FOA-Energy: A Multi-objective Energy-Aware Scheduling Policy for Serverless-based Edge-Cloud Continuum

Anderson Andrei Da Silva, Yiannis Georgiou, Michael Mercier, Gregory Mounié, Denis Trystram

▶ **To cite this version:**

Anderson Andrei Da Silva, Yiannis Georgiou, Michael Mercier, Gregory Mounié, Denis Trystram. FOA-Energy: A Multi-objective Energy-Aware Scheduling Policy for Serverless-based Edge-Cloud Continuum. SAC 2025 - 40th ACM/SIGAPP Symposium on Applied Computing, Mar 2025, Catania International Airport Catania Italy, Italy. pp.225-232, ⟨10.1145/3672608.3707941⟩. ⟨hal-05156857⟩

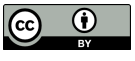
HAL Id: hal-05156857

https://hal.science/hal-05156857v1

Submitted on 10 Jul 2025

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons CC BY 4.0 - Attribution - International License

FOA-Energy: A Multi-objective Energy-Aware Scheduling Policy for Serverless-based Edge-Cloud Continuum

Anderson Andrei Da Silva*
Hewlett Packard Labs
Milpitas, United States of America
silva.andersonandrei@gmail.com

Yiannis Georgiou
Michael Mercier
Ryax Technologies
Lyon, France
firstname.lastname@ryax.tech

Gregory Mounié
Denis Trystram
Univ. Grenoble Alpes, CNRS, Inria,
Grenoble INP, LIG
Grenoble, France
firstname.lastname@imag.fr

Abstract

The cloud is evolving into a computing continuum by extending its capabilities toward the edge. This continuum better addresses the needs of modern applications, but it also introduces new challenges, particularly in resource management and scheduling. Serverless is a driving force in consolidating the continuum, allowing quick adaptations toward the edge level while keeping the applications' footprints low. Data-centric applications that deal with massive data and require deploying large software environments are becoming a common use-case for the combination of these new technologies. Standard cloud scheduling policies are based on greedy algorithms that do not efficiently handle platforms' heterogeneity nor deal with problems such as cold start delays. In this paper, we address these issues by extending a methodology to investigate serverless platforms on the edge-cloud continuum, and to study new scheduling policies in simulated environments. We also propose a multi-objective algorithm to allocate serverless functions in the continuum while considering heterogeneity to optimize energy consumption, data transfers, makespan, and resource utilization. As a baseline, we are inspired by a standard greedy algorithm from a widely used platform, Kubernetes. Our approach outperforms the baseline regarding energy consumption, data transfers, makespan, and resource utilization by up to three orders of magnitude.

CCS Concepts

• **Computing methodologies** → **Modeling and simulation**; • **Theory of computation** → **Scheduling algorithms**; • **Hardware** → **Power estimation and optimization**.

Keywords

Scheduling Policies, Serverless Computing, Energy Consumption, Edge-Cloud Continuum, Heterogeneous Platforms

*This paper was submitted by this author while he was a Post-Doctoral Research Engineer at Hewlett Packard Labs, but the work was previously done while he was a PhD student attached to the Univ. Grenoble Alpes and Ryax Technologies.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SAC '25, March 31-April 4, 2025, Catania, Italy
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0629-5/25/03
https://doi.org/10.1145/3672608.3707941

ACM Reference Format:
Anderson Andrei Da Silva, Yiannis Georgiou, Michael Mercier, and Gregory Mounié, Denis Trystram. 2025. FOA-Energy: A Multi-objective Energy-Aware Scheduling Policy for Serverless-based Edge-Cloud Continuum. In *The 40th ACM/SIGAPP Symposium on Applied Computing (SAC '25), March 31-April 4, 2025, Catania, Italy*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3672608.3707941

1 Introduction

Cloud computing has been overwhelmed by the demands of new-generation applications, characterized by massive data, shorter execution times, and security and privacy vulnerabilities. Edge computing has emerged as a complement to cloud computing, with their integration, alongside intermediate infrastructures and multi-cluster scenarios, forming the paradigm of an edge-cloud computing continuum [33]. The edge-cloud continuum, illustrated in Figure 1, interconnects multi-layer infrastructures as cloud clusters, edge clusters and edge resources. Such interconnection of different layers enhance the use of heterogeneous resources while keeping a unified control through common abstractions and mechanisms per cluster. The continuum addresses the demands of next-generation applications but introduces new challenges and complexities, particularly in management and scheduling, to handle such heterogeneous, multi-layered infrastructures.

In parallel, serverless computing has emerged as an innovative paradigm for programming and deploying applications in the cloud [25]. It enables the seamless deployment and execution of fast-running functions, optimizing resource usage through improved scaling mechanisms, efficient management of concurrent requests, and smoother adaptation to heterogeneous resources at the edge of the continuum. Nevertheless, serverless computing still requires adaptations to manage massive data streaming applications and complex software environments, such as those used in high-performance computing or artificial intelligence [22]. Cold start delays happen when there are no instances of the required environment ready for use, forcing a full download and deployment, causing significant overheads on functions' execution. In the context of this paper, these environments are containers, and their deployment can cause an overhead of up to 20x of the platform slowdown, with cold start delays up to 10x longer [34]. Containers, being composed of layers, can leverage layer-sharing mechanisms or caching to accelerate their deployment [32]. In this paper, we exploit these facts to speed up functions' deployment, and thus, reduce cold start delay. To maintain Quality of Services (QoS), scheduling and management mechanisms must minimize function completion time, known as makespan, prevent bandwidth saturation, and

support batch processing and data aggregation. While digital applications are becoming more dynamic, the amount of data generated at the edge by IoT devices continues to grow. Therefore, on one hand serverless can handle such dynamism and local resource management, but on the other hand, the edge-cloud continuum requires a better management of heterogeneous multi-layered platforms.

