Energy Metrics for Edge Microservice Request Placement Strategies

Klervie Toczé*
k.m.tocze@vu.nl
Vrije Universiteit Amsterdam
The Netherlands

Simin Nadjm-Tehrani
simin.nadjm-tehrani@liu.se
Linköping University
Sweden

Abstract

Microservices are a way of splitting the logic of an application into small blocks that can be run on different computing units and used by other applications. It has been successful for cloud applications and is now increasingly used for edge applications. This new architecture brings many benefits but it makes deciding where a given service request should be executed (i.e. its placement) more complex as every small block needed for the request has to be placed.

In this paper, we investigate energy-centric request placement for services that use the microservice architecture, and specifically whether using different energy metrics for optimization leads to different placement strategies. We consider the problem as an instance of a traveling purchaser problem and propose an integer linear programming formulation. This formulation aims at minimizing energy consumption while respecting latency requirements. We consider two different energy consumption metrics, namely overall or marginal energy, when applied as a measure to determine a placement. Our simulations show that using different energy metrics indeed results in different request placements.

CCS Concepts

• **Computer systems organization → Cloud computing**; • **Networks → Cloud computing**; **Network resources allocation**.

Keywords

Energy metrics, optimization, edge/fog computing.

ACM Reference Format:
Klervie Toczé and Simin Nadjm-Tehrani. 2025. Energy Metrics for Edge Microservice Request Placement Strategies . In *Proceedings of the 16th ACM/SPEC International Conference on Performance Engineering (ICPE '25), May 5–9, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3676151.3719380

1 Introduction

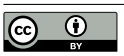
The edge computing paradigm, consisting of moving computational and storage resources to the edge of the network, is envisioned to obtain lower latencies, increase privacy and alleviate the amount of data sent to a distant cloud. As edge computing infrastructures are being deployed, the microservice architecture is subject to intensive study, both in the cloud and at the edge. With this architecture, services are decomposed into a chain of different functions, allowing for higher flexibility and sharing of the function logic between different services.

We consider a distributed edge infrastructure where each edge device receives service requests coming from end users through end devices. An end device can e.g. be an IoT device (a video surveillance camera, a sensor), a mobile phone, a connected vehicle, etc.. The edge computing infrastructure is composed of heterogeneous resource-limited devices. This means that functions cannot deploy function instances on all available edge devices as this will be too resource-hungry and also because some specific function instances can only run on specific edge hardware. For example, some machine learning algorithms require GPU resources to run.

In this work, we study microservice **request** placement, i.e. for a given set of function instances already deployed in the edge infrastructure, which one(s) to select for executing a particular request, incoming at a given physical location and at a given point in time. How to best deploy the function instances for improved performance and how many of them should be deployed are related placement problems which are tackled by other works [16, 22]. The request placement problem is critical to be addressed since it is where the demand side (the application requirements) and the supply side (the infrastructure provisioning) meet. The application provider is interested in the requests completing in time to guarantee a high quality of experience for its users. The edge infrastructure provider is interested because the request placement can be used to improve load balancing or load consolidation.

Providing (micro)services using edge computing consumes energy. As part of an effort to minimize the impact of edge computing on energy consumption, we want to study request placements with an energy-centric perspective. Indeed, a common optimization objective, as many edge applications are latency-critical, is to minimize the completion time of a service request. Instead, we argue that there is no benefit in minimizing the service completion time as long as it is below the deadline requirement, i.e. that its responsiveness is good enough. Therefore, we use the deadline as a constraint and choose to optimize with regards to energy consumption only. We formulate two different energy metrics with different underlying ideas: one considers the energy consumption as traditionally done and the other considers the energy consumption increment created by the current request being placed. Our goal is to investigate whether the choice of energy metric results in different placement decisions and if so, what are the characteristics of the difference obtained and which factors influence it. As an aid

*Part of the work was performed while the author was at Linköping University.



ICPE '25, Toronto, ON, Canada

ACM ISBN 979-8-4007-1073-5/2025/05
https://doi.org/10.1145/3676151.3719380

in this investigation, we formulate the microservice request placement problem as an instance of the Traveling Purchaser Problem (TPP). The corresponding optimization problem is solved using Integer Linear Programming (ILP).

The contributions of this paper are the following:

- We formulate two energy metrics leading to two energy-centric placement strategies.
- We express the microservice request placement problem as a Traveling Purchaser Problem and propose an Integer Linear Programming formulation for it, using the two formulated energy metrics for the two different optimization objectives.
- We analyze the obtained placements and study the impact of the two metrics on the placement decision.

The paper is organized as follows: Section 2 presents the system model. Section 3 describes the energy models used and the proposed metrics. Section 4 introduces the two problem formulations. The evaluation is presented in Section 5. This work is discussed in Section 6 and related works in Section 7. We conclude in Section 8.

2 System model

2.1 Edge infrastructure

The edge devices included in the considered part of the edge infrastructure are represented as a directed graph[1] $G = (V, L)$. Each vertex of the graph corresponds to an edge device and each link of the graph corresponds to a communication link between two edge devices. Thus, we assume we have information about which edge devices are in the system and how they are connected. Additionally, such information may include the device current utilization level, performance profile, and energy profile. Information about the links may include their propagation delay and transmission power. How this information is retrieved and where/how often this retrieval takes place is out of the scope of this paper.

2.2 Service

The end users want to use various *services*. For example, an augmented reality application is used to visualize an architectural project, or the video footage of a crossing needs to be analyzed.

In this paper, the considered services are following the microservice architecture, i.e. "an approach for developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms" [9]. Therefore, a service S can be defined as a directed acyclic graph $S = (F, D)$ where a vertex $f \in F$ is one of the functions (micro services) the service is composed of and an edge $d \in D$ is a dataflow between two different functions [11]. A service S is therefore a chain of functions connected with a sequence of edges in the same direction. A function $f \in F$ is associated with a computing size f^s and a dataflow $d \in D$ is associated with a data size d^s.

