Towards Federated Serverless Computing: An Investigation on Global Workload Distribution to Mitigate Carbon Intensity, Network Latency, and Cost

Robert Cordingly, Jasleen Kaur, Divyansh Dwivedi, Wes Lloyd
School of Engineering and Technology
University of Washington
Tacoma, Washington USA
rcording, jaslkaur, ddwivedi, wlloyd@uw.edu

***Abstract*—The high demand for energy consumption and the resulting carbon footprint of the cloud pose significant sustainability challenges, as cloud data centers consume vast amounts of energy. The emergence of serverless cloud computing platforms has opened up new avenues for more sustainable cloud computing. Serverless Function-as-a-Service (FaaS) cloud computing platforms facilitate the deployment of applications as decoupled microservices to leverage automatic rapid scaling, high availability, fault tolerance, and on-demand pricing. The absence of always-on hosting costs associated with virtual machines enables serverless functions to be deployed with many different function configurations and cloud regions to achieve high performance, low network latency, and reduced costs. In this paper, we investigate the utility of global federations of serverless platforms aggregating resources from up to 19 distinct cloud regions. We prototype a serverless load distribution system to distribute client requests across serverless federations to minimize performance objectives including: network latency, runtime, hosting costs, and carbon footprint. To evaluate our serverless distribution system's ability to meet performance objectives, we executed large experiments across 19 regions around the world continuously from November 2022 thru March 2023. Our serverless load distribution approach using federated resources was able to reduce the carbon intensity of a global federation by up to 99.8%, network latency by 65%, or hosting costs by 58%.**

***Index Terms*—Cloud Federation, Serverless Computing, Function-as-a-Service, Green Computing**

I. Introduction

The cloud computing paradigm has enabled access to nearly unlimited computational resources to anyone. While the cloud has enabled many new technologies and services that are broadly used across the world, the underlying infrastructure has massive environmental impacts. The energy consumption of a single cloud data center can be up to two gigawatt hours, the equivalent electricity of over 50,000 homes [1]. The energy consumption of cloud data centers around the world is expected to rise from 200 terawatt hours (TWh) in 2016, to close to 3,000 TWh by 2030 [2]. To put that into perspective, 3,000 TWh is over 10% of the global electricity consumption in 2021 (25,343 TWh) [3].

The emergence of the serverless computing paradigm has provided developers with a plethora of appealing features for deploying applications to the cloud. Serverless platforms not only abstract away the management of underlying infrastructure but also add desirable features such as high availability, fault tolerance, and automatic application scaling. Despite the fact that most aspects of the underlying infrastructure are managed by the cloud provider, developers are still required to define configuration parameters and deployment location(s) for their application. Serverless platforms, such as Function-as-a-Service (FaaS), favor the deployment of applications as many decoupled microservices to leverage automatic scaling and on-demand pricing. In contrast to traditional Infrastructure-as-a-Service applications where hosting applications incurs the always-on costs of virtual machine(s), FaaS platforms present no upfront or always-on costs for deploying an application and ensuring high availability because FaaS platforms only incur costs when serverless functions are run.

Existing serverless platforms, however, limit deployment and management of software to a single cloud region or cluster. Harnessing resources from multiple regions, cloud providers, or private clusters requires deployments to be managed separately by the user. Transparent aggregation of serverless resources from multiple platforms or regions has potential to deliver new serverless resource abstractions to users. We refer to the combination of resources from multiple serverless regions or platforms to transparently host serverless workloads as "Federated Serverless Computing". Federated serverless computing has potential to leverage the best resources at any given time and place to satisfy a range of goals such as minimizing carbon intensity, network latency, and runtime.

In this paper, we prototype the federation of serverless computing resources to investigate their utility. We develop and test a prototype serverless load distribution system, which is capable of distributing requests across various serverless federations. We investigate implications of serverless federations to enhance multiple performance level objectives including runtime, network latency, and environmental goals. By leveraging serverless federations, the overall carbon intensity of hosting an application can be reduced by distributing requests to locations with a higher proportion of low carbon energy sources.

Our findings indicate that serverless load distribution across a federation is capable of satisfying a diverse range of objectives depending on its configuration. For instance, in the case of a globally deployed application with clients distributed worldwide, we were able to reduce overall network latency by 65%. With carbon-aware load distribution, we were able to reduce the fossil fuel usage of an application by up to 99%.

Finally, by utilizing multiple configurations of a function and employing a model to predict optimal memory settings, we were able to reduce the overall cost of a deployment by 58%, from $833 to $349.

A. Research Questions

To evaluate our federated serverless computing prototype and new load distribution system, this paper investigates the following research questions:

(RQ-1 Performance Variation): How does function network latency and runtime of a serverless application vary over time by region?

(RQ-2 Carbon Intensity): How is the carbon intensity of a serverless application impacted by different cloud federations (e.g. deployment to America/Europe/Asia/Global)? How does the carbon intensity of cloud regions change over time?

(RQ-3 Sustainability Costs): What are the latency and performance implications of minimizing the carbon footprint of a serverless application through carbon-aware load distribution?

(RQ-4 Multi-configuration Federation): How can serverless federations be leveraged to reduce application hosting costs by utilizing function deployments with many different configurations?

B. Paper Contributions

This paper makes the following research contributions:

1) We created prototype federation tools to enable serverless cloud federations using the FaaSET framework [4].
2) We collect and present carbon intensity and network latency data for serverless platforms spanning 19 regions around the world from November 2022 to March 2023.
3) Using a suite of 12 functions, we investigated serverless load distribution across regional and global cloud federations. We evaluate five different load distribution techniques and report trade-offs between reducing carbon intensity and increasing network latency.
4) We demonstrate the ability to leverage serverless cloud federations to combine resources from different function deployments with distinct configurations to improve the performance and cost of serverless applications.

II. Background

A. Green Serverless Computing

Cloud providers such as Amazon Web Services (AWS) and Google Cloud have stated renewable energy goals to achieve net-zero carbon by 2040 and 2030 respective through various renewable energy commitments [5], [6]. Bashir et al. discuss the growing energy demand and carbon emissions of cloud platforms and their impact on environmental sustainability. Their work advocates for a "carbon first" approach to cloud design that elevates carbon efficiency to a first-class metric by virtualizing the energy system to expose visibility and control directly to applications [7]. Farahani et al. discussed a high-performance, scalable, and sustainable platform for processing massive graphs. One of the tools described by the project, Graph-Greenifier, collects, studies, and archives performance and sustainability data from operational data centers and national energy suppliers [8].

