Vrije Universiteit Amsterdam

Bachelor Thesis

An analysis of the performance and carbon footprint of workloads in datacenters using serverless models in simulation

Author: Ana-Maria Muscă (2757719)

1st supervisor:	Prof. dr. ir. Alexandru Iosup
daily supervisor:	Dante Niewenhuis, MSc
2nd reader:	Prof. dr. ir. Tiziano de Matteis

A thesis submitted in fulfillment of the requirements for the VU Bachelor of Science degree in Computer Science

June 8, 2026

Abstract

Over the past decade, the volume of processed data has grown exponentially, driving a corresponding rise in datacenter energy consumption and, consequently, higher carbon emissions. While the scientific community has proposed multiple carbon-aware workload processing approaches, most of them are limited to the scope of reducing emissions without considering the tradeoff between performance and carbon emissions. The serverless paradigm represents an opportunity to address this trade-off. Serverless computing has proven to enhance performance in Function-as-a-Service applications thanks to its efficient way of managing resources. However, the benefits of serverless computing, when extended to more complex contexts, such as the datacenter scheduling system, have not been explored.

This thesis aims to address the performance-carbon emissions tradeoff by designing, implementing, and evaluating a serverless sustainable workload processing model that combines serverless computing with load shifting techniques. Our results indicate that, while a serverless workload processing approach can reduce the overall carbon emissions (up to 3%), the improvements in carbon vary depending on the task transfer costs, the geographical region, and the reliability of the datacenters.

Acknowledgments

First, I would like to thank Dante Niewenhuis for his patience and his guidance through this thesis journey. His willingness to engage in meaningful discussions and respond to my never ending questions, even outside normal working hours, demonstrated a level of commitment for which I am truly grateful.

Second, I want to express my gratitude to Alexandru Iosup for introducing me to the world of distributed systems and for continuously inspiring me through both his teaching and his guidance. His combination of technical insight and motivational speeches made every meeting productive and encouraging.

Third, I would like to thank my friends for always being there for me and cheering me up. They always find the best ways to show me the bright side of things, even in my most negativistic moments.

Last but not least, I want to thank my family for their unconditional support and belief in me. Your constant encouragement gave me the strength to continue fighting for my goals and stay focused, even when things felt overwhelming.

Contents

1

Introduction

Over the past decade, the annual amount of collected data has increased exponentially, reaching zettabytes in magnitude (1, 2). This led to a significant increase in the energy consumption of datacenters globally. Currently, the energy consumption of data centers worldwide represents more than 1% of the global energy consumption (3), and it is expected to reach approximately 325 MtCO2 by 2030 (4). The more energy is required by a datacenter, the more carbon emissions will be produced, as the energy consumption directly influences the carbon emissions. As a result, carbon emissions from datacenters currently represent 3% of carbon emissions worldwide (5). Even though the scientific community estimates that after 2030, the carbon emissions generated by datacenters will slowly be reduced, it will still take decades until we reach a carbon level close to the one in 2020, which was already significantly high (approximately 125 MtCO2) (4). In order to speed up this carbon reduction, more sustainable and smarter ways in which workloads would be scheduled and executed in a datacenter need to be developed. Those new approaches have to take into consideration the performance-carbon emissions tradeoff in order to guarantee an optimal execution of datacenters' tasks and minimal carbon emissions.

The serverless paradigm is an emerging topic in the scientific community, which, from the client point of view, is described as an application deployment architecture that completely hides server management from tenants (6). An important quality of serverless computing is that it provides required resources on demand during the execution of an application and reduces idle time, resulting in better resource management and performance improvements (7). Although the benefits of serverless computing were mostly explored in the context of Function-as-a-Service (FaaS) applications (6, 8, 9, 10), this approach constitutes a starting point for better datacenter resource management and reduction in carbon emissions.

In this work, we propose a sustainable serverless processing model that integrates serverless computing with operational techniques like spatial and temporal load shifting, aiming to demonstrate the benefits of serverless task scheduling and execution at the datacenter level in terms of performance and carbon footprint. As experimenting directly with real datacenter infrastructure is often impractical due to high costs, operational risks, and limited scalability, we implement our model in a discrete event simulator and analyze its behavior regarding performance and carbon emissions in comparison with a non-serverless workload processing model through a series of experiments.

1.1 Problem Statement

Despite active research into reducing the carbon footprint of datacenters, many datacenters still rely on traditional resource allocation strategies during task scheduling and execution. These approaches often prioritize the hardware requirements of a task, assigning tasks to the first suitable host without considering the trade-off between performance and carbon emissions. As a result, resources may be used inefficiently, leading to execution delays and increased energy consumption and carbon emissions.

Although serverless computing was previously used to better manage resources in FaaS applications (11, 12), there remains a limited understanding of its potential benefits when extended to more complex contexts, such as the datacenter scheduling system. **A deep understanding of how serverless aspects apply to the scheduling process of a task and the lack of a general serverless datacenter model represent a key challenge in identifying these benefits**.

Similarly, the development of new operational techniques, such as spatial shifting, has been shown to significantly reduce the carbon emissions of processing a workload (5, 13, 14), but these approaches also experience difficulties in finding ways to address the trade-off between performance and carbon emissions. As a result, we identify **a knowledge gap regarding the benefits of combining serverless computing with load shifting techniques in order to address the performance-carbon emissions tradeoff**.

Regarding spatial shifting, previous works show the benefits of this load shifting approach by considering ideal scenarios, in which the transfer costs and the reliability of a datacenter are negligible (5, 13). For a more accurate representation of the benefits of spatial shifting when applied to real datacenters, we emphasize the importance of simulating workload scenarios and running experiments that cover those aspects.

1.2 Research Questions

To address the challenges and the contributions of this project, we will answer the following main research question:

MRQ How to analyze the performance and carbon footprint of workloads in datacenters?

The main research question highlights the goal of the project, which is to analyze the performance and carbon emissions of workloads in datacenters using serverless models. As the main research question includes multiple stages of the project, such as modeling, implementation, and experimentation, we designed three sub-research questions, each prioritizing one component of the project. The sub-research questions are:

RQ1 How to model workloads running in datacenters using serverless models?

The serverless paradigm is a concept that was mostly linked with Function-as-a-Service, and was very little explored in combination with workload processing. Due to this, the first step in analyzing the benefits of serverless in workload processing is to design a model that includes serverless characteristics and that also takes into account the carbon intensity of a datacenter. Building such a model requires an in-depth understanding of the datacenter components that will be involved in the process of workload processing, such as the scheduler. This is a crucial part of the project, as without an appropriate model, we would not be able to proceed to the following parts of the project. The challenge of this question is mostly related to finding the best hardware configurations and scheduling strategies to be implemented in the model that take into consideration both performance and carbon emissions.

RQ2 How to simulate datacenters that use a serverless model for workload processing to optimize for performance and climate impact?

Once a model for serverless workload processing is built, the next step is to implement it in a discrete event simulator, namely OpenDC (15). A good implementation of the model needs to consider the system requirements and structure of OpenDC and the aspects of serverless. In order to make a good implementation, there are some challenges to overcome, such as the integration of different scheduling techniques in the process of task scheduling,

which might conflict with the existing code and execution. Moreover, changes in policies have to be made such that the carbon footprint will play a significant role in the process of selecting hosts for scheduling tasks and in the process of task execution. Any modification of the current OpenDC code base caused by the integration of our model will require an in-depth understanding of the main components and code structure of OpenDC. To avoid altering the core functionality of the discrete event simulator, extra time will be allocated for code development and debugging. As a result, the second research question will have the integration of the model developed in the previous stage of the project as its main goal, while ensuring the structural consistency and the functionality of OpenDC.

RQ3 What is the impact of datacenters using serverless models for workload processing, compared to non-serverless models for performance and climate impact?

The last stage of this project is the evaluation of our model, which will be composed of three parts: the designing and implementation of experiments, the execution of the experiments for our model, and the comparison of the results and the analysis of the benefits and disadvantages of our serverless workload processing model. This step is crucial as we need to make sure that our experiments are reproducible and replicable in order to guarantee the credibility and authenticity of our results. Moreover, another challenge of this project stage is concerned with the design and implementation of the experiments, as we will have to create experiments that test and evaluate as much as possible the functionality of our model and produce enough feedback in order to be able to conduct an in-detail analysis of the advantages and disadvantages of our model.

1.3 Research Methodology

Because this work includes multiple steps, starting from designing a serverless sustainable workload processing model to implementing it in a discrete event simulator and testing it through a series of experiments, we consider that this work integrates multiple methodologies.

In order to answer **RQ1**, we first performed a quantitative investigation by reading previous related papers regarding the serverless paradigm, workload processing, and sustainability in datacenters, and by finding patterns that could apply to our model (16). The goal of this question was to create a sustainable, serverless model for workload processing, so during the design stage and while answering **RQ2**, we used *The AtLarge Design Process*

(17) as a reference. Therefore, during the implementation stage (**RQ2**), we integrated multiple versions of the model designed in **RQ1** until we reached an implementation that also aligns with the discrete event simulation principles.

During the evaluation stage of this work, meant to answer **RQ3**, we used experimental research in order to highlight the benefits and limitations of our model (18). This stage required an in-depth exploration of different workloads and datacenter topologies, and an analysis of the link between the number of tasks performed in a datacenter per time unit and the carbon intensity at that time unit. Moreover, this work is aligned with the reproducibility and open source principles (19) in order to maintain the credibility and authenticity of the results and to encourage future research in this area of study.

1.4 Thesis Contributions

By answering the main research question and the sub-research questions, we will add the various conceptual and technical contributions to the scientific body of knowledge.

1.4.1 Conceptual Contributions

1. We propose a design for a serverless workload processing model with carbon emissions implementation for scheduling and execution of trivial and complex tasks. The model is a close-to-reality visualization of a datacenter and the processes involved in the scheduling of a task in a serverless sustainable manner.

2. We propose an abstract and simplified serverless workload processing model that is consistent with the principles of discrete event simulation. This model provides an insight into the components of the discrete event simulator, OpenDC, that were added or modified such that the simulator would support the model designed in **RQ1**.

3. We provide a detailed analysis of the performance-carbon intensity tradeoff in the context of spatial shifting. In our analysis, we highlight the cases in which spatial shifting is the most optimal, and we discuss possible limitations and their solutions.

4. We provide an analysis of the changes in carbon emissions when spatial shifting is combined with task delays and a failure model. We highlight the benefits and the tradeoffs of performing spatial shifting while considering datacenter's limitations, such as host failures and the costs of moving one task from one datacenter to another.

1.4.2 Technical Contributions

1. We provide an implementation of spatial shifting in a discrete event simulator, namely OpenDC. Now, in a simulation, the scheduler can decide the best host for a task by considering the task's hardware requirements and the carbon intensities of different datacenters.

2. We provide an experimental setup in which experiments can be performed in a serverless and non-serverless manner and explored in depth. Our setup is customizable, allowing the user to choose whether the scheduler should run in a serverless or non-serverless manner.

3. We propose unique experiments that explore the spatial shifting in combination with carbon forecasting, task delays, and failure models. Our experiments will simulate close-to-reality situations such as host failure or the cost of moving a task from one datacenter to another, and will constitute a base for future research regarding the limitations of spatial shifting in datacenters.

1.5 Plagiarism Declaration

I declare that this is my own work and that no other sources were used besides the ones in the References section. Moreover, I declare that LLMs like ChatGPT or Gemini were not used as sources of inspiration or for writing text.

1.6 Thesis Structure

To guide the reader through the contents of this work, we provide a small overview of each chapter included in this paper. In Chapters 2 and 6, we discuss the most relevant background information and related work for this project, while Chapters 3, 4, and 5 are reserved for answering and discussing RQ1, RQ2, and RQ3, respectively, in more detail. Each chapter provides, at the beginning, a summary of the contents discussed in it.

A summary of the entire paper can be found in Chapter 7, in which the highlights of each research question are presented, and the most important findings are briefly explained.

2

Background

In this section, we provide an introduction to the main concepts and techniques used during this project. We start by discussing the importance of simulation and how a simulation is conducted, and then we offer a background on the serverless paradigm and on load shifting techniques.

2.1 OpenDC and General Datacenter Simulators

Nowadays, we live in a highly digitalized society in which every action that we perform on a device generates data. In order to process, analyze, and link large amounts of data, we need datacenters. Because of this, datacenters are considered to be "the informational backbone of the increasingly digitalized world" (3). The increase in the amount of data comes with an increase in datacenter demands, resulting in a larger datacenter energy consumption. The energy consumption of a datacenter not only affects the carbon footprint (20), but also the quality of services for the end-users in the datacenter operation (21). As a result, the scientific community needs to find new ways of managing the datacenter resources. However, experimenting directly on real datacenter infrastructure is often impractical due to high costs, operational risks, and limited scalability. This is where datacenter simulators become essential. A simulator can run experiments with millions of jobs and machines in a matter of minutes or hours, offering a faster and less resource-consuming way of exploring the behavior of datacenters in different scenarios (15). Thanks to those benefits, the scientific community built many high-quality datacenter simulators (22, 23, 24, 25).

In this project, we use OpenDC, a discrete event simulator, which integrates serverless executions and "models all the major operational layers of typical clouds, from datacenter infrastructure and virtualization, to resource management and scheduling" (15). The

difference between a *discrete-event simulator* (26) and other types of simulators (e.g., continuous simulators) is that in a discrete-event simulator, an operation of the system is represented as a sequence of events over a certain period of time, and between events, no changes in the system occur.

In OpenDC, each task in a workload is composed of multiple fragments, each fragment having a specified duration and hardware configuration (27). A workload runs on a physical datacenter infrastructure, which is represented as a set of clusters, each cluster having multiple hosts. On a host, one or multiple tasks can be run depending on the available resources of that host. In order to keep track of the resources and how they are distributed over multiple workloads, OpenDC implements a *Resource Manager*. The resource manager can be configurable and supports different *allocation policies* based on which it allocates the necessary resources to each task in a workload. OpenDC offers the possibility to model cloud datacenters and simulate the behavior of a model in combination with different phenomena, such as failures or performance interferences. OpenDC supports *Failure Models*, which indicate how often there is a failure of a datacenter, how many hosts are affected by a failure, and the duration of each failure. Failure models are essential in analyzing the reliability and performance of a datacenter. Moreover, OpenDC can simulate and estimate the carbon emissions of a datacenter based on a carbon forecast.

2.2 Serverless Paradigm

The *Serverless Paradigm* is an emerging concept in the scientific community, which is defined as an application deployment architecture that completely hides server management from the tenants (6). This paradigm is usually described by three characteristics: (i) serverless computing provides a level of abstraction that hides the servers and the way they are operated, (ii) the tenants pay for the service only during the time their applications are running, and they are not charged for idle time, and (iii) serverless computing allocates resources dynamically and ensures their scalability and availability (6, 8, 9). Moreover, it is considered that in serverless computing, the functions are event-driven, automatically scaling in response to different events such as user requests (28).

To understand serverless computing at a more intuitive level, it can be associated with programming in a higher-level language, while serverful computing would be programming in Assembly (29). In an Assembly language, to calculate a simple sum, we need to perform a number of steps such as selecting registers, loading the values into those registers, performing the arithmetic, and storing the result. In contrast, in a higher-level language,

to perform the same sum, we only need some variables and the right operator to use, and we will get the right result without knowing any of the underlying processes. Similarly, in serverless computing, a tenant benefits from the resources of a server without needing to know how they are internally managed.

2.3 Load Shifting

Load shifting represents a group of operational techniques that targets carbon emission reduction of datacenters by moving workloads across time and locations when and where carbon intensities are low (13). Out of these techniques, spatial and temporal shifting are the ones we mostly explored in the context of carbon reduction of datacenters. The carbon intensity represents the amount of greenhouse gases released by the electrical grid into the atmosphere per unit of energy used/generated (5).

