Project - Eco-FaaS (revised)

André Ferreira
Número: 92423
Email: andre.nelas@tecnico.ulisboa.pt

Instituto Superior Técnico

Abstract. Cloud computing is a model for delivering information technology services in which resources such as storage, computing power and applications are provided over the internet on a pay-as-you-go basis. One of its services, Function-As-A-Service (FaaS), allows users to run and scale code in a serverless way: without the need for provisioning or managing servers. With the growing concerns about the environmental impact of data centers, Ecological Function-As-A-Service (Eco-FaaS) has been proposed as a way to reduce CO2 emissions. This research paper explores the concept of Eco-FaaS, its potential benefits, current challenges and potential solutions for sustainable management of ecosystems. Overall, it aims to demonstrate the feasibility and value of Eco-FaaS as a sustainable management tool.

Keywords: Function-As-A-Service (FaaS), CO2 emissions, Green energy, Scheduling

1 Introduction

Cloud computing is a rapidly growing technology that has revolutionized the way organizations access and manage their computing resources while granting benefits like flexibility, cost savings and the ability to access the latest technologies. This article provides a brief overview of cloud computing technology and delves deeper into one of its services: Function-As-A-Service (FaaS).

Cloud computing has already met the energy efficiency paradigm, from a financial perspective. The ever growing data centers require more and more energy which has created an area of focus to achieve significant cost savings.

Now, the increasing pressure on ecosystems and the need for sustainable management practices has led to the development of a new concept: Ecological Function-As-A-Service (Eco-FaaS). Eco-FaaS refers to the deployment of functions services with the sustainable perspective to reduce CO2 emissions, in line of previous [23] and recent work [18]. In this research paper, we explore the concept of Eco-FaaS and its potential to provide mutually beneficial outcomes for both ecosystems and society. We also examine current related case studies and challenges in the implementation of Eco-FaaS and suggest potential solutions for overcoming these obstacles. Overall, this paper aims to demonstrate the value and feasibility of Eco-FaaS as a tool for achieving sustainable management of ecosystems.

1.1 Function-As-A-Service

Function-As-A-Service (FaaS) is a cloud computing model that allows developers to build, run and manage application functionalities, known as functions without the need for provisioning or managing infrastructure. This means that developers can simply write their code and deploy it to a FaaS provider such as AWS Lambda, Google Cloud Functions or Azure Functions and the provider will take care of the rest [17].

FaaS is a form of serverless computing which means that the underlying infrastructure is abstracted away and the developer only pays for the resources and compute time used by their functions. This can result in saving upfront costs, as well as increased scalability and reliability since the FaaS provider will automatically handle the scaling and availability of the functions [19].

FaaS functions are typically event-driven which means that they are triggered by a specific event such as a new file being uploaded to a storage service or an HTTP request being made to an API endpoint. The function will then run and perform its specified tasks.

FaaS functions can be written in a variety of programming languages, like JavaScript, Python, C# and Go. They can also be integrated with a wide range of services and technologies, like databases or storage services.

In summary, FaaS is a cloud computing model that allows developers to build and execute functions without the need to manage infrastructure, resulting in cost savings and increased scalability and reliability. It's an important service of cloud computing.

1.2 Motivation

Reducing carbon dioxide (CO2) emissions in the context of cloud computing is motivated by the growing concern about the impact of climate change and the need to reduce the carbon footprint of the technology industry. The cloud computing industry is a significant contributor to global carbon emissions, as data centers consume a significant amount of energy [35]. Reducing CO2 emissions in cloud computing is also important to promote environmental care and social responsibility. Climate change is a global problem that affects everyone and we must do our part to mitigate its impact.

It is also motivated due to the research gap in this area and the necessity to venture into this recent paradigm. Nevertheless, there are multiple relevant papers written about adjacent topics which greatly supported this work. One such example is the paper "Let's Wait Awhile: How Temporal Workload Shifting Can Reduce Carbon Emissions in the Cloud" [31] which also brings a possible solution to this challenge. This paper builds its own solution in the dimension of time while mentioning the existing constraints and trade-offs. This work is similar, but it aims to build a solution through the space dimension, concretely geographical location. There are also constraints and trade-offs to mention, like latency due to communication and constraints related to the data center's region.

1.3 Objectives

The aim of this work is to propose a scheduling algorithm that brings CO2 emissions into the calculations. In order to accomplish this we set the following objectives:

1. Examine the latest advancements in FaaS, gather a comprehensive list of crucial design choices that comprise these systems and comprehend the limitations of existing ones.
2. Develop an environment ready to test the proposed algorithms. The environment should approximate reality as much as possible so to enable representative results.
3. Design a scheduling algorithm to deploy and test on the environment.
4. Develop a systematic evaluation process for determining if our future work conforms to our requirements when tested.

1.4 Document organization

This document is structured in the following way:

- Section 2: Describes the current technologies and its maturity in the cloud computing landscape.
- Section 3: Presents systems related to the topics mentioned and that showcase results of solutions for similar challenges.
- Section 4: Describes in detail the proposed solution and reflects on all benefits and shortcomings.
- Section 5: Provides an explanation on how the testing process and data collecting will execute.
- Section 6: The final words of this work.

2 Related work

In this section we will present the result of our analysis regarding the topics more relevant to this work. Starting with the related cloud computing service: Function-As-A-Service. Following it with Edge Computing works since it's essential for our focus. Finishing with Ecological Efficiency, our main perspective on how to look at Cloud Computing.

2.1 Function-As-A-Service

Progression strives in Cloud Computing and its pushes in Virtualization and Software Architecture brought us Serverless Computing [17]. Serverless consists in a new programming philosophy which distances the developer from the server concept and its management.
Function-As-A-Service [19] came to be thanks to this new philosophy and the

technological advances that enabled it. Developing software in this new paradigm implies abstraction regarding the context in which the application is run.

This brings significant advantages since the development process doesn't need to take into account computing resources, like ecological inefficiencies in having a computer turned on with a very low CPU usage or dealing with bottlenecks due to high CPU usage.

It also brings disadvantages, for example: developers are mostly restricted to deploying stateless functions, in which its work is only determined by the input and easy to handle since they don't share data structures.

This progression also brought interest for clients with specific demands. For example, a client with quick and intense workload demands was not satisfied with Infrastructure As A Service (IaaS) since reserving or launching Virtual Machines for these cases was not economically friendly. This new type of service: FaaS, brings a cost-effective solution, with attention to the pricing options and its current development [3].

There are multiple ways to host a system holding FaaS capabilities, starting with the major cloud service providers: Google Cloud Functions, Amazon Web Services Lambda and Azure Functions. These hosts need to support of multitude of libraries and programming languages to accommodate their client's demands and bring them a large offer. At the same time, they need to restrict developers to certain standards so they can keep accommodating their multitudes of users. Due to theses restrictions, multiple Open Source frameworks are developing and exploring this new paradigm with interesting benchmark results in their current maturity [17].

