Energy-Aware Scheduling of a Serverless Workload in an ISA-Heterogeneous Cluster

Simon Arys
UCLouvain
Louvain-la-Neuve, Belgium

Romain Carlier
UCLouvain
Louvain-la-Neuve, Belgium

Etienne Rivière
etienne.riviere@uclouvain.be
UCLouvain
Louvain-la-Neuve, Belgium

Abstract

The serverless model decouples computation from infrastructure. This results in flexibility for running serverless functions on heterogeneous hardware, such as emerging x86/ARM ISA-heterogeneous clusters. We present a method for scheduling serverless workloads across ISA-heterogeneity boundaries to reduce energy usage. Our method combines the offline profiling of functions for energy use and performance, the construction of performance/energy affinity models, and an energy-aware scheduler. Our evaluation with servers equipped with Xeon x86 and Ampere Altra Max ARM processors and 22 serverless functions shows that energy usage can be reduced by up to 15.2%.

CCS Concepts

• **Computer systems organization** → **Cloud computing**; • **Social and professional topics** → *Sustainability*.

Keywords

Serverless computing, scheduling, energy-efficiency, ISA heterogeneity.

ACM Reference Format:
Simon Arys, Romain Carlier, and Etienne Rivière. 2024. Energy-Aware Scheduling of a Serverless Workload in an ISA-Heterogeneous Cluster. In *10th International Workshop on Serverless Computing (WOSC '24), December 2–6, 2024, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3702634.3702954

1 Introduction

Serverless computing is a recent computing paradigm that allows code execution to be decoupled from the underlying infrastructure. This deployment model, also known as Function as a Service (FaaS), has gained popularity over the last years due to its simplicity in deployment and management and its cost-effectiveness, with a pay-per-use billing scheme for customers. Many applications are suited for the serverless model, including web and mobile application backends, data processing, or event-driven workflows [12, 24].

WOSC '24, December 2–6, 2024, Hong Kong, Hong Kong

ACM ISBN 979-8-4007-1336-1/24/12
https://doi.org/10.1145/3702634.3702954

At the same time, data centers hosting serverless workloads are becoming increasingly heterogeneous. Among all sources of heterogeneity, the Instruction Set Architecture (ISA)-heterogeneity is of particular interest. ARM CPUs, with their promises of better energy efficiency and performance, now complement traditional x86 CPUs. For instance, AWS Lambda functions can now be deployed on ARM Graviton2 processors [2] and, more recently, Google Cloud introduced their new Axion ARM-based CPUs [26].

The decoupling of computation and execution infrastructure, as well as the statelessness of functions enforced by the serverless model, gives cloud operators ample flexibility for consolidation [15], scale-out, or cross-site migrations [18]. This flexibility also allows for deploying workloads in heterogeneous environments. Serverless functions can be compiled for (or interpreted on) different CPU architectures, allowing the scheduling of serverless workloads across ISA-heterogeneity boundaries.

We wish to evaluate whether informed scheduling of serverless functions in a heterogeneous cluster can improve infrastructure efficiency, particularly in the energy necessary to support a target workload. Serverless functions typically have different characteristics and responses to heterogeneous resources, e.g., they could be computation-, memory-, or I/O-dominated [12]. We need precise models of each function's performance and energy consumption. We build such models using offline function profiling, accounting for the impact of consolidating different functions on the same server and sharing resources. We use these models in a scheduler, allocating CPU cores to functions to match a desired throughput while minimizing the expected energy consumption.

Our study is driven by experimental validation on hardware representative of a modern, ISA-heterogeneous data center: a cluster of two medium-end servers. The first is a dual-socket server with two 24-core (48 threads) Intel Xeon Gold 5318Y CPUs (i.e., 48 cores or 96 threads), a base frequency of 2.1GHz, and a max frequency of 3.4GHz. Both Intel CPUs have a Thermal Design Power (TDP) of 175W, for a total of 350W. The second server features a single 128-core Ampere Altra Max ARM processor [6], with frequencies ranging from 1 to 3GHz and a TDP of 250W. Both servers have similar equipment besides CPUs: 256GB of RAM, a Connect-X5 NIC, a single M2 SSD, and no GPU. A connected power distribution unit (PDU) measures servers' global electricity consumption. For conciseness, we will refer to the two servers as x86 and ARM for the rest of this paper.