Spatial Shifting

Spatial workload shifting can help reduce carbon emissions by exploiting variations in carbon intensity across geographic regions. Specifically, during the scheduling phase, the carbon intensity levels of datacenters in different locations are evaluated, and the task is assigned to the datacenter with the lowest associated carbon footprint (30).

While the underlying concept of this approach is relatively straightforward, there are limitations and tradeoffs that need to be considered. One limitation consists of the amount of data that needs to be moved with the task in order to be able to be executed in a different datacenter (5). Moving a large amount of data from one datacenter to another might generate transfer costs, which can have more impact on the overall carbon emissions as extra resources are allocated for this process. One tradeoff that also needs to be considered is the performance-emissions tradeoff. Relying solely on carbon intensity as the basis for task migration may lead to unintended environmental consequences. For example, executing a task in a low-carbon region could result in a significantly longer runtime, ultimately causing greater total carbon emissions than if the same task was executed in a region with higher carbon intensity but for a shorter amount of time (13, 31, 32).

Temporal Shifting

In contrast to spatial shifting, temporal shifting reduces carbon emissions by exploiting variations in carbon intensities across time periods, without the need to move a task to another datacenter. When a scheduling request is made, a temporal shifting approach

will consider the carbon intensity at the time of the scheduling and, based on it, the scheduler will decide whether it is more carbon efficient to schedule the task immediately or to delay it. There are multiple ways in which temporal shifting is implemented, but the most straightforward one is based on a carbon threshold. More precisely, based on a fixed threshold, a task is delayed if the carbon intensity is above the threshold until the carbon level gets to a value below the threshold (33, 34).

Although temporal shifting does not have to take into account possible transfer costs, this approach still needs to consider the performance-emissions tradeoff. If a task is delayed, then there is the possibility that it will be assigned to a host with fewer resources, resulting in a longer execution in a low-carbon period. This scenario can have a greater impact on the environment than if the task was run in a high-carbon period but for a short amount of time (13, 31, 32).

3

Design

In this section, we discuss the design of our serverless sustainable workload processing model. First, we discuss the design requirements and a basic model that considers serverless aspects. Then we describe a more complex model in which operational techniques such as temporal and spatial shifting are integrated in order to highlight the sustainability component of our model.

3.1 Design Requirements

In this section, we present the functional and non-functional requirements that our model needs to fulfill in order to align with *The AtLarge Design Process* (17). These requirements define the core functionality of our model and act as criteria for evaluating its performance.

3.1.1 Functional Requirements

FR1 The model should be able to handle any kind of workload, no matter the size of the workload or the type of tasks.

FR2 Tasks should be able to be scheduled on hosts based on a selected policy.

FR3 The model should be able to operate multiple datacenters at the same time.

FR4 In the execution phase, a task should be able to be interrupted based on the carbon intensities.

FR5 The model should be able to decide if a task should be delayed or not based on the carbon intensity at the time of the scheduling request.

FR6 The model should consider the performance-carbon emission tradeoff in the scheduling phase.

3.1.2 Non-Functional Requirements

NFR1 The model should be able to make decisions about at least 2 datacenters at the same time.

NFR2 The model should minimize carbon emissions by selecting low-emission regions for task execution whenever available.

NFR3 The model should log any task-related information, such as the status of the task and the host it is run on, during the entire workload execution.

3.2 Basic Serverless-Workload Processing Model

In Figure 3.1 we present a first model of a datacenter using the serverless paradigm for workload processing. The main goal of this model is to represent a very basic scheduling process in which task are scheduled in a serverless manner by taking into consideration the hosts, the time each host becomes available to start the task, the expected execution time if a task is run on a certain host and the hardware requirements of the task. We first start with a walk-through of our model, and then we discuss the serverless implications of this basic model.

3.2.1 Model Description

To explain the model, we traverse it from the point of a task submitted by a user. Once the request is made, the task goes through a *Ranking* phase. Using the size of the task, the *Ranking Policy* **(2)** places the task into a *Task List* **(3)** by comparing it to the size of the other tasks. A *Task List* is a list of tasks in a datacenter that need to be scheduled and that is ordered based on the priority of the tasks. An example of a *Ranking Policy* can be a policy that prioritizes small tasks by placing them towards the top of the *Task List*. This ranking process is based on the assumption that it is more efficient to schedule

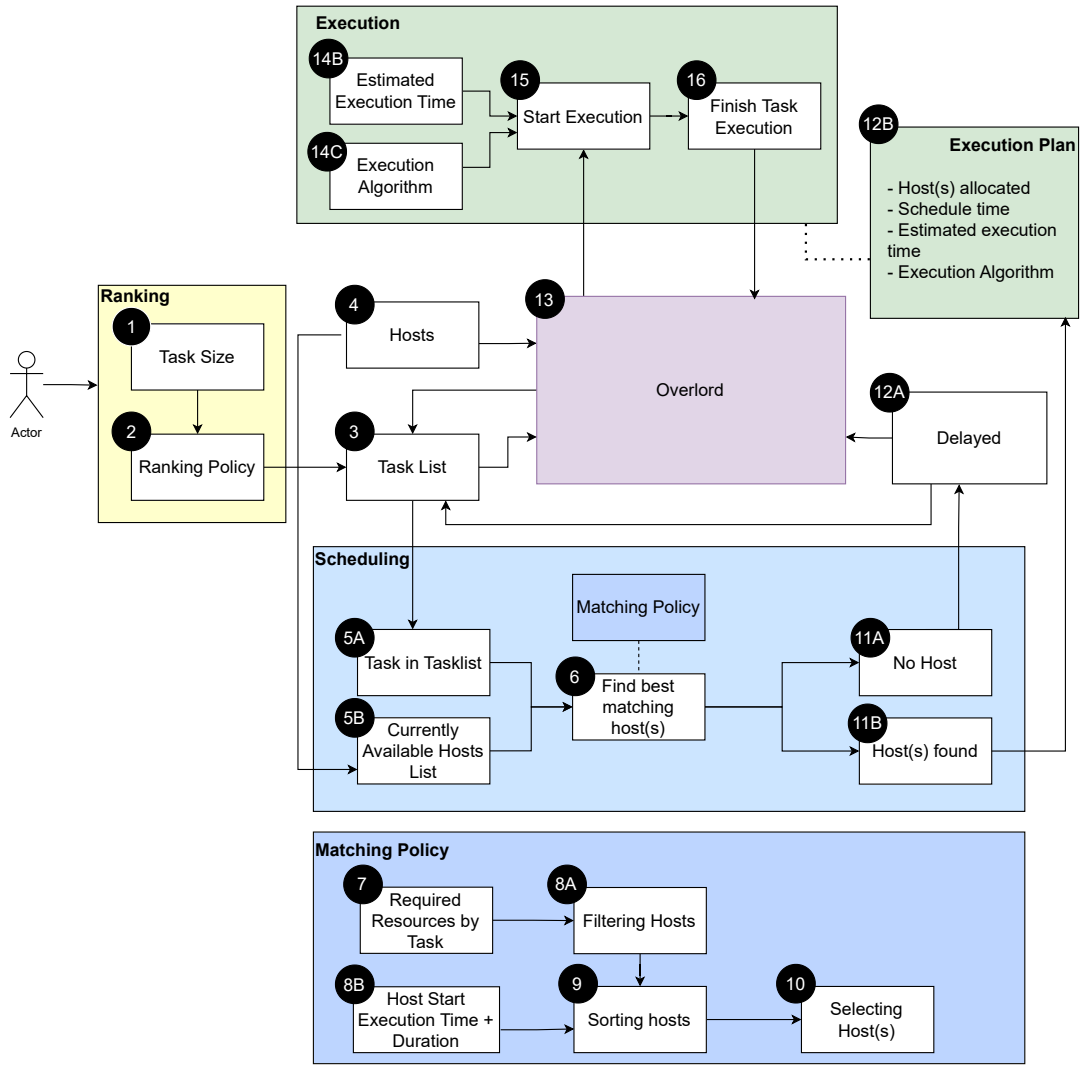

Figure 3.1: Serverless Workload Processing Without Sustainability Concerns

smaller tasks first and then deal with the larger ones. Once a new task is included in the Task list, the *Overlord* **(13)** is informed, and the *Scheduler* is triggered. The *Scheduler* takes the task at the top of the *Task List* and tries to find the best host(s) to execute the task on.

In our design, the *Hosts* box **(4)** represents a list of all hosts in the datacenter that can be used to execute tasks. For each host, the unique ID and the time the host is expected to become available are stored. The availability of the hosts is essential in the process of

Finding the Best Matching Host(s) **(6)** for a task, as it reduces the computational power by checking only the currently available hosts and not going over all hosts in the datacenter. When the *Scheduler* is triggered, a list of all hosts currently available **(5B)** is computed. To be more precise, this list includes all hosts that have no task scheduled at the time of the scheduling request or occupied hosts that are expected to finish execution soon. The scheduler takes a task from the Task List, and it tries to find the best matching host(s) for that specific task based on a *Matching Policy*. If no host is found among the available ones, then the task is delayed **(11A)**and the *Overlord* **(13)** is informed not to remove the task from the *Task List*. If at least one host is found **(11B)**, then an *Execution Plan* **(12B)** is computed and sent to the *Overlord*.

Matching Policy

In this model, the role of a *Matching Policy* is to find the best host(s) based on the required hardware configuration of the task (CPUs, GPUs, computational power, and memory), the expected start time on each host, and the estimated execution time if the task is run on each host. The first step performed by the *Matching Policy* is to filter the hosts **(8A)** from the *Currently Available Hosts List* based on the resources needed by the task **(7)**. If a host does not fulfill the hardware requirements of the task, then it is removed from the list. The second step is to sort the remaining hosts **(9)** based on the estimated start time of the task execution and the duration of the execution. During host sorting, there is a tradeoff that needs to be taken into account, which considers the time when a host becomes available and the duration of the task if the task is run on a specific host. This tradeoff can be described by the following question: *Should an available host be chosen for a task even though the execution time might be longer than if we pick another host that becomes available after some time t?*. To solve this problem, during implementation, we use a sorting system in which a host is deducted points based on the difference between the estimated start time and the current time, and based on the task execution time on that host. The fewer points a host has, the lower it will be placed in the final hosts list. Thanks to the serverless approach of host selection, we traverse this tradeoff and find the best matching host **(10)** for a task.

Overlord and Execution Plan

The *Overlord* **(13)** structure in Figure 3.1 keeps track of all the changes that can happen during the scheduling and execution processes. The Overlord is responsible for keeping track of the states of the hosts, the number of tasks in the task list, and the delayed tasks,

as well as for triggering the Scheduler once the execution of a task is finalized. Moreover, the Overlord stores the Execution Plan **(12B)** of each task and monitors the execution based on it. The *Execution Plan* is computed if a host is found for a certain task. The process of creating such a plan integrates the execution particularities of a task that were defined during the scheduling phase, such as the execution time or the host on which to run the task. When the time comes for a task to be executed **(15)**, the *Overlord* will guide the execution based on the Execution Plan.

3.2.2 Serverless Implication

The serverless paradigm is integrated in the scheduling part of our basic model. Compared to a "traditional" scheduling process in which a task enters the scheduling phase once it is requested to be executed, and it is matched to the first host that satisfies the hardware requirements, our scheduler is more complex, and it makes smarter decisions by considering different factors. The scheduler in Figure 3.1 assigns a task to a host by considering not only the hardware requirements of the task, but also aspects regarding the host, such as the time a host becomes available and the execution time if the task is executed on a certain host. Moreover, the scheduler can decide to delay a task if it considers that, performance-wise, the task should be scheduled on a host that becomes available later than on a host that is immediately available.

The serverless paradigm is also integrated into the process of ranking a task once an execution request is made. The *Ranking Policy* considers the size of the new task, and it includes the task in the *Task List* such that it maintains the sorted nature of the list. The policy places small tasks towards the top of the list based on the assumption that executing small tasks first is more efficient than executing large tasks first. The problem that executing large tasks first brings all other tasks to be delayed for a long amount of time until the large tasks finish, raising the risk of increasing the volume of tasks waiting to be scheduled. This will lead to higher energy consumption and even to the starvation phenomenon (very small tasks might never be scheduled).

3.3 Carbon-Aware Serverless Workload Processing Model

In this section, we present a more complex version of the serverless model depicted in Figure 3.1 in which the datacenter's carbon intensity is only considered at the time of task scheduling. Moreover, we introduce the concept of *load shifting* and how it is integrated in the serverless model described in the previous section.

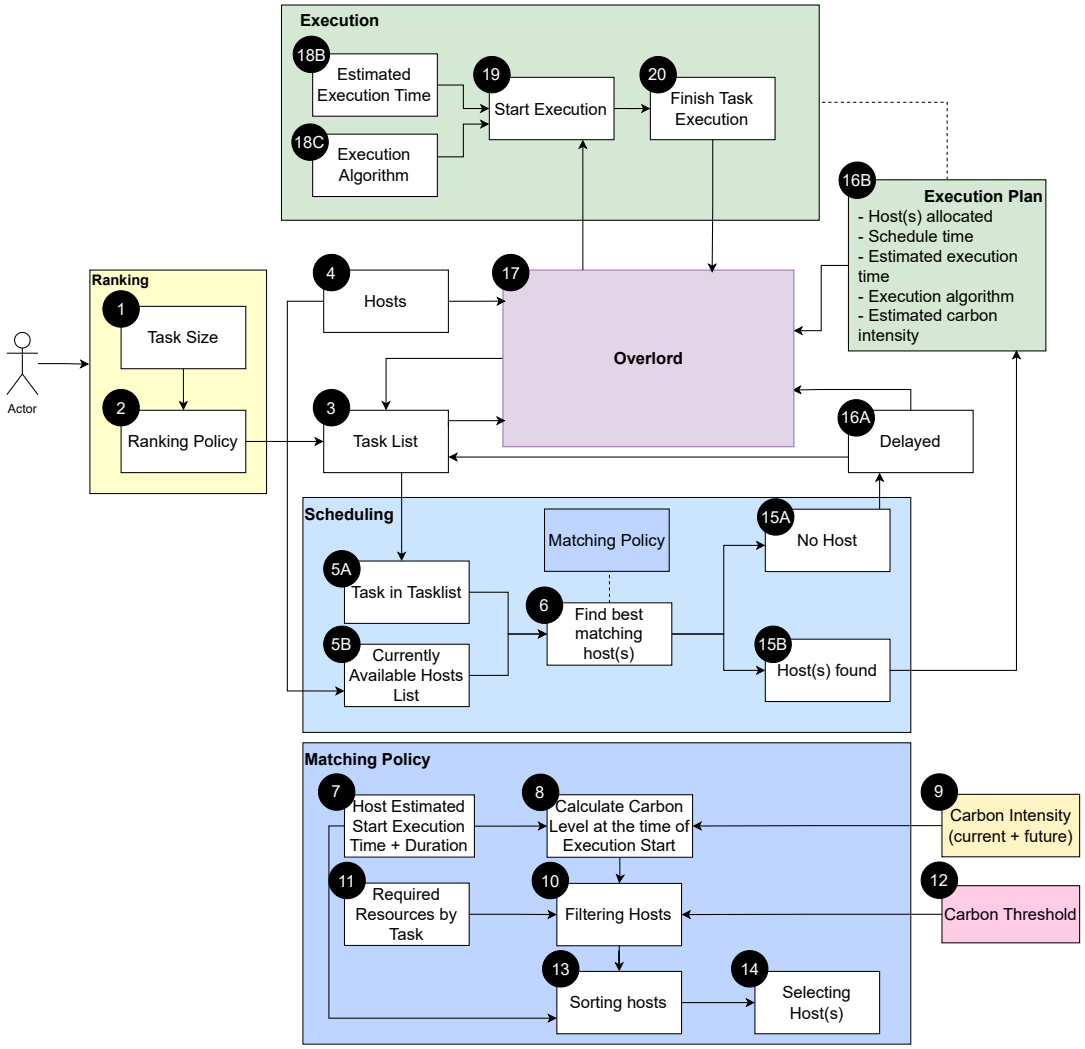

Figure 3.2: Serverless Workload Processing With Carbon Considerations at The Time of Scheduling

3.3.1 Model Description

In this section, we present the main components of the previous model that needed to be modified such that the scheduling phase is more carbon-aware.