B. Serverless Federation

Smith et al. created FaDO (FaaS Functions and Data Orchestrator), a tool designed to allow data-aware functions scheduling across multiple serverless compute clusters present at different locations, such as at the edge and in the cloud [9]. FaDO further provides users with an abstraction of the serverless compute cluster's storage, allowing users to interact with data across different storage services through a unified interface. Chasins, Stoica, and Shenker present the concept of Sky computing [10], [11]. Sky computing is the idea of abstracting resources from multiple clouds through a common interface so that resources can be leveraged as a federation. The authors suggest that the barriers to achieving sky computing are more economic than technical, and propose reciprocal peering, where cloud providers create agreements to exchange services with each other, as a key enabling step. Mao expanded on the Sky Computing concept and developed SkyBridge, a storage system compatibility layer to manage data across many cloud storage backends [12].

Baarzi et al. define the idea of Virtual Serverless Providers (VSPs) that aggregate serverless offerings from multiple cloud providers [13]. The VSP system architecture adds an additional controller that invocations are passed through. The system demonstrated up to 4.2x improved throughput, reduced SLO violations by 98.8%, and reduced costs by 54%. Sampé et al. discuss a novel toolkit to enable transparent execution of Python code against disaggregated cloud resources called Lithops [14]. Lithops provides the same API as Python's standard multiprocessing library to enable any program to run on major serverless computing platforms. Jindal et al. developed a scheduling system called Courier that utilizes multiple round robin distribution techniques to route function requests between on-premises OpenWhisk, AWS Lambda, and Google Cloud Functions [15]. They show that Courier can improve the overall performance of the invocation of functions within a heterogeneous FaaS deployment compared to traditional load balancing algorithms. As a limitation we note that their FaaS deployments were relatively small with functions only deployed to three regions.

While several tools have been developed to automate deployment and federate serverless resources, there has been limited research on the potential to improve performance, reduce network latency, and minimize the environmental impact of serverless workloads through serverless federation. Our study expands the scope of previous work by investigating regional and large scale federations aggregating up to 19 cloud regions, encompassing the majority of AWS regions. Our investigation utilized twelve workload function resulting in 228 deployments. We investigated the use of five load distribution techniques for our serverless load distribution system and examined multi-configuration federation with functions deployed with five memory setting options.

III. Methodology

This section details tools and methods used to investigate our research questions defined in Section I-A. We discuss the tools used to enable our research on serverless federations in Section III-A, the architecture for our load distribution system in Section III-B, and finally how we designed our experiments in Section III-C.

A. Supporting Tools and Workloads

The Function-as-a-Service Experiment Toolkit (FaaSET) provides a unified workspace for developing, testing, profiling, and deploying serverless functions to AWS Lambda, Google Cloud Functions, IBM Cloud Functions, Azure Cloud Functions, and OpenFaaS deployments [4]. FaaSET abstracts platform specific deployment APIs and packaging requirements enabling developers to write functions once and then deploy them with multiple configurations to each platform. With FaaSET, you can define function configurations and group them into federations. When function code or configuration parameters are changed, FaaSET supports updating all functions in the federation as a deployment tool. Within FaaSET, the FaaS Runner tool supports automation of experiments and processing results [16]. The Serverless Application Analytics Framework (SAAF) performs server-side profiling to collect runtime metrics of function instances on FaaS platforms [17].

In addition to these tools, we used 12 functions as experimental workloads as shown in Table I. Each function was deployed to every region on AWS Lambda with available carbon data (19 regions). The Minimum Spanning Tree (MST), Breadth First Search (BFS), Page Rank, Compress, Resize, and DNA functions are from the Serverless Benchmarking Suite (SeBS) [18]. Stress is a common Linux tool [19]. We developed the remaining functions to support this research [20]–[22].

To collect environmental data about the electricity grid for each AWS region, we utilized the Electricity Maps API [23]. Electricity Maps is a leading resource for up-to-date electricity and CO_2 emissions data and is utilized by major corporations such as Google, Microsoft, and Cisco. We can not know the specific sources of energy for an AWS region because a region may be supplemented with additional renewable energy sources. Information about proprietary energy sources are not disclosed by the cloud provider. The Electricity Maps API provides a publicly available estimate for specific energy sources and consumption at any given time. We utilized this API for collecting current electricity carbon emission metrics for our carbon-aware load distribution system.

B. Serverless Load Distribution System

Serverless FaaS platforms provide developers near instantaneous elasticity to match the changing demand of a workload. Serverless platforms however, limit the deployment of individual functions to a single cloud region. The platform itself handles load balancing of requests across the compute resources provided within a single region. To expand beyond resources in a single region, client applications are responsible for load distribution to the appropriate function deployments.

TABLE I
Function Names and Descriptions - *SeBS

Function	vCPUs	Description
MST*	1	Generates a graph and calculates the min spanning tree.
BFS*	1	Generates a graph and processes a breadth first search.
Page Rank*	1.2	Generates a graph and processes page rank of each node.
DNA*	0.9	Pulls DNA sequence from S3 and creates visualization data.
Compress*	1	Generates files and compresses them into a zip file.
Resize*	1	Pulls an image from S3, resizes it and saves it back to S3.
Stress	n	Tool used to generate CPU stress.
Writer	1	Generates text and repeatedly writes it to disk and deletes.
CSV Processor	1	Generates a large CSV file and performs calculates on columns.
Calcs	n	Executes random math operations.
Matrix Calcs	n	Generates random large matrices and performs matrix operations.
HTTP Request	1	Makes a HTTP request with a defined payload to a URL.

We created a prototype serverless load distribution system to distribute requests to federations of serverless FaaS functions. This system is implemented using the same serverless platforms (i.e. AWS Lambda regions) as the federated resources, providing high scalability and elasticity to support proxying user requests across our federations. Our system consists of two primary serverless functions: the *Analyzer Function* and the *Proxy Function*.

1) Analyzer Function: The Analyzer function collects metrics that are fed to each proxy function deployment. In this study, the Analyzer collected carbon intensity data for each AWS region around the world that our proxy functions were deployed to. Carbon data was obtained using Electricity Maps.