The common business model states that clients only pay accordingly to their function's execution time and memory usage, ignoring the server details' like the function deployment, scheduling and orchestration overheads. Therefore, the cloud providers are economically inclined to reduce these costs [22]. Consequently, this creates an appealing opportunity for computer architecture to branch itself deeper into FaaS support since there's a considerate slowdown running functions in the providers infrastructure. Most cloud providers were focused on supporting applications with high time usage while FaaS functions are usually of little execution time. Aspects depending on temporal locality like branch prediction, for example, are severely hurt in the servers running as many functions as they can [22].

The following sub chapters will approach the state-of-the-art in the following categories:

- Orchestration;
- Scheduling;
- Security;
- Performance;

2.1.1 Orchestration

Orchestration consists in the management ability to create connections reactively between containers (representing workloads) in a cloud environment.

In other cloud services, the deployment orchestration was usually in charge of the Load Balancer, implemented by the client. In FaaS the cloud provider has greater control over its resources and more responsibility as well since it needs to manage the Orchestration and Scheduling [26].

Orchestration is a very useful capability that can be implemented on top of FaaS infrastructures. Creating a workflow with functions (as in functions supported and deployed in FaaS) is a highly sought after service. The workflow based on micro services provides an incredibly simple platform to develop that it's still capable of providing powerful features.

The first feature is that stateless functions are inherently very parallelizable due to their inability to synchronously share data structures. This feature is highly capitalizable from the provider's perspective when taking care of function deployment. This segment of the cloud technology can be further developed since most of the systems available are not utilizing this to its fullest.

There are multiple models to consider when deploying orchestration capabilities, we will mention two:

- The first option is to leave the responsibility to the client. In this case, the developers must deploy functions that orchestrate other functions. These orchestrator functions must manage the connections and deployment of the other serverless functions. This model increases costs since there are intrinsically more functions running. It can be called double billing since both orchestrator and orchestrated functions need to run in parallel.
- The second option is to create an external orchestrator in charge of accommodating the functions executions correctly. This solution breaks the principle of a true serverless service [5].

2.1.2 Scheduling

Scheduling is an area where Cloud Providers were already focused on due to other services supported. Despite that, FaaS introduced big changes to how Cloud Providers capitalize their resources utilizing management technologies.

The deployed function might find its container in one of the following states:

- Cold Start: The slowest possible, the container needs to load the function code and all the dependencies required;
- Pre Warmed: A medium state where the environment is correctly set up and its only missing the code itself;
- Warmed: The fastest conditions, the container has already ran this specific function and is ready to immediately run it again;

There are multiple approaches surging regarding all the difficulties mentioned before:

- Allocation heuristics: Schedulers focused on allocation algorithm responsible for allocating resources.
  For example NOAH [26], a non cooperative game between controllers and FaaSRank [34], based on monitoring and reinforcement learning.
- Topology Awareness: This perspective takes into account the physical or logical layout of the underlying infrastructure. In this approach, the scheduler considers factors such as the location of the nodes and the connectivity between them, the locality of functions relative to their data sources or dependencies and the capacity and utilization of different subsections of the framework when determining where to place functions [8].
- Predictive Scheduling: This proposition utilizes statistical models or even machine learning to allocate the functions according to their predicted execution times.
  For example, ETAS estimates functions' execution times based on their history of previous executions [4].
- Deadline Awareness: This suggestion factors in the time remaining or its life cycle. It also schedules its functions based on their Service Level Objective (SLO) and its hardware requirements. Prioritizing the execution of functions with a sooner deadline ensures that important functions are completed in a timely manner which can boost the system's overall performance.
  For example, ENSURE [27] develops the focus on Service Level Objectives (SLOs) while still maintaining acceptable application latency and greatly boosting resource efficiency while CAS [32] focuses on Cold Start resolution without adding a significant overhead.
- Quality-of-Service Awareness: These approaches extend the concept of Quality-of-Service (QoS) to Function-As-A-Service (FaaS). Due to the lack of availability from the user's part to enforce QoS requirements, there are multiple suggestions on systems to enable this without adding a significant overhead [20].

2.1.3 Security

Security is very important in the cloud computing scope, consequently its important in the smaller Function-As-A-Service scope.

The first step is to protect the system against possible data breaches caused by the paralleled and shared nature of the cloud provider layout. Since multiple entities are placed inside this infrastructure, the risk is bigger since the severity is intrinsically bigger as well.

The second step is to defend against malicious code execution or unintended function's executions. Providing a sandbox environment is a staple in Cloud Computing to solve this risk.

The third step is to handle sensitive data in such way that confidentiality and integrity of the user's input is retained. Sustaining the General Data Protection Regulation (GDPR) principles is vital to any Cloud Provider system.

For example, CLEMMYS [28] is a platform conceived with the purpose of ensuring privacy over client's functions sources and data through a message encryption protocol without a considerate overhead.

Concluding, security is vital to the design process of Cloud Computing and subsequently FaaS platforms.

2.1.4 Performance

Performance in a Function-As-A-Service environment consists in the focus of minimizing application execution time.

It's deeply related to the Scheduling and Orchestration contexts since we already mentioned various systems that focus on performance on those sub sections. In this sub section we will focus on other techniques:

- Sandbox Design: Sandboxing is vital to any cloud computing infrastructure as explained in the Security sub section. It also impacts greatly on the performance of the given infrastructure. Some platforms can reuse containers to run the same specific functions since both the libraries and code are already imported onto the container.
  Other systems propose different sandbox designs such as SAND [1]. SAND proposes two different types of isolation for its containers. Isolation between functions from different applications is identical to the standard isolation practices but isolation between functions from the same application differs. Making use of the repetitive nature and similarity between these functions, SAND creates an environment that focus on the locality advantages.

- Communication: Communication between functions with dependencies are one of the most significant reasons for overhead in FaaS. Usually FaaS platforms work with a centralized message system, there are alternatives being explored like an hierarchical message bus [1].

- Cold Start Latency: As explained in the scheduling sub chapter, cold start refers to the time it takes to allocate resources to invoke a specific function. There are various techniques to mitigate the impact of cold stars on performance such as keeping resources for the most frequently called functions allocated or using a predicting statistical system or a reinforcement learning trained agent decide which containers to pre warm [6] [29].

System	Topic	Feature Application	Focus
NOAH [26]	Scheduling	Cooperative Game Scheduling	Lower resource cost
FaaSRank [34]	Scheduling	Reinforcement Learning	Lower execution time
ENSURE [27]	Scheduling	Efficient Resource Management	Service Level Objectives
CAS [32]	Scheduling	Lifecycle Awareness	Cold Start Solution
CLEMMYS [28]	Security	Message Encryption Protocol	Low Overhead
SAND [1]	Performance	Isolation and Communication	Fast Resource Allocation

2.2 Edge Computing

Edge Computing came to be thanks to the advancements of Cloud Computing infrastructures and the hardware related to Internet of Things (IoT) devices. It consists in decentralizing computation in a cloud environment. This means that part of the computation is brought the cloud center to the "edge" which most of the time, represents IoT devices or any system that can run a browser [30] [13].