Nowadays, environmental and ecological aspects such as carbon emissions and energy consumption has become a crucial societal topic. The International Energy Agency (IAE) estimates that data centers consume 1% of the total power generated in the world [9]. One of the objectives of this paper is to reduce the energy consumption of serverless platforms. However, since users only have access to virtual environments with pre-defined settings, rather than physical machines, it is challenging to measure the energy consumption of serverless functions. Energy observability tools such as Kepler [16] and Scaphandre [6] can be used, but they mostly rely on estimations in scenarios with virtual machines. Therefore, we investigate solutions for accessing accurate measurements relying on physical sensors and bare-metal platforms.

Figure 1 summarizes our main motivations which rely on reducing energy consumption, makespan and data transfers of batches of functions on serverless platforms deployed in the continuum. In this paper, we extend the infrastructure of simulations of serverless in the edge-cloud continuum from the work of Angelelli et al. [17], by integrating the measurement, analyses and simulation of the energy consumption of serverless functions. We use that environment to evaluate our new scheduling policies. We propose a multi-objective scheduling policy, called *FOA*-energy, that enables the allocation of batches of serverless functions. The scheduling policy aims to minimize energy consumption, makespan, and data transfers. We demonstrate that accounting for platform heterogeneity during the scheduling phase significantly enhances the efficiency of serverless platforms in the edge-cloud continuum.*FOA*-energy outperforms our baseline for energy consumption, makespan, data transfers, and number of machines used by up to three orders of magnitude. We enlist our main contributions as:

- The proposition of *FOA*-energy, a two-level multi-objective scheduling policy for the edge-cloud continuum, which improves energy consumption, makespan, data transfers, and resources usage compared to a baseline greedy-based policy;
- The extension of an evaluation methodology, with reproducible setup and artifacts, that can be extended to other serverless-based heterogeneous edge-cloud platforms [10];
- The study of energy consumption on serverless platforms, though bare-metal infrastructures with physical sensors, or cloud-designed tools, going one step further in this direction;
- A serverless-function energy-based benchmark that evaluates several combinations of heterogeneous machines.

The remainder of this paper is organized as follows: In Section 2 we present the background and literature review.In Section 3 we present our multi-objective scheduling algorithm. In Section 4 we present our experimental methodology to study serverless functions, platforms, and scheduling policies. In Section 5 we present our experimental results, and in Section 6, we present general discussions and our conclusions.

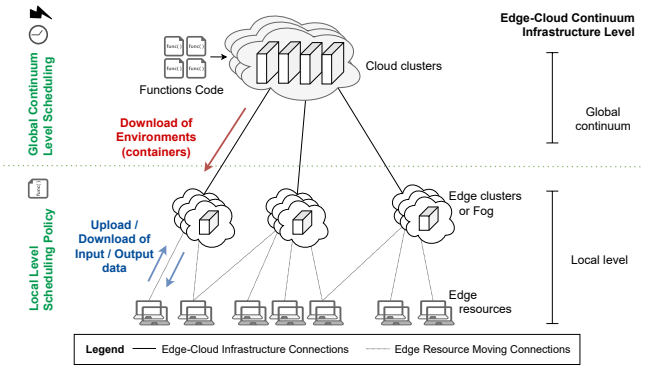


Figure 1: Two levels of scheduling in the Cloud Continuum.

2 Background and Related Works

Serverless computing has emerged as a new paradigm of abstraction, platform, and implementation of cloud functions [19, 20, 25]. However, the edge-cloud continuum still presents several challenges in such a multi-layered heterogeneous infrastructure that serverless alone can not address to that. For instance, platforms migration [37], data storage and management [28], and scheduling policies [30]. The serverless computing paradigm comprises the Function as a Service (FaaS) concept, avoiding server infrastructure management by automatically handling and deploying functions environments. As these environments can be containers, and Kubernetes is a well-known tool that manages containerized applications, it can be of great support for serverless. Kubernetes was not developed for serverless, but for microservices based workloads. However, serverless platforms such as Kubeless [2], OpenWhisk [3], and OpenFaas [8], were developed on top of Kubernetes. Also, it is possible to use external schedulers on Kubernetes such as, YuniKorn [7], IBM Safe Scheduler [5], and IBM Multiple Cluster Dispatcher [4]

To develop our scheduling policies, we were inspired by the dual-approximation algorithm of Shmoys and Tardos [35]. It assigns independent tasks to unrelated machines and bounds their approximation to, at most, the cost and twice the makespan of the optimal solution. Their linear program gets an assignment of tasks to machines, and their integral matching processes assure that their solution is integral. The properties and existence of such a matching are detailed in depth by Plummer and Lovász [31]. Rausch et al. proposed a container scheduling system, called Skippy, that enables serverless frameworks to support edge functions [32]. With Pagurus, Li et al. reduced cold startup [29] using a runtime container management system that comprises an inter-action container scheduler that schedules shared containers among actions. Suresh and Gandhi used OpenWhisk to implement a scheduling policy, named FnSched, which focuses on cost (resources) reduction [36], by considering the regulation of how much CPU the instances will use, and a defined latency ratio verified over time. Aumala et al. proposed PAScha, a package-aware scheduler that considers the affinity of packages needed to execute a function [18]. Differently than [18], we deal with the functions' requirement at the container-level, and differently from [29, 32, 36], with *FOA*-energy's input,

we schedule all functions at once, without consuming online scheduling time for updating parameters or variables, and we maximize the sharing of containers without compromising makespan, and reducing the energy consumption of the entire platform. We do not gather historical data as it is done in [38], however, we assume that the performances of all functions in all machines is already known or collected in advance. In addition, our approach makes our algorithm available for any serverless workload and functions.

We consider simulation a key practice to study different scheduling policies on serverless, we can save resources and more easily perform large experiments campaigns. For this study, we consider benchmarks designed for studying serverless [26, 27]. As a simulator, we used Batsim [24], a resource and job management system simulator designed on top of SimGrid [23].