The Electricity Maps API takes latitude and longitude coordinates and returns information about the electricity grid of that region, including the carbon intensity measurements in grams carbon dioxide equivalent per kilowatt hour of energy used (gCO_2eq/kWh) [23]. This measure quantifies the greenhouse gas emission intensity of electricity generation, calculated as the ratio of CO_2 emissions from electricity production. Alongside gCO_2eq/kWh, the API also returns the percentage of the energy in that region that was derived from fossil fuel sources. We use this percentage to estimate the energy impact of our serverless applications. Given that serverless platforms obfuscate information regarding underlying server infrastructure - including power consumption, we are unable to accurately determine electricity consumption directly (e.g. watt-hours for function invocations). If we could, the amount of carbon released in grams could be estimated. Instead we specify carbon intensity in fossil fuel gigabyte seconds (FF-GBS). FF-GBS is calculated by multiplying the runtime of a function invocation, the memory setting in GB, and the fossil

fuel percentage of the region running it:

$$FFGBS = Runtime_{sec} * Memory_{GB} * FossilFuel_{\%}$$

2) Proxy Function: To minimize the runtime of the serverless proxy functions, the Analyzer pushes information to the proxy functions by updating function environment variables. Editing environment variables is very fast as it does not require redeploying the entire function package and does not require the function to make a request to an external service or storage, but does add an additional cold-start. The Electricity Maps API updates carbon information on an hourly basis enabling the Analyzer function to be run periodically to check for new carbon data and update function environments. We configured a CloudWatch Events rule to trigger the Analyzer function to run every 15 minutes. Environment variables provide the proxy function immediate access to the carbon data of each region. To enable accounting, the proxy functions report carbon data in the response enabling FF-GBS of the workload to be calculated.

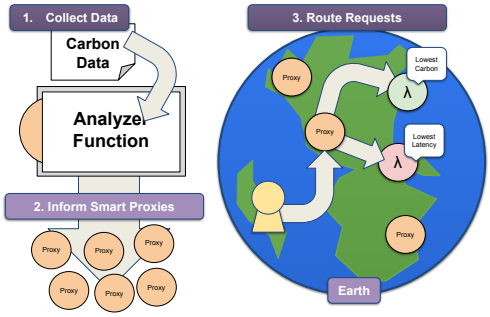

Fig. 1. Load Distribution Architecture for Analyzer and Proxy Functions

To minimize additional costs, the proxy functions used the minimum memory setting of the serverless platform (e.g. 128 MB for AWS Lambda). The proxy function performs minimal computations, and is designed to make forwarding the request as fast as possible. When deploying computationally intense serverless functions, a best practice is to choose a high memory setting for an ideal price to performance ratio. This is not the case for our proxy functions as the lowest memory setting offers the lowest cost and equal performance to higher memory settings. Using serverless proxy functions to federate resources leverages the pay-as-you-go model of serverless computing to mitigate always-on VM hosting costs while also offering high elasticity and high availability.

For the initial version of our load distribution system, our proxy functions used synchronous forwarding. With synchronous forwarding the proxy provides load distribution but features overhead known as "double billing" because the proxy function must wait for the called function to complete [24], [25]. Our approach where proxy functions runs using the lowest memory setting (e.g. 128MB), however, is far less than "double" billing. The cost overhead varies depending on the function being called. For example, where the federated function has a high memory configuration (e.g. 10 GB), the proxy function overhead which waits for the called function to complete provides only a 1.25% cost increase. With synchronous load distribution additional costs are equivalent to the target function having an additional 128 MB of allocated memory. For computational workloads that require at minimum one vCPU, the target function should always be deployed with at minimum 2 GB of memory per vCPU (e.g. a function that uses two vCPUs should have 4 GB) on AWS Lambda. To federeate most serverless functions, utilizing the minimum memory setting for the proxy function results in much lower cost than what is implied by "double billing". Figure 1 shows how the Analyzer function is used to inform the Proxy functions and how the Proxy function is used to route requests around the world.

TABLE II
AWS REGIONS USED FOR SERVERLESS FEDERATIONS
AWS LAMBDA PRICING, AVERAGE CPU STEAL PER MINUTE, AND CPU CLOCK SPEED DISTRIBUTION.

Region Location	Price (1E-5)	CPU Steal/min	CPUs % 2.5GHz	% 2.9GHz	% 3.0GHz
Hong Kong	2.29	6.0	100.0	0.0	0.0
Tokyo	1.67	24.0	98.33	0.0	1.67
Seoul	1.67	9.6	96.5	0.0	3.5
Osaka	1.67	6.6	98.9	0.0	1.1
Mumbai	1.67	15.0	99.38	0.0	0.62
Singapore	1.67	15.6	98.57	0.0	1.43
Sydney	1.67	9.6	94.2	0.0	5.8
Frankfurt	1.67	27.6	100.0	0.0	0.0
Stockholm	1.67	7.8	94.86	0.0	5.14
Milan	1.95	11.4	100.0	0.0	0.0
Ireland	1.67	35.4	93.51	0.0	6.49
London	1.67	16.8	98.62	0.0	1.38
Paris	1.67	7.8	99.67	0.0	0.33
Canada	1.67	7.2	96.19	0.0	3.81
Sao Paulo	1.67	15.6	98.48	0.0	1.52
N. Virginia	1.67	30.0	98.66	0.0	1.34
Ohio	1.67	20.4	80.69	2.16	17.15
N. California	1.67	18	99.95	0.0	0.05
Oregon	1.67	29.4	100.0	0.0	0.0

In the future, we plan to investigate alternate proxy function architectures. One such option is asynchronous forwarding, where the proxy is called asynchronously by the client and the proxy calls the target function asynchronously. This approach features very little cost overhead, where the proxy function only adds on average 2 ms of runtime, and one additional function invocation. After the request is made, the client is responsible for retrieving the function's response.

C. Experiment Design

To test our workloads across a global serverless federation we deployed each of our benchmark functions defined in Table I to 19 AWS regions as shown in Table II. Proxy functions were also deployed to each region and the Analyzer function was deployed in Ohio. Using these functions we conducted five experiments:

EX-1 (Carbon Data Collection): The first experiment used an AWS CloudTrail trigger to invoke the Analyzer function every 15 minutes. The Analyzer collects carbon intensity data for the 19 regions and saves the results for future use. This experiment ran from November 2022 to March 2023 and was used to observe how the carbon intensity of each region varied.

