].

A *wafer* is a very thin slice of semiconductor material such as crystalline silicon. Wafer manufacturing is a long and complex process involving doping, etching, and photolithography. A wafer is cut to obtain a set of *die*, which is a square of silicon containing the microstructures forming an integrated circuit. The size of the die area is an essential piece of information when estimating the environmental impact of CPU, RAM, and SSD components. For each of these components, the Boavizta Working Group has developed a method to estimate the die area. For example, to estimate the die area for a CPU, one should know the underlying microarchitecture (lithography) or processor family along with the number of physical cores on the processor. For memory components like RAM and SSD, the die area is determined by the storage density [51].

The second part of the third iteration revolves around a report by Gröger *et al.* called *Green Cloud Computing: Lifecycle-based data collection on the environmental impacts of cloud computing* [57]. The report proposes a bottom-up approach to impact analysis, i.e. the impact of a server is the sum of the impact of its parts. The report also proposes a multi-criteria life-cycle analysis using Global Warming Potential, Primary Energy, and Abiotic Depletion Potential indicators. Lastly, the report is well documented and very transparent on the source for impact factors. Based on this report, the Boavizta Working Group proposes a multi-criteria evaluation of the environmental impacts of a server. The formula is made generic to support multiple impact indicators by replacing *criteria* with the desired indicator [51]. Equation (6) shows the general formula, and eq. (7) shows the formula for the RAM. Formulas for other components were produced by the authors and many emissions factors were reported, however, these are omitted in this summary.

$$\begin{aligned} server_{manuf_{criteria}} =& CPU_{manuf_{criteria}} + RAM_{manuf_{criteria}} \\ &+ SSD_{manuf_{criteria}} + HDD_{manuf_{criteria}} \\ &+ motherboard_{manuf_{criteria}} + PSU_{manuf_{criteria}} \\ &+ enclosure_{manuf_{criteria}} + assembly_{manuf_{criteria}} \end{aligned} \tag{6}$$

$$\begin{aligned} RAM_{manuf_{criteria}} =& RAM_{units} \cdot ((RAM_{size}/RAM_{density}) \\ &\cdot RAM_{manuf\_die_{criteria}} + RAM_{manuf\_base_{criteria}}) \end{aligned} \tag{7}$$

The result when using this iteration's formula (eq. (6)) is consistent with the life-cycle analysis of Dell's R740 server ([56]) regarding the total impact and the impact of each component [51]. The emission factors estimated by the authors corroborate Boyd's findings on the importance of the *die* area for the impact of server components. Another finding is that RAM is considerably more important than CPU when evaluating the Global Warming Potential criteria. However, the component impacts are not correlated across criteria, meaning that a multi-criteria evaluation is essential to avoid impact transfers when optimizing for just one indicator.

The article ends with some final remarks: The digital domain is deeply immature when it comes to environmental impact assessment. Very few data exist, and those provided by manufacturers are poorly documented, based on a single criterion, and not transparent. However, as shown, it *is* possible to perform a multi-criteria assessment of a server's embodied emissions based on a bottom-up approach [51].

Since the release of this blog post in 2021, Boavizta has further developed the methodology to increase accuracy and made it accessible through an API known as BoaviztAPI.

2.7 SUMMARY

In this chapter, various works and methodologies have been examined, providing an understanding of the current state of the art.

Various methodologies have been proposed for estimating the carbon impact of software systems [31, 39, 41, 51], and multiple works integrate a carbon impact into compute management systems using carbon awareness [31, 36] (and in section 1.1: [16, 17, 23, 25]). Later in this thesis we will revisit the Boavizta methodology which has evolved into BoaviztAPI, and explore how it can be leveraged in the proposed solution.

Generally, the field proves to be rather young and immature (but evolving) as scientific consensus is yet to be reached [39, 41]. Nevertheless, this thesis aims to advance this field by introducing and exploring the promising but uncharted concept of *carbon efficient cluster autoscaling*.

3 FUNDAMENTALS OF GREEN SOFTWARE

Green Software is a large and complex field at the intersection of many disciplines, including software engineering, climate science, data center design, electrical engineering, carbon accounting, and chip design. Although the scope of Green Software is incredibly wide, its definition is relatively straightforward: "Green Software is carbon efficient software" [21]. Carbon Efficient software aims to "emit the least amount of carbon possible" [21], but that doesn't mean the goal is to emit zero carbon. Instead, the aim is to use carbon efficiently when creating value. For example, to provide an unchanged service level at a reduced carbon impact is to increase carbon efficiency. Or using another example, it would not be carbon efficient to emit 65.4 $MtCO_2e$ by computing hashes of arbitrary numbers. That is what the Bitcoin mining network does, and its yearly impact (65.4 $MtCO_2e$) is larger than the country-level emissions in Greece (56.6 $MtCO_2e$ in 2019) [58]. Therefore, carbon efficiency is about maintaining value while minimizing carbon emissions. Based on this interpretation of carbon efficiency, a definition of carbon efficient cluster autoscaling is proposed.

Definition 1 (Carbon Efficient Cluster Autoscaling)
Autoscale a cluster to meet its needs, while emitting the least amount of carbon possible.

Three activities improve carbon efficiency: energy efficiency, hardware efficiency, and carbon awareness (as shown in fig. 6) [21]. This chapter explains these activities and covers a couple of concepts that will be used throughout this thesis.

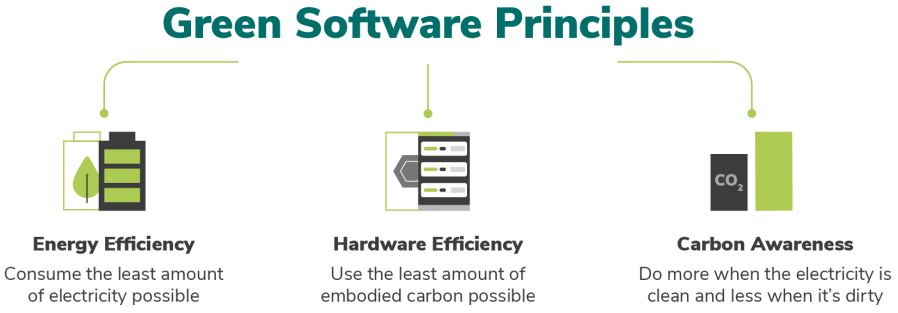

Figure 6: Green Software Principles (CC BY 4.0) [21].

3.1 ENERGY EFFICIENCY

Energy is the primary unit for work and is measured in joule (symbol: J). The electricity in the electrical grid is energy. Electrical energy (or power) is a secondary energy e.g. being converted from chemical energy when coal is burned or from kinetic energy when wind turbines rotate. In this way, electricity is energy, and energy is a measure of the electricity used. Electricity is measured in watt ($1\,W = 1\,J/s$), kilowatt, or kilowatt-hour ($1\,kW\,h = 3.6\,MJ$). In most cases, carbon is emitted when electricity is produced to meet a demand. Therefore, we say that energy - or electricity - is a proxy for carbon. Since software applications run on servers that consume electricity (by proxy, emit carbon) it is important that software as a whole is energy efficient.

The operational carbon impact of software applications is primarily based on the electricity (by proxy, carbon emissions) that is necessitated by the software application.

In this section, a couple of relevant concepts will be explained.

3.1.1 *Power Usage Effectiveness*

When you have tens of thousands of servers running inside a building, it gets very hot because some of the electrical energy is converted to heat. High server temperatures reduce server performance and reliability. Therefore, data centers deploy cooling mechanisms to keep the temperature down and performance up. When considering the energy use of software, one should also include the energy use it necessitates, such as cooling and other supporting overheads. Power Usage Effectiveness (PUE) describes how large this overhead is.

$$PUE = \frac{\text{Total Facility Power}}{\text{IT Equipment Power}} \tag{8}$$

A PUE of 1.5 means that for every $1\,kW\,h$ used by IT equipment, $0.5\,kW\,h$ is used by the facility to power supporting equipment. A lower PUE is better and a PUE of 1 is optimal. The PUE has become a de facto standard [41, 59], and a metric that many cloud providers publish. The PUE for Amazon Web Services (AWS), Google Cloud Platform (GCP), and Azure can be seen in table 2. Therefore, to get the energy that an application or server necessitates, one would multiply the energy it uses with the data center's PUE.

The PUE metric has increased competition among cloud providers and motivated data center operators to drive up efficiencies. However, despite the wide adoption, it is heavily criticized [41, 59, 61]. Some of the issues highlighted in the literature are:

- PUE values can not be directly compared [59].

CLOUD PROVIDER	PUE
AWS	1.135
GCP	1.1
Azure	1.185

Table 2: Power Usage Effectiveness (PUE) for three large cloud providers [60].

- The PUE can be reduced by making IT equipment less energy efficient [59].
- The PUE does not reflect the meteorologic climate the data center is in. It is obviously easier to keep servers cold in Finland than in Mexico [41].

3.1.2 *Thermal Design Power*

Thermal Design Power (TDP) is measured in watts (W), and is the power consumption of a component under maximum load [62]. When performance is not maxed out, the power consumption is lower than the TDP. Typically, producers publish the TDP of a chip (e.g. CPU) in a data sheet along with other details. Modern CPUs used in hyper-scale data centers typically has a TDP of 220 W to 385 W [63]. Data center operators use the TDP to understand what power draw to design the data center for. However, chips can consume more power than their specified TDP during turbo [62] or if the operating frequency is permanently increased beyond the base frequency. For that reason, Intel has renamed TDP to *Processor Base Power* since their 12th generation of CPUs [62, 64]. Additionally, Intel is publishing *Maximum Turbo Power* which specifies the power draw when turbo is sustained for >1 s. Yet, the Maximum Turbo Power can still be exceeded for ≤ 1 ms [64, 65].

Nevertheless, the TDP is a useful piece of information when modeling the power consumption of a component as a function of its utilization. It is used in various works to assist quantification of the operational carbon impact of servers and software [41, 57, 63].

3.1.3 *Energy Proportionality*

Energy proportionality measures the relationship between power consumption and utilization [21]. A system is *energy proportional* if there is linearity between utilization and power consumption and no constant factors (e.g. static power draw) [61]. Modern servers are far from energy proportional. In fact, they have the highest energy efficiency at the highest utilization. Unfortunately, servers are rarely fully utilized and instead spend most time in an area with sub-optimal energy efficiency. Increasing energy proportionality can substantially reduce en-

ergy consumption, by improving the energy efficiency in areas where servers spend most time. Yet, linearity is not necessarily the optimal relationship between power consumption and utilization when optimizing for energy savings [61].

3.2 HARDWARE EFFICIENCY

In the context of Green Software, *hardware efficiency* is to "use the least amount of embodied carbon possible" [21]. In this section, the concept of embodied carbon is explained, and two generic approaches to improve hardware efficiency are introduced.

3.2.1 *Embodied Carbon*

Embodied carbon is the carbon that is emitted when creating and disposing a piece of hardware, and it is probably one of the most fundamental concepts in Green Software. It is measured in $kgCO_2e$. Embodied carbon indicates how carbon-intensive the manufacturing process has been, and because it encapsulates value chain emissions, it is accounted as Scope 3 emissions in the Greenhouse Gas Protocol [21, 39, 66]. One study estimates that most emissions from modern data center equipment come from embodied carbon (manufacturing and infrastructure) [8].

Software and technology are some of the modern economy's most complex products, as they have very long and complex supply chains [39]. As a result, estimating embodied carbon is difficult and often comes with large uncertainties. Despite this, several works attempt to estimate embodied carbon with varying accuracy [41, 51, 60, 63, 67].

Because embodied carbon is emitted before and after usage, but not during usage, its carbon impact is usually amortized over the device's lifespan. For example, if a server's embodied carbon is $1000\,kgCO_2e$ and it has an expected lifespan of four years, then amortizing its embodied carbon over its lifespan means allocating $250\,kgCO_2e$ to each year of usage or $29\,gCO_2e$ to each hour (assuming linear allocation).

3.2.2 *Improving Hardware Efficiency*

Improving hardware efficiency is about using embodied carbon efficiently and getting the most "bang" for each $kgCO_2e$ of embodied carbon. The two main approaches to improving hardware efficiency are described below.

3.2.2.1 *Extending Lifespan*

Since embodied carbon is usually amortized over the device's lifespan, extending the lifespan can reduce the embodied carbon per unit

of time. For personal computers, that means taking care of them, for example, by avoiding physical damage, using antivirus software, repairing broken parts, etc. The typical lifetime for a server in a data center is 3-4 years [61, p. 131]. For consumers of virtual cloud machines, it is practically impossible to extend the lifespan of hardware, as it is simply owned by cloud providers and controlled by data center operators.

3.2.2.2 *Increasing Utilization*

When talking about cloud computing, hardware efficiency is best improved by increasing hardware utilization, because it reduces the number of servers that are needed. For example, it is more hardware efficient to use two servers at 100 % utilization than to use four at 50 % utilization. This is unfortunately conflicting with the widespread position among Site Reliability Engineers that servers should be somewhat over-provisioned. Cloud providers market the public cloud as an environmentally sustainable option compared to the private cloud because the public cloud's scalability diminishes the need for over-provisioning, thereby allowing organizations to run servers at a higher utilization [68, 69]. This is somewhat true, but it is not the whole story.

3.3 CARBON AWARENESS

Carbon awareness is based on a rather simple principle: *do more when the electricity is cleaner and do less when the electricity is dirtier* [21]. In this context, *aware* means that something is informed about its carbon impact, and regulates its behavior based on that information. This section introduces a couple of concepts that are essential to carbon awareness and describes how a system can be carbon aware.

3.3.1 *Carbon Intensity*

Carbon intensity is a measure of how many grams of carbon-equivalents that are emitted per kilowatt-hour of electricity consumed. Thus, carbon intensity is measured in gCO_2e/kWh [21]. Most energy grids distribute energy from a mix of power sources like wind, solar, nuclear, coal, oil, hydro, and gas. Each source produces electricity at a specific carbon intensity; for example, the carbon intensity of coal is $880\,gCO_2e/kWh$ [70] which is extremely high. The carbon intensity for an energy grid is a weighted *average* depending on each source's carbon intensity and how much electricity it contributes to the energy mix. This is called *location-based* carbon intensity.

Most cloud providers use another approach to calculate carbon intensity which is called *market-based* carbon intensity. Market-based

carbon intensity measures the carbon intensity of the electricity that is *purchased* – instead of the carbon intensity of the electricity that is *consumed*. Market-based carbon intensity is complicated by the existence of Power Purchasing Agreements (PPAs). A PPA is a long-term contract between an electricity producer and a customer about purchase of energy at a pre-negotiated price. PPAs allow consumers to claim that they use renewable energy while being powered by electricity from a grid that contains non-renewable sources. PPAs are private transactions, and details are therefore rarely shared with grid operators. This leads to double counting of renewable energy [10]. Some PPAs does not necessitate a spatial and temporal relation between the electricity produced and the electricity consumed by the buyer of the PPA. For example, a data center in Germany can make a (private) PPA with a hydropower operator in Brazil for electricity in July. Using this PPA, the data center in Germany can claim to run on 100 % renewable energy in December. For this reason, market-based carbon intensity is not relevant for the purpose of this thesis. Throughout the rest of this thesis, the term *carbon intensity* will refer to the location-based approach unless noted otherwise.

Currently, the most reliable way to get real-time (hourly) carbon intensity data is through a commercial data provider like ElectricityMaps or WattTime. In reality, this data is not an exact measurement of carbon intensity, but just a best-effort estimate based on public data power generation. Carbon-aware systems usually rely on real-time carbon intensity to efficiently perform temporal shifting (explained later). Some works may just rely on a constant carbon intensity based on historical data or simple models like the Weighted Moving Average. Using inaccurate but simple models can have multiple advantages:

- Avoids monetary costs.
- Avoids relying on the availability of an external data provider.
- Improve execution time by removing network dependencies.
- Defers integration with a real-time data provider to a point in time where data is more accurate and verifiable.

To provide an intuition about carbon intensities, the values for some countries are presented. The electricity in Sweden in 2022 on average had an extremely low carbon intensity of 7 gCO_2e/kWh [71] being powered primarily by nuclear, wind, and hydro. The electricity in the USA in 2020 on average had a carbon intensity of 350 gCO_2e/kWh [72], and the world average in 2020 was 436 gCO_2e/kWh [73].

Electricity demand varies throughout the day. In Denmark, for example, it typically peaks around 6 p.m. when many households cook dinner. Electricity production must constantly be in balance with the

demand to avoid a power outage. Specifically, production must neither be lower nor higher than the demand. Most renewable energy sources can not be regulated to increase their power output. For example, one can not make the wind blow harder at will. Energy sources powered by fossil fuel, on the other hand, are more easily controlled to increase power production; that is, they are *dispatchable*. Conversely, if power sources produce more electricity than can be used, that electricity is "thrown away" to avoid a power outage; that is called *curtailment* [21].

Fossil-fueled power plants rarely scale down to zero because they have a minimum operating threshold and may be difficult to turn back on once fully powered down. For this reason, a scenario can occur where renewable energy is curtailed to balance production and demand. In practice, that is sometimes observed when the wind blows so hard, that some wind turbines are stopped (curtailed) to reduce energy production to balance production and demand. Let's imagine that someone thinks it is a waste that the wind turbines are stopped when it is finally windy. To avoid renewable energy being curtailed, this someone could increase his or her demand. For example, by turning on a power-hungry server and training a machine learning model. In that example, a wind turbine would be unblocked (dispatched) to match the new demand created by the server. The power that is dispatched, to meet a new demand is called marginal power, and we say it comes from a marginal power plant. In this example, the marginal carbon intensity would be very low because any new demand will be matched by the renewable energy that is being curtailed.

A heavily discussed question has been whether one should use marginal or average carbon intensity when calculating the carbon impact of software [41]. On one hand, the marginal carbon intensity captures exactly the cause-effect relationship of your software's energy demand. On the other hand, if everyone can claim to use marginal carbon intensity, then the very real impact and contribution from fossil-fueled power plants will never be accurately represented in the estimated carbon impact of software. Therefore, whenever you hear someone say that the carbon impact of their software is this or that, a very relevant question is: "Do you use marginal or average carbon intensity?" Luckily, however, the Green Software community seems to finally have reached a consensus on this – average carbon intensity should be used [23, 36, 41, 74]. This appears mainly to be of practical reasons, since it is hard to identify the marginal energy source, and the decision of a power supplier to scale production up or down is decentralized and usually incentivized by electricity prices [23].

The next two subsections cover how carbon awareness can reduce the carbon footprint of software.

3.3.2 *Demand Shifting*

To be carbon aware is to reduce the energy demand when carbon intensity is high. One way to do that is to *shift* demand from high carbon intensity to low carbon intensity. Shifting demand in time is called *temporal shifting*. An example of that is to postpone non-time-critical workloads when carbon intensity is high and then resume these workloads when carbon intensity is low. Demand can also be shifted geographically, which is called *spatial shifting*. An example of that is moving workloads from a data center in Germany to one in Sweden. Demand shifting can be a good approach to mitigate the curtailment of variable renewable energy. One study estimates that demand shifting of workloads in data centers in California in 2019 could have absorbed up to 62 % of the curtailed variable renewable energy in California that year [75]. That corresponds to eliminating 239 ktCO_2e or the carbon footprint of 22 760 californians [76].

3.3.2.1 *Spatial Shifting*

Spatial shifting is a carbon-aware demand-shifting method that typically can reduce the long-term carbon impact of software substantially. For example, one can permanently move workloads from Ireland to Sweden to achieve a large carbon impact reduction. Another approach is to "follow the sun around the world", assuming a high share of solar power in the energy grids. One can also move workloads between the northern and southern hemispheres depending on the season [21]. Spatial shifting is mostly relevant when network latency is not a concern. Some works that implement spatial shifting are: [36, 37].

3.3.2.2 *Temporal Shifting*

Temporal shifting can reduce the carbon impact of non-time-critical workloads. It does, however, require that the near-team carbon intensity can be accurately predicted. For example, if you choose to postpone a workload that should be done within 24 hours, it is a good idea to be sure that the carbon intensity will be lower than it currently is within the next 24 hours. Some works that implement temporal shifting are: [18, 23, 25, 36].

3.3.3 *Demand Shaping*

Demand *shaping* is similar to demand shifting, but with a notable difference. Instead of moving demand, demand is "just" increased or decreased based on carbon intensity. An example of demand shaping is an AC-powered computer (no battery) that reduces CPU frequency when carbon intensity is high and increases the frequency when car-

bon intensity is low [21]. Features can also be turned on or off at select levels of carbon intensity.

3.4 CARBON REDUCTION TERMINOLOGY

When organizations intend to reduce carbon emissions, multiple mechanisms are available (shown in fig. 7). As we will see, it is important to understand the differences between the mechanisms, as they require very different efforts and vary in reliability and cost-effectiveness. After reading this section, you will know what each term refers to and why some mechanisms are more appropriate than others in the green software domain.

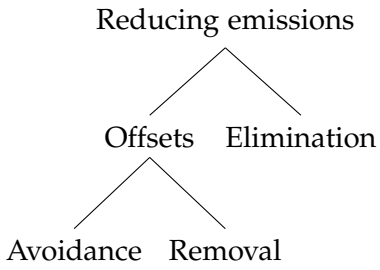

Figure 7: Logical relationship between carbon reduction mechanisms.

3.4.1 *Offsets*

Carbon offsets refer to a reduction in greenhouse gas emissions and are used to compensate for emissions that occur elsewhere [21]. A carbon offset credit is a transferable instrument certified by governments or independent certification bodies to represent an emission reduction of one ton of carbon [77]. Carbon offset credits can be bought by an organization to allow the subtraction of emissions in their carbon accounting. An organization or a product can become *carbon neutral* if all emissions are offset with carbon offset credits, but it can only be used for some emissions when reaching for net zero.

Carbon offsets do not reduce emissions from products per se, but only when we consider products as part of a larger system. Therefore, it is considered inappropriate to include carbon offsets in metrics that are utilized to compare the sustainability of software systems. If carbon offsets were used in such a metric, two identical applications could have different emission values. That would be misleading when considering which *engineering* efforts to apply. That is also why carbon offsets are not included when calculating the Software Carbon Intensity score of an application.

In the following two subsections, we will quickly explore two mechanisms for offsetting carbon.

3.4.1.1 *Carbon Avoidance*

Carbon avoidance, also called *compensating*, refers to actions that organizations take to avoid emissions outside of their value chain [21]. It can be thought of as investing in other organizations' abatement projects, such as forest *conservation*.

3.4.1.2 *Carbon Removal*

Carbon removal, also called *neutralizing*, is when carbon is pulled out of the air after it has been emitted, basically attempting to undo emissions [21]. This could be through Carbon Capture and Storage or forest *restoration* projects.

3.4.2 *Carbon Elimination*

Carbon elimination, also called *abatement*, is when emissions are eliminated from an organization's own operations or value stream so that they do enter the atmosphere [21]. When we as engineers, rather than accountants, put an effort into reducing emissions it is often elimination. Examples of carbon elimination efforts are reducing a server's power draw, improving software efficiency to handle more users without increasing resource utilization, or switching to on-site renewable energy. Carbon elimination is the only way to drive down the Software Carbon Intensity [21]. The focus of this thesis is to reduce carbon emissions through elimination.

3.5 QUANTIFICATION OF CARBON EMISSIONS

When minimizing the carbon impact of cloud instances (servers), it is good to know their carbon impact. Ideally, cloud providers would disclose accurate, granular, and reliable carbon emission data about cloud servers. Unfortunately, that is not the case. Microsoft Azure and Google Cloud Platform appear to genuinely attempt to provide a useful tool to estimate carbon emissions while AWS provide very incomplete tools[1] [78]. No cloud provider guarantees that the disclosed data is accurate. For example, Microsoft disclaims any responsibility for the appropriateness, accuracy, reliability, and correctness of its Cloud Sustainability API [79]. Because of that, only transparently produced carbon emission estimates are used in this thesis. The methodology is described in chapter 4.

1 The AWS Carbon Calculator has no transparency, is primarily based on operational carbon emissions, and uses a market-based carbon intensity (including Power Purchasing Agreements). This results in incorrect carbon accounting and could lead organizations to overstate their progress towards sustainability goals [10].

3.6 CLIMATE COMMITMENTS

In this section, some fundamental climate science terminology will be covered.

3.6.1 *The Greenhouse Gas Protocol*

The Greenhouse Gas Protocol is a comprehensive international standardized framework for measuring and managing greenhouse gases [21]. In this thesis, it is mostly used for the three scopes that it divides

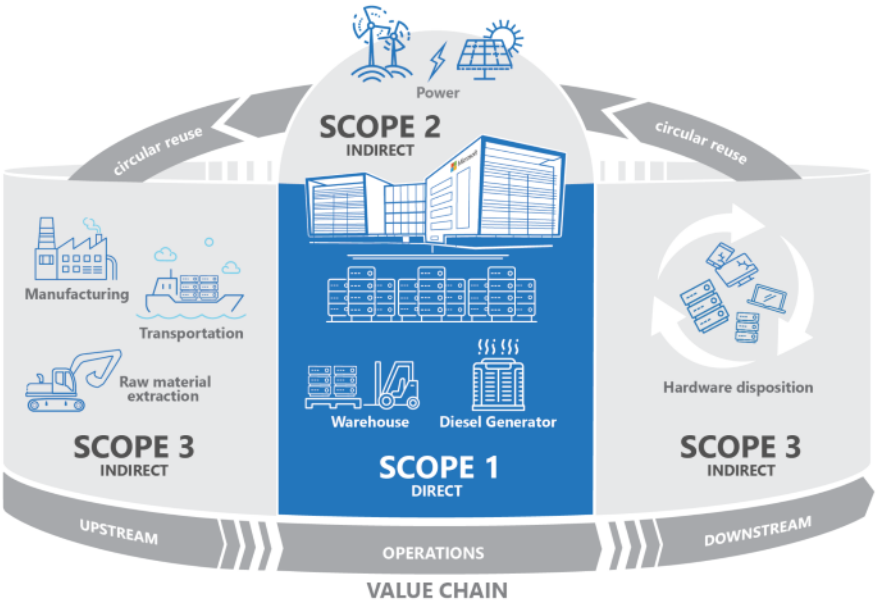

Figure 8: An illustration of the the Greenhouse Gas Protocol scopes from the view of a data center operator like Microsoft Azure [66].

greenhouse gas emissions into:

SCOPE 1 contains *direct* emissions within a company. That is, for example, the burning of fossil fuels or emissions from operating company-owned vehicles. IT organizations typically have no or very, very limited Scope 1 emissions.

SCOPE 2 contains *indirect* emissions from the production of *purchased* electricity and heat used by an organization. This typically constitutes a large share of cloud providers' emissions. Software companies that run their software on others' platform typically have a low amount of Scope 2 emissions.

SCOPE 3 contains *indirect* emissions occurring as a result of an organization's activities, but in a way so they are not controlled by the organization or are part of Scope 2. Scope 3 often contains value-chain and embodied emissions. For example, if a

company buys a laptop for an employee, the greenhouse gases that other organizations have emitted to produce that laptop fall within our Scope 3. Typically, Scope 3 contains the largest portion of emissions for software companies.

Most often, organizations are only required to account for Scope 1 and Scope 2 emissions. This can be an issue for accountability, as Scope 3 often contains the largest portion of emissions. The result of this is that by outsourcing operations, emissions can be accounted as "reduced", while in reality they have just been moved to another organization.

3.6.2 *Commitment Terminology: Carbon Neutral & Net Zero*

There is an important distinction between the effort required to be *carbon neutral* and having *net zero* emissions. The difference between the two concepts is clarified:

CARBON NEUTRAL To become carbon neutral, an organization must measure its emissions and *reduce* Scope 1 and 2 emissions. The usage of the word *reduce* in this context means that it is allowed to offset all carbon emissions to claim carbon neutrality. And most often, carbon offsetting is all that is done. No efforts toward carbon elimination are required.

NET ZERO It is harder to have net zero emissions than it is to be carbon neutral. Having net zero emissions means that emissions are reduced according to the latest climate science – that is to make reductions that limit global warming to 1.5 °C at the end of the century. In essence, this means to *eliminate* nearly all carbon emissions *and neutralize* the few remaining emissions by 2050.

To limit global warming to 1.5 °C with limited overshoot, global net zero CO_2 (notice the missing *e*) must be reached in early 2050 [1, p. 20]. Global warming in year 2010–2019 is at 1.1 °C [1, p. 4] and is projected to reach 3.2 °C at the end of the century [1, p. 22]. It is *likely* that global warming will exceed 1.5 °C in the near term [1, p. 12].

4

FUNDAMENTALS OF BOAVIZTAPI

BoaviztAPI can assess the environmental impacts induced by Information and Communications Technologies and expose them through an API. The BoaviztAPI source code is public on GitHub and is licensed under AGPL-3.0 [63]. The API is stateless and only depends on static data that is included in its container image. While this thesis focuses on carbon emissions, the environmental impact of Information and Communications Technology is unfortunately not limited to that; hence, BoaviztAPI also provides other impact criteria like Use of Water Resources and Abiotic Depletion Potential.

In this chapter, we will explore how BoaviztAPI estimates the carbon impact (also called Global Warming Potential) of cloud instances. This is highly relevant as BoaviztAPI will be used in the solution in chapter 7 and the evaluation hereof. Thereby, the proposed solution inherits BoaviztAPI's advantages, weaknesses, and limitations.

The methodology in BoaviztAPI is based on a report from Jun. 2021 commissioned by the German Federal Environment Agency (Umweltbundesamt) called *Green Cloud Computing: Lifecycle-based data collection on the environmental impacts of cloud computing* [57]. The report proposes a bottom-up approach to carbon impact estimation. Using this approach a server's carbon impact is the sum of the carbon impact of its parts. BoaviztAPI uses a bottom-up approach to carbon impact estimation. This mitigates some of the pitfalls listed by Bashir *et al.* in "On the Promise and Pitfalls of Optimizing Embodied Carbon."

This chapter is structured so that the most general and abstract formulas are presented first. These formulas are then deconstructed into more specific and detailed formulas. The description of BoaviztAPI in this chapter is based on version `1.1.0` of the documentation and source code [63, 80].

In contrast to mathematics, where variables often are expressed using a single symbol, variables in programming are often written as a word or a sequence of words. BoaviztAPI is a program, but in this chapter, its functionality is expressed mathematically. To improve readability, variables are deliberately written as words in this chapter.

