



## International Journal of Computer Science Engineering and Research

Journal homepage: <https://trpub.org/journal.php?id=EihcudPEIP>

TRPub

IJCSE

International Journal of  
Computer Science  
Engineering and  
Research

Vol. 1 Issue 1 2026

Methodology Article

Open Access

# Energy-Aware Scheduling of Serverless Functions Across Heterogeneous Edge and Cloud Nodes: A Proposed Design and Evaluation Protocol

Mateo Yamamoto<sup>1\*</sup>, Madina Le<sup>2</sup>, Suresh Ellery<sup>3</sup>

<sup>1</sup>Department of Computer Science, Sharavati Institute of Computer Science, Norway

<sup>2</sup>Department of Computer Science, Netravati Institute of Computer Science, Georgia

<sup>3</sup>Department of Computer Science, Mekong Research Institute of Computer Science, Chile

### Article Info

Volume 1, Issue 1, 2026

doi: [10.62715/ijcser.2026.v1.i1.pp1-6](https://doi.org/10.62715/ijcser.2026.v1.i1.pp1-6)

Received: 25 September 2025

Accepted: 29 November 2025

Published: 21 January 2026

### Abstract

Serverless platforms decide, for every invocation, where a function runs and whether a warm instance is kept ready for the next call. When the candidate hosts range from single-board edge devices to cloud servers, those two decisions fix most of the energy an application consumes, yet production schedulers make them on latency and utilisation alone. This article proposes a design for an energy-aware scheduler and a protocol for evaluating it. The design has four parts: a per-node power model that separates idle from dynamic power, a per-function profile of execution time and payload size on each class of hardware, a placement rule that picks the lowest estimated energy among the nodes able to meet the latency deadline, and a keep-alive policy that weighs the standby power of a warm instance against the cost of a cold start. The evaluation protocol specifies workloads, baselines, metering and the form in which results should be reported, so that independent groups can compare schedulers on equal terms. Both the design and the protocol are proposals assembled from the published literature. Neither has been implemented or measured here, and the article sets out what an implementation would have to demonstrate.

**Keywords:** *Serverless computing, Function as a service, Energy-aware scheduling, Edge computing, Cold start, Energy proportionality*

© 2026 Mateo Yamamoto et al. This is an open access article under the CC BY license (<https://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

## 1. Introduction

Function as a service hides servers from the programmer. Code is uploaded as small stateless functions, the platform starts instances when requests arrive, and the bill follows execution time (Baldini et al., 2017; Jonas et al., 2019). The same abstraction is now offered outside the data centre. Open-source frameworks that run on Kubernetes can be installed on gateways and micro data centres close to sensors and users, which is where the edge computing literature has long argued that latency-sensitive work belongs (Shi et al., 2016).

\* Corresponding author: Mateo Yamamoto, Department of Computer Science, Sharavati Institute of Computer Science, Norway. E-mail: [mateo.yamamoto@sharavati.edu](mailto:mateo.yamamoto@sharavati.edu)

© 2026 Mateo Yamamoto et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

A deployment that spans edge and cloud is heterogeneous in every respect that matters for energy. An ARM single-board computer draws a few watts and executes a function slowly. A cloud server draws two orders of magnitude more, finishes sooner, and shares its idle power among many tenants. Between them lies a wireless or wide-area link whose use costs energy at both ends. A scheduler that ignores these differences can meet every latency target and still waste most of the energy it spends.

This article proposes a scheduler design that treats energy as the quantity to be minimised and latency as a constraint, together with a protocol for testing such a scheduler fairly. It is a methodological contribution. No prototype has been built and no measurements are reported; the claims made are about how the problem should be structured and what evidence would settle whether the design works.

## 2. Background

Three bodies of work supply the parts of the design: studies of how serverless platforms behave, the energy literature on servers and data centres, and research on offloading between devices, edge and cloud.