EX-2 (Network Latency): The second experiment focused on measuring network latency between cloud regions and

observing variation in latency over time for **(RQ-1)**. We utilized the HTTP Request function as a client which called the Hello World function deployed to the North and South American regions (N. California, Oregon, Ohio, N. Virginia, Canada, and Sao Paulo). For the experiment we invoked the HTTP Request functions in every region to provide FaaS clients in six different locations. These clients then called every Hello World function in every region, including the source region, to measure round-trip latency. The experiment ran with a 15 minute interval from November 2022 to March 2023. The goal of this experiment was to quantify network latency statistics between regions over a long period of time, and to also measure how round trip network latency correlates with the distances between regions. This experiment is vital for understanding the latency impact of aggregating serverless computing resources from multiple regions to form serverless federations.

EX-3 (Dual-region Load Distribution): The third experiment evaluated the best and worst-case trade-offs for reducing carbon intensity to determine the ensuing impact on function latency resulting from using the proxy function for **(RQ-2 and RQ-3)**. For this experiment, we examined the outcomes of proxying function requests to small two-region federations. We selected the Calcs function as a compute bound function and then had the proxy function distribute requests dynamically to one of the two target regions in the federation based on conditions. We used Ohio and Oregon for the American continents, London and Frankfurt for Europe, and Hong Kong and Sydney for Asia/Oceania. The choice of these regions was based upon the results of EX-1, where all of these regions have similar, but constantly changing, carbon intensity.

EX-4 (Global Load Distribution): The fourth experiment expands on EX-3. Instead of focusing on small two-region federations, we expanded the scope to a global scale. For this experiment we used all of the functions from Table I and deployed them to every region in Table II to create a global 19 region serverless federation. The proxy function was also deployed to every region. We executed the experiment by using every region as a client where each client made requests to our serverless load distribution proxy function in the next nearest region. This design simulates users being distributed throughout the world instead of being located at the source region. The proxy function distributed requests to other regions based on five different load distribution techniques:

1) Ohio: Simulates a FaaS application that is only deployed to a single region, in this case, Ohio (us-east-2).
2) Minimize Carbon: Routes functions to the nearest region with the lowest possible carbon intensity.
3) Minimize Distance: Routes requests to the nearest region (other than its own region) to minimize network latency.
4) Balanced Weight: Weights the competing objectives of low carbon and low network latency equally. To simplify routing, distance is used as a proxy for network latency. For **(RQ-1)** we verify that the physical distance between client and server cloud regions correlates strongly with network latency and can serve as a reasonable facsimile. The percent increase in carbon intensity and the percent increase in distance between two regions are weighted 50-50. If a region has slightly worse carbon intensity than another but is significantly closer this method will choose the closer region.
5) Weighted on Distance: Same as Balanced Weight but applies three times more weight for distance. This way low network latency will be favored more than low carbon intensity.

This experiment then ran every function on every region with each load distribution technique every 30 minutes for 10 days. The goal of this experiment was to evaluate the carbon, latency, cost, and performance implications of each load distribution technique.

EX-5 (Performance based load distribution): The fifth experiment does not optimize for carbon or network latency but instead focuses on improving function performance **(RQ-4)**. We deployed six instances of the Stress function to the Ohio region having different memory settings: 1.7 GB, 3.4 GB, 5.1 GB, 6.8 GB, 8.5 GB and 10 GB. These memory settings represent the points where AWS Lambda provisions an additional vCPU core for the serverless function up to six total vCPUs. We then invoked the function by specifying the number of threads in the request payload and our proxy function routed our requests to a function deployment with a memory setting that provided the required number of vCPU cores for the specified number of threads. The Stress function sustains vCPUs at 100% utilization for five seconds. We then compared function invocations using the proxy versus function invocations running at 10 GB to determine the reduction in CPU idle time. Function calls reporting significant idle time indicate that the function instance had over provisioned memory. By leveraging our proxy function to dynamically adapt memory and CPU resources relative to the function call's requirements we study the potential to reduce function execution costs.

IV. Experimental Results

The following sections present the results of experiments defined in Section III-C.

A. Carbon Data Analysis

The Analyzer function collected carbon data for 19 cloud regions around the world. We attempted to collect carbon data for every AWS region but some regions did not have available data, so that is why we chose the 19 regions described in Table II. Carbon data was collected from November 2022 to March 2023. Over this time we make a number of useful observations and describe our analysis for three serverless federations: Americas, Europe, and Asia/Oceania **(RQ-2)**.

1) American Carbon Intensity: From our observations of the American regions, our serverless load distribution proxy function can take advantage of several factors. Regions with large fluctuations in fossil fuel percentage can be exploited to run workloads when carbon intensity is low. Oregon, Sao Paulo, and North California had a high coefficient of

variation (CV) of 51%, 32%, and 25%, respectively. However, Sao Paulo's average fossil fuel percentage was only 6.5%, compared to all other regions in the United States, which were over 40%. In the United States, since all regions have similar average fossil fuel usage (40% to 58%) with large CVs, the serverless proxy function can distribute requests across all of the regions at different times. However, Canada is a special outlier: at nearly all times, Canada had 0% fossil fuel usage. The overall average fossil fuel usage in Canada was only 0.02%, with a peak of 3%, making Canada better than all other regions in North and South America at any time. These observations are illustrated in Figure 2.

2) European Carbon Intensity: The carbon intensity of European regions on average had higher fossil fuel usage than the American regions. Milan, Ireland, Frankfurt, London, and Paris had average fossil fuel usage of 78.7%, 56.7%, 47.9%, 41%, and 12%, respectively. The CV ranged between 4% and 18.5%. However, many regions in Europe achieved 0% fossil fuel usage at some point; including Frankfurt, Stockholm, London, and Paris. Like Canada in the American regions, Stockholm had 0% fossil fuel usage nearly all the time, with an overall average of just 0.2% fossil fuel usage and only occasional jumps to 1-5%.

3) Asia/Oceania Carbon Intensity: In contrast to Europe or America, for all regions in Asia, the minimum usage of fossil fuel was above 49%. There was at no point a region with 0% fossil fuel usage. These regions also demonstrated a consistent diurnal pattern in which the carbon intensity plateaus for approximately 12 hours each day before decreasing for the remaining 12 hours. Singapore, Tokyo, Mumbai, Osaka and Seoul averaged 96.1%, 84.7%, 80.2%, 69.8% and 69.7% respectively. All of these regions had comparatively low CV compared to regions in Europe or America ranging from 1% to 9%. To illustrate this plateauing behavior, Figure 3 depicts the fossil fuel percentage of each region in Asia/Oceania over the course of January 2023. The only region that did not exhibit this plateauing behavior was Sydney, which had a much higher fossil fuel CV of 25%, an average of 65%, and a minimum fossil fuel percentage of 8%. Sydney's fossil fuel profile more closely resembled those observed in Europe or the Americas.