4.1 CLOUD INSTANCE

The goal is to estimate the carbon impact of a cloud instance, which is defined as

$$cloudInstance = cloudInstance_e + cloudInstance_o, \tag{9}$$

where $cloudInstance_e$ is the embodied impact and $cloudInstance_o$ is the operational impact. This impact is defined for the lifetime of the cloud instance, so the unit is $kgCO_2e$. However, as cloud instances are rented for a time period and not bought, it is more appropriate to estimate the carbon impact of a cloud instance *per hour*. This is done by dividing the carbon impact by its lifetime T:

$$cloudInstance_{\text{rate}} = cloudInstance / T. \tag{10}$$

Since it is very difficult to know the exact lifetime of a server, the lifetime is assumed to be 4 years (35 040 h) for all devices. This means that both embodied and operational carbon emissions are allocated linearly.

NAME	UNIT	DESCRIPTION
$cloudInstance$	$kgCO_2e$	Total impact of a cloud instance over its lifetime.
$cloudInstance_e$	$kgCO_2e$	Embodied impact for a cloud instance.
$cloudInstance_o$	$kgCO_2e$	Operational impact for a cloud instance.
$cloudInstance_{\text{rate}}$	$kgCO_2e/h$	Rate of carbon impact for a cloud instance.
T	h	Estimated instance lifetime. Default: 35 040 h (4 years).

Table 3: Units for eqs. (9) and (10).

4.2 EMBODIED IMPACT

The embodied impact is the carbon impact of manufacturing and production. Since a cloud instance is a fraction of a server, it is defined as:

$$cloudInstance_e = \frac{server_e}{cloudInstancesPerServer}. \tag{11}$$

The embodied impact of a server is defined as

$$server_e = \sum_{\{components\}} component_e, \tag{12}$$

which specifically means that

$$\begin{aligned} server_e =& CPU_{\text{units}} \cdot CPU_e \\ &+ RAM_{units} \cdot RAM_e \\ &+ SSD_{units} \cdot SSD_e \\ &+ HDD_{units} \cdot HDD_e \\ &+ PS_{units} \cdot PS_e \\ &+ motherboard_e \\ &+ assembly_e \\ &+ enclosure_e. \end{aligned} \tag{13}$$

NAME	UNIT	DESCRIPTION
$cloudInstance_e$	$kgCO_2e$	Embodied impact for a cloud instance.
$server_e$	$kgCO_2e$	Embodied impact for a server.
$cloudInstancesPerServer$	None	The amount of cloud instances per server. The amount can be a fraction, e.g. 2.5.

Table 4: Units for eqs. (11) and (12).

In the following subsections, the embodied impact of each component in eq. (13) will be defined.

4.2.1 *CPU*

The embodied impact of the CPU is defined as

$$\begin{aligned} CPU_e =& CPU_{die_e} \cdot (CPU_{core_{units}} \cdot CPU_{die_{size/core}} + 0.491\,\text{cm}^2) \\ &+ CPU_{base_e}, \end{aligned} \tag{14}$$

where $CPU_{die_e} = 1.97\,kgCO_2e/cm^2$ and $CPU_{base_e} = 9.14\,kgCO_2e$.

4.2.2 *RAM*

The embodied impact of the RAM is defined as

$$RAM_e = (RAM_{capacity}/RAM_{die_{density}}) \cdot RAM_{die_e} + RAM_{base_e}, \tag{15}$$

where $RAM_{die_e} = 2.2\,kgCO_2e/cm^2$ and $RAM_{base_e} = 5.22\,kgCO_2e$.

4.2.3 *SSD*

The embodied impact of the SDD is defined as

$$SSD_e = (SSD_{capacity}/SSD_{die_{density}}) \cdot SSD_{die_e} + SSD_{base_e}, \tag{16}$$

NAME	UNIT	DESCRIPTION
CPU_e	$kgCO_2e$	Embodied impact of the CPU component.
CPU_{die_e}	$kgCO_2e/cm^2$	Embodied impact of the CPU die.
$CPU_{core_{units}}$	None	Number of physical cores per CPU.
$CPU_{die_{size/core}}$	mm^2	Size of the CPU die area per core.
CPU_{base_e}	$kgCO_2e$	Baseline impact of a CPU.

Table 5: Units for eq. (14).

NAME	UNIT	DESCRIPTION
RAM_e	$kgCO_2e$	Embodied impact of the RAM component.
$RAM_{capacity}$	GB	RAM capacity.
$RAM_{die_{density}}$	GB/cm^2	RAM die density.
RAM_{die_e}	$kgCO_2e/cm^2$	Embodied impact of the RAM die.
RAM_{base_e}	$kgCO_2e$	Baseline impact of a RAM component.

Table 6: Units for eq. (15).

where $SSD_{die_e} = 2.2\,kgCO_2e/cm^2$ and $SSD_{base_e} = 6.34\,kgCO_2e$.

NAME	UNIT	DESCRIPTION
SSD_e	$kgCO_2e$	Embodied impact of the SSD component.
$SSD_{capacity}$	GB	SSD capacity.
$SSD_{die_{density}}$	GB/cm^2	SSD die density.
SSD_{die_e}	$kgCO_2e/cm^2$	Embodied impact of the SSD die.
SSD_{base_e}	$kgCO_2e$	Baseline impact of a SSD.

Table 7: Units for eq. (16).

4.2.4 *HDD*

The embodied impact of a HDD is a constant:

$$HDD_e = 31.10\,kgCO_2e. \tag{17}$$

NAME	UNIT	DESCRIPTION
HDD_e	$kgCO_2e$	Embodied impact of the HDD component.

Table 8: Units for eq. (17).

4.2.5 *Power Supply (PS)*

The embodied impact of a power supply is defined as

$$PS_e = PS_{weight} \cdot PS_{weight_e}, \tag{18}$$

where $PS_{weight_e} = 24.30\,kgCO_2e/kg$.

NAME	UNIT	DESCRIPTION
PS_e	$kgCO_2e$	Embodied impact of the Power Supply component.
PS_{weight}	kg	Weight of a power supply.
PS_{weight_e}	$kgCO_2e/kg$	Embodied impact of a power supply per unit of weight.

Table 9: Units for eq. (18).

4.2.6 *Motherboard*

The motherboard's embodied impact is a constant:

$$motherboard_e = 66.10\,kgCO_2e. \tag{19}$$

NAME	UNIT	DESCRIPTION
$motherboard_e$	$kgCO_2e$	Embodied impact of the motherboard component.

Table 10: Units for eq. (19).

4.2.7 *Assembly*

Assembly is the final process of assembling the components into a server. The embodied impact of the assembly process is a constant:

$$assembly_e = 6.68\,kgCO_2e. \tag{20}$$

NAME	UNIT	DESCRIPTION
$assembly_e$	$kgCO_2e$	Embodied impact of the assembly component.

Table 11: Units for eq. (20).

4.2.8 *Enclosure*

The impact of the enclosure depends on the type used for the server. If it is a rack server then

$$enclosure_e = rack_e = 150\,kgCO_2e, \tag{21}$$

but if it is a blade server then

$$enclosure_e = blade_e + blade_{enclosure_e}/16, \tag{22}$$

where $blade_e = 30.90\,kgCO_2e$ and $blade_{enclosure_e} = 880.00\,kgCO_2e$.

Servers at Amazon Web Services (AWS) use a rack enclosure.

NAME	UNIT	DESCRIPTION
$enclosure_e$	$kgCO_2e$	Embodied impact of the enclosure component.
$rack_e$	$kgCO_2e$	Embodied impact from the rack enclosure for a server.
$blade_e$	$kgCO_2e$	Base blade impact.
$blade_{enclosure_e}$	$kgCO_2e$	Blade enclosure impact.

Table 12: Units for eqs. (21) and (22).

4.3 OPERATIONAL IMPACT

As with the embodied impact defined in eq. (11), the operational impact of a cloud instance is a fraction of a server's operational impact:

$$cloudInstance_o = \frac{server_o}{cloudInstancesPerServer}. \tag{23}$$

The operational impact for a server is given by the equation

$$server_o = E \cdot U \cdot I \cdot t. \tag{24}$$

If a server's electrical consumption, E, is known, then eq. (24) can be used without further action. Unfortunately, cloud providers rarely disclose the energy consumption of servers. Two emerging open-source

tools, Kepler and Scaphandre, attempt to circumvent this and estimate the energy consumption using advanced techniques [81, 82]. Kepler, for instance, uses eBPF[1] to probe system performance counters and uses machine learning models to convert these statistics to power-level estimates. Still, these tools are rarely deployed, so when the electrical consumption is not known, we use a consumption profile

$$E \cdot U = CP_{server}(U), \tag{25}$$

which is defined as

$$\begin{aligned} CP_{server}(U) =& (CP_{CPU}(U) \cdot CPU_{units} + CP_{RAM}(U) \cdot RAM_{units}) \\ & \cdot (1 + other) \end{aligned} \tag{26}$$

where $other = 0.33$. In the following two subsections, we will explore how the CPU and RAM consumption profiles are calculated.

NAME	UNIT	DESCRIPTION
$cloudInstance_o$	$kgCO_2e$	Operational impact for a cloud instance.
$server_o$	$kgCO_2e$	Operational impact for a server.
$cloudInstancesPerServer$	None	The amount of cloud instances per server. The amount can be a fraction, e.g. 2.5.
E	W	Electrical consumption.
I	$\frac{kgCO_2e}{kWh}$	Carbon intensity. Based on [83–85].
t	h	Duration – often just the server lifetime.
U	%	Utilization or workload.
$other$	None	Power consumption ratio of other components relative to CPU and RAM.
$CP_{server}(U)$	W	Consumption profile for a server at utilization U.

Table 13: Units for eqs. (23) to (26).

1 eBPF, put simply, is a method to safely and efficiently extend the capabilities of a kernel (e.g. Linux) at runtime.

4.3.1 *CPU*

The consumption profile for CPUs are given by the equation

$$CP_{CPU}(U) = a \cdot \ln(b \cdot (U + c)) + d. \tag{27}$$

The parameters a, b, c, and d are found by performing a lookup for a consumption profile in table 14. If the consumption profile for a CPU is unknown, then Intel Xeon Platinum's consumption profile is assumed. Figure 9 graphs the power consumption using eq. (27) and the parameters from table 14.

MODEL RANGE	a	b	c	d
Intel Xeon Platinum	171.1813	0.0354	36.8953	-10.1336
Intel Xeon Gold	35.5688	0.2438	9.6694	-0.6087
Intel Xeon Silver	20.7794	0.3043	8.4241	0.8613
Intel Xeon E5	48.9167	0.1349	15.7262	-4.6540
Intel Xeon E3	342.3624	0.0347	36.8952	-16.4022
Intel Xeon E	55.6501	0.0467	20.4146	4.24362

Table 14: Parameter values for some CPU model ranges.

If a CPU's Thermal Design Power (TDP) is known, then its consumption profile is adapted to its TDP. That is done by calculating four power-workload data points given by

$$E(U, R) = R(U) \cdot TDP, \tag{28}$$

using the preset ratios from table 15. Table 15 shows the ratio of TDP that is drawn at a workload level. Lower and upper bounds are calculated for each parameter (a, b, c, and d) using the CPU's consumption profile from table 14. Finally, BoaviztAPI use non-linear least squares to fit a new consumption profile to the power-workload data points using the base consumption profile and the bounds[2].

An example of a consumption profile that is adapted to the TDP of a CPU is the consumption profile for the Annapurna Labs Graviton3 processor. This processor has a TDP of 220 W. Figure 10 graphs its consumption profile.

2 This process is difficult to explain. If you find yourself in need of a more accurate description, reading BoaviztAPI's source code is probably the best option: `/boaviztapi/model/consumption_profile/consumption_profile.py` [63].

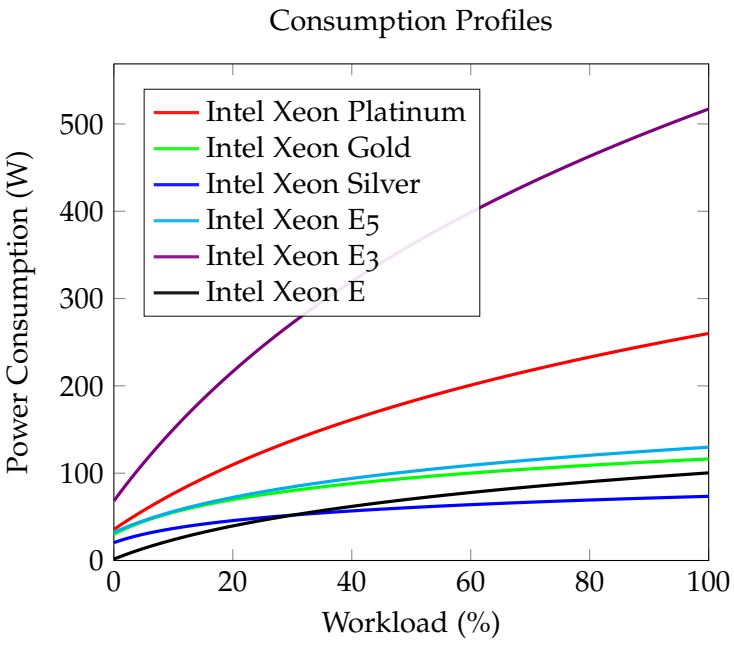

Figure 9: Consumption profiles for six CPUs.

UTILIZATION / U (%)	RATIO / R (%)
0	0.12
10	0.32
50	0.75
100	1.02

Table 15: Ratios of TDP (R) as a function of utilization (U).

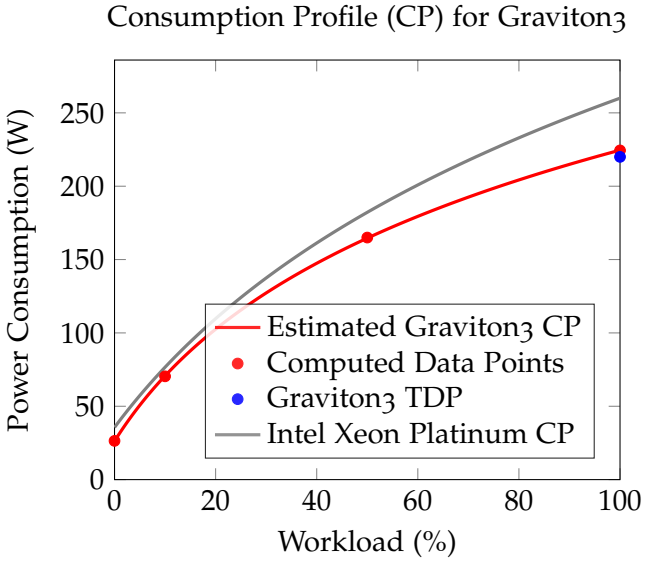

Figure 10: The consumption profile for a Graviton3 is estimated by adapting the Intel Xeon Platinum consumption profile (fig. 9 and table 14) to the 220 W TDP of Graviton3.

NAME	UNIT	DESCRIPTION
$CP_{CPU}(U)$	W	Consumption profile for a CPU at utilization U.
U	%	Utilization.
R	%	Ratio of TDP at utilization.
TDP	W	Thermal Design Power (TDP).

Table 16: Units for eqs. (27) and (28).

4.3.2 *RAM*

The consumption profile for RAM is simple. It is given by

$$CP_{RAM}(U) = z \cdot RAM_{capacity}, \tag{29}$$

where $z = 0.284\,\mathrm{W/GB}$.

NAME	UNIT	DESCRIPTION
$CP_{RAM}(U)$	W	Consumption profile for a RAM at utilization U.
z	W/GB	Constant impact factor for RAM.
$RAM_{capacity}$	GB	RAM capacity.

Table 17: Units for eq. (29).

4.4 COMPLETION

As we have seen in the two previous sections, many details about instance types, servers, and components are needed to complete the carbon impact calculation. Consumers of the BoaviztAPI are not expected to have all these details at hand. Therefore, a lot of data is baked directly into the BoaviztAPI source. This data is collected by volunteers who search the Internet for component data. Not all component data is disclosed by manufacturers. As a result, BoaviztAPI does not have all data on all components. One might be concerned that BoaviztAPI, for this reason, only supports a very limited number of cloud instances. Luckily, however, one underpinning principle in BoaviztAPI is that any valid input always should return an impact evaluation, even if the input or BoaviztAPI's database is missing some details [80]. This principle is realized by BoaviztAPI's *data autocomplete strategy*, which supports four approaches to obtain information:

COMPELETE infers the value based on user input. For example, given a CPU name of *Intel Core i7-13790F*, a TDP of 45 W can be completed.

DEFAULT uses the default value from its configuration. For example, if no location is provided by the user it can fallback to Europe.

ARCHETYPE uses a value from an *archetype*. For example, given an instance type like *c7g.2xlarge* it can lookup the cloud instance archetype and figure out that $cloudInstancesPerServer = 8$.

CHANGED changes the value to make the computation possible. For example, given a bad CPU name *Intel Cyre i7*, it can change the value to a close match in the database.

To fully understand the above approaches, one should know what data is stored by BoaviztAPI. Two datasets in BoaviztAPI that are closely related to the present thesis are described below.

AWS.CSV contains information on the AWS cloud instance archetypes (instance types). For example, it specifies the vCPU, memory (RAM), CPU model, and storage size of cloud instances.

CPU_SPECS.CSV contains details on specific CPUs. For example, it specifies operating frequency, TDP, number of cores, transistors, and die size.

This thesis has contributed component and cloud instance data to BoaviztAPI, improving its accuracy and expanding its cloud instance support significantly. More on that in chapter 7.

4.5 ON OPERATIONAL AND EMBODIED CARBON EMISSIONS

In this section, I will answer a question from the problem formulation (section 1.3): *Does it make sense to minimize both operational and embodied carbon emissions using Karpenter?*

This question will be answered using the instance types in table 18 and regions in table 19. The three regions (excluding *world*) are geographically representative and cover two edge cases. India (having a high-carbon electricity grid) and Sweden (having a low-carbon electricity grid) represent the two extremes, while the USA is in-between with its average carbon intensity. Additionally, the three regions also host hyper-scale AWS data centers. The instance types have been chosen for this analysis because they are popular and general-purpose.

3 The year where the country had the specific carbon intensity. Not the year that the study was made or the year it was published.

4 Average of all countries, not considering population or area.

INSTANCE TYPE	CPU	MEMORY	PROCESSOR
`m6g.xlarge`	4 vCPUs	16 GiB	AWS Graviton2
`m6a.xlarge`	4 vCPUs	16 GiB	AMD EPYC 7R13
`m6i.xlarge`	4 vCPUs	16 GiB	Intel Xeon 8375C

Table 18: Specifications for three general purpose instance types.

COUNTRY	CARBON INTENSITY		YEAR[3]	SOURCE
India	0.626	$kgCO_2e/kW\,h$	2020	[84]
Sweden	0.04	$kgCO_2e/kW\,h$	2019	[83]
USA	0.37	$kgCO_2e/kW\,h$	2020	[84]
World[4]	0.39	$kgCO_2e/kW\,h$	2011,19,20	[83–85]

Table 19: Carbon intensity of electricity used in BoaviztAPI. The differences in numeric precision are intentional.

If we take the three instance types from table 18, and calculate their embodied and operational impact using the world-average carbon intensity from table 19 and BoaviztAPI, then we get fig. 11. By examining fig. 11, it becomes clear that the operational impact is the main contributor to the total carbon impact in the *world* region. Furthermore, when looking at fig. 11, the embodied impact seems to be small and almost identical across the three different instance types. Because of this, a hypothesis can be created, stating that the embodied impact is insignificant and could be discarded.

Let's explore that hypothesis by comparing the embodied and operational impact for one instance type (`m6g.xlarge`) in the USA, Sweden, and India. Doing so results in fig. 12. Looking at fig. 12, one can see that the operational impact varies significantly between the three regions and that it is surprisingly low in Sweden. It even seems like the embodied impact is larger than the operational impact in Sweden. This could suggest that the hypothesis is not true.

So let's explore the case of Sweden and investigate if we should reject the hypothesis that the embodied impact is insignificant and could be discarded. Again, by calculating the embodied and operational impact of the three instance types from table 18 in Sweden using BoaviztAPI, one arrives at fig. 13. Here it is seen that the embodied impact is larger than the operational impact for all three instance types (just barely $0.01\,gCO_2e/h$ for `m6i.xlarge`). Furthermore, the embodied impact is the factor deciding which of `m6a.xlarge` and `m6g.xlarge` has a lower total carbon impact. Therefore, it seems that the embodied impact is the dominating contributor to the total carbon impact of cloud instances in regions with low carbon intensity. Thus, we should reject the hypothesis that the embodied impact is in-

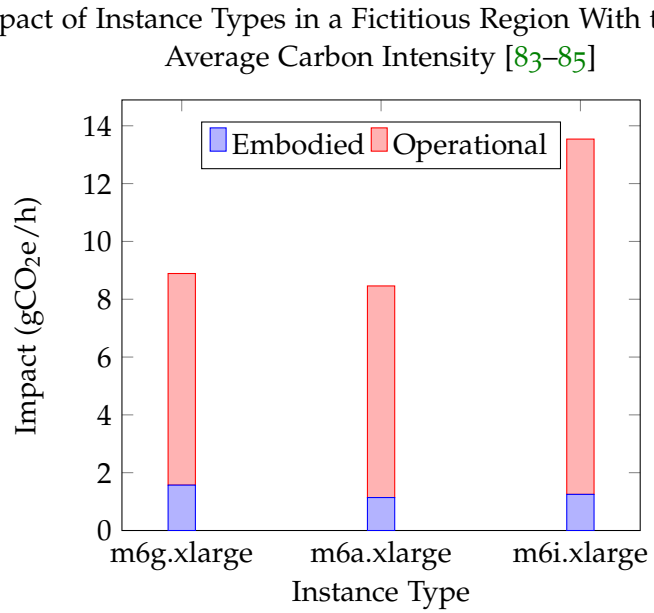

Figure 11: Key take: The operational impact is the main contributor to the total carbon impact globally.

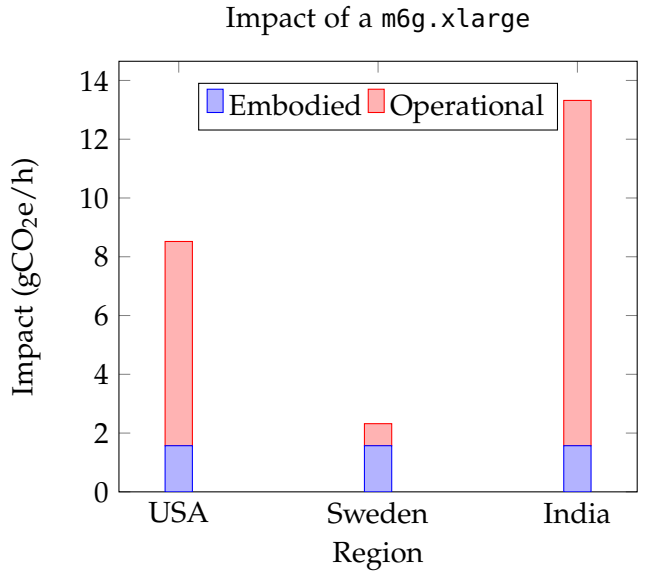

Figure 12: Key take: The operational impact of cloud instances vary significantly between regions. Huge reductions in carbon impact can be achieved with spatial shifting.

significant and could be discarded. The carbon intensity used in this section and figs. 11 to 13 is the intensity of the energy grid without accounting for potential Power Purchasing Agreements (PPAs). There are good reasons not to account for PPAs in its current form. However, if we do account for PPAs, the electricity's carbon intensity for data centers will become very low in most regions. The result of this is that the carbon intensity for many regions will be similar to that of

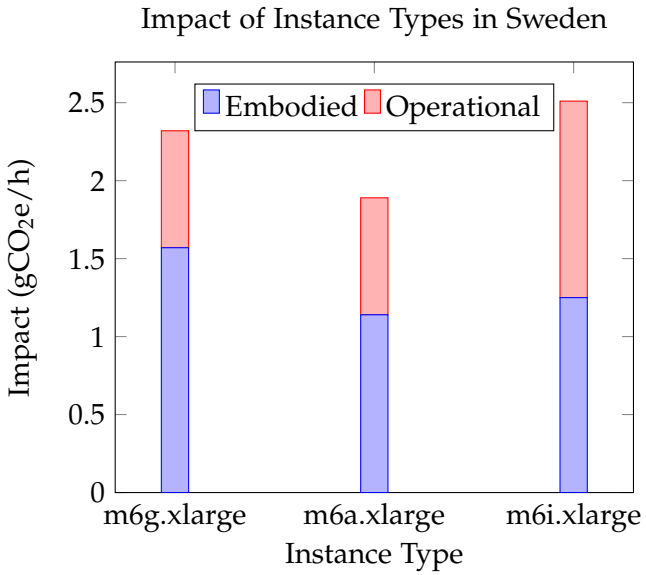

Figure 13: Key take: The embodied impact is a significant contributor to the total carbon impact of cloud instances in regions with low carbon intensity of electricity.

Sweden, implying that embodied impact will be dominant in those cases as well. Moreover, if nations around the world reduce the carbon intensity of electricity (as promised), then the operational impact will become smaller, making the embodied impact generally more dominant.

To summarize, figs. 11 and 12 show that the operational carbon impact is currently the dominating contributor to the total carbon impact in most regions. However, in regions with low carbon intensity, like Sweden, the embodied carbon impact is dominating. As more nations transition to renewable energy and lower the carbon intensity of electricity, the number of regions where the embodied carbon impact is dominating is expected to rise. In conclusion, for the solution to be suitable for various geographic regions now and in the future, it must minimize both the operational and embodied carbon impact of cloud instances.

Response to problem formulation.

5 FUNDAMENTALS OF CLOUD COMPUTING

Cloud Computing is a vast and complex field, making it impractical to cover even the fundamentals in this chapter. For that reason, this chapter covers only a carefully selected array of indispensable topics. Most of the concepts, methods, and technologies that are introduced in this chapter are vendor-agnostic. This means that abstractions have been built into open-source projects, so they can work with various cloud providers. However, since the proposed solution in this thesis is developed on Amazon Web Services (AWS), some AWS specifics will be covered.

5.1 KUBERNETES

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications [12]. Kubernetes is by far too complex and large to be fully (or even partially) covered in this section. Therefore, I will only cover the tiny bits that are necessary for understanding the chapters that follow.

5.1.1 *Pods*

Pods are the smallest deployable unit that Kubernetes can manage. A Pod is a group of containers with shared storage and network resources [12]. A Pod typically contains one container running an application. A simple Pod Spec is shown in listing 1. Pods are scheduled onto Nodes by the Kubernetes API Server. A Node is a Kubernetes concept and a type of machine. When a Node is instructed to run a Pod, it runs the Pod's containers specified in the Pod Spec with the configurations specified therein.

A Pod can specify requests and limits for CPU and memory. It is guaranteed to get as much CPU and memory as it requests, given that it is available on a Node. A Pod cannot use more CPU than the configured limit. A Pod is not allowed to exceed its memory limit, but it is technically possible for it to do so. Therefore, the memory limit is enforced by terminating Pods that exceed their limit. Both CPU and memory requests and limits are optional, and one can be configured without configuring the other. CPU is measured in *CPU* units, and one CPU is equivalent to 1 AWS vCPU, 1 GCP Core, 1 Azure vCore, and 1 Hyperthread on a bare-metal Intel processor. Fractions and *milli* (symbol m) can be used to quantify CPU ($1\,\mathrm{m} = 0.001\,\mathrm{CPU}$). Memory

is measured in bytes and can be quantified using binary and decimal prefixes such as k (1000) and Ki (1024) [12].