### 2.1. Serverless execution and where its energy goes

A function invocation is served either by a warm instance that is already loaded or by a cold start, in which a sandbox is created, the runtime is initialised and the code is fetched. Measurement studies of commercial platforms found cold starts to be slow and variable and showed that providers keep idle instances alive for minutes to avoid them (Wang et al., 2018). Lightweight virtual machine monitors such as Firecracker shortened sandbox creation considerably (Agache et al., 2020), but runtime and dependency loading remain. A production trace analysis by Shahrad et al. (2020) showed that most functions are invoked rarely, so a fixed keep-alive window holds memory for instances that are seldom reused, and it proposed windows adapted to each function's invocation pattern. Warm instances are not free in energy terms: they occupy memory and prevent the host from being consolidated or put into a low-power state.

### 2.2. Power behaviour of servers and small devices

Barroso and Hölzle (2007) observed that servers draw a large share of their peak power when idle and argued for energy-proportional machines. Fan et al. (2007) showed that a simple model, idle power plus a term that grows with processor utilisation, predicts the power of a server well enough for provisioning. These two results explain why consolidation saves energy in data centres and underlie the allocation heuristics of Beloglazov et al. (2012), which pack virtual machines onto few hosts and switch the others off. Frequency scaling, the other classical lever, yields less on recent hardware than it once did because static power has grown relative to dynamic power (Le Sueur and Heiser, 2010). Small edge boards behave differently: their idle power is low in absolute terms, but so is their throughput, and energy per request depends strongly on the workload.

### 2.3. Offloading and metering

The mobile edge computing literature frames offloading as a comparison between the energy of computing locally and the energy of transmitting the input and waiting for the answer, subject to a delay bound (Mao et al., 2017). The same comparison applies to a function that could run on the gateway where its trigger arrives or on a distant server. Applying it needs measurements. On Intel processors the running average power limit interface exposes energy counters for the package and memory (David et al., 2010), and models built on utilisation counters can apportion host power among co-located tenants (Kansal et al., 2010). Boards without such counters must be metered externally at the supply.

## 3. The Proposed Scheduler

The scheduler is conceived as an extension of the placement component of a Kubernetes-based function platform, since Kubernetes is the common substrate of edge and cloud clusters (Burns et al., 2016). It consists of four components, summarised in Table 1, that operate in a loop around every invocation.

**Table 1: Components of the proposed energy-aware scheduler, with their inputs, outputs and basis in prior work**

| Component              | Input                                                     | Output                                  | Basis                                   |
|------------------------|-----------------------------------------------------------|-----------------------------------------|-----------------------------------------|
| Node power model       | Idle power, peak power, current utilisation of each node  | Marginal power of adding one invocation | <a href="#">Fan et al., 2007</a>        |
| Function profile       | Past execution times and payload sizes per hardware class | Expected duration and transfer volume   | <a href="#">Shahrad et al., 2020</a>    |
| Transfer cost estimate | Link type and payload size between trigger and node       | Energy of moving input and result       | <a href="#">Mao et al., 2017</a>        |
| Placement rule         | Energy estimates and latency deadline                     | Chosen node for the invocation          | <a href="#">Beloglazov et al., 2012</a> |
| Keep-alive policy      | Inter-arrival history, standby power, cold-start energy   | Time to retain a warm instance          | <a href="#">Shahrad et al., 2020</a>    |
| Metering feedback      | Counter or supply readings after execution                | Corrected profile and model parameters  | <a href="#">Kansal et al., 2010</a>     |

### 3.1. Node power model

Each node is described by its idle power, its power at full load and a utilisation-to-power curve fitted once during commissioning. The quantity the scheduler needs is marginal: the additional energy the node will draw if it accepts one more invocation. On a node that is already active this is the dynamic power attributable to the function multiplied by its expected duration. On a node that would otherwise be idle or asleep, the cost of waking it and keeping it awake must be added. Charging idle power only to the invocation that causes it is what steers work towards machines that are already busy, the serverless analogue of consolidation.

