CASPER: Carbon-Aware Scheduling and Provisioning for Distributed Web Services

Abel Souza[1] Shruti Jasoria[1] Basundhara Chakrabarty[1] Alexander Bridgwater[2] Axel Lundberg[2] Filip Skogh[2]
Ahmed Ali-Eldin[2] David Irwin[1] Prashant Shenoy[1]
[2]Chalmers University of Technology
[1]University of Massachusetts Amherst

***Abstract*—In recent years, there has been a significant commitment to reducing carbon emissions and shifting towards more sustainable practices, including in computing. Computations such as web-services exhibit various spatiotemporal and performance flexibility, enabling the possibility of adjusting the location, schedule, and processing intensity to align with the availability of renewable or low-carbon energy. An example is a geographically distributed web application hosted across multiple cloud regions, each with varying carbon costs based on their local electricity mix. By spatially load-balancing web requests, it becomes feasible to reduce applications operational carbon footprint. However, operators still face challenges in efficiently optimizing their footprint due to a lack of insights into carbon costs. In here, we present `CASPER`, a carbon-aware scheduling and provisioning system for distributed web services. `CASPER` primarily aims to minimize the carbon costs associated with resource provisioning while also respecting the applications's Service Level Objective (SLO) requirements. To this end, we formulate `CASPER` as an optimization problem and deploy a prototype in Kubernetes. Our empirical results reveal the significant potential of `CASPER` in achieving substantial reductions in carbon emissions: In comparison to baseline methods, our approach demonstrates improvements of up to 70% with no latency performance degradation.**

***Index Terms*—Carbon-Aware Computing, Web Services**

I. Introduction

In recent years, the global focus on sustainability and environmental responsibility has brought renewable energy to the forefront of the discussions on energy systems, leading to an increased focus on reducing the carbon footprint of cloud platforms in both research and industry [23], [22], [14], [13], [10]. Although there has been substantial progress in reducing usage, today's datacenter infrastructures consume around three to five percent of electricity worldwide and in ten years, five times as much [4], [18], [12]. These estimations may be lower than reality, as the growth of computing demand has been increasing exponentially for decades [6], which is triggering serious datacenter efficiency concerns because demand is outpacing supply [20], and more importantly, positioning cloud platforms as one of the largest contributors to global emissions [6]. that have solely relied on enhancements in energy efficiency, which is unlikely to lead to significant reductions in carbon emissions as modern datacenters have already achieved high levels of optimization in energy efficiency. For instance, the Power Usage Effectiveness (PUE), a measurement of the total operational efficiency of most datacenters,

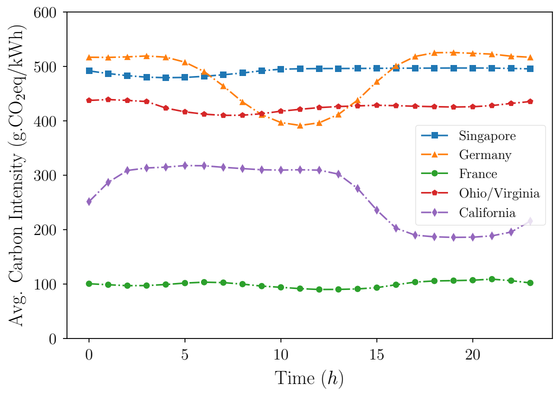


Figure 1: *Grid carbon intensity for six different cloud regions showing 6× spatial variations.*

is already near the optimal value of 1.0. Therefore, while energy efficiency improvement is important, it is insufficient to counterbalance the rising energy consumption from the rapidly growing demand for cloud services. To effectively reduce carbon emissions, cloud platforms must shift their focus towards low-carbon energy sources. This entails harnessing energy derived from renewable sources such as wind, solar, hydro, nuclear, geothermal, and other sustainable alternatives.

To reduce cloud platforms' carbon emissions, many have suggested leveraging computing workloads' spatial and temporal flexibility, which is often significant, to dynamically shift the location and time of execution to better align with when and where low-carbon energy is available. Yet, despite the prominence of such simple carbon-aware spatiotemporal workload shifting as an abstract idea, prior work has only quantified its benefits in specific settings such as batch workloads. Web applications, in particular, serve as an excellent case for exploring the untapped potential of carbon-aware computing. These applications are typically distributed across multiple cloud servers located in different regions worldwide. Traditional approaches reduce latency by forwarding user requests to the geographically closest replica server, reducing load times and offering a better user experience. On the other hand, different cloud regions have varying carbon costs associated with their electricity sources, leading to different carbon footprints for user requests depending on which replica server services them. Consequently, optimizing the scheduling of user requests with respect to the carbon costs associated with different replicas presents an intriguing opportunity to make web applications more sustainable.