3 A Two-level Multi-Objective Scheduling Policy for Serverless

Our multi-objective scheduling policy, *FOA*-energy, aims to minimize energy consumption, makespan, and data transfers of batches of serverless functions. *FOA*-energy works by tackling two different levels of the continuum: the global continuum and the local level. Figure 1 illustrates the edge-cloud continuum and the different objectives we tackle. At the global level, we apply our scheduling algorithm to batches of serverless functions that are submitted to the platform to select the cluster that minimize the objectives. At the local level, once the cluster executing the functions is chosen, we use different well-known scheduling policies to decide which machine is going to perform the computation. These scheduling policies are FirstFit, LargestFirst, SmallestFirst and FCFS (First Come First Serve). Regarding data transfers, we try to minimize the container images download required by the functions, here called environment. For that, we choose to execute in the same machine the functions that require the same container image or different images sharing common layers. This technique, along with minimizing the amount of data transferred for function I/O, speed up functions' initialization and thus reduces cold start delay which impacts the total execution time.

3.1 *FOA*-energy

Our scheduling policy named *FOA*-energy (Function Orchestration Algorithm, version Energy) has three objectives, to reduce energy consumption, data transfers, and makespan. For the remainder of this section, we call the energy consumption as cost, meaning that it is the cost variable treated in our algorithm. Figure 2 illustrates all steps that compose *FOA*-energy. Step *1. Linear Program* uses function and environment data to produce a fractional function schedule. Then, step *2. Minimum Cost Integral Matching* converts the fractional function schedule into an integral function schedule. Then, step *3. Local-level Scheduling Policy* uses the scheduling decision taken regarding the clusters, and processes the allocation of the functions to the machines inside each cluster. Finally, step *4. Container Layer Download Optimization* reflect the container layers sharing mechanism introduced by Docker [13] and handled by the Container Runtime in Kubernetes, for each function that arrives in the machines. The *Binary Search Looping Controller* component, illustrated in Figure 2, decides between re-executing steps 1 and 2 to get better performance, or to go to step 3. The set of results

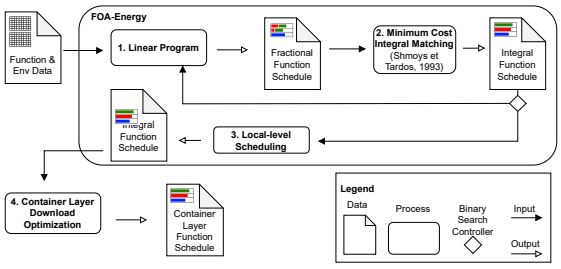


Figure 2: *FOA*-Energy's algorithm step by step.

N	Description
T	The makespan constraint
N	Number of scheduled functions
H	Number of clusters (heterogeneous classes)
M	Number of machines
K	Number of container environments
c_{ih}	Energy cons. (cost) of the *i-th* function on the *h-th* cluster
$\tilde{c}_{kh}$	Energy cons. (cost) of the *k-th* environment on the *h-th* cluster
p_{ih}	Processing time of the *i-th* function on the *h-th* cluster
$\tilde{p}_{kh}$	Processing time of the *k-th* env. on *h-th* cluster
env_i	Environment id of the *i-th* function
x_{ih}	Placement of the *i-th* function on the *h-th* cluster
y_{kh}	Placement of the *k-th* env. on the *h-th* cluster
m_h	Number of machines on the *h-th* cluster

Table 1: *FOA*-Energy's list of notation and descriptions.

by then characterizes a trade-off between cost and makespan, in a Pareto Front. Empirically, eight iterations are sufficient to study such a Pareto Front.

(i) *FOA*-energy's Linear Program: *FOA*-energy' linear program minimizes the energy consumption of the entire workload, under a constraint on the makespan, of an arbitrary value T. *FOA*-energy works under the following **assumptions**: a) functions depend on environments and all dependencies are known; b) functions and environments are independent between themselves and known in advance; c) their energy consumption, and execution time on each machine are known; d) the number of machines per cluster for all clusters is known. *FOA*-energy expects the following **inputs**: 1) environment energy consumption: the energy consumption to download and deploy functions environments (containers), in joules (J); 2) environment execution time: the time to download and deploy functions environments (containers), in seconds (s); 3) function energy consumption: the energy consumption of serverless functions, in joules (J); 4) function execution time: the execution time of serverless functions, in seconds. *FOA*-energy's linear program uses the **list of variables** and **notations** presented in Table 1.

Equation 1 describes *FOA*-energy's objective function, which minimizes the energy consumption (cost) of executing the functions and deploying their environments:

$$Min \sum_{i=1}^{N} \sum_{h=1}^{H} c_{ih} \times x_{ih} + \sum_{k=1}^{K} \sum_{h=1}^{H} \tilde{c}_{kh} \times y_{kh} \quad (1)$$

Each cluster h may execute several identical environments, but only one per machine:

$$y_{kh} = 0..m_h \quad (2)$$

For all functions i, each one is executed at least once by one of the clusters h:

$$\forall i, \sum_{h=1}^{H} x_{ih} \geq 1 \tag{3}$$

For each cluster h, the cluster does not process more than $m_h \times T$ because of the constraint T per machine inside the cluster. Thus, in total, the solution does not exceed $T \times m_h$, where m_h is the number of machines on cluster h:

$$\forall h, \sum_{i=1}^{N} p_{ih} \times x_{ih} + \sum_{k=1}^{k} \tilde{p}_{kh} \times y_{kh} \leq m_h \times T \tag{4}$$

For all clusters h and environments k, the amount of processing we can do $p_{ih} \times x_{ih}$, for each environment i $s.t$ $env_i = k$, is smaller or equal to the number of machines y_{kh} times the available time T in each machine after the deployment of the environment $\tilde{p}_{kh}$:

$$\forall h, \forall k, \sum_{\forall i \ s.t \ env_i = k} p_{ih} \times x_{ih} \leq y_{kh} \times (T - \tilde{p}_{kh}) \tag{5}$$

(ii) *FOA*-energy's Minimum Cost Integral Matching: Shmoys and Tardos [35] show that an X-perfect fractional matching from an LP solution can be converted efficiently to an X-perfect integral matching. We use this algorithm to convert the fraction solution of our LP into an integral scheduling solution.