### 3.2. Function profiles

For every function the scheduler keeps, per class of hardware, a running summary of execution time, memory footprint, input and output size, and cold-start duration. New functions have no history, so their first invocations are spread across hardware classes deliberately to populate the profile. Profiles are keyed by function version, since a new deployment can change behaviour completely, and they are updated after each execution from the metering feedback described below.

### 3.3. Placement rule

When a request arrives, the scheduler lists the nodes that hold or can start an instance of the function. For each it estimates completion time as queueing delay plus cold-start time, if no warm instance exists, plus execution and transfer time. Nodes whose estimate exceeds the function's deadline are discarded. For the remainder it estimates energy as the sum of marginal compute energy, cold-start energy where applicable, and transfer energy at both ends of the link. The request goes to the node with the lowest estimate. If no node meets the deadline, the fastest is chosen and the violation is logged. The rule is greedy and local by design: it runs on the request path and must cost far less than the function it places.

Figure 1 shows the loop formed by these steps. Its final arc matters as much as the decision itself, because the estimates are only as good as the profiles and power curves behind them.

### 3.4. Keep-alive policy

After an invocation completes, the instance may be retained. Retaining it costs standby energy, which is the memory it pins plus any share of idle power it prevents the node from shedding. Releasing it risks a cold start whose energy is known from the profile. The policy keeps the instance for as long as the

expected cold-start energy avoided, given the function's observed inter-arrival distribution, exceeds the standby energy spent. This is the histogram-based idea of [Shahrad et al. \(2020\)](#) with energy substituted for memory as the cost. On a battery or solar-powered edge node the standby term is weighted more heavily than on a mains-powered server.