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
    resources:
      requests:
        cpu: "0.5"
        memory: "100Mi"
      limits:
        cpu: "1"
        memory: "200Mi"
```

Listing 1: A simple Pod Spec [12].

5.1.2 *Deployments*

A Deployment provides declarative and automated management of Pods. Applications are usually deployed as a Deployment. A desired state is described in a Deployment, and the Kubernetes Deployment Controller will attempt to make the desired state come true. Deployments can, for example, be used to scale applications horizontally, create controlled rollouts of application updates, and automatically restart crashed applications.

5.1.3 *Daemonsets*

The concept of a DaemonSet is similar to that of Deploymens, however, with one important difference. While Deployments ensure that one or more replicas of a Pod runs *somewhere*, a DaemonSet ensures that one or more replicas of a Pod runs *everywhere* – that is, it runs on all Nodes. If a new Node is added to the Cluster, the DaemonSet will ensure that a new Pod is then added to that Node. DaemonSets are typically used for log collection daemons, node monitoring daemons, and network plugins [12].

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Listing 2: A simple Deployment Spec [12].

5.1.4 *Nodes*

Nodes run Pods. Within Kubernetes, a Node can be considered a physical or virtual machine running Linux. Nodes run specialized components that enable the node to be part of a Cluster by interacting with the Kubernetes API Server and running Pods. A Cluster typically contains multiple Nodes. To *cordon* a Node means to mark it as unschedulable, preventing new Pods from being scheduled onto the Node. To *drain* a Node means to evict all Pods from a Node by terminating them and possibly restarting them on another Node.

5.1.5 *Operators*

The Operator Pattern is one of the more advanced topics in Kubernetes, so it is ok if you don't fully understand it after reading this subsection.

Kubernetes is designed to be extendable, and one of the primary methods to extend the behavior of a Cluster is to create an Operator. A common way to deploy an Operator is to add a Controller and a Custom Resource Definition to a Cluster. A Custom Resource Definition creates a Custom Resource which is a Resource that can be interacted with just like any other Resource, like a Pod or a Deployment. The Controller is usually deployed as a Deployment that

runs an executable that interacts with the Kubernetes API Server to manage a specific Custom Resource.

Karpenter is an example of an Operator because it uses the Operator Pattern to extend the behavior of a Cluster. As we will see in the next chapter, Karpenter extends a Cluster with Custom Resources, one of which is called a Provisioner. By doing this, Karpenter enables Kubernetes Administrators to add Provisioners to a Cluster in the same way that Pods can be added.

5.2 SCALING

Applications and Pods can be scaled. Usually, we talk about horizontal and vertical scaling. Horizontal scaling is to increase the number of application replicas, while vertical scaling is to increase the resources available to an application replica. In the context of Kubernetes, Cluster Autoscaling refers to horizontal and/or vertical scaling of Nodes. Various metrics can be calculated to measure how "good" autoscaling systems are [45].

5.3 AMAZON WEB SERVICES

Amazon Web Services (AWS) is a comprehensive and widely used cloud platform. It offers a vast array of cloud services including computing, storage, networking, and databases. In this section, the focus will be on two specific services, that are relevant for the subsequent chapters in this thesis: Amazon Elastic Compute Cloud (EC2) and Amazon Elastic Kubernetes Service (EKS).

5.3.1 *Amazon Elastic Compute Cloud*

Amazon Elastic Compute Cloud (EC2) is one of AWS' most popular services. It allows customers to rent virtual computers on a pay-as-you-go basis. A virtual computer on EC2 is referred to as an *instance* or a *cloud instance*. With over 750 different types of instances available, there should be an instance type for pretty much every need [86]. Table 20 shows the specifications of a few instance types. Figure 14 explains how to read the name of an AWS instance type.

In EC2 there is a concept called *EC2 Fleet*. An EC2 Fleet contains a configuration for launching a fleet (multiple) of instances across multiple Availability Zones in one API call [87].

There are a couple of instance purchasing options that should be known [87]:

ON-DEMAND Pay by the second. Default option.

Instance Type	vCPU	Memory (GiB)	Storage (GB)	Network Bandwidth (Gbit/s)
`m7g.medium`	1	4	EBS-Only	12.5
`m7g.large`	2	8	EBS-Only	12.5
`m7gd.2xlarge`	8	32	474 NVMe SSD	15
`c7g.large`	2	4	EBS-Only	12.5
`r7iz.xlarge`	4	32	EBS-Only	12.5

Table 20: An example of a few AWS instance types [86]. *EBS-Only* means that the EC2 instance must be attached to an Amazon Elastic Block Store (EBS) volume if storage is needed.

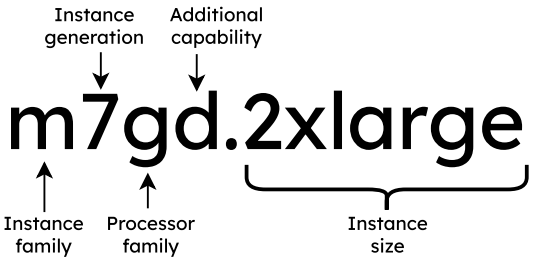

Figure 14: Instance Type syntax [87].

SPOT Pay for unused EC2 instances by the second. Much cheaper than *on-demand*, but they can be "taken away" from you if someone orders them *on-demand*.

RESERVED Reserve EC2 instances for 1 to 3 years. Reduce costs, but you commit to paying for them - even if you don't use the instances.

DEDICATED Pay by the hour for a whole server.

5.3.2 *Amazon Elastic Kubernetes Service*

Amazon Elastic Kubernetes Service (EKS) is AWS' managed Kubernetes service. Since Kubernetes Clusters are not easy to manually bootstrap, EKS makes it easier to create, upgrade, delete, and manage Kubernetes Clusters. This makes life easier for Kubernetes Administrators, but comes with a vendor lock-in. An alternative to EKS is Kuberntes Operations (kOps) [88], which was launched well before EKS.

5.3.3 *Regions and Zones*

AWS data centers exist in many locations across the globe. These locations are mainly specified by a *Region* and *Availability Zone* [86, 89]. A Region is a separate geographic area. Examples are listed in table 21. Each Region has multiple isolated Availability Zones. The code for an Availability Zone is the Region's code followed by a letter. For ex-

ample, London has Availability Zones called `eu-west-2a`, `eu-west-2b`, and so forth [86]. The idea is that geographic isolation stops effects from cascading into other Availability Zones. Applications should be replicated across multiple Availability Zones to be resilient to failures in one Availability Zone. Nodes in Kubernetes Clusters should therefore be distributed across multiple Availability Zones.

REGION NAME	CODE
Europe (London)	`eu-west-2`
Europe (Stockholm)	`eu-north-1`
US West (N. California)	`us-west-1`
Asia Pacific (Mumbai)	`ap-south-1`

Table 21: Example of a few AWS regions [86].

6 FUNDAMENTALS OF KARPENTER

As described in the problem formulation, the purpose of this thesis is to adapt Karpenter to minimize carbon emissions. Before Karpenter can be investigated for this purpose, we must learn how it works and how it is used. That is what this chapter is for.

Karpenter is a vendor-agnostic and open-source Kubernetes operator written in Go and licensed under Apache License 2.0. Karpenter consists of two parts: a "core" library that is vendor-agnostic, and a vendor-specific "provider" that uses the core library. The core library is owned by the Cloud Native Computing Foundation and mostly maintained by people employed at Amazon Web Services (AWS). Karpenter providers are currently available for Amazon Elastic Kubernetes Service (EKS) and Azure Kubernetes Service [13, 90]. Until very recently, the Karpenter provider for EKS was the only Karpenter provider available[1]. Therefore, when I write *Karpenter*, I refer to the Karpenter provider for EKS and its use of the core library.

This thesis is based on Karpenter's `v1alpha5` API version and version 0.31.0 of the EKS Karpenter provider. Karpenter is under heavy development, and new versions have been released when you read this. Concepts that are paramount to this chapter, like the `Provisioner`, have been changed and renamed in `v1beta1` [91, 92].

6.1 INSTALLATION

Karpenter is designed to run inside the Kubernetes Cluster it manages [27]. It is deployed to Kubernetes using a Helm Chart, which is a package that contains the Kubernetes resources required to deploy Karpenter [93]. For Karpenter to add and remove nodes in a cluster, it requires access to the cloud provider's infrastructure. These credentials are assumed by Karpenter using a Kubernetes Service Account that provides an AWS Identity Access Management Role.

6.2 PROVISIONING

Karpenter adds nodes to a cluster when pods are unschedulable, and removes nodes when they become unneeded [27]. To enable this functionality, a Kubernetes resource called a `Provisioner` must be created. Provisioners specify what nodes Karpenter should provision when pods are unschedulable. Karpenter can have multiple Provisioners,

1 The Karpenter provider for Azure Kubernetes Service was released during this thesis project period on November 7, 2023.

but they should be mutually exclusive so no pod matches multiple Provisioners. An example Provisioner Spec is shown in listing 3. This Provisioner sets requirements for the instance family, instance size, and region on lines 8-10, 12-14, and 16-18. It also constrains the Provisioner from creating nodes with more than an aggregate of 100 CPU on line 21. On line 25 consolidation is enabled, which means that Karpenter will try to reduce the overall cost of nodes if workloads have changed.

When pods are pending, Karpenter will batch and binpack them on the cheapest instance type based on their CPU and memory requests. Karpenter then selects 59 other instance types that are larger than binpacked instance type, because the instance type might not be available or the cheapest option. All 60 instance types are passed as an EC2 Fleet to the EC2 Fleet API which will launch the cheapest instance type available (of the 60 options).

```
1  apiVersion: karpenter.sh/v1alpha5
2  kind: Provisioner
3  metadata:
4    name: example
5  spec:
6    requirements:
7      # Only provision general purpose instances
8      - key: "karpenter.k8s.aws/instance-family"
9        operator: In
10       values: [c5, m5, r5]
11     # Exclude small instance sizes
12     - key: "karpenter.k8s.aws/instance-size"
13       operator: NotIn
14       values: [nano, micro, small, large]
15     # Only provision nodes in zone a and b in Ireland
16     - key: "topology.kubernetes.io/zone"
17       operator: In
18       values: ["eu-west-1a", "eu-west-1b"]
19   limits:
20     resources:
21       cpu: 100
22   providerRef:
23     name: default
24   consolidation:
25     enabled: true
```

Listing 3: Example of a Provisioner Spec.

6.3 DEPROVISIONING

Deprovisioning a node is to remove it from the cluster and make it available to the cloud provider's other customers. Nodes that have

been provisioned by Karpenter can have a deprovisioning triggered by (1) the Deprovisioning Controller, (2) the Kubernetes Administrator, or (3) an external system. In our context, the Deprovisioning Controller is the only relevant trigger. The Deprovisioning Controller deprovision nodes by executing one automatic method at a time in the following sequence [27]:

1. Expiration
2. Drift
3. Emptiness
4. Consolidation

The methods follow a standard deprovisioning process, but each of them serves a specific purpose. Below is a description of each method.

EXPIRATION If `ttlSecondsUntilExpired` is set in the Provisioner, then nodes will expire after the configured time. Once a node has expired, Karpenter will remove it from the cluster [27].

DRIFT If the *Drift* feature is enabled, then Karpenter detects nodes that have drifted and safely replaces them. A node has drifted if it no longer matches the configuration in a Provisioner. That can happen for multiple reasons. For example, if a node is of instance type `m5.large` and the requirements in its Provisioner are changed from

```
karpenter.k8s.aws/instance-family In [m5]
```

to

```
karpenter.k8s.aws/instance-family In [m6g]
```

the node has drifted from its specification [27, 94].

EMPTINESS When the last non-daemonset pod stops running on a node Karpenter will wait for `ttlSecondsAfterEmpty` seconds. If the timer runs out without any new pods being scheduled onto the node, then Karpenter removes the node from the cluster. This is a simple and low-risk method when nodes are de facto unneeded. However, this does probably not happen very often in clusters with daily activity as the Kubernetes Scheduler would likely schedule new pods to nodes with many available resources [27].

CONSOLIDATION If consolidation is enabled in the Provisioner, Karpenter will attempt to consolidate nodes to reduce costs. Consolidation is explained in the following section.

6.4 CONSOLIDATION

If consolidation is enabled in the Provisioner, Karpenter can consolidate the Provisioner's nodes to reduce their aggregate monetary costs. Two consolidation actions are available:

DELETE A node can be deleted if its pods can be redistributed to other nodes with free capacity.

REPLACE A node can be replaced if its pods can be redistributed to other nodes with free capacity and a cheaper replacement node.

Using these two actions, three types of consolidation can be performed:

EMPTY NODE CONSOLIDATION If the node doesn't have any pods running, it is simply deleted. The emptiness method (mentioned previously) and Empty Node Consolidation are mutually exclusive, thus only one of these can operate at a time.

MULTI-NODE CONSOLIDATION Delete multiple nodes in parallel, possibly launching a single node that is cheaper and can contain all pods from the deleted nodes. It is infeasible to examine all possible combinations of nodes that can be consolidated, so a simple heuristic is used to determine likely combinations.

SINGLE-NODE CONSOLIDATION Delete a single node, possibly launching a replacement that is cheaper. All nodes are individually examined for this consolidation.

6.5 SETTINGS

Karpenter's settings are defined in two places, each with its own set of settings: (1) environment variables or CLI parameters to the controller and (2) a ConfigMap named `karpenter-global-settings`. Among other things, the ConfigMap contains the feature gates that Karpenter uses. Currently, the only feature that is gated is `Drift`. Karpenter follows the feature gate semantics that are used in Kubernetes. A future can be in *Alpha*, *Beta*, or *General Availability (GA)* stage. Put simply, *Alpha* features are disabled by default and can be enabled, *Beta* features are usually enabled by default and can be disabled, and *GA* features are enabled by default and can not be disabled. In table 22 you can see the details of Karpenter's feature gates as of December 19, 2023.

FEATURE	DEFAULT	STAGE	SINCE	UNTIL
Drift	false	Alpha	`v0.21`	`v0.32`
Drift	true	Beta	`v0.33`	

Table 22: Feature gates in Karpenter [27].

7 DESIGN AND IMPLEMENTATION

In this chapter, I will explain how Karpenter is improved to minimize carbon emissions from Kubernetes clusters that run on cloud infrastructure. This improvement is not only enabled by design changes to Karpenter, but (especially) also by composing and improving an ecosystem of projects that efficiently can support Karpenter in minimizing carbon emissions. Therefore, this chapter will importantly describe how this ecosystem supports Karpenter, and how my contributions have been pivotal in this regard. My solution where Karpenter is improved to be carbon efficient is referred to as *Carbon Efficient Karpenter* while plain Karpenter is referred to as *Original Karpenter*.

The software in this master's thesis is available in two online Git repositories at the *thesis* tag:

- `https://github.com/JacobValdemar/carbon-efficient-karpenter/tree/thesis`
- `https://github.com/JacobValdemar/carbon-efficient-karpenter-utilities/tree/thesis`

7.1 ARCHITECTURE AND HIGH-LEVEL DESIGN

Karpenter is a Kubernetes Node Autoscaler that is good at managing nodes to improve application availability and minimize operational overhead. Improving Karpenter to minimize the carbon impact of Kubernetes Clusters requires estimating the carbon impact of each instance type. This is a process with a lot of inherent complexity. Besides satisfying the goals of the present thesis, the solution should take Karpenter's non-functional requirements into consideration. Therefore, the change should not create long-term technical debt, negatively impact performance, or add complexity that can not be justified. In conclusion, a good solution architecture enables Karpenter to minimize carbon emissions without harming Karpenter's technical quality.

To satisfy these architectural requirements, an ecosystem of software projects has been constructed to support Karpenter in minimizing carbon emissions. The ecosystem is visualized in fig. 15 and consists of three projects. The responsibilities of each project are described below.

KARPENTER manages nodes in a Kubernetes cluster to minimize their carbon impact. It uses a static dataset that maps cloud instances to carbon impacts ($kgCO_2e/h$).

BOAVIZTAPI leverages information and methodologies to calculate the carbon impact of cloud instances and expose them through an API.

CARBON QUANTIFIER generates a static dataset that maps cloud instances to carbon impacts by querying BoaviztAPI.

Carbon Quantifier importantly decouples BoaviztAPI and Carbon Efficient Karpenter by generating a static emissions map that can be used in Karpenter. This removes the need for network requests between the two. It would also complicate Carbon Efficient Karpenter's deployment mode if BoaviztAPI had to be deployed alongside Karpenter.

By detaching the complex *carbon impact estimation process* from Karpenter we significantly simplify the changes within Karpenter. This allows the solution to benefit from the existing work on BoaviztAPI. Moreover, the ongoing improvement of BoaviztAPI can effortlessly translate into ongoing improvements of Carbon Efficient Karpenter, and spare the Karpenter community from maintaining and improving the complex carbon estimation process.

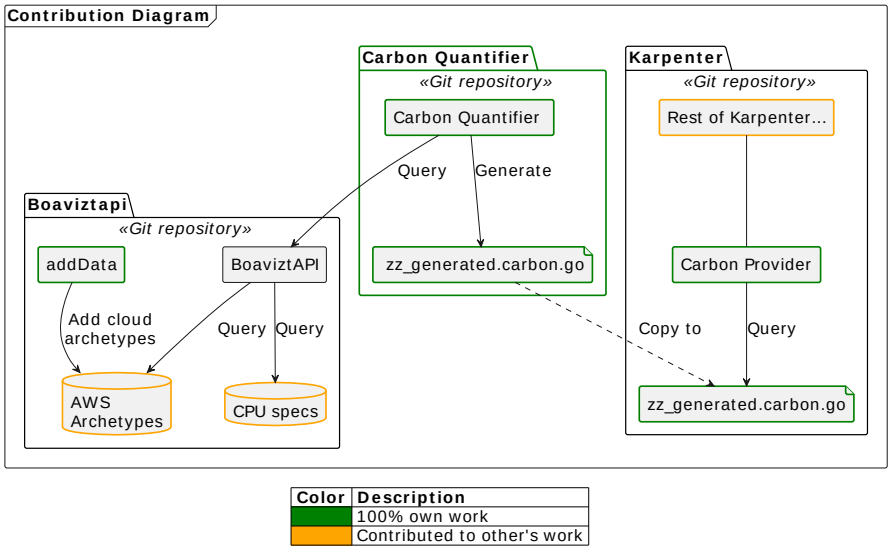

Figure 15: Illustration of the contributions from the present thesis.

7.2 KARPENTER

As mentioned in chapter 1, the proposed solution must be widely adopted to be successful, and thus be integrated into upstream Karpenter. To that end, a design document has been developed. This design document formulates and contextualizes a problem and proposes a design change to Karpenter along with four design options. Appendix A contains the design document which has been presented

to the Karpenter open-source community on GitHub, and discussed in several online Working Group meetings. The document has been adapted based on feedback and discussions with the community. The design changes have not been accepted and merged into *upstream* Karpenter at the time of writing. After the conclusion of this thesis, I will continue to advocate for upstream integration of the developed solution. Table 23 lists my open source contributions to Karpenter during this master's thesis project.

	№	💬	TITLE
PR	4686	63	docs: Add Carbon Efficient design document
Issue	675	1	RFC: Environmental Sustainability Mode / Carbon-Aware Mode

Table 23: Open source contributions to Karpenter from this master's thesis project.

A design is a story that connects a need with a technical direction that solves the need. There is no substitute for an author thinking deeply about a problem space, and mapping that to a clear story walking readers through ideas and helping them reason about a solution space. In this project, multiple designs have been proposed and discussed with the Karpenter community. Yet, this thesis covers only the one design that was moved forward with.

As the reader should be familiar with by now, the goal of this thesis is to minimize the carbon impact of Kubernetes clusters that run on cloud infrastructure. Karpenter is flexible, but it is not designed with this specific goal in mind. However, one need that understandably is part of Karpenter's design, is the need to minimize monetary costs. That fact can be used to think about our problem space in a particular way. Let's imagine that there is a monetary price of emitting carbon, e.g. €1 per $kgCO_2e$, and let's imagine that this price is the only relevant price. How could Karpenter be modified to minimize the carbon price? Answering that question is essentially the same as answering the first question in the problem formulation. The design idea is essentially to minimize carbon emissions by defining a price per $kgCO_2e$ and override the real price for cloud instances with a carbon price. To override the cloud instance price, a Carbon Pricing Provider is added to Karpenter and used instead of the original Pricing Provider. By using the *prioritized* launch strategy for EC2 Fleets and using carbon prices as priorities, carbon emissions will be minimized during provisioning. Additionally, Karpenter's consolidation strategies will (unknowingly) consolidate to minimize carbon emissions as they receive a carbon price instead of a monetary price. The structure of this is visualized in fig. 16. Listing 4 shows how the Carbon Pricing Provider uses carbon impacts to set the price.

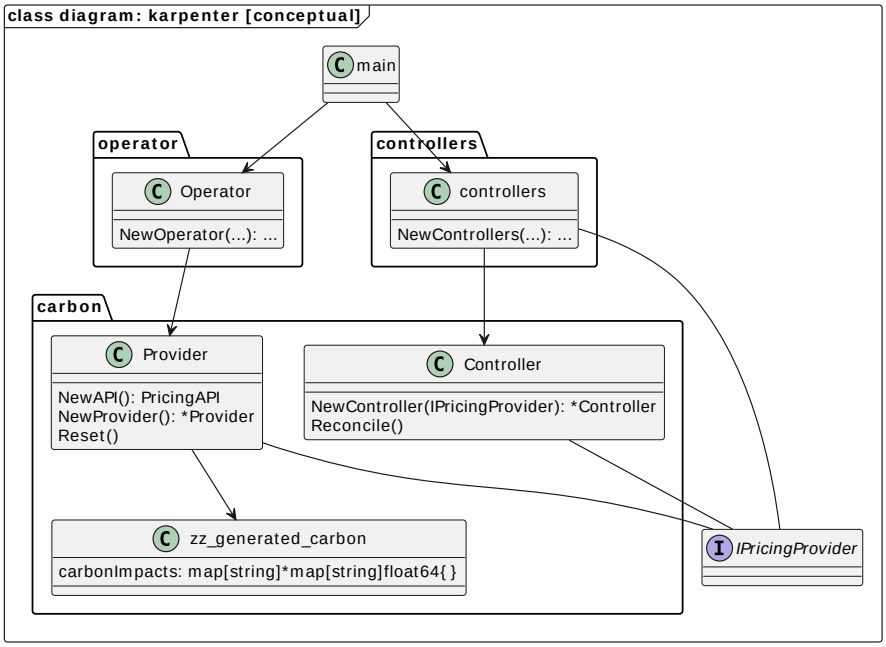

Figure 16: This conceptual diagram *visualize* relationships and structs that are essential to the change made in Karpenter. Note that Go does not have a class concept, nor support any such hierarchies or relations, so the diagram is indeed a visualization of concepts and ideas and not a documentation of the programmatic implementation.

This improvement can be activated by enabling a feature gate called `CarbonEfficient` in the `karpenter-global-settings` Config Map (see listing 5) – just like the drift feature. When the feature is enabled, the Pricing Provider is substituted with the Carbon Pricing Provider. Because no further action is required from the user, it is extremely simple to "go green" and make a Kubernetes Cluster carbon efficient.

Karpenter does not support multi-region autoscaling [95], meaning that Karpenter can not perform spatial shifting. Enabling support for multi-region autoscaling in Karpenter is out of scope for this thesis.

```
177 func (p *Provider) Reset() {
178     staticPricing, ok := carbonImpacts[p.region]
179     if !ok {
180         fallbackRegion := "eu-west-1"
182         staticPricing = carbonImpacts[fallbackRegion]
183     }
185     p.onDemandPrices = *staticPricing
```

Listing 4: `pkg/providers/carbon/pricing.go`.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: karpenter-global-settings
data:
  featureGates.carbonEfficient: true
```

Listing 5: Extract from `karpenter-global-settings` Config Map.

7.2.1 *Deployment*

From the view of a Kubernetes Administrator, the deployment and use of Carbon Efficient Karpenter is the same as with Original Karpenter. If a developer applies a Kubernetes Deployment to a cluster and the pods are unschedulable, Carbon Efficient Karpenter will create a Launch Template containing a list of instance types prioritized by their carbon impact (pricing). At Carbon Efficient Karpenter's behest, Amazon Web Services (AWS) will use the Launch Template to provision a new node. The pods are then scheduled on the new node when it is ready. This process is illustrated in fig. 17.

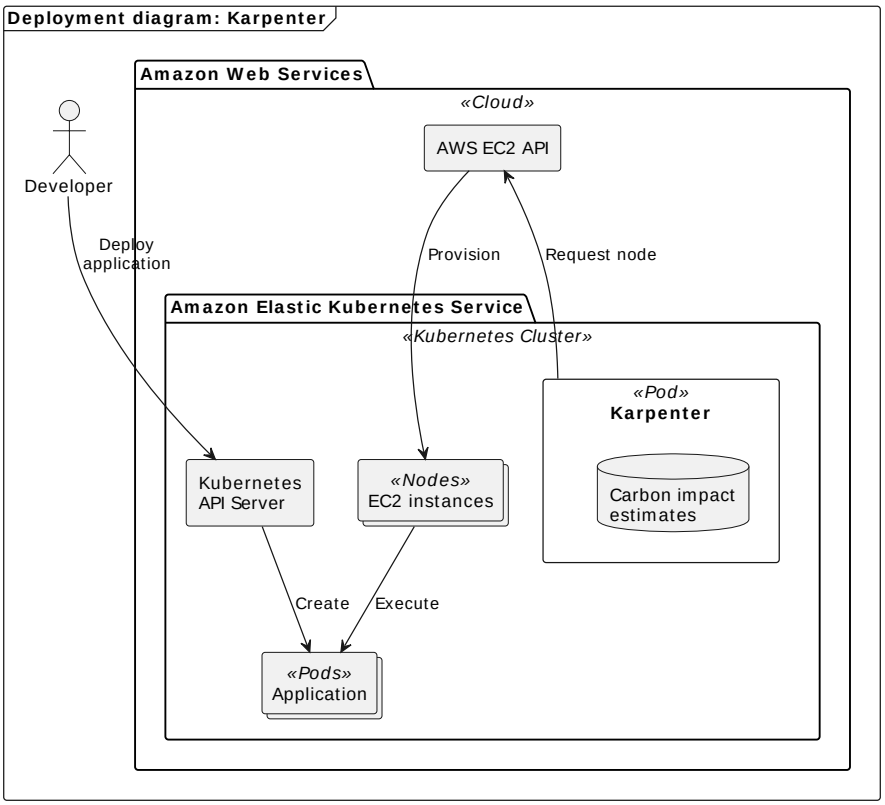

Figure 17: Deployment diagram for Karpenter.

7.3 BOAVIZTAPI

Since BoaviztAPI has been described in detail in chapter 4 this section will be brief.

For BoaviztAPI to be an adequate carbon estimate provider for Carbon Efficient Karpenter, it must have cloud instance parity with Karpenter. This means that BoaviztAPI should provide a carbon impact estimate for each cloud instance known to Karpenter. This requires BoaviztAPI to contain a cloud instance *archetype* for each AWS instance type. When the present master's thesis commenced, 336 AWS cloud instances were unsupported by BoaviztaAPI – a substantial part of Amazon Elastic Compute Cloud (EC2) instance types. To patch this gap and create instance parity, an application that collects cloud instance data has been developed (shown as the `addData` component in fig. 15). Using this application, some manual corrections, and feedback from Benjamin Davy, 336 instance types were added to BoaviztAPI's cloud instance database. This contribution made it into upstream BoaviztAPI in Pull Request (PR) 237. An array of open source contributions to BoaviztAPI (Pull Requests[1] and Issues[2]) has emerged from this thesis, as shown in table 24. The contributions ensured instance parity with Karpenter, and improved the carbon impact estimates from BoaviztAPI. As a result, BoaviztAPI is now capable of providing solid carbon impact estimates for Karpenter.

TYPE	Nº	🗪	TITLE
PR	227	1	Fix typo in aws.csv
PR	235	1	Fix: Two HDD.units columns in aws.csv
PR	237	25	Add missing aws instances
PR	239	2	Add independent Dockerfile
PR	246	3	Fix rounding USAGE.instance_per_server
Issue	232	4	Missing instance types in aws.csv
Issue	233	3	Negative CPU USE for workloads below 2%

Table 24: Contributions to BoaviztAPI from this master's thesis project.

7.4 CARBON QUANTIFIER

Carbon Quantifier is developed to decouple Karpenter and the carbon impact provider (BoaviztAPI). This follows the *dependency in-*

1 A Pull Request (PR) on GitHub is a request to merge a code change from one Git branch into another Git branch.

2 Issues on GitHub are used to track and discuss ideas, feedback, tasks, or bugs.

version principle[3] and enable Karpenter to depend on an abstraction rather than a concrete implementation. This allows an easy migration of the carbon impact provider if a better option emerges. Carbon Quantifier generates a static Go file containing carbon impacts for all instance types in all geographic regions. This data is obtained by systematically querying BoaviztAPI. An extract of the generated file is seen in listing 6. For each geographic location, every instance type is mapped to a carbon impact, and each region is mapped to a location. As a result, the generated file contains carbon impacts ($kgCO_2e/h$) on 759 instance types for each of the 22 locations mapping to 28 AWS regions. That is a total of $759 \cdot 22 = 16\,698$ carbon impacts.

```
3  package carbon
10 var carbonImpacts = map[string]*map[string]float64{}
12 func init() {
13     carbonImpacts["ARE"] = &map[string]float64{
14         "a1.2xlarge":        0.022194,
15         "a1.4xlarge":        0.044389,
16         "a1.large":          0.005549,
17         "a1.medium":         0.002774,
18         "a1.metal":          0.044389,
19         "a1.xlarge":         0.011097,
20         "c1.medium":         0.006110,
21         "c1.xlarge":         0.024786,
773    }
775    carbonImpacts["AUS"] = &map[string]float64{
776        "a1.2xlarge":        0.025116,
1535   }
16792  carbonImpacts["me-central-1"] = carbonImpacts["ARE"]
16801  carbonImpacts["ap-southeast-2"] = carbonImpacts["AUS"]
16802  carbonImpacts["ap-southeast-4"] = carbonImpacts["AUS"]
16805 }
```

Listing 6: `zz_generated.carbon.go`. `carbonImpacts` maps locations and instance types to a carbon impact ($kgCO_2e/h$) based on calculations from BoaviztAPI. ARE is the contry code for United Arab Emirates and AUS is the contry code for Australia.

Listing 7 shows the main code for the Carbon Quantifier.

3 The Dependency Inversion Principle states: (1) High level modules should not depend upon low level modules. Both should depend upon abstractions. (2) Abstractions should not depend upon details. Details should depend upon abstractions [96].

```
28 func main() {
29     allInstances := getAllInstances()
30     allInstances = filterInstances(allInstances)
33     var locations map[string][]string = map[string][]string{
37         "IND": {"ap-south-1", "ap-south-2"},
46         "IRL": {"eu-west-1"},
51         "SWE": {"eu-north-1"},
56     }
57
58     var result []ImpactList
59
60     for location := range locations {
62         var impacts []Impact
63
64         for _, instance := range allInstances {
65            embodied, operational := getImpact(instance, location)
66            impacts = append(impacts, Impact{
67                APIName:    instance,
68                GWPperHour: embodied + operational,
69            })
70         }
71
72         sort.Slice(impacts, func(i, j int) bool {
73             return impacts[i].APIName < impacts[j].APIName
74         })
75
76         result = append(result, ImpactList{
77             Location: location,
78             Impacts:  impacts,
79         })
80     }
81
82     sort.Slice(result, func(i, j int) bool {
83         return result[i].Location < result[j].Location
84     })
85
86     writeToGoFile(OutputLocation, result, locations)
87 }
```

Listing 7: `carbon-quantifier.go`. Query BoaviztAPI for carbon impacts and writes it to a Go file.

8 EXPERIMENTS AND RESULTS

In this chapter, a central question from the problem formulation will be answered (section 1.3): *How much does this improvement change the aforementioned carbon emissions?* To answer this question, multiple experiments will be executed to evaluate different aspects of the solution. Specifically, Original Karpenter and Carbon Efficient Karpenter will be evaluated and compared in multiple regions, provisioning nodes for various types of workloads. The result of each experiment run is a carbon impact ($kgCO_2e/h$), because the objective is to analyze the change in carbon emissions. Further details on the configuration and result of each experiment can be found in the appendices being referenced.

Three countries have been selected for the experiments that cover different regions: The United States of America (USA), India (IND), and Sweden (SWE). They are geographically representative and cover two edge cases. India (having a high-carbon electricity grid) and Sweden (having a low-carbon electricity grid) represent the two extremes, while the USA is in-between with its average carbon intensity. Ireland is also used, and it has a carbon intensity similar to that of the USA. Ireland is the most popular data center region in Europe as it was the first country to host an Amazon Web Services (AWS) data center in Europe (launched in 2007) [89]. In table 25 you can see the carbon intensity that is used in the implementation of BoaviztAPI for countries relevant to the present thesis.

COUNTRY	CARBON INTENSITY		YEAR[1]	SOURCE
India	0.626	$kgCO_2e/kW\,h$	2020	[84]
Sweden	0.04	$kgCO_2e/kW\,h$	2019	[83]
USA	0.37	$kgCO_2e/kW\,h$	2020	[84]
Ireland	0.384	$kgCO_2e/kW\,h$	2019	[83]
Germany	0.422	$kgCO_2e/kW\,h$	2019	[83]
World[2][3]	0.39	$kgCO_2e/kW\,h$	2011,19,20	[83–85]
Denmark[3]	0.158	$kgCO_2e/kW\,h$	2019	[83]

Table 25: Carbon intensity of electricity. The differences in numeric precision are intentional.