The main difference between Figure 3.1 and Figure 3.2 is the *Matching Policy*. In the previous section, we described how hosts are filtered and sorted based on the required resources by the task, the estimated execution time, and the duration of the task. As this approach adds a level of optimization with respect to the performance of a task and

the management of resources, it does not yet consider the impact that the scheduling of a task can have on the environment. In order to perform a more environmentally friendly scheduling, we consider the carbon intensity **(9)** at the time a host becomes available, and we filter the hosts based on a carbon threshold **(12)**. To be more precise, if the carbon intensity when a host becomes available is larger than the carbon threshold, then we remove that host from the list of potential hosts. Otherwise, if the carbon level is below the threshold, then the host is considered a potential candidate for the task.

3.3.2 Load Shifting Implications

The processes described in this section are commonly known as load shifting, which implies that a task should be scheduled only in low-intensity carbon time frames.

In summary, the carbon intensity represents the amount of greenhouse gases released by the electrical grid into the atmosphere per unit of energy used/generated (5). The carbon intensity is directly influenced by the type of energy source used for electricity production and indirectly by the weather forecast, the time of the day, and the climate of the region where the datacenter is (14). In this work, we will focus on the carbon intensity as a parameter given by a forecast.

The load shifting methods consider the tradeoff between the performance of a task and the carbon intensity at the starting time of the task execution. More precisely, the performance-carbon intensity tradeoff represents the decision regarding whether to allocate a host which will execute a task in a short amount of time but in a high carbon intensity time frame for a task, or to pick a host that will execute the task in a larger amount of time, but will benefit from a low carbon time frame. There are two load shifting methods that are supported by our model. One of them is *spatial shifting*, which considers hosts from multiple datacenters and schedules a task on a host from the best carbon intensity region. The other one is *temporal shifting*, which, based on a carbon forecast, decides whether it is more carbon efficient to schedule a task immediately or to delay it until the carbon intensity of the dataceter improves.

In the model from Figure 3.2 we filter the hosts according to the required resources by a task **(11)** and the carbon level at the time when the execution of the task is supposed to start **(8)** and remove all hosts that do not have enough resources and that have a carbon intensity greater than the Carbon Threshold **(12)**. To sort the hosts, we consider in the implementation section of this paper a point system to rank the hosts, which considers the duration of the task and the carbon intensity at the start of the execution.

3.4 Carbon-Aware Serverless Workload Processing Model with Interrupts

In this section, we describe the final version of our model, which considers both the carbon intensity at the execution start time of a task and the carbon intensity during the execution of a task. We also introduce the concept of task interruption to improve the overall carbon emissions. Because of this, we use the carbon intensity during the execution of a task to decide whether a task should be interrupted at a certain point in the execution phase.

3.4.1 Model Description

Figure 3.3 presents the Serverless Workload Processing Model in which interrupting the task during execution is included. The main difference from the previous model in Figure 3.2 is the *Carbon Level Check* box added during the execution process. The *Carbon Level Check* is modeled as a continuous process that happens during the execution of a task. More precisely, at constant intervals of time during the execution of a task, the carbon intensity is checked and compared with the carbon threshold that we established. If the carbon level is below the threshold **(12)**, then the execution continues until the next *Carbon Level Check*. Otherwise, the task is interrupted until the carbon intensity reaches a value below the threshold.

Important to mention is that we assume that the workloads we use are interruptible, meaning that once a task is interrupted, it can be continued after a certain amount of time without the need to start the execution from scratch. In order to ensure this property, we create a *Interruption policy*, which will record the stage of completion of the task, the execution algorithm, the way the graph is partitioned, and the allocated host. Once a task is interrupted, this policy is responsible for informing the *Overlord* about the nature of the task (i.e, interrupted). The Overlord puts the task back on the Task List **(3)** until the carbon intensity drops below the carbon threshold **(12)**. When this happens, the *Task List* informs the Overlord that an interrupted task can be resumed, and based on the task specifications, the Overlord signals how the task execution should continue.

3.4.2 Task Interruption Implications

The benefit of using task interruption is the considerable reduction of carbon emissions. Previous work demonstrates how load shifting (temporal and spatial) of tasks in time frames with a small carbon intensity level can reduce up to 61.7% the carbon emissions worldwide(5, 13, 14). Taking this into account, we combine spatial and temporal shifting

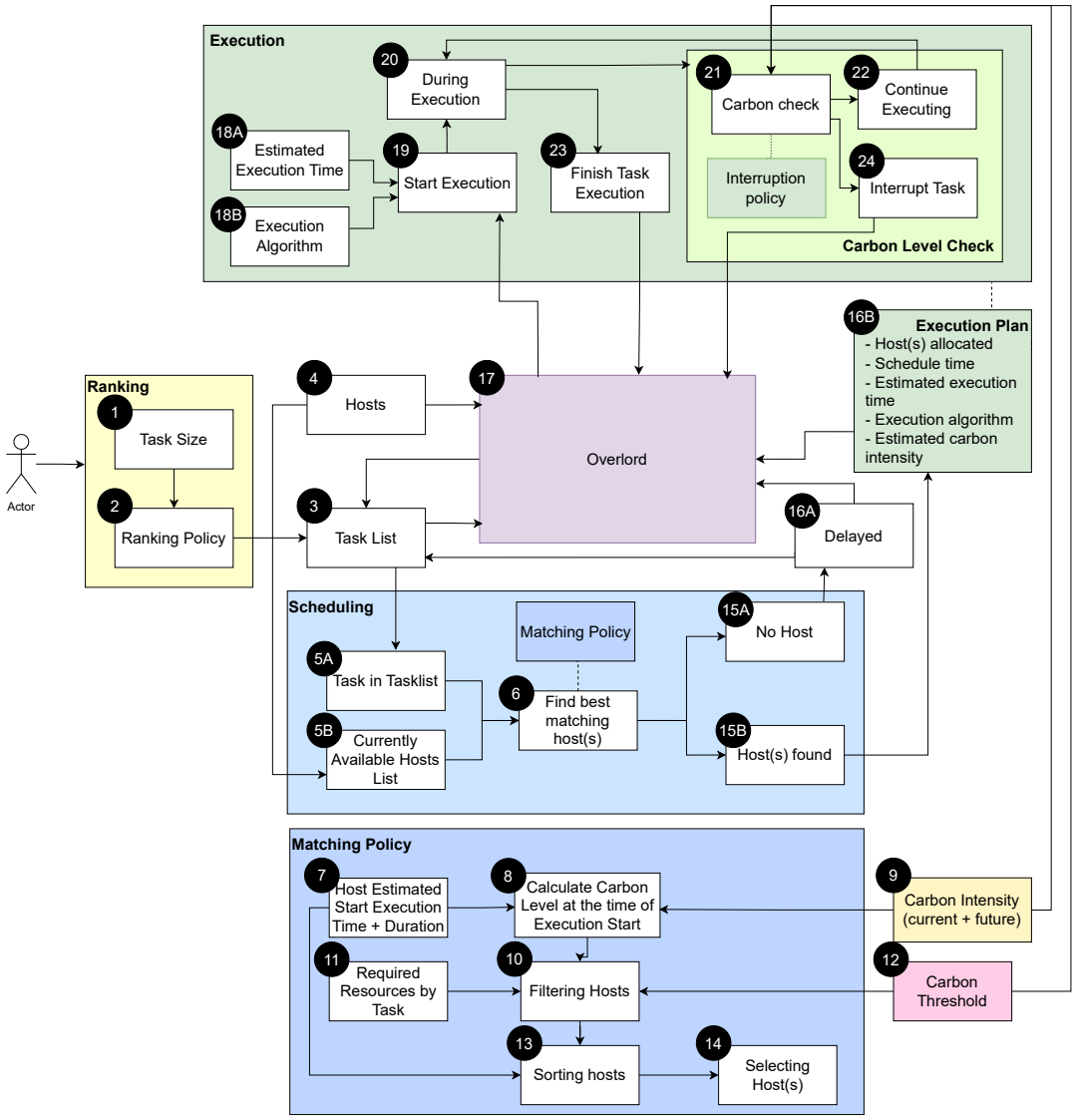

Figure 3.3: Serverless Workload Processing With Interrupts During Execution of a Task

in the model presented in Figure 3.3 to reduce the carbon emissions of workload processing as much as possible.

We combine temporal and spatial shifting in the scheduling phase in order to find the best host for a task. We consider hosts from multiple datacenters, and we sort them based on the carbon intensity and the time each host becomes available. If a host from a different datacenter than the one the task originates from is chosen, then we perform spatial shifting

by moving the task to the new datacenter and executing the task there. If a host is not immediately available, but is the best match for a task, then the task will be delayed until the host becomes available. Both methods of load shifting at the scheduling phase ensure the best carbon intensities at the start of the execution phase of a task.

During the execution phase of a task, temporal shifting is considered in the form of interrupts. If at a certain timestamp, the carbon intensity of a datacenter exceeds the predefined threshold **(12)**, then the execution of the task will be stopped until the carbon intensity drops to a value below the threshold. This form of temporal shifting ensures the most optimal carbon intensities during the entire execution of the task.

4

Implementation

In this chapter, we present the implementation of our serverless sustainable workload processing model, as described in Section 3.4, within a discrete-event simulator, OpenDC (15). The goal of this chapter is to answer **RQ2** and to present the main implementation decisions made during this process. The chapter will start with an overview of the OpenDC components that were modified or added in order to integrate the serverless model into the discrete-event simulator, and it will end with a discussion regarding the main implementation decisions and possible alternative implementations.

Because the model shown in Figure 3.2 is a close-to-reality representation of the processes that take place in a real datacenter, we create a more abstract and simplified serverless workload processing model that is consistent with the principles of discrete event simulation and with the OpenDC infrastructure. Figure 4.2 depicts such a model, in which we can see the modified components of OpenDC together with the newly added components and the links between them. For a better understanding of the logic behind those components, we first describe the process of task scheduling in OpenDC before we implemented our model, and then we explain how the implementation of the model changed the code base of OpenDC.

4.1 Key Concepts in OpenDC Task Scheduling

In this section, we provide an overview of the most important concepts that are applied to task scheduling in OpenDC.

Host A machine with a specific hardware configuration on which a task or multiple tasks can be scheduled. Each host has a unique ID and link to the power source of the

datacenter it belongs to.

Allocation Policy An allocation policy specifies the way a task is executed, the time the execution phase is started, and the host on which the task is run. There are two allocation policies available in OpenDC, namely *Filter Policy* and the *TimeShift Policy*. In this work, we only work with the *Filter Policy*.

Filter Policy In this policy, a host is chosen for a task after a list of filters and weighters is applied.

Host Filter A requirement that a host needs to fulfill in order to be eligible for a task. If a host does not fulfill a specific task requirement (e.g., available RAM or CPU capacity), it is removed from the list of possible candidates for that task. Filters are defined in the JSON objects of the experiment file.

Host Weighter After the hosts are filtered, the hosts are ranked based on one or multiple weighters (e.g., carbon intensity). Once all weighters are applied, the host with the highest weight is assigned to the task. Weighters are defined in the JSON objects of the experiment file.

4.2 Task Scheduling in OpenDC

Figure 4.1 displays the main components of OpenDC that were involved in the process of task scheduling before the serverless sustainable workload processing model was implemented. In the simulation of the scheduling process, each host is associated with a *Host View* **(6)**, a Java class in OpenDC, used by the scheduler to keep track of the order of the hosts that can be used to execute tasks. Because the *Host View* class has mostly a monitoring purpose, all the characteristics of a host (e.g., the name of the host, the number of CPU cores, the CPU capacity, and the memory size) are included in the *SimHost* **(7)** class. A *SimHost* **(7)** instance is created through the execution of the *Host Provisioning Step* **(8)** class. Similarly, a task is represented in OpenDC by an instance of the *Service Task* **(2)** class. This class contains all the important information about the task, such as the unique identifier and the hardware requirements. The process of scheduling a task is performed in *Filter Scheduler* **(3)**, where a host is picked for a task from a list of *Host*

View instances. In this version of OpenDC, one task can be assigned to only one host, so each *SimHost* is associated with one *Host View* instance.

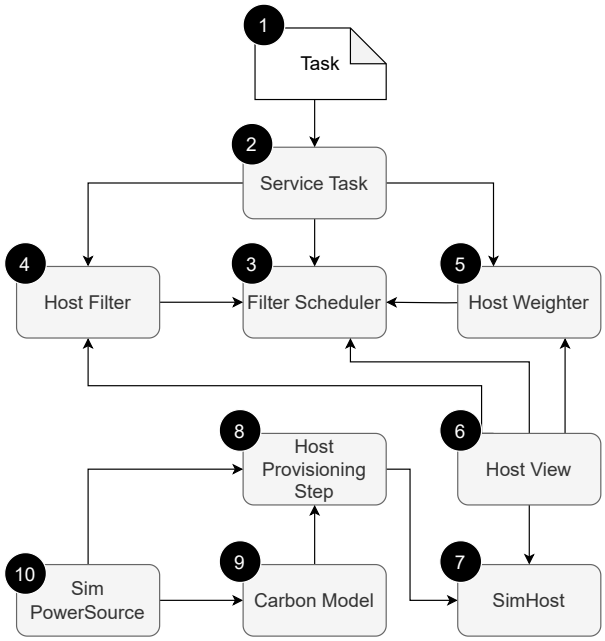

Figure 4.1: Visualization of the OpenDC components before the implementation of our model.

In OpenDC, one datacenter is identified by only one power source, which estimates the power consumption based on the CPU usage of the datacenter. Because power consumption directly influences the carbon intensity and the carbon emissions of a datacenter, this information is included in the *SimPowerSource* **(10)** class. Moreover, one power source is associated with one cluster, and each cluster contains a number of hosts on which tasks can be executed. The *SimPowerSource* contains a direct link to the *Carbon Model* class **(9)**, which is responsible for providing and updating the carbon intensity of a power source based on a carbon forecast.

In OpenDC, a task is assigned to a host through an **Allocation Policy**. OpenDC supports multiple allocation policies, but the one we use in this work is called **Filter Policy**. In this policy, a series of filters **(4)** and weighters **(5)** are applied to the list of hosts in order to find the best host for a task.