In order to provide a service S, function instances for every function $f \in F$ are deployed on different edge devices v present in the infrastructure. At most one function instance for a given function f is deployed on a given node v. We therefore introduce $\phi = (v, f)$ as a shorthand for describing the function instance for function f deployed on node v. The function instances can then handle incoming requests.

To execute a function which is a part of a service, an edge device needs to have the necessary hardware and software available. This is why all functions may not execute on all edge devices. The necessary software is for example provided using a dedicated container [13]. We denote $V_f \subseteq V$ the subset of edge devices able to execute a given function f, i.e. the subset of edge devices having a function instance for function f deployed. We consider that each function is available on a subset of edge devices only ($\forall f, V_f \subset V$) and that every function is available on at least one edge device ($\forall f, V_f \neq \emptyset$).

2.3 Service request

The load coming to the edge infrastructure consists of service requests. A service request $r = (S, v_b, v_e, \theta, \delta) \in R$ is defined by: the requested service S, the edge device receiving the request (beginning device) $v_b \in V$, the edge device receiving the request answer (end device) $v_e \in V$, the request arrival time θ, and the request deadline δ. The deadline δ is relative to the service request arrival time θ and corresponds to the maximum time allowed for the request to go through all the functions comprising the service and reach the destination device. In this work, we consider services for which it is necessary to complete requests before the deadline, otherwise their quality of service (QoS) is severely degraded.

2.4 Request placement

In this paper, we focus on the request placement problem. That is, for each request r with service S incoming to the edge system, the edge system has to decide on a placement p so that the request r will be completed before its deadline δ. We assume that 1) the edge devices have been provisioned and 2) a set of function instances has been placed on various edge devices in order to be able to provide for the services asked by the end users.

When handling a given service request, deciding on which function instance (and by extension on which edge device) each of the functions of a service S will be executed is called placing the request. Such a service request placement is denoted $p = (\phi_1, ..., \phi_{|F|})$ where ϕ_i corresponds to the function instance chosen for executing the i^{th} function of service S.

For a given placement p, the set of edge devices included in the placement $V_p \subseteq V$ (i.e. the ones where the selected function instances are deployed) is denoted by $V_p = \{v | \phi = (v, f) \in p\}$. $L_p \subseteq L$ is the set of links connecting these edge devices.

In this work, we focus on request placement taking place at one decision point at a time, with a specific current resource utilization. The aim is to be able to study how different optimization objectives will influence the placement decision.

2.5 Request completion time

Once the placement p of a service request r is known, it is possible to calculate the completion time of the request in order to see whether it is below the deadline δ. The completion time is composed of two parts: 1) the transmission time and 2) the execution time.

The transmission time depends on the link propagation delay (in ms), the available link bandwidth (in byte/ms), and the size of the

[1] We choose not to use the classic E notation for edges to avoid confusion with the edge of edge computing. Instead we use an L for "Links".

data (in byte) that needs to be transferred on the link. We assume that the request utilizes all the available link bandwidth to transmit. Therefore, the transmission time of dataflow d with data size d^s on a given link l is:

$$\lambda_{ld} = l^l + \frac{d^s}{l^c} \tag{1}$$

where l^l is the propagation delay for link l, and l^c is the currently available bandwidth for the link l. We assume that the link(s) chosen correspond to the shortest path between the two edge devices.

The execution time depends on the available computing capacity of the edge device (in million of instruction (MI) per millisecond) and the size of the function needing to be computed (in MI). Each function instance running on the edge device gets a share of the full capacity. This share (i.e. the available computing capacity for the function instance) can account e.g. for the need to always have some free capacity and thus may vary over time. We assume that the request utilizes all the available computing capacity in the corresponding function instance for the function execution. The execution time of the function instance $\phi = (v, f)$ corresponding to function f with computing size f^s deployed on edge device v is therefore calculated as follows:

$$\lambda_\phi = \frac{f^s}{\phi^c} \tag{2}$$

where ϕ^c is the available computing capacity allocated to ϕ.

The total completion time of a service request r to service $S = (F, D)$ using a given placement p is the sum of the transmission times and execution times for all link transfers and function executions required to complete the service request. This can be expressed as:

$$\Lambda_{rp} = \sum_{l \in L_p} \sum_{d \in D} \mathcal{I}^p_{ld} * \lambda_{ld} + \sum_{v \in V_p} \sum_{f \in F} \mathcal{I}^p_{vf} * \lambda_\phi$$

$$\text{where} \quad \mathcal{I}^p_{ld} \begin{cases} 1, & \text{if } d \text{ is sent over } l \text{ according to } p \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

$$\text{and} \quad \mathcal{I}^p_{vf} \begin{cases} 1, & \text{if } f \text{ executes on } v \text{ according to } p \\ 0, & \text{otherwise.} \end{cases}$$

3 Energy-centric placement

This work places the focus on the energy footprint of placement decisions and considers that lowering the energy consumption should be the sole optimization objective, with performance being a constraint. We want to study whether considering different energy metrics to optimize for can result in different placement decisions. In this section, we present the models used as well as the studied energy metrics and associated envisioned strategies.

3.1 Energy models

Following Baccarelli et al. [2] and Ahvar et al. [1], we model the energy consumed by an edge device or an edge link as having both a static part (i.e. the device/link being in the idle state) and a dynamic part (i.e. the energy needed for processing/transmitting).