Contributions and outline. Our method combines three steps, illustrated by the workflow in Figure 1. We first describe how we can isolate the energy consumption of *individual* functions on servers (§2). We leverage the PowerAPI toolchain [11]. PowerAPI

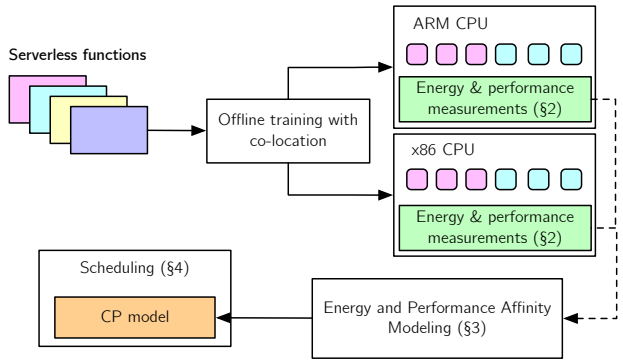

Figure 1: Overview of our method.

only supported x86 CPUs, so we implemented the Ampere Altra Max ARM CPU support.

Then, we detail how we use these individual energy measurements with performance observations to build models that characterize each function's affinity towards the different machines (§3). Our approach uses offline profiling with different mixes of functions. We consider a highly-consolidated context where multiple functions are deployed on the same hardware. In this context, taking the impact of function colocation is crucial for building accurate models.

Finally, we present a scheduling policy building upon affinity models to decide on a resource allocation that can support a target throughput for each function (§4). We implement this policy as a Constraint Programming model.

We evaluate our approach with a representative set of 22 serverless functions (§5), including classical serverless benchmarks and new workloads representative of modern application backends in the cloud. While most functions perform better and are more energy efficient on ARM than on x86, the extent of this difference varies significantly across functions. In a consolidated setting where both servers are necessary to achieve the desired throughput for each function in a workload, our results show that energy-aware scheduling can reduce the necessary energy by as much as 15.2%.

We terminate this paper by positioning to related work (§6) and discussing suggestions for future research (§7).

2 Energy and Performance Measurements

Our first step is to measure the performance and individual power consumption of functions running in a consolidated environment (i.e., with functions co-located on the same server). Throughput is the number of executions a function performs sequentially during an observation. In what follows, we focus on the more delicate case of energy measurement.

The PDU reports per-server electricity consumption. Intel's Running Average Power Limit (RAPL) interface provides information about the consumption of an entire processor. Both metrics are too coarse-grained for our purpose: We need to measure the consumption of *individual* functions when many run on the same server or CPU. Note that, in this work, we run functions as processes, but the same reasoning applies when running them as containers.

Process-level energy monitoring is challenging, as it requires the aggregation from multiple hardware sensors that are ISA and vendor-specific [11, 13], and processes are an OS-level abstraction to which sensors such as RAPL are oblivious. Systems such as Kepler [1] or PowerAPI [11] solve these issues with pre-trained power models that allow splitting global energy consumption amongst supported processes.

We selected PowerAPI [11] for its maturity and ease of use. We extended it to support the Ampere Altra Max processor.

PowerAPI uses two components for energy estimation. The *sensor* collects a variety of performance counters. The *formula* aggregates these counters to build self-calibrating power models and estimate per-process energy consumption. The formula trains one model per frequency level and selects one at runtime, depending on the current average frequency of the CPU cores.[1] Each power model is an Elastic Net regression, i.e., a regular linear regression with combined L1 and L2 priors as regularizers. The features used to train the model are the CPU consumption and multiple hardware performance counters, such as the number of CPU cycles and cache misses. Based on these per-process hardware performance counters, the model infers the share of global CPU energy that each process consumes.

PowerAPI obtains the CPU-level energy consumption from RAPL on an Intel CPU, and we added support for the SMpro interface of the Ampere Altra Max CPU, which exposes the same metrics [7]. We experimentally selected the Ampere-specific hardware performance counters that were the most correlated with CPU power consumption when running multiple instances of the same process. We selected the number of CPU cycles spent executing the process, the number of retired instructions, and the number of stalled cycles as metrics.

