Complex Systems Informatics and Modeling Quarterly (CSIMQ)
eISSN: 2255-9922
Published online by RTU Press, https://csimq-journals.rtu.lv
Article 242, Issue 44, September/October 2025, Pages 17–30
https://doi.org/10.7250/csimq.2025-44.02

User-Centric and Community-Based Microservices Placement for Energy Efficiency

Imane Taleb and Jean-Loup Guillaume*

L3i - La Rochelle University, Bâtiment Pascal Avenue Michel Crépeau 17042
La Rochelle, France

imane.taleb@univ-lr.fr, jean-loup.guillaume@univ-lr.fr

Abstract. The growth of IoT and connected devices has increased demand for low-latency, energy-efficient processing across the Cloud-Fog-Edge continuum. While microservices enable scalable distributed computing, their placement remains challenging due to dynamic resource needs and interdependencies. This work proposes a graph-based microservice placement approach using user-centered local community detection. By integrating user nodes, our method adapts to shifting demands and resource availability, reducing energy consumption and communication overhead. Additionally, strategic mutualization and controlled duplication further enhance efficiency while preserving response time and resource constraints. Our results highlight the effectiveness of user-centric strategies in achieving scalable and sustainable deployments, reducing energy consumption by approximately 50% compared to state-of-the-art global methods while slightly improving deployment time.

Keywords: Microservices Placement, User-Centric Placement, Local Community Detection, Energy Efficiency.

1 Introduction

IoT and connected devices have transformed digital systems, driving the need for networks to process vast volumes of data with minimal delay. To ensure low latency and seamless interactivity, applications are moving closer to users [1]. The Cloud-Fog-Edge continuum addresses this by integrating computing and storage across the network. Fog computing extends cloud capabilities through localized nodes, enabling latency-sensitive applications like smart transportation, industrial automation, and healthcare. However, this geo-distributed and dynamic infrastructure poses node failures, mobility, and congestion challenges [2].

Microservices provide a scalable solution to these issues. Unlike monolithic systems, they consist of modular components that can be independently developed, deployed, and scaled. Yet, their interconnected nature, diverse resource requirements, and frequent communications make optimal placement complex [3]. Poor placement can lead to bottlenecks, higher costs, energy

* Corresponding author

Reference: I. Taleb and J.-L. Guillaume, "User-Centric and Community-Based Microservices Placement for Energy Efficiency", Complex Systems Informatics and Modeling Quarterly, CSIMQ, no. 44, pp. 17–30, 2025. Available: https://doi.org/10.7250/csimq.2025-44.02

Additional information. Author's ORCID iD: J.-L. Guillaume – https://orcid.org/0000-0002-4615-1563. PII S225599222500242X. Received: 6 August 2025. Accepted: 21 October 2025. Available online: 31 October 2025.

inefficiency, and degraded service quality. Therefore, placement strategies must account for the dynamic nature of the Cloud-Fog-Edge continuum, ensuring a balance between performance, cost, and energy efficiency for reliable and optimized deployments.

While many heuristics have been proposed for microservices placement to optimize performance and latency, or reduce costs, graph-based approaches are less commonly used, despite their clear relevance to the problem [4]. Both the communication network and the dependencies between application components naturally lend themselves to graph modeling, providing a structured and intuitive representation. Graph-based approaches for microservices placement often rely on classical community detection, where nodes in the network are partitioned globally, grouping network nodes that are close together, and placing frequently communicating microservices within the same community to minimize communication costs [5], [6], [7]. However, these approaches overlook the location of users requesting the services.

To address these limitations, we propose an extension of our previous work [6] that integrates user nodes into the network topology alongside traditional infrastructure elements. This inclusion provides a more comprehensive representation of the network, incorporating end-user devices to better capture real-world interactions. Our approach combines this enhanced network model with a heuristic that shifts from global to local community detection, prioritizing user-centered communities. This ensures that microservices requiring frequent communication are placed close to one another, and near the requesting users, reducing both communication latency and response time. Additionally, we optimize the placement through mutualization and controlled duplication of microservices. By grouping frequently interacting microservices and selectively duplicating others when necessary, our method minimizes communication overhead and maximizes resource efficiency. This not only lowers energy consumption but also improves system performance by aligning placement decisions with user demand patterns.