B. Latency Impact of Serverless Federations

We measured the network latency between each of the 19 regions over the same time period as the carbon intensity data for **(RQ-1)**. We found that distance was a strong predictor of network latency. Figure 4 shows the relationship between the distance between regions and the measured request latency. Using linear regression the variance explained when using distance as the sole predictor for network latency was (R^2=0.992).

There are multiple types of latency in the context of serverless platforms. One type is network latency, the time for a request to travel from the client to the FaaS platform. This latency increases with distance and can be easily predicted (see Figure 4). Another type of latency is function cold-start latency, the delay resulting from infrastructure initialization on the FaaS platforms to create runtime environments to service client requests for user applications. To mitigate cold-start latency for this experiment, we utilized the same FaaS function instance for both the client and host functions by deploying multi-purpose functions to effectively warm the infrastructure, make cold-start latency negligible, and allowing any region to be a workload or proxy function. Out of over 203,000 client requests, zero requests were serviced by cold infrastructure. Alongside making requests across regions, we also measured intra-region latency, where client calls were made to functions in the same region, enabling us to measure the baseline warm function latency, with as little network "travel" latency as possible. For all regions, the baseline latency was on average between 45-48 ms, with on average 3.1 ms of latency added for every 100 km of travel distance. If requests are made to a nearby data center, resulting in 100 ms of total latency and our application requires less than 200 ms of latency, based on our observations we could make a request to a data center up to an additional 1,612 km away and still meet the requirement.

For all regions we observed variation in network latency between 2% and 29% throughout the day. For example, Figure 6 shows the latency between the Ohio and North Virginia regions normalized over a 24-hour day. We aggregated 5 months of data across each hour of the day using local time (e.g. hour 0 to hour 23) to calculate average network latency and the coefficient of variation to observe trends relative to the human wake, sleep, and work cycles. Even with high CV between 12% and 20%, the average latency only varied +/-10 ms.

C. Runtime Impact of Serverless Federations

Working at a global scale with functions having the ability to run on nearly any region in the world presented new challenges not identified in our previous work. First, all regions had

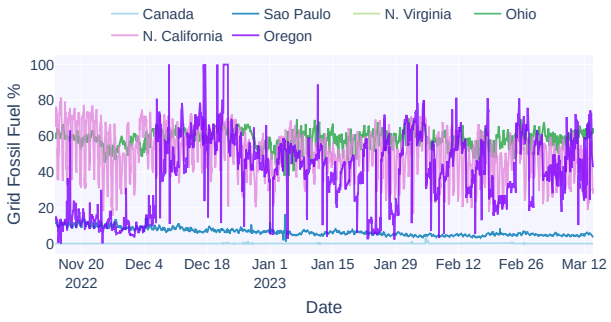

Fig. 2. Grid fossil fuel usage percent for North and South America.

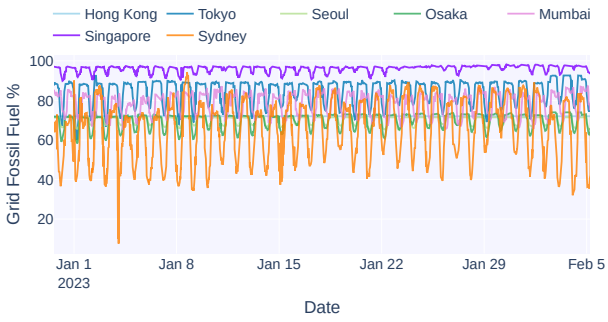

Fig. 3. Fossil fuel usage percent for Asia/Oceania Regions in January 2023.

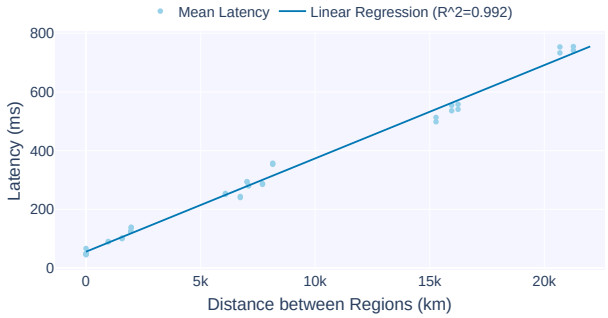

Fig. 4. Request round trip latency (ms) versus distance (km) between any two regions.

varying degrees of hardware heterogeneity. We found across all regions there were three different reported vCPU clock speeds: 2.5 GHz, 2.9 GHz, and 3.0 GHz. Each region then had varying quantities of each CPU clock speed and varying average CPU Steal per minute as shown in Table II. CPU Steal has previously been shown to be useful for estimating the number of tenants sharing host infrastructure [20]. We found Ohio had the most diverse range of vCPUs with 80.69%, 2.16% and 17.15% with function requests serviced by 2.5 GHz, 2.9 GHz, or 3.0 GHz clock speed vCPUs respectively. All other regions had over 90% of their function invocations fulfilled by a 2.5 GHz vCPU with all remaining requests fulfilled with the 3.0 GHz option. All function invocations used equivalent payloads and seeds resulting in deterministic work at 2 GB of memory (allowing function instances one full vCPU). In Ohio, the most heterogeneous region, we saw on average 3.2% CV for function runtime across all workloads except Stress and HTTP Request. **When expanded to a global federation and using the Minimize Distance distribution technique (to invoke requests on as many regions as possible), we saw the CV of function server-side runtime double to 6.5%.**

For other distribution techniques, such as Minimizing Carbon, we observed runtime CV of 3.1%. This test leveraged infrastructure from a smaller number of regions compared to Minimize Distance load distribution technique. Overall, we did not observe a large increase in performance variation (increasing only by 3%) when expanding our federation to 19 regions globally. For specific workloads, such as Calcs and CSV Processor, we did observe a reduction in runtime CV by expanding from a single region to global. The runtime of the Calcs function is shown in Figure 5, where the CV was

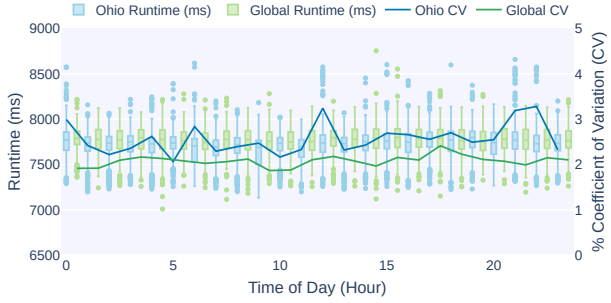

Fig. 5. Server-side runtime variation of running a workload in Ohio compared to a global federation of functions.