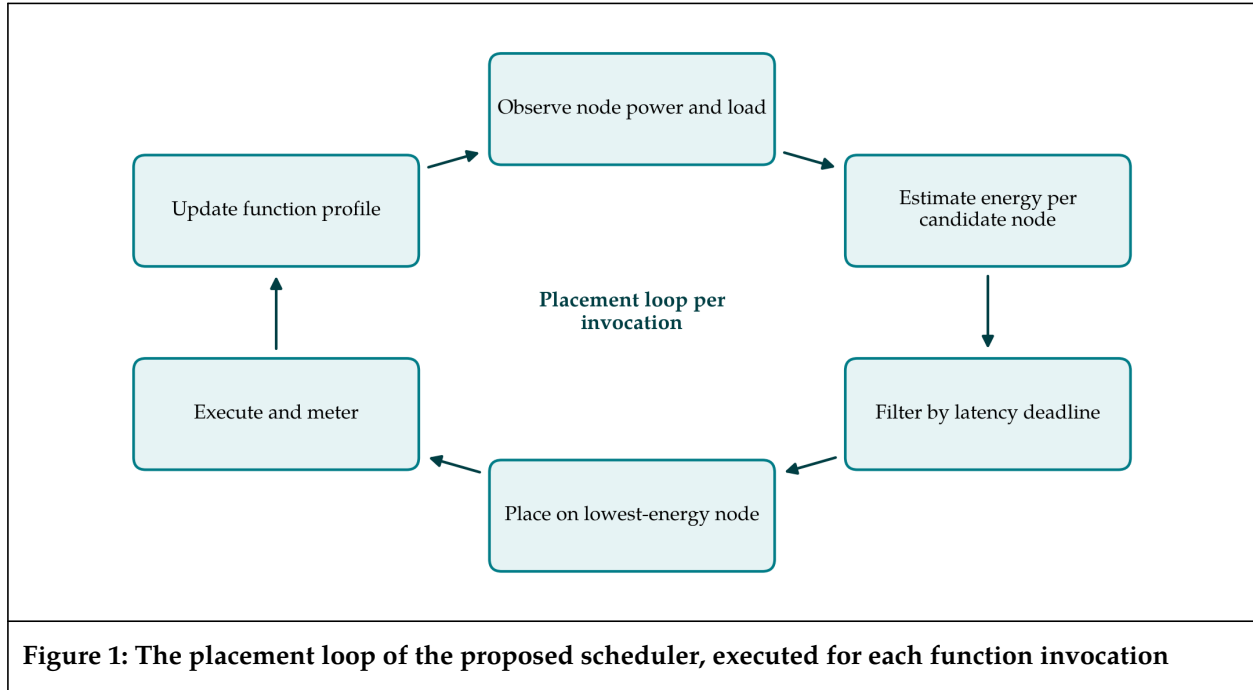


**Figure 1: The placement loop of the proposed scheduler, executed for each function invocation**

#### 4. Evaluation Protocol

A scheduler of this kind is easy to make look good against a weak baseline or a favourable workload. The protocol below is intended to prevent that, and follows the general discipline of experimental design for computer systems set out by [Jain \(1991\)](#).

The testbed should contain at least three hardware classes, for example an ARM board, a small x86 edge server and a cloud-class server, joined by links whose bandwidth and delay are controlled. Workloads should combine a published function benchmark suite such as SeBS ([Copik et al., 2021](#)), which covers compute-bound, input-output-bound and multimedia functions, with invocation patterns replayed from a public production trace so that the mix of frequent and rare functions is realistic. Synthetic constant-rate load alone is not sufficient, since keep-alive policies are trivial under it.

Four baselines are needed: the platform's default scheduler; a latency-greedy policy that always picks the fastest node; a cloud-only and an edge-only placement; and the proposed placement rule with a fixed keep-alive window, which isolates the contribution of the adaptive policy. Energy should be metered for whole nodes at the supply, with on-chip counters used as a cross-check where available, and the energy of network equipment should be either measured or declared out of scope.

Results should be reported as energy per successful invocation alongside the tail of the latency distribution and the fraction of deadlines missed, never as energy alone. Each configuration should be repeated enough times to give confidence intervals, and the trace, function code, power curves and scheduler configuration should be published with the results. Where the carbon intensity of the supply differs between sites or over time, a second analysis may weight energy by carbon, in the manner explored for delay-tolerant cloud workloads by [Wiesner et al. \(2021\)](#).

#### 5. Limitations and What Remains to Be Shown

The design has not been implemented, and several of its assumptions could fail in practice. The power model assumes that the energy of co-located functions adds up, which interference in caches and memory

bandwidth can violate. The profiles assume that past executions predict future ones, which holds poorly for functions whose running time depends on their input. The greedy rule ignores the effect of one placement on later ones and may fragment load across nodes that a global optimiser would have emptied. Whether these simplifications cost much energy is exactly what the evaluation protocol is meant to reveal.

The serverless model also limits what any scheduler can do. Functions that exchange large state through remote storage spend their energy on data movement however well they are placed, a structural weakness of the model identified by Hellerstein *et al.* (2019). Finally, the protocol measures operational energy only. Manufacturing energy of edge hardware, cooling overheads and the energy of the wider network fall outside it and should be acknowledged as such in any report that uses it.

## 6. Conclusion

Energy-aware scheduling of serverless functions across edge and cloud can be built from results that are individually well established: additive power models, per-function profiles, the offloading comparison and adaptive keep-alive. This article has combined them into a scheduler design that minimises estimated energy under a latency deadline and has specified how such a design should be tested. The design remains a proposal until it is implemented and measured, and the protocol is offered so that this can be done in a form others can check.

## Conflict of interest

To be declared by the author(s) before publication.

## Funding

To be declared by the author(s) before publication.