4 A Methodology for Experimental Campaigns

For the experimental campaign, we perform the following steps: (i) measure containers' deployment time, functions' execution time, and energy consumption in bare-metal platforms; (ii) model simulations of serverless platforms; and (iii) run our experiments. We design a set of experiments, which is presented in Table 2. We execute all scenarios with all *FOA*-energy based scheduling policies and *K8S ImageLocality*. In total, we performed 1350 experiments.

4.1 Measuring Resources Utilization

In a bare-metal infrastructure, on top of GRID5000 [11], we deploy the serverless platform OpenWhisk [3], and execute an adapted version of the serverless functions from FunctionBench [26]. For each type of function adapted, we execute them using two different inputs, as described in Table 3. We measure their CPU and memory usage, as well as the execution time and total energy consumption of batches of serverless functions. To avoid interferences, we reserve the GRID5000 nodes entirely to be sure that no other tasks or processes are running in parallel with our experiments. For each function studied, we repeat its execution 5 times. In our setup, OpenWhisk is deployed on top of Kubernetes, and both run in the bare-metal platform GRID5000. As each one exports its own metrics, we do our measurements in three different steps:

(i) Measuring Containers' Deployment Time, CPU and Memory Usage: Kubernetes exports information about containers (including deployment time) in their computing units (Pods). To measure and visualize metrics such as CPU and memory usage, we deploy Prometheus [1] and Grafana [14] on top of Kubernetes.

(ii) Measuring Functions' Execution Time: We instrument the functions from our selected benchmark to measure their execution time. Also, the OpenWhisk platform provides timestamps per event and function. In the end of (i) and (ii), we cross the timestamps of OpenWhisk and Kubernetes to know when a pod is created, when its container is ready, when a function is executed, and when a pod is destroyed.

(iii) Measuring Energy Consumption: GRID5000 provides an API, called Kwollect[12], that scrapes the instant power consumption captured by physical wattmeters. However, Kwollect does not separate the power consumption per process or function that has been executed. For that reason, (a) we reserve entirely the nodes we use, and (b) we also retrieve the logs of OpenWhisk to have the timestamps of each function execution. With (b) we can merge both data, coming from Kwollect and OpenWhisk, to know the function responsible for the respective power measurements collected. Since OpenWhisk requires a minimum of two nodes, one to be a controller and another to be a worker, we analyze their energy consumption separately. Note that OpenWhisk controller nodes have no functions being executed on them, but worker nodes runs functions along with management services (network, metrics, ...). For simplicity, we sum the energy consumption of management services and the functions. To perform such measurements, we run the experiments on seven different clusters of GRID5000, located in two different sites in France: Grenoble and Lyon. Table 4 shows their number of nodes. A whole configuration description can be found in its documentation [11]. We choose these clusters because of the presence of the wattmeters on them. Finally, we compute the energy consumption (in Joules) for each functions as the integral of the instant power consumption (in watts) over their execution time through the Trapezoidal Rule [21].

4.2 Modeling Simulations of Serverless

Following the simulated environment described in [17], we use the Batsim/ Simgrid simulator [23], the description of machines from GRID5000 [11], and we model workloads based on the FunctionBench [26] benchmark. We extend this methodology by adding the energy consumption measurements collected with the bare-metal infrastructure to our models of platform and workloads. As a baseline, we made a scheduling policy inspired by Kubernetes called *K8S ImageLocality*.

(i) Modeling Workloads: we model our workloads as a set of different combinations of functions and inputs, as described in Table 3. In addition, each function has its required container layers' composition attached to the generated workloads to reproduce the layers' sharing mechanism in simulation. Finally, following Batsim/ Simgrid requirements, we estimate the number of floating-point operations (flops) necessary to simulate each function.

(ii) Modeling Platforms: we model the platforms by describing the nodes' CPU speed for different sets of clusters, following GRID5000's configuration [11]. Each platform represents a variation of our bare-metal scenario, which comprises a fixed set of 7 different clusters, with different number of machines. We vary the number of machines available on each cluster, as presented in Table 4. The baseline platform set-up represents exactly our bare-metal scenario, having 100% of the platform size.

(iii) A Model for Container Layer Download Optimization: we keep a list of container layers that are cached in each machine. We use this information to verify, per function, during scheduling decision, which layer can be re-used.

(iv) A Container Locality Baseline Policy: our baseline is inspired by the Kubernetes scheduler, more specifically, by the Image Locality plugin [15], that selects the machines to execute the functions by first scoring all available machines based on the

Parameters	Values
Workload Size	200, 600, 1000 functions
Fixed Clusters	7 clusters
Platform Size	14, 30, 57 machines
Scheduling Policies	FOA-energy + {FirstFit, SmallestFirst, LargestFirst, FCFS}, FCFS + K8SImgLocality
Random Seeds	30 seeds

Table 2: Design of Experiments, covering 1350 experiments.

Function	Input Values	Unit	Container Size
Chameleon	2.10^3, 3.10^3	Matrix size	170 MiB
Face Detect.	30, 50	Vid. Size (MiB)	2.46 GiB
Float Oper.	10.10^7, 20.10^7	Operations	155 MiB
Image Proc.	40, 60	Img size (MiB)	2.45 GiB
Linpack	5.10^3, 6.10^3	Matrix size	214 MiB
Model T.	50, 100	Dataset size	2.45 GiB
Pyaes	3.10^3, 4.10^3	Length of msg.	170 MiB
Video Proc.	30, 50	Vid. size (MiB)	2.46 GiB

Table 3: Functions adapted from FunctionBench [26].