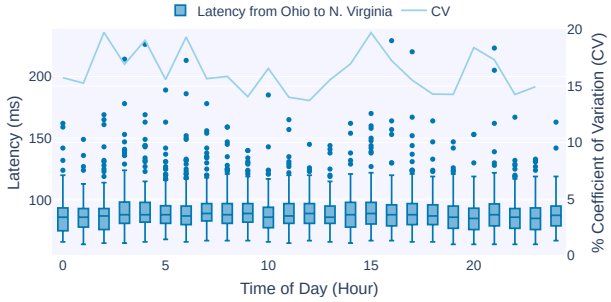

Fig. 6. Hourly average network latency and coefficient of variation between North Virginia and Ohio from November 2022 to March 2023

between 0.5 and 1% lower using the global federation **(RQ-1)**.

D. Serverless Load Distribution on Regional Federations

To analyze the efficacy of our serverless load distribution techniques we first deployed the proxy function to a small set of regions, to compare distributing requests to small two-region federations before expanding the experiment to a global scale **(RQ-2 and RQ-3)**.

Initially, we focused on the Minimize Carbon load distribution technique using Oregon and Ohio, as these two regions presented a challenging distribution scenario. The region with the lowest carbon footprint among these two regions frequently changed. Over the 19 days of the experiment, the load distribution system switched between regions 36 times. When using these two regions as a North American federation, with client requests coming from all other North American regions, the proxy function reduced the overall carbon footprint of every workload by an average of 16% compared to running the workload in Ohio and 3% compared to running in Oregon **(RQ-2)**. Conversely, it reduced overall latency by 18% compared to running all requests through Oregon but increased latency by 9% compared to running all requests in Ohio. Network latency CV was 61%, 69%, and 65% for Oregon, Ohio, and the proxy function respectively **(RQ-3)**. Figure 7 shows the proxy making a choice to send requests between the two regions and the total fossil fuel usage of each distribution option.

We executed a similar dual-region experiment in Europe using the London and Frankfurt regions. Compared to North America, this experiment resulted in 33% fewer region switches (24 total). At the start, London had lower fossil fuel usage than Frankfurt for about the first week. After

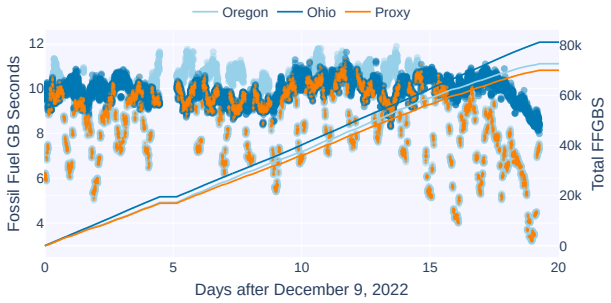

Fig. 7. Carbon intensity reduction using our Minimize Carbon load distribution approach to distribute requests between two regions in North America.

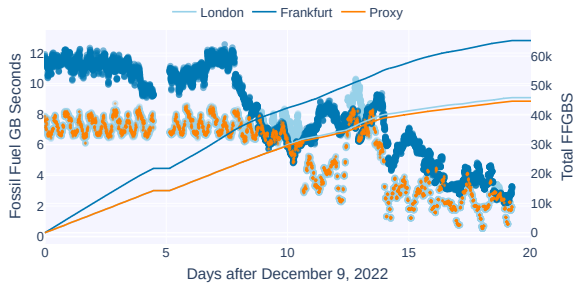

Fig. 8. Reduction in carbon intensity using our Minimize Carbon load distribution approach to distribute requests between two regions in Europe.

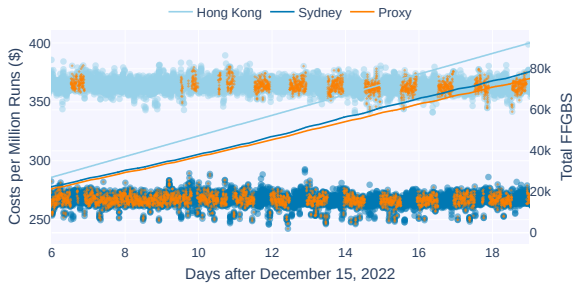

Fig. 9. Reduction in carbon intensity and hosting costs using our Minimize Carbon load distribution approach to distribute requests between two regions in Asia/Oceania with different pricing models.

that, Frankfurt improved and the proxy began distributing requests between the two regions. This difference in fossil fuel usage resulted in the proxy reducing carbon intensity by 46% compared to running all requests in Frankfurt **(RQ-2)**. This federation resulted in requests to London having 35% more latency than running all requests in Frankfurt. Since the proxy function favored London in the beginning, we saw a 29% increase in latency using the proxy compared to running all requests in Frankfurt. Latency CV was 61%, 69%, and 62% for London, Frankfurt, and the proxy function respectively **(RQ-3)**. Figure 8 shows the proxy making a choice to send requests between the two regions and the total fossil fuel usage of each deployment.

The regions selected for the Asia/Oceania federation provide a unique scenario that serverless federations can take advantage of. For this experiment we selected the Sydney and Hong Kong regions. Similar to all of the other tests, our serverless proxy was able to reduce carbon intensity by 23% and 4% respectively compared to running all of the requests on Hong Kong or Sydney **(RQ-2)**. Where this experiment was different was in terms of the hosting costs, in the American regions the runtime of our functions running on either Oregon or Ohio was within 1% of each other. Since both these regions use the same pricing model, the overall hosting costs of using either of these regions was also within 1% of each other. Hong Kong is a unique region that has a different AWS Lambda pricing model than most other regions, in Hong Kong the price per GB/sec of runtime is 37.5% higher. This price difference makes it so that our proxy function is not only reducing the carbon intensity of a workload running in Hong Kong but also reducing the cost. Running the same workload in Sydney resulted in a 36% decrease in cost, which is expected, while running the workload with the load distributor resulted in a 19% cost decrease compared to running all requests in Hong Kong. Of our 19 regions, only two featured different pricing models compared to the rest as shown in Table II. In larger federations where resources are federated across multiple cloud providers, this demonstrates how we can exploit differences in pricing models to reduce the overall hosting costs of an application. Figure 9 shows the difference in pricing models and how the proxy distributed requests.