While renewable energy still continues to be unreliable due to its dependence on natural factors, web applications can still benefit from it without sacrificing performance. For instance, the inherent fault tolerance achieved through replication and load-balancing mechanisms can safeguard web applications against the intermittency and unpredictability of renewable energy sources. By ensuring that replicas are spread across diverse regions and backed by different energy sources, these applications can maintain high availability while capitalizing on the potential of cleaner energy. However, although cloud providers maintain information about their energy resources, this information is not readily available at a software level. Consequently, provisioners and load balancers cannot leverage this information to optimize the carbon efficiency of web applications. By providing them with the visibility into the carbon costs associated with different datacenters, it becomes possible to design heuristics to provision servers and schedule user requests, directing them to the replicas with the lowest carbon impacts.

As such, the objective of this paper is to evaluate the performance of spatial server provisioning and geo-distributed request scheduling in a distributed web application with a focus on optimizing the carbon and latency costs of computation. We assume full information of energy data, formulating this scenario as a multi-objective optimization problem. We present `CASPER`, a carbon-aware scheduler and provisioner for distributed web applications. We implement `CASPER` as a Kubernetes scheduler, and our results highlight its significant potential in achieving considerable reductions in carbon emissions while respecting latency constrains. When compared to baseline methods, our approach showcases improvements of up to 70% without any degradation in latency performance.

II. Background

This section provides background information on the grid carbon intensity, cloud model, carbon-aware workload optimizations, and cloud schedulers.

Carbon Intensity The electric grid relies on a combination of generation sources to meet the demand for electricity. These sources include fossil fuel-based generators using coal or natural gas, renewable sources like hydro, wind, and solar, as well as non-carbon sources such as nuclear power. Since electricity demand fluctuates throughout the day and follows diurnal patterns, the mix of generation sources and their relative proportions also vary over time. It is worth noting that renewable sources — such as wind and solar — are intermittent, which further impacts the overall generation mix. The carbon intensity (CI) of electricity supply, measured in grams of CO2 equivalent per watt or g$\cdot$$CO_2$eq/kWh, represents the average weighted carbon intensity of the generation sources used at any given moment. As fossil-based sources have high, and renewable sources have low or zero carbon weights, the average CI depends on the proportion of each source in the overall generation mix.

Figure 1 illustrates the carbon intensity of the grid electricity over the 2022 period for six different geographical regions. The figure reveals significant variations in the carbon intensity across different regions. On the vertical axis, the carbon intensity exhibits temporal (error lines) variations, while the horizontal axis shows the spatial variations across regions. Specifically, the carbon intensity shows a $4\times$ variation over a year in France, and an even higher 6-$8\times$ variation in Germany and California for the same period. These variations imply that the carbon footprint of a job can be up to $8\times$ higher depending on whether it is executed during a high or low carbon-intensity period. They suggest that the same job executed in different cloud regions will result in different footprints, underscoring the need for techniques that intelligently schedule jobs on clusters based on the current and projected carbon intensity of grid electricity. While cluster managers can leverage many temporal variations by aligning jobs execution with low carbon periods, our current work primarily focuses on exploiting spatial variations by scheduling jobs on geographically distributed resources across regions with lowest carbon intensity and enough latency performance. Finally, our work concentrates on scheduling techniques aimed at reducing a datacenter's scope 2 emissions as defined by the GHG (Greenhouse Gas) protocol, in which the majority of operational emissions are attributed to grid energy (including scope 1). We do not consider embodied emissions (scope 3).

Spatiotemporal Shifting The potential for any job to reduce its footprint via temporal or spatial shifting is a function of many characteristics. Particularly, a job's overhead is a function of its state (i.e., memory and disk) and the network distance, i.e., latency and bandwidth, between locations. Additionally, there may be regulatory constraints, such as HIPPA and GDPR, that prevent spatially shifting a job outside of a specific country, region or jurisdiction.