When a *Service Task* **(2)** needs to be scheduled, in the *Filter Scheduler* **(3)** class, the list of *Host Views* will be filtered based on the task hardware requirements specified in the **Allocation Policy** (e.g., the amount of available RAM on the host, the CPU capacity of

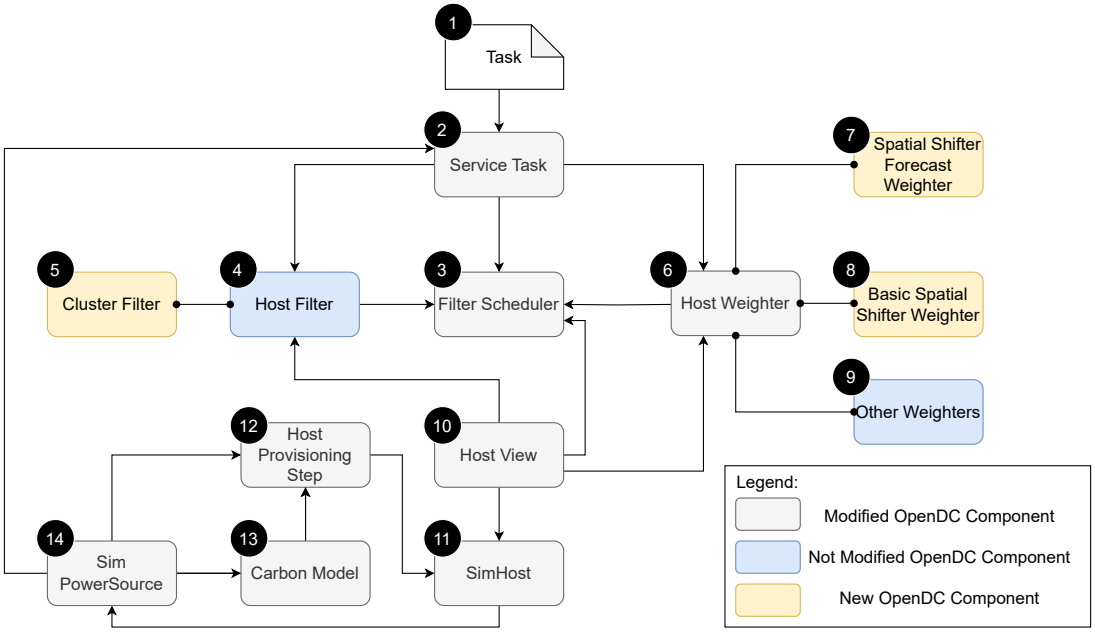

Figure 4.2: Visualization of the OpenDC components involved in the scheduling process.

the host, or the number of CPU cores). Only the hosts that pass all filters are kept in the updated list of hosts, and then they are ranked based on a series of weighters. The idea behind this ranking is that each task receives a score based on the weighters specified in the **Allocation Policy**. The hosts are sorted from the largest score to the smallest, and the host at the top of the list is assigned to the task.

4.3 Serverless Sustainable Workload Processing Model Implementation in OpenDC

Figure 4.2 displays the main components of OpenDC that were involved in the task scheduling process after the serverless processing model was implemented. In this figure, the components colored with yellow are newly added components in OpenDC, while the ones in gray are components that needed modifications in order to be compatible with our model.

In order to be able to perform spatial shifting, we needed a way in which we could access the carbon intensity of the datacenter from a *SimHost* **(11)** instance. Our solution was to create a link from the *SimHost* **(11)** to *SimPowerSource* **(14)** class through a variable that identifies the datacenter instance from which a host originates. Once this link was

made, we could also identify the carbon intensity of the datacenter at the time a task is scheduled on a host. This is an important aspect of the implementation as the carbon intensity plays an important role in our serverless-based task scheduling.

A task is represented by a *Service Task* **(2)** instance, which includes the unique identifier of a task, the duration of the task, the hardware requirements of the task, and a link to the trace workload of the task. In order to implement spatial shifting and to keep track of the shifted tasks, we included in the *Service Task* **(2)** class a variable that identifies the cluster origin of a task. As mentioned in Section 2.1, a task consists of a series of fragments, so, to perform the experiment described in Section 5.5, we also needed to include a variable that identifies the series of fragments that form a task. Additionally, the *Filter Scheduler* **(3)** class is modified such that extra fragments are added to a task.

In order to include spatial shifting in the OpenDC code base, we had to implement a *Basic Spatial Shifter Weighter* **(8)**, which ranked the hosts based on the carbon intensity at the time the scheduling process started. Each weighter has a multiplier which can be a negative or positive number. For the *Basic Spatial Shifter Weigher* **(8)**, we use a negative multiplier so that hosts with smaller carbon intensities are ranked higher than the ones with larger carbon intensities. To make our spatial shifting more complex, we implemented one more version of the spatial shifter weighter **(8)** in which the hosts are ranked based on the estimated average carbon intensity during the entire execution of the task. To make this possible, we needed to access the carbon forecast provided by the *Carbon Model* **(13)** from a *SimHost* **(11)** instance. Based on the carbon forecast, we were able to see the carbon intensities during the entire duration of the task execution.

For experimentation purposes, we created a *Cluster Filter* **(5)** which filters the hosts based on a specified cluster. As an example, if in the experiment to be run it is specified that all tasks should be run on cluster "C1", then the hosts from other clusters will always be filtered out. This was implemented in order to see the differences in carbon emissions when spatial shifting is performed or not.

4.4 Main Implementation Decisions

In this section, we present the main implementation decisions that we needed to take during the implementation stage of this project in order to integrate the serverless workload-processing model into OpenDC.

1. **Implementing spatial shifting as a weighter.** We decided to implement the spatial shifting as a weighter in order to keep a balance between performance and

carbon emissions. If this concept was implemented as a filter, there was the risk that we would remove hosts that fulfill the hardware requirements of a task just based on the carbon intensity, meaning that we would not consider at all the performance of a task, only the carbon emissions. Moreover, if all the hosts fulfilling the hardware requirements of a task were in the datacenter with worse carbon intensities, then there was the possibility that the task was never executed because no host would be eligible. As a weighter, we do not eliminate any host that fulfills the hardware requirements, and we rank them based on the carbon intensities.

2. **Linking each host directly to the power source of the datacenter.** In OpenDC, the carbon intensity of a datacenter at a certain timestamp is determined by the power source of that datacenter. Every host in a datacenter has the carbon intensity determined by the power source. In the previous implementation of OpenDC, there was no stored information about the datacenter origin of each host, so no information about the carbon intensities of each host could be retrieved. In order to solve this, we created a direct link between each host and the power source of the datacenter they belong. This link guarantees easier access to the carbon intensity of a datacenter.

3. **Using the carbon forecast for the entire task duration and not only at the time of scheduling.** In the initial implementation of spatial shifting (Figure 4.1), we ranked the hosts based on the carbon intensities at the time of task scheduling. We ran the experiment described in Section 5.3 and we observed that no benefits were obtained from our spatial shifting implementation. To solve this issue, we decided to make a new version in which the carbon intensities during the entire execution of the task are considered. This version proved to add some improvements to the carbon emissions of running a workload, as described in Section 5.4.2, so we set this version as the default implementation of spatial shifting for the follow-up experiments.

4. **Storing information about the datacenter a task originates from.** In order to analyze the effects of moving a task from one datacenter to another, we decided to add one more parameter to the *ServiceTask* class in OpenDC, which names the datacenter the task originates from. This parameter was essential for the experiment described in Section 5.5 and was mainly used to track the tasks on which spatial shifting was performed.

5

Evaluation

In this chapter, we evaluate the benefits of using spatial shifting in a carbon-aware serverless workload processing system, and we discuss the results and the limitations encountered during this process. In the evaluation phase of the project, we designed and ran three complexity-level experiments, which are classified in *Basic Experiments*, *Delay Experiment* and *Failure Experiment*, and are discussed in Sections 5.3, 5.4, 5.5, and 5.6, respectively. We first start with the *Basic Spatial Shifting Experiment*, in which a basic implementation of spatial shifting in an ideal environment is discussed. Next, in the *Delay Experiment*, we investigate the effect of transfer costs. Finally, in the *Failure Experiment*, we evaluate the impact of failures on spatial shifting.

5.1 Main Findings

In this section, we briefly present the main findings that we discovered after running all four experiments. Each finding will be discussed in more detail in the specified sections.

MF1 Our serverless workload-processing model registers better carbon emissions when we consider carbon intensities during the entire execution of a task before performing spatial shifting. In Sections 5.3 and 5.4, we present two experiments, one which performs spatial shifting based on the carbon intensities of two datacenters at the time of the scheduling, and one that considers the carbon intensities of the datacenters during the entire duration of the tasks. Using spatial shifting, 7 scenarios resulted in more than 2% less carbon emissions, while the other 17 registered between 0.01% and 2% less carbon emissions.

MF2 Spatial shifting is more efficient when datacenters with similar mean carbon intensities are considered. Datacenters with similar mean carbon intensities had high spatial shifting rates and registered better carbon improvements. This finding is described in Section 5.4.

MF3 The cost of moving a task from one datacenter to another has a significant impact on the carbon emissions. In Section 5.5, we describe how the size of transfer costs together with the differences in carbon intensities between datacenters can impact the overall carbon emissions of running a workload. Out of 24 datacenter combinations, two of them stopped registering carbon benefits from spatial shifting when small delays were applied, while 17 of them reached the 0delays were used.

MF4 The reliability of datacenters can significantly reduce the benefits of spatial shifting. In Section 5.6, we use failure models in order to see whether the scheduler should be aware of the reliability of each datacenter when it performs spatial shifting. When the datacenters had similar failure models, 9 datacenter combinations registered carbon improvements, while in the other scenario, when the datacenters had different failure models, only one combination registered improvements.

MF5 The number of carbon regions for which the carbon intensity over time matches well to use for spatial shifting is low. Out of 24 datacenter combinations, only 5 of them had very similar carbon intensities, none of them having perfectly matching carbon intensities. This finding is discussed in detail in Sections 5.3 5.4

MF6 The number of carbon regions on which spatial shifting does very little is high. 17 datacenter combinations registered carbon improvements less than 1,5% than in the case when spatial shifting is not performed. Section 5.4 describes this finding in more detail.

MF7 The impact of spatial shifting is much lower than reported in previous works. The best result that we got was 3.1% less carbon emissions than when we do not use spatial shifting, which is significantly less than reported in previous work.

5.2 Experimental Setup

All experiments were structured based on the guidelines provided in the OpenDC documentation (15). The workload trace used for this experiment considers tasks with various durations for one month (35). The topologies that have been used consider two clusters from two different geographical regions (usually one in the US and the other in the EU), both clusters being composed of hosts with similar characteristics. Most datacenters are located in the US and the EU, and their carbon intensity can vary based on the local energy mix and timezone. Therefore, we use datacenters from both regions to analyze the effects of spatial shifting over long distances. The carbon traces consider the carbon intensities of the datacenters from 2021 to 2024 (36, 37, 38, 39, 40, 41, 42, 43, 44, 45). As an allocation policy, the *Filter policy* is used, which considers the RAM and CPU capacity of the hosts for host filtering and our *Spatial Shifting Forecast Weigher* for host ranking. A detailed explanation of the structure of the experiments can be found in Appendix A.

In order to evaluate our model, we consider three different scenarios: one in which we use our model with spatial shifting, one in which we run all tasks on the first datacenter, and one in which we run all tasks only on the second datacenter. For each run, a different combination of topologies is used. We run each scenario 24 times, each run considering a different datacenter combination.

5.3 Experiment 1: Basic Spatial Shifting

In this section, we discuss the first experiment that we performed to show the benefits of our carbon-aware serverless model for workload processing. The goal of this experiment is to see the improvements in carbon emissions if we perform spatial shifting by considering the carbon intensities of the datacenter at the time of the scheduling request. For this experiment, we use the *Experimental Setup* described in Section 5.2. We first present the *Results* 5.3.1 of this experiment, and then discuss them in the *Discussion* 5.3.2 section.

5.3.1 Results

To observe the behavior of our model in the spatial-shifting scenario, we calculated the number of tasks running at a certain time, and we plotted this information together with the carbon intensities of each cluster at that time. Figure 5.1 shows the results of an experiment in which we use a cluster in the Netherlands and a cluster in California. In an ideal case, the number of tasks running on a cluster should be inversely correlated to the

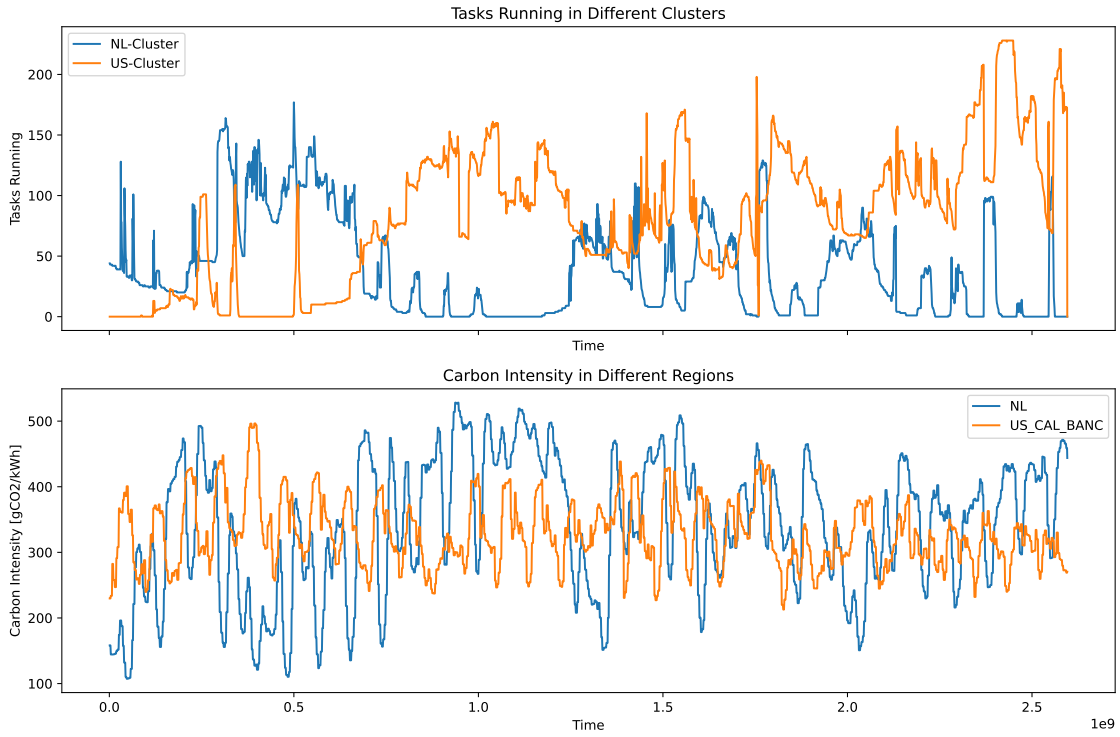

Figure 5.1: NL-US_CAL_BANC: The number of tasks running in each cluster vs the carbon intensities of each cluster at a specific moment of time.

curve of the carbon intensity of that cluster. For example, when the US_CAL_BANC cluster has a lower carbon intensity than the NL cluster, the number of tasks running on the US cluster should be larger than the number of tasks running on the NL cluster. In Table 5.1 we present the carbon emissions when the workload is run on the NL cluster, the US_CAL_BANC cluster, and when spatial shifting is used.

Although we can observe from Figure 5.1 that, in most cases, more tasks tend to be executed on the datacenter with lower carbon intensity, we can see in Table 5.1 that the

Scenario	Carbon Emissions (kg CO2)
Running all tasks on NL cluster	7719.00
Running all tasks on US_CAL_BANC cluster	7413.31
Spatial Shifting	7432.70

Table 5.1: The carbon emission results of the basic carbon emissions experiment when one datacenter is in the Netherlands and on in the US.

carbon emissions when spatial shifting is performed are worse than in the other two scenarios. This is not a result that we obtained for only the NL-US_CAL_BANC combination. Of the 24 datacenter combinations, we did not observe one combination that would be close enough to the ideal case to bring true benefits to this implementation of spatial shifting in terms of carbon emissions. In all runs, we observe that the carbon emissions in the *Spatial Shifting* scenario are higher than the minimum of the carbon emissions of the other two scenarios. The closest to ideal result was the one presented in Figure 5.1.

5.3.2 Discussion

As stated in the previous Section 5.3.1, we do not see any improvements regarding the carbon emissions of running a workload on our serverless spatial shifting model. After a long analysis of any possible factors that could influence our unexpected results, we reached the following conclusion:

- Making the task shifting decisions based solely on the current carbon intensity results in a large number of tasks being shifted incorrectly.

This conclusion is based on the fact that in the current implementation, if at the time of the scheduling, cluster C1 has a better carbon intensity than cluster C2, but after that, the carbon intensities of cluster C1 are way worse than those of C2, a long task will still be scheduled on a host of C1. This results in more energy consumption and overall increased carbon emissions. This conclusion is explored in more detail in the **Spatial Shifting with Carbon Forecast Experiment**, explained in Section 5.4.