Site/ Cluster	#Nodes	P.25%	P.50%	P.100%
Grenoble/ Troll	4	1	2	4
Grenoble/ Yeti	4	1	2	4
Lyon/ Gemini	2	2	2	2
Lyon/ Neowise	10	3	5	10
Lyon/ Nova	22	6	11	22
Lyon/ Orion	3	2	2	3
Lyon/ Taurus	12	4	6	12
Total of machines	57	21	30	57

Table 4: Experimental platforms' composition, representing 25%, 50%, and 100% of GRID5000 available nodes per cluster.

presence of the required container, and then selecting the highest scored one. Finally, to be comparable with *FOA*-energy, we adapt this algorithm to operate in the two levels of the continuum: it uses a FCFS policy for the global continuum and its container image locality plugin for the local level.

5 Experimental Results

We present the experimental results in two main parts. The first one concerns the measurements of energy consumption on serverless platforms in the edge cloud continuum. The second part concerns the impacts of scheduling policies applied to the continuum, investigated through simulations. Such impacts are depicted in terms of energy consumption, makespan, amount of data downloaded, and scheduling time.

5.1 Measuring Energy Consumption

Figures 3 and 4 present the instant power consumption of different clusters. The x-axis represents the timestamp, the y-axis represents the instant power measurements in Watts. In Figure 3, the colors represent the different clusters, while in Figure 4, the colors represent the different functions, and the facets (squares) represent the different clusters, but not all clusters are illustrated because of similar results.

(i) Power Measurements of Controller Nodes: We observe in Figure 3 that the results of *nova* are similar to *taurus* therefore they are overlapped. As the controllers manage the platform there are several services running together but we represent all of them

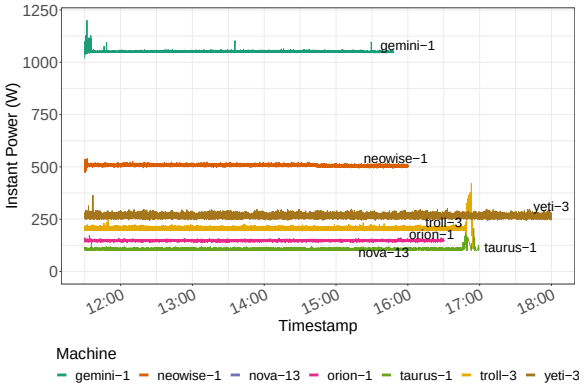


Figure 3: Instant power measurements (in Watts) of serverless platforms controller nodes in our bare-metal setup.

as one unique curve in the figure. We emphasize that this analysis is an important step to reduce overall energy consumption, once it becomes clear that different clusters consume different ranges of instant power to deploy and run the same serverless platform. For instance, the difference between *gemini* and *nova* is of 10x. We understand that these differences in performance are due to different hardware configurations. For instance, *gemini* is the only cluster in our set-ups that has GPU and their consumptions are taken into account even if the functions are not using them.

(ii) Power Measurements of Worker Nodes: We execute different kinds of functions to investigate if it would be possible to profile them. In fact, in Figure 4, it is possible to see identical behaviors for each function in the different clusters. However, the range of instant power changes across the clusters. Therefore, we understand that profiling the functions is possible, but it should happen per cluster. We remark that in the context of serverless computing, there are several events that can interfere with the execution of functions, such as services in the management of the platform failing and restarting, network connection failing, container download or deployment failing, a function reaching a time or resource limit, and others. For instance, an entire sequence of *chameleon* functions fail in *yeti-4* around 13:30. Trying to understand these failures, we saw that both functions *chameleon* and *facedetection* have a few successful executions before the failures, so their respective containers are already deployed. Also, the platform does not present any different behavior during their executions. Therefore, we find out that failures occur due to time or resource limits. Maybe resources used by the previous functions are not released on time before the executions start.

(iii) Measuring Energy Consumption of Functions: once we studied the behavior of each kind of function, we profile their duration and energy consumption, considering five repetitions. Figure 5 illustrates the median of the functions' execution time (in seconds) and the energy consumption (in W/h) of each function on each one of the clusters, with minimums and maximums indicated. It was expected that different clusters would perform differently for both metrics, but interestingly, not necessarily the most efficient machine for execution time is the most efficient for energy consumption. For instance, *chameleon_2000* runs faster in *neowise*, but does not present the lowest energy consumption.

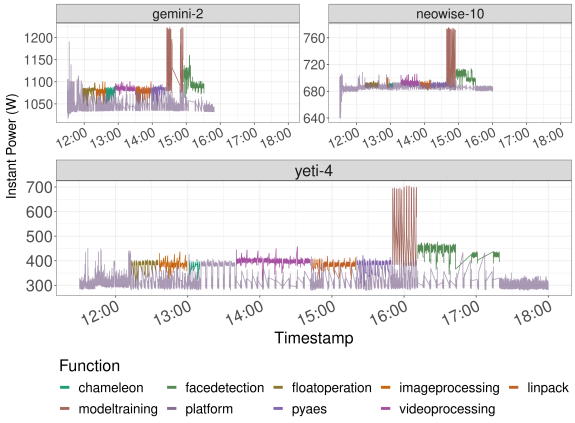


Figure 4: Instant power measurements (in Watts) of serverless platforms worker nodes in our bare-metal setup.

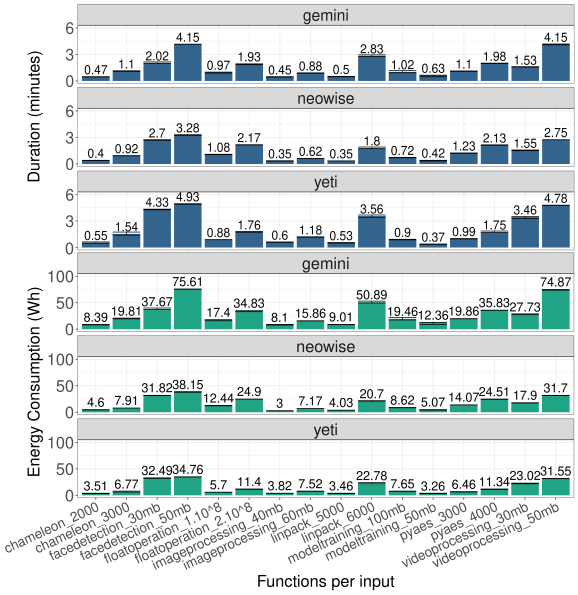


Figure 5: Median Duration and Energy Consumption for different serverless functions in our bare-metal setup.