E. Serverless Load Distribution on Global Federations

After evaluating load distribution across regional serverless federations to verify that carbon footprint reductions were possible without massive increases in network latency, we investigated using a global federation **(RQ-2)**. 19 AWS regions were combined as a single serverless federation and local proxy functions were deployed in every region. We also used each of these regions as clients. We then deployed our workloads in Table I. To evaluate each of our load distribution techniques we made over 360,000 proxy function invocations which made 360,000 more calls to workload functions.

TABLE III
COMPARISON OF SERVERLESS LOAD DISTRIBUTION TECHNIQUES

Name	Regions Used	Average Latency	Latency CV	Average FF-GBS	Cost Per 1m
Ohio	1	474	50	568,000	$65.25
Minimize Carbon	2	600	49	128	$64.64
Minimize Distance	12	166	72	560,000	$67.01
Weighted Evenly	2	516	70	134	$64.05
Weighted Distance	6	489	71	440	$64.64

At a global scale the proxy function has the potential to move a serverless workload entirely off of using predominantly fossil fuel based electricity grids as some regions have 0% fossil fuel usage. As expected, we saw a massive decrease in FF-GBS when comparing the Minimize Carbon distribution technique to our other distribution schemes. For example, a single region deployment to Ohio resulted in 776 thousand FF-GBS to fulfil 6,000 function requests for each of our workloads (72,000 requests total). When using the Minimize Carbon technique we saw that the same 72,000 requests became just 172 FF-GBS because functions were hosted in either Canada or Stockholm depending on which is closer as both of these regions would often have 0% fossil fuel usage (99.8% reduction). Compared to Ohio, the Minimize Carbon distribution technique only increased average latency by 20%, but neither of these techniques are very good compared to our Minimize Distance distribution technique. Using Ohio or Minimize Carbon increased the overall network latency by 152% and 161% respectively compared to Minimize Distance. For applications that require low latency, federating resources across many regions and utilizing a load distribution technique that minimizes request travel distance has immense potential for performance improvements **(RQ-3)**.

For applications that are not latency dependent, we can obtain a similar fossil fuel reduction to the Minimize Carbon technique with lower latency by using a weighted approach that takes into account both parameters. We investigated two schemes for weighting physical distance and carbon intensity: equal weighting and weighting distance 3x more. The equal

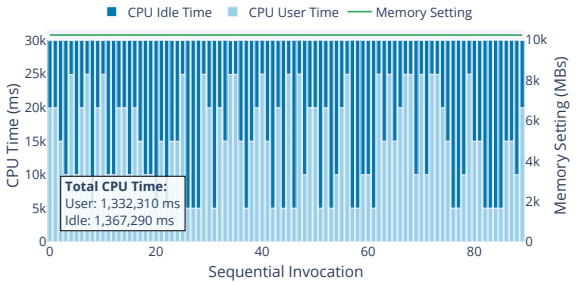

Fig. 10. Stress function invocations with a random number of stressed vCPUs at 10240 MBs. High idle time show runs with over provisioned memory settings, resulting in equivalent runtime but significantly higher cost.

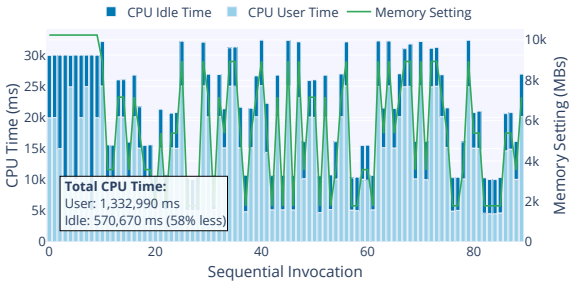

Fig. 11. Proxy routing requests to function deployments with optimal configurations. CPU User time remains nearly identical. Idle time is significantly reduced which results in lower costs and equivalent performance.

weighted distribution technique behaved very similar to Minimize Carbon as it tended to run functions in either Canada or Stockholm due to their incredibly low carbon footprint while also lowering average network latency by an average of 17%. By weighting physical distance more heavily (for low network latency) this increased the number of regions that workloads ran on up to six, reducing network latency 22% compared to the Minimize Carbon technique. The evenly weighted load distribution approach achieved nearly identical low total FF-GBS compared to the Minimize Carbon Technique, with an average of 134 and 128 FF-GBS for each approach respectively. The low distance weighting option increased FF-GBS by up to 440 **(RQ-2)**. Table III shows the results of each load distribution technique on average with our set of workloads.

F. Multi-configuration Federation

To evaluate a multi-configuration serverless proxy function we deployed the Stress function to the Ohio region with six different memory configurations **(RQ-4)**. The idea is that the multi-configuration proxy function directs client requests to the function deployment with an ideal configuration for the client payload. Here our Stress function takes a parameter to specify the number of vCPUs to stress. For our client testing we created a random set of request payloads that request from one to six vCPUs. The proxy then read the input payload, and used the CPU Time Accounting Memory Selection (CPU-TAMS) model to predict a function memory setting, and distribute client requests to the function deployments that offered the appropriate number of vCPUs and memory to achieve the best price to performance ratio [22].

Figure 10 and 11 show how the function performance and costs can be optimized by eliminating over-provisioning of function memory. Dynamic distribution of client requests enabled the function invocations to achieve nearly the same runtime with a lower memory allocation. In figure 11, we utilize the proxy function to reduce over-provisioning of function memory with minimal impact on function runtime. Across the 90 function invocations depicted, we retained equal CPU user time while reducing the idle time by 58% (which resulted in a ∼50% reduction in cost), while increasing the overall runtime by just 7.8% versus running all functions with maximum memory (i.e. 10 GB). When extrapolating this savings for one million function invocations, our approach can reduce application hosting cost from $833 to $349, a savings of ∼58%, using multi-configuration load distribution **(RQ-4)**.