The advantages of Edge Computing consist mainly in following aspects:

- Latency: in the Cloud Computing context, it refers to the time it takes for the request to travel between the client and the cloud user summed to the time it takes to process the request.
  With Edge Computing there is a great reduce in latency through its inherent local processing. Since computation is closer to the data source, there's a significant reduction from both the time from communication and the time for processing [7].
- Efficiency: local processing reduces the amount of data that is transferred and stored, helping lower the running costs of the cloud center.
- Security: limiting the storage of data and its communication is particularly useful for reducing the risk of a data breach.
  There also difficulties worth mentioning such as the coverage of security solutions through heterogeneous IoT devices in the Edge and the available solutions to these devices [7].
- Bandwidth: Edge Computing thrives in low-bandwidth environments due to the significant reduction in the amount of data that needs to be transmitted. Data caching and replication are techniques performed in these environments to both enable and improve performance. This causes lower entry barriers for community clouds (CN) [21].

The uses of Edge Computing capitalize on these advantages. Some applications (augmented reality and autonomous vehicles for example) might require real-time outputs from their functions and local processing brings them the necessary minimal latency.

The following sub chapters will approach the following categories inside the Edge Computing context:

- Volunteer Computing

– Orchestration
– Integration with Function-As-A-Service
– Energy Efficiency
– Community Cloud

2.2.1 Volunteer Computing

Volunteer Computing consists in harnessing idle processing resources to compute distributed functions. The amount of IoT devices and of personal computers represents a colossal untapped resource which can be harvested to greatly boost performance of specialized systems [2,25].

Volunteer Computing has multiple benefits such as:

– Cost: Offloading computation to otherwise idle devices from volunteers incurs in significant cost savings. The costs related to obtaining processing power, mounting it under a private infrastructure and maintaining it represents a great save in a monetary and time-related perspectives [12].
– Community: Volunteer computing offers engaging options for individuals to make use of their resources. Creating a simple manner for someone to contribute to a variety of causes, commonly represented by Community Clouds [21].
– Scalability: Compared to traditional Cloud Computing an organization utilizing Volunteer Computing can flexibly scale their existing computation power with minimal cost.
  Volunteer Computing also lowers entry barriers for organizations with propelling causes since it offers computation power with minimal capital investment. The adherence of volunteers stands in place of most of the processing power that the organization could require [12].

2.2.2 Orchestration

Orchestration is critical in the Edge Computing context to guarantee that the resource management and communication is efficient despite all the heterogeneity and the geographic distribution of the computing devices.

Let's enumerate the main characteristics of Orchestration in our particular context:

– Deployment: Deployment in centralized cloud environments usually has a considerate overhead which does not adhere to the computation restrictions in IoT devices.
  A lightweight solution to not only deployment, but management and monitoring is needed to effectively utilize IoT devices [30].

– Reliability: Relying on distributed resources implies that there's a need for mechanisms that permit correct operations even after faults in edge nodes [14].
  One commonly used solution is replication which consists in creating redundant nodes to achieve fault tolerance [7].
– Complexity: It's challenging to effectively manage resources with the possible magnitude of computing nodes in an Edge Cloud.
  The additional layer of computation or even multi-layers in Fog Computing generates gaps in the current research in the Monitoring and Management fields [7].

An effective orchestration consists in the following steps:

– Resource discovery: This sub process is responsible for perceive new nodes that can join the Fog infrastructure. These nodes can declare their own availability to the orchestration framework or can be found by scanning the Internet which involves security protocols [7].
– Provisioning: After the node integration in framework, the orchestrator is responsible for the resource allocation needed to support the functions deployment [7].
– Scheduling: This step is in charge of obtaining the best viable performance with the available resources acquired earlier [7].

2.2.3 Integration with Function-As-A-Service

The Function-As-A-Service model fulfills various needs for Edge Computing and sustains its benefits.
First off its simplicity makes it easy for developers to create and deploy functions thanks to serverless abstraction layer. Edge Computing focus on real-time processing and the dynamically scalability in the FaaS model can fulfill that need. [33].

To address a real integration, the following design principles must be sustained:

– Heterogeneity:. The computing nodes on the edge environment can consist of a large variety of different devices. The orchestration must coordinate all these different devices in a harmonious way [33].
– Scalability: The flexibility when provisioning devices must be sustained when handling the increase of available devices. The system must ensure that the recognition and allocation of a new node doesn't affect the performance of other nodes [33].
– Performance: Usual FaaS models operate with powerful machines, whereas Edge Computing mostly supports lightweight devices. The fast execution time procured in FaaS is especially challenging in an Edge infrastructure [33].

– Reliability:. When considering multiple devices and without being able to ensure their timely functionality in Edge Computing, there's a significant effort made for reliability. Fault-tolerance techniques must account for device failures, network disruptions, data loss and resource constraints and still manage to provide QoS thresholds [33].

2.2.4 Energy Efficiency

Energy efficiency refers to the ability of a system to perform its intended function using as little energy as possible. In the context of edge computing, energy efficiency is an important consideration because edge devices are often battery-powered and may be deployed in locations where access to power is limited or unavailable.

There are a number of approaches to improving energy efficiency in edge computing systems:

– Optimizing resource utilization: By optimizing the use of resources, it is possible to reduce the amount of energy that is required to perform a given task. This can involve techniques such as minimizing the amount of data that is transmitted and processed, reducing the number of idle resources and scaling resources up or down as needed based on demand [16,11].
– Energy-efficient hardware: Another approach is to use energy-efficient hardware such as low-power processors and energy-efficient storage devices, to reduce the amount of energy that is required to perform a given task [11,23].
– Power management: By using power management techniques such as turning off or hibernating idle devices, it is possible to reduce the amount of energy that is consumed by edge devices [11].
– Energy and Compute co-harvesting: In some cases, it may be possible to use energy harvesting technologies such as solar panels or kinetic energy generators, to power edge devices [11], or use abundant energy to process computation that brings value to the network [24]. This can be particularly useful in remote or inaccessible locations where access to power is limited.

Improving energy efficiency is an important consideration in the design and operation of edge computing systems. By implementing appropriate techniques and technologies, it is possible to reduce the energy consumption of edge devices and improve the overall sustainability of edge computing systems.

2.2.5 Community Cloud

A community cloud is a type of cloud computing infrastructure that is shared by a group of organizations with similar requirements and concerns. In the context of edge computing, a community cloud can provide a way for organizations to share resources and infrastructure in order to support the deployment and operation of edge functions [21].

Community Clouds bring great impact in response to both urban and rural areas' increasing demands for network access. This type of cloud computing is typically build from the bottom-up, due to the involvement and the community's members and their wiring and building process.

The largest Community Cloud in the world Guifi.net, an effective example of this paradigm. Guifi.net is described as an open, free and neutral Community Cloud being developed by its users. In other words, individuals and groups pool their resources and work together to create and maintain a local network infrastructure. The ultimate goal of Guifi.net is to build a usable digital ecosystem that serves a very small geographic area. This is very challenging, since that when looking at the Internet services consumers mainly require streaming services causing a heavy weight in the internet links. This situation generated the creation of Micro-Clouds. A Micro-Cloud resides inside a Community Cloud with the purpose of creating a sort of small clusters to provide services to its users. Services like video streaming or personal storage sharing. The goal is to give a better service and reduce overload on the backbone links by utilizing the shorter and faster communication between customers.