5.3.3 Findings

The main finding during this experiment was that the impact of spatial shifting, compared to previous work, was significantly lower **(MF7)**. If Murillo et al. and Souza et al. reported carbon improvements up to 70% (5, 30), all the runs of our experiment registered negative improvements, meaning that it is more carbon efficient to run all tasks on the datacenter with the overall better carbon intensities than performing spatial shifting.

5.4 Experiment 2: Spatial Shifting with Carbon Forecast

In this section, we discuss the second experiment that we performed in order to evaluate our model. This experiment was designed as a response to the results obtained from the

previous experiment to assess the conclusion made in Section 5.3.2. Considering this, we first explain the updated approach regarding spatial shifting in Section 5.4.1, then we will show the results in Section 5.4.2, and discuss the results and possible limitations in Section 5.4.3.

5.4.1 Incorporating Carbon Forecast

Unlike the experiment in Section 5.3, when a task scheduling request is made, for each host, the mean of the carbon intensities during the entire duration of the task is computed, and based on this, the tasks are ranked. In order to find the carbon intensities of a datacenter for a certain period of time, we use a carbon forecast. As a result, we use the same setup as the one described in Section 5.2, the only difference between the first two experiments being the way the spatial shifting concept is implemented in OpenDC.

In this experiment, we introduce the spatial shifting rate notion. The spatial shifting rate represents the minimum percentage of tasks that run on one of the datacenters in the combination.

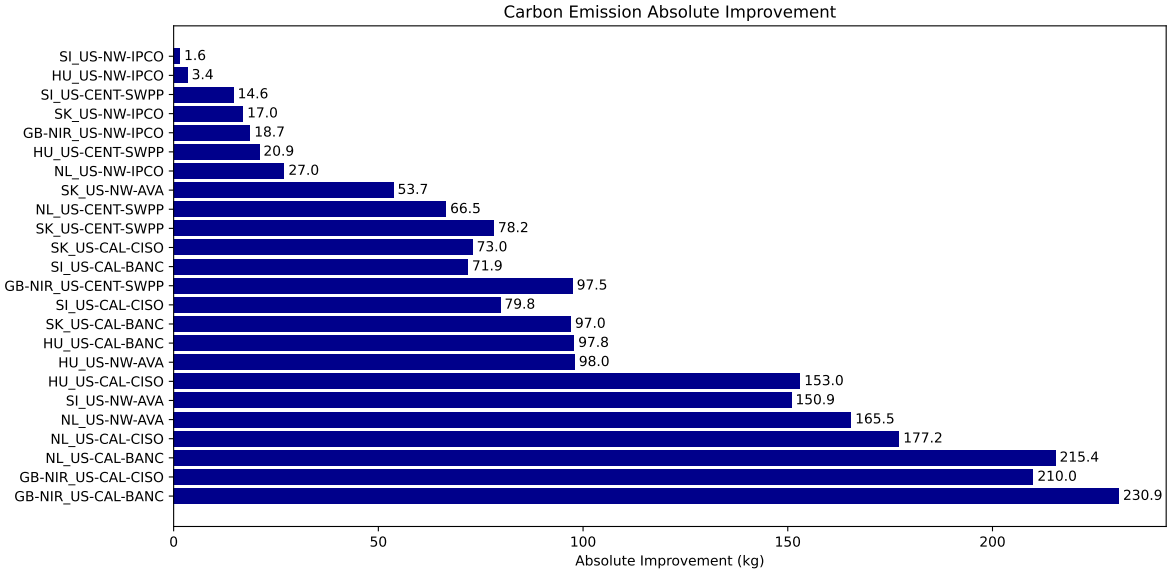

Figure 5.2: Absolute value of carbon emissions improvements when running our model.

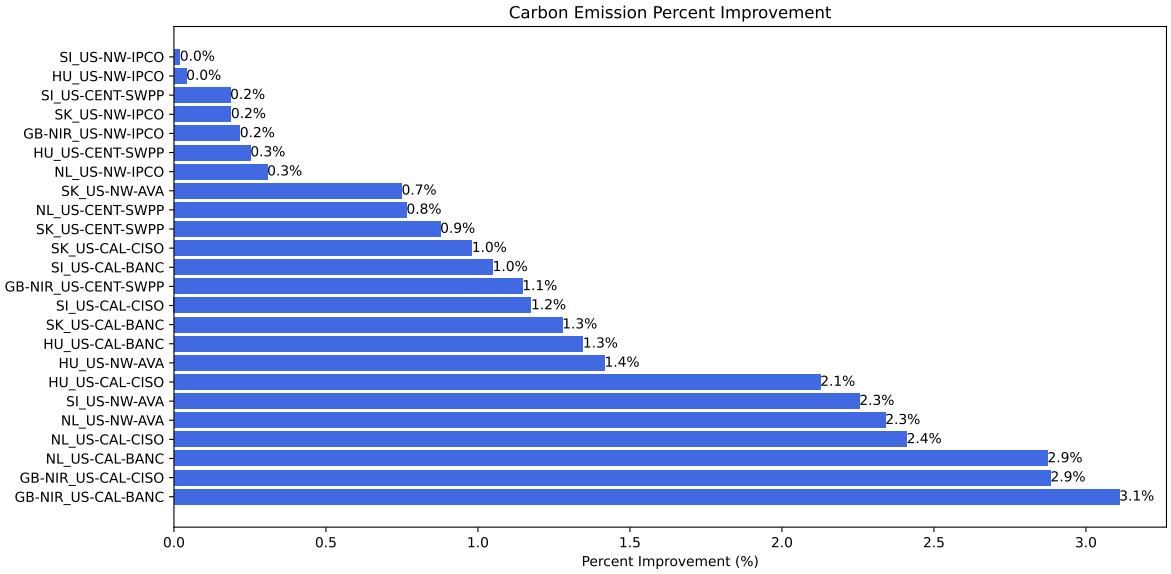

Figure 5.3: Absolute percentage of carbon emissions improvements when running our model.

5.4.2 Results

After running 24 simulations that considered the datacenter combinations from the previous experiment and the updated implementation of spatial shifting, we observed improvements in carbon emissions.

In Figure 5.2, we can see the absolute value of the carbon emissions improvements when running our model with forecast spatial shifting compared to the case when all the tasks are run on the datacenter with the best carbon emissions out of the two. In Figure 5.3, we can see the same improvements but in percentage. Of 24 runs, 15 datacenter combinations register improvements in carbon emission with a percentage of at least 1% better overall carbon. The maximum improvement percentage obtained is 3.1%, and this result was reached when two datacenters with similar carbon emissions were used, namely one datacenter in Northern Ireland and another in California.

During experimentation, we observed that, in most cases, the datacenter combinations that registered a spatial shifting rate close to 50% had better carbon improvements. In Figure 5.4, we can see the spatial shifting rate of the combination. In this figure, we can observe that the combinations with spatial shifting rates close to 50%, correspond, in most cases, to the combinations with the best carbon emissions improvements displayed in Figure 5.3. For example, the combination **GB-NIR_US-CAL-BANC** has a spatial

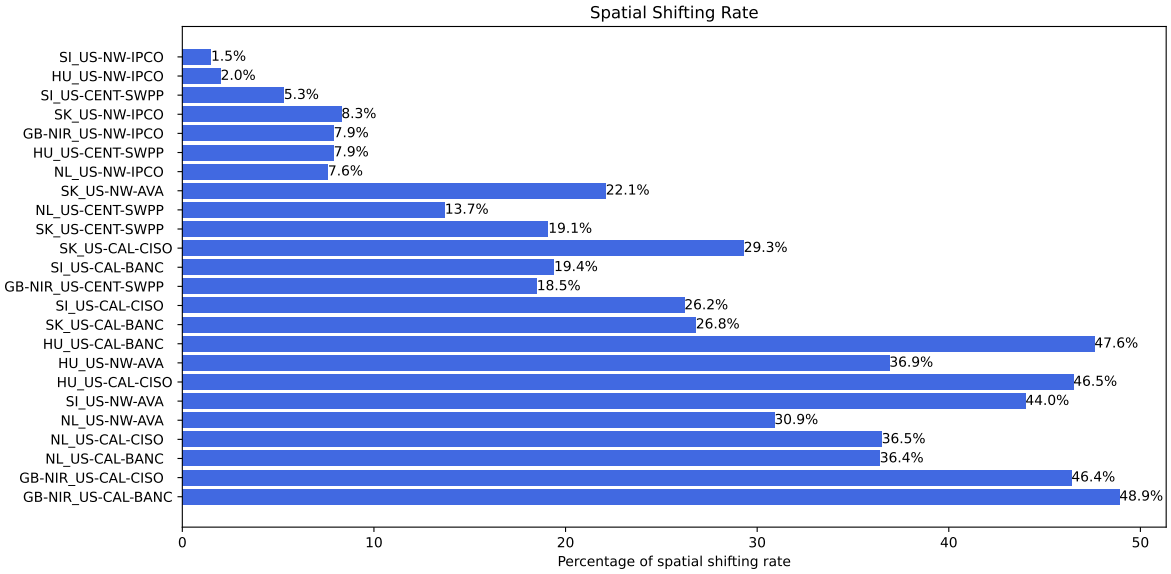

Figure 5.4: The spatial shifting rate.

shifting rate of 48.9%, and, at the same time, it registers the highest carbon emissions improvements out of all combinations. By considering those results, we can state that, in order to have noticeable carbon improvements, we need to perform spatial shifting between datacenters with similar overall carbon intensities.

5.4.3 Discussion

Compared to the results of the first experiment, we can state that the version of our model that considers the carbon intensities during the entire duration of the task brings improvements in overall carbon emissions when a workload is processed. Out of all runs and cluster combinations, the best carbon improvement was with 3.1% better overall carbon emissions than if we had scheduled all the tasks in the workload on the cluster with the best carbon intensities. We also observed that the best improvements in carbon emissions were registered in the combinations in which the percentage of tasks run on each cluster was close to 50%. This led to the conclusion that two datacenters have to have similar carbon intensities in order to have a balanced distribution of tasks per cluster.

Even though the improvements in carbon emissions were not as significant as we expected, there are some aspects and limitations that we still need to take into account:

1. Most previous papers on the benefits of spatial shifting use analytical oracle models

to determine what is the perfect way to schedule a task, while we need to make online decisions with the information we have at a certain time, which means that we do not know 100% what will happen with the tasks that are coming. Moreover, those analytical models do not consider the costs of moving a task from one datacenter to another, or any privacy regulations that might forbid the movement of a task from one region to another. All of these factors might reduce the improvements in carbon emissions that result from spatial shifts.

2. We observed that in order to have significant improvements in carbon emissions, we need to perform spatial shifting between datacenters with similar carbon intensities, so that we have a percentage of tasks run per cluster close to 50%. This is a demanding search because, in order to find the best combination of datacenters, a more in-depth analysis of all datacenters from all continents needs to be conducted. In this work, we only consider a limited number of datacenters from the US and the EU.

5.4.4 Findings

One observation that we made during this experiment is that the version of spatial shifting, when the carbon intensities during the entire execution of a task are considered, is more carbon efficient than the version in Section 5.3 **(MF1)**. Even though this version proved to be carbon-wise more efficient, only 7 out of 24 datacenter combinations registered noticeable improvements between 1.5% and 3.1%. As a result, the number of carbon regions on which spatial shifting does very little is pretty high **(MF6)**.

During the execution of the experiment, we observed that the combinations with noticeable carbon improvements were the combinations in which the datacenters had similar mean carbon intensities **(MF2)**. Moreover, when selecting datacenters for the combinations used during the experiments, we observed that the number of carbon regions for which the carbon intensity over time matches well is low **(MF5)**.

5.5 Experiment 3: Spatial Shifting with Carbon Forecast and Transfer Costs

The third experiment that we performed evaluates the tradeoff between the costs of moving a task from one datacenter to another and the benefits of performing spatial shifting. During the previous experiments, we performed spatial shifting in an ideal environment, in which the cost of moving a task is negligible. In reality, performing spatial shifting

between two datacenters comes with costs such as data transferring costs, data processing costs, or energy consumption.

The goal of our experiment is to see how much the transfer costs can be such that spatial shifting is still bringing benefits regarding carbon emissions and energy usage. To explore this, we perform different runs on the same 24 datacenter combinations used in the previous experiments, each run having a different delay for the tasks. We make a distinction between small and large tasks, and we apply different delays for each category. Based on a previous work that uses static task check-point overhead (33), we initially start with a static delay of 3000 ms for small tasks and 6000 ms for large tasks, and in each run, we exponentially increase the delay. We use milliseconds for the time unit because the duration of a task in the workload we used is represented in milliseconds. The delay of the large tasks is always double the delay of the small tasks. The way we sort the tasks based on the duration is as follows: tasks with a duration smaller than two hours are considered small tasks, while those with a duration larger than 2 hours are large tasks.

5.5.1 Task Delay Implementation

In order to perform this experiment, we had to modify the scheduler of OpenDC such that, based on the duration of the tasks, it adds a fragment with a specific delay. As previously mentioned in Section 2.1, in OpenDC, workload tasks are composed of fragments which specify the duration of the fragment, the CPU core, and memory requirements.

5.5.2 Results

Figure 5.5 shows the improvement percentage of performing spatial shifting against the delay applied for each task when we have a datacenter in the Netherlands and one in the US, CENT-SWPP. The x-axis indicates how much delay was added to small tasks compared to the base delay of 3000 ms. As an example, when 64 is represented on the x-axis, the delay applied to small tasks is 64 $\times$ 3000 ms, while for large tasks is 64 $\times$ 3000 $\times$ 2 ms. In this Figure, we also indicate the delay at which the improvement percentage of carbon emissions becomes 0. We can see that, in this case, the spatial shifting stops being beneficial when the delay is greater than 45.96 $\times$ 3000 ms. This means that, for this scenario, we need a delay at least equal to 45.96 $\times$ 3000 ms for small tasks and double for large tasks in order for spatial shifting to fail to present any carbon improvements.