## References

- Agache, A., Brooker, M., Iordache, A., Liguori, A., Neugebauer, R., Piwonka, P. and Popa, D.-M. (2020). [Firecracker: Lightweight virtualization for serverless applications](#). *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 419-434.
- Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A. and Suter, P. (2017). [Serverless computing: Current trends and open problems](#). In: [Chaudhary, S., Somani, G. and Buyya, R. \(eds\), Research Advances in Cloud Computing](#). Springer, Singapore, 1-20. [https://doi.org/10.1007/978-981-10-5026-8\\_1](https://doi.org/10.1007/978-981-10-5026-8_1)
- Barroso, L.A. and Hölzle, U. (2007). [The case for energy-proportional computing](#). *Computer*, 40(12): 33-37. <https://doi.org/10.1109/MC.2007.443>
- Beloglazov, A., Abawajy, J. and Buyya, R. (2012). [Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing](#). *Future Generation Computer Systems*, 28(5): 755-768. <https://doi.org/10.1016/j.future.2011.04.017>
- Burns, B., Grant, B., Oppenheimer, D., Brewer, E. and Wilkes, J. (2016). [Borg, Omega, and Kubernetes](#). *Communications of the ACM*, 59(5): 50-57. <https://doi.org/10.1145/2890784>
- Copik, M., Kwasniewski, G., Besta, M., Podstawski, M. and Hoefler, T. (2021). [SeBS: A serverless benchmark suite for function-as-a-service computing](#). *Proceedings of the 22nd International Middleware Conference*.
- David, H., Gorbato, E., Hanebutte, U.R., Khanna, R. and Le, C. (2010). [RAPL: Memory power estimation and capping](#). *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, 189-194.
- Fan, X., Weber, W.-D. and Barroso, L.A. (2007). [Power provisioning for a warehouse-sized computer](#). *Proceedings of the 34th Annual International Symposium on Computer Architecture*, 13-23. <https://doi.org/10.1145/1250662.1250665>

- Hellerstein, J.M., Faleiro, J., Gonzalez, J.E., Schleier-Smith, J., Sreekanti, V., Tumanov, A. and Wu, C. (2019). *Serverless computing: One step forward, two steps back. Proceedings of the 9th Biennial Conference on Innovative Data Systems Research (CIDR 2019)*.
- Jain, R. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, New York.
- Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C.-C., Khandelwal, A., Pu, Q., Shankar, V., Carreira, J., Krauth, K., Yadwadkar, N., Gonzalez, J.E., Popa, R.A., Stoica, I. and Patterson, D.A. (2019). *Cloud programming simplified: A Berkeley view on serverless computing*. arXiv preprint arXiv:1902.03383.
- Kansal, A., Zhao, F., Liu, J., Kothari, N. and Bhattacharya, A.A. (2010). *Virtual machine power metering and provisioning. Proceedings of the 1st ACM Symposium on Cloud Computing*, 39-50.
- Le Sueur, E. and Heiser, G. (2010). *Dynamic voltage and frequency scaling: The laws of diminishing returns. Proceedings of the 2010 Workshop on Power Aware Computing and Systems (HotPower 10)*.
- Mao, Y., You, C., Zhang, J., Huang, K. and Letaief, K.B. (2017). *A survey on mobile edge computing: The communication perspective. IEEE Communications Surveys & Tutorials*, 19(4): 2322-2358. <https://doi.org/10.1109/COMST.2017.2745201>
- Shahrad, M., Fonseca, R., Goiri, Í., Chaudhry, G., Batum, P., Cooke, J., Laureano, E., Tresness, C., Russinovich, M. and Bianchini, R. (2020). *Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. Proceedings of the 2020 USENIX Annual Technical Conference*, 205-218.
- Shi, W., Cao, J., Zhang, Q., Li, Y. and Xu, L. (2016). *Edge computing: Vision and challenges. IEEE Internet of Things Journal*, 3(5): 637-646. <https://doi.org/10.1109/JIOT.2016.2579198>
- Wang, L., Li, M., Zhang, Y., Ristenpart, T. and Swift, M. (2018). *Peeking behind the curtains of serverless platforms. Proceedings of the 2018 USENIX Annual Technical Conference*, 133-146.
- Wiesner, P., Behnke, I., Scheinert, D., Gontarska, K. and Thamsen, L. (2021). *Let's wait awhile: How temporal workload shifting can reduce carbon emissions in the cloud. Proceedings of the 22nd International Middleware Conference*.