The main contributions of this work are as follows: (1) a user-centric microservices placement strategy that leverages local community detection, (2) the use of mutualization and controlled duplication to optimize resource usage and communication efficiency, and (3) a focus on energy efficiency by reducing consumption across network nodes and minimizing communication overhead.

The rest of the article is structured as follows. Related work is briefly discussed in Section 2. Models for network topology and microservices application graphs are presented in Section 3. A heuristic for energy efficiency microservices placement is explained in Section 4. Evaluation experiments and results are discussed in Section 5. Brief conclusions and future research opportunities are available in Section 6.

2 Related Work

Microservice placement is a critical challenge in distributed and resource-constrained environments. Among existing strategies, graph-based approaches stand out by explicitly modeling infrastructure nodes and service dependencies. By leveraging the inherent structure of network topologies, these methods enable systematic optimization aligned with various objectives, including latency, availability, and resource efficiency [4].

For instance, Saboor et al. [8] introduced a rank matrix optimization framework that uses eigenvector centrality to identify and group critical microservices into energy-efficient clusters. Similarly, Samani et al. [5] proposed a multilayer graph partitioning algorithm where resources such as CPU, memory, storage, and network are modeled as separate layers. Using the Louvain algorithm [9], nodes with similar attributes are grouped into clusters, enabling efficient placement in diverse environments. Lera et al. [7] extended this approach with a two-phase strategy: first, community detection is used to partition nodes and allocate applications based on deadlines; second, service assignment is performed using a first-fit method. In the context of micro-clouds,

Selimi et al. [10] developed a deployment strategy combining bandwidth- and availability-aware clustering using K-means to dynamically optimize service placement. Wang et al. [11] propose a latency estimator for microservice placement in Edge-Cloud Collaborative Smart Manufacturing to minimize latency under resource and location constraints. They model the system as a directed graph and applications as a directed acyclic graph (DAG) of microservice dependencies. Breadth-First Search is used to prioritize latency-sensitive services, assigning them to edge devices with minimal latency that meets constraints. Dynamic energy cost and carbon emission-efficient application (DECA), introduced in [12], optimizes energy- and carbon-efficiency using A* algorithm. Khan et al. [13] used the Leiden community detection algorithm and Integer Linear Programming to optimize latency-aware placement in federated edge-cloud architectures. In [6], we proposed a heuristic for microservice placement that uses global community detection to identify groups of nodes, and a greedy algorithm to allocate resources efficiently.

While most microservices placement strategies focus on latency to ensure a good user experience, the growing focus on sustainability has brought energy efficiency to the forefront. Kayal et al. [14] consider that the energy consumed is proportional to the CPU usage of each node. They proposed a decentralized placement model that reduces energy by limiting the number of microservices moved between nodes. Djemai et al. [15] use a metaheuristic to minimize energy consumption and calculate consumption according to the energy used by computations and network communications while taking into account the idle and loaded consumption states of physical nodes. In [12], the authors reduce the overall energy consumption of edge clouds by choosing locations where energy is cheaper, local, or with low carbon impact, and consolidating loads to reduce the number of active nodes and limit unnecessary migrations. Finally, in [6], we proposed to reduce energy consumption by minimizing communication distances between microservices and grouping them in highly connected communities. Yet, users are not considered in this approach.

The current literature reveals two key gaps in microservice placement: (1) a lack of strategies considering user-specific needs and dynamic demands, and (2) a limited focus on energy efficiency in handling user requests, despite its importance for sustainable deployment. To address these gaps, we propose a novel approach that integrates energy-aware measures into user-centric placement strategies using community detection.