We represent two more datacenter combinations in Figure 5.6 and Figure 5.7, in order to visualize the differences between two edge cases: one that reported in the previous exper-

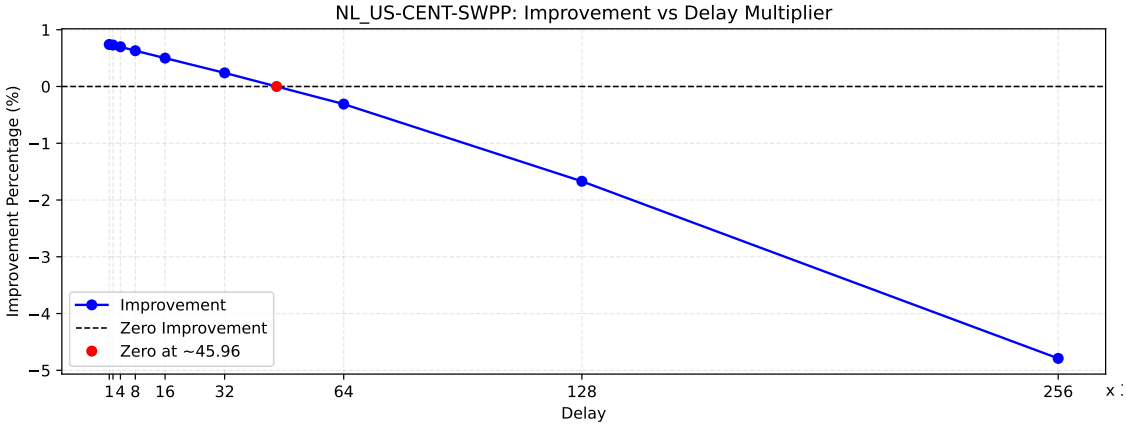

Figure 5.5: The percentage of carbon emissions improvement vs the delay multiplier in the NL_US-CENT-SWPP combination.

iment little improvements in carbon emissions and one that reported high improvements. In Figure 5.6, we can see that any delay slightly larger than 1% × 3000 ms can significantly reduce the benefits of spatial shifting, reaching a point at which it is more carbon-friendly to run all tasks of a workload on the datacenter with the smallest carbon emissions. On the other hand, in Figure 5.7, we can observe that the GB-NIR_US-CAL-BANC loses the benefits of spatial shifting when we consider large delays. This combination registers 0% carbon emissions improvement when the delay is approximately 130.83 × 3000 ms, which

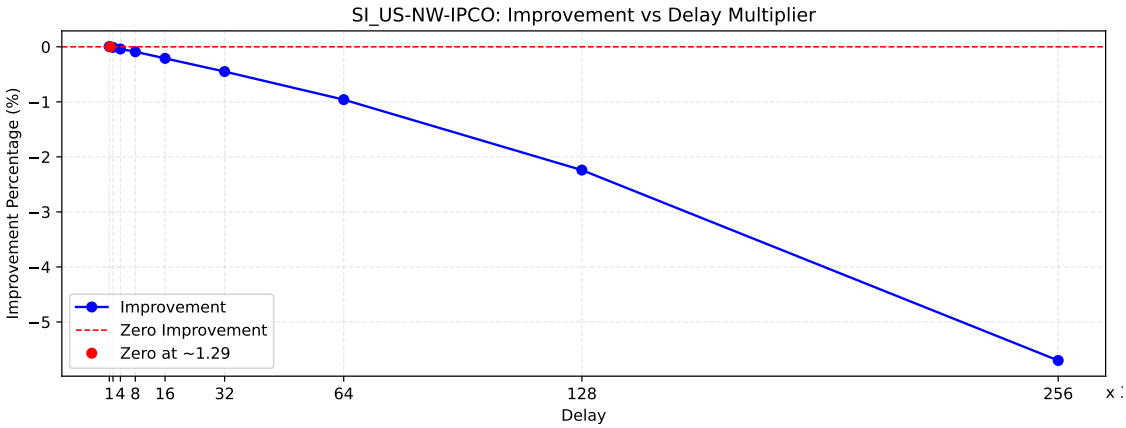

Figure 5.6: The percentage of carbon emissions improvement vs the delay multiplier in the SI_US-NW-IPCO combination.

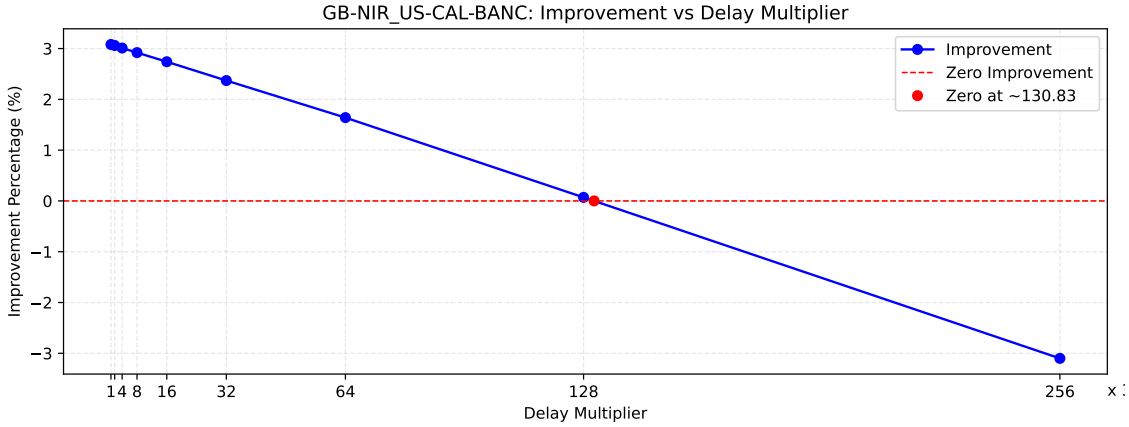

Figure 5.7: The percentage of carbon emissions improvement vs the delay multiplier in the GB-NIR_US-CAL-BANC combination.

is significantly higher than the SI_US-NW-IPCO case, indicating that, in this scenario, spatial shifting will not be effected by the costs of moving a task.

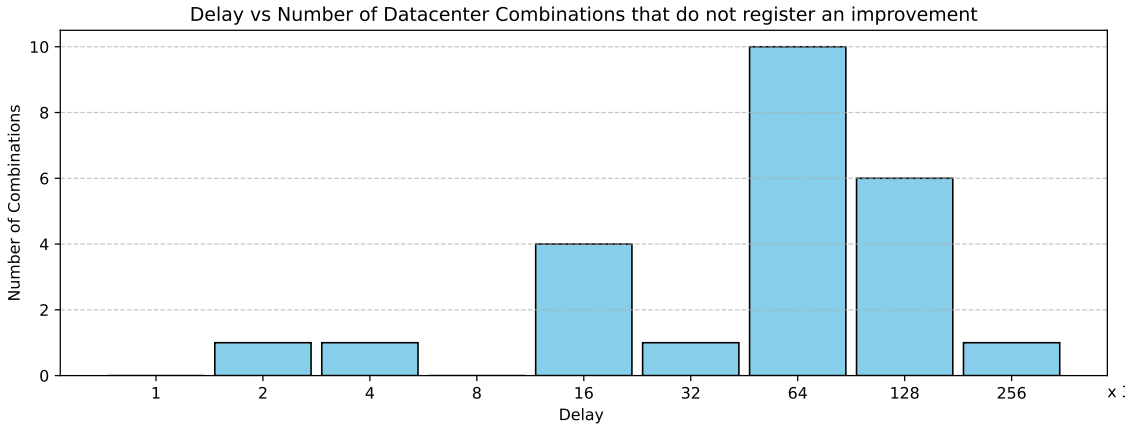

Figure 5.8: The delay multiplier vs the number of tasks that stopped registering carbon improvements when that delay was applied.

For this experiment, we ran 24 datacenter combinations in order to analyze the effects of transfer costs on the spatial shifting. Similarly to the previous three combinations that we discussed at the beginning of this section, we wanted to find for each combination the delay around which our serverless model with spatial shifting becomes inefficient from a carbon point of view. In Figure 5.8, we can observe the number of datacenter combinations that

stop registering carbon benefits when a certain delay is reached. Out of 24 combinations, we can see that 17 of them lose the benefits of spatial shifting between delays of 64 × 3000 ms and 256 × 3000 ms, meaning that only very high transfer costs could significantly reduce the benefits of spatial shifting. We can also see that only two combinations were affected by very small delays, namely 2 × 3000 ms and 4 × 3000 ms. This implies the fact that those two datacenter combinations will be significantly affected by any task moved from one datacenter to another.

5.5.3 Discussion

In order to test the benefit of our serverless sustainable workload-processing model in a more close-to-reality environment, we considered the transfer costs of moving a task from one datacenter to another, and we represented those costs as task delays. We used a static base delay of 3000 ms for small tasks and 2 × 3000 ms for large tasks, and we implemented this task delay to be customizable, so that users can experiment with more delay ranges in future work.

In Section 5.5.2, we have seen that the effects of task delays on the benefits of spatial shifting depend on the datacenter combination. Out of 24 datacenter combinations, two of them stopped registering carbon benefits from spatial shifting when small delays were applied, while 17 of them reached the 0% carbon emissions improvement when very high delays were used. As a result, we can state that, in most cases, it is still more carbon-friendly to schedule tasks in a spatial shifting manner than running all the tasks in one datacenter, as long as the transfer costs are not extremely high. It is to be mentioned that our results can slightly change in a real-world scenario because, due to customizability, we add a static delay to every moved task, while in real life, the delay might be task-specific. As each delay can impact the carbon emissions of processing a workload, this idea of task-specific delay needs to be explored more in-depth in future work. This experiment constitutes a starting point for such future work.

5.5.4 Findings

The main finding of this experiment is that, depending on the datacenter combination, even a small delay can have a significant impact on the benefits of performing spatial shifting **(MF3)**. Based on previous findings, we also observed that the combinations that registered small carbon improvements in Experiment 2 were more easily affected by the transfer costs than the ones that reported high improvements.

5.6 Experiment 4: Spatial Shifting with Carbon Forecast and Failure Models

The fourth experiment combines our serverless workload-processing model with failure models in order to explore the effect that the failure of a datacenter can have on spatial shifting. A failure model indicates how often there is a failure of a datacenter, how many hosts are affected by the failure, and the duration of each failure. If a host is affected by a failure, then all tasks that are running on that host are stopped until the failure is stopped. In this experiment, we want to see if our model needs to be aware of the reliability of each datacenter when spatial shifting needs to be performed on a task.

This experiment is based on the hypothesis that moving a task to a less reliable datacenter, but with better carbon intensity, might generate more carbon emissions than running the task on the datacenter it originates from. In order to test this hypothesis, we considered two scenarios: one in which the two datacenters have similar failure models and one in which the failure models are very different. In Section 5.6.1, we present the setup of the experiment and the way the failure models were chosen for each scenario, and then we delve into the results in Section 5.6.2. In Section 5.6.3, we discuss the results and any possible limitations.

5.6.1 Failure Traces Integration

In this experiment, we used failure traces from the archive developed by Talluri et al. (46). In order to find the best models for the scenarios we want to explore, we analyzed all models in the archive by calculating for each model the mean of the failure intervals, the mean of the failure durations, and the mean number of affected hosts per failure. Based on those values, we picked two similar failure models for the first scenario, namely *YouTube_user_reported* and *Netflix_user_reported*, and two very different models, such as *FB_Msgr_user_reported* and *Whatsapp_user_reported*. The characteristics of each failure trace can be seen in Table 5.2.

The failure models are specified per cluster in the JSON objects of the experiment file. In the first scenario, on the first datacenter, the YouTube failure model runs, while on the second datacenter, we have the Netflix failure model. As an example, if the combination is *NL_US-CAL-CISO*, on the NL one, we have the YouTube failure model, and on the US-CAL-CISO one, we have the Netflix model. Similarly, for the second scenario, on the first datacenter we run the FB_Msgr failure model, while on the second one we run the

Failure Trace	TBF	Duration	Intensity
YouTube_user_reported	0 days 02:22:08	00:51:06	0.621
Netflix_user_reported	0 days 02:47:44	01:06:41	0.598
FB_Msgr_user_reported	2 days 15:50:22	00:36:51	0.299
Whatsapp_user_reported	0 days 14:35:48	00:26:44	0.579

Table 5.2: Failure traces used in experiments. TBF: average Time Between Failures, Duration: The average duration of a failure, Intensity: The average ratio of hosts that are affected by a failure (46).

WhatsApp failure model. To be mentioned that in this experiment, we do not consider the costs of moving a task from one datacenter to another.

5.6.2 Results

In this section, we present the results of combining our serverless workload-processing model with failure models in two different scenarios: the first scenario illustrates the case when the two datacenters have similar reliabilities (similar failure models), and the second scenario covers the case when one datacenter is less reliable than the other one. In both scenarios, we use the same 24 datacenter combinations that were used in the previous experiments.

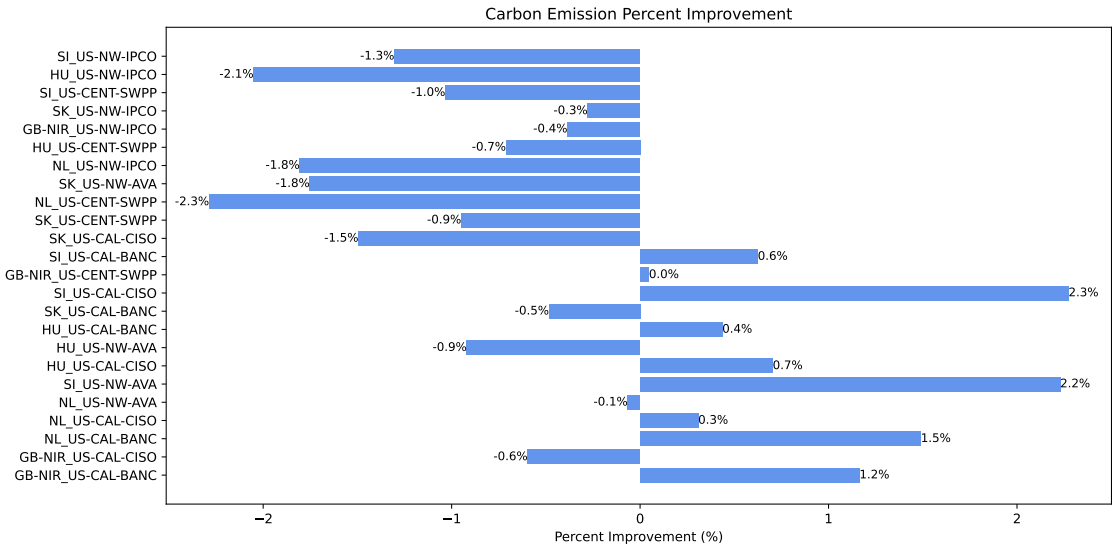

Figure 5.9: Absolute percentage of carbon emissions improvements when running our model with YouTube and Netflix failure models.

In Figure 5.9, we can see the carbon emissions improvements of each datacenter combination when the datacenters have similar failure models, namely the YouTube and Netflix ones. The improvement percentage represents the percentage difference between running a workload with our model that uses spatial shifting and running all the tasks in a workload only on the datacenter with the best carbon intensities. Out of 24 combinations, 16 of them registered negative improvements, meaning that it is more carbon-friendly not to perform spatial shifting. The other nine combinations registered positive improvements, meaning that despite the failures of each datacenter, it is still more carbon efficient to perform spatial shifting.

In order to observe how much the failure models influence the carbon emissions when we run our model, we calculated the difference between the percentage improvement when we do not consider failures and the percentage when we use failures. The results can be seen in Figure 5.10. In this figure, the positive percentage indicates how much worse the carbon emissions are when we have failures compared to when we do not consider failure models. Out of 24 combinations, 23 perform worse when we combine failure models with our model, while one of them, namely *SI_ US-CAL-CISO*, is more efficient when we use failures, registering approximately 1% better carbon emissions. Those results indicate that the failure models influence the carbon emissions of running a workload and that when performing spatial shifting, the scheduler should be aware of the reliabilities of the

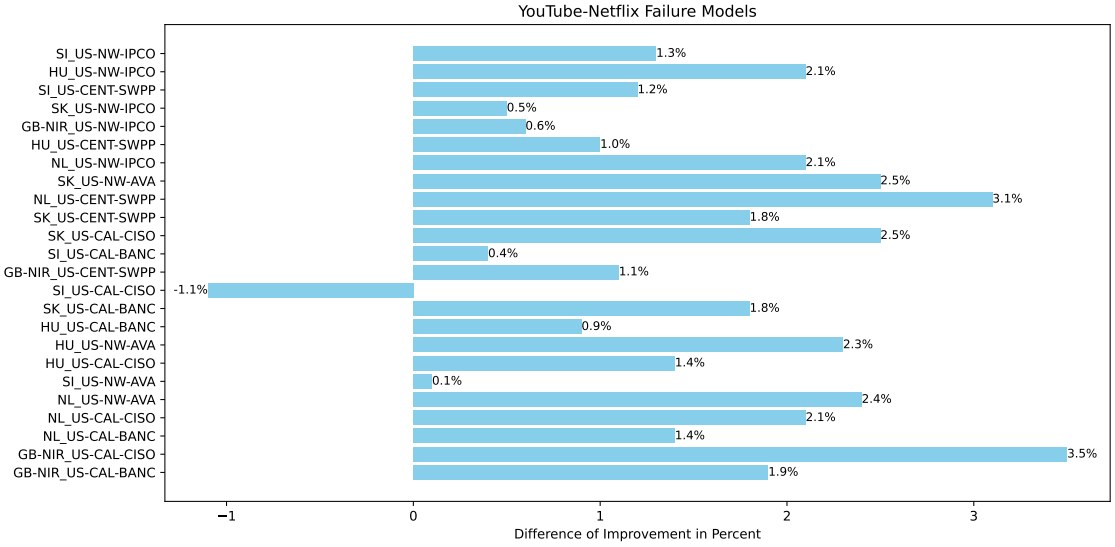

Figure 5.10: The percentage difference of carbon emissions improvements when running our model without and with YouTube and Netflix failure models.

datacenters.