3 Energy and Performance Affinity Models

Our next step is to determine, for a workload composed of multiple functions, the performance and energy consumption expected from each function when running on different ISAs. The result is two matrices, E and W, each containing $m \times n$ elements, where m is the number of servers and n is the number of functions. E_{ij} is the performance of function j on server i, i.e., how many sequential executions of the functions a single core of processor i can support. W_{ij} is the energy efficiency of function j on server i, expressed in watt-hours per execution.

A straightforward approach to compute E and W is to run every function sequentially and in isolation on each server, using one or many cores, and record the number of execution and consumption levels using PowerAPI. Unfortunately, this approach is ineffective in a consolidated context. These metrics vary significantly based on the colocated other functions on our target multi-core CPUs. This is due to concurrent access to shared system resources (i.e., memory bandwidth, file system access, etc.). This observation is consistent with previous work [27].

Running every possible combination of functions on each server to capture all colocation impacts is impractical. Our offline profiling

[1]The PowerAPI sensor collects the cores' frequency by reading two Model-Specific Registers (MSRs) available on recent x86 processors. The Linux CPPC driver [9] exposes a set of Ampere Altra Max CPU registers that provide this information.

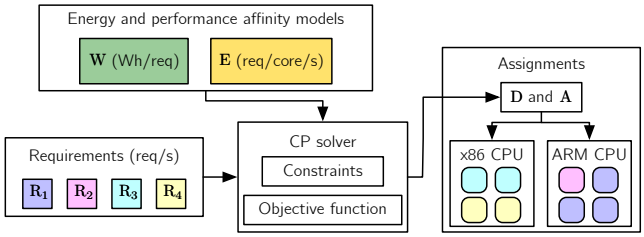

Figure 2: Example of a scheduling decision.

uses a middle-ground strategy that proved effective in practice. We select random groups of functions (eight in our configuration), equally split the CPU cores on the machine between them (i.e., 16 cores per function on the ARM machine, 12 on the x86 machine), and collect each function's performance and energy consumption. We repeat the process for 20 random groups, ensuring that each function is represented in at least two groups. The value used for E_{ij} and W_{ij} is the average of the measurements on server i for function j, in groups where it was represented.

4 Scheduling

Our final step is to use the affinity matrices E and W to schedule functions over the two heterogeneous servers and attempt to reduce energy consumption compared to non-energy-aware baselines. Our scheduler receives as input the throughput requirements for a set of functions as a vector R, where R_i is the minimum necessary throughput for function i (e.g., 4,000 requests per second). Its output is an allocation of cores to functions across the two machines. Figure 2 exemplifies the scheduling process.

We aim to determine if energy-aware scheduling has saving potential under the following assumptions. First, we focus on the case where no single server is sufficient to support the workload requirements R (otherwise, the best strategy is always to fill one server and shut down the other). Second, each function is assigned to a number of cores on a single server. Cores are not shared between functions, and functions are not deployed *across* the two servers. Finally, we consider fixed assignments and do not implement function migration across hosts during the execution. While some of these constraints would need to be relaxed for production-ready, energy-aware schedulers, they allow for keeping the model simple and do not impair our conclusions.

We model the scheduling problem as a Constraint Programming (CP) model. A CP model defines variables (input and decision), constraints, and an objective function. Our scheduling model features five variables. E, W, and R are input variables. The first decision variable, D, is a boolean matrix of size $m \times n$ indicating whether a function is assigned to a server. D_{ij} is true if function j is assigned to server i. The second decision variable, A, is a vector representing the number of cores A_i allocated to function i on its server.

Constraints allow defining what schedules are valid. First, each function is assigned to exactly one server:

$$\sum_{i=1}^{m} \left(D_{ij}\right) = 1 \quad \forall j \in [1, n] \tag{1}$$

Then, we need to ensure that the capacity of each server is not exceeded:

$$C_i \geq \sum_{j=1}^{n} \left(D_{ij} \cdot A_i\right) \quad \forall i \in [1, m] \tag{2}$$

where C_i is the total number of cores of server i. Finally, we assign cores to functions as follows:

$$A_j = \left\lceil \sum_{i=1}^{m} \left(D_{ij} \cdot \frac{R_j}{E_{ij}}\right) \right\rceil \quad \forall j \in [1, n] \tag{3}$$

The ceiling in the constraint ensures that each function receives an integer number of cores.