Workload Flexibility As mentioned earlier, the variations in carbon intensity of electricity supply have a direct impact on the emissions generated when computing across different locations. While batch jobs such as AI and machine learning often have flexible completion requirements and can accommodate temporal variations, interactive workloads have strict low-latency requirements and limited temporal flexibility. This is especially true in web-services environments where requests pass through multiple microservices before a response is produced. For instance, load balancing tools enable modern workloads with the ability to shift their execution location to align with renewable energy sources and that are cost-effective. These techniques work mostly with workloads that have lightweight memory states and do not need to carry data around locations. In this study we consider lightweight web-requests – specifically HTTP requests –, that can be seamlessly processed across various locations. These requests have latency requirements that need to be limited within a

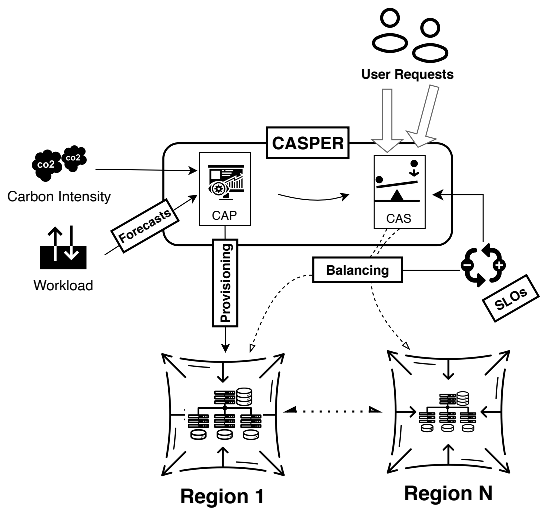


Figure 2: CASPER*: CAP and CAS provision and coordinate user workloads.*

maximum threshold. Given that numerous services are highly optimized, minor deviations within a specified target are unlikely to impact the user experience significantly, and can enable the exercise of spatial shifting to optimize for carbon. Cloud-based systems are specifically designed to optimize a combination of throughput and (scheduling) response time [3], [25], [16]. They are built to handle dynamic workloads, varying resource needs, and leverage scaling and migration techniques to dynamically acquire and release resources while reducing waiting times to improve cluster utilization.

Model The cloud model operates on the principle of providing users with an abstract view of the underlying physical resources that accommodate multiple concurrent workloads. However, when an application or job scales, migrates, or interrupts execution based on any dynamic signal such as carbon intensity, it may face challenges due to limitations at the datacenter level that arise from other jobs utilizing resources. This highlights a broader concept: optimizing carbon consumption decisions must take into account the platform's constraints, including the availability of resources. We introduce CASPER to tackle this specific planning issue.

III. System Design and Implementation

This section outlines the design and architecture of CASPER, along with the key components required for its carbon-awareness. We also introduce a load-balancer that optimizes the workload performance by redirecting requests across servers.

A. Architecture

CASPER is designed as a modular system that can be integrated into any existing cloud-enabled scheduler. Figure 2 illustrates the overall system architecture, highlighting its two main components: the Carbon-Aware Provisioner (CAP) and the Carbon-Aware Scheduler (CAS). CASPER includes various components for interfacing with interactive jobs, such as the resource manager, monitoring, and the carbon-aware load-balancing and scheduling policies.

Parameter	Description
x_{ij}	Requests redirected from region i to region j
$\bar{x_j}$	Requests not sent to region j
s_j	Number of servers in region j
n	Number of regions $\mathcal{R}$
I_j	Carbon intensity in region j
α	Normalized weight for the carbon intensity (in relation to number of servers s_j)
λ_i	Incoming request rate at region i
ℓ_{ij}	Expected latency from region i to j
c_j	Resource capacity of region j (in # of requests)
L_i	Maximum tolerated latency for a request
K	Maximum number of servers across all locations
t_j	Number of requests submitted to region j

Table I: *List of parameters used by CAP.*

Carbon-Aware Provisioner. CAP acts as an intelligent provisioner that analyzes the inter-regional network latency, the region's carbon intensity, and the expected workload. Besides reducing carbon, CAP needs to provide operators with an important estimator: the optimal number of servers needed in each region such that the expected workload is correctly handled for each time period. This intuition leads us to formalize the datacenter provisioning as a multi-objective problem.