3.1.1 Edge links. The energy used by an edge link l to service a request through the transmission of the dataflow d is calculated as follows:

$$E_{ld} = \underbrace{\mathcal{P}^{IDLE}_l * \lambda_{ld}}_{\text{static part}} + \underbrace{\mathcal{P}^{DYN}_l * \lambda_{ld}}_{\text{dynamic part}} \tag{4}$$

where $\mathcal{P}^{IDLE}_l$ is the idle power needed for maintaining the link l (e.g. the power consumed by the NIC cards at both ends when in the idle state) and $\mathcal{P}^{DYN}_l$ is the sum of the power needed by the link l for transmitting from the transmitting node and receiving at the receiving node. These are related to a wide range of characteristics of the communication link (e.g. number of antennas) and the current link throughput [2]. λ_{ld} is the duration of using l for transmitting d, according to Section 2.5.

3.1.2 Edge devices. The energy consumption for an edge device v used to service a request through the execution of the function f is:

$$E_\phi = \underbrace{\mathcal{P}^{IDLE}_v * \lambda_\phi}_{\text{static part}} + \underbrace{\mathcal{P}^{DYN}_v(u_\phi) * \lambda_\phi}_{\text{dynamic part}} \tag{5}$$

where $\mathcal{P}^{IDLE}_v$ is the power needed for device v to be on and $\mathcal{P}^{DYN}_v(u_\phi)$ is the extra power needed for device v for executing at a utilization level of u_ϕ (that includes the execution of the function instance ϕ). λ_ϕ is the duration of executing f on v, according to Section 2.5.

To model the dynamic power, we use the piecewise-linear model proposed by Ahvar et al. [1] with their measurements for the Parasilo server. Hence, the dynamic power is written as $\mathcal{P}^{DYN}_v(u_\phi) = (P^{cores}_{j+1} - P^{cores}_j) * k * u_\phi + [(j+1) * P^{cores}_j - j * P^{cores}_{j+1}]$ where $j/k \le u_\phi \le (j+1)/k$ with $j \in \{0, ..., k-1\}$ where k is the total number of cores that the devices has, and P^{cores}_j is the dynamic power consumption value when j cores are utilized.

Note that $\mathcal{P}^{IDLE}_v$ is the idle power of the *full* edge device. This corresponds to the worst case where only one function instance is deployed. If several function instances are deployed, one can define variants of Equation 5 where only an apportioned chunk of the idle power is taken into account as the energy consumption of a given instance. Determining this chunk can be done using energy apportionment [26]. Such variants are out of the scope of this work.

3.2 Energy metrics

We introduce two different energy metrics for calculating the energy consumption associated with executing a function instance ϕ:

(1) The overall energy consumption, i.e. how much the device will consume after placing the execution of f on top of what it is already executing.
(2) The marginal energy consumption, i.e. how much additional energy does the execution of f consumes on the device.

3.2.1 Associated envisioned placement strategies. Each of the above two metrics is associated with an envisioned placement strategy.

For the overall energy metric, the idea is to place the requests on the devices that will lead to the lowest energy consumption, when looking at the total energy consumption of all devices involved in the placement. This is inline with traditionally used energy consumption metrics.

For the marginal energy metric, the idea is to favor load consolidation, i.e. to avoid putting load on a device which is currently idle. Load consolidation makes it possible to gather the load on a

few devices and switch off the other ones instead of having more devices on but at a low utilization.

For both metrics, the idea behind considering the response time as a constraint only and not an optimization objective, is to allow for placement on devices that require some additional communication but offer better characteristics with regards to the target energy metrics than closer devices.

3.2.2 Overall energy. The overall energy consumption for an edge device v used to service a request through the execution of the function f is therefore:

$$E^O_\phi = E_\phi \qquad \text{as defined in Equation 5} \tag{6}$$

3.2.3 Marginal energy. The marginal energy consumption for an edge device v used to service a request through the execution of the function f depends on what was the utilization of device v before the execution of f (u_v). It is calculated as:

$$E^M_\phi = \begin{cases} E^O_\phi, & \text{if } u_v = 0 \\ \mathcal{P}^{DYN}_v (u_\phi - u_v) * \lambda_\phi, & \text{otherwise.} \end{cases} \tag{7}$$

The difference between the overall and marginal energy consumption for a given execution of a function instance represents whether we need to use a previously unused device for the execution (first line in Equation 7) or whether the function is added to an already used device. This is useful to know because if the static energy consumption represents an important part of the energy consumption, then an energy-efficient placement should favor the devices already in use instead of turning on new ones.

3.3 Request energy consumption

The request energy consumption, in a similar way to the request completion time (see Section 2.5), has two parts: 1) the transmission energy consumption and 2) the execution energy consumption.

Given the models from Section 3.1 and the two metrics from Section 3.2, the total overall/marginal energy consumption of a service request r to service $S = (F, D)$ with placement p is obtained by summing the corresponding energy consumption of all dataflow transmissions and function executions. This is expressed as:

$$\begin{aligned} E^O_{rp} &= \sum_{l \in L_p} \sum_{d \in D} \mathcal{I}^p_{ld} * E_{ld} + \sum_{v \in V_p} \sum_{f \in F} \mathcal{I}^p_{vf} * E^O_\phi \\ \text{and} \quad E^M_{rp} &= \sum_{l \in L_p} \sum_{d \in D} \mathcal{I}^p_{ld} * E_{ld} + \sum_{v \in V_p} \sum_{f \in F} \mathcal{I}^p_{vf} * E^M_\phi \\ \text{where} \quad \mathcal{I}^p_{ld} &\begin{cases} 1, & \text{if } d \text{ is sent over } l \text{ according to } p \\ 0, & \text{otherwise.} \end{cases} \\ \text{and} \quad \mathcal{I}^p_{vf} &\begin{cases} 1, & \text{if } f \text{ executes on } v \text{ according to } p \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{8}$$

4 Problem formulations

In this work, the request placement problem is solved for each individual request coming to the edge.