2.3 Green Energy

Energy efficiency is a challenge tackled by many research fronts in the past years. It has been mainly driven by the business perspective which focuses on reducing the cost as most as possible while still maintaining a desirable performance [35].

Data centers represent a significant slice of the world's consumed electrical energy [35], so there's a significant incentive to find better solutions to the current technological landscape.

Energy efficiency refers to the use of technologies and techniques that minimize the amount of energy required to perform a specific task or achieve a specific goal. This can be applied in various fields, one of them being information technology. Here we will discuss the energy efficiency related to the cloud computing context.

Despite all, there's a research gap in the "greenness" of the energy consumed: the energetic profile. The CO2 emissions represent harmful consequences to the planet and not all electrical energy is derived from polluting sources. The energetic profile of data centers may vary in their clean energy ratio. Renewable energy production changes dynamically during time. Atmospheric conditions and daylight hours affect its production. These variants might be taken into account for new research to delve into "greener" data centers with an environmental perspective, instead of a business one.

2.3.1 Data centers

Energy-efficient data centers is a key topic in the field of energy efficiency in cloud computing. The design of energy-efficient servers, storage devices, data

centers and the use of energy-efficient cooling and power systems can significantly reduce the energy consumption of cloud computing systems.

One aspect of energy-efficient hardware is the design of servers and storage devices that consume less power. This can be achieved through the use of energy-efficient hardware, like processors, memory and other components.

Another aspect of energy-efficient hardware is the design of data centers that consume less energy. This can be achieved through the use of energy-efficient cooling and power systems [23]. For example, using fresh air cooling, instead of traditional air conditioning units, can reduce the energy consumption of a data center. Additionally, using efficient power supply units, like 80 plus, that provide high-efficiency power conversion, can also help reducing the energy consumption.

Moreover, research in this area also looks into server and data center power management to reduce energy consumption through the use of dynamic voltage and frequency scaling and other techniques [23].

Overall, designing energy-efficient hardware and data centers is an important step in reducing the energy consumption of cloud computing systems which can help to decrease the carbon emissions and make the cloud more sustainable.

2.3.2 Virtualization and consolidation

Virtualization and consolidation are two key techniques that can be used to increase the energy efficiency of cloud computing systems by reducing the number of physical servers required to run a workload.

Virtualization is a technology that allows multiple virtual machines (VMs) to run on a single physical machine [15]. Each VM functions as a separate and isolated operating system environment with its own applications, resources and system settings. This means that a single machine is capable of running multiple container instances. This approach reduces the number of physical servers required to run a given workload and can lead to significant energy savings.

Consolidation is a technique that enables multiple servers to be combined into a single, more powerful server [15]. This can be done through the use of virtualization. The goal of consolidation is to reduce the number of physical servers required to run a workload, resulting in an energy consumption reduction.

The combination of virtualization and consolidation allows server management to efficiently manage and allocate resources to a large number of VMs running on a smaller number of physical servers. This can lead to a reduction in energy consumption and cooling needs as well as a decrease in the overall cost of operating a cloud computing infrastructure.

There are strategies that combine of virtualization and consolidation techniques to optimize the allocation of resources to virtual machines [15].

2.3.3 Renewable Energy Sources

Renewable energy sources like solar, wind and water are becoming increasingly important in the field of energy efficiency in cloud computing. The use

of renewable energy sources reduce the need for fossil fuels which consequently decreases the carbon emissions of cloud computing systems.

One way to integrate renewable energy sources into cloud computing systems is by using them to power data centers. For example, solar panels or wind turbines can be installed at a data center site to generate electricity which can then be used to power the data center.

Different Data Centers are fueled by different energetic profiles which can be exploited to achieve environment sustainability.

It's important to consider that the integration of renewable energy sources into a data center can also bring some challenges such as the high initial cost of equipment and installation and the need for energy storage to smooth out the variability of renewable energy sources.

3 Relevant Systems

In this section we provide an overview of current systems related to the paradigm being discussed. These systems contain resourceful material to support the perspective brought by this work itself.

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

"Let's Wait Awhile: How Temporal Workload Shifting Can Reduce Carbon Emissions in the Cloud" is a research paper that proposes a technique called temporal workload shifting to reduce carbon emissions in cloud computing systems.

The technique involves shifting workloads to times when renewable energy sources are more plentiful, in order to reduce the need to use fossil fuels. This can be done by scheduling jobs to run during periods of low demand or by using machine learning algorithms to predict and optimize energy usage.

This paper categorizes workloads in various aspects:

- Duration:
  - Short-Running Workloads: These workloads represent the majority of workloads executed in data centers. FaaS tasks are included in this category and usually don't offer great flexibility for shifting since its purpose is to meet SLAs and execute as fast as possible.
  - Long-Running Workloads: These workloads contain a high potential for shifting. They usually are very energy intensive and expect runtimes of several days. Usually representing machine learning or big data tasks, the deadlines are also moderately flexible.
  - Continuously Running Workloads: These are not further discussed in the paper since they do not have a deadline date by inherent design.
- Execution Time:

    - Ad Hoc Workloads: These workloads can include both FaaS and machine learning tasks. The scheduler should decide whether to postpone or start its execution immediately. Due to definition, these can only be taken into account with estimation techniques like forecasting.
    - Scheduled Workloads: Usually periodic backups or batches which can be taken into account beforehand. These contain a significant shifting potential due to the large time constraints.
- Interruptibility:
    - Interruptible Workloads: This possibility is usually only included in Long-Running Workloads and enables a high control over temporal shifting. The overhead of stopping and restarting the task is negligible.
    - Non-Interruptible Workloads: Some workloads don't contain the possibility of stopping and restarting due to the significant overhead. For example, FaaS functions are not interruptible due to relative overhead when compared to the full execution time.

The results achieved with temporal shifting when taking into consideration real world data are astounding. A 20% savings is credible in the regions tested.

3.2 An Experience-Based Scheme for Energy-SLA Balance in Cloud Data Centers

The research paper "An Experience-Based Scheme for Energy-SLA Balance in Cloud Data Centers" proposes a scheme for balancing energy consumption and Service Level Agreement (SLA) in cloud computing data centers. The authors propose an experience-based algorithm that uses historical data to predict future workloads and adjust the resource allocation accordingly.

The algorithm is divided into two stages: the learning stage and the prediction stage. During the learning stage, the algorithm analyzes historical workload data and builds a model to predict future workloads. In the prediction stage, the algorithm uses the model to predict future workloads and adjusts the resource allocation to balance energy consumption and support SLA constraints.

The authors evaluate the proposed scheme using a simulation of a cloud data center with a varying workload. The results show that the proposed scheme can effectively balance energy consumption and SLA, with a minimal SLA violation rate. The scheme also reduces energy consumption to very desirable levels.

3.3 A green energy optimized scheduling algorithm for cloud data centers

The research paper "A green energy optimized scheduling algorithm for cloud data centers" presents a scheduling architecture aiming to reduce energy consumption in cloud data centers. The authors also propose an certainty and uncertainty algorithm that enables live VM merging and reallocation with the same goal of saving energy.