1 The year where the country had the specific carbon intensity. Not the year that the study was made or the year it was published.
2 Average of all countries, not considering population or area.
3 For reference only, the numbers are not used in the upcoming experiments.

8.1 EXPERIMENT DESIGN

In this section, I will provide an overview of how the experiments are performed to evaluate the changes in carbon emissions induced by Carbon Efficient Karpenter compared to Original Karpenter.

The experiments are run with a real Karpenter deployment that provisions real compute instances in a real Kubernetes cluster. This makes the experiments structurally realistic and true to the actual behavior of Original Karpenter and Carbon Efficient Karpenter. The experiments evaluate both Original Karpenter and Carbon Efficient Karpenter, enabling a comparison between the two.

An *experiment suite* in the Karpenter project has been developed. It is based on the *integration test suite* in Karpenter. Although it uses the same tools as the integration tests and programmatically is the same as the integration tests, it is not an integration test. There is an important conceptual distinction. While tests verify behavior, experiments evaluate behavior and do not compare it to any expected behavior. Therefore, in contrast to tests, an experiment does not have a binary pass/fail output.

All the experiments are programmed and then executed in an automated and asynchronous way. This means that peers can easily check the results by executing the same experiment program (assuming access to a similar cluster). The steps needed to create a similar cluster environment are detailed in appendix B. In fig. 18 you can see a screenshot of an experiment being executed.

The experiments have the general structure that, first, some pods are deployed to a cluster where all nodes are cordoned and Karpenter is running. When Karpenter notices that the new pods are in a Pending state, it provisions one or multiple nodes. The size and type of the new nodes depend on the number of pods that have been deployed and the resource requests of each pod. The size and type of the new nodes will also be different depending on whether `carbonEfficient` is turned on or off. When all new pods have been scheduled on the new nodes and the pods are in a Running state, then I will start gathering data about the cluster state. The experiment suite will automatically obtain information about the instance types of all new nodes, their utilization, and their geographic location. This information is used to estimate their carbon impact using BoaviztAPI. Afterwards, the carbon impacts are analyzed. If Carbon Efficient Karpenter generally provisions a set of instance types resulting in a lower carbon impact than those from Original Karpenter, then I have made an improvement to Karpenter resulting in a carbon impact reduction.

In both experiments, the instance types are constrained to the requirements in table 26. These specific requirements are created to make the provisioning realistic. In practice, the chosen requirements in table 26 can be considered relatively nonrestrictive.

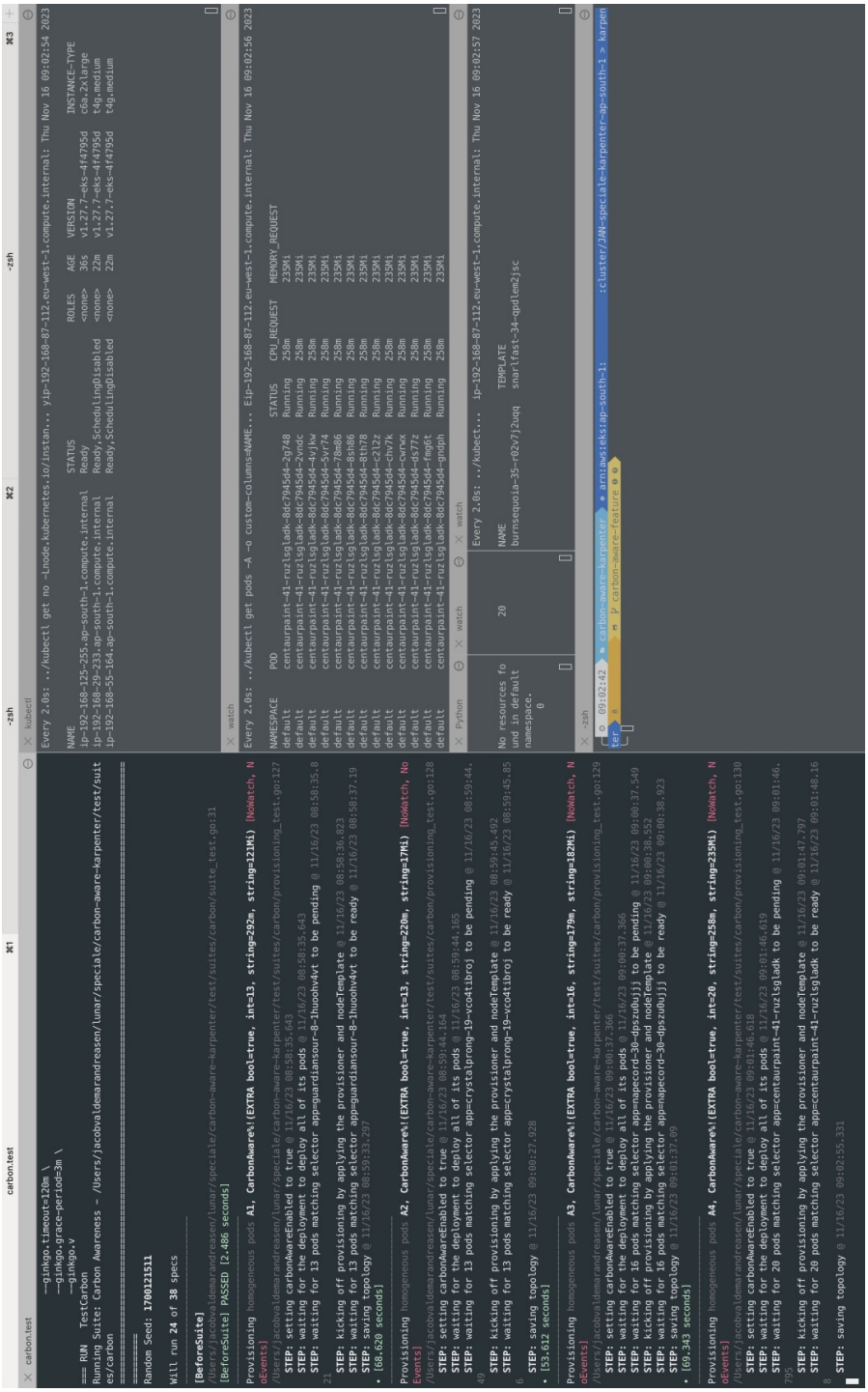

Figure 18: Screenshot of a preliminary experiment being executed. Since the image is from a preliminary test run, the data in the image shouldn't be relied on as the result of a final experiment.

REQUIREMENT	ALLOWED VALUES
Instance Family	`m`, `t`, `c`, `r`, `a`
Purchase Options	On-demand
Operating System	Linux
Architecture	AMD64 (both Intel and AMD), ARM64

Table 26: Instance type requirements imposed by the Provisioner in both experiments.

8.2 EXPERIMENT: HOMOGENEOUS WORKLOADS

The purpose of this experiment is to examine how the carbon emissions in Kubernetes clusters running Carbon Efficient Karpenter compare to those running Original Karpenter when handling homogeneous workloads. In this experiment, we will specifically look at the carbon impact ($kgCO_2e/h$) of the cloud instances that are provisioned when a deployment is applied to a cluster requesting a specific amount of replicas and CPU and memory resources. All replicas have the same CPU and memory request, making it a homogeneous workload.

8.2.1 *Experiment design*

An overview of the experiment is shown in fig. 19. If we just execute the experiment for one set of CPU, memory, and replicas (called a configuration), a fair criticism is that the result is mere luck and not representative. Therefore, the experiment will be executed using 35 different configurations (see table 27). The configuration parameters are:

- `(bool) carbonEfficient` defines if this experiment is for Original Karpenter (disabled/false) or Carbon Efficient Karpenter (enabled/true).
- `(int) replicaCount` defines the number of pods that should be created.
- `(string) cpuRequest` defines how much compute (unit: CPU) each pod should request.
- `(string) memoryRequest` defines how much memory (unit: GiB) each pod should request.

The specific configurations for this experiment are listed in table 27. Each configuration is executed using `carbonEfficient=false` and then `carbonEfficient=true`, which is why the `carbonEfficient` parameter is not shown in the table. The execution order does not matter.

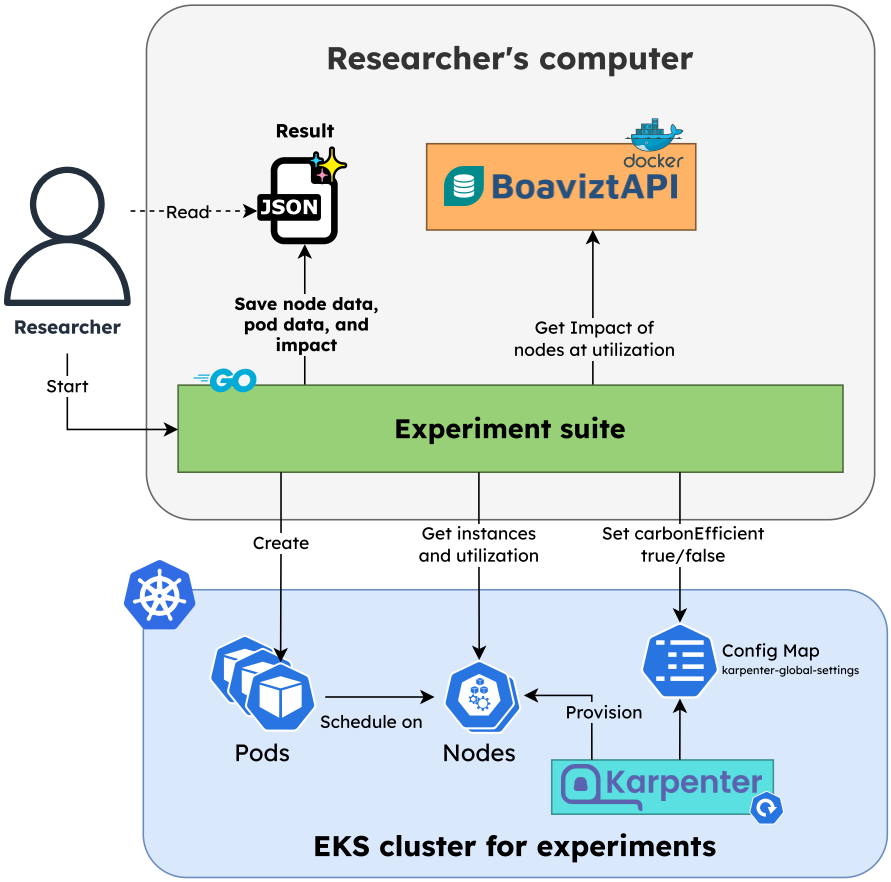

Figure 19: Overview of *Experiment: Homogeneous Workloads*.

The specific amount of CPU, memory, and replicas in each configuration is not important in itself. However, it is important that the configurations are diverse so the result does not become biased. The configuration parameters have been randomly generated from a uniform distribution. CPU is in the range of 10 mCPU to 3000 mCPU[4], memory is in the range of 10 MiB to 3000 MiB, and replicas is in the range of 1 to 20. The ranges are based on practical experience as a Kubernetes Administrator, estimating that most applications will fall within those ranges. Using random values improves the generalizability of the result. Each configuration in table 27 has an identifier (ID) which helps identify it in section 8.2.2. An example of how to read table 27 is "configuration A35 deploys 19 replicas (pods), and each pod requests 2887 mCPU and 1229 MiB". Furthermore, the experiment is executed in multiple regions (Ireland, USA, Sweden, and India) as described above. The purpose of this is to ensure that the results are not biased by a single region's low or high carbon intensity in its electrical grid. Figures 34 and 35 in appendix C shows the distribution of the configuration parameters.

As mentioned in the previous section, the experiments are defined programmatically and executed in a predictable manner. The steps that are performed in this experiment are shown below in figs. 20 and 21.

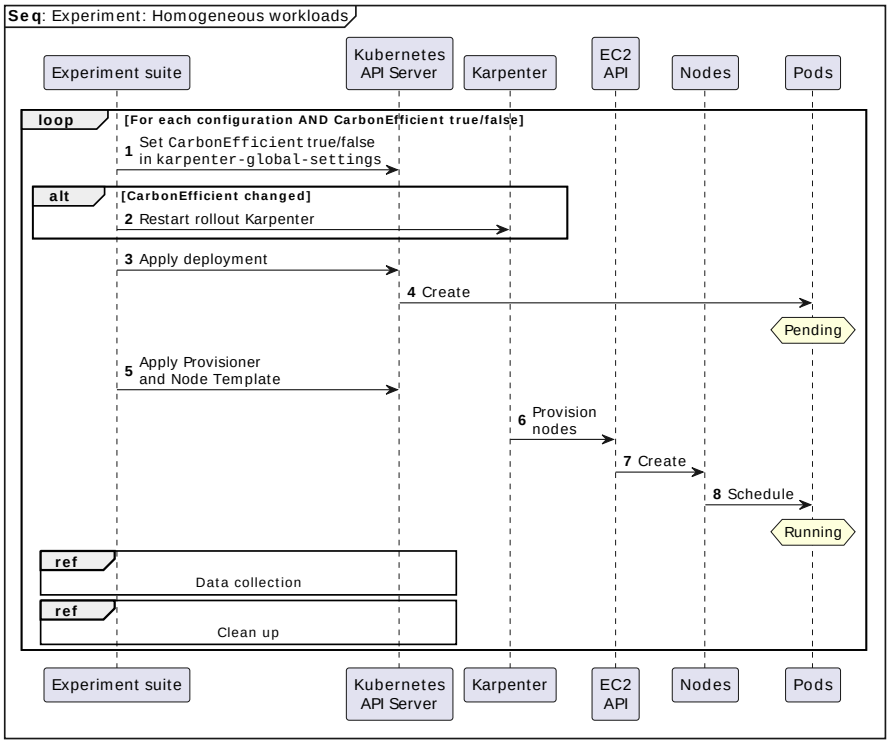

Figure 20: Sequence diagram that shows how this experiment unfolds.

4 CPU is a de-facto standard unit for cloud compute resources and a mCPU is one-thousandth of a CPU.

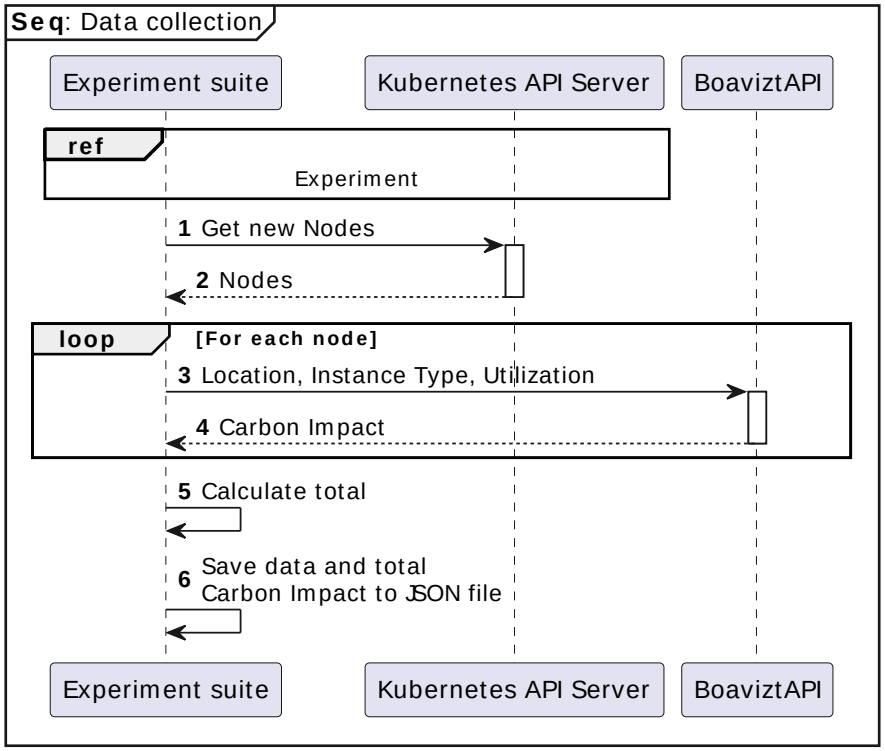

Figure 21: Diagram that shows how data is collected and saved at the end of an experiment.

Listing 8 shows an example of the file saved in step 6 in fig. 21. This file contains the resulting carbon impact (gCO_2e/h) on line 2. It also contains some details about the executed experiment, documenting the cluster state to improve the traceability of the result. However, there are hundreds of saved files containing hundreds of lines, so it is not feasible to include the raw outputs in the thesis – not even in an appendix.

When BoaviztAPI estimates the total environmental impact of the configuration, it uses each node's instance type, utilization percentage, and location. The environmental impact of all nodes is added to arrive at a total for the whole experiment. For all experiments in this section (section 8.2), only one node is provisioned (by design). However, for the experiments in section 8.3 multiple nodes can be provisioned.

ID	CPU	MEMORY	REPLICAS
A1	485 mCPU	1625 MiB	3
A2	718 mCPU	100 MiB	10
A3	2110 mCPU	2092 MiB	10
A4	1133 mCPU	1564 MiB	12
A5	2922 mCPU	186 MiB	17
A6	2918 mCPU	2672 MiB	15
A7	1935 mCPU	997 MiB	18
A8	2582 mCPU	697 MiB	2
A9	1212 mCPU	350 MiB	5
A10	1900 mCPU	939 MiB	10
A11	2956 mCPU	693 MiB	20
A12	1683 mCPU	1960 MiB	16
A13	2802 mCPU	207 MiB	10
A14	2164 mCPU	833 MiB	7
A15	1457 mCPU	852 MiB	2
A16	1921 mCPU	2642 MiB	15
A17	2664 mCPU	1338 MiB	11
A18	604 mCPU	2270 MiB	4
A19	1192 mCPU	1814 MiB	9
A20	2977 mCPU	2352 MiB	4
A21	1213 mCPU	350 MiB	16
A22	1980 mCPU	2936 MiB	8
A23	2705 mCPU	2548 MiB	19
A24	2987 mCPU	161 MiB	1
A25	1963 mCPU	1404 MiB	17
A26	334 mCPU	984 MiB	13
A27	118 mCPU	1894 MiB	11
A28	1858 mCPU	698 MiB	14
A29	1706 mCPU	1744 MiB	15
A30	2887 mCPU	1814 MiB	2
A31	2241 mCPU	1804 MiB	18
A32	1991 mCPU	1351 MiB	1
A33	1575 mCPU	115 MiB	6
A34	787 mCPU	1546 MiB	4
A35	2887 mCPU	1229 MiB	19

Table 27: Configurations for *Experiment: Homogeneous Workloads*.

```
{
  "Impact": 49.689181229999996,
  "Summary": [
    "c6a.8xlarge: 0.849388"
  ],
  "Verbose": [
    {
      "NodeName": "ip-192-168-80-187.ec2.internal",
      "InstanceType": "c6a.8xlarge",
      "Pods": [
        {
          "name": "scourgewhite-129-bkpo1tj1mb-6cd7bf7cd-84k68",
          "cpu_request": "1683m",
          "memory_request": "1960Mi"
        },
        {
          "name": "scourgewhite-129-bkpo1tj1mb-6cd7bf7cd-m8b2p",
          "cpu_request": "1683m",
          "memory_request": "1960Mi"
        },
      ],
      "Utilization": 0.8493877551020408
    }
  ]
}
```


Listing 8: Extract of the saved result for experiment A12 in USA with Carbon Efficient enabled. The high level of detail improve the trustworthyness of the result.

8.2.2 *Results*

This experiment's results are presented in figs. 22 to 25. Details on the results can be found in appendix C.

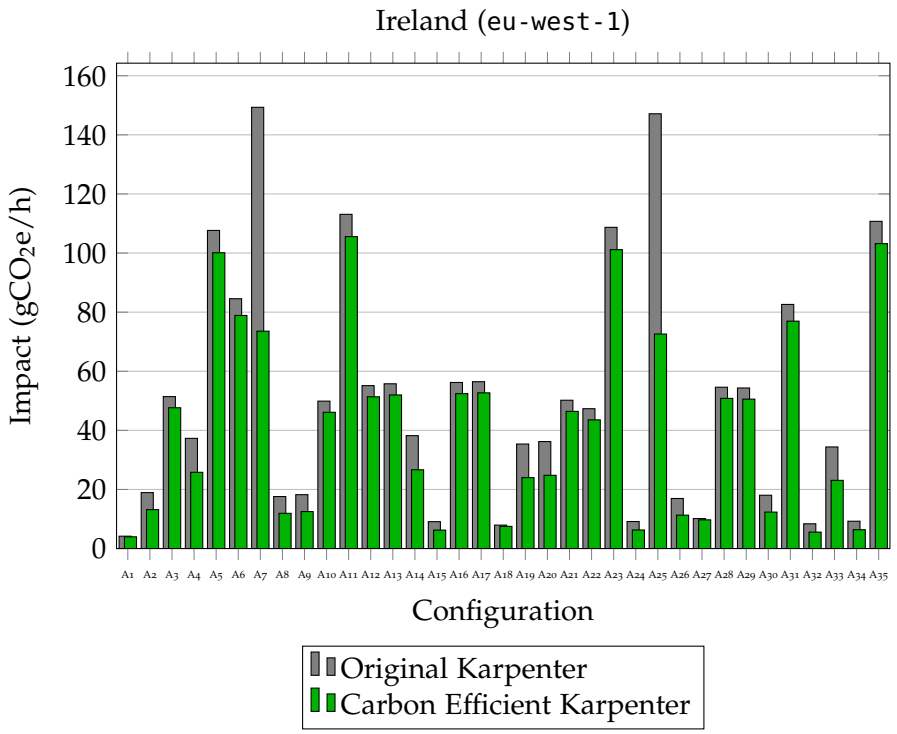

Figure 22: Carbon impact of cloud instances provisioned by Original Karpenter and Carbon Efficient Karpenter to run 35 different homogeneous workloads in a Kubernetes Cluster in Ireland.

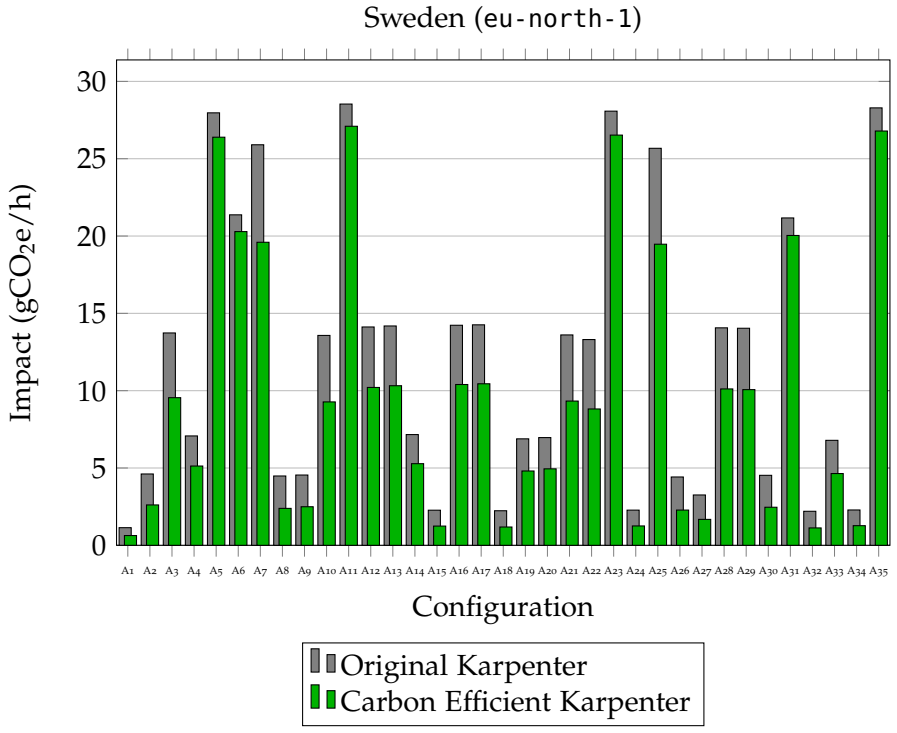

Figure 23: Carbon impact of cloud instances provisioned by Original Karpenter and Carbon Efficient Karpenter to run 35 different homogeneous workloads in a Kubernetes Cluster in Sweden.

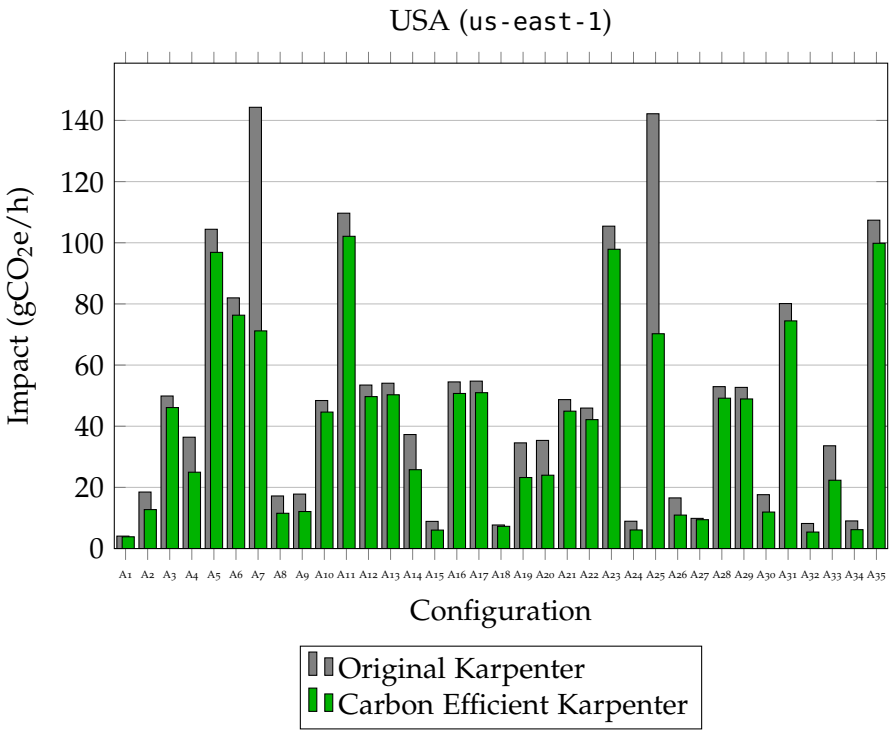

Figure 24: Carbon impact of cloud instances provisioned by Original Karpenter and Carbon Efficient Karpenter to run 35 different homogeneous workloads in a Kubernetes Cluster in USA.

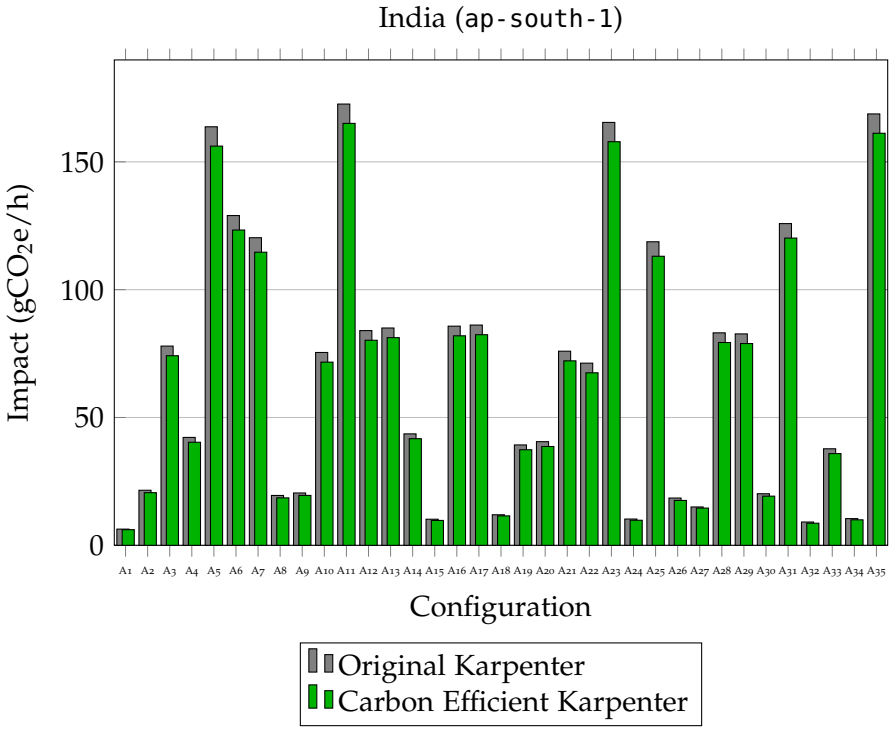

Figure 25: Carbon impact of cloud instances provisioned by Original Karpenter and Carbon Efficient Karpenter to run 35 different homogeneous workloads in a Kubernetes Cluster in India.

8.2.3 *Selecting a Method for Hypothesis Testing*

In this subsection, a method for performing two-sample hypothesis testing on the experiment results is selected.

The *Wilcoxon Signed-Rank Test* is a nonparametric statistical hypothesis test used to compare the locations of two populations using two matched samples [97]. It is an alternative to *paired Student's t-test* when the data can not be assumed to be normally distributed – which is the case here, as shown in appendix D. If we denote the difference between two paired random variables $D_i = Y_i - X_i, i = 1, \ldots, N$, then the Wilcoxon-signed-rank test "assumes that the differences D_i are mutually independent and $D_i, i = 1, \ldots, N$ comes from a continuous distribution F that is symmetric about a median θ" [97]. To verify this symmetry, the skewness is examined for each region by calculating the third standardized moment $\tilde{\mu}_3 = E((X - \mu)/\sigma)^3$, where X is the differences between the matched pairs (Original Karpenter minus Carbon Efficient Karpenter), μ is the mean of the difference, σ is the standard deviation, and r is the region:

$$\tilde{\mu}_3[r] = \begin{cases} 3.5991 & r = \text{Ireland} \\ 0.8614 & r = \text{Sweden} \\ 3.5844 & r = \text{USA} \\ 0.5431 & r = \text{India} \end{cases} \tag{30}$$

Symmetric distributions have a third standardized moment that is zero, and it is rare that the third standardized moment exceeds 2 or 3 in absolute value [98]. For that reason, it is rejected that the differences are symmetric about the median[5]. In other words, one of the assumptions in the Wilcoxon Signed-Rank Test is violated, and therefore it cannot be used for this experiment.

In this case, the *Sign Test* is an appropriate method. It is a nonparametric statistical hypothesis test that can test for consistent differences between pairs of observations. Although it performs worse than the Wilcoxon Signed-Rank Test, it does not assume symmetry around the median [99]. In the next subsection, the results will be analyzed using Sign Test.