4.1 TPP formulation

The service request placement problem can be expressed as an instance of the Traveling Purchaser Problem (TPP) [12]. The TPP is a generalization of the well-known Traveling Salesman Problem (TSP), where there are different marketplaces that sell a given set of items at a given price. The problem is defined for a purchaser that has a given list of items to buy, to find the route between the marketplaces that minimizes both the cost of travel and purchase.

In our case, the purchaser corresponds to the service request that has to travel to different edge devices (marketplaces) offering the functions (items) composing the service requested (the list of items). The approach is energy-centric and considers the costs of travel and purchase to be the energy consumption of using a link or an edge device, according to their descriptions in Section 3.

In order to adopt this approach, we need to add two constraints to the generic formulation of the TPP. The first one is that there is a given order in which the functions should be executed, so that the service is executed properly. Secondly, it is not enough to minimize the cost of the request placement, the placement should also meet the request deadline to be an acceptable solution.

Symbol	Meaning
$x_{\phi\psi}$	Decision variable indicating whether the link between function instances ϕ and ψ is included in the solution
y_ϕ	Decision variable indicating whether the function instance ϕ is included in the solution
$o_{\phi\psi}$	Decision variable indicating the order in which the link between function instances ϕ and ψ is visited in the solution
Φ	Set of function instances
$F = (f_1, ..., f_{\lvert F \rvert})$	Ordered list of service functions to be placed
ϕ_b	Beginning function instance located on device v_b
ϕ_e	End function instance located on device v_e
$E_{(\phi,\psi)}$	Energy consumption of the link between any two function instances ϕ and ψ (including zero for modelling both being on the same device)
E_ϕ	Energy consumption of the function instance ϕ
a^ϕ_f	Boolean indicating whether function instance ϕ is an instance of function f (=1) or not (=0).
$O_f = i$	Positive integer representing the position of the function f in the list F
$\lambda_{\phi\psi}$	Transmission time for the link between the function instances ϕ and ψ
λ_ϕ	Execution time for the function instance ϕ
β	Integer upper bound used when ordering the visited links. It should be greater that the number of links contained in the service chain.

Table 1: ILP notations.

4.2 ILP formulation

An optimal solution to the problem can be found using integer linear programming (ILP). Table 1 summarizes the notations used.

4.3 The infrastructure graph

In order to enforce the ordering constraint between the different functions composing the service, we use a variable assigning an

integer to the links that are part of the placement (in a similar way to the work by Shameli-Sendi et al. [20]). In order to both enable a placement to have several service instances on the same physical node (good for load consolidation) and to allow the same edge device to be selected for different (non-consecutive) functions, we consider a device in the ILP infrastructure graph to be a function instance and not a hardware device. Consequently, a link is a virtual link between function instances, that can be mapped to a physical one. We also represent the beginning and end devices (v_b and v_e) as two specific "virtual" function instances denoted ϕ_b and ϕ_e respectively.

Note that since the nodes in the ILP graph are function instances, it is not necessary to specify both the edge device and the function (as a function instance is the combination of both) where referring to a node. This means that for the completion time and energy consumption equations (Equations 1 to 8), the subscript ld can be replaced by the corresponding start and end function instances.

4.4 Decision variables

This formulation uses three different decision variables. $x_{\phi\psi}$ indicates whether the link between function instances ϕ and ψ is selected to be part of the solution. y_ϕ indicates whether the function instance ϕ is included in the placement. $o_{\phi\psi}$ captures the order of visiting the links in the solution. If the link (ϕ, ψ) is not visited, then $o_{\phi\psi} = 0$. If the link (ϕ, ψ) is visited before the link (χ, ζ) then we have $o_{\phi\psi} > o_{\chi\zeta}$. In case the link (ϕ, ψ) is visited just before the link (χ, ζ) then we have $o_{\phi\psi} = o_{\chi\zeta} + 1$.

4.5 Objective function

The objective function in this work is focusing on energy consumption only. The performance requirements (here that a request should complete before a deadline), are considered as a constraint instead of an optimization objective. The energy consumption metric considered here is either the overall or the marginal one described in Section 3. This work is focusing on minimizing resource use, therefore the objective function is:

$$\text{Minimize} \sum_{\phi\in\Phi}\sum_{\psi\in\Phi} E_{\phi\psi} * x_{\phi\psi} + \sum_{\phi\in\Phi\setminus\{\phi_b,\phi_e\}} E_\phi * y_\phi$$

The device energy consumption metric E_ϕ in the equation above is replaced by the overall (E^O_ϕ) or the marginal (E^M_ϕ) energy consumption metric depending on the one chosen for optimizing.

4.6 Constraints

The ILP formulation contains the following constraints:

$$\sum_{\psi\in\Phi} x_{\psi\phi} = \sum_{\zeta\in\Phi} x_{\phi\zeta} \qquad \forall\phi \in \Phi \setminus \{\phi_b, \phi_e\} \tag{9}$$

$$\sum_{\psi\in\Phi} x_{\psi\phi} + 1 = \sum_{\zeta\in\Phi} x_{\phi\zeta} \qquad \phi = \phi_b \tag{10}$$

$$\sum_{\psi\in\Phi} x_{\psi\phi} = \sum_{\zeta\in\Phi} x_{\phi\zeta} + 1 \qquad \phi = \phi_e \tag{11}$$

$$\sum_{\phi\in\Phi\setminus\{\phi_b,\phi_e\}} (y_\phi * a^\phi_f) = 1 \qquad \forall f \in F \tag{12}$$

$$\sum_{\psi\in\Phi} x_{\phi\psi} - y_\phi = 0 \qquad \forall\phi \in \Phi \setminus \{\phi_b, \phi_e\} \tag{13}$$