$$\min_{x} \quad \alpha \sum_j I_j \sum_i x_{ij} + (1-\alpha) \sum_j s_j \tag{1a}$$

$$\text{s.t.} \quad \sum_i x_{ij} \leq s_j c_j \tag{1b}$$

$$\sum_j s_j \leq K \tag{1c}$$

$$x_{ij}\left(\ell_{ij} - L\right) \leq 0 \tag{1d}$$

$$\sum_{i,j} x_{ij} = \mathbb{E}[\lambda_i], \forall i, j \tag{1e}$$

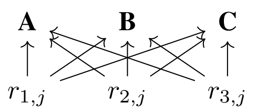
$$\alpha \in [0,1] \tag{1f}$$

$$\bar{x}_j s_j = 0 \tag{1g}$$

$$x_{ij}, s_j \in \mathbb{Z}_{\geq 0} \tag{1h}$$

We present CAP's formulation in Equation 1, and Table I describes all of its parameters. All indices i, j represent the set of available regions $\mathcal{R}$ for resource allocation, request processing and redirection. We let $\bar{x_j} \in \{0, 1\}$ be the variable that represents requests that are not sent to a region j, i.e., $\sum_j x_{ij} = 0$. This constraint effectively means that if it is anticipated that region j won't receive any requests, there should be no server allocation in that region. Moreover, x_{ij} represents the optimal count of requests from region i that is redirected to region j, while s_j is the number of servers provisioned in region j to handle incoming requests. Eq. 1(a) aims to minimize both the total carbon cost of executing requests (Eq. 1(b)) and the cumulative number of servers s_j across all regions (Eq. 1c) such that the minimum latency target is guaranteed (Eq. 1d). Finally, the following assumptions are made. First, the scope of the problem is limited to minimizing carbon emissions while respecting applications latency constraints. Second, since we consider cloud datacenters, we ignore issues regarding capacity planning or resource limits, although we do include a maximum threshold in the total number of servers (Eq. 1c) used by CASPER. We also assume the communication across load-balancers in different regions (as seen in Figure 2) is negligible when compared to requests' average service times. Finally, the provisioner uses forecasts for carbon intensity and hourly workload request rates that are expressed in terms of expected arrivals in region i (Eq. 1e).

B. Carbon Aware Scheduler



(a) CAS Balancing

$$r_{i,j} = \frac{\sum_i x_{ij}}{\sum_i \sum_j x_{ij}} t_j$$

(b) Weight Factors

Figure 3: *Illustration of CAS and weight calculation.*

Figure 3 shows how the Carbon Aware Scheduler (CAS) distributes requests between regions x_{ij}, which denotes the load of requests that need to be redirected from region i to j. It uses a vector as shown by Equation 3(b) to model each region r's weight, following the timely estimates obtained from CAP (e.g., hourly). We implement CAS as a load-balancer module in CASPER, whereas local incoming requests are redistributed across all regions according to their proportional weights. More importantly, CAS ensures that unforeseen workload events — not accounted for in the CAP's optimization — can be effectively handled.

C. Implementation

CASPER is designed as a Kubernetes (K8S) cluster with $s < K$ workers, where each represents a cloud region. Each deployment comprises of a single K8S pod and each runs the application. A prototype has been developed to emulate the operations of a Wikipedia-like service across six distinct AWS regions, as detailed in the Table II. These regions were selected as the closest regions to the original Wikimedia servers.

CAP has been developed using Python and the optimization is solved using the PuLP library [19], an interface to the Coin-or branch and cut (CBC) solver [21]. The region wise server deployment array obtained as an output of CAP is used to scale the size of each regions using server collected metrics. Additionally, CAP computes the optimal request distribution matrix, which is forwarded to CAS.

CAS utilizes a set of load balancers, one per-region. As mentioned, it redirects incoming requests to the appropriate regions based on weights derived from the CAP's optimal request distribution matrix. Traefik [27] is used to establish the load balancer layer for the cluster setup. It creates a HTTP proxy for every region to receive requests. Each traefik proxies can dynamically route the traffic to one of the corresponding Kiwix backend regions based on the hourly weights calculated by CAP.

Geographical Region	AWS Region
California	US-West-1
Virginia	US-East-1
Ohio	US-East-2
Germany	EU-Central-1
France	EU-West-3
Singapore	AP-SouthEast-2

Table II: *AWS Regions used in the evaluation.*

IV. Evaluation

In this section, we first discuss the real-world application, workload, carbon, and network traces that are utilized to evaluate CASPER. Then, we discuss the policies briefly introduced in the previous section. Finally, we demonstrate and quantify the trade-offs between carbon savings and latency performance for various targets.