8.2.4 *Analysis*

In this section, the results from the homogeneous experiments will be analyzed.

First, the Sign Test is used to compare the median difference between Original Karpenter and Carbon Efficient Karpenter. A positive

5 This skew can also be verified visually in appendix D.

median difference means that Carbon Efficient Karpenter has reduced the carbon impact.

NULL HYPOTHESIS: The median difference between the paired samples is zero ($H_0 : M = 0$). In other words, it is equally likely that Carbon Efficient Karpenter provisions nodes with a lower carbon impact than Original Karpenter and vice versa.

ALTERNATIVE HYPOTHESIS: The median difference is greater than zero (one-tailed, right) ($H_1 : M > 0$). In other words, Carbon Efficient Karpenter provisions nodes with a lower carbon impact than Original Karpenter.

Let $I = \text{A1}, \ldots, \text{A35}$ be the configurations and $R = \text{Ireland, Sweden, USA, India}$ be the regions. Let $C_{r,i}$ and $O_{r,i}$ denote the carbon impact data from Carbon Efficient Karpenter and Original Karpenter, respectively, for region $r \in R$ and configuration $i \in I$. The differences D_r are given by:

$$\forall r \in R \quad D_r = \{O_{r,i} - C_{r,i} \mid i \in I\}. \tag{31}$$

Performing the Sign Test on the differences for all regions reveals the following p-values:

$$p[D_r] = \begin{cases} 2.9104 \cdot 10^{-11} \approx 0.00 \implies H_0 \text{ rejected} & r = \text{Ireland} \\ 2.9104 \cdot 10^{-11} \approx 0.00 \implies H_0 \text{ rejected} & r = \text{Sweden} \\ 2.9104 \cdot 10^{-11} \approx 0.00 \implies H_0 \text{ rejected} & r = \text{USA} \\ 2.9104 \cdot 10^{-11} \approx 0.00 \implies H_0 \text{ rejected} & r = \text{India} \end{cases} \tag{32}$$

The Sign Test (eq. (32)) reveals a p-value of 0.00 for all regions. This extremely low p-value, far below the conventional threshold ($\alpha = 0.05$), strongly *rejects* the null hypothesis (H_0) of a zero median difference between Original Karpenter and Carbon Efficient Karpenter. This indicates that Carbon Efficient Karpenter provisions nodes with a lower carbon impact than Original Karpenter, and that it is extremely unlikely that this occurred by chance.

Let us now inspect the size of the reductions in gCO_2e/h. The median difference (or reduction) is

$$M[D_r] = \begin{cases} 5.6370\,gCO_2e/h & r = \text{Ireland} \\ 2.0505\,gCO_2e/h & r = \text{Sweden} \\ 5.6186\,gCO_2e/h & r = \text{USA} \\ 3.7775\,gCO_2e/h & r = \text{India} \end{cases} \tag{33}$$

The mean difference is

$$\mu[D_r] = \begin{cases} 9.3639\,\mathrm{gCO_2e/h} & r = \text{Ireland} \\ 2.5450\,\mathrm{gCO_2e/h} & r = \text{Sweden} \\ 9.2038\,\mathrm{gCO_2e/h} & r = \text{USA} \\ 3.0801\,\mathrm{gCO_2e/h} & r = \text{India} \end{cases} \tag{34}$$

The standard deviation is

$$\sigma[D_r] = \begin{cases} 14.5167 & r = \text{Ireland} \\ 14.2759 & r = \text{Sweden} \\ 14.6358 & r = \text{USA} \\ 00.4615 & r = \text{India} \end{cases} \tag{35}$$

The mean reduction is larger than the median reduction in Ireland, Sweden, and the USA, indicating that the reductions are right-tailed and have a positive skew [98]. The median and mean reductions are the highest in Ireland and the USA at approximately $M_D \approx 6\,\mathrm{gCO_2e/h}$ and $\mu_D \approx 9\,\mathrm{gCO_2e/h}$. Although the reduction in Sweden is low compared to the USA and Ireland, it seems substantial when considering Sweden's low carbon intensity.

The relative reductions are given by

$$\forall r \in R \quad P_r = \left\{ \frac{O_{r,i} - C_{r,i}}{O_{r,i}} \cdot 100 \mid i \in I \right\}. \tag{36}$$

Using this definition, first quartile Q_1, second quartile Q_2 (the median M), and the third quartile Q_3 of the relative reductions are shown in table 28.

r	$Q_1[P_r]$	$Q_2[P_r]$	$Q_3[P_r]$
Ireland	6.8299 %	7.5752 %	31.5062 %
Sweden	26.3956 %	30.1954 %	44.8766 %
USA	7.0433 %	7.8063 %	32.1395 %
India	4.4161 %	4.5679 %	4.8005 %

Table 28: The three quartiles of the relative reduction by region.

The mean of the relative reductions are

$$\mu[P_r] = \begin{cases} 19.2124\,\% & r = \text{Ireland} \\ 30.5770\,\% & r = \text{Sweden} \\ 19.5748\,\% & r = \text{USA} \\ 4.5481\,\% & r = \text{India} \end{cases} \tag{37}$$

This shows that although Sweden has the lowest reduction potential in gCO_2e/h, it has the highest median and mean relative reduction. Figure 26 illustrates the relative reductions in the four regions.

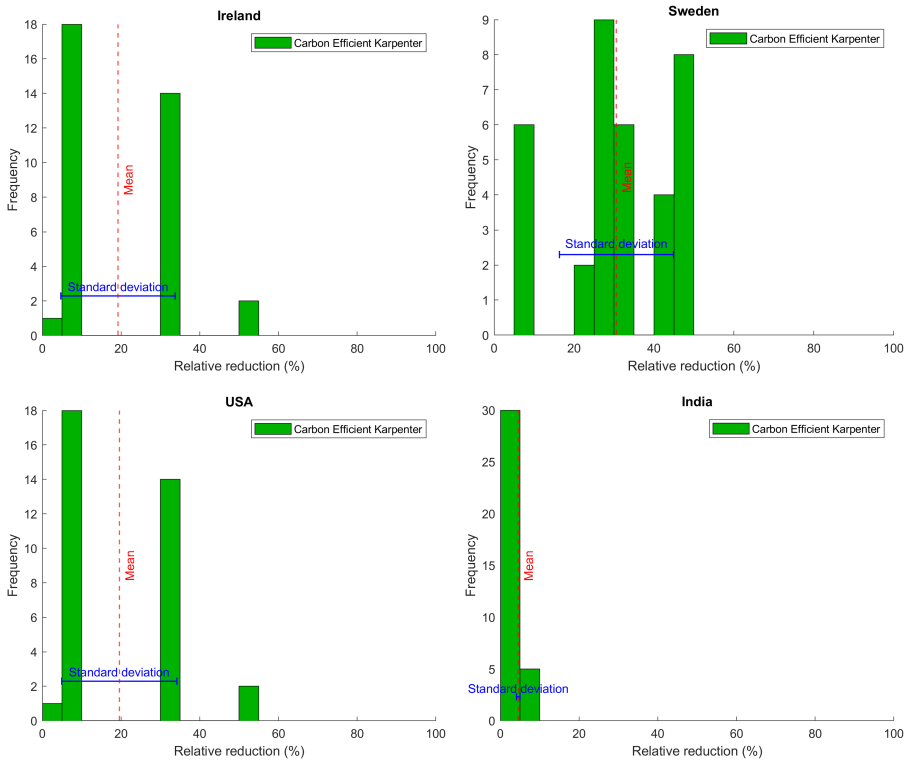

Figure 26: Histogram of relative reduction in each region in *Experiment: Homogeneous Workloads*.

To summarize, there is a statistically significant median difference in carbon impact between Carbon Efficient Karpenter and Original Karpenter when provisioning homogeneous workloads. For homogeneous workloads, the median reduction in carbon impact when using Carbon Efficient Karpenter compared to Original Karpenter ranges from approximately $2\,gCO_2e/h$ *to* $6\,gCO_2e/h$ *depending on the region. The lower and upper quartile relative reduction in carbon impact when using Carbon Efficient Karpenter compared to Original Karpenter for the four regions are* (Q_1, Q_3)*: Ireland* $(7\,\%, 32\,\%)$*, Sweden* $(26\,\%, 45\,\%)$*, USA* $(7\,\%, 32\,\%)$*, and India* $(4\,\%, 5\,\%)$*.*

Response to problem formulation.

This will be discussed in chapter 9.

8.3 EXPERIMENT: REAL CLUSTERS

In this experiment, data about real clusters will be used. This will help us understand how Carbon Efficient Karpenter can improve the environmental sustainability in real scenarios. This is made possible by two generous organizations, that have donated Kubernetes cluster metadata for this experiment. It has been a condition for the use and publication of their data that it is anonymized to hide their iden-

tity. There can be good reasons why they wish to be anonymous. For example, to avoid internal political conflict, negative press, or providing hackers with technical intelligence. Hence, the donating organizations' clusters will be denoted by pseudonyms, specifically geometric shapes devoid of any connection to the organizations' true identity.

The anonymity could hinder *replicability*, but not *reproducibility*[6]. The experiment can be reproduced as I will happily provide the anonymized cluster data to anyone who requests it for the purpose of reproducing the experiment[7]. In practice, the anonymity shouldn't have an impact on replicability either. That is because the datasets are very small ($N = 4$), and therefore the results can not be generalized. Hence, it is not certain (maybe even improbable) that someone *replicating* this experiment with data from a different cluster will achieve the exact same results.

There is a total of four datasets, meaning four different clusters. Each dataset describes the CPU and memory requests of each pod running in the cluster, along with each node's instance type and CPU utilization[8]. The clusters contain many pods, therefore it is not possible to list the resource requests of each pod here. However, fig. 27 shows how the resource requests are distributed for each dataset. The four datasets are used to create four experiments where a set of pods are deployed with the same resource requests as those in the real cluster. I don't use the same container images as those running in the real cluster. Instead, a *Pause* container image from the Amazon Elastic Kubernetes Service (EKS) distribution is deployed. By design, this Pause application doesn't really do anything. Using a container image that is different from the container image used by the original workloads, doesn't impact the validity of the experiment. That is because Karpenter manages nodes based on resource requests by pods and not based on actual resource usage.

The four datasets also describe the nodes in the real clusters, specifically each node's instance type and CPU utilization. I use this data to compare the real clusters' estimated carbon impact to that of the nodes provisioned by Original Karpenter and Carbon Efficient Karpenter.

In this experiment, the resource requests specified by the pods are assumed to represent the resources used by the applications. From my experience, this assumption may not be valid, however, in the absence of a more accurate metric for CPU and memory usage, resource requests are the optimal choice.

6 I use the definitions coined by Claerbout/Donoho/Peng, summarized: reproducibility is *same data + same methods = same results* and replicability is *new data and/or new methods in an independent study = same findings* [100].

7 The raw dataset is too large to include in the thesis.

8 CPU utilization is defined as the sum of each pod's requested CPU (that is running on the node) divided by allocatable CPU.

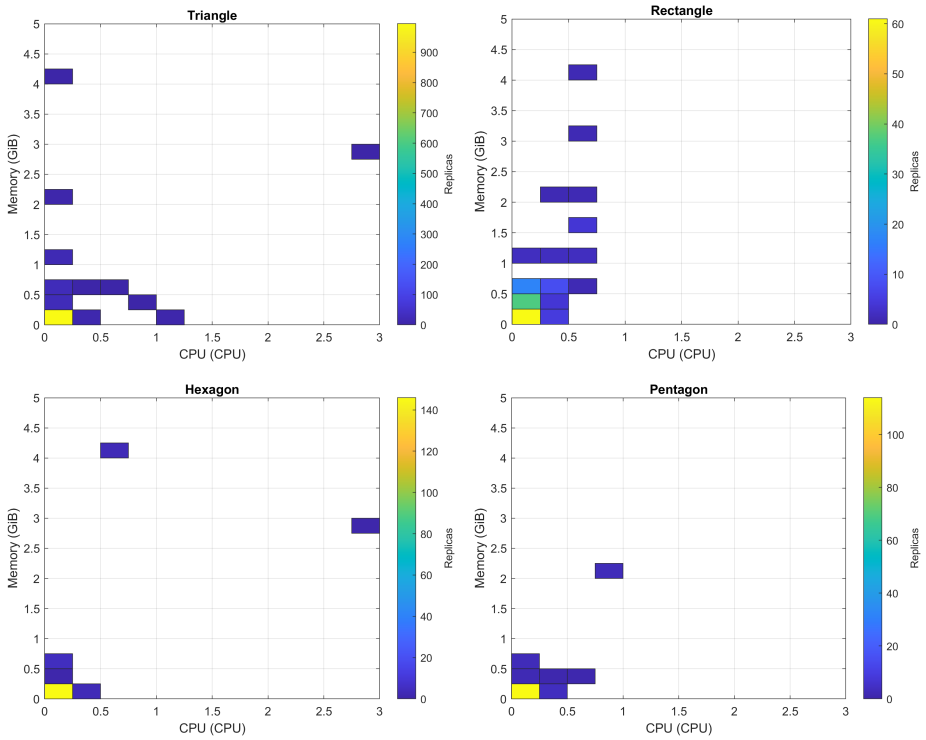

Figure 27: Overview of pod resource requests for the four different clusters. See figs. 36 to 39 (pages 114 to 115) in appendix C for larger and more detailed heatmaps.

In these experiments, the pods from each dataset are deployed to four regions: (1) the region where the real cluster resides and (2-4) the USA, Sweden, and India. By running experiments in the regions where the real clusters reside, the carbon impact of the real clusters can be compared with that of Original Karpenter and Carbon Efficient Karpenter. Furthermore, by running the experiments in the USA, Sweden, and India, the carbon impact of Original Karpenter can be compared with that of Carbon Efficient Karpenter in different regions when provisioning workloads that mimic those from real clusters.

It is also possible to estimate the carbon impact that the real clusters would have had, had they been running the same instance types in the USA, Sweden, and India. However, it has been decided not to show that, as it could be misleading. That is because the contributing organizations maybe would have chosen different instance types if their cluster was located in a different region.

Figure 28 provides an overview of the experiment.

8.3.1 *Experiment Design*

The design of this experiment is similar to the previous experiment but with some important differences. In contrast to the previous experiment, which only used the provisioning feature in Karpenter, this

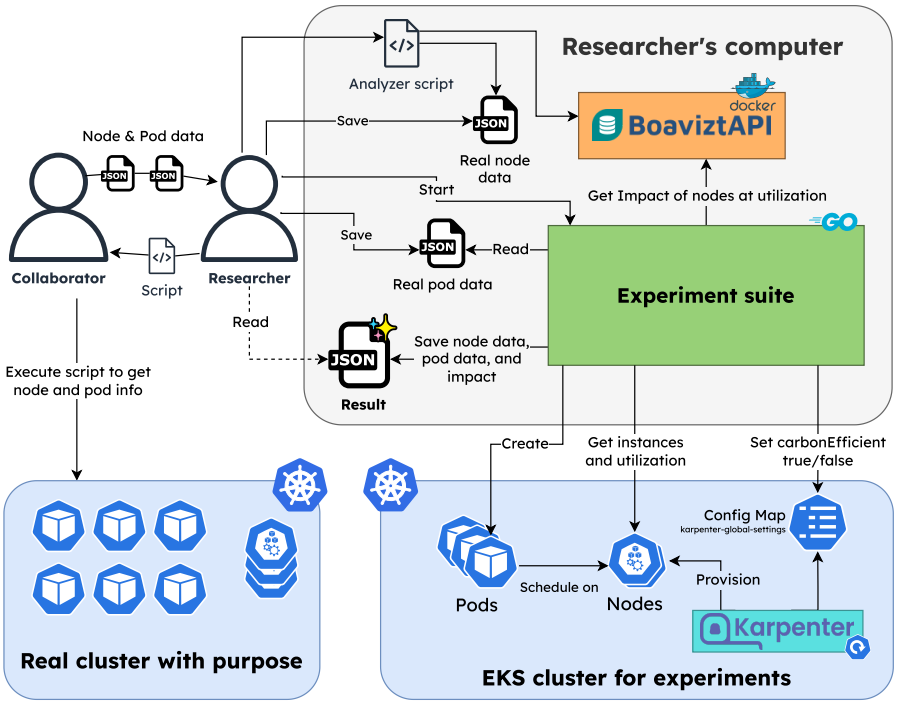

Figure 28: Overview of *Experiment: Real Clusters*.

experiment use both provisioning and consolidation. If all pods from a cluster dataset were to be deployed at the same time, Karpenter would create *one* very large instance. That would not be a fair imitation of the behavior found in real clusters. Therefore, pods are deployed in chunks: deploy some pods, wait for consolidation, deploy more pods, wait for consolidation; repeating until all pods are deployed. This is a more accurate imitation of the behavior found in real clusters where new applications are deployed as they are developed. The configuration parameters for this experiment are found below:

- `(bool) carbonEfficient` specifies if Original Karpenter (disabled / false) or Carbon Efficient Karpenter (enabled / true) is used.
- `(string) inputFile` specifies the pod dataset file containing the resource requests from the real cluster.

The steps that are performed in this experiment are shown in fig. 29. The environmental impact is calculated in the same way as described in the previous experiment and in fig. 21. The output is a file with the same format as shown in listing 8.

8.3.2 *Results*

This experiment's results are presented in figs. 30 to 33. Further details can be found in appendix C.

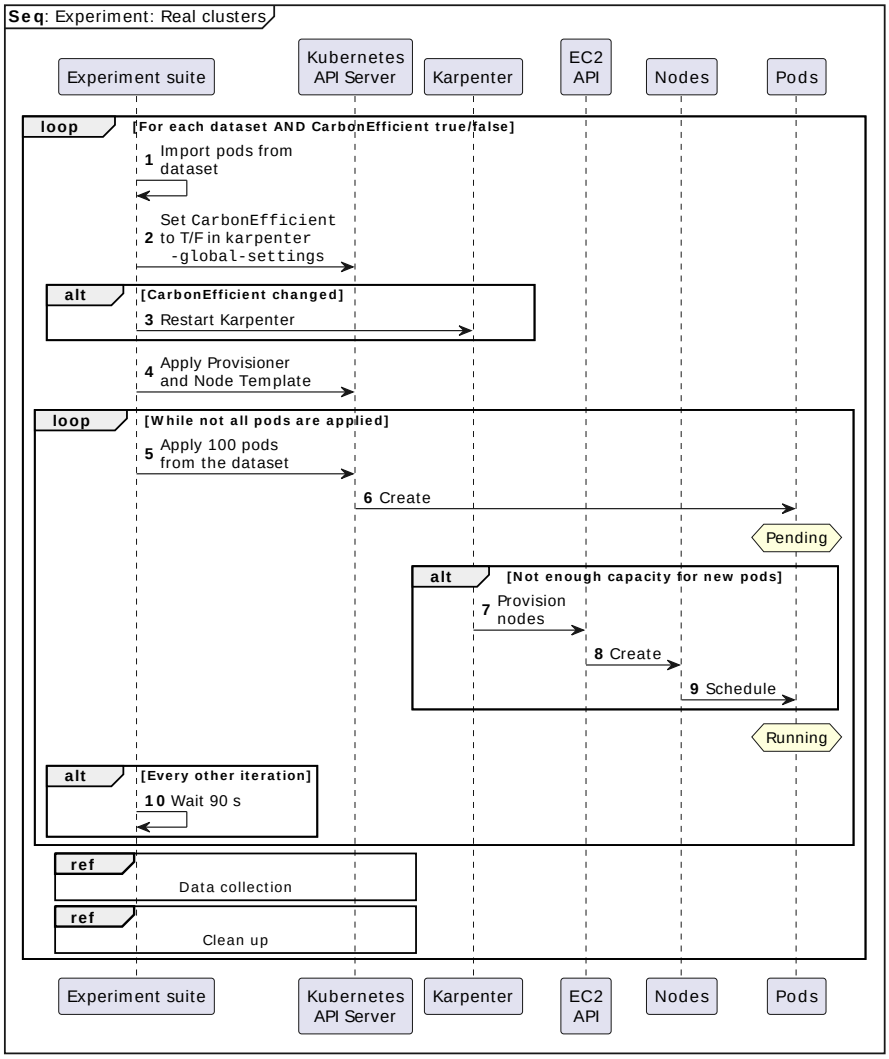

Figure 29: Sequence diagram that shows how this experiment unfolds.

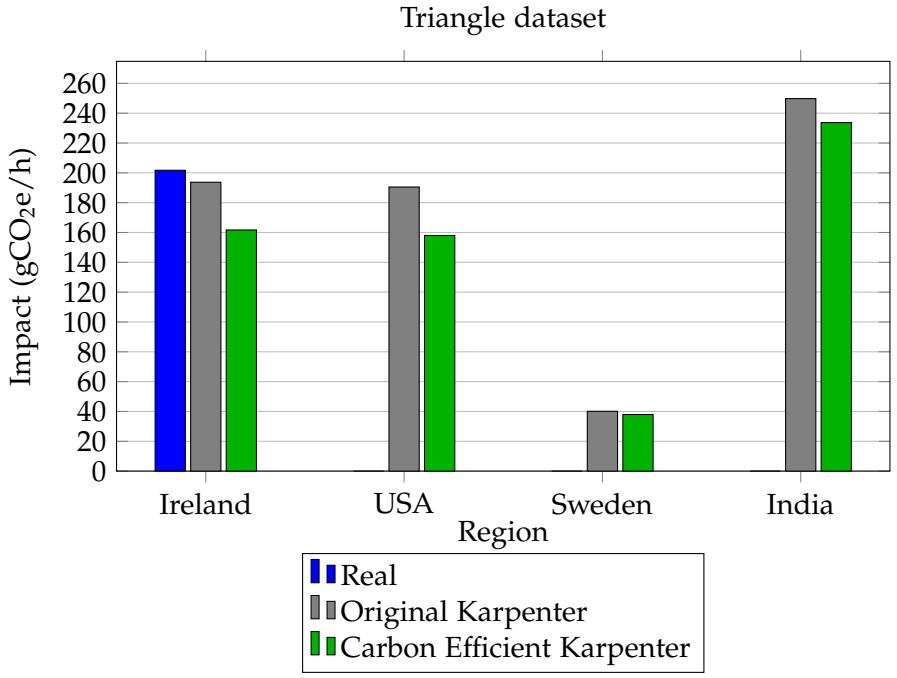

Figure 30: Impact of cloud instances provisioned to run support pods from the Triangle dataset in three regions. Additionally, the carbon impact of the cloud instances in the real Triangle cluster.

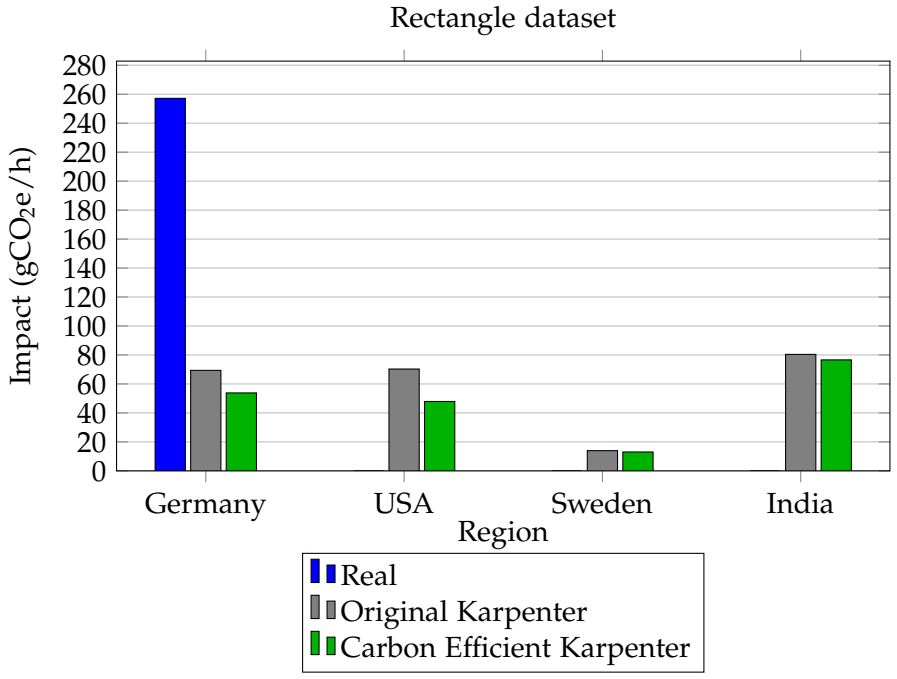

Figure 31: Impact of cloud instances provisioned to run support pods from the Rectangle dataset in three regions. Additionally, the carbon impact of the cloud instances in the real Rectangle cluster.

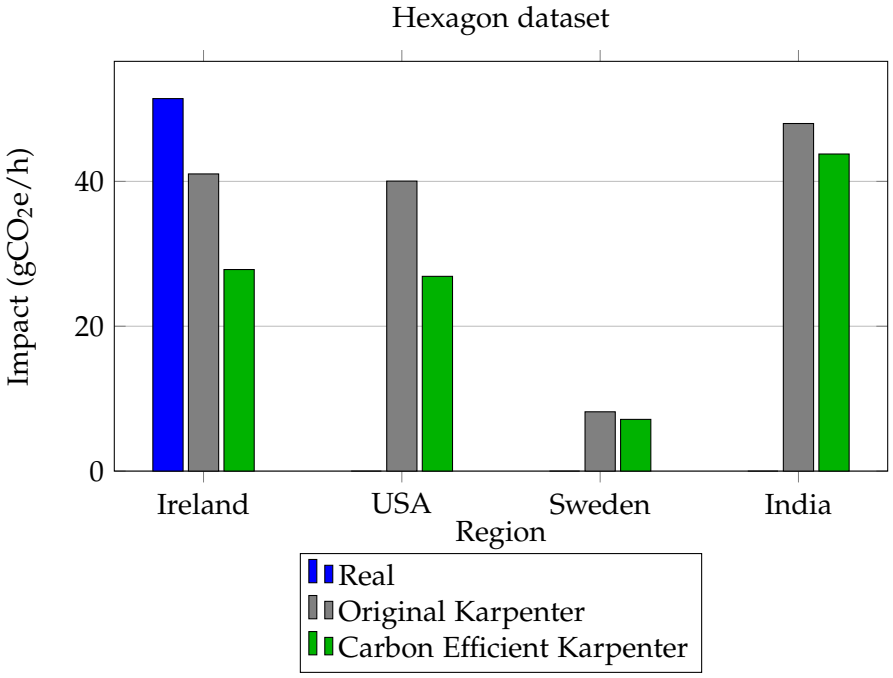

Figure 32: Impact of cloud instances provisioned to run support pods from the Hexagon dataset in three regions. Additionally, the carbon impact of the cloud instances in the real Hexagon cluster.

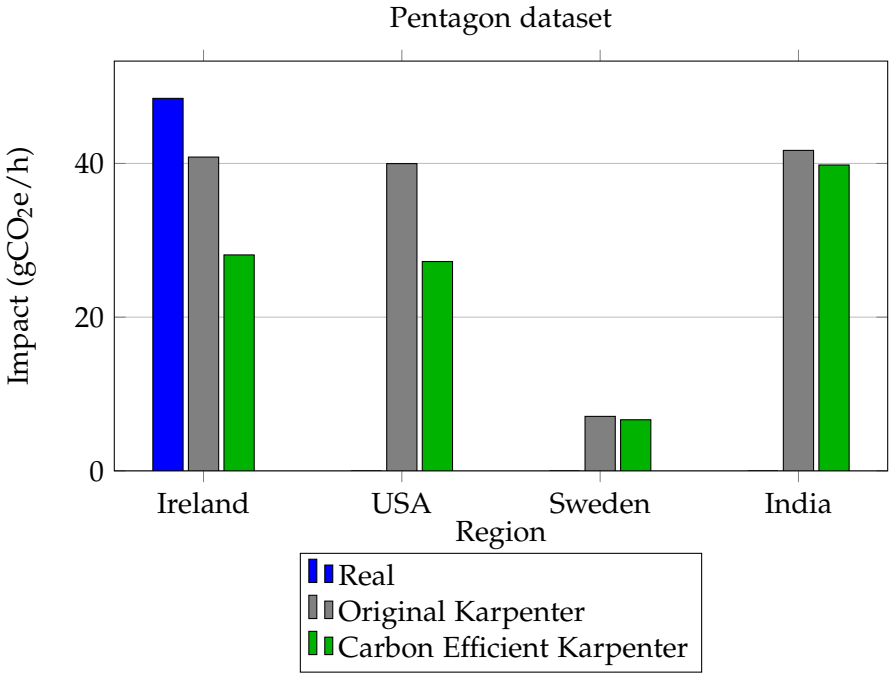

Figure 33: Impact of cloud instances provisioned to run support pods from the Pentagon dataset in three regions. Additionally, the carbon impact of the cloud instances in the real Pentagon cluster.

8.3.3 *Analysis*

Since the number of datasets for each region is quite small ($N = 4$) it can not be rejected that the result occurred by chance. Since statistical hypothesis testing can not help us analyze the results in this case, a couple of facts inferred from the results will be provided (figs. 30 to 33).

Let $J = \text{Triangle, Rectangle, Hexagon, Pentagon}$ be the datasets and $R = \text{Ireland, Sweden, USA, India}$ be the regions. Let $C_{r,j}$ and $O_{r,j}$ denote the carbon impact data from Carbon Efficient Karpenter and Original Karpenter, respectively, for a region $r \in R$ and a dataset $j \in J$. The relative reduction is given as:

$$P_{r,j} = \frac{O_{r,j} - C_{r,j}}{O_{r,j}} \cdot 100 \tag{38}$$

$$\forall r \in R \quad P_r = \left\{ P_{r,j} \mid j \in J \right\} \tag{39}$$

$$\forall j \in J \quad P_j = \left\{ P_{r,j} \mid r \in R \right\} \tag{40}$$

For real-cluster workloads, the median relative reduction in carbon impact (by region) when using Carbon Efficient Karpenter compared to Original Karpenter is

$$M[P_r] = \begin{cases} 31.2835\,\% & r = \text{Ireland} \\ 6.3981\,\% & r = \text{Sweden} \\ 31.8634\,\% & r = \text{USA} \\ 5.5839\,\% & r = \text{India} \end{cases} \tag{41}$$

For real-cluster workloads, the mean relative reduction in carbon impact (by cluster) when using Carbon Efficient Karpenter compared to Original Karpenter is

$$\mu[P_j] = \begin{cases} 16.5204\,\% & j = \text{Triangle} \\ 22.4560\,\% & j = \text{Rectangle} \\ 22.4740\,\% & j = \text{Hexagon} \\ 18.6920\,\% & j = \text{Pentagon} \end{cases} \tag{42}$$

For the real clusters used in this experiment, the relative reduction in carbon impact when migrating to Carbon Efficient Karpenter from *not using Karpenter* is given as

$$L_{r,j} = \frac{Y_{r,j} - C_{r,j}}{Y_{r,j}} \cdot 100, \tag{43}$$

where $Y_{r,j}$ denote the carbon impact data from the real-cluster dataset j in region r. The relative reductions in carbon impact when migrating to Carbon Efficient Karpenter in the region where the real clusters actually reside are

$$\begin{aligned} L_{\text{Ireland,Triangle}} &= 19.85\,\% \\ L_{\text{Germany,Rectangle}} &= 73.02\,\% \\ L_{\text{Ireland,Hexagon}} &= 45.89\,\% \\ L_{\text{Ireland,Pentagon}} &= 42.03\,\% \end{aligned} \tag{44}$$

This cannot be generalized. It only shows the reduction potential for the exact clusters used in this experiment.