$$\sum_{\phi\in\Phi} x_{\phi\psi} \le 1 \qquad \forall\psi \in \Phi \tag{14}$$

$$\sum_{\psi\in\Phi} x_{\phi\psi} \le 1 \qquad \forall\phi \in \Phi \tag{15}$$

$$o_{\phi\psi} \le \beta * x_{\phi\psi} \qquad \forall\phi \in \Phi, \forall\psi \in \Phi \tag{16}$$

$$\sum_{\psi\in\Phi} o_{\psi\phi} = \sum_{\psi\in\Phi} (o_{\phi\psi} + x_{\phi\psi}) \qquad \forall\phi \in \Phi \setminus \{\phi_b, \phi_e\} \tag{17}$$

$$\sum_{\psi\in\Phi} o_{\psi\phi} + \beta = \sum_{\psi\in\Phi} (o_{\phi\psi} + x_{\phi\psi}) \qquad \phi = \phi_b \tag{18}$$

$$\begin{aligned}(\sum_{\alpha\in\Phi\setminus\{\phi_e\}} o_{\alpha\phi}) - x_{\phi\psi} - \beta * y_\phi + \beta \ge \\ (\sum_{\alpha\in\Phi\setminus\{\phi_e\}} o_{\alpha\psi}) - \beta * y_\psi + \beta * \sum_{\alpha\in\Phi\setminus\{\phi_e\}} x_{\alpha\psi} \\ \forall\phi, \psi \in \Phi \setminus \{\phi_b, \phi_e\}, \phi \ne \psi, \\ \forall f, k \in F, f \ne k, O_f = O_k - 1, a^\phi_f = a^\psi_k = 1\end{aligned} \tag{19}$$

$$\sum_{\phi\in\Phi}\sum_{\psi\in\Phi} \lambda_{\phi\psi} * x_{\phi\psi} + \sum_{\phi\in\Phi\setminus\{\phi_b,\phi_e\}} \lambda_\phi * y_\phi \le \delta \tag{20}$$

$$\sum_{\phi\in\Phi} x_{\phi\phi} = 0 \tag{21}$$

$$x_{\phi\psi} \in \{0, 1\} \qquad \forall\phi, \psi \in \Phi \tag{22}$$

$$y_\phi \in \{0, 1\} \qquad \forall\phi \in \Phi \setminus \{\phi_b, \phi_e\} \tag{23}$$

$$o_{\phi\psi} \in \mathbb{N}, \qquad 0 \le o_{\phi\psi} \le \beta, \qquad \forall\phi, \psi \in \Phi \tag{24}$$

Constraint 9 ensures that for any function instance ϕ, there is one edge going in and out the function instance. For the special cases of the source (Constraint 10) and destination (Constraint 11), a virtual incoming (respectively outgoing) edge is added to them. Constraint 12 ensures that only one function instance is selected per function to be placed and that the function instance selected can actually run this function. Constraint 13 ensures that if a function instance is chosen to be included in the solution, then it has to be on the solution path. Constraints 14 and 15 ensure the solution does not include any cycle. This is needed since the ordering constraint can only assign one order number to each link, hence cycles are impossible. Constraint 16 ensures that if a link is not selected, the value of the corresponding $o_{\phi\psi}$ is set to 0. If a link is selected, the corresponding ordering variable $o_{\phi\psi}$ is less than β. Constraint 17 is used to calculate the values of $o_{\phi\psi}$ for each selected link. The special case of the start node is covered in Constraint 18. Constraint 19 ensures that the functions are placed in the indicated order. The order is specified by O, where if $O_l = O_k - 1$, it means that the function l is placed before function k and that the function instance ϕ where $y_\phi * a^\phi_l = 1$ is visited before the function instance ψ where $y_\psi * a^\psi_k = 1$. Constraint 20 ensures that the latency corresponding to the selected path (i.e. a set of function instances connected through ordered links) is below the latency requirement, i.e. the deadline of the service request. Constraints 22 and 23 indicate that the decision variables are binary. Constraint 24 indicates that the decision variable is a positive integer.

5 Evaluation

The proposed ILP formulation is implemented and evaluated. The question this evaluation should answer is: do the two optimization objectives actually lead to different placement decisions?

5.1 Edge system

The edge system considered in this evaluation has the same topology as the Abilene network [8]. The distances between the edge devices are scaled to correspond to a scenario where all the edge devices are spread within a neighborhood of a city. The propagation delay between two edge devices is taken as proportional to the distance between these two devices.

All the edge devices and links in the system have the same characteristics with regards to processing/transmitting capacity, as well as the same energy profile. The energy values are taken from the measurements performed by Ahvar et al. [1].

5.2 Service

The edge service considered in this evaluation is composed of four functions that are executed sequentially. We consider that the same device that issues the request waits for the result, meaning that the execution result has to be transmitted back to the beginning device after the service request execution is completed. An example of such a service is a mixed reality application where service requests contain images that have to be decoded (F1), analyzed to create a virtual representation of the scene (F2), modified with the addition of virtual elements (F3) and encoded (F4), before being sent back to the initial (issuing) device for rendering to the end user [23].

Moreover, the number of function instances for each function and their localization is predefined. We recall that where to place them and how many of them should be placed is out of the scope of this paper. In this evaluation, two function instances are initially deployed for each of the functions on two different edge devices and the edge devices chosen are different for the different functions.

5.3 Load scenario

The load in the evaluation consists of different utilization levels for the edge devices and communication links. These utilization levels indicate which share of the total computation/communication capacity is available for the request being placed. Each function instance gets up to one core of computing capacity. For this evaluation, the utilization levels (in percentage) are decided randomly using a Normal distribution with a standard deviation of 10.