5.2 Analysis of Scheduling Results

In the remain of this section, we present the analysis of the Pareto Front of *FOA*-energy' linear program, and the results of our simulations to study the two levels of *FOA*-energy's performance, compared against our baseline. Figures 7-(a), (b), (c), (d) and (e), present workload sizes over different metrics, with facets representing the platform sizes, and the colors stands for scheduling policies (from the left to the right they are respectively *FOA-energy+FirstFit*, *FOA*-energy + LargestFirst, *FOA*-energy + SmallestFirst, *FOA*-energy + FCFS, and FCFS + *K8S ImageLocality*).

(i) Pareto Front of *FOA*-energy Linear Program: optimizing energy consumption and makespan is a compromise, and we investigate such a compromise by re-executing our linear program several times, reducing the constraint of makespan each time (see Section 3.1). Figure 6 illustrates this compromise through a Pareto

Figure 6: *FOA*-Energy Linear Program Pareto Front for the different combinations of workload and platform size.

Front for each combination of workload and platform size, having makespan (in minutes) over energy consumption (in kW/h). The first iteration computes the smallest sum of energy consumption and hence the highest relevant makespan. As far as we constrain the makespan over the iterations, the energy consumption increases. We highlight that (i) four or five iterations achieve good trade-offs between both objectives; (ii) the Pareto's shape is smooth in all scenarios. In addition, we remark that (a) not all instances found a solution of energy consumption with small values for makespan; (b) since we need to re-execute the linear program at each iteration, the scheduling time to produce a final solution is cumulative. (b) the scheduling time to produce a final solution increases as we need to re-execute the linear program.(c) we use an open-source solver, CBC, through a *Python* library, *python-pulp*, to compute our linear program. A more advanced solver might reduce its processing time.

(ii) Energy Consumption: Figure 7-(a) shows that in all scenarios, *FOA*-energy-based algorithms outperform *K8S ImageLocality* with gaps from one to three orders of magnitude regarding the energy consumption. It is clear that the greedy strategy of the *K8S ImageLocality* does not contribute to reductions in energy consumption. Concerning the *FOA*-energy-based algorithms, all performed identically thanks to their basis algorithm in the first level of scheduling, which is *FOA*-energy, that optimizes the energy consumption of the whole platform. It is also interesting to discuss why energy consumption does not change across the different platform sizes for each fixed combination of workload. The difference between the platforms with 25%, 50% and 100% is the number of similar machines inside each cluster. The clusters and their energy consumption characteristics remains the same. So, if *FOA*-energy is capable of optimizing the placement of the workloads in the smallest platform, it is not going to use more resources only because the platforms increase and the workloads remain the same.

(iii) Makespan: Figure 7-(b) shows that all *FOA*-energy-based algorithms, except FirstFit, are robust concerning the makespan, in all platforms, and their performance improves as far as there are more resources available. Comparing the makespan, our best-performing algorithms reduce it by more than two orders of magnitude in comparison to *K8S ImageLocality* for the smallest platform

with the biggest workload size. More generally, *FOA*-energy-based algorithms performs better on overloaded resources.

FirstFit is the least performing in the other platforms due to its algorithm, that respects a limit of acceptable makespan, which we consider as the makespan of the last solution of *FOA*-energy linear program. FCFS, LargestFirst, and SmallestFirst better distribute the functions across the different machines by scheduling one function per machine at a time. Finally, *K8S ImageLocality*, use more machines as the platform size increases, as a greedy algorithm, improving its performance in makespan, but ignoring other aspects such as energy consumption.

(iv) Amount of Downloaded Data: Figure 7-(c) shows that an algorithm that optimizes function placement, such as *FOA*-energy, really outperforms greedy algorithms such as *K8S ImageLocality*. We can also see that *FOA*-energy-based algorithms present different performances. The LargestFirst policy outperforms the SmallestFirst as we increase the size of the platforms, by scheduling the largest functions first. The largest functions may require the biggest containers, as well as utilize and produce the biggest inputs and outputs. All of that is summed up as the amount of data downloaded. The FCFS policy shows that a random approach of dispatching functions to the machines without any regard for their duration can be a good fit in terms of matching the containers that are used by other functions. The FirstFit approach, using the same machine as much as possible, optimizes the most the number of duplicated containers in the platform. *K8S ImageLocality* only prioritizes machines where the entire container is present. Even if it does share container layers if present at deployment time, container layer aware scheduling policy in comparison reduces the amount of downloaded layers. Our best *FOA*-energy-based, FCFS, approach outperforms *K8S ImageLocality* by from one to three orders of magnitude.

(v) Time to Compute a Scheduling Solution: *FOA*-energy is based on a linear program with several constraints, as shown in Equation 1, which is expensive regarding the time to produce a solution. Even so, we achieve comparable performances between *FOA*-energy-based algorithms and *K8S ImageLocality* in terms of scheduling time (the time to produce a scheduling solution). We remark that we have two main choices concerning the linear program that results in such a performance: (1) the use of fractional instead of integral solutions, and (2) the modeling of clusters instead of machines. Both choices speed up the linear program's response time because (1) makes it easier for the solver to find optimal solutions, and (2) results in much fewer variables to be considered in the model since we have just a few clusters against hundreds of machines. Figure 7-(d) shows that all *FOA*-energy-based algorithms perform in order of seconds and are stable regarding the different scenarios. We recall that *FOA*-energy's decision considered five repetitions of its binary search process. Fewer iterations would result in even faster scheduling time, but higher energy consumption.