In summary, the scheduling architecture consists of three phases: user, resource and scheduling. The resource phase is divided into VM level and host level, the VM layer is scaled up and down based on workload. The scheduling phase includes Scheduling Task Queue (STQ) and Non-completed Task Queue (NTQ) where new tasks, urgent tasks and waiting tasks are incorporated through the queueing mechanism. Tasks that are not completed in STQ are held in NTQ and are given priority by the scheduler. Urgent tasks are compared to previous tasks in the queue and are given priority over non-urgent tasks. Tasks that fail in STQ due to insufficient resources are transferred to NTQ for further scheduling.

According to the demands of the hosts in the data center, VMs are created and destroyed in the resource phase. Job allocation is done in the VMs based on the certainty and uncertainty algorithm. VMs can fail due to a shortage of resources in the data center. The paper goal also includes following Service Level Agreement (SLA) to ensure job allocation through VMs with the best assignment policy that manages to reduce energy consumption in the data center. The more refined the SLA, the better the job allocation in VMs is performed.

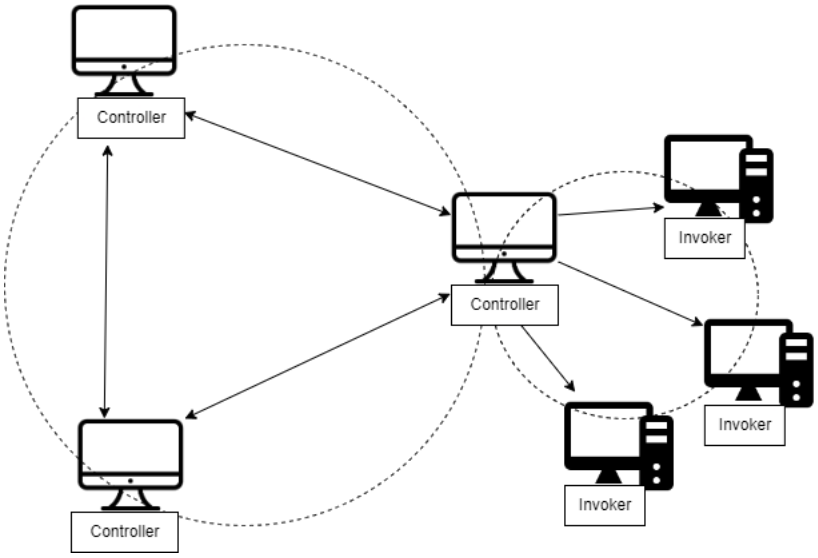
4 Architecture

This work proposes a thesis focused on exploring the challenging problem that is: a Green Function-As-A-Service.

My proposal aims to expand on the ecological perspective in the Function-As-A-Service context. It is going to explore scheduling algorithms and fine-tune its parameters through extensive testing. The testing should consider different layouts and ecological scenarios to obtain insightful results. The results are going to be weighted against traditional algorithms provided by OpenWhisk. All trade-offs should be considered when exploring the solution such as the financial perspective, communication latency, function execution time and function deployment time.

4.1 Distributed Architecture

Let us start with the global layout present in the following figure.



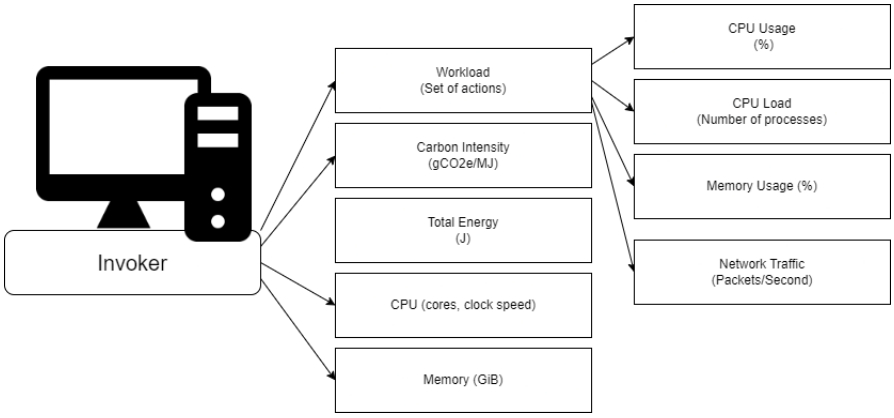
In this figure we can observe multiple nodes that form a decentralized net composed of Clusters, each one with a central Controller node and various Invoker nodes. Each node represents a device capable of computation and correctly integrated in the net. There are only two node categories: Controller and Invoker. These categories represent different responsibilities for the device in that same position in the net. Detailed descriptions are present in the following sub sections.

We will describe this architecture from a bottom-up perspective through the following order: Invoker, Controller, Cluster and Top Level View.

4.2 Invoker

In OpenWhisk, an invoker is a component that is responsible for the actual execution of actions. When a client sends a request to invoke an action, the controller schedules the action for execution on an available invoker. The invoker then retrieves the code for the action, loads it into memory and executes it. The invoker is also responsible for managing the resources allocated to the assigned workload such as CPU and memory. It is also responsible for monitoring the execution of the action. If the action exceeds its allocated resources, the invoker kills the action and returns an error to the client.

In this system, the Invoker node is adapted to the proposed architecture. he Invoker nodes are still responsible for running the actions, collecting the logs and metrics and returning the results to the client or Controller. They also communicate with the Controller node, by reporting the usage statistics of the invoker which can be used for monitoring and billing purposes. In this proposal, the metrics used are represented in the following figure.



The available parameters studied in the Invoker node are:

- Workload: A workload in an Invoker node refers to the set of actions that are being executed by the Invoker at any given time. The workload can be analyzed on various metrics, for example:
  - CPU Usage: represents the percentage of time that the CPU is busy executing instructions. A high CPU utilization indicates that the instance is heavily utilized and may be running at capacity which could lead to poor performance and slow response times. A low CPU utilization could indicate that the instance is underutilized and that the instance's work is not optimal.
  - CPU Load: it's measure of the number of processes that are waiting to be executed by the CPU. A high CPU load indicates that there are more processes waiting to be executed than the CPU can handle. A low CPU load, on the other hand, indicates that the CPU has enough resources to handle the current workload.
  - Memory usage: This metric measures the amount of memory that is being used by an instance or container. It is a good indicator of the workload on an instance or container and a high memory usage indicates that the instance or container may be running at capacity.
  - Network traffic: This metric measures the amount of network traffic that is being sent and received by an instance or container. It is a good indicator of the workload on an instance or container and a high network traffic indicates that the instance or container is heavily utilized.
- Carbon Intensity: The amount of carbon dioxide emitted per unit of energy. This can vary during time due to conditions involving the electricity source. This variable is updated regularly and communicated to the Controller so that it can be taken into the calculation for the scheduling algorithm. Expressed in grams of carbon dioxide per megajoule.
- Total Energy: The amount of energy spent during a specific time slice.
- CPU units: A CPU unit represents the ability to execute one thread of code at a time. The number of CPU units that are allocated to an instance or a container can affect the performance of the instance or container. A

higher number of CPU units means that the instance or container has more processing power available to it which can result in faster execution times for CPU-bound workloads. The amount of memory, I/O and network bandwidth also play an important role in the overall performance of the instance or container. It's also worth mentioning that different cloud provider may have different naming conventions and ways to define CPU units. As an example, Amazon Web Services (AWS) and Google Cloud Platform (GCP) use the term "vCPU" to refer to a virtual CPU core while in Azure it's called "Core".
- Memory units: The memory unit represents the capacity of an instance or container to store and access data in the form of random access memory (RAM). The amount of memory allocated to an instance or container can affect its performance. A higher amount of memory can result in faster execution times for memory-bound workloads and also allows more applications or processes to run at the same time. Most cloud providers use "GiB" (gibibytes) to represent memory unitswhich is the unit of memory in the International System of Units (SI). It's also worth mentioning that some providers also offer burstable instances or containers, in which instances or containers are guaranteed a baseline amount of memory and CPU, with the ability to burst above that baseline as additional resources become available.