The final element of the model is the objective function that the solver will attempt to minimize. We define an objective function O_E, or *energy-aware*–EA, that sums the total projected energy across all functions:

$$O_E : \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \left(D_{ij} \cdot W_{ij} \cdot R_j\right) \right) \tag{4}$$

For comparison purposes, we also use an energy-oblivious function (*cores-aware*–CA) that only counts the number of allocated cores and disregards the content of W:

$$O_C : \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \left(D_{ij} \cdot A_j\right) \right) \tag{5}$$

This objective is interesting as a baseline that would maximize consolidation and may use fewer cores.

We define our variables, constraints, and objective functions using MiniZinc [19], a solver-independent constraint modeling language. We use the Gecode [23] solver to produce schedules. Using the model as-is with simple search strategies is sufficient to solve small instances (i.e., a few dozen functions and a few servers). However, the complexity of the problem grows exponentially with the number of functions and servers. Larger instances call for better search strategies and heuristics, which we leave to future work.

5 Evaluation

Our evaluation is in three parts. We first validate the power measurements using PowerAPI for the ARM server (§5.1) Then, we detail our target workload of 22 serverless functions and the results of offline performance and energy affinity modeling (§5.2). Finally, we evaluate the ability of our scheduler to reduce energy in a highly-consolidated environment and compare it to standard scheduling strategies (§5.3).

5.1 Energy and Performance Measurements

The PowerAPI formula is self-calibrating. It (re)trains the multiple power models when the power consumption estimation error exceeds a defined threshold. The formula computes the error by comparing the model's prediction (at the CPU level) to the values the CPU power sensors expose. We validate the accuracy of the process-level estimations from the PowerAPI's model on the ARM server using `stress-ng`. We generate load levels using 0% to 100% of the cores of the CPU in increments of 10% every 3 minutes.

Figure 3 shows the power estimation and error level. We first observe that higher loads lead to lower estimation errors. The measures correspond to the expected TDP (250W) at 100% load. With

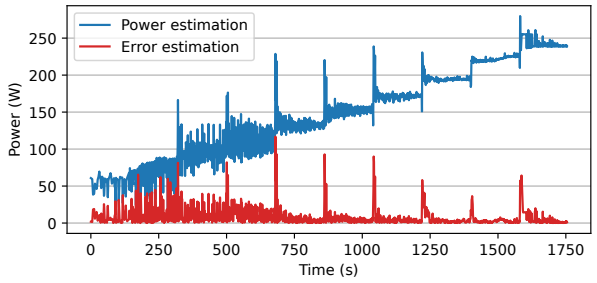

Figure 3: Process-level power estimation on the ARM server.

	Function	Runtimes	Description
		Category 1: Crypto	
1–2	**El Passo**	C++, WebAssembly	Compute anonymous credentials for single-sign-on [28].
3–5	**JWT**	Python, Node.js, Java	Create and sign JWT token.
		Category 2: Media	
6–8	**Thumbnail**	Python, Node.js, Java	Resize a base64-encoded image.
9–10	**Video to GIF**	Python, Node.js	Transcode a video to a GIF using ffmpeg.
11	**Img Rec.**	Python	Object recognition in an image using the AlexNet model of Torchvision.
		Category 3: Scientific	
12	**Matrix NumPy**	Python	Compute dot product of two random 30x30 matrices with NumPy.
13–15	**Matrix native**	Python, Node.js, Java	Compute dot product of two random 30x30 matrices using native code.
16	**PageRank**	Python	Compute pagerank on a 500-vertex, 250-edge graph using igraph.
		Category 4: Web	
17–19	**HTML**	Python, Node.js, Java	Fill HTML template using jinja2 (Python), Mustache (Node.JS), or FreeMaker (Java).
20–22	**Zip**	Python, Node.js, Java	Compress 3 files into a single zip archive.

Table 1: Workload of 22 serverless functions (10 functions, each for 1 to 3 different languages and runtimes).

only 10% initial load (≃12.8 cores), the consumption is around 50W. The estimation error spikes after a change in load but reduces over time. This results from the self-calibration of the PowerAPI models, which adjust the model over time to correct prediction errors. We account for these two observations in the profiling phase by running and monitoring functions under high load and for a sufficient duration, i.e., three minutes of observations following a one-minute warm-up.