Infrastructure. The cluster hosting CASPER runs Ubuntu Linux 20.04, and is compromised of 16 Dell PowerEdge R430s with a two-socket Intel Xeon processor with 8 cores and 32GB of memory. The cluster consists of a control plane and worker nodes. Each of the nodes run a Kubernetes deployment that represents a region. For intra-cluster communication, an overlay network is created using Flannel [5].

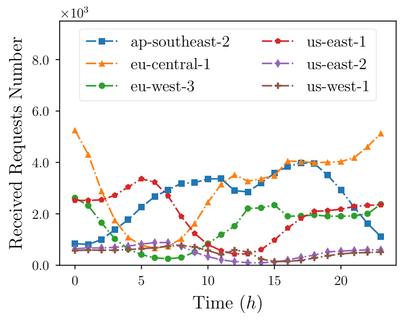
Application. To evaluate CASPER, we deploy Kiwix [8], a platform to host and distribute compressed versions of the Wikipedia. Specifically, we use the pre-built version of the German Wikipedia from May 2023, which comprises a total of 32 GB of content, hosted as a Zim file [9]. Requests are sent through the CAS load-balancer, which exposes an external HTTP port interconnecting all nodes in the cluster. Since we are only interested in the intra-network latency and emissions across regions, only the Kiwix frontend page is accessed.

Telemetry. Each region's load-balancer exports their service-level metrics, specifically the total count of HTTP requests served by each endpoint and their associated service time. To calculate the carbon cost of request execution, this metric is multiplied by the region's current hour's carbon intensity.

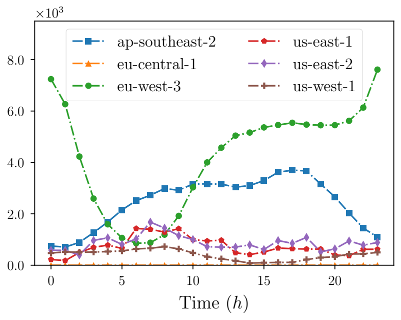
A. Datasets

Carbon Intensity. The carbon intensity data for all the aforementioned geographical regions (Table II) at an hourly granularity, and is represented by Figure 1. This data has been collected from ElectricityMaps [7] for the year 2022.

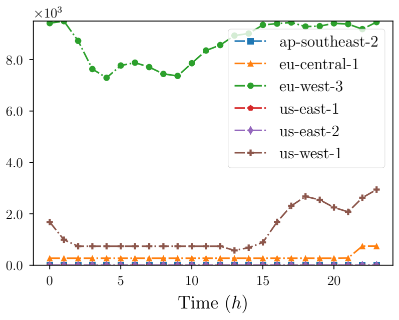
Workload and Network Traces. We use the Wikimedia's dataset [11] covering six datacenters across the USA, Europe, and Asia. For each region, the dataset includes the request rates (requests per-second) and datacenter hourly utilization covering 2022. Since two of the AWS regions do not match those from Wikimedia's – i.e., Netherlandas and Texas –, we select the two closest AWS regions i.e., Germany and Ohio (Table II). Average latency data (in milliseconds) across all



(a) Latency



(b) Carbon-20



(c) Carbon-500

Figure 4: *Request Redirection Rate per-region: Loads tend to be forwarded to greener nearby regions.*

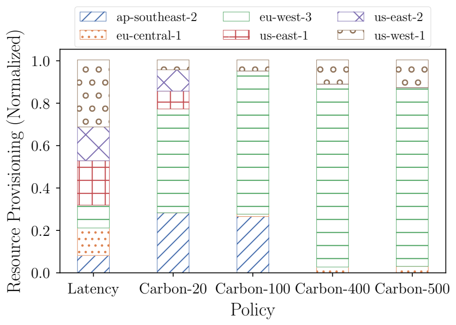


Figure 5: *Average Resource Provisioning per-region*

AWS regions are obtained from Cloudping [1] for the 2022 year.

Workload Generation A sample of the workload is represented in Figure 4(a), with incoming requests in all regions. Each hour is divided into timesteps, and the request rate for each timestep is selected from a set of values that follow an exponential distribution. Parameters to generate the distributions are selected such that the upper limit of the generated values is approximately $1.5\times$ the request rate for the hour.