3 Model Formulation

This section presents our models for network topology and microservices application graphs. Then we expand upon this model to discuss the microservices placement problem.

3.1 Network and Microservices Applications Models

We use a conventional network model where the Cloud-Fog-Edge continuum is a connected, undirected graph $G_P = (V_P, E_P)$, where V_P represents physical nodes and E_P denotes network links. This model divides the infrastructure into three distinct layers, as in [5], [16], and [17]. The **Cloud layer** contains high-performance data centers with virtually unlimited resources; the **Fog layer** is a set of intermediate nodes providing computational and storage services closer to end-users; and the **Edge layer** corresponds to end-user devices, such as IoT devices or smartphones, that initiate placement requests. This inclusion of user nodes addresses the limitations of previous models and allows a more realistic representation of user-driven placement requests and traffic patterns.

Each node $n_i \in V_P$ is characterized by its available resources: processing capacity cpu_i (MIPS), memory size ram_i (GB), and power consumption. Each link $l = \{n_i, n_j\} \in E_P$ connecting nodes n_i and n_j, is characterized by a bandwidth bw_l and a propagation delay Pr_l.

Given their interdependencies, microservices cannot be placed independently: if a user queries a microservice, all its dependent microservices may also be invoked to handle the request. In

telecommunication networks, the concept of Service Function Chain (SFC) represents an ordered sequence of dependencies to perform a specific task. More recent work [18] extends this concept by allowing tasks to be executed in parallel, modeling them as DAGs. We introduced a similar concept for microservices applications in [6]: a Microservices Function Path (MFP) begins with a source microservice, directly queried by the end-user, and includes all its direct and indirect successors, as illustrated in Figure 1. It therefore represents a potential execution flow.

More formally, applications are modeled as DAGs $G_S = (V_S, E_S)$, where V_S is the set of microservices, and E_S represents dependencies or function calls between microservices. Each microservice $s_k \in V_S$ is defined by its resource requirements (cpu_k, ram_k), while edges (s_k, s_l) are characterized by the amount of data exchanged ($data_{k,l}$). These DAGs capture the interdependencies between microservices and serve as the foundation for their placement across the Cloud-Fog-Edge continuum.

We define the depth of a node in an MFP as the length of the longest path from the source node to that node. Microservices at the same depth can execute in parallel, while microservices at different depths will execute sequentially if one is a direct and indirect successor of the other. The depth of an MFP is the highest depth of any node in the MFP.

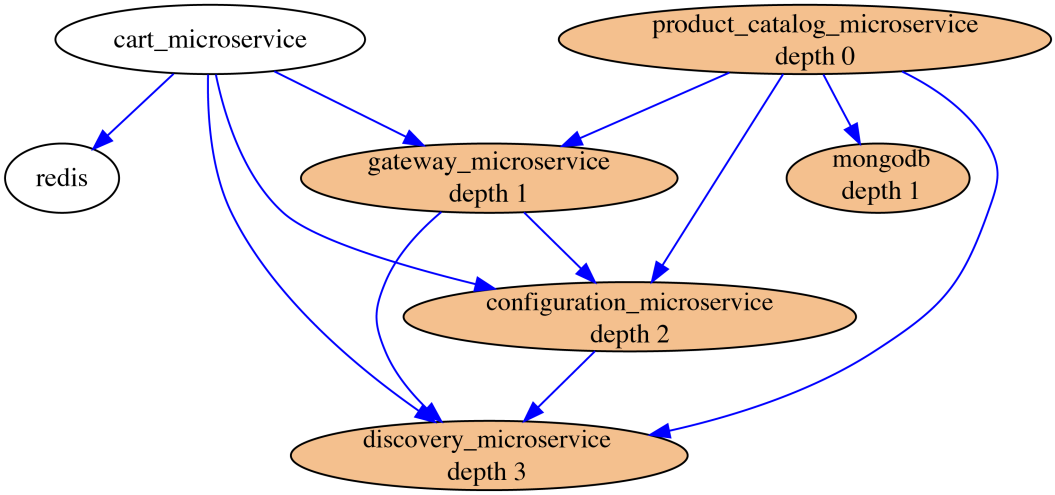


Figure 1. Example of dependencies of an E-commerce application from [19]. The MFP, having depth 3, whose source is "product_catalog" is highlighted. Another MFP, with "cart_microservice" as source, has shared microservices.