5.2 Energy and Performance Affinity Models

We now evaluate the offline affinity modeling process. We use the 22-function workload detailed in Table 1. This represents the variety of functions supported by serverless platforms. Seven triplets of functions (all but #11, #12, and #16) are different implementations of the same task using three different programming languages and runtimes. The numbering of these functions follows the order of

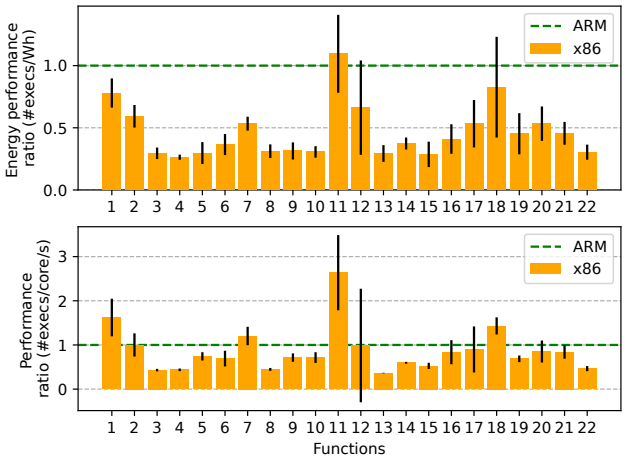

Figure 4: Relative energy (top) and performance (bottom) affinity profiles of the 22 functions on x86 and ARM.

options in the "Runtimes" column of Table 1. For instance, functions #3, #4, and #5 implement the generation of JWT tokens using Python, Node.JS, or Java, respectively. Eight functions (#6, #7, #9, #11, #16, #17, #18, and #20) are from the SeBS benchmark suite [8], while we implemented the others to improve diversity. Functions #1 and #2 are representative of modern, computationally intensive cryptography operations used in privacy-preserving systems and web3 [28]. They use either native (C++) or WebAssembly code. We group functions into four categories depending on their application domain.

Figure 4 presents the energy and performance profiles of the 22 functions obtained from offline profiling (§3). We choose to present relative values comparing the ARM and x86 estimations. A ratio of 1 means that both servers have comparable performance or energy efficiency. A ratio >1 means x86 is better, and a ratio < 1 means ARM is.

21/22 functions have *either* better performance or better energy efficiency on ARM. Only one function (#11, image recognition) has better results for both metrics on x86. Five functions (#1, #2, #7, #11 and #18) execute more instances per core per second on x86, but their energy efficiency remains lower. Some functions (e.g., #11, #12, and #18) show high metrics variability, meaning that their performance and energy efficiency are heavily influenced by colocated functions.[2] Overall, while there is a general advantage in performance and energy efficiency for ARM, the extent of this advantage varies significantly, from 21% (#18) to 280% (#4). Intuitively, functions that benefit the most from one type of machine should have a higher priority to be scheduled on that machine than functions that perform more similarly on the two heterogeneous platforms.

5.3 Scheduling

We evaluate the ability of our scheduler to improve the heterogeneous cluster's energy consumption, as measured by the PDU level

[2]Note that we leave for future work the analysis of resource sharing at the CPU and memory level to understand the reasons for these variations. For the present study, high-level "opaque box" models are sufficient.

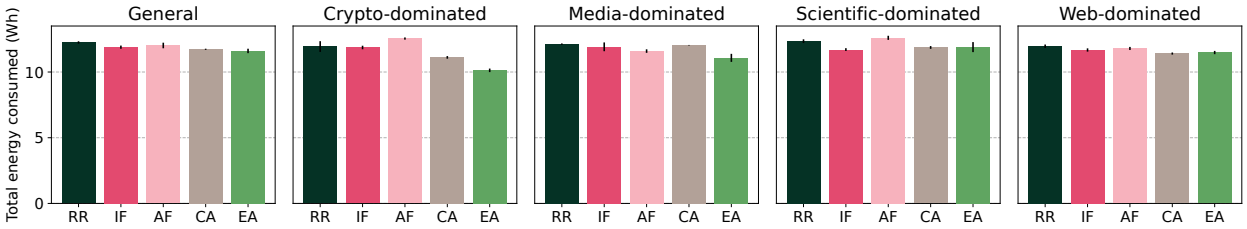

Figure 5: Total energy consumed by the cluster for a balanced workload (General) and workloads dominated by an application class. Policies: Round-Robin (RR), Intel-First (IF), Ampere-First (AF), Cores-aware (CA), and Energy-aware (EA).