B. Experimental Setup

We conduct a comprehensive evaluation of CASPER throughout the entire year of 2022. The parameters for CAP are set as follows: $n = 6$ (representing the AWS regions), $\alpha = 0.5$ (equal weights to both carbon and latency costs), $c_j = 100$ (one server can handle up to 1k requests), and $K = 500$ (global maximum number of servers). We also introduce several variations in the values of L_i (see below), which establish the maximum acceptable latency for each request. CAP runs at the beginning of every hour to determine the provisioning of servers at each location. The CAS weights are then calculated based on the output of CAP. To assess the system's performance, we execute a real workload simulation spanning 24 hours. Metrics are collected at 10-minute intervals and aggregated at the end of each hour. We conduct evaluations using the following policies:

Latency. This simulation serves as the baseline scenario without any carbon optimization, where requests are solely served based on the lowest latency, i.e., locally in the originating region, without any load balancing. This configuration strictly aligns with Mediawiki's operational approach, as it adheres to their stringent latency performance requirements.

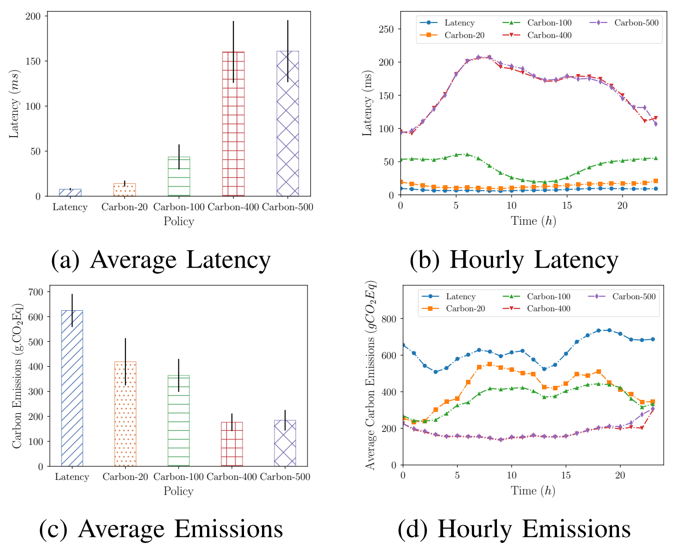
Carbon-L Policies. These runs focus on carbon optimization with various latency L threshold guarantees, ranging from 20 to 500ms. This approach involves a trade-off in terms of performance, as requests can be redirected as long as the latency requirements remain below L ms. These thresholds represent intermediate cases where the carbon cost of execution is reduced by permitting increases in latency as requests reach more distant, lower carbon regions. Among the tested scenarios, the one with a latency threshold of $L = 500$ ms represents the highest value among all possible inter-region latencies. In this particular case, the carbon cost of execution is minimized by allowing unrestricted redirections, as long as regions can accommodate the amount of requests.

C. Results

*1) **Effects on Request Redirection**:* Figures 4(a)-(c) present workload redirection results for *Latency*, *Carbon-20*, and *Carbon-500*. As shown in Figure 1, Zone "eu-west-3" (France) has the lowest carbon intensity, followed by "us-west-1" (California), "us-east-[1,2]" (Ohio/Virginia), "eu-central-1" (Germany), and "ap-southeast-2" (Singapore). Figure 4(a) simply shows the original workload, where no redirection happens. Notably in Figure 4(b), due to close proximity and low carbon intensity, CASPER redirects as many requests as possible from Germany towards France. And due to the latency constrains (20ms), Ohio and Virginia cannot induce savings. This behavior is more evident in 4(c): As the latency constraint is relaxed (500ms), France and California receive as many requests as possible from all regions. However, eu-west-3 reaches capacity at various moments, triggering CASPER to forward load to California (us-west-1).

*2) **Effects on Resource Provisioning**:* Figure 5 illustrates the resource provisioning across the six AWS regions. The *Latency* policy represents the original provisioning with no redirections. As the latency constraints increase, the CASPER initiates re-provisioning of servers from the German ("eu-central-1") region to France due to its lower carbon and lower network latency. Under *Carbon-100*, a significant portion of the Ohio and Virginia workloads are redirected exclusively to France, as the 100ms latency requirement can be fulfilled. It is worth noting that requests originating from Singapore are only directed to greener locations under the *Carbon-400* policy. This limitation arises from the end-to-end latency from Singapore to any other region surpassing the 100ms threshold. Moreover, the capacity in France reaches its limit under the *Carbon-400* policy, prompting redirections towards California, in addition to few towards Germany to meet the latency requirements.