5.4 Evaluation approach

The evaluation presented in this article consists of 40 runs per load level. Each run corresponds to a different random load scenario. The load level varies between 0 and 100 in steps of 10. In all runs, we place one request using our ILP formulation, both when minimizing overall energy consumption and also marginal energy consumption. These are named Overall and Marginal for short. The beginning device for the request is always the same. An extended evaluation, including additional load scenarios and varying the number of function instances or beginning device, is available online [24, 25].

The evaluation is performed on a Surface Laptop 5 equipped with an Intel Core i7-1265U CPU (2.7 GHz) and 16GB of RAM. Version 11.0.3 of the Gurobi Optimizer is used as the optimization solver. Parameter details, the code used for running the evaluation as well as the result files are provided as an artifact [24].

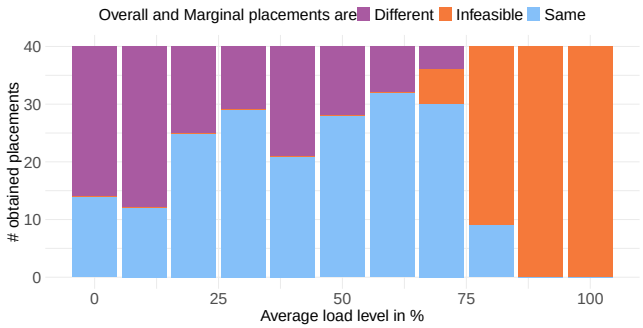


Figure 1: Categorization of the placements for a given request with varying load.

Optimization objective	Overall energy	Marginal energy
Request completion time	92 ms	90 ms
Overall energy consumed	2240 J	2289 J
Marginal energy consumed	906 J	850 J
Solver execution time	220 ms	70 ms

Table 2: Evaluation results for one request placement.

5.5 Outcomes

Figure 1 shows the number of times per load level in which the placement is infeasible/the same/different between the two optimization objectives. On this figure, it can be seen that it is indeed the case that optimizing for different objectives will lead to different placement decisions. How often this is the case varies strongly between the load levels (from 0% to 70%).

We now look into a specific run in order to analyze it in depth. The aim is to identify potential factors influencing the two objectives leading to different placement solutions. The specific run is randomly chosen among the 40 runs obtained for the load level 50% where the placement solution differs with respect to the two optimization objectives. This was the run with number 12. We select this load level as it represents a case when the edge infrastructure is neither underloaded nor overloaded. When the infrastructure is underloaded, there will be more devices that are idle, and hence, it is easier to find placements that differ between the two objectives (this is confirmed by Figure 1). When the infrastructure is overloaded, the number of placements that still meet the deadline requirements is limited, hence it is less likely that the placement will be different between the two objectives. It is therefore interesting to look at load levels that are in between.

As seen in Table 2, the different optimization objectives lead to similar request completion times. Regarding the energy metrics, the Overall alternative leads to 2.2% less overall energy consumed (- 49 J) than the Marginal one and the Marginal alternative leads to 6.6% less marginal energy consumed (- 56 J) than the Overall one. To understand the actual decisions better, the placement solutions

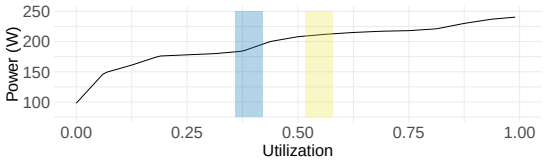


Figure 2: Power profile of a Parasilo device according to [1]

.

obtained are analyzed from the logs to understand how the difference appears. The placement solution differs in the following way between the different optimization objectives: functions 1, 3, and 4 are placed in the same function instances for both objectives, only the placement of function 2 is different. The difference in energy consumption shown in Table 2, which is small but non-negligible, is therefore the result of only one of the four functions being placed in a different way.

In the logs, we look deeper at the placement, and especially the placement of F2. It is placed on device 5 in the Overall case, and on device 7 in the Marginal case. For the run considered, the devices 5 and 7 differ in the following ways: 1) device 5 is less loaded than device 7, 2) the links used when choosing device 5 are less loaded than when using device 7, and 3) the latency on the links used when choosing device 5 is higher than when using device 7.

Therefore, the Overall energy metric seems to be able to leverage the time remaining to the deadline to pick edge devices located further away (leading to a higher completion time) but with lower utilization, and hence lower overall energy consumption. On the contrary, the Marginal energy metric favours a placement that is using edge devices with a higher utilization and closer.

To understand the logic behind the Marginal placement, it is useful to look at the considered power profile for edge devices. It is shown in Figure 2. The highlights correspond to the devices 5 (in blue) and 7 (in yellow), where the start of the highlight is the utilization before the function instance is placed and the end, the utilization after the placement. It can be seen that the increase in power is larger when placing on device 5 than on device 7, hence the choice of the Marginal alternative to place on device 7.

These initial observations are inline with the envisioned strategies associated with the different metrics (see Section 4.5), and give directions for further studies of factors influencing the different placement decisions. Several such studies are available online [25].

6 Discussion

The goal of this work was to investigate whether using different energy metrics as optimization objectives could lead to different energy-saving microservice request placement decisions, according to different strategies. To perform this study, we formulate the problem as an ILP and solve it optimally for a small-scale problem. This method is used for several reasons. First, the choice of a small-scale edge infrastructure and microservice deployment serves several purposes: 1) it is easier to analyze the outcomes as the solution space is limited, 2) it demonstrates the relevance of the idea in small setups while suggesting that it is even more useful for larger setups, and 3) it allows to solve the problem optimally in a reasonable time. Solving the problem optimally allows the conclusion to be drawn on the actual optimal and not some approximation. It is therefore a useful tool for understanding the proposed approach.