The Invoker node also provides isolation between different actions, ensuring that one action does not interfere with another action.

One of the benefits of these nodes is that it allows OpenWhisk to scale horizontally, meaning that new nodes can be added to the system to handle increased load. Additionally, these can be automatically added or removed depending on the system's usage patterns, allowing the system to scale up or down as needed.

Overall, the Invoker node is a critical component in OpenWhisk that is responsible for executing actions and communicating important metrics to the Controller. It allows OpenWhisk to run actions in different environments and platforms and provides isolation between actions. It permits OpenWhisk to handle increased load by scaling out their number nodes.

4.3 Controller

The Controller node functionality consists in managing the lifecycle of actions, triggers and rules. The controller also manages the underlying infrastructure such as allocating resources for actions and monitoring their execution.

The controller is responsible for coordinating the execution of actions which are the basic units of computation in OpenWhisk. An action is a piece of code that can be invoked with a set of inputs and returns a set of outputs. The controller receives requests to invoke an action and then schedules the action for execution on an available invoker.

Triggers and rules are used to automatically invoke actions in response to certain events. A trigger is an event source, representing a change relevant to the execution of specific application. A rule is a mapping between a trigger and one or more actions. It specifies which actions should be invoked when a

specific trigger is fired. The controller is responsible for managing the lifecycle of triggers and rules. It is also responsible for making sure that the correct actions are invoked when a trigger is fired.

This work's adaptation consists in the following. The Controller also has the functionality of maintaining the systems' information of Invokers. When making the decision of where to distribute its attributed workload, it has two possible options: send to other Controllers node or send to one of its assigned Invoker nodes.

The approach to incorporate carbon intensity into a FaaS scheduling function is to use data on the carbon emissions of different Invoker nodes and factor that data into the decision-making process for where to provision a given function.

The Carbon Footprint (CFP) of an Invoker node is derived from the following:

- Total Energy: The total amount of energy that the machine consumes in a given time slice.
- Carbon Intensity (CI): Represents the amount of carbon dioxide released to the atmosphere per unit of energy is commonly expressed in grams of carbon dioxide equivalent per megajoule (gCO2e/MJ). For example, a data center located in a region with a high percentage of renewable energy sources such as wind or solar power, will have a carbon intensity really close to zero. A data center located in a region where the majority of energy is generated from fossil fuels, the value will be significantly higher [9].

$$CFP = E_{TOTAL} \times CI$$

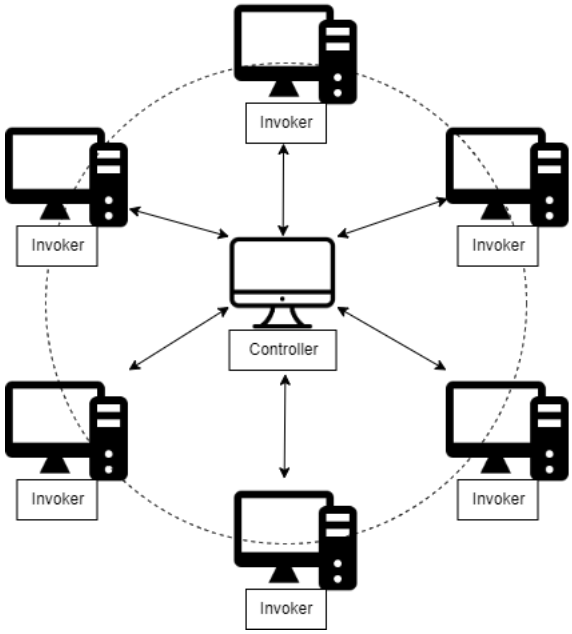
Each Invoker must also keep track of its own Workload (WL). The WL can be estimated with registered millions of instructions per second (MIPS).

$$WL \approx MIPS = \frac{InstructionCount}{ExecutionTime \times 10^6}$$

Conjugating these two metrics we can calculate a good estimation for an energetic profile in every Invoker. This energetic profile (EP) is defined as the carbon emissions related to the work realized. In this situation, the total energy taken into account when calculating the CFP should correspond to the same time slice in which the MIPS was measured.

$$EP = \frac{CFP}{WL}$$

4.4 Cluster

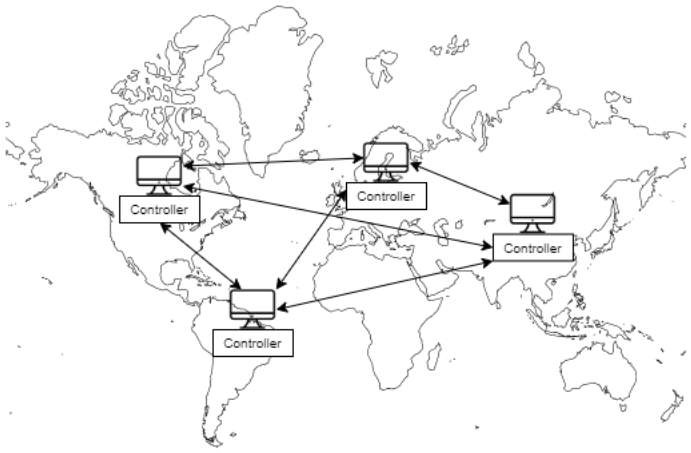


Each cluster is assumed to be geographically close forming a low latency net. The Controller knows all Invokers in this net and can integrate new Invokers. The devices fulfilling the Invoker role might be fueled by different energy sources, with different carbon footprint profiles. They are also are defined by their heterogeneity in all parameters: energy efficiency, computation power and memory capacity. The Controller must take into account all of these parameters when making the decision on how to distribute its assigned tasks.

The Controller should calculate its own Energetic Profile (EP) for its own Cluster and keep estimations of the EP for the others:

$$EP_{Cluster} = \frac{\sum EP_{Invoker} + EPCluster}{N_{Invoker} + 1}$$

4.5 Top Level View



At the end of our bottom-up description we can finally piece the whole proposed system together.

In this architecture, every Controller node knows every other Controller node. This system is not meant to achieve maximum scalability in the Controller hierarchy. In such a case where that could be needed, an additional hierarchical layer of Controllers could be created.

The main objective is to aim to achieve a better energetic profile through the available options in the net created. Assigning tasks to less carbon intensive devices while possible and doing so while carefully observing the all trade-offs.