3) ***Effects on Carbon and Latency****:* Figures 6(a)-(d) present a comparative analysis of all policies. Figures 6(a) and (c) clearly illustrate the primary tradeoff of CASPER, wherein *the relaxation of latency constraints leads to an increased potential for emissions reduction*. The Latency policy, despite achieving an average response time as low as 6ms, exhibits the highest carbon emissions due to requests remaining localized in high-intensity regions such as Germany and Singapore. Notably, *Carbon-20* demonstrates that even small relaxations in latency constraints can result in a 25% carbon reductions. *Carbon-100* achieves a 37% reduction, while *Carbon-400* reaches a point of diminishing returns with a 70% reduction, similar to *Carbon-500* which represents an unrestricted carbon optimization scenario. Moreover, Figures 6(b) and (d) display the hourly variations in average latency and emissions, respectively. In comparison to the Latency policy, *Carbon-20* shows a minimal increase in latency of 6ms while simultaneously reducing emissions. *Carbon-100* through *Carbon-500* exhibit latency increases ranging from 5-16×, although delivering the most substantial reductions. Finally, it is important to note that results would change with other α values. This is primarily due to the fact that CASPER would redirect requests differently due to the trade-off between carbon emissions and the number of servers needed to satisfy latency SLOs. Specifically, as α increases, CASPER would redirect more requests to greener regions at the cost of latency because this would reduce the number of servers in browner regions. In contrast, as α decreases, CASPER would prioritize latency, opting to handle requests locally despite the carbon costs of setting additional servers.



(a) Average Latency (b) Hourly Latency

(c) Average Emissions (d) Hourly Emissions

Figure 6: *Performance and carbon tradeoffs across policies.*

V. Related Work

Carbon-aware computing encompasses several key research, some with similar tradeoffs as presented in here [24]. Treehouse [2] proposes a software-centric approach to reduce the carbon intensity of datacenter computing by making energy and carbon visible at the application layer. CADRE focuses on carbon-aware data replication to reduce overall carbon footprint, while leveraging load flexibility and interactions with the electricity market to minimize carbon emissions [29]. [28] investigates the potential of shifting computational workloads to less carbon-intensive periods based on the fluctuating carbon intensity of energy supply. [17] introduces a low-carbon extension to the Kubernetes scheduler, sorting cloud regions by carbon intensity and migrating workloads to regions with low carbon cost. However, the proposed framework is evaluated primarily for batch jobs. On the other hand, numerous works have employed integer programming techniques to devise new techniques for low-carbon scheduling [26]. Carbon-aware geo-distributed scheduling is particularly relevant for Machine Learning (ML) workloads requiring long periods of execution [15]. [30] proposes Cucumber, an admission control policy that leverages load and energy forecasting techniques to determine scheduling strategies to use renewables. Unlike the previous works, CASPER is the first framework that seamlessly integrates server provisioning and request scheduling for a geo-distributed web application, with a particular focus on interactive web requests.

VI. Conclusion and Future Work

This paper introduced CASPER, a carbon-aware scheduler and provisioner designed for distributed web applications. We implement CASPER as a Kubernetes scheduler, and our results highlight its significant potential in achieving considerable reductions in carbon footprint for web applications. We observe substantial savings in the carbon footprint for said applications running in geo-distributed settings, reaching up to 70% and negligible loss in performance. CASPER represents a crucial advancement in carbon-aware schedulers for distributed applications. However, further analysis and exploration of additional strategies are warranted to enhance the system's efficiency. As a potential avenue for future work, the implementation of auto-scaler policies that continuously monitors resource usage across regions and that dynamically adjusts allocations could be explored. The analysis of such auto-scaling algorithms can further improve the overall carbon-saving capabilities presented with CASPER.

VII. References

[1] Matt Adorjan. Cloudping, 2017. www.cloudping.co.

[2] Thomas Anderson, Adam Belay, Mosharaf Chowdhury, Asaf Cidon, and Irene Zhang. Treehouse: A case for carbon-aware datacenter software. *arXiv preprint arXiv:2201.02120*, 2022.

[3] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, Omega, and Kubernetes: Lessons Learned from Three Container-Management Systems Over a Decade. *ACM Queue - Containers*, 14(1), January-February 2016.

[4] Gary Cook and David Pomerantz. Clicking clean: A guide to building the green internet. *Greenpeace International, Tech. Rep.*, 2015.

[5] CoreOS. Flannel, 2016. https://github.com/flannel-io/flannel.

[6] Peter J. Denning and Ted G. Lewis. Exponential Laws of Computing Growth. *Communications of the ACM*, 60(1):54–65, January 2017.