for the two machines (i.e., not only their CPUs). We compare our energy-aware (EA) policy to four baselines. Core-aware (CA) minimizes the number of cores used but ignores energy-efficiency profiles. Round-Robin places functions alternatively on the two servers, allocating the necessary cores according to the performance model. Finally, Ampere-First (AF) and Intel-First (IF) consider functions in a randomized order and start by filling one server (ARM or x86) before assigning the remaining functions to the other. AF and IF do not consider energy efficiency.

We consider five workloads, all using the 22 functions. The "General" workload assigns a request requirement vector R such that the computational volume, as measured as the average between the number of cores necessary on ARM and x86, is balanced across functions. This allows accounting for the vast difference between absolute throughput across functions (i.e., from 0.48 req/core/s for #9 to 49,508 req/core/s for #5) and balancing the contributions of all functions. The "x-dominated" workloads, where 'x' is a category of Table 1 assign R such that functions of category 'x' account for 80% of the load, with 20% remaining for all other functions.

Figure 5 presents the overall consumption for the workloads, with resources statically assigned by the different policies. RR is systematically the worst, except for the Crypto-dominated workload, where using ARM first leads to the highest energy consumption. IF shows a similar performance to RR. CA is an improvement to RR in all cases. Minimizing the number of cores across machines improves performance from 0.6% to 6.9%. EA achieves the best improvement in the General, Media-dominated, and particularly Crypto-dominated workloads, with gains ranging from 5.3% to 15.2% and improving upon the CA policy. For the Scientific- and Web-dominated workloads, however, there is no significative gain from reducing expected energy consumption compared to reducing the number of cores used. For the Scientific-dominated workload, EA uses more ARM cores than CA (128/158 total vs. 115/155). EA and CA use the 128 ARM cores for the Web-dominated workload and yield the same assignment. In summary, we observe that EA is better for some workloads and never worse than CA.

Summary. Our evaluation shows that energy-efficient scheduling, by leveraging heterogeneity, may positively impact the electrical consumption of highly consolidated serverless infrastructures. The most significant gains are with the crypto-dominated workload, which is very CPU-intensive and includes functions that benefit more from x86. In practice, we also observe that ARM's advantage in performance and energy efficiency is real for individual functions, but high colocation leads to performance losses that reduce the gap with x86, which is less subject to such colocation effects. Measuring the micro-architectural reasons for this difference is an interesting path for future work.

6 Related work

Chen *et al.* [5] study the relative performance of serverless functions on x86 (Intel) and ARM64 (Graviton2) running on AWS Lambda. Their study highlights that functions requiring more system calls tend to perform better on x86, while ARM performance tends to be more stable. Their evaluation targets a closed cloud system, so they could not relate this to energy.

The use of heterogeneous hardware to support serverless workloads has been explored by Molecule [10], a serverless platform supporting the execution of functions over various accelerators (e.g., GPUs and FPGA), and Icebreaker [22], which leverages a heterogeneous fleet of servers (i.e., inexpensive servers together with expensive ones using the same ISA) to reduce the bootstrap time of functions. Molecule and Icebreaker aim, however, to maximize function execution throughput, not reduce energy consumption.

EcoFaaS [25] is an energy-aware serverless execution framework that uses a model of the execution time vs. functions' input characteristics. This model assigns optimal core frequency to functions and reduces energy consumption. While we did not consider frequency throttling in this paper, this approach would complement ours. EcoFaaS does not target environments formed of ISA-heterogeneous nodes.

HEATS [21] is a serverless scheduler that allows users to express a tradeoff between energy and performance and schedule functions to the best suitable and sufficient node. Similarly to our work, HEATS uses a profiling phase, but, in contrast to it, it only considers energy consumption at the PDU level, i.e., for an entire server.