Response to problem formulation.

To summarize, it cannot be rejected that the following result occurred by chance. For real clusters, the median relative reduction in carbon impact when using Carbon Efficient Karpenter compared to Original Karpenter (by region, across clusters) ranges from approximately 6 % *to* 32 % *depending on the region. For real clusters, the mean relative reduction in carbon impact when using Carbon Efficient Karpenter compared to Original Karpenter (by cluster, across regions) ranges from approximately* 17 % *to* 22 % *depending on the cluster.*

This will be discussed in chapter 9.

9 DISCUSSION

In this chapter, we take a step back, reflect on the results from the previous chapter, and discuss the impact of this thesis.

9.1 ON PROVISIONING FOR HOMOGENEOUS WORKLOADS

In this section, I will attempt to explain why a decrease in carbon emissions is observed when using Carbon Efficient Karpenter instead of Original Karpenter to provision homogeneous workloads.

In principle, the reductions observed in the experiment with homogeneous workloads can be explained by a discrepancy between the price of cloud instances and their carbon impact. However, in reality, cloud computing is intricate, and its quirky details affect reduction potentials in unexpected ways, resulting in a causality that is not so straightforward.

Large relative reductions appear to primarily occur when the instance type that is provisioned by Carbon Efficient Karpenter and Original Karpenter differs in instance family and/or instance generation[1]. For example, in experiment USA-A7, Carbon Efficient Karpenter provisions a `c6a.12xlarge` (71.18 gCO_2e/h) and Original Karpenter provisions a `c5.9xlarge` (144.30 gCO_2e/h). Or take Sweden-A1 for example, where Carbon Efficient Karpenter provisions a `m7i-flex.2xlarge` (0.63 gCO_2e/h) and Original Karpenter provisions a `t4g.2xlarge` (1.14 gCO_2e/h). In line with this, small relative reductions appear to occur when the instance types is of the same instance family and/or instance generation – that is, they only differ in the processor family. Take for example Ireland-A11 where Carbon Efficient Karpenter provisions a `c6a.16xlarge` (105.55 gCO_2e/h) and Original Karpenter provisions a `c6g.16xlarge` (113.10 gCO_2e/h). Here, the difference in impact is relatively small, and the two instances only differ in the processor family.

The observation that small reductions appear when instance types are similar (same instance family and/or instance generation) can be used to explain why reductions in India are consistently low. India has a lower number of available instances compared to the USA and Ireland (see table 29). This means that it is less likely that an instance type exists which is cheaper but has a higher carbon impact than the instance type which has the lowest carbon impact. That explains one part of why reductions in India are low. An extremely interesting finding is that for every configuration in India, Carbon Efficient Karpen-

1 Instance type concepts can be refreshed by revisiting section 5.3.

ter provisioned a `c6a` instance and Original Karpenter provisioned a `c6g` instance. For example, in India-A7 Carbon Efficient Karpenter provisioned a `c6a.12xlarge` (114.65 gCO_2e/h) and Original Karpenter provisioned a `c6g.12xlarge` (120.32 gCO_2e/h). This means that in India the sixth generation of the `c` instance family both contains the cheapest and lowest-impact instance type for all configurations that were tested in this thesis. Since the cheapest and lowest-impact instance type only differ in processor family, the cheapest instance type in India is actually very carbon efficient compared to the cheapest instance types in Ireland and the USA (considering the large difference in carbon intensity). For that reason, Carbon Efficient Karpenter was only able to create a small (yet still statistically significant) reduction in the carbon impact of homogeneous workloads in India.

REGION	COUNTRY	AVAILABLE INSTANCE TYPES
`us-east-1`	USA	721
`eu-west-1`	Ireland	697
`ap-south-1`	India	454
`eu-north-1`	Sweden	381

Table 29: Number of Linux-based instance types available in four Amazon Web Services (AWS) regions [101].

Sweden naturally contained the lowest median reduction potential of the regions covered ($M_{D_{\text{Sweden}}} = 2.0505\,gCO_2e/h$) because of its extremely low carbon intensity. However, in spite of that and its low number of available instances, Carbon Efficient Karpenter proved to create the largest median relative reduction in Sweden ($M_{\mathcal{R}_{\text{Sweden}}} = 30.1954\,\%$). An explanation for this could be, that there is a smaller correlation between the embodied carbon of cloud instances and their price – and since embodied carbon is a significant contributor to the carbon impact of cloud instances in Sweden, a discrepancy between instance price and carbon impact is prevalent.

Generally, it can be considered a remarkable finding that homogeneous workloads on two of the most popular AWS regions (`eu-west-1` and `us-east-1`) achieve a 19% mean relative reduction in carbon impact when using Carbon Efficient Karpenter instead of Original Karpenter. Two workload configurations (A7 and A25) even reach a 50% reduction in carbon impact.

9.2 ON THE BENEFITS OF ADOPTING CARBON EFFICIENT KARPENTER

The four real clusters that have been covered in the second experiment in this thesis are expected to achieve a carbon impact reduction of 20 %, 73 %, 46 % and 42 %, respectively, if they migrate to Carbon Ef-

ficient Karpenter. None of the clusters covered in this experiment use Karpenter at the moment. It can not be proved with the current sample size ($N = 4$) that other clusters will experience a similar reduction in carbon impact if they adopt Carbon Efficient Karpenter. However, it would be surprising if Carbon Efficient Karpenter couldn't induce any reduction in the carbon impact of other clusters. The magnitude of reductions achieved by adopters of Carbon Efficient Karpenter will be contingent upon a multitude of factors. Over-provisioning and low utilization cause a large part of carbon emissions from Kubernetes clusters that run on cloud infrastructure. Since Karpenter in itself can mitigate over-provisioning, the act of merely adopting Karpenter would probably result in reduced emissions in many cases. For example, Grafana Labs recently switched from Cluster Autoscaler to Karpenter and saw utilization jump from approximately 70% to 85% (an increase of 15 percentage points) [102]. A cluster with no cluster autoscaler employed may be extensively over-provisioned, and when such a cluster adopts Carbon Efficient Karpenter, reductions attributed to both improved utilization and carbon efficiency will be experienced. Likewise, a cluster running on old instance types might see larger reductions in carbon impact than one running on new instance types, as the older instances are more likely to be carbon inefficient.

Carbon Efficient Karpenter can be combined with an array of other initiatives to boost carbon reductions. It can be hard to determine appropriate resource requests and limits for pods in Kubernetes. A Vertical Pod Autoscaler can help rightsize workloads. The combination of rightsized Pods, carbon efficient cloud instances, and high resource utilization is an excellent recipe for creating a cluster with high carbon efficiency.

Organizations that wish to take their carbon efficiency to new heights could consider migrating their cluster to a region with a low-carbon electricity grid. For example, moving a cluster from Ireland to Sweden can easily cut emissions by 75% without reducing the compute and storage resources available to workloads. Karpenter does not support multi-region clusters [95]. Nevertheless, it can help further reduce the carbon impact of clusters that have migrated to a region with low-carbon electricity.

It is anticipated that many organizations will express an interest in a financial comparison of Carbon Efficient Karpenter and Original Karpenter. Although the economic ramification of adopting Carbon Efficient Karpenter is a legitimate concern, such financial evaluations fall beyond the scope of this thesis.

9.3 SUPPLY CHAIN EFFECTS OF WIDESPREAD CARBON EFFICIENT CLUSTER AUTOSCALING

Multiple benefits can be derived from widespread adoption of Carbon Efficient Cluster Autoscaling in Kubernetes clusters that run on cloud infrastructure. Initially, a large shift in demand towards carbon efficient cloud instances would be observed. If the shift in demand is large enough, cloud providers would presumably accommodate the demand by acquiring more carbon efficient hardware. As a result of this, a trickle-down effect could occur, incentivizing the entire supply chain to develop more carbon efficient products. In an attempt to catch this new-born market for carbon efficient hardware, some producers would probably increase research in carbon efficiency and launch new products fitting this market.

However, when optimizing embodied carbon in relation to operational carbon, it can introduce some harmful incentives [39]. In this regard, it is advantageous that operational emissions are estimated to be the most significant contributor to total carbon emissions in most regions. A bottom-up approach to the estimation of embodied carbon may mitigate compounding inaccuracy and double-counting of embodied carbon. At large, this may mitigate the risk of harmful incentives.

Carbon Efficient Cluster Autoscalers can motivate more transparency regarding the carbon impact of cloud services and their components. This can be done by establishing a fundamental transparency standard and requiring it as a mandatory condition in the provisioning process.

9.4 COMPARISON TO EXISTING RESEARCH

In contrast to most works, Carbon Efficient Karpenter reduces both operational and embodied carbon emissions. This might result in more accurate scaling compared to some other works.

Many carbon-aware systems limit their scope to short-running non-critical workloads. Since Carbon Efficient Karpenter manages the cloud instances that all workloads in a Kubernetes cluster run on, it breaks free from the limitation to short-running non-critical workloads.

Using the terminology about carbon awareness and carbon efficiency from chapter 3, Carbon Efficient Karpenter could exhibit carbon awareness, but it doesn't.

In contrast to most research in carbon-aware cloud computing, Carbon Efficient Karpenter is not a research proof of concept. It is production-ready and capable of being used in real clusters, as is. I intended to incorporate Carbon Efficient Karpenter into upstream Karpenter as a carbon-efficiency feature, making the fruits of this master's thesis readily accessible to Kubernetes Cluster Administrators.

10 OUTLOOK

This thesis introduced a novel concept: *carbon efficient cluster autoscaling*. Carbon Efficient Karpenter is the first carbon efficient cluster autoscaler known to exist. The results presented in this thesis suggest that carbon efficient cluster autoscaling is a promising avenue to further explore, given a desire to improve the environmental sustainability of Kubernetes clusters in the cloud. An array of unanswered questions arises from the contributions of this thesis. This chapter brings some of these questions forward.

10.1 REDUCTION POTENTIAL ON OTHER CLOUD PLATFORMS

In the present thesis, all experiments have been conducted on Amazon Web Services (AWS). It is expected that at least some reductions could also be achieved on other cloud platforms, for example, Microsoft Azure and Google Cloud Platform. However, it is not verified, and therefore, we can not be sure that this is the case. Is the reduction potential larger, smaller, or the same on other cloud platforms? Currently, that question remains unanswered.

10.2 MULTI-CRITERIA IMPACT EVALUATION

This thesis focused on greenhouse gas emissions from Kubernetes clusters and how to reduce them (measured in carbon equivalents). However, environmental sustainability is not only about greenhouse gas emissions. It is also about water resources, land use, minerals, nature, wildlife, etc. For a cluster to be truly environmentally sustainable many factors are to be considered. A natural extension of the present work is therefore to leverage a multi-criteria impact evaluation. In principle, BoaviztAPI supports multiple impact criteria such as those just mentioned. However, the modeling of the other impacts in BoaviztAPI is not nearly as mature as the *Global Warming Potential* impact criteria. Optimizing for multiple impact criteria begs the question of how different criteria should be balanced against each other.

10.3 ESTIMATION METHODOLOGIES

In this thesis, BoaviztAPI was used to estimate the carbon impact of cloud instances. The methodology is sound, but it is not the only one. Other methods exist, and some have a larger focus on operational

carbon. Could other methodologies bring about larger carbon reductions?

The carbon intensity has a significant impact on the carbon emissions from cloud instances. In the present thesis, a static carbon intensity is used. Could larger (or more accurate) reductions be achieved by using a real-time carbon intensity?

10.4 COMBINATION WITH CARBON-AWARE SCHEDULING

Some works already explore the concept of carbon-aware scheduling in Kubernetes. Carbon Efficient Karpenter is not carbon aware, but its interactions with a carbon-aware scheduling systems would be interesting to uncover. Could the two approaches collaborate to achieve a compound reduction?

10.5 FINANCIAL ANALYSIS OF CARBON EFFICIENT KARPENTER

How does Carbon Efficient Karpenter compare to Original Karpenter financially? That question has been asked in almost every professional conversation I have had about my thesis. Rightfully so, because it is a very legitimate question. Corporations face a variety of methods that convert money into emission reductions. If they had to choose between investing in a solar panel for the office roof or adopting Carbon Efficient Karpenter, which one would bring about the best return on investment (in carbon equivalents)? Currently, there is no clear answer to that question. If the cluster is over-provisioned, adopting Carbon Efficient Karpenter might reduce both cost and emissions. But if Original Karpenter is already adopted, the answer might not be that straightforward.

11 CONCLUSION

This thesis aimed to explore how Karpenter could be improved to minimize the carbon impact of Kubernetes clusters in the cloud and to evaluate the effect of such improvements.

Karpenter is designed to minimize financial costs, but by substituting cloud instance prices with carbon impacts, it was adapted to minimize carbon emissions. Since no primary source disclose the carbon impact of cloud instances, estimates from BoaviztAPI were used.

An analysis of operational and embodied carbon emissions in various scenarios indicated that both should be minimized when global and future applicability is desirable.

The improvement of Karpenter was evaluated in 35 experiments with different homogeneous workloads in four cloud regions. It showed a statistically significant median reduction in the carbon impact of the provisioned cloud instances. The lower and upper quartile carbon impact reductions was (Q_1, Q_3): Ireland $(7\%, 32\%)$, Sweden $(26\%, 45\%)$, USA $(7\%, 32\%)$, and India $(4\%, 5\%)$. The improvement was also evaluated in four cloud regions, with workloads replicating the workloads in four real Kubernetes clusters. This evaluation did not refute the previous findings, suggesting that the proposed solution can reduce the carbon impact of most Kubernetes cloud clusters. Widespread adoption may incentivize carbon emission reductions throughout the cloud computing supply chain.

The novel concept of *carbon efficient cluster autoscaling* is introduced, defined, and explored in this thesis, generating an array of new research questions waiting to be explored.

11.1 CONTRIBUTIONS

Key scientific contributions in this thesis:

- A thorough evaluation, critical analysis, and summation of existing literature on Green Cloud Computing, providing a comprehensive overview of the field.
- A novel method for reducing the carbon footprint of Kubernetes clusters, termed *Carbon Efficient Cluster Autoscaling*.
- *Carbon Efficient Karpenter*, a cluster autoscaler designed for carbon efficiency.
- An experimentation suite to facilitate automated and reproducible carbon impact evaluation of cluster autoscalers.

- Extensive evaluations of the carbon impact of both Carbon Efficient Karpenter and Original Karpenter across 35 random workload configurations in four separate cloud regions.
- An assessment of the carbon impact of Carbon Efficient Karpenter and Original Karpenter on workloads derived from four actual clusters in four separate cloud regions.
- An analysis comparing Carbon Efficient Karpenter and Original Karpenter, based on the aforementioned evaluations.
- A discussion of the potential for carbon efficient cluster autoscaling in the context of green cloud computing.
- An array of new research questions emerging from this thesis.

APPENDIX

A

DESIGN DOCUMENT

This document is converted from markdown to LaTeX using an online tool. The original is located at https://github.com/JacobValdemar/carbon-efficient-karpenter/blob/design/carbon-aware/designs/carbon-aware.md.

A.1 CARBON EFFICIENT KARPENTER: OPTIMIZING KUBERNETES CLUSTER AUTOSCALING FOR ENVIRONMENTAL SUSTAINABILITY

Author: @JacobValdemar

A.1.1 *Context & Problem*

There is a growing concern about the environmental impact of Kubernetes clusters. Karpenter's opportunities within environmental sustainability is referenced in multiple comments that back `karpenter-core`'s move to CNCF.

I am currently working on my master's thesis in Computer Engineering (Master of Science in Engineering) at Aarhus University located in Denmark. The objective of the thesis is to enable Karpenter to minimize carbon emissions from Kubernetes clusters that run on cloud infrastructure (scoped to AWS).

RFC: https://github.com/kubernetes-sigs/karpenter/issues/675

A.1.2 *Fundamentals of Green Software*

I will try to keep it simple. The reader should be familiar with the following.

A cluster's emissions is made of two elements: embodied emissions and operational emissions. To get the total emissions, one can add them togeather.

- **Embodied carbon emissions**: Manufacturing emissions (CO_2e) amortized over instance lifetime (usually 4 years) divided by how long we use the instance
- **Operational carbon emissions**: Carbon emitted by electricity grid to produce electricity for the instance in the region where it is used, multiplied by PUE

Feature	Default	Config Key	Stage	Since	Until
Carbon-Efficient	false	featureGates-.carbon-Efficient-Enabled	Alpha	?	?

An instance is a share of a physical machine, so when we refer to "instance lifetime" we mean the lifetime of the physical machine that the instance partially constitutes.

There is a lot more to Green Software. If you want to learn more, I recommend you to visit Green Software Practitioner (a Green Software Foundation project - an affiliate of the Linux Foundation).

A.1.3 *Solution*

A.1.3.1 *Feature Gate*

The feature is proposed to be controlled using a feature gate.

A.1.3.2 *Carbon emissions data source*

Currently the best option is to create estimates based on the methodology used in BoaviztAPI.

Try out BoaviztAPI on the Datavizta demo website.

LICENSING BoaviztAPI is licensed under `GNU Affero General Public License v3.0`. Therefore, as far as I know, we must license their data under the same license if used in the Karpenter repository.

LIMITATIONS Currently there is no discrepancy between the instances known by BoaviztAPI and Karpenter. However, it will probably not always be possible to have an updated list of estimated carbon emissions for all instances as AWS continues to release new instance types. We should consider what to do with instance types that we do not have carbon emission estimates for.

Approaches to handle this:

1. Estimate extremely high emissions to effectively filter out unknown instance types (recommended)
2. Estimate zero emissions

A.1.3.3 *Launch strategy*

To enable emission based priotization, the launch strategy should be changed from `lowest-price` to `prioritized`.

A.1.3.4 *Changes to consolidation (karpenter-core)*

Single Machine Consolidation (`singlemachineconsolidation.go`) and Multi Machine Consolidation (`multimachineconsolidation.go`) as well as `consolidation.go` is currently consolidating nodes to reduce costs. We want to change this when Carbon Efficient is enabled. They should consolidate to minimize carbon emissions.

A.1.3.5 *Changes to Provisioning*

Currently, provisioning (roughly) filter instances based on requirements, sort instances by price, and launch the cheapest instance. We want to change this when Carbon Efficient is enabled. It should sort instances by carbon emissions and launch the instance which has the lowest Global Warming Potential[1].

A.1.3.6 *Option 1: Use Carbon Efficient provisioning and consolidation methods*

CONSOLIDATION Create two new consolidation methods `carbonefficientsinglemachineconsolidation.go` and `carbonefficientmultimachineconsolidation.go` that will be used when Carbon Efficient is enabled.

[Code listing omitted]

PROVISIONING In `karpenter-core`, create a new method `types.go/OrderByCarbonEmissions` and use that in `nodeclaimtemplate.go/ToMachine` and `nodeclaimtemplate.go/ToNodeClaim` instead of `types.go/OrderByPrice` when Carbon Efficient is enabled.

In `karpenter`, create a new method `CarbonEfficientCreate` in `pkg/providers/instance/instance.go` that is used in `pkg/cloudprovider/cloudprovider.go/Create` instead of `pkg/providers/instance/instance.go/Create` when Carbon Efficient is enabled.

CONSIDERATIONS

1. + Current consolidation methods are unaffected.
2. − There might be copy-paste of code from the original consolidation methods to the carbon efficient consolidators.

A.1.3.7 *Option 2: Use Carbon Efficient filtering/sorting methods*

CONSOLIDATION Create carbon efficient implementations of low-level functions like `filterByPrice`, `filterOutSameType`, `getCandidatePrices`,

1 The potential impact of greenhouse gases on global warming. Measured in terms of CO_2e.

etc. that are used when Carbon Efficient is enabled. Usage of aforementioned functions might assume that it is price that they are getting, but in reality it is data about carbon emissions.

PROVISIONING Use same changes to provisioning as in option 1.

CONSIDERATIONS

1. + Less code copy-paste.
2. + Improvements to original consolidation methods also improve the Carbon Efficient feature.
3. − Has a risk of breaking undocumented invariants.
4. − Adds complexity to the original consolidation methods.

A.1.3.8 *Option 3: Override instance price with carbon price (recommended)*

Minimize carbon emissions by defining a price per $kgCO_2e$ and override the instance price with the carbon price ($USD/kgCO_2e$). Using the `prioritized` launch strategy, carbon emissions will be minimized during provisioning. Consolidation will unknowingly consolidate to minimize carbon emissions.

The carbon price will depend on `region` and `instanceType` and assume constant resource utilization (e.g. always 80% utilization). The carbon price will be generated in a "hack" and included as consts (same method as used for generating initial pricing[2]). The carbon price / emission estimates can be updated with new versions.

Another feature (added later) can be to add carbon price to instance price to simulate a carbon tax. Administrators could configure a custom carbon price or use a default.

CONSIDERATIONS

1. + Change is constrained to the pricing domain, so most of Karpenter's logic remains unaffected.
2. ++ A simulated carbon tax could be appealing for *Beta* or *General Availability*[3] as it combines the real price with the carbon price.
3. − Adds complexity to the *price* concept. Price is not *just* price, but rather becomes an optimization function.
4. − Depending on implementation, the `karpenter_cloudprovider_instance_type_price_estimate` metric *may* represent more than just price when Carbon Efficient is enabled.

2 See prices_gen.go and zz_generated.pricing.go

3 https://kubernetes.io/docs/reference/command-line-tools-reference/feature-gates/#feature-stages

A.1.3.9 *Option 4: Enable custom instance price overrides*

Enable administrators to configure custom instance price overrides, e.g. in a ConfigMap. A configuration using emission factors (varying with region and instance type) masked as prices can be pre-generated. Administrators then copy-paste a Carbon Efficient `priceOverride` into their environment.

[Code listing omitted]

Alternatively, a more flexible interface could be:

[Code listing omitted]

A ConfigMap with price overrides for all combinations of instance types and regions will be very huge. 632 instances * 29 regions = 18,328 pairs. Four lines per pair gives a file with 73,312 lines. The file/configmap will approximately have a size of 2 MB. That exceeds the `1 MiB` limit on ConfigMap size in Kubernetes.

CONSIDERATIONS

1. + Simple solution.
2. + Can be used for other purposes.
3. −− ConfigMap cannot contain all data.
4. − Hard to discover the carbon efficient "feature".
5. − Carbon emission price cannot be combined with actual price.
6. − Carbon emissions are completely static without possibility to improve it in the future.
7. − Feature can not be enabled as a toggle.
8. − Depending on implementation, the `karpenter_cloudprovider_instance_type_price_estimate` metric *may* represent more than just price when Carbon Efficient is enabled.

B

CLUSTER SETUP

This appendix describes how an Kubernetes Cluster on Amazon Elastic Kubernetes Service (EKS) is created for the purpose of running the experiments in this thesis and reproduce the results. Prerequisites:

- You have access to an Amazon Web Services (AWS) account
- `kubectl`, the `go` toolchain, `eksctl`, and the AWS CLI is installed
- Your computer's operating system is either macOS or GNU/Linux-based

To create a cluster follow the steps shown below.

```
4  export AWS_DEFAULT_REGION="eu-north-1" # Change this if you want
       to create a cluster in another region
5
6  export KARPENTER_VERSION=v0.31.0
7  export AWS_PARTITION="aws"
8  export CLUSTER_NAME="karpenter-test-$AWS_DEFAULT_REGION"
9  export AWS_ACCOUNT_ID="$(aws sts get-caller-identity --query
       Account --output text)"
10 export TEMPOUT=$(mktemp)
11
12 curl -fsSL https://raw.githubusercontent.com/aws/karpenter/"${
       KARPENTER_VERSION}"/website/content/en/preview/getting-
       started/getting-started-with-karpenter/cloudformation.yaml  >
        $TEMPOUT \
13 && aws cloudformation deploy \
14   --stack-name "Karpenter-${CLUSTER_NAME}" \
15   --template-file "${TEMPOUT}" \
16   --capabilities CAPABILITY_NAMED_IAM \
17   --parameter-overrides "ClusterName=${CLUSTER_NAME}"
19 eksctl create cluster -f - <<EOF
20 #---
21 apiVersion: eksctl.io/v1alpha5
22 kind: ClusterConfig
23 metadata:
24   name: ${CLUSTER_NAME}
25   region: ${AWS_DEFAULT_REGION}
26   version: "1.27"
27   tags:
28     karpenter.sh/discovery: ${CLUSTER_NAME}
29
30 iam:
31   withOIDC: true
32   serviceAccounts:
33   - metadata:
```

```
      name: karpenter
      namespace: karpenter
    roleName: ${CLUSTER_NAME}-karpenter
    attachPolicyARNs:
    - arn:${AWS_PARTITION}:iam::${AWS_ACCOUNT_ID}:policy/
        KarpenterControllerPolicy-${CLUSTER_NAME}
    roleOnly: true

iamIdentityMappings:
- arn: "arn:${AWS_PARTITION}:iam::${AWS_ACCOUNT_ID}:role/
    KarpenterNodeRole-${CLUSTER_NAME}"
  username: system:node:{{EC2PrivateDNSName}}
  groups:
  - system:bootstrappers
  - system:nodes

managedNodeGroups:
- instanceType: t4g.medium
  amiFamily: AmazonLinux2
  name: ${CLUSTER_NAME}-ng
  desiredCapacity: 2
  minSize: 1
  maxSize: 5
EOF
export CLUSTER_ENDPOINT="$(aws eks describe-cluster --name ${
    CLUSTER_NAME} --query "cluster.endpoint" --output text)"
export KARPENTER_IAM_ROLE_ARN="arn:${AWS_PARTITION}:iam::${
    AWS_ACCOUNT_ID}:role/${CLUSTER_NAME}-karpenter"
echo $CLUSTER_ENDPOINT $KARPENTER_IAM_ROLE_ARN

aws ecr create-repository \
    --repository-name karpenter-test \
    --image-scanning-configuration scanOnPush=true \
    --region "${AWS_DEFAULT_REGION}"

export KO_DOCKER_REPO="${AWS_ACCOUNT_ID}.dkr.ecr.${
    AWS_DEFAULT_REGION}.amazonaws.com/karpenter-test"
aws ecr get-login-password --region "${AWS_DEFAULT_REGION}" |
    docker login --username AWS --password-stdin "${
    KO_DOCKER_REPO}"

aws eks update-kubeconfig --region $AWS_DEFAULT_REGION --name
    $CLUSTER_NAME
kubectl create ns karpenter
kubectl config set-context --current --namespace=karpenter
```

```
# To run the tests:
# (in carbon-efficient-karpenter repository)

make apply

# cordon nodes
# kubectl cordon

make carbontest TEST_TIMEOUT=60m
```

C

EXPERIMENTS

C.1 EXPERIMENT: HOMOGENEOUS WORKLOADS

The configuration for A1...A35 can be found in table 27 on page 70.

C.1.1 *Configuration*

The distribution of the configuration parameters is visualized in figs. 34 and 35.

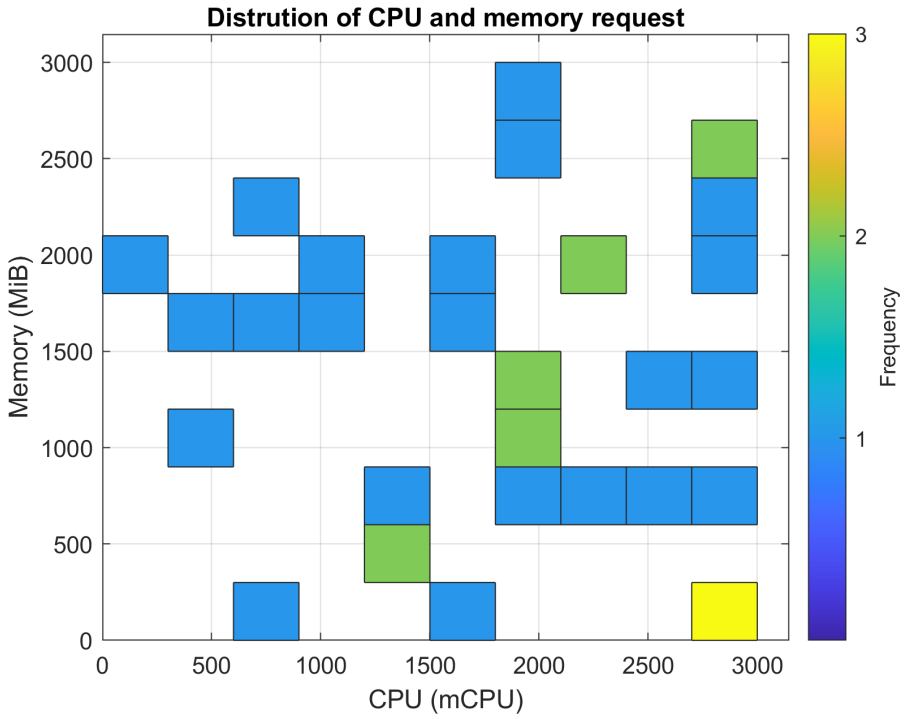

Figure 34

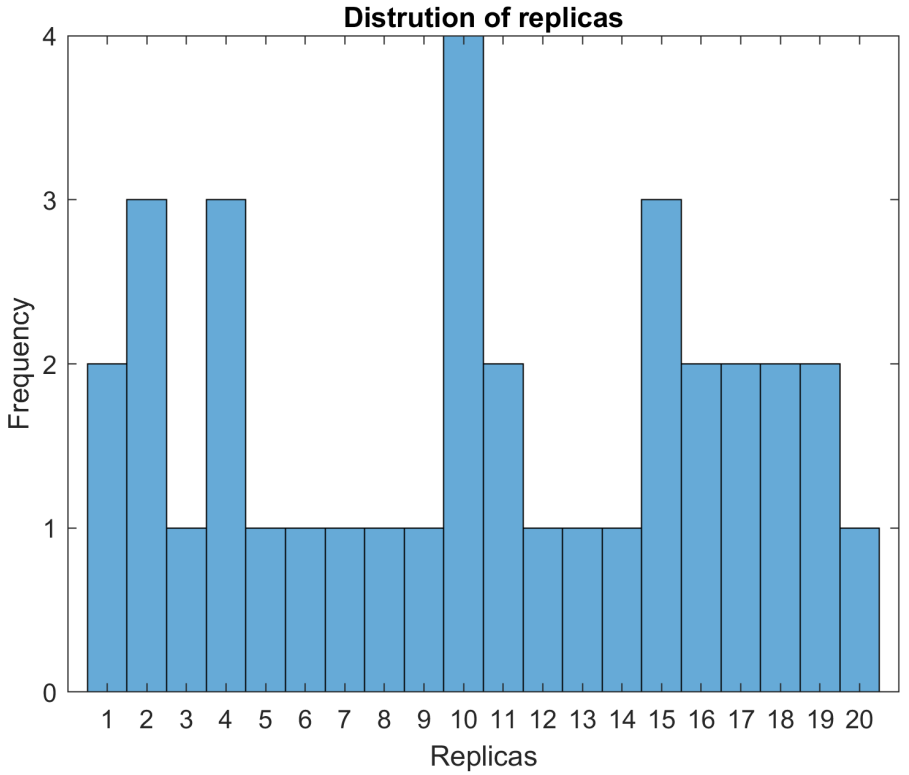

Figure 35

C.1.2 *Result*

The raw carbon impacts for this experiment are listed in table 30.