3.2 Energy Model for MFPs Placement

Efficient energy management is vital for ensuring optimal resource utilization. We define the total energy E_{tot} as the sum of two main components: the energy consumed by nodes for the execution of microservices and the energy required for communication across the network.

The energy consumed by a node n_i includes a fixed cost p_i^{idle} incurred when at least one microservice is running on n_i, and a term proportional to the fraction of CPU resources used by each microservice s_k deployed on n_i. As proposed in prior works [15], [20], [21], we assume a linear correlation between resource usage and energy consumption: the energy required by each microservice corresponds to a fraction of $\left(p_i^{\max} - p_i^{\text{idle}}\right)$. Formally, the energy consumed by n_i is:

$$E_{node}(n_i) = p_i^{idle}\delta(\_, n_i) + \sum_{\substack{s_k \in V_S \\ \delta(s_k, n_i)=1}} \frac{cpu_k}{cpu_i}\left(p_i^{\max} - p_i^{\text{idle}}\right) \quad (1)$$

where $\delta(s_k, n_i) = 1$ if microservice s_k is deployed on n_i and $\delta(\_, n_i) = 1$ if at least one microservice is deployed on n_i.

The communication energy when two microservices s_k and s_l, placed on nodes n_i and n_j respectively, is determined by the duration of data transmission, which includes the time to transfer the data $data_{k,l}$ over a link with bandwidth $bw_{i,j}$ and the propagation delay $Pr_{i,j}$. The total energy consumed includes contributions from both the sending node n_i and the receiving node n_j, accounting for the transition from idle to active states on each node. We consider that communication energy is negligible if two microservices are on the same node, due to the low energy cost of memory transfers, even though other impacts can happen when collocating services [22]. Deeper discussion on these impacts is beyond the scope of this paper. If n_i and n_j are not adjacent, all intermediate nodes along the path must remain active, contributing to the overall energy consumption. The communication energy between n_i and n_j is given by:

$$E_{\text{com}}(n_i, n_j) = \sum_{\substack{(s_k,s_l)\in V_S^2 \\ \delta(s_k,n_i)=1 \\ \delta(s_l,n_j)=1}} \left(\frac{data_{k,l}}{bw_{i,j}} + Pr_{i,j} \right) \left(p_i^{\max} - p_i^{\text{idle}} + p_j^{\max} - p_j^{\text{idle}} \right) \tag{2}$$

The total energy is then $E_{tot} = \sum_{n_i \in V_P} E_{node}(n_i) + \sum_{(n_i,n_j)\in V_P^2} E_{com}(i,j)$.

We aim to minimize this energy subject to each node's resource constraints. For instance: $\sum_{s_k \in V_S, \delta(s_k,n_i)=1} cpu_k \leq cpu_i, \quad \forall n_i \in V_P$ ensures that the node's available CPU is larger than the needs of the microservices deployed on it. Any other node-related constraint (e.g., RAM) can be expressed similarly.

This model, by accounting for both computational and communication energy while respecting resource constraints, offers a flexible framework that can be extended to include additional factors such as storage limitations.

4 Heuristic for Energy Efficiency Microservices Placement

In a nutshell, for a given MFP and its requesting users, our approach identifies central network nodes for this request based on the local communities of users and places the MFP on one or more of these nodes to minimize energy consumption. In a dynamic scenario, if some microservices are already deployed, we assess whether to reuse them or create duplicates.