One possible overhead will consist in the scheduling algorithm which represents the decision on whom to assign the functions to. Another possible overhead can exist due to the latency in sending batches of functions to other Clusters with a better ecological profile to execute them.

The geographical dispersion of Cluster intends to bring diverse ecological profiles for the available computation power. The geographical disparity will inherently bring changes in ecological efficiency with variants such as time and weather.

When taking the decision of which device to assign a function or whether to send it to other Cluster, the Controller (responsible for making the decision) must keep and estimate and update that estimate so that is doesn't overload the ideal candidate. There are multiple solutions to test when solving this problem:

- Increment the workload assigned to the Cluster that received the tasks sent with an effective estimation.
- Distribute the workload through the best candidates, not only focusing on one Cluster.

Any Workload can be sent to any Cluster and that Cluster's Controller should schedule it according to the best Energetic Profile (EP).

The approach proposed is described in Algorithm 1. This algorithm makes the scheduling decision of where to send the Workload received by a specific Cluster. The Workload can either be sent away to other Cluster in the case that there is another node with better energetic profile with relative low utilization. Else, the Workload must be executed locally which involves choosing the best Invoker for that matter. The Controller is unable to send Workload to another Cluster whose capacity is over the threshold. Additionally, the Controller must update upon each Workload sent with an estimation corresponding to the increase of the capacity threshold. The algorithm is based on the best effort principle. In this case the focus is not to ensure QoS terms.

There is also a principle of communication between Clusters to propagate the EP values of each one. This should enable eventual consistency so that the values are represented correctly, even when given long intervals of time. This communication step should also update the current capacity of each Cluster so that the estimations don't drift too far off from reality.

Other interesting option for the scheduler could factor in the idle resources conjugated with the energetic profile when making the decision of whom to send the workload too.

There's also room to expand with other ideas. For example, considering interrupting workloads and migrating workload when another cluster achieves a severely beneficial position.

Algorithm 1 Eco-Scheduler

$Cluster_{Best} \leftarrow Cluster_{Self}$
for $Cluster \in \mathcal{N}et$ **do**
  if $Capacity_{Cluster} < Threshold$ **then**
    $Cluster_{Best} \leftarrow \min(EP_{Cluster_{Best}}, EP_{Cluster})$
  end if
end for
if $Cluster_{Best}$ is $Cluster_{Self}$ **then**
  for $Invoker \in \mathcal{P}ool$ **do**
    if $Capacity_{Invoker} < Threshold$ **then**
      $Invoker_{Best} \leftarrow \min(EP_{Invoker_{Best}}, EP_{Invoker})$
    end if
  end for
  Send Workload to $Invoker$
  Update $EP_{Cluster_{Self}}$
  Update $Capacity_{Cluster_{Self}}$
else if
  thenSend Workload to $Cluster$
  Update $Capacity_{Cluster}$
end if

5 Evaluation Methodology

This section will describe the evaluation process proposed for this system. It aims to assess the effectiveness and environmental impact of Eco-FaaS through a systematic and structured approach.

5.1 Test-Bed

To achieve the desired results for testing the following steps will be performed:

1. Set up a test environment that simulates the desired ecosystem with a network of cluster like described in the previous chapters. Initially, this network will consist of a simulation [10], and then of a small local deployment. In further testing it can be augmented to a larger deployment with the use of specialized tools such as Testground.
2. Implement the Eco-FaaS scheduler and deploy it correctly to all the Controllers.
3. Test the scheduling process with different parameters given to the Cluster network.
4. Balance possible weights or threshold accordingly to the test results.
5. Iterate multiple times the last previous two steps.

The metrics collected from the testing process are: CPU Usage, Memory Usage, Allocation Success Rate (during the workload migration decision making), Function Execution Time, Function Allocation Time, Total Energy consumed, MIPS, Network Traffic. The following metrics can also be derived: Energy Efficiency (instruction/J) and Eco Efficiency (instruction/gCO2e).

5.2 Workloads and Data Sets

To extensively test the proposed the system, the following workloads will be executed.

- CPU Bound:
  - Rendering complex 3D graphics or animations.
  - Encrypting or decrypting data.
  - Performing image or video processing tasks.
  - Running machine learning algorithms.
- Memory Bound:
  - Database management.
  - Data analysis.

These workloads represent different possible needs for a FaaS system. We can expect interesting results analyzed through the use of the metrics defined in the previous sub section. Experiments will be carried out in desktop and server machines, as well as in cloud deployments or simulations [10].

5.3 Results

The expected results consist in a high ecological benefit to the environment. This benefit would be composed of a significant reduction in the production of carbon dioxide while executing the exact same load of work.

The inherent trade-off is expected to be divided in the communication latency and the execution latency. A desirable result would aim for a minimal increase in both these latencies.

6 Conclusion

Cloud computing is a rapidly growing technology that has revolutionized the way organizations access and manage their computing resources. Delving deeper into Cloud computing we find Function-As-A-Service: a service that enables serverless computing which is abstraction from the server management while providing function execution capabilities. This work presented Eco-FaaS, a scheduling approach to Function-As-A-Service from an ecological perspective. We explored the concept of Eco-FaaS and its potential to provide beneficial outcomes for both ecosystems and society. We also examined current related case studies and challenges in the implementation of Eco-FaaS and suggested potential solutions for overcoming these obstacles. In conclusion, this paper goal is to demonstrate the feasibility of Eco-FaaS as a tool for achieving sustainable scheduling in the Function-As-A-Service landscape.

References

1. Akkus, I.E., Chen, R., Rimac, I., Stein, M., Satzke, K., Beck, A., Aditya, P., Hilt, V.: Sand: Towards high-performance serverless computing. In: Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference. p. 923–935. USENIX ATC '18, USENIX Association, USA (2018)
2. Antelmi, A., D'Ambrosio, G., Petta, A., Serra, L., Spagnuolo, C.: A volunteer computing architecture for computational workflows on decentralized web. IEEE Access 10, 98993–99010 (2022)
3. Back, T., Andrikopoulos, V.: Using a microbenchmark to compare function as a service solutions. In: Kritikos, K., Plebani, P., de Paoli, F. (eds.) Service-Oriented and Cloud Computing. pp. 146–160. Springer International Publishing, Cham (2018)
4. Banaei, A., Sharifi, M.: Etas: predictive scheduling of functions on worker nodes of apache openwhisk platform. The Journal of Supercomputing 78(4), 5358–5393 (Mar 2022), https://doi.org/10.1007/s11227-021-04057-z
5. Barcelona-Pons, D., García-López, P., Ruiz, A., Gómez-Gómez, A., París, G., Sánchez-Artigas, M.: Faas orchestration of parallel workloads. In: Proceedings of the 5th International Workshop on Serverless Computing. p. 25–30. WOSC '19, Association for Computing Machinery, New York, NY, USA (2019), https://doi.org/10.1145/3366623.3368137
6. Bermbach, D., Karakaya, A.S., Buchholz, S.: Using application knowledge to reduce cold starts in faas services. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing. p. 134–143. SAC '20, Association for Computing Machinery, New York, NY, USA (2020), https://doi.org/10.1145/3341105.3373909