EcoLife [16] and Casa [20] are two serverless schedulers whose goal is to minimize the carbon footprint of functions' execution. Interestingly, EcoLife considers the hardware's embodied (manufacturing) footprint and not only its consumption when executing functions.

GreenCourier [4] and Caribou [14] propose to schedule serverless workloads across geo-distributed sites, accounting for their *carbon efficiency*, i.e., the amount of carbon dioxide emitted by the electricity sources used by these sites. Lin and Shahrad [17] further suggest that the carbon footprint of serverless applications be integrated into pricing models and make it financially attractive to propose and use more sustainable execution infrastructure.

These approaches complement ours, as the scheduling of serverless workloads on each site and for a given electricity source could still leverage heterogeneity to reduce the local footprint.

While this paper focuses on data center environments, the scheduling of serverless functions at the edge also involves heterogeneity. Aslanpour *et al.* [3] show how the scheduling of functions can consider the nature of the energy used by support nodes (e.g., battery-powered vs. using a local source of renewable energy).

7 Conclusion

We evaluated the possibility of scheduling serverless workloads in ISA-heterogeneous infrastructures based on offline performance and energy efficiency modeling. Our results with different mixes of 22 serverless functions show that such an approach can improve energy efficiency. An immediate follow-up direction for this work is the analysis (e.g., at the micro-architecture level) of the impact of function colocation on performance and the integration of such information in the model and scheduling policies. Evaluations at a larger-scale, and combining heterogeneity between servers with CPU of the same ISA (e.g., of different generations) is also an interesting possibility for extension.

Acknowledgments

We would like to thank Tom Barbette and Nikita Tyunyayev for their comments on an earlier version of this work. We are grateful to Guillaume Fieni and Romain Rouvoy for their help and guidance on using and extending the Power API framework.

This work received funding from the Wallonia-Brussels Federation under the ARC project "RAINDROP".

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, several Universities, as well as other organizations (see https://www.grid5000.fr).

References

[1] 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 *16th International Conference on Cloud Computing (CLOUD)*. IEEE, 69–71.

[2] Amazon. 2024. *AWS EC2 Graviton.* Retrieved September 5, 2024 from https://aws.amazon.com/ec2/graviton/

[3] Mohammad Sadegh Aslanpour, Adel N Toosi, Muhammad Aamir Cheema, and Raj Gaire. 2022. Energy-aware resource scheduling for serverless edge computing. In *22nd International Symposium on Cluster, Cloud and Internet Computing*. IEEE, 190–199.

[4] Mohak Chadha, Thandayuthapani Subramanian, Eishi Arima, Michael Gerndt, Martin Schulz, and Osama Abboud. 2023. GreenCourier: Carbon-Aware Scheduling for Serverless Functions. In *Proceedings of the 9th International Workshop on Serverless Computing (WoSC)*. 18–23.

[5] Xinghan Chen, Ling-Hong Hung, Robert Cordingly, and Wes Lloyd. 2023. X86 vs. ARM64: an investigation of factors influencing serverless performance. In *Proceedings of the 9th International Workshop on Serverless Computing (WoSC)*. 7–12.

[6] Ampere Computing. 2024. *Ampere Altra Family Product Brief.* Retrieved September 20, 2024 from https://amperecomputing.com/briefs/ampere-altra-family-product-brief

[7] Ampere Computing. 2024. *Ampere Altra Family SoC BMC Interface Specification.* Retrieved October 1, 2024 from https://amperecomputing.com/customer-connect/products/altra-family-device-documentation

[8] Marcin Copik, Grzegorz Kwasniewski, Maciej Besta, Michal Podstawski, and Torsten Hoefler. 2021. SeBS: A serverless benchmark suite for function-as-a-service computing. In *Proceedings of the 22nd International Middleware Conference*. 64–78.

[9] Linux Kernel documentation. 2024. *Collaborative Processor Performance Control (CPPC).* Retrieved October 1, 2024 from https://docs.kernel.org/admin-guide/acpi/cppc_sysfs.html

[10] Dong Du, Qingyuan Liu, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. 2022. Serverless computing on heterogeneous computers. In *Proceedings of the 27th ACM international conference on architectural support for programming languages and operating systems (ASPLOS)*. 797–813.

[11] Bourdon et al. 2013. Powerapi: A software library to monitor the energy consumed at the process-level. *ERCIM News* 92 (2013), 43–44.