[7] ElectricityMap. Electricitymap. https://electricitymaps.com/, 2022.

[8] Emmanuel Engelhart and Renaud Gaudin. Kiwix, 2007. https://kiwix.org.

[9] Emmanuel Engelhart, Tommi Makitalo, and Manuel Schneider. openzim, 2016. https://openzim.org.

[10] Carole-Jean Wu et al. Sustainable AI: Environmental Implications, Challenges and Opportunities. In *MLSys*, August 2022.

[11] Wikimedia Foundation. Wikimedia's grafana installation, 2023. https://grafana.wikimedia.org/.

[12] C. Garcia. AKCP, the real amount of energy a data center uses. https://www.akcp.com/blog/the-real-amount-of-energy-a-data-center-use/, February 2022.

[13] Udit Gupta, Mariam Elgamal, Gage Hills, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. ACT: Designing Sustainable Computer Systems with an Architectural Carbon Modeling Tool. In *ISCA*, June 2022.

[14] Udit Gupta, Young Geun Kim, Sylvia Lee, Jordan Tse, Hsien-Hsin S. Lee, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu. Chasing Carbon: The Elusive Environmental Footprint of Computing. In *HPCA*. ACM, February 2021.

[15] Kawsar Haghshenas, Brian Setz, and Marco Aiello. Co2 emission aware scheduling for deep neural network training workloads. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 1542–1549, 2022.

[16] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A Platform for Fine-grained Resource Sharing in the Data Center. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, March 2011.

[17] Aled James and Daniel Schien. A low carbon kubernetes scheduler. In *6th International Conference on ICT for Sustainability, ICT4S 2019*, volume 2382 of *CEUR Workshop Proceedings*. CEUR-WS, June 2019. 6th International Conference on ICT for Sustainability, ICT4S 2019 ; Conference date: 10-06-2019 Through 14-06-2019.

[18] Nicola Jones. How to stop data centres from gobbling up the world's electricity. *Nature*, 561(7722):163–167, 2018.

[19] Stuart Mitchell, Michael OSullivan, and Iain Dunning. Pulp: a linear programming toolkit for python. *The University of Auckland, Auckland, New Zealand*, 65, 2011.

[20] Sparsh Mittal. A survey of techniques for approximate computing. *ACM Computing Surveys (CSUR)*, 48(4):1–33, 2016.

[21] Matthew J Saltzman. Coin-or: an open-source library for optimization. *Programming languages and systems in computational economics and finance*, pages 3–32, 2002.

[22] Abel Souza, Noman Bashir, Jorge Murillo, Walid Hanafy, Qianlin Liang, David Irwin, and Prashant Shenoy. Ecovisor: A Virtual Energy System for Carbon-Efficient Applications. In *ASPLOS*, pages 252–265, New York, NY, USA, March 2023. ACM.

[23] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and Policy Considerations for Modern Deep Learning Research. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 13693–13696, New York, NY, USA, February 2020. ACM.

[24] Thanathorn Sukprasert, Abel Souza, Noman Bashir, David Irwin, and Prashant Shenoy. Quantifying the benefits of carbon-aware temporal and spatial workload shifting in the cloud. *arXiv preprint arXiv:2306.06502*, 2023.

[25] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: The Next Generation. In *European Conference on Computer Systems (EuroSys)*, pages 1–14, New York, NY, USA, 2020. ACM.

[26] Samuel Trevino-Martinez, Rapinder Sawhney, and Oleg Shylo. Energy-carbon footprint optimization in sequence-dependent production scheduling. *Applied Energy*, 315:118949, 2022.

[27] Emile Vauge. Traefik, 2016. https://traefik.io/traefik/.

[28] Philipp Wiesner, Ilja Behnke, Dominik Scheinert, Kordian Gontarska, and Lauritz Thamsen. Let's wait awhile:

How temporal workload shifting can reduce carbon emissions in the cloud. In *Proceedings of the 22nd International Middleware Conference*, pages 260–272, 2021.

[29] Zichen Xu, Nan Deng, Christopher Stewart, and Xiaorui Wang. Cadre: Carbon-aware data replication for geo-diverse services. In *2015 IEEE International Conference on Autonomic Computing*, pages 177–186. IEEE, 2015.

[30] Siyue Zhang, Minrui Xu, Wei Yang Bryan Lim, and Dusit Niyato. Sustainable aigc workload scheduling of geo-distributed data centers: A multi-agent reinforcement learning approach, 2023.