7. Costa, B., Bachiega Jr, J., Araújo, A.: Orchestration in fog computing: A comprehensive survey. ACM Computing Surveys 55, 1–34 (01 2022)
8. De Palma, G., Giallorenzo, S., Mauro, J., Trentin, M., Zavattaro, G.: A declarative approach to topology-aware serverless function-execution scheduling. In: 2022 IEEE International Conference on Web Services (ICWS). pp. 337–342 (2022)
9. Eilam, T.: Towards transparent and trustworthy cloud carbon accounting. In: Proceedings of the 22nd International Middleware Conference: Extended Abstracts. p. 1–5. Middleware '21, Association for Computing Machinery, New York, NY, USA (2021), https://doi.org/10.1145/3501255.3501408
10. Kathiravelu, P., Veiga, L.: Concurrent and distributed cloudsim simulations. In: 2014 IEEE 22nd International Symposium on Modelling, Analysis Simulation of Computer and Telecommunication Systems. pp. 490–493 (2014)
11. Kaur, N., Sood, S.K.: An energy-efficient architecture for the internet of things (iot). IEEE Systems Journal 11(2), 796–805 (2017)
12. Lavoie, E., Hendren, L.: Personal volunteer computing. In: Proceedings of the 16th ACM International Conference on Computing Frontiers. p. 240–246. CF '19, Association for Computing Machinery, New York, NY, USA (2019), https://doi.org/10.1145/3310273.3322819
13. Lavoie, E., Hendren, L., Desprez, F., Correia, M.: Genet: A quickly scalable fat-tree overlay for personal volunteer computing using webrtc. In: 2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO). pp. 117–126 (2019)
14. Lavoie, E., Hendren, L., Desprez, F., Correia, M.: Pando: Personal volunteer computing in browsers. pp. 96–109 (12 2019)
15. Lin, W., Wu, W., He, L.: An on-line virtual machine consolidation strategy for dual improvement in performance and energy conservation of server clusters in cloud data centers. IEEE Transactions on Services Computing 15(2), 766–777 (2022)
16. Mendes, S., Simão, J., Veiga, L.: Oversubscribing micro-clouds with energy-aware containers scheduling. In: Hung, C., Papadopoulos, G.A. (eds.) Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019. pp. 130–137. ACM (2019), https://doi.org/10.1145/3297280.3297295
17. Mohanty, S.K., Premsankar, G., di Francesco, M.: An evaluation of open source serverless computing frameworks. In: 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). pp. 115–120 (2018)
18. Patros, P., Spillner, J., Papadopoulos, A.V., Varghese, B., Rana, O., Dustdar, S.: Toward sustainable serverless computing. IEEE Internet Computing 25(6), 42–50 (2021)
19. Quevedo, S., Merchán, F., Rivadeneira, R., Dominguez, F.X.: Evaluating apache openwhisk - faas. In: 2019 IEEE Fourth Ecuador Technical Chapters Meeting (ETCM). pp. 1–5 (2019)
20. Russo, G.R., Milani, A., Iannucci, S., Cardellini, V.: Towards qos-aware function composition scheduling in apache openwhisk. In: 2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops). pp. 693–698 (2022)
21. Selimi, M., Cerdà-Alabern, L., Freitag, F., Veiga, L., Sathiaseelan, A., Crowcroft, J.: A lightweight service placement approach for community network micro-clouds. Journal of Grid Computing 17 (03 2019)
22. Shahrad, M., Balkind, J., Wentzlaff, D.: Architectural implications of function-as-a-service computing. In: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture. p. 1063–1075. MICRO '52, Association for

Computing Machinery, New York, NY, USA (2019), https://doi.org/10.1145/3352460.3358296

23. Sharifi, L., Cerdà-Alabern, L., Freitag, F., Veiga, L.: Energy efficient cloud service provisioning: Keeping data center granularity in perspective. J. Grid Comput. 14(2), 299–325 (2016), https://doi.org/10.1007/s10723-015-9358-3
24. Sharifi, L., Freitag, F., Veiga, L.: ARTA: an economic middleware to exchange pervasive energy and computing resources. In: 2016 IEEE International Conference on Smart Grid Communications, SmartGridComm 2016, Sydney, Australia, November 6-9, 2016. pp. 478–483. IEEE (2016), https://doi.org/10.1109/SmartGridComm.2016.7778807
25. Silva, J.N., Ferreira, P., Veiga, L.: Service and resource discovery in cycle-sharing environments with a utility algebra. In: 2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS). pp. 1–11 (2010)
26. Stein, M.: The serverless scheduling problem and noah. ArXiv abs/1809.06100 (2018)
27. Suresh, A., Somashekar, G., Varadarajan, A., Kakarla, V.R., Upadhyay, H., Gandhi, A.: Ensure: Efficient scheduling and autonomous resource management in serverless environments. In: 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS). pp. 1–10 (2020)
28. Trach, B., Oleksenko, O., Gregor, F., Bhatotia, P., Fetzer, C.: Clemmys: Towards secure remote execution in faas. In: Proceedings of the 12th ACM International Conference on Systems and Storage. p. 44–54. SYSTOR '19, Association for Computing Machinery, New York, NY, USA (2019), https://doi.org/10.1145/3319647.3325835
29. Vahidinia, P., Farahani, B., Aliee, F.S.: Mitigating cold start problem in serverless computing: A reinforcement learning approach. IEEE Internet of Things Journal pp. 1–1 (2022)
30. Wang, Z., Goudarzi, M., Aryal, J., Buyya, R.: Container orchestration in edge and fog computing environments for real-time iot applications. In: Buyya, R., Hernandez, S.M., Kovvur, R.M.R., Sarma, T.H. (eds.) Computational Intelligence and Data Analytics. pp. 1–21. Springer Nature Singapore, Singapore (2023)
31. Wiesner, P., Behnke, I., Scheinert, D., Gontarska, K., Thamsen, L.: Let's wait awhile: How temporal workload shifting can reduce carbon emissions in the cloud. In: Proceedings of the 22nd International Middleware Conference. p. 260–272. Middleware '21, Association for Computing Machinery, New York, NY, USA (2021), https://doi.org/10.1145/3464298.3493399
32. Wu, S., Tao, Z., Fan, H., Huang, Z., Zhang, X., Jin, H., Yu, C., Cao, C.: Container lifecycle-aware scheduling for serverless computing. Software: Practice and Experience 52 (07 2021)
33. Xie, R., Tang, Q., Qiao, S., Zhu, H., Yu, F.R., Huang, T.: When serverless computing meets edge computing: Architecture, challenges, and open issues. IEEE Wireless Communications 28(5), 126–133 (2021)
34. Yu, H., Irissappane, A.A., Wang, H., Lloyd, W.J.: Faasrank: Learning to schedule functions in serverless platforms. In: 2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS). pp. 31–40 (2021)
35. Zhou, X., Li, K., Liu, C., Li, K.: An experience-based scheme for energy-sla balance in cloud data centers. IEEE Access 7, 23500–23513 (2019)

A Schedule

Tasks	**Duration**
Design	1 month
Architecture deployment	1.5 months
Scheduler development	1.5 months
Evaluation and testing	0.5 months
Optimization	0.5 months
Dissertation writing	2 months