Similarly to the previous scenario, we run 24 datacenter combinations with two failure models in order to see the effects of failures on the carbon emissions of running a workload. The only difference is that the failure models have different reliabilities. For the *WhatsApp* model, the failures are more frequent and affect more hosts than the failures of the *Facebook Messenger* model, making it less reliable. In Figure 5.11, we can see the result of this scenario. Out of 24 combinations, 23 of them register no or worse carbon improvements, while only one of them registers a carbon improvement of 0.3% better than running all tasks on the datacenter with the best carbon intensities. Out of those 23 combinations, 10 of them register very bad percentages, more than 10% worse carbon emissions than when spatial shifting is not performed. While in the scenario with similar failure models, spatial shifting brought some benefits for some cases (the improvement percentages ranged from -2.3 to 2.3), in the scenario presented in Figure 5.11, the spatial shifting is not efficient at all. The only improvement is of 0.3%, which is insignificant.

We can also see how impactful those failure models are on our model and on the carbon emissions in Figure 5.12, where we can observe that there are up to 15.6% worse carbon emissions when we consider failures than when we do not. It is to be noted that, for combinations that had a low spatial shifting rate in the experiment described in Section 5.4, were not influenced too much by the failure models. This indicates that the failure

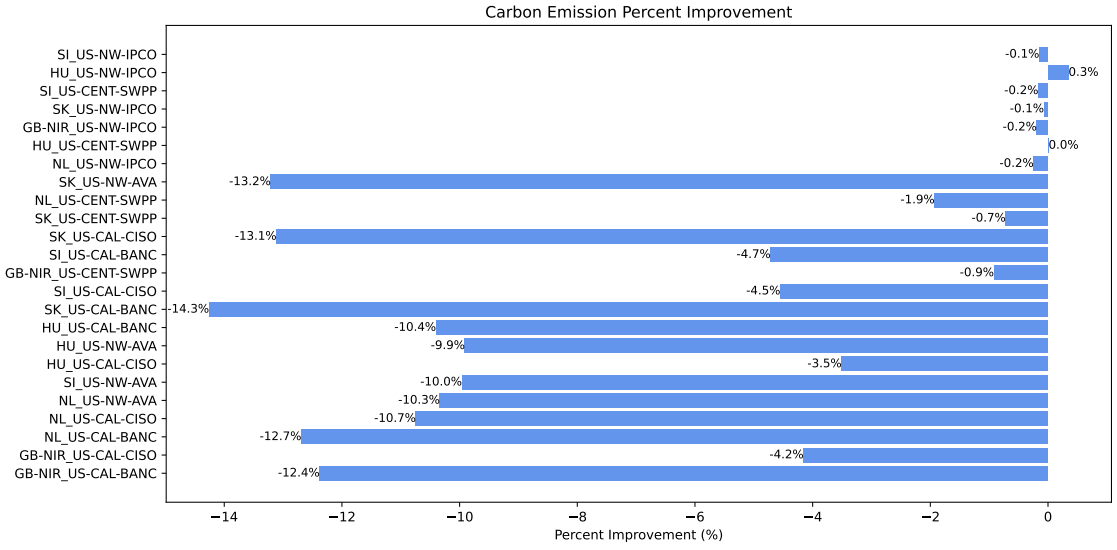

Figure 5.11: Absolute percentage of carbon emissions improvements when running our model with Facebook Messenger and WhatsApp failure models.

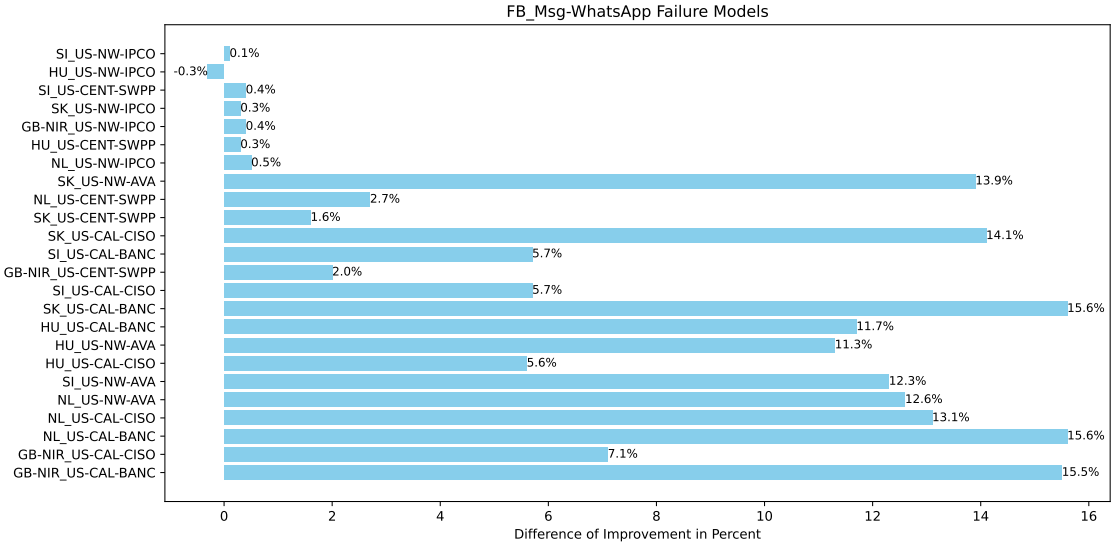

Figure 5.12: The percentage difference of carbon emissions improvements when running our model without and with Facebook Messenger and WhatsApp failure models.

models do not have a big impact on datacenter combinations that have a big difference in carbon intensities, meaning that the scheduler will constantly schedule tasks on the best carbon-wise datacenter, no matter the reliability of that datacenter.

5.6.3 Discussion

Considering the results presented in Section 5.6.2, we can state that failures have a significant impact on the carbon emissions of running workloads and that the scheduler should be aware of the reliability of each datacenter when it performs spatial shifting. We have seen that the larger the difference in reliability between datacenters is, the worse the carbon improvements are. Moreover, in the scenario when the reliabilities of datacenters are very different, failures are less impactful in combinations where data centers have different carbon intensities and spatial shifting is less often performed. This also indicates that the scheduler should be more aware of the reliability of a datacenter before scheduling a task to run on it, because there is a tradeoff between the carbon intensities and the frequency and duration of failures of a datacenter.

It is also important to mention that the results in Section 5.6.2 do not consider the costs of moving a task from one datacenter to another. If we were to consider the costs as well, then the results would be slightly changed depending on the combinations and the duration of each task, as we have seen in 5.5.2. As a possible limitation, we can consider the small

number of combinations on which we run this experiment. More datacenter combinations with different carbon intensities might reveal more interesting edge cases.

5.6.4 Findings

The main finding of this experiment is that the scheduler should consider the reliability of the datacenters before performing spatial shifting **(MF4)**. We have seen that the carbon improvements of each datacenter combination were significantly impacted by the failures. In most cases, it was more carbon-efficient to run the tasks on the datacenter from which they originate than to perform spatial shifting. This result was not only observed in the scenario when one datacenter was less reliable than the other, but also when the datacenters had similar reliabilities.

6

Related Work

Serverless computing in the context of Function-as-a-Service (FaaS) applications is actively explored by the scientific community through simulation. Mahmoudi and Khazaei propose *SimFaaS*, a simulation platform that offers the possibility of developing FaaS applications and analyzing them in terms of cost and performance (11). Moreover, SimFaaS has the benefit of predicting performance metrics such as the quality of service or amount of wasted resources without the need to run long and expensive experiments. Similarly, Raith et al. propose through their simulation framework, *faas-sim*, "a generalized model of serverless system that builds on the function-as-a-service abstraction" and facilitates fine-grained evaluation of serverless platforms on edge-cloud infrastructure (12).

Sharma discusses the energy and carbon implications of FaaS and offers a detailed analysis of the opportunities and challenges of serverless computing in the context of reducing the carbon footprint of datacenters (10). While Sharma states that making serverless computing sustainable can be a significant challenge, Awwad et al. consider that the solution to this challenge is in the configuration adjustments of FaaS applications and emphasize the need for research on fine-grained real-time carbon emission reporting (20).

As the datacenters became a significant contributor to the global energy consumption, the scientific community began exploring more carbon-aware scheduling techniques and developing tools to determine the energy usage and carbon impact of workload processing. Niewenhuis et al. propose *FootPrinter*, a tool that determines the operational carbon footprint of executing a workload on a datacenter and facilitates comparison between datacenter locations (47). Souza et al. present *CASPER*, a carbon-aware scheduler that uses spatial shifting as an operational technique in order to schedule tasks in low carbon regions and report substantial improvements in carbon footprint, up to 70% (30). The benefits of spatial shifting were also discussed by Murillo et al.in the context of Content Delivery

Networks (CDNs). Using spatial shifting led to carbon improvements up to 61.7% globally (5). Detailed analysis of the benefits and challenges of load shifting, more precisely spatial and temporal shifting, was performed by Sukprasert et al. and Wiesner et al. (13, 14).

7

Conclusion

In this thesis, we analyzed the performance and carbon emissions of workload processing in datacenter when serverless models are used. Starting from the idea of reducing the datacenters' environmental impact while still considering the performance-carbon emission tradeoff, we designed, implemented, and evaluated a serverless-sustainable-workload processing model in which we combined the serverless paradigm with load shifting techniques. In order to address the challenges and contributions of this project, we addressed and answered the following three research questions, summarized in Section 7.1.

7.1 Answering Research Questions

RQ1 How to model workloads running in datacenters using serverless models?

In order to answer this research question, we propose a serverless sustainable workload processing model, which addresses the performance-carbon emissions tradeoff. Our model combines the serverless paradigm with load shifting techniques such that the tasks in a workload are scheduled considering both performance and environmental impact. We started from a basic serverless model for workload processing, which assigns hosts to a task based on the hardware requirements of the task, the time each host becomes available, and the estimated duration of the task on each host. Then we developed a more complex model in which we integrated load shifting techniques, such as spatial shifting, in order to perform a more environmentally friendly scheduling. The last version of our model also considers the carbon level while a task is executed. During the execution phase, a task can be interrupted based on the carbon intensity at a specific point in time. If the carbon intensity is higher than a certain threshold, then the task is interrupted until the carbon

level drops below the threshold. A more detailed explanation of our model can be found in Chapter 3.

RQ2 How to simulate datacenters that use a serverless model for workload processing to optimize for performance and climate impact?

We address **RQ2** by integrating our serverless-sustainable-workload processing into a discrete event simulator, namely OpenDC. We designed a more abstract version of our model in order to visualize the main OpenDC components that needed to be modified or included, and we changed the OpenDC code base according to it. Our model implementation presented in Chapter 4 added additional functionality to OpenDC, such as task scheduling in a spatial shifting manner, host filtering based on a preferred datacenter, and host sorting based on the time a host becomes available.

RQ3 What is the impact of datacenters using serverless models for workload processing, compared to non-serverless models for performance and climate impact?

To answer **RQ3**, we designed and executed four experiments that evaluate our model in three different scenarios. The first two experiments consider the scenario in which spatial shifting is performed in an ideal environment, while the third experiment also considers the costs of moving a task from one datacenter to another. The fourth experiment combines our model with failure models in order to see the impact the reliability of a datacenter can have on the scheduling phase and on the overall carbon emissions of executing a workload. By considering those scenarios, we reached the following findings, which are summarized below. To be mentioned that a more detailed description of the experiments can be found in Chapter 5.

MF1 Our serverless workload-processing model registers better carbon emissions when we consider carbon intensities during the entire execution of a task before performing spatial shifting.

MF2 Spatial shifting is more efficient when datacenters with similar mean carbon intensities are considered.

MF3 The cost of moving a task from one datacenter to another has a significant impact on the carbon emissions.

MF4 The reliability of datacenters can significantly reduce the benefits of spatial shifting.

MF5 The number of carbon regions for which the intensity of the carbon over time is well matched to use for spatial shifting is low.

MF6 The number of carbon regions on which spatial shifting does very little is high.

MF7 The impact of spatial shifting is much lower than reported in previous works.

7.2 Limitations and Future Work

Although our proposed serverless sustainable workload processing model proved to improve, in most scenarios, the carbon emissions of processing workloads while also considering the performance of the tasks, there are still limitations that need to be considered and addressed in future work.

One limitation is that our implementation of task delay might not cover all transfer costs of moving a task from one datacenter to another. Our implementation delays a task based on an estimated transfer time, but in real-world scenarios, there might be extra costs to be considered, such as data transfers. In a workload in which tasks are dependent on each other, moving a task to another datacenter might require the entire workload to be moved, an action that would require additional computation power and storage.

Another limitation is the type of data used by a task. Due to different privacy rules in different geographical regions, a task using confidential data might not be allowed to be moved to a different datacenter than the one it originates from.

One more limitation to consider is that a datacenter with overall better carbon intensities might be overloaded, resulting in tasks being delayed for a long time. In future work, more datacenter combinations should be analyzed, and the effects of overloading a datacenter with tasks from different other datacenters should be discussed in more detail.

This thesis represents the starting point of carbon-aware task scheduling by combining the serverless paradigm with load shifting. In this work, we address the sustainability concern of workload processing by including spatial shifting in our scheduling method. In

order to obtain a higher carbon reduction, future work should explore a scheduling method in which both temporal and spatial shifting are implemented. Such implementation would not only find the best location to execute a task but also the most optimal time frame from both a performance and carbon emissions point of view.