[12] Eismann et al. 2021. The state of serverless applications: Collection, characterization, and community consensus. *IEEE Transactions on Software Engineering* 48, 10 (2021).

[13] Möbius et al. 2013. Power consumption estimation models for processors, virtual machines, and servers. *IEEE Transactions on Parallel and Distributed Systems* 25, 6 (2013), 1600–1614.

[14] Viktor Gsteiger, Pin Hong Daniel Long, Yiran Jerry Sun, Parshan Javanrood, and Mohammad Shahrad. 2024. Caribou: Fine-Grained Geospatial Shifting of Serverless Applications for Sustainability. In *Proceedings of the 30th ACM Symposium on Operating Systems Principles (SOSP)*. ACM.

[15] M Reza HoseinyFarahabady, Javid Taheri, Albert Y Zomaya, and Zahir Tari. 2021. Data-intensive workload consolidation in serverless (Lambda/FaaS) platforms. In *20th International Symposium on Network Computing and Applications (NCA)*. IEEE, 1–8.

[16] Yankai Jiang, Rohan Basu Roy, Baolin Li, and Devesh Tiwari. 2024. EcoLife: Carbon-Aware Serverless Function Scheduling for Sustainable Computing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*.

[17] Changyuan Lin and Mohammad Shahrad. 2024. Bridging the sustainability gap in serverless through observability and carbon-aware pricing. In *Proceedings of the 3rd Workshop on Sustainable Computer Systems (HotCarbon)*.

[18] Viyom Mittal, Shixiong Qi, Ratnadeep Bhattacharya, Xiaosu Lyu, Junfeng Li, Sameer G Kulkarni, Dan Li, Jinho Hwang, KK Ramakrishnan, and Timothy Wood. 2021. Mu: An efficient, fair and responsive serverless framework for resource-constrained edge clouds. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*. 168–181.

[19] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. 2007. MiniZinc: Towards a standard CP modelling language. In *International Conference on Principles and Practice of Constraint Programming (CP)*. Springer, 529–543.

[20] Sirui Qi, Hayden Moore, Ninad Hogade, Dejan Milojicic, Cullen Bash, and Sudeep Pasricha. 2024. CASA: A Framework for SLO and Carbon-Aware Autoscaling and Scheduling in Serverless Cloud Computing. *arXiv preprint arXiv:2409.00550* (2024).

[21] Isabelly Rocha, Christian Göttel, Pascal Felber, Marcelo Pasin, Romain Rouvoy, and Valerio Schiavoni. 2019. Heats: Heterogeneity-and energy-aware task-based scheduling. In *27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. EuroMicro, 400–405.

[22] Rohan Basu Roy, Tirthak Patel, and Devesh Tiwari. 2022. Icebreaker: Warming serverless functions better with heterogeneity. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 753–767.

[23] Christian Schulte, Mikael Lagerkvist, and Guido Tack. 2006. Gecode. *Software download and online material at the website: http://www.gecode.org* (2006).

[24] Hossein Shafiei, Ahmad Khonsari, and Payam Mousavi. 2022. Serverless computing: a survey of opportunities, challenges, and applications. *Comput. Surveys* 54, 11s (2022), 1–32.

[25] Jovan Stojkovic, Nikoleta Iliakopoulou, Tianyin Xu, Hubertus Franke, and Josep Torrellas. 2024. EcoFaaS: Rethinking the Design of Serverless Environments for Energy Efficiency. In *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA)*.

[26] Amin Vahdat. 2024. *Introducing Google Axion Processors, our new Arm-based CPUs.* Retrieved September 5, 2024 from https://cloud.google.com/blog/products/compute/introducing-googles-new-arm-based-cpu

[27] Felippe Vieira Zacarias, Vinicius Petrucci, Rajiv Nishtala, Paul Carpenter, and Daniel Mossé. 2019. Intelligent colocation of workloads for enhanced server efficiency. In *31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 120–127.

[28] Zhiyi Zhang, Michał Król, Alberto Sonnino, Lixia Zhang, and Etienne Rivière. 2021. EL PASSO: efficient and lightweight privacy-preserving single sign on. *Proceedings on Privacy Enhancing Technologies* (2021).