(vi) Number of Machines Used: Figure 7-(e) presents the number of machines used for each solution. *K8S ImageLocality*, as a greedy algorithm, uses as many machines as possible, while all *FOA*-energy-based algorithms better choose the machines to be used. When comparing the *FOA*-energy-based algorithms, the FirstFit policy provides solutions using always the same number of machines due to its algorithm of using each machine as much as possible. The other three *FOA*-energy-based algorithms exploit the increase

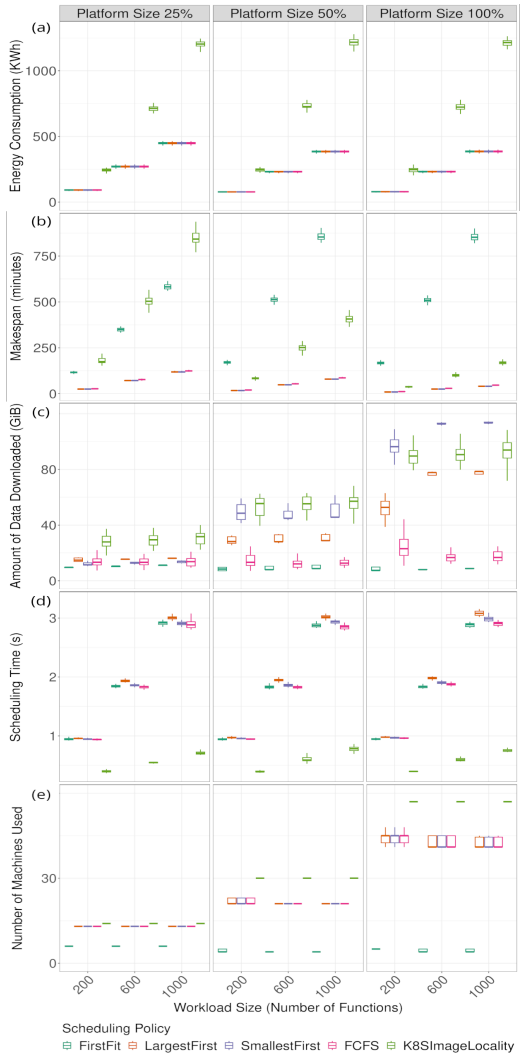


Figure 7: Comparison of *FOA*-Energy based algorithms and *K8S ImageLocality* for different metrics.

in the number of machines per cluster while they still do not use the whole platform, reducing up to one order of magnitude of the number of machines used when compared to *K8S ImageLocality*. In summary, *FOA*-energy-based algorithms provide optimal solutions regarding the number of machines used, even in small scenarios when there are many functions per machine.

6 Conclusions and Future Directions

We evaluate the impacts of considering a multi-objective approach for a two-level scheduling policy on serverless platforms in the heterogeneous edge-cloud continuum. We propose our multi-objective scheduling policy, called *FOA*-energy. Our experimental results show that standard cloud greedy algorithms, like the baseline inspired by Kubernetes, may not fully leverage the potential efficiency of heterogeneous serverless platforms at the edge. *FOA*-energy, on the contrary, optimizes the placement of functions by its multi-objective policy. It outperforms the baseline algorithm when combinging energy consumption, data transfers, and makespan

minimization objectives, in addition to the number of machines used, by up to three orders of magnitudes. We remark that *FOA*-energy is robust regarding the heterogeneity, and it is not affected by the different levels of heterogeneity studied, producing results with the same quality for all of them. By reducing the amount of data transferred to download containers, we minimize cold start delays through faster container deployments. In addition, it also speeds up the functions' execution time. Thus, efficient scheduling policies for serverless computing at the edge-cloud continuum require better management of the heterogeneity to drastically reduce the amount of data downloaded and the number of machines used.

In the future, we plan to follow three main directions. (i) *FOA*-energy is going to be deployed as the main scheduling policy of a real-world serverless platform, and we are going to investigate its response in such an environment. (ii) we plan to study applications that can be modeled as workflows of serverless functions, which is an increasingly used programming style and allows developers to describe complete and complex application logic. (iii) after implementing *FOA*-energy on real-world serverless platforms, we plan to connect it to forecasting components to make possible an online utilization of *FOA*-energy's offline approach.

Acknowledgments

This work was supported by the research program on Edge Intelligence of the Multi-disciplinary Institute on Artificial Intelligence MIAI at Grenoble Alpes (ANR-19-P3IA-0003), and by the European Union's Horizon project EMPYREAN (GA 101136024). Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