SETUP		RESULT (gCO_2e/h)	
COUNTRY	CONFIG	ORIGINAL	CARBON EFFICIENT
INDIA	A1	6.3383	6.1208
INDIA	A2	21.5371	20.5930
INDIA	A3	77.9348	74.1574
INDIA	A4	42.2062	40.3174
INDIA	A5	163.7126	156.1576
INDIA	A6	128.9914	123.3293
INDIA	A7	120.3180	114.6496
INDIA	A8	19.5045	18.5595
INDIA	A9	20.4724	19.5280
INDIA	A10	75.4371	71.6568
INDIA	A11	172.6046	165.0496
INDIA	A12	83.9919	80.2130
INDIA	A13	85.0076	81.2287
INDIA	A14	43.5945	41.7064
INDIA	A15	10.2091	9.7370
INDIA	A16	85.7278	81.9503
INDIA	A17	86.1607	82.3832
INDIA	A18	11.9533	11.5186
INDIA	A19	39.2672	37.3784
INDIA	A20	40.5431	38.6536
INDIA	A21	75.9367	72.1592
INDIA	A22	71.2576	67.4801
INDIA	A23	165.4277	157.8727
INDIA	A24	10.2841	9.8119
INDIA	A25	118.7381	113.0719
INDIA	A26	18.4981	17.5541
INDIA	A27	15.0024	14.5674
INDIA	A28	83.1135	79.3332
INDIA	A29	82.6972	78.9197
INDIA	A30	20.1758	19.2318
INDIA	A31	125.8442	120.1821
INDIA	A32	9.1185	8.6463

Table 30, continued on following page

Table 30, continued from previous page

COUNTRY	CONFIG	ORIGINAL	CARBON EFFICIENT
INDIA	A33	37.7602	35.8700
INDIA	A34	10.4438	9.9718
INDIA	A35	168.7414	161.1864
SWEDEN	A1	1.1410	0.6317
SWEDEN	A2	4.6134	2.6113
SWEDEN	A3	13.7326	9.5471
SWEDEN	A4	7.0733	5.1277
SWEDEN	A5	27.9664	26.3914
SWEDEN	A6	21.3714	20.2873
SWEDEN	A7	25.9021	19.5973
SWEDEN	A8	4.4835	2.3890
SWEDEN	A9	4.5454	2.4949
SWEDEN	A10	13.5730	9.2739
SWEDEN	A11	28.5346	27.0986
SWEDEN	A12	14.1197	10.2091
SWEDEN	A13	14.1846	10.3203
SWEDEN	A14	7.1620	5.2796
SWEDEN	A15	2.2710	1.2445
SWEDEN	A16	14.2306	10.3991
SWEDEN	A17	14.2583	10.4466
SWEDEN	A18	2.2357	1.1842
SWEDEN	A19	6.8855	4.8064
SWEDEN	A20	6.9670	4.9459
SWEDEN	A21	13.6050	9.3287
SWEDEN	A22	13.3060	8.8172
SWEDEN	A23	28.0760	26.5276
SWEDEN	A24	2.2757	1.2527
SWEDEN	A25	25.6739	19.4714
SWEDEN	A26	4.4192	2.2791
SWEDEN	A27	3.2560	1.6770
SWEDEN	A28	14.0635	10.1132
SWEDEN	A29	14.0369	10.0680
SWEDEN	A30	4.5264	2.4625
SWEDEN	A31	21.1703	20.0370
SWEDEN	A32	2.2013	1.1252

Table 30, continued on following page

Table 30, continued from previous page

COUNTRY	CONFIG	ORIGINAL	CARBON EFFICIENT
SWEDEN	A33	6.7892	4.6415
SWEDEN	A34	2.2860	1.2702
SWEDEN	A35	28.2878	26.7911
IRELAND	A1	4.1920	3.9745
IRELAND	A2	18.9319	13.1707
IRELAND	A3	51.4213	47.6438
IRELAND	A4	37.2958	25.8086
IRELAND	A5	107.6536	100.0986
IRELAND	A6	84.5476	78.8839
IRELAND	A7	149.3200	73.5597
IRELAND	A8	17.6017	11.9233
IRELAND	A9	18.2352	12.5174
IRELAND	A10	49.8891	46.1100
IRELAND	A11	113.1081	105.5531
IRELAND	A12	55.1368	51.3585
IRELAND	A13	55.7599	51.9815
IRELAND	A14	38.2049	26.6607
IRELAND	A15	9.1000	6.2421
IRELAND	A16	56.2016	52.4242
IRELAND	A17	56.4672	52.6897
IRELAND	A18	7.9403	7.5054
IRELAND	A19	35.3729	24.0058
IRELAND	A20	36.2080	24.7880
IRELAND	A21	50.1956	46.4181
IRELAND	A22	47.3253	43.5478
IRELAND	A23	108.7057	101.1507
IRELAND	A24	9.1489	6.2881
IRELAND	A25	147.1290	72.5919
IRELAND	A26	16.9436	11.3066
IRELAND	A27	10.1515	9.7165
IRELAND	A28	54.5980	50.8188
IRELAND	A29	54.3426	50.5651
IRELAND	A30	18.0414	12.3357
IRELAND	A31	82.6171	76.9534
IRELAND	A32	8.3863	5.5731

Table 30, continued on following page

Table 30, continued from previous page

COUNTRY	CONFIG	ORIGINAL	CARBON EFFICIENT
IRELAND	A33	34.3862	23.0805
IRELAND	A34	9.2536	6.3862
IRELAND	A35	110.7383	103.1834
USA	A1	4.0678	3.8503
USA	A2	18.4795	12.7413
USA	A3	49.8875	46.1100
USA	A4	36.4118	24.9693
USA	A5	104.4105	96.8555
USA	A6	81.9765	76.3127
USA	A7	144.2971	71.1826
USA	A8	17.1978	11.5394
USA	A9	17.8083	12.1119
USA	A10	48.4112	44.6320
USA	A11	109.6661	102.1112
USA	A12	53.4675	49.6892
USA	A13	54.0679	50.2895
USA	A14	37.2877	25.7902
USA	A15	8.8872	6.0399
USA	A16	54.4935	50.7160
USA	A17	54.7494	50.9719
USA	A18	7.7081	7.2733
USA	A19	34.5590	23.2321
USA	A20	35.3636	23.9859
USA	A21	48.7064	44.9289
USA	A22	45.9408	42.1633
USA	A23	105.4242	97.8693
USA	A24	8.9343	6.0842
USA	A25	142.1860	70.2500
USA	A26	16.5637	10.9452
USA	A27	9.8709	9.4359
USA	A28	52.9483	49.1692
USA	A29	52.7023	48.9248
USA	A30	17.6215	11.9368
USA	A31	80.1164	74.4526
USA	A32	8.1994	5.3953

Table 30, continued on following page

Table 30, continued from previous page

COUNTRY	CONFIG	ORIGINAL	CARBON EFFICIENT
USA	A33	33.6083	22.3406
USA	A34	9.0352	6.1788
USA	A35	107.3828	99.8278

Table 30: Raw carbon impacts for *Experiment: Homogeneous Workloads*.

C.2 EXPERIMENT: REAL CLUSTERS

C.2.1 *Configuration*

The distribution of pod's resource requests in the real cluster datasets are visualized in figs. 36 to 39. Beware, the scale of the axes is not the same in the figures.

C.2.2 *Result*

The raw carbon impacts for this experiment are listed in tables 31 to 34.

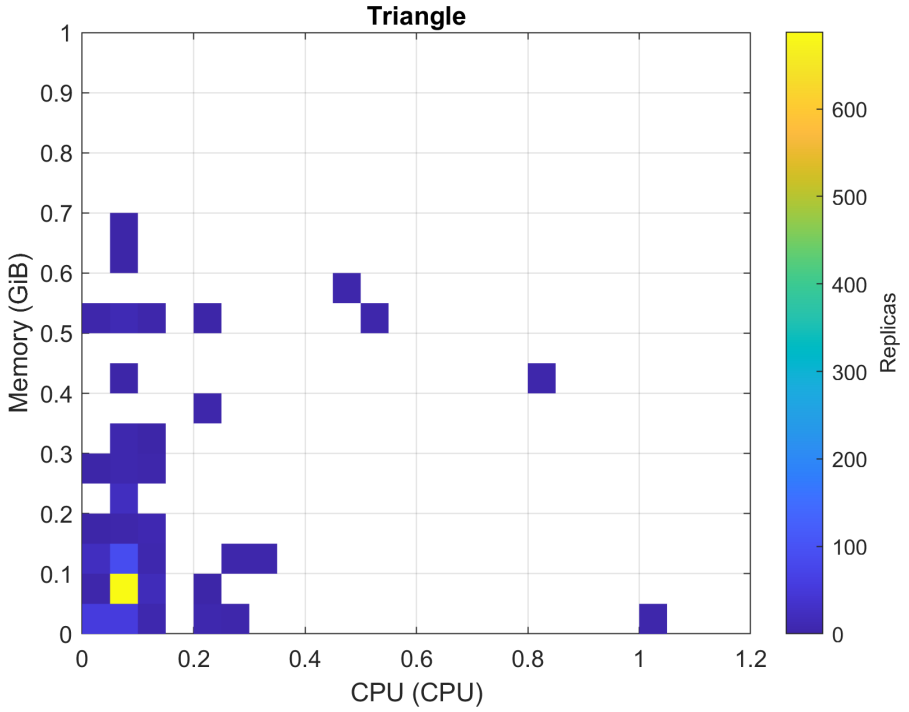

Figure 36

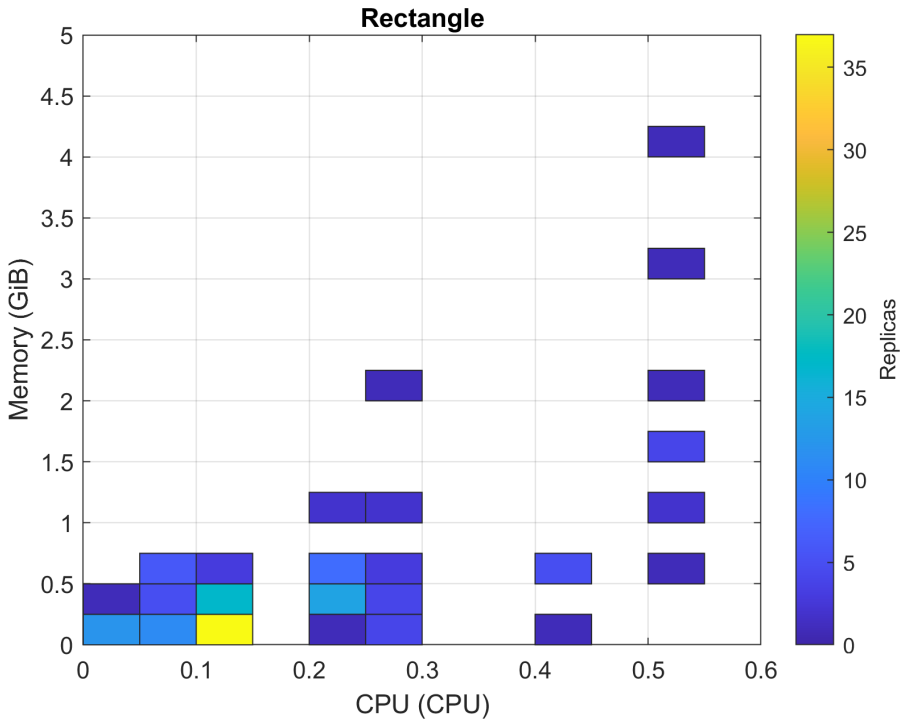

Figure 37

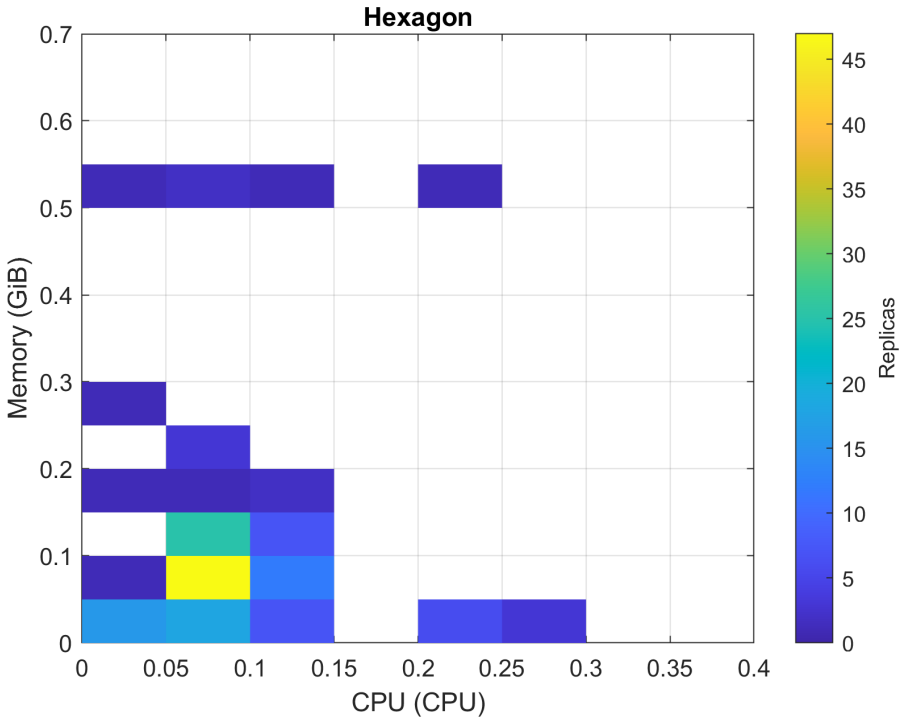

Figure 38

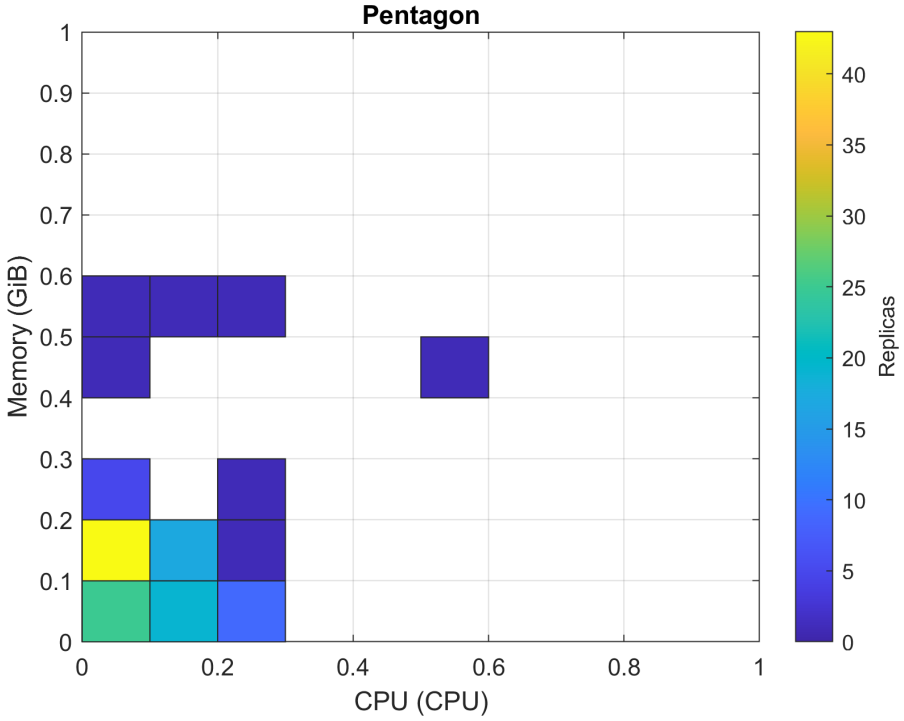

Figure 39

ANONYMOUS TRIANGLE

	CONTEXT	IMPACT (gCO_2e/h)
IRELAND	Real	201.750
	Original Karpenter	193.700
	Carbon Efficient Karpenter	161.700
SWE	Original Karpenter	40.110
	Carbon Efficient Karpenter	37.940
IND	Original Karpenter	249.790
	Carbon Efficient Karpenter	233.700
USA	Original Karpenter	190.510
	Carbon Efficient Karpenter	158.000

Table 31

ANONYMOUS RECTANGLE

	CONTEXT	IMPACT (gCO_2e/h)
GERMANY	Real	257.140
	Original Karpenter	69.380
	Carbon Efficient Karpenter	53.800
SWE	Original Karpenter	13.960
	Carbon Efficient Karpenter	13.040
IND	Original Karpenter	80.400
	Carbon Efficient Karpenter	76.670
USA	Original Karpenter	70.300
	Carbon Efficient Karpenter	47.900

Table 32

ANONYMOUS HEXAGON

	CONTEXT	IMPACT (gCO_2e/h)
IRELAND	Real	51.430
	Original Karpenter	41.029
	Carbon Efficient Karpenter	27.830
SWE	Original Karpenter	8.187
	Carbon Efficient Karpenter	7.141
IND	Original Karpenter	47.990
	Carbon Efficient Karpenter	43.780
USA	Original Karpenter	40.040
	Carbon Efficient Karpenter	26.890

Table 33

ANONYMOUS PENTAGON

	CONTEXT	IMPACT (gCO_2e/h)
IRELAND	Real	48.470
	Original Karpenter	40.830
	Carbon Efficient Karpenter	28.099
SWE	Original Karpenter	7.090
	Carbon Efficient Karpenter	6.649
IND	Original Karpenter	41.700
	Carbon Efficient Karpenter	39.800
USA	Original Karpenter	39.980
	Carbon Efficient Karpenter	27.240

Table 34

D

ANALYSIS

To show that the differences are not normally distributed in *Experiment: Homogeneous Workloads*, a histogram of the differences in each region is presented in figs. 40 to 43.

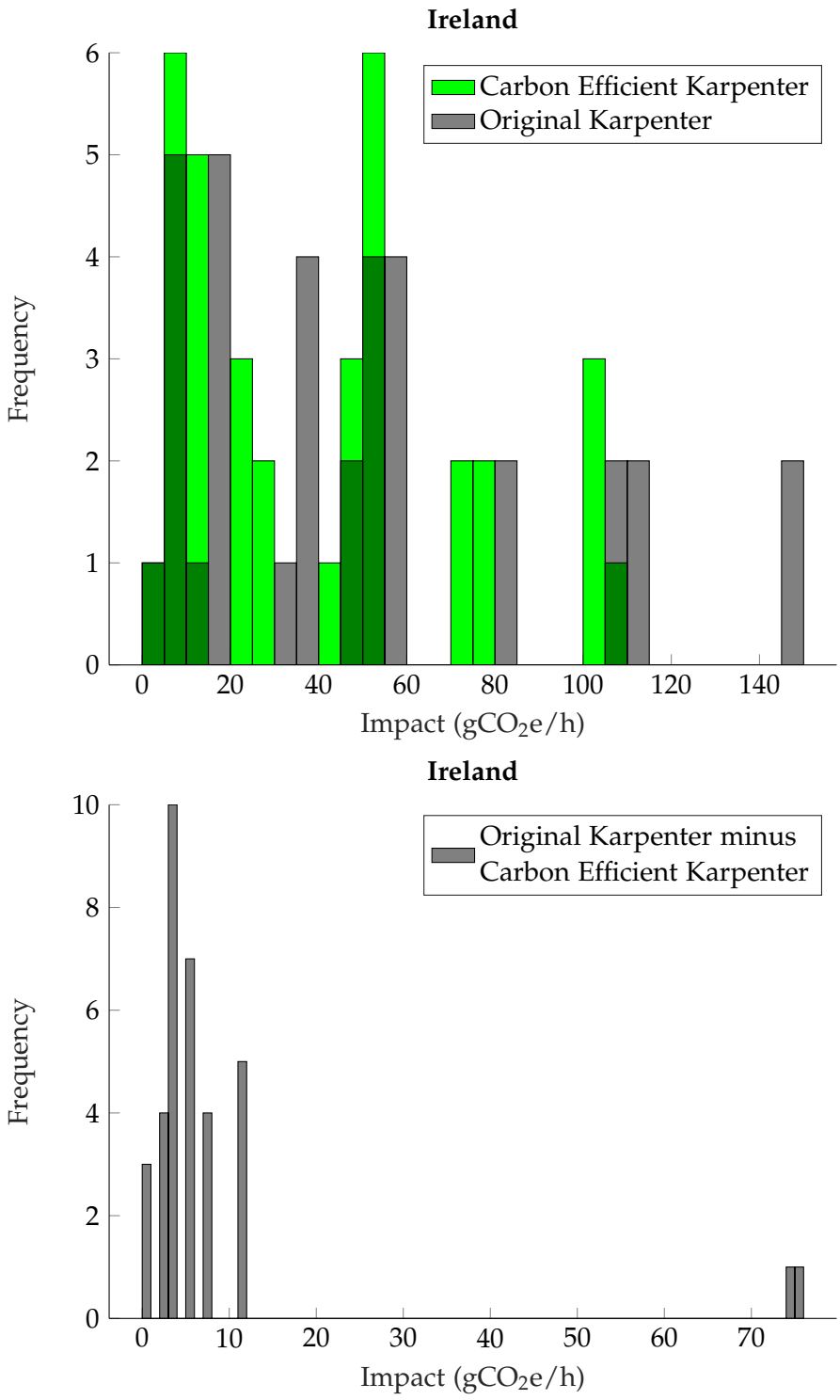
Ireland
Carbon Efficient Karpenter
Original Karpenter
Frequency
Impact (gCO2e/h)
0
1
2
3
4
5
6
0
20
40
60
80
100
120
140
Ireland
Original Karpenter minus
Carbon Efficient Karpenter
Frequency
Impact (gCO2e/h)
0
2
4
6
8
10
0
10
20
30
40
50
60
70

Figure 40

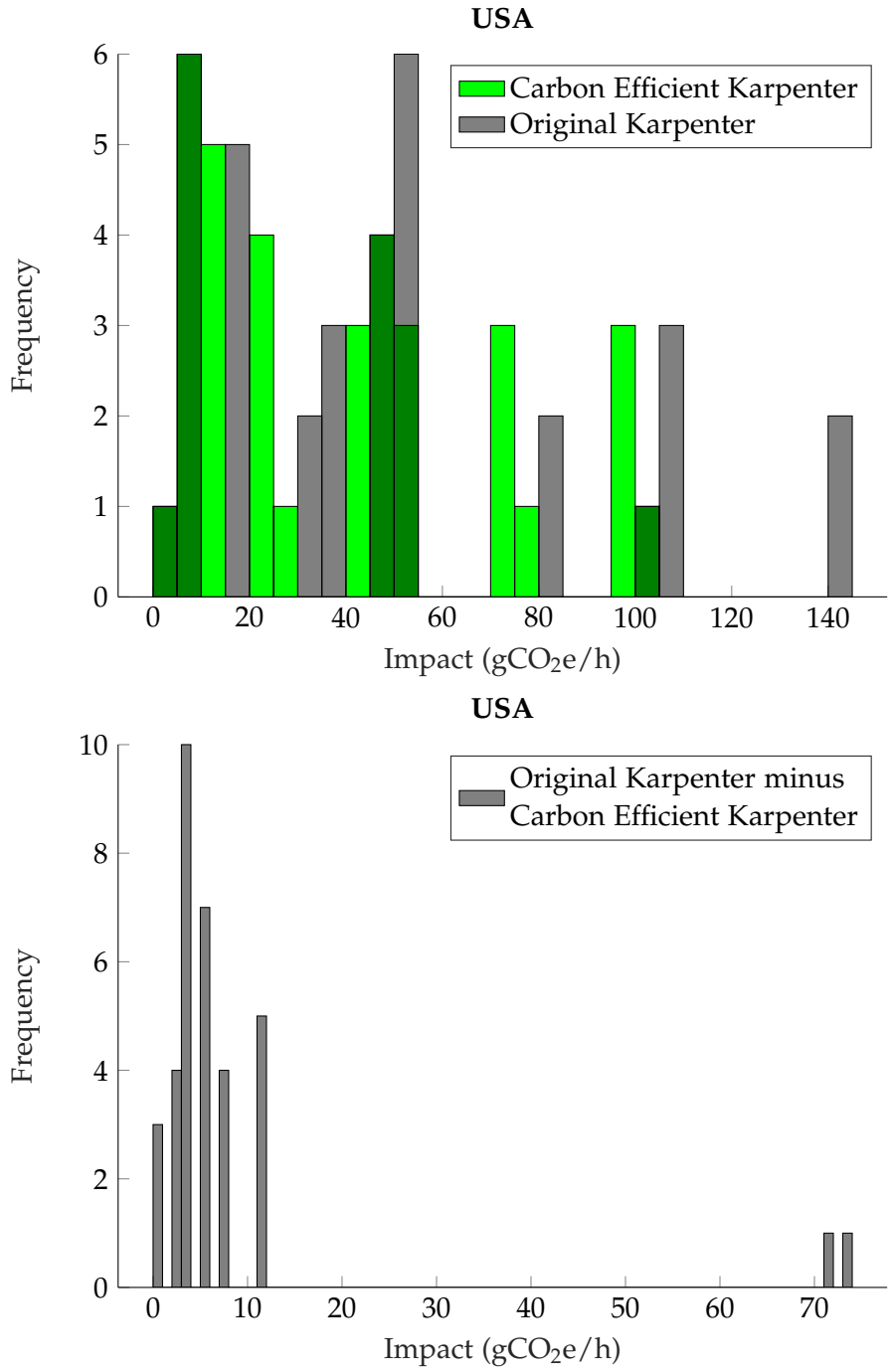
USA
Carbon Efficient Karpenter
Original Karpenter
Frequency
Impact (gCO2e/h)
USA
Original Karpenter minus
Carbon Efficient Karpenter
Frequency
Impact (gCO2e/h)

Figure 41

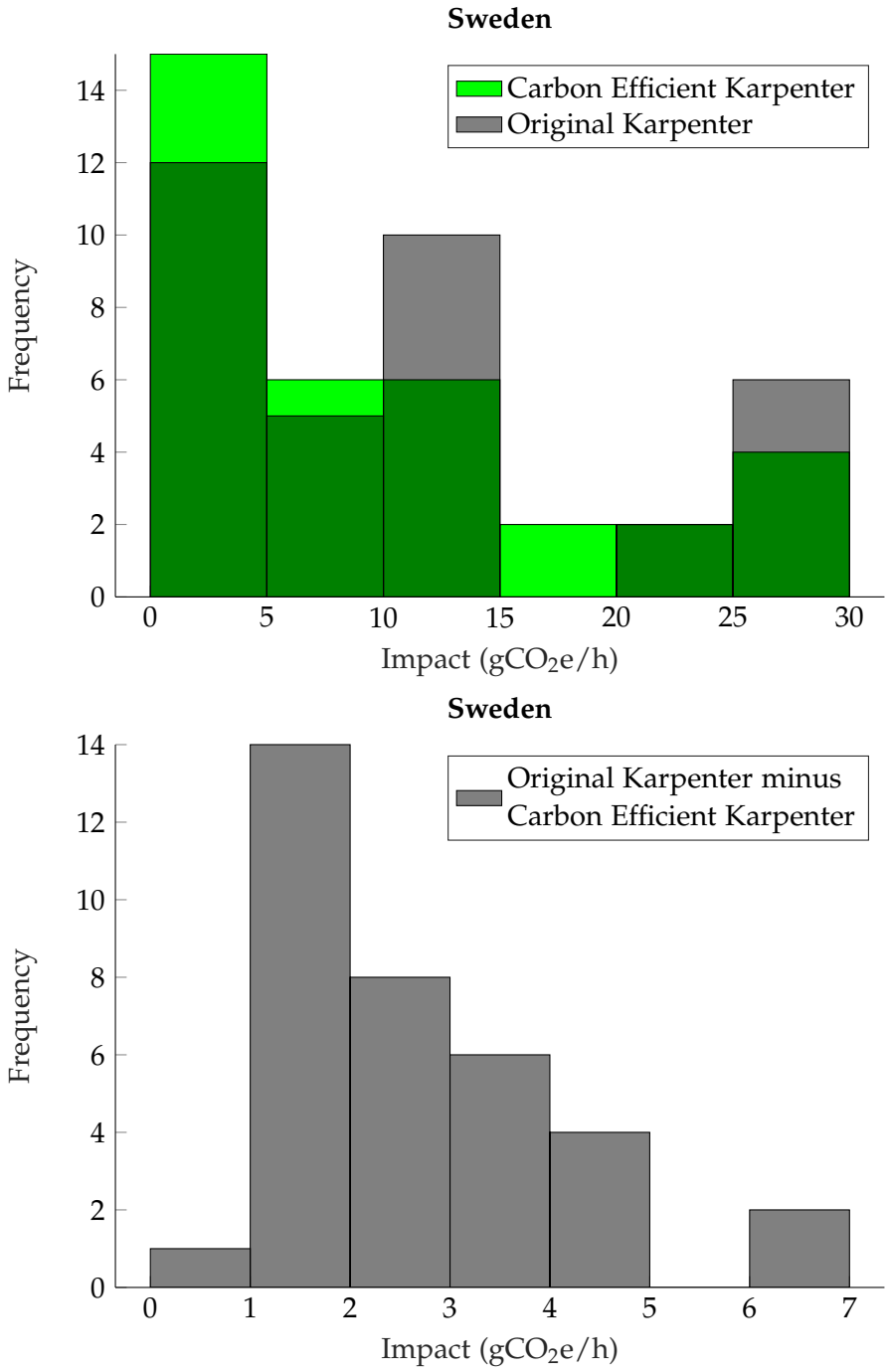
Sweden
Carbon Efficient Karpenter
Original Karpenter
Frequency
Impact (gCO2e/h)
Sweden
Original Karpenter minus
Carbon Efficient Karpenter
Frequency
Impact (gCO2e/h)