V. Conclusions

This paper has introduced the concepts of "Federating Serverless Computing" by harnessing our prototype federated load distribution system. To investigate the potential benefits and implications for serverless federations, we first observed how carbon intensity and network latency changed from November 2022 to March 2023 across 19 cloud regions. **(RQ-1 Performance Variation):** We found that latency had a coefficient of variation between 2-29% during the day, varying on average +/-10 ms. Function runtime varied much less, with 3 to 6% CV across all workloads. **We found that distance was a strong predictor for latency, with an R^2 of 0.992. (RQ-2 Carbon Intensity):** We evaluated 19 regions across the world. Canada and Stockholm exhibited the lowest fossil fuel percentage for electricity generation which was 0% for the majority of time. **(RQ-3 Sustainability Costs):** Using our twelve workload functions and using each region in the world to simulate a globally distributed application with users around the world, we evaluated our load distribution system with multiple federations and distribution techniques. **Compared to workloads being deployed in a single region, by utilizing our serverless proxy deployed globally, we were able to reduce latency by on average 65% while reducing the carbon intensity by up to 99.8%. (RQ-4 Multi-configuration Federation):** By deploying a function with multiple different memory configurations we were able to leverage the CPU-TAMS model [22] in our proxy function. Using this model we were able to distribute function requests to function deployments to avoid over-provisioning vCPUs or memory to obtain the best price to performance ratio. **Multi-configuration federations were able to reduce function hosting cost by 58% reducing the cost of one million function invocations from $833 to $349.**

Acknowledgments

This research has been supported by AWS Cloud Credits for Research.

References

[1] "Environmental impacts of data centers and the cloud," popsci.com/environment/data-centers-environmental-impacts/, 2022, accessed: 2023-02-25.

[2] A. Katal, S. Dahiya, and T. Choudhury, "Energy efficiency in cloud computing data centers: a survey on software technologies," *Cluster Computing*, vol. 25, no. 5, pp. 1–18, 2022. [Online]. Available: link.springer.com/article/10.1007/s10586-022-03713-0

[3] S. R. Department. (2023) Global electricity consumption 1980-2021. Accessed: 2023-02-25. [Online]. Available: statista.com/statistics/280704/world-power-consumption/

[4] R. Cordingly and W. Lloyd, "Faaset: A jupyter notebook to streamline every facet of serverless development," in *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering*, 2022, pp. 49–52.

[5] "Sustainability in the cloud," aws.amazon.com/sustainability/, accessed: 2023-02-18.

[6] "Google cloud sustainability," cloud.google.com/sustainability, accessed: 2023-02-18.

[7] N. Bashir, T. Guo, M. Hajiesmaili, D. Irwin, P. Shenoy, R. Sitaraman, A. Souza, and A. Wierman, "Enabling sustainable clouds: The case for virtualizing the energy system," in *ACM Symposium on Cloud Computing (SoCC)*, 2021.

[8] R. Farahani, D. Kimovski, S. Ristov, A. Iosup, and R. Prodan, "Towards sustainable serverless processing of massive graphs on the computing continuum," in *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*. ACM, 2023, pp. 221–226.

[9] C. P. Smith, A. Jindal, M. Chadha, M. Gerndt, and S. Benedict, "Fado: Faas functions and data orchestrator for multiple serverless edge-cloud clusters," in *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*. IEEE, 2022, pp. 17–25.

[10] S. Chasins, A. Cheung, N. Crooks, A. Ghodsi, K. Goldberg, J. E. Gonzalez, J. M. Hellerstein, M. I. Jordan, A. D. Joseph, M. W. Mahoney *et al.*, "The sky above the clouds," *arXiv preprint arXiv:2205.07147*, 2022.

[11] I. Stoica and S. Shenker, "From cloud computing to sky computing," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, 2021, pp. 26–32.

[12] Y. Mao, "Skybridge: A cross-cloud storage system for sky computing," in *23rd International Middleware Conference Doctoral Symposium*. ACM, 2022, pp. 15–17.

[13] A. F. Baarzi, G. Kesidis, C. Joe-Wong, and M. Shahrad, "On merits and viability of multi-cloud serverless," in *Proceedings of the ACM Symposium on Cloud Computing*, 2021, pp. 600–608.

[14] J. Sampé, P. García-López, M. Sánchez-Artigas, G. Vernik, P. Roca-Llaberia, and A. Arjona, "Toward multicloud access transparency in serverless computing," *IEEE Software*, vol. 38, no. 1, pp. 68–74, 2021.

[15] A. Jindal, J. Frielinghaus, M. Chadha, and M. Gerndt, "Courier: Delivering serverless functions within heterogeneous faas deployments," in *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing*. ACM, 2021.

[16] R. Cordingly, H. Yu, V. Hoang, Z. Sadeghi, D. Foster, D. Perez, R. Hatchett, and W. Lloyd, "The serverless application analytics framework: Enabling design trade-off evaluation for serverless software," in *Proc of the 2020 Sixth Int. Workshop on Serverless Computing*, 2020, pp. 67–72.

[17] R. Cordingly, N. Heydari, H. Yu, V. Hoang, Z. Sadeghi, and W. Lloyd, "Enhancing observability of serverless computing with the serverless application analytics framework," in *Companion of the 2021 ACM/SPEC Int. Conf. on Performance Engineering, Tutorial*, 2021.

[18] M. Copik, G. Kwasniewski, M. Besta, M. Podstawski, and T. Hoefler, "Sebs: A serverless benchmark suite for function-as-a-service computing," *arXiv preprint arXiv:2012.14132*, 2020.

[19] "Stress(1)," 2012. [Online]. Available: linux.die.net/man/1/stress

[20] R. Cordingly, W. Shu, and W. J. Lloyd, "Predicting Performance and Cost of Serverless Computing Functions with SAAF," in *6th IEEE Int. Conf. on Cloud and Big Data Computing (CBDCOM 2020)*, 2020.

[21] R. Cordingly, "Serverless performance modeling with cpu time accounting and the serverless application analytics framework," 2021.

[22] R. Cordingly, S. Xu, and W. Lloyd, "Function memory optimization for heterogeneous serverless platforms with cpu time accounting," in *2022 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2022, pp. 104–115.

[23] "Electricity maps," electricitymaps.com, accessed: 2022-12-01.

[24] S. Quinn, R. Cordingly, and W. Lloyd, "Implications of alternative serverless application control flow methods," in *Proceedings of the Seventh International Workshop on Serverless Computing (WoSC7) 2021*, 2021, pp. 17–22.

[25] I. Baldini, P. Cheng, S. J. Fink, N. Mitchell, V. Muthusamy, R. Rabbah, P. Suter, and O. Tardieu, "The serverless trilemma: Function composition for serverless computing," in *Onward! 2017 - Proc of the 2017 ACM SIGPLAN Int. Symp. on New Ideas, New Paradigms, and Reflections on Programming and Software, co-located with SPLASH 2017*, 2017.