Of course, to implement this approach of using energy metrics to achieve a given placement strategy in practice, solving the ILP using a solver is not feasible as it takes too long to execute. Even if the solver execution time for performing the evaluation presented was short (mostly in the range of hundred milliseconds, up to a few seconds), it is still too important considering that the service considered has a deadline at 100 ms. Since the problem is formulated as a TPP instance, it is possible to use the numerous very good (meta-) heuristics which have been developed for such problems. Example of these are tabu search or Ant Colony Optimization. The model presented in this work can then be used a a reference to know how close to optimality these heuristics perform.

7 Related works

The microservice architecture [4] is being extensively considered in the neighboring field of cloud computing. It is a cornerstone of both the cloud-native [5] and serverless [11] computing. Several surveys have presented the microservice works in cloud computing, including focusing on anomaly detection [21] or practical dimensions such as Kubernetes scheduling [3]. Microservices are also increasingly used in the edge computing paradigm. The fine-grained modularity they offer is for example an asset for IoT applications [14].

Microservice request placement is tackled by Shameli-Sendi et al. [19, 20]. They also use a TPP formulation for simultaneously placing multiple service requests in a cloud environment. Contrary to this work, they do not consider energy, nor that the request placement has to meet a deadline. They also assume that all devices are able to provide all function instances, which is not the case in edge and therefore in our work.

Schneider et al [18] study placement problems at the edge and propose a solution for service coordination, which they define as combining the online scaling, placement, scheduling and routing of a service request. Their solution uses distributed deep reinforcement learning. They do not consider energy and assume that you can add function instances to serve the request.

Wang et al. [27] study the placement of function instances and requests in the context of the Internet of Vehicles. They present a two-layer system including three algorithms to 1) place the service requests to devices, 2) enable different service requests to use the same function instance and 3) reduce the number of service instances when they are not needed anymore. Instead, we focus on the first layer to understand how optimizing towards energy instead of performance can be used for different placement strategies.

Russo Russo et al. [17] propose a microservice request placement also with two levels. The first level is a simple heuristic algorithm that places every incoming request according to a probability distribution. The second level is optimizing these probabilities at regular time intervals. The optimization is performed using linear programming considering monetary cost and resource availability constraints. The aim is to maximize the requests satisfying their response time requirements. Contrary to our work, the energy consumption is not considered. Moreover, the optimization has a different purpose as it is not directly performing the placement.

There are several works considering energy consumption within edge computing, e.g. [6, 7, 10, 15]. To evaluate the proposed methods and techniques, relevant energy models are required. Baccarelli et al. [2] propose models for the energy consumption of both devices and communication links. Their focus is on virtualized and multi-core devices in the context of 5G. Similarly, Ahvar et al. [1] propose energy models for different types of so-called cloud-related architectures. It also includes models for both the devices and the communication links. Their aim is to compare different types of architectures (more or less distributed) with regards to their energy consumption. To the best of our knowledge, no study considers energy-centric microservice request placement.

8 Conclusion

In this work, we propose and investigate an energy-centric approach to place microservice requests: optimizing for different energy metrics. We formulate the problem as an instance of the TPP problem and solve the corresponding ILP as a tool to study the effect of different metrics and variations in the problem space.

Our evaluation suggests that optimizing using the two proposed energy metrics does lead to the envisioned placement strategies. In both cases, the completion time is a leverage to select devices that are most suitable for the objective but located further away. Higher heterogeneity in the system load and number of available function instances are factors leading to different placement decisions between the two objectives.

While our work initiates the study of energy metrics for energy-centric placement strategies, more work remains in the area. For example, a two-tier placement strategy combining the two optimization objectives based on the current system state would be interesting to develop. Moreover, alternative fast decision strategies for finding a good enough solution should be looked into to implement the approach in practice.

Acknowledgments

This work is supported by the Swedish national graduate school in computer science (CUGS). The second author was supported by ELLIIT, Excellence Center at Linköping-Lund on Information Technology.

References

[1] Ehsan Ahvar, Anne-Cécile Orgerie, and Adrien Lebre. 2022. Estimating Energy Consumption of Cloud, Fog, and Edge Computing Infrastructures. *IEEE Transactions on Sustainable Computing* 7, 2 (2022), 277–288. doi:10.1109/TSUSC.2019.2905900

[2] Enzo Baccarelli, Michele Scarpiniti, and Alireza Momenzadeh. 2019. EcoMobiFog–Design and Dynamic Optimization of a 5G Mobile-Fog-Cloud Multi-Tier Ecosystem for the Real-Time Distributed Execution of Stream Applications. *IEEE Access* 7 (2019), 55565–55608. doi:10.1109/ACCESS.2019.2913564

[3] Carmen Carrión. 2022. Kubernetes Scheduling: Taxonomy, Ongoing Issues and Challenges. *ACM Comput. Surv.* 55, 7, Article 138 (dec 2022), 37 pages. doi:10.1145/3539606

[4] Tomas Cerny, Michael J. Donahoo, and Michal Trnka. 2018. Contextual understanding of microservice architecture: current and future directions. *SIGAPP Appl. Comput. Rev.* 17, 4 (jan 2018), 29–45. doi:10.1145/3183628.3183631

[5] CNC Foundation. 2023. *Cloud Native Computing Foundation Charter.* Retrieved March 27, 2024 from https://github.com/cncf/foundation/blob/main/charter.md

[6] Wedan Emmanuel Gnibga, Anne Blavette, and Anne-Cécile Orgerie. 2023. Latency, Energy and Carbon Aware Collaborative Resource Allocation with Consolidation and QoS Degradation Strategies in Edge Computing. In *IEEE International Conference on Parallel and Distributed Systems (ICPADS)*. 2630–2639. doi:10.1109/ICPADS60453.2023.00349

[7] Hugo Hadjur, Doreid Ammar, and Laurent Lefevre. 2023. Services Orchestration at the Edge and in the Cloud for Energy-Aware Precision Beekeeping Systems. In *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 769–776. doi:10.1109/IPDPSW59300.2023.00129