Figure 42

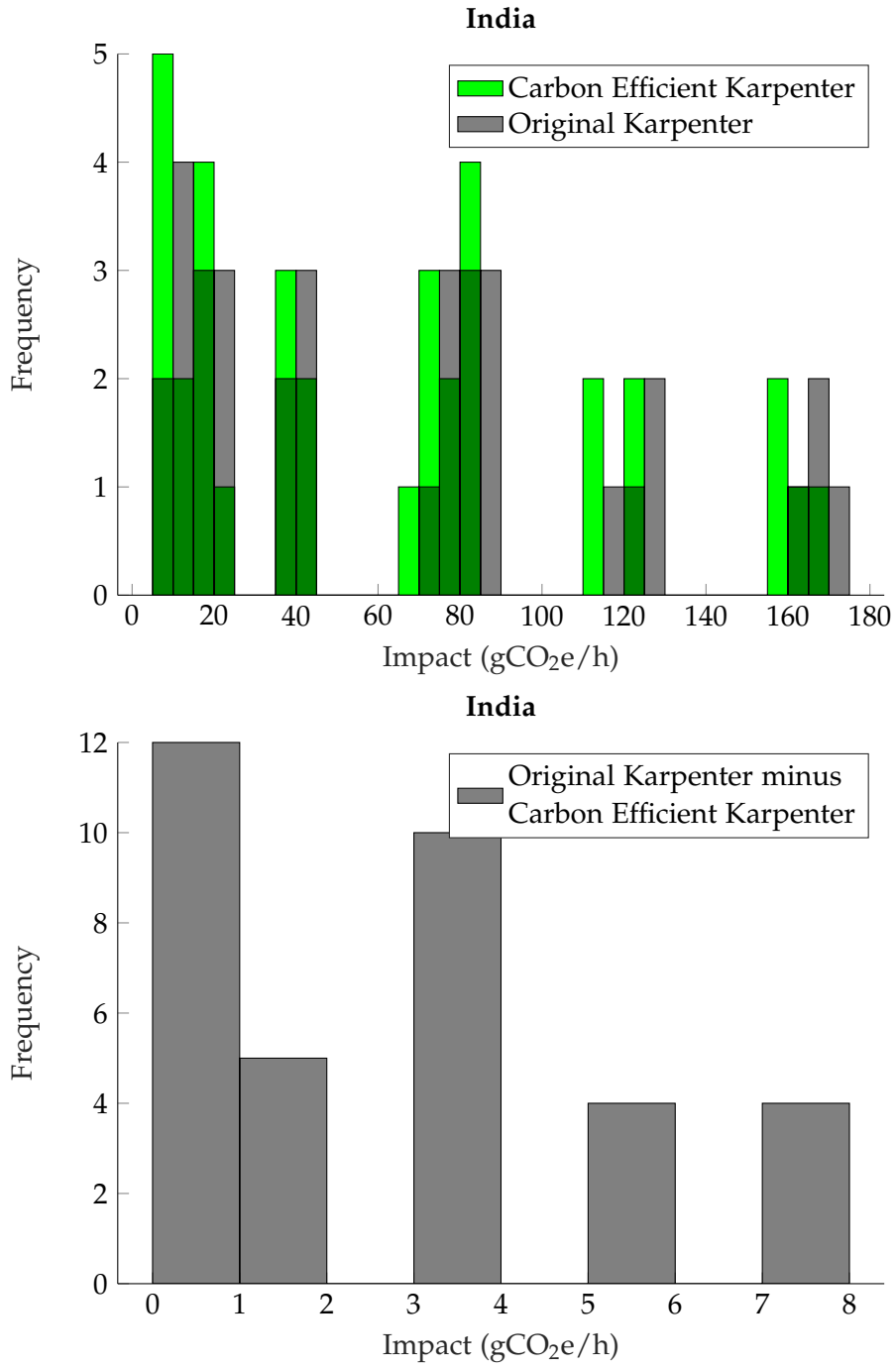
India
Carbon Efficient Karpenter
Original Karpenter
Frequency
Impact (gCO2e/h)
India
Original Karpenter minus
Carbon Efficient Karpenter
Frequency
Impact (gCO2e/h)

Figure 43

BIBLIOGRAPHY

[1] IPCC, *Climate Change 2023: Synthesis Report*. Geneva, Switzerland: IPCC, 2023. [Online]. Available: doi.org/10.59327/IPCC/AR6-9789291691647.

[2] International Energy Agency. "Data centres & networks," IEA. (Jul. 2023), [Online]. Available: https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks (visited on 09/05/2023).

[3] D. Mytton, "Hiding greenhouse gas emissions in the cloud," *Nature Climate Change*, vol. 10, no. 8, pp. 701–701, 8 Aug. 2020, ISSN: 1758-6798. DOI: 10.1038/s41558-020-0837-6.

[4] S. G. Monserrate, "The Cloud Is Material: On the Environmental Impacts of Computation and Data Storage," *MIT Case Studies in Social and Ethical Responsibilities of Computing*, Winter 2022 Jan. 27, 2022. DOI: 10.21428/2c646de5.031d4553.

[5] G. Kamiya and O. Kvarnström. "Data centres and energy – from global headlines to local headaches?" IEA. (2019), [Online]. Available: https://www.iea.org/commentaries/data-centres-and-energy-from-global-headlines-to-local-headaches (visited on 09/01/2023).

[6] J. Malmodin, N. Lövehagen, P. Bergmark, and D. Lundén. "ICT Sector Electricity Consumption and Greenhouse Gas Emissions – 2020 Outcome." (Apr. 20, 2023), preprint.

[7] A. S. G. Andrae, "New perspectives on internet electricity use in 2030," 2020.

[8] U. Gupta *et al.*, "Chasing Carbon: The Elusive Environmental Footprint of Computing," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2021, pp. 854–867. DOI: 10.1109/HPCA51647.2021.00076.

[9] Á. Hamburger, "Is guarantee of origin really an effective energy policy tool in Europe? A critical approach," *Society and Economy*, vol. 41, no. 4, pp. 487–507, Dec. 1, 2019, ISSN: 1588-9726, 1588-970X. DOI: 10.1556/204.2019.41.4.6.

[10] 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." arXiv: 2308.06680. (Aug. 13, 2023), preprint.

[11] A. Malviya and R. K. Dwivedi, "A Comparative Analysis of Container Orchestration Tools in Cloud Computing," in *2022 9th International Conference on Computing for Sustainable Global Development (INDIACom)*, Mar. 2022, pp. 698–703. DOI: 10.23919/INDIACom54597.2022.9763171.

[12] The Kubernetes Authors. "Kubernetes Website and Documentation," Kubernetes. (Sep. 4, 2023), [Online]. Available: https://kubernetes.io (visited on 09/04/2023).

[13] The Karpenter Contributers, *Karpenter Provider AWS*, version v0.31.0, Amazon Web Services, Sep. 27, 2023. [Online]. Available: https://github.com/aws/karpenter-provider-aws (visited on 12/30/2023).

[14] The Cluster Autoscaler Contributors, *Cluster Autoscaler*, version v1.28.2, Cloud Native Computing Foundation, Dec. 5, 2023. [Online]. Available: https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler (visited on 12/23/2023).

[15] The KEDA Contributors, *KEDA*, version v2.12.1, Cloud Native Computing Foundation, Nov. 27, 2023. [Online]. Available: https://github.com/kedacore/keda (visited on 12/30/2023).

[16] A. James and D. Schien, "A Low Carbon Kubernetes Scheduler," presented at the ICT for Sustainability, 2019. [Online]. Available: https://ceur-ws.org/Vol-2382/ICT4S2019_paper_28.pdf (visited on 08/28/2023).

[17] T. Piontek, K. Haghshenas, and M. Aiello, "Carbon emission-aware job scheduling for Kubernetes deployments," *The Journal of Supercomputing*, Jun. 27, 2023, ISSN: 1573-0484. DOI: 10.1007/s11227-023-05506-7.

[18] P. Yu, Y. E. Ghali, Q. Petraroia, and T. Kerkhove, *Carbon Aware KEDA Operator*, version 0.2.0, Microsoft, Apr. 18, 2023. [Online]. Available: https://github.com/Azure/carbon-aware-keda-operator (visited on 09/11/2023).

[19] D. Bianchi, *Kube-green*, in collab. with A. Quintino and L. Russo, version v0.5.2, Oct. 7, 2023. [Online]. Available: https://github.com/kube-green/kube-green (visited on 10/20/2023).

[20] R. Fairbanks. "Carbon aware spatial shifting of Kubernetes workloads using Karmada." (Jul. 12, 2023), [Online]. Available: https://rossfairbanks.com/2023/07/12/carbon-aware-spatial-shifting-with-karmada/ (visited on 08/23/2023).

[21] Green Software Foundation. "Green Software Practitioner." (Nov. 2023), [Online]. Available: https://learn.greensoftware.foundation (visited on 05/22/2023).

[22] Electricity Maps. "Electricity Maps." (2023), [Online]. Available: https://www.electricitymaps.com (visited on 08/29/2023).

[23] P. Wiesner, I. Behnke, D. Scheinert, K. Gontarska, and L. Thamsen, "Let's wait awhile: How temporal workload shifting can reduce carbon emissions in the cloud," in *Proceedings of the 22nd International Middleware Conference*, ser. Middleware '21, New York, NY, USA: Association for Computing Machinery, Dec. 2, 2021, pp. 260–272, ISBN: 978-1-4503-8534-3. DOI: 10.1145/3464298.3493399.

[24] P. Wiesner, I. Behnke, D. Scheinert, K. Gontarska, and L. Thamsen, *Let's Wait Awhile - Datasets, Simulator, Analysis*, New York, NY, USA: DOS Group at TU Berlin, Oct. 28, 2021. [Online]. Available: https://github.com/dos-group/lets-wait-awhile (visited on 09/07/2023).

[25] W. A. Hanafy, Q. Liang, N. Bashir, D. Irwin, and P. Shenoy, "CarbonScaler: Leveraging Cloud Workload Elasticity for Optimizing Carbon-Efficiency," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 7, no. 3, 57:1–57:28, Dec. 12, 2023. DOI: 10.1145/3626788.

[26] The Kubeflow Authors. "Kubeflow," Kubeflow. (2023), [Online]. Available: https://www.kubeflow.org (visited on 09/07/2023).

[27] The Karpenter Contributers. "Karpenter Documentation," Karpenter. (Jun. 2023), [Online]. Available: https://karpenter.sh/docs/ (visited on 08/28/2023).

[28] Cloud Native Computing Foundation. "Cloud Native Computing Foundation." (Sep. 14, 2023), [Online]. Available: https://www.cncf.io (visited on 09/18/2023).

[29] The Karpenter Contributers, *Karpenter*, version 0.27.5, Cloud Native Computing Foundation, May 18, 2023. [Online]. Available: https://github.com/kubernetes-sigs/karpenter (visited on 11/23/2023). Note: *Previously karpenter-core*.

[30] Cloud Native Computing Foundation. "FAQ," Cloud Native Computing Foundation. (2023), [Online]. Available: https://www.cncf.io/about/faq/ (visited on 09/18/2023).

[31] P. Wiesner, I. Benke, and O. Kao. "A Testbed for Carbon-Aware Applications and Systems." arXiv: 2306.09774 [cs, eess]. (Jun. 16, 2023), [Online]. Available: https://arxiv.org/abs/2306.09774 (visited on 08/30/2023), preprint.

[32] Aydin Mir Mohammadi, *Carbon Aware Computing*, Bluehands, 2023. [Online]. Available: https://github.com/bluehands/Carbon-Aware-Computing (visited on 08/24/2023).

[33] R. Fairbanks. "Carbon Aware Scheduling on Nomad and Kubernetes," The Green Web Foundation. (Oct. 10, 2022), [Online]. Available: https://www.thegreenwebfoundation.org/news/carbon-aware-scheduling-on-nomad-and-kubernetes/ (visited on 08/24/2023).

[34] V. Knight, J. L. Madiedo, K. Tantia, pritipath, and B. DeRusha, *Carbon Aware SDK*, version v1.1.0, Green Software Foundation, Jul. 25, 2023. [Online]. Available: https://github.com/Green-Software-Foundation/carbon-aware-sdk (visited on 05/27/2023).

[35] R. Fairbanks. "Carbon aware temporal shifting of Kubernetes workloads using KEDA." (Jun. 5, 2023), [Online]. Available: https://rossfairbanks.com/2023/06/05/carbon-aware-temporal-shifting-with-keda/ (visited on 08/23/2023).

[36] A. Radovanović *et al.*, "Carbon-Aware Computing for Datacenters," *IEEE Transactions on Power Systems*, vol. 38, no. 2, pp. 1270–1280, Mar. 2023, ISSN: 1558-0679. DOI: 10.1109/TPWRS.2022.3173250.

[37] R. Fairbanks, *Carbon-Aware Karmada Operator*, Aug. 19, 2023. [Online]. Available: https://github.com/rossf7/carbon-aware-karmada-operator (visited on 08/24/2023).

[38] A. Souza *et al.*, "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*, ser. ASPLOS 2023, New York, NY, USA: Association for Computing Machinery, Jan. 30, 2023, pp. 252–265, ISBN: 978-1-4503-9916-6. DOI: 10.1145/3575693.3575709.

[39] N. Bashir, D. Irwin, and P. Shenoy, "On the Promise and Pitfalls of Optimizing Embodied Carbon," in *Proceedings of the 2nd Workshop on Sustainable Computer Systems*, ser. HotCarbon '23, New York, NY, USA: Association for Computing Machinery, Aug. 2, 2023, pp. 1–6, ISBN: 979-8-4007-0242-6. DOI: 10.1145/3604930.3605710.

[40] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing*, ser. SoCC '12, New York, NY, USA: Association for Computing Machinery, Oct. 14, 2012, pp. 1–13, ISBN: 978-1-4503-1761-0. DOI: 10.1145/2391229.2391236.

[41] T. Kennes. "Measuring IT Carbon Footprint: What is the Current Status Actually?" arXiv: 2306.10049 [cs]. (Jun. 12, 2023), preprint.

[42] Green Software Foundation. "SCI Guide." (2023), [Online]. Available: https://sci-guide.greensoftware.foundation (visited on 08/28/2023).

[43] WattTime. "Watttime." (2023), [Online]. Available: https://www.watttime.org (visited on 09/07/2023).

[44] M. Steinke, P. Kilian, P. Wiesner, and A. Malkowski, *Vessim*, DOS Group at TU Berlin, Sep. 8, 2023. [Online]. Available: https://github.com/dos-group/vessim (visited on 09/11/2023).

[45] N. Herbst *et al.*, "Ready for Rain? A View from SPEC Research on the Future of Cloud Metrics," SPEC Research Group, Technical Report SPEC-RG-2016-01, Mar. 8, 2016. arXiv: 1604.03470 [cs]. [Online]. Available: https://research.spec.org/fileadmin/user_upload/documents/rg_cloud/endorsed_publications/SPEC-RG-2016-01_CloudMetrics.pdf (visited on 09/11/2023).

[46] M. A. Tamiru, J. Tordsson, E. Elmroth, and G. Pierre, "An Experimental Evaluation of the Kubernetes Cluster Autoscaler in the Cloud," in *2020 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Dec. 2020, pp. 17–24. DOI: 10.1109/CloudCom49646.2020.00002.

[47] J. von Kistowski, S. Eismann, N. Schmitt, A. Bauer, J. Grohmann, and S. Kounev, "TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research," in *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Sep. 2018, pp. 223–236. DOI: 10.1109/MASCOTS.2018.00030.

[48] A. V. Papadopoulos *et al.*, "Methodological Principles for Reproducible Performance Evaluation in Cloud Computing," *IEEE Transactions on Software Engineering*, vol. 47, no. 8, pp. 1528–1543, Aug. 2021, ISSN: 1939-3520. DOI: 10.1109/TSE.2019.2927908.

[49] S. N. Shirazi, A. Gouglidis, A. Farshad, and D. Hutchison, "The Extended Cloud: Review and Analysis of Mobile Edge Computing and Fog From a Security and Resilience Perspective," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2586–2595, Nov. 2017, ISSN: 1558-0008. DOI: 10.1109/JSAC.2017.2760478.

[50] A. Bauer, N. Herbst, S. Spinner, A. Ali-Eldin, and S. Kounev, "Chameleon: A Hybrid, Proactive Auto-Scaling Mechanism on a Level-Playing Field," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 800–813, Apr. 2019, ISSN: 1558-2183. DOI: 10.1109/TPDS.2018.2870389.

[51] Boavizta Working Group. "Digital & environment : How to evaluate server manufacturing footprint, beyond greenhouse gas emissions?" Boavizta. (Nov. 26, 2021), [Online]. Available: https://www.boavizta.org/en/blog/empreinte-de-la-fabrication-d-un-serveur (visited on 08/29/2023).

[52] Dell, "Product Carbon Footprint of PowerEdge R740," Jan. 2019. [Online]. Available: https://i.dell.com/sites/csdocuments/CorpComm_Docs/en/carbon-footprint-poweredge-r740.pdf (visited on 11/20/2023).

[53] S. B. Boyd, *Life-Cycle Assessment of Semiconductors*. New York, NY: Springer New York, 2012, ISBN: 978-1-4419-9988-7. DOI: 10.1007/978-1-4419-9988-7.

[54] Dell. "Product Carbon Footprints." (2021), [Online]. Available: https://www.dell.com/en-us/dt/corporate/social-impact/advancing-sustainability/climate-action/product-carbon-footprints.htm (visited on 11/20/2023).

[55] M. G. Bardon and B. Parvais. "The environmental footprint of logic CMOS technologies," imec. (Dec. 17, 2020), [Online]. Available: https://www.imec-int.com/en/articles/environmental-footprint-logic-cmos-technologies (visited on 11/20/2023).

[56] Thinkstep, "Life Cycle Assessment of Dell R740," Dell, Jun. 2019. [Online]. Available: https://www.delltechnologies.com/asset/en-us/products/servers/technical-support/Full_LCA_Dell_R740.pdf (visited on 11/20/2023). Note: *ISO 14040/14044 compliant study*.

[57] J. Gröger, R. Liu, D. L. Stobbe, J. Druschke, and N. Richter, "Green Cloud Computing: Lifecycle-based data collection on the environmental impacts of cloud computing," German, German Federal Environment Agency, Dessau-Roßlau, 94/2021, Jun. 2021. [Online]. Available: https://www.umweltbundesamt.de/sites/default/files/medien/5750/publikationen/2021-06-17_texte_94-2021_green-cloud-computing.pdf (visited on 11/20/2023).

[58] A. de Vries, U. Gallersdörfer, L. Klaaßen, and C. Stoll, "Revisiting Bitcoin's carbon footprint," *Joule*, vol. 6, no. 3, pp. 498–502, Mar. 16, 2022, ISSN: 2542-4351. DOI: 10.1016/j.joule.2022.02.005.

[59] G. A. Brady, N. Kapur, J. L. Summers, and H. M. Thompson, "A case study and critical assessment in calculating power usage effectiveness for a data centre," *Energy Conversion and Management*, vol. 76, pp. 155–161, Dec. 1, 2013, ISSN: 0196-8904. DOI: 10.1016/j.enconman.2013.07.035.

[60] Thoughtworks. "Cloud Carbon Footprint," Cloud Carbon Footprint. (2023), [Online]. Available: https://www.cloudcarbonfootprint.org (visited on 05/23/2023).

[61] L. A. Barroso, U. Hölzle, and P. Ranganathan, *The Datacenter as a Computer* (Synthesis Lectures on Computer Architecture), 3rd ed. Cham: Springer Nature, 2019, ISBN: 978-3-031-01761-2.

[Online]. Available: https://library.oapen.org/handle/20.500.12657/61844 (visited on 12/08/2023).

[62] Intel. "Thermal Design Power (TDP) in Intel Processors," Intel. (Apr. 4, 2023), [Online]. Available: https://www.intel.com/content/www/us/en/support/articles/000055611/processors.html (visited on 12/08/2023).

[63] S. Rincé, D. Ekchajzer, B. Petit, O. de Meringo, and Bruno Thomas, *BoaviztAPI*, in collab. with J. V. Andreasen, version v1.1.0, Boavizta, Nov. 19, 2023. [Online]. Available: https://github.com/Boavizta/boaviztapi (visited on 11/20/2023).

[64] B. Linder, "PBP vs TDP: Intel changes power consumption starting with 12th-gen chips," *Liliputing*, Jan. 10, 2022. [Online]. Available: https://liliputing.com/pbp-vs-tdp-intel-changes-power-consumption-starting-with-12th-gen-chips/ (visited on 12/08/2023).

[65] Intel. "Intel Core i9-12900HX Processor," Product Specifications. (2022), [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/228441/intel-core-i912900hx-processor-30m-cache-up-to-5-00-ghz.html (visited on 12/08/2023).

[66] Microsoft, *A new approach for Scope 3 emissions*, 2021. [Online]. Available: https://download.microsoft.com/download/7/2/8/72830831-5d64-4f5c-9f51-e6e38ab1dd55/Microsoft_Scope_3_Emissions.pdf (visited on 11/20/2023).

[67] B. Davy. "Building an AWS EC2 Carbon Emissions Dataset," Teads Engineering. (Sep. 23, 2021), [Online]. Available: https://medium.com/teads-engineering/building-an-aws-ec2-carbon-emissions-dataset-3f0fd76c98ac (visited on 05/22/2023).

[68] J. Barr. "Cloud Computing, Server Utilization, & the Environment," AWS News Blog. (Jun. 5, 2015), [Online]. Available: https://aws.amazon.com/blogs/aws/cloud-computing-server-utilization-the-environment/ (visited on 12/08/2023).

[69] AWS Public Sector Blog Team. "Why moving to the cloud should be part of your sustainability strategy," AWS Public Sector Blog. (Oct. 25, 2021), [Online]. Available: https://aws.amazon.com/blogs/publicsector/why-moving-cloud-part-of-sustainability-strategy/ (visited on 12/08/2023).

[70] International Energy Agency. "Average CO2 intensity of power generation from coal power plants, 2000-2020," IEA. (Jun. 29, 2020), [Online]. Available: https://www.iea.org/data-and-statistics/charts/average-co2-intensity-of-power-generation-from-coal-power-plants-2000-2020 (visited on 12/06/2023).

[71] European Environment Agency, *Greenhouse gas emission intensity of electricity generation*, Data Visualization, Oct. 25, 2023. [Online]. Available: https://www.eea.europa.eu/data-and-maps/daviz/co2-emission-intensity-14 (visited on 12/21/2023).

[72] International Energy Agency. "Carbon intensity of electricity generation in selected countries and regions, 2000-2020," IEA. (2021), [Online]. Available: https://www.iea.org/data-and-statistics/charts/carbon-intensity-of-electricity-generation-in-selected-countries-and-regions-2000-2020 (visited on 09/14/2023).

[73] International Energy Agency. "Global average carbon intensity of electricity generation in the Stated Policies, Sustainable Development and Net Zero scenarios, 2000-2040," IEA. (2021), [Online]. Available: https://www.iea.org/data-and-statistics/charts/global-average-carbon-intensity-of-electricity-generation-in-the-stated-policies-sustainable-development-and-net-zero-scenarios-2000-2040 (visited on 09/14/2023).

[74] O. Corradi. "Marginal vs average: Which one to use for real-time decisions?" Electricity Maps Blog. (Jun. 6, 2022), [Online]. Available: https://www.electricitymaps.com/blog/marginal-vs-average-real-time-decision-making (visited on 08/24/2023).

[75] 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, Oct. 2020, ISSN: 25424351. DOI: 10.1016/j.joule.2020.08.001.

[76] California Air Resources Board, "California Greenhouse Gas Emissions for 2000 to 2019," California, Jul. 28, 2021. [Online]. Available: https://ww2.arb.ca.gov/sites/default/files/classic/cc/ghg_inventory_trends_00-19.pdf (visited on 12/21/2023).

[77] D. Broekhoff, M. Gillenwater, T. Colbert-Sangree, and P. Cage, "Securing Climate Benefit: A Guide to Using Carbon Offsets," Stockholm Environment Institute & Greenhouse Gas Management Institute, Nov. 13, 2019. [Online]. Available: https://www.offsetguide.org/pdf-download (visited on 09/12/2023).

[78] Boavizta. "Understanding the results of cloud providers' carbon calculators," Boavizta. (May 29, 2023), [Online]. Available: https://www.boavizta.org/en/blog/calculettes-carbone-clouds-providers (visited on 12/12/2023).

[79] A. Bichsel and G. Skladman. "Cloud for Sustainability API calculation methodology," Microsoft Cloud for Sustainability. (Oct. 19, 2023), [Online]. Available: https://learn.microsoft.com/en-us/industry/sustainability/api-calculation-method (visited on 12/12/2023).

[80] Boavizta. "Boavizta API documentation." (Dec. 2023), [Online]. Available: https://doc.api.boavizta.org (visited on 12/01/2023).

[81] B. Petit, *Scaphandre*, version v0.3, Hubblo, May 22, 2023. [Online]. Available: https://github.com/hubblo-org/scaphandre (visited on 05/22/2023).

[82] Sustainable Computing Collaborators, *Kepler*, Sustainable Computing, May 21, 2023. [Online]. Available: https://github.com/sustainable-computing-io/kepler (visited on 05/22/2023).

[83] N. Scarlat, M. Prussi, and M. Padella, "Quantification of the carbon intensity of electricity produced and used in Europe," *Applied Energy*, vol. 305, p. 117 901, Jan. 1, 2022, ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2021.117901.

[84] Ember. "Electricity Data Explorer," Ember. (2023), [Online]. Available: https://ember-climate.org/data/data-tools/data-explorer/ (visited on 11/07/2023).

[85] ADEME. "Base Impacts." French, l'Agence de l'Environnement et de la Maîtrise de l'Energie. (2023), [Online]. Available: https://base-empreinte.ademe.fr (visited on 11/07/2023).

[86] Amazon Web Services. "Amazon EC2," AWS. (2023), [Online]. Available: https://aws.amazon.com/ec2/ (visited on 12/07/2023).

[87] Amazon Web Services. "Amazon Elastic Compute Cloud Documentation," AWS. (2023), [Online]. Available: https://docs.aws.amazon.com/ec2/ (visited on 12/07/2023).

[88] J. S. Barbara, J. G. Myers, and C. Hacman, *Kubernetes Operations*, Cloud Native Computing Foundation, Dec. 7, 2023. [Online]. Available: https://github.com/kubernetes/kops (visited on 12/07/2023).

[89] Amazon Web Services. "Global Infrastructure Regions & AZs," AWS. (2023), [Online]. Available: https://aws.amazon.com/about-aws/global-infrastructure/regions_az/ (visited on 11/07/2023).

[90] A. Leites, C. McBride, B. Soghigian, and R. Gregory, *AKS Karpenter Provider*, Microsoft Azure, Nov. 18, 2023. [Online]. Available: https://github.com/Azure/karpenter (visited on 11/23/2023).

[91] J. Innis and T. Bannister, *Karpenter v1beta1 Full Change List*, version 6ea6132, Sep. 15, 2023. [Online]. Available: https://github.com/aws/karpenter-provider-aws/blob/main/designs/v1beta1-full-changelist.md (visited on 12/30/2023).

[92] J. Innis and T. Bannister, *Karpenter v1beta1 APIs*, version 6ea6132, Sep. 15, 2023. [Online]. Available: https://github.com/aws/karpenter-provider-aws/blob/main/designs/v1beta1-api.md (visited on 12/30/2023).

[93] The Helm Authors. "Helm Documentation," Helm. (2023), [Online]. Available: https://helm.sh/docs (visited on 09/19/2023).

[94] N. Tran, *Deprovisioning - Drift*, version 18fe199, Cloud Native Computing Foundation, Jun. 21, 2023. [Online]. Available: https://github.com/kubernetes-sigs/karpenter/blob/18fe199/designs/drift.md (visited on 09/20/2023).

[95] D. Carrion and J. Innis. "Multi region support in same cluster," Karpenter Provider AWS. (Nov. 6, 2023), [Online]. Available: https://github.com/aws/karpenter-provider-aws/issues/5024 (visited on 12/23/2023).

[96] R. C. Martin, "The Dependency Inversion Principle," *C++ Report*, vol. 8, no. 6, pp. 61–66, Jun. 1996, ISSN: 1040-6042. [Online]. Available: https://web.archive.org/web/20110714224327/http://www.objectmentor.com/resources/articles/dip.pdf (visited on 12/16/2023).

[97] D. Rey and M. Neuhäuser, "Wilcoxon-signed-rank test," in *International Encyclopedia of Statistical Science*, M. Lovric, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1658–1659, ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_616.

[98] P. von Hippel, "Skewness," in *International Encyclopedia of Statistical Science*, M. Lovric, Ed., Berlin, Heidelberg: Springer, 2011, pp. 1340–1342, ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_525.

[99] P. Sprent, "Sign Test," in *International Encyclopedia of Statistical Science*, M. Lovric, Ed., Berlin, Heidelberg: Springer, 2011, pp. 1316–1317, ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_515.

[100] L. A. Barba. "Terminologies for Reproducible Research." arXiv: 1802.03311 [cs]. (Feb. 9, 2018), preprint.

[101] Amazon Web Services. "EC2 On-Demand Instance Pricing," AWS. (2023), [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/ (visited on 12/10/2023).

[102] P. Julve and L. Ballard. "How Grafana Labs switched to Karpenter to reduce costs and complexities in Amazon EKS," Grafana Labs. (Nov. 9, 2023), [Online]. Available: https://grafana.com/blog/2023/11/09/how-grafana-labs-switched-to-karpenter-to-reduce-costs-and-complexities-in-amazon-eks/ (visited on 12/11/2023).