References

[1] 2014. Prometheus. https://prometheus.io/. Accessed: October-2024.
[2] 2016. Kubeless. https://github.com/vmware-archive/kubeless. Accessed: October-2024.
[3] 2016. OpenWhisk. https://openwhisk.apache.org/ Accessed: October-2024.
[4] 2019. IBM MultiCluster Dispatcher. https://github.com/IBM/multi-cluster-app-dispatcher. Accessed: October-2024.
[5] 2019. IBM Safer Scheduler. https://github.com/IBM/Kube-Safe-Scheduler. Accessed: October-2024.
[6] 2020. Scaphandre. https://hubblo-org.github.io/scaphandre-documentation/ Accessed: October-2024.
[7] 2020. Yunikorn. https://yunikorn.apache.org/. Accessed: October-2024.
[8] 2021. OpenFaas. https://www.openfaas.com/. Accessed: October-2024.
[9] 2023. Data centres and data transmission networks, IEA, Paris, Tech. https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks. Accessed: October-2024.
[10] 2023. FOA Energy - GitLab Repository. https://gitlab.com/andersonandrei/foa-energy. Accessed: October-2024.
[11] 2023. Grid5000. https://www.grid5000.fr/ Accessed: October-2024.
[12] 2023. Kwollect GRID5000. https://www.grid5000.fr/w/Monitoring_Using_Kwollect. Accessed: October-2024.
[13] 2024. Docker. https://www.docker.com/. Accessed: October-2024.
[14] 2024. Grafana. https://grafana.com/. Accessed: October-2024.
[15] n.d. Kubernetes Plugins. https://github.com/kubernetes-sigs/scheduler-plugins. Accessed: October-2024.
[16] Marcelo Amaral, Huamin Chen, Tatsuhiro Chiba, Rina Nakazawa, Sunyanan Choochotkaew, Eun Kyung Lee, and Tamar Eilam. 2023. Kepler: A Framework to Calculate the Energy Consumption of Containerized Applications. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*. 69–71.
[17] Luc Angelelli, Anderson Andrei da Silva, Yiannis Georgiou, Michael Mercier, Gregory Mounié, and Denis Trystram. 2023. Towards a Multi-objective Scheduling Policy for Serverless-based Edge-Cloud Continuum. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 485–497. https://doi.org/10.1109/CCGrid57682.2023.00052
[18] Gabriel Aumala, Edwin Boza, Luis Ortiz-Aviles, Gustavo Totoy, and Cristina Abad. 2019. Beyond Load Balancing: Package-Aware Scheduling for Serverless Platforms. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, Larnaca, Cyprus, 282–291.
[19] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, and Philippe Suter. 2017. Serverless Computing: Current Trends and Open Problems. *arXiv:1706.03178 [cs]* (June 2017). http://arxiv.org/abs/1706.03178 arXiv: 1706.03178.
[20] Daniel Barcelona-Pons, Marc Sánchez-Artigas, Gerard París, Pierre Sutra, and Pedro García-López. 2019. On the FaaS Track: Building Stateful Distributed Applications with Serverless Architectures. In *Proceedings of the 20th International Middleware Conference*. ACM, Davis CA USA, 41–54.
[21] J.O. Bird. 2017. *Higher Engineering Mathematics*. Routledge. https://doi.org/10.1201/9781315265025
[22] Pedro Bruel, Sai Rahul Chalamalasetti, Aditya Dhakal, Eitan Frachtenberg, Ninad Hogade, Rolando Pablo Hong Enriquez, Alok Mishra, Dejan Milojicic, Pavana Prakash, and Gourav Rattihalli. 2024. Predicting Heterogeneity and Serverless Principles of Converged High-Performance Computing, Artificial Intelligence, and Workflows. *Computer* 57, 1 (2024), 136–144. https://doi.org/10.1109/MC.2023.3332973
[23] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. 2014. Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms. *J. Parallel and Distrib. Comput.* 74, 10 (June 2014), 2899–2917. http://hal.inria.fr/hal-01017319
[24] Pierre-François Dutot, Michael Mercier, Millian Poquet, and Olivier Richard. 2016. Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator. In *20th Workshop on Job Scheduling Strategies for Parallel Processing*. Chicago, United States. https://hal.archives-ouvertes.fr/hal-01333471
[25] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. 2019. Cloud Programming Simplified: A Berkeley View on Serverless Computing. *arXiv:1902.03383 [cs]* (Feb. 2019). http://arxiv.org/abs/1902.03383
[26] Jeongchul Kim and Kyungyong Lee. 2019. FunctionBench: A Suite of Workloads for Serverless Cloud Function Service. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, Milan, Italy, 502–504.
[27] Jeongchul Kim and Kyungyong Lee. 2019. Practical Cloud Workloads for Serverless FaaS. In *Proceedings of the ACM Symposium on Cloud Computing* (Santa Cruz, CA, USA) *(SoCC '19)*. Association for Computing Machinery, New York, NY, USA, 477.
[28] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. 2019. Elastic Ephemeral Storage for Serverless Analytics. 44, 1 (2019), 6.
[29] Zijun Li, Linsong Guo, Quan Chen, Jiagan Cheng, Chuhao Xu, Deze Zeng, Zhuo Song, Tao Ma, Yong Yang, Chao Li, and Minyi Guo. 2022. Help Rather Than Recycle: Alleviating Cold Startup in Serverless Computing Through Inter-Function Container Sharing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 69–84.
[30] Nima Mahmoudi and Hamzeh Khazaei. 2021. SimFaaS: A Performance Simulator for Serverless Computing Platforms. *arXiv:2102.08904 [cs]* (Feb. 2021).
[31] M.D. Plummer and L. Lovász. 1986. *Matching Theory*. Elsevier Science.
[32] Thomas Rausch, Alexander Rashed, and Schahram Dustdar. 2020. Optimized container scheduling for data-intensive serverless edge computing. *Future Generation Computer Systems* 114 (08 2020). https://doi.org/10.1016/j.future.2020.07.017
[33] Daniel Rosendo, Alexandru Costan, Patrick Valduriez, and Gabriel Antoniu. 2022. Distributed intelligence on the Edge-to-Cloud Continuum: A systematic literature review. *J. Parallel Distributed Comput.* 166 (2022), 71–94.
[34] Mohammad Shahrad, Jonathan Balkind, and David Wentzlaff. 2019. Architectural Implications of Function-as-a-Service Computing. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 1063–1075. https://doi.org/10.1145/3352460.3358296
[35] David B. Shmoys and Éva Tardos. 1993. An Approximation Algorithm for the Generalized Assignment Problem. *Math. Program.* 62, 1–3 (Feb. 1993), 461–474.
[36] Amoghavarsha Suresh and Anshul Gandhi. 2019. FnSched: An Efficient Scheduler for Serverless Functions. 19–24. https://doi.org/10.1145/3366623.3368136
[37] Vladimir Yussupov, Uwe Breitenbücher, Frank Leymann, and Christian Müller. 2019. Facing the Unplanned Migration of Serverless Applications: A Study on Portability Problems, Solutions, and Dead Ends. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing* (Auckland, New Zealand) *(UCC'19)*. New York, NY, USA, 273–283.
[38] Paweł Żuk, Bartłomiej Przybylski, and Krzysztof Rzadca. 2022. Call Scheduling to Reduce Response Time of a FaaS System. https://doi.org/10.48550/ARXIV.2207.13168