References

[1] John Rydning David Reinsel, John Gantz. **Data Age 2025: The Evolution of Data to Life-Critical Don't Focus on Big Data; Focus on the Data That's Big**, 2017. https://www.idc.com/. 1

[2] Varsha Rao and Andrew A. Chien. **Understanding the Operational Carbon Footprint of Storage Reliability and Management**. *SIGENERGY Energy Inform. Rev.*, **4**(5):180–187, April 2025. 1

[3] Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, and Jonathan Koomey. **Recalibrating global data center energy-use estimates**. *Science*, **367**(6481):984–986, 2020. 1, 7

[4] International Energy Agency. **Global data centre CO2 emissions, Base Case, 2020–2035**. IEA, Paris, 2025. Licence: CC BY 4.0. Accessed: 2025-07-05. 1

[5] Jorge Murillo, Walid A. Hanafy, David Irwin, Ramesh Sitaraman, and Prashant Shenoy. **CDN-Shifter: Leveraging Spatial Workload Shifting to Decarbonize Content Delivery Networks**. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*, SoCC '24, page 505–521, New York, NY, USA, 2024. Association for Computing Machinery. 1, 2, 9, 17, 18, 31, 48

[6] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. **Peeking Behind the Curtains of Serverless Platforms**. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 133–146, Boston, MA, July 2018. USENIX Association. 1, 8

[7] IBM. **What Is Serverless Computing?** `https://www.ibm.com/think/topics/serverless`, n.d. Accessed: 2025-08-05. 1

[8] Johann Schleier-Smith, Vikram Sreekanti, Anurag Khandelwal, Joao Carreira, Neeraja J. Yadwadkar, Raluca Ada Popa, Joseph E. Gonzalez, Ion Stoica, and David A. Patterson. **What serverless computing is and should become: the next phase of cloud computing**. *Commun. ACM*, **64**(5):76–84, April 2021. 1, 8

[9] Simon Eismann, Joel Scheuner, Erwin van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L. Abad, and Alexandru Iosup. **Serverless Applications: Why, When, and How?** *IEEE Software*, **38**(1):32–39, 2021. 1, 8

[10] Prateek Sharma. **Challenges and Opportunities in Sustainable Serverless Computing**. *SIGENERGY Energy Inform. Rev.*, **3**(3):53–58, October 2023. 1, 47

[11] Nima Mahmoudi and Hamzeh Khazaei. **SimFaaS: A Performance Simulator for Serverless Computing Platforms**, 2021. 2, 47

[12] Philipp Raith, Thomas Rausch, Alireza Furutanpey, and Schahram Dustdar. **faas-sim: A trace-driven simulation framework for serverless edge computing platforms**. *Software: Practice and Experience*, **53**(12):2327–2361, 2023. 2, 47

[13] Thanathorn Sukprasert, Abel Souza, Noman Bashir, David Irwin, and Prashant Shenoy. **On the Limitations of Carbon-Aware Temporal and Spatial Workload Shifting in the Cloud**. In *Proceedings of the Nineteenth European Conference on Computer Systems*, EuroSys '24, page 924–941, New York, NY, USA, 2024. Association for Computing Machinery. 2, 9, 10, 18, 48

[14] 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*, Middleware '21, page 260–272, New York, NY, USA, 2021. Association for Computing Machinery. 2, 17, 18, 48

[15] Fabian Mastenbroek, Georgios Andreadis, Soufiane Jounaid, Wenchen Lai, Jacob Burley, Jaro Bosch, Erwin van Eyk, Laurens Versluis, Vincent van Beek, and Alexandru Iosup. **OpenDC 2.0: Convenient Modeling and Simulation of Emerging Technologies in Cloud Datacenters**. In *2021*

IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), pages 455–464, 2021. 3, 7, 21, 29, 59, 64

[16] Anahita Ghanad. **An Overview of Quantitative Research Methods**. *INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH AND ANALYSIS*, **06**, 08 2023. 4

[17] Alexandru Iosup, Laurens Versluis, Animesh Trivedi, Erwin Van Eyk, Lucian Toader, Vincent Van Beek, Giulia Frascaria, Ahmed Musaafir, and Sacheendra Talluri. **The atlarge vision on the design of distributed systems and ecosystems**. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, Proceedings - International Conference on Distributed Computing Systems, pages 1765–1776, United States, October 2019. Institute of Electrical and Electronics Engineers Inc. 39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019 ; Conference date: 07-07-2019 Through 09-07-2019. 5, 11

[18] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques For Experimental Design, Measurement, Simulation, and Modeling, NY: Wiley.* Wiley Professional Computing, 04 1991. 5

[19] GitBook Bot, Lambert Heller, datawomanHUB, mcancellieri, Bianca Kramer, Tony Ross-Hellauer, ilabastida, helenebr, Pedro Fernandes, and Jon Tennant. **Open Science Training Handbook**, April 2018. 5

[20] Hanan Awwad, Changyuan Lin, Rabab Ward, and Mohammad Shahrad. **Estimating the Carbon Footprint of Serverless Functions on a Public Cloud Platform**. In *Proceedings of the 3rd Workshop on SErverless Systems, Applications and MEthodologies*, SESAME' 25, page 12–20, New York, NY, USA, 2025. Association for Computing Machinery. 7, 47

[21] Kazi Main Uddin Ahmed, Math H. J. Bollen, and Manuel Alvarez. **A Review of Data Centers Energy Consumption and Reliability Modeling**. *IEEE Access*, **9**:152536–152563, 2021. 7

[22] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. **Versatile, scalable, and accurate simulation of distributed applications and platforms**. *Journal of Parallel and Distributed Computing*, **74**(10):2899–2917, 2014. 7

[23] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. **CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms**. *Softw. Pract. Exper.*, **41**(1):23–50, January 2011. 7

[24] Alberto Núñez, Jose L. Vázquez-Poletti, Agustin C. Caminero, Gabriel G. Castañé, Jesus Carretero, and Ignacio M. Llorente. **iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator**. *Journal of Grid Computing*, **10**(1):185–209, Mar 2012$_0$3.7

[25] Dzmitry Kliazovich, Pascal Bouvry, Yury Audzevich, and Samee Ullah Khan. **GreenCloud: A Packet-Level Simulator of Energy-Aware Cloud Computing Data Centers**. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pages 1–5, 2010. 7

[26] Jerry Banks, John S. Carson, Barry L. Nelson, and David M. Nicol. *Discrete-Event System Simulation*. Prentice-Hall, 4 edition, 2005. 8

[27] Georgios Andreadis, Laurens Versluis, Fabian Mastenbroek, and Alexandru Iosup. **A reference architecture for datacenter scheduling: design, validation, and experiments**. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018*, pages 37:1–37:15. IEEE / ACM, 2018. 8

[28] Adel N. Toosi, Bahman Javadi, Alexandru Iosup, Evgenia Smirni, and Schahram Dustdar. **Serverless Computing for Next-generation Application Development**. *Future Generation Computer Systems*, **164**:1–5, March 2025. Part of special issue: Severless computing for Next-generation Application Development. Publisher Copyright: © 2024. 8

[29] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, João Carreira, Karl Krauth, Neeraja Jayant Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. **Cloud Programming Simplified: A Berkeley View on Serverless Computing**. *CoRR*, **abs/1902.03383**, 2019. 8

[30] Abel Souza, Shruti Jasoria, Basundhara Chakrabarty, Alexander Bridgwater, Axel Lundberg, Filip Skogh, Ahmed Ali-Eldin, David Irwin, and

Prashant Shenoy. **CASPER: Carbon-Aware Scheduling and Provisioning for Distributed Web Services**. In *Proceedings of the 14th International Green and Sustainable Computing Conference*, IGSC '23, page 67–73, New York, NY, USA, 2024. Association for Computing Machinery. 9, 31, 47

[31] Amy Li, Sihang Liu, and Yi Ding. **Uncertainty-Aware Decarbonization for Datacenters**, 2024. 9, 10

[32] Walid A. Hanafy, Roozbeh Bostandoost, Noman Bashir, David Irwin, Mohammad Hajiesmaili, and Prashant Shenoy. **The War of the Efficiencies: Understanding the Tension between Carbon and Energy Optimization**. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems*, HotCarbon '23, New York, NY, USA, 2023. Association for Computing Machinery. 9, 10

[33] Roozbeh Bostandoost, Walid A. Hanafy, Adam Lechowicz, Noman Bashir, Prashant Shenoy, and Mohammad Hajiesmaili. **Data-driven Algorithm Selection for Carbon-Aware Scheduling**. *SIGENERGY Energy Inform. Rev.*, **4**(5):148–153, April 2025. 10, 36

[34] Ana Radovanovic, Ross Koningstein, Ian Schneider, Bokan Chen, Alexandre Duarte, Binz Roy, Diyue Xiao, Maya Haridasan, Patrick Hung, Nick Care, Saurav Talukdar, Eric Mullen, Kendal Smith, MariEllen Cottman, and Walfredo Cirne. **Carbon-Aware Computing for Datacenters**, 2021. 10

[35] Laurens Versluis, Mehmet Cetin, Caspar Greeven, Kristian Laursen, Damian Podareanu, Valeriu Codreanu, Alexandru Uta, and Alexandru Iosup. **Less is not more: We need rich datasets to explore**. *Future Generation Computer Systems*, **142**:117–130, 2023. 29

[36] **Electricity Maps (2025). The Netherlands 2021 - 2024 Carbon Intensity Data (Version January 27, 2025)**. 29, 59

[37] **Electricity Maps (2025). Slovenia 2021 - 2024 Carbon Intensity Data (Version January 27, 2025)**. 29, 59

[38] **Electricity Maps (2025). Slovakia 2021 - 2024 Carbon Intensity Data (Version January 27, 2025)**. 29, 59

[39] **Electricity Maps (2025). Hungary 2021 - 2024 Carbon Intensity Data (Version January 27, 2025)**. 29, 59

[40] **Electricity Maps (2025). Northern Ireland 2021 - 2024 Carbon Intensity Data (Version January 27, 2025)**. 29, 59

[41] **Electricity Maps (2025). Balancing Authority of Northern California 2021 - 2024 Carbon Intensity Data (Version January 27, 2025)**. 29, 59

[42] **Electricity Maps (2025). CAISO 2021 - 2024 Carbon Intensity Data (Version January 27, 2025)**. 29, 59

[43] **Electricity Maps (2025). Southwestern Power Administration 2021 - 2024 Carbon Intensity Data (Version January 27, 2025)**. 29, 59

[44] **Electricity Maps (2025). Avista Corporation 2021 - 2024 Carbon Intensity Data (Version January 27, 2025)**. 29, 59

[45] **Electricity Maps (2025). Idaho Power Company 2021 - 2024 Carbon Intensity Data (Version January 27, 2025)**. 29, 59

[46] Sacheendra Talluri, Dante Niewenhuis, Xiaoyu Chu, Jakob Kyselica, Mehmet Cetin, Alexander Balgavy, and Alexandru Iosup. **Cloud Uptime Archive: Open-Access Availability Data of Web, Cloud, and Gaming Services**, 2025. 40, 41, 59

[47] Dante Niewenhuis, Sacheendra Talluri, Alexandru Iosup, and Tiziano De Matteis. **FootPrinter: Quantifying Data Center Carbon Footprint**. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering*, ICPE '24 Companion, page 189–195, New York, NY, USA, 2024. Association for Computing Machinery. 47

Appendix A

Reproducibility

A.1 Abstract

In this section, we provide an overview of how to reproduce the result in Section 5 and how to execute the same experiments used in this work but with different input.

A.2 Artifact check-list (meta-information)

- **Program: OpenDC**
- **Data set: (15), (36, 37, 38, 39, 40, 41, 42, 43, 44, 45), (46)**
- **Experiments: described in Chapter 5**
- **How much time is needed to complete experiments (approximately)?: 3h**
- **Publicly available?: Yes (See** `https://github.com/amusca/opendc.git`)

A.3 Description

In this section, we describe the structure of our experiments.

A.3.1 How to access

The implementation of our serverless sustainable workload processing model can be accessed on GitHub (`https://github.com/amusca/opendc.git`).

A.3.2 Data sets

Workload: surf_month (15)
Carbon traces: Electricity Maps traces(36, 37, 38, 39, 40, 41, 42, 43, 44, 45)
Failure traces: Cloud Uptime Archive (46)

A.3.3 Experiment Structure and Particularities

Listening 1 shows the structure of the experimental files used for Experiments 1-3, while Listening 2 shows the structure of Experiment 4. The difference between those two structures is that Experiment 4 has an extra field, *failureModels*, which allows us to integrate the failure traces in our experiment configuration.

```
{
    "name": "",
    "topologies": [],
    "workloads":[],
    "exportModels": [],
    "allocationPolicies":[{
            "type": "filter",
            "filters": [],
            "weighers": []
        }
    ]
}
```

Listing 1: The structure of the JSON file of the Experiments 1-3 presented in Section 5.3. 5.4, 5.5

```
{
    "name": "",
    "topologies": [],
    "workloads":[],
    "exportModels": [],
    "failureModels": []
    "allocationPolicies":[{
            "type": "filter",
            "filters": [],
            "weighers": []
        }
    ]
}
```

Listing 2: The structure of the JSON file of the Experiment 4 presented in Section 5.6

Listings 6, 5, 7, 3, and 4 show the configuration that were used in our experiments for each of the fields Listings 1 and 2. A detailed explanation of each field in an experiment field can be accessed at `https://atlarge-research.github.io/opendc/docs/category/input`.

```
"topologies": [
    {
        "pathToFile": "topologies/spatial_shifting_NL_US-CAL-BANC.json"
    }
]
```

Listing 3: Example of topology field.5.6

```
"workloads": [
        {
            "pathToFile": "workload_traces/surf_month",
            "type": "ComputeWorkload"
        }
    ]
```

Listing 4: The workload field specifications used during the experiments.

```
"exportModels": [
        {
            "exportInterval": 1000,
            "printFrequency": 168,
            "filesToExport": [
                "host",
                "powerSource",
                "service",
                "task"
            ]
        }
    ],
```

Listing 5: The exportModel field specifications used during the experiments.

To be noted that a topology field can contain multiple paths to multiple topologies in order to execute the experiment with different datacenter combinations.

```
"allocationPolicies": [
        {
            "type": "filter",
            "filters": [
                {
                    "type": "Compute"
                },
                {
                    "type": "VCpu",
                    "allocationRatio": 1.0
                },
                {
                    "type": "Ram",
                    "allocationRatio": 1.5
                }
            ],
            "weighers": [
                {
                    "type": "SpatialShifterForecast",
                    "multiplier": -1.0
                }
            ]
        }
    ]
```

Listing 6: The allocationPolicies field specifications used during the experiments.

The *filters* field in Listing 6 can include multiple other filters than the ones we applied. We applied these filters in order to remove the hosts that do not meet the hardware requirements of a task from the list of eligible hosts. Similarly, for *weighers* we only use the **SpatialShifterForecast** weighter in order to perform spatial shifting, but, in general, more weighters can be added.

```
"failureModels": [[
            {
                "type": "trace-based",
                "clusterName": "C01",
                "pathToFile": "path_to_failure_trace"
            },
            {
                "type": "trace-based",
                "clusterName": "C02",
                "pathToFile": "path_to_failure_trace"
            }
        ]
    ]
```

Listing 7: The failureModel field specifications used during the experiments.

A.3.4 Topology Structure

Listing 8 shows the structure of the topologies used during the experimentation phase of this work. An example of the configuration of a cluster can be seen in Listing 9. For each cluster in a datacenter combination we use the same hosts specifications as in the given example, the only differences are in the *powerSource* field, where the path to the carbon trace of the datacenter has to be specified.

```
{    "clusters": [
        {
            "name": "C01",
            "hosts": [],
            "powerSource": {}
        },
        {
            "name": "C02",
            "hosts": [],
            "powerSource": {}
        },
    ]}
```

Listing 8: The structure of a topology used for our experiments.

```
{
    "name": "C01",
    "hosts": [
        {
            "name": "H01",
            "cpu": {
                "coreCount": 16,
                "coreSpeed": 1500
            },
            "memory": {
                "memorySize": 100000
            },
            "cpuPowerModel": {
                "modelType": "sqrt",
                "power": 400.0,
                "idlePower": 32.0,
                "maxPower": 180.0
            },
            "count": 300
        }
    ],
    "powerSource": {
        "name": "datacenter_name",
        "carbonTracePath": "path_to_carbon_trace"
    }
}
```

Listing 9: An example of a datacenter topology used for the Experiments.

A.4 Installation

In order to be able to run the experiments, OpenDC (15) should be set up on the local machine. Once OpenDC is set up, the experiments can be run by executing the ExperimentCli.kt file with the following specifications:

Program arguments –experiment-path experiments/experiment.json
Working directory: path_to_working_directory

where experiment.json is the JSON file of the experiment. A full guide of how to install OpenDC can be accessed as `https://atlarge-research.github.io/opendc/docs/getting-started/start-using-intellij`.

A.5 Evaluation and expected results

The results should be similar to the ones in Chapter 5.

A.6 Experiment customization

In order to run the same experiments but with different input, the following changes need to be made:

1. To create a new datacenter combination, the *powerSource* field of each cluster needs to be changed, more precisely the *name* and *carbonTracePath*.
2. To run different failure models than the ones used in Experiment 4, the *pathToFile* field from Listing 7 needs to be changed.

In order to force all tasks in a workflow to run on only one of the clusters, we need to add the filter in Listing 10 to the experiment file.

```
{
    "type": "Cluster",
    "selectCluster": "name_of_cluster"
}
```

Listing 10: Cluster filter configuration.