[8] Simon Knight, Hung X. Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. 2011. The Internet Topology Zoo. *IEEE Journal on Selected Areas in Communications* 29, 9 (2011), 1765–1775. doi:10.1109/JSAC.2011.111002

[9] James Lewis and Martin Fowler. 2014. *Microservices: A definition of this new architectural term.* Retrieved September 6, 2022 from https://martinfowler.com/articles/microservices.html

[10] Wei Li, Ting Yang, Flavia C. Delicato, Paulo F. Pires, Zahir Tari, Samee U. Khan, and Albert Y. Zomaya. 2018. On Enabling Sustainable Edge Computing with Renewable Energy Resources. *IEEE Communications Magazine* 56, 5 (2018), 94–101. doi:10.1109/MCOM.2018.1700888

[11] Zijun Li, Linsong Guo, Jiagan Cheng, Quan Chen, Bingsheng He, and Minyi Guo. 2022. The Serverless Computing Survey: A Technical Primer for Design Architecture. *ACM Comput. Surv.* 54, 10s, Article 220 (sep 2022), 34 pages. doi:10.1145/3508360

[12] Daniele Manerba, Renata Mansini, and Jorge Riera-Ledesma. 2017. The Traveling Purchaser Problem and its variants. *European Journal of Operational Research* 259, 1 (2017), 1–18. doi:10.1016/j.ejor.2016.12.017

[13] Omogbai Oleghe. 2021. Container Placement and Migration in Edge Computing: Concept and Scheduling Models. *IEEE Access* 9 (2021), 68028–68043. doi:10.1109/ACCESS.2021.3077550

[14] Samodha Pallewatta, Vassilis Kostakos, and Rajkumar Buyya. 2023. Placement of Microservices-based IoT Applications in Fog Computing: A Taxonomy and Future Directions. *ACM Comput. Surv.* 55, 14s, Article 321 (jul 2023), 43 pages. doi:10.1145/3592598

[15] Giovanni Perin, Michele Berno, Tomaso Erseghe, and Michele Rossi. 2022. Towards Sustainable Edge Computing Through Renewable Energy Resources and Online, Distributed and Predictive Scheduling. *IEEE Transactions on Network and Service Management* 19, 1 (2022), 306–321. doi:10.1109/TNSM.2021.3112796

[16] Adam Rubak and Javid Taheri. 2024. Machine Learning for Predictive Resource Scaling of Microservices on Kubernetes Platforms. In *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing (UCC '23)*. Article 24, 8 pages. doi:10.1145/3603166.3632165

[17] Gabriele Russo Russo, Daniele Ferrarelli, Diana Pasquali, Valeria Cardellini, and Francesco Lo Presti. 2024. QoS-aware offloading policies for serverless functions in the Cloud-to-Edge continuum. *Future Generation Computer Systems* 156 (2024), 1–15. doi:10.1016/j.future.2024.02.019

[18] Stefan Schneider, Haydar Qarawlus, and Holger Karl. 2021. Distributed Online Service Coordination Using Deep Reinforcement Learning. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. 539–549. doi:10.1109/ICDCS51616.2021.00058

[19] Alireza Shameli-Sendi, Yosr Jarraya, Mohamed Fekih-Ahmed, Makan Pourzandi, Chamseddine Talhi, and Mohamed Cheriet. 2015. Optimal placement of sequentially ordered virtual security appliances in the cloud. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 818–821. doi:10.1109/INM.2015.7140384

[20] Alireza Shameli-Sendi, Yosr Jarraya, Makan Pourzandi, and Mohamed Cheriet. 2019. Efficient Provisioning of Security Service Function Chaining Using Network Security Defense Patterns. *IEEE Transactions on Services Computing* 12, 4 (2019), 534–549. doi:10.1109/TSC.2016.2616867

[21] Jacopo Soldani and Antonio Brogi. 2022. Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey. *ACM Comput. Surv.* 55, 3, Article 59 (feb 2022), 39 pages. doi:10.1145/3501297

[22] Klervie Toczé, Ali J. Fahs, Guillaume Pierre, and Simin Nadjm-Tehrani. 2023. VioLinn: Proximity-Aware Edge Placement with Dynamic and Elastic Resource Provisioning. *ACM Trans. Internet Things* 4, 1, Article 7 (feb 2023), 31 pages. doi:10.1145/3573125

[23] Klervie Toczé, Johan Lindqvist, and Simin Nadjm-Tehrani. 2020. Characterization and Modeling of an Edge Computing Mixed Reality Workload. *Journal of Cloud Computing* 9, 46 (2020). doi:10.1186/s13677-020-00190-x

[24] Klervie Toczé and Simin Nadjm-Tehrani. 2025. *Artifact: Energy Metrics for Edge Microservice Request Placement Strategies.* doi:10.5281/zenodo.14646315

[25] Klervie Toczé and Simin Nadjm-Tehrani. 2025. Energy Metrics for Edge Microservice Request Placement Strategies. arXiv:2503.09450 [cs.DC]

[26] Ekhiotz Jon Vergara, Simin Nadjm-Tehrani, and Mikael Asplund. 2016. Fairness and Incentive Considerations in Energy Apportionment Policies. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 2, 1, Article 2 (nov 2016), 29 pages. doi:10.1145/2970816

[27] Leilei Wang, Xiaoheng Deng, Jinsong Gui, Xuechen Chen, and Shaohua Wan. 2023. Microservice-Oriented Service Placement for Mobile Edge Computing in Sustainable Internet of Vehicles. *IEEE Transactions on Intelligent Transportation Systems* 24, 9 (2023), 10012–10026. doi:10.1109/TITS.2023.3274307