4.1 Static Placement Algorithm

4.1.1 Local Community Detection

To ensure that microservices are close to the users requesting them, we compute user-centered communities and place the MFPs within. Specifically, for each user, we compute the local community around them to identify a set of potentially relevant network nodes. We evaluated several local community detection methods, as reviewed in [23]:

- *Personalized PageRank* [24] (PPR) is a variant of PageRank that simulates a "random walker" that starts from a node u and returns to it with a certain probability at each step. It estimates the local importance of the nodes around u and is computed as $\vec{p} = \alpha \cdot \vec{e}_u + (1-\alpha) \cdot P^T \vec{p}$, where $\vec{p}$ is the probability vector, $\vec{e}_u$ is the indicator vector (equal to 1 at node u, 0 elsewhere), $\alpha \in (0,1)$ is the teleportation probability, and P is the transition matrix.
- *Carryover Opinion* [25] (CO) assigns an influence score to each node by iteratively propagating these scores to neighboring nodes, with a progressive decrease in influence based on distance. Given the transition matrix M of the graph (normalized by degree), each iteration t consists of an update of the score vector $C_t(u) = M \cdot C_{t-1}(u)$, a rescaling $C_t(u) = \frac{C_t(u) - \min(C_t(u))}{1 - \min(C_t(u))}$, and a reset of the influence of u on itself $C_t(u)[u] = 1$.

- *KernelDiff* [26] (KD) is based on the analysis of local information diffusions. It models the diffusion of heat from a source node via a weighted version of the random walk, using the Taylor series of the matrix exponential. Then, a source vector $\vec{s}$ is diffused across the graph via a transition matrix M (normalized by degree). This diffusion is controlled by a time parameter t with an exponential decay $\vec{h} = e^{-t(I-M)}\vec{s}$, where I is the identity matrix.
- *FlowPro* [27] (FP) uses the flow of information through the graph, identifying groups of nodes connected by frequent data exchanges. A node belongs to the same community as the source s, if the information transmitted from it reaches the node with sufficient intensity, taking into account a natural attenuation based on distance. This decay is modeled by a naive estimate of the probability of a node's membership in the community, given by $p(x) = \rho^{-d(x)}$, and using a propagation constraint $p(x) \leq \frac{1}{|n(x)|}\sum_{y\in n(x)} p(y)$, where $d(x)$ is the distance from x to s, $\rho \in (0, 1)$ is a decay parameter for the flow, and $n(x)$ denotes the set of neighbors of node x.

All these vector updates are performed until convergence and are then used to detect the local communities. CO proved the most effective (see Section 5.2), and we therefore used it in our experiments.

4.1.2 Network's Mode Selection

Once the local community of each user requesting the MFP is determined, the system computes the intersection of these communities to identify network nodes close to all users, and the MFP is placed with priority on these nodes. We simply order these nodes based on their available resources (e.g., CPU, RAM), referred to as fitness hereafter.

If no intersection exists, or if none of the nodes in the intersection have sufficient resources, we apply a fallback strategy by computing a proximity score for each node based on user influence. We tested three variants: the *cumulative score* $\sum_{u\in U} \text{Score}_u(n)$, which favors nodes close to dominant users; the *minimum score* $\min_{u\in U} \text{Score}_u(n)$, which avoids penalizing any single distant user; and the *geometric mean* $\left(\prod_{u\in U} \text{Score}_u(n)\right)^{1/|U|}$, which smooths extremes. The geometric mean proved slightly more effective and we used it in our approach.

4.1.3 MFP Placement Strategy

From the previous step, we obtain a list of nodes ordered by fitness or score as potential candidates for MFP placement. If the MFP cannot be placed in its entirety due to insufficient resources on the best node, it is divided into two subsets of microservices[†]. We attempt to place the largest sub-MFP that fits on our candidate node. If it does not fit, we further divide it, and the process repeats with the smaller subparts. The microservices that do not fit will be placed on the next best node. This workflow is depicted in Figure 2.

After successfully placing an MFP, the system updates the network state and the available resources of the nodes where the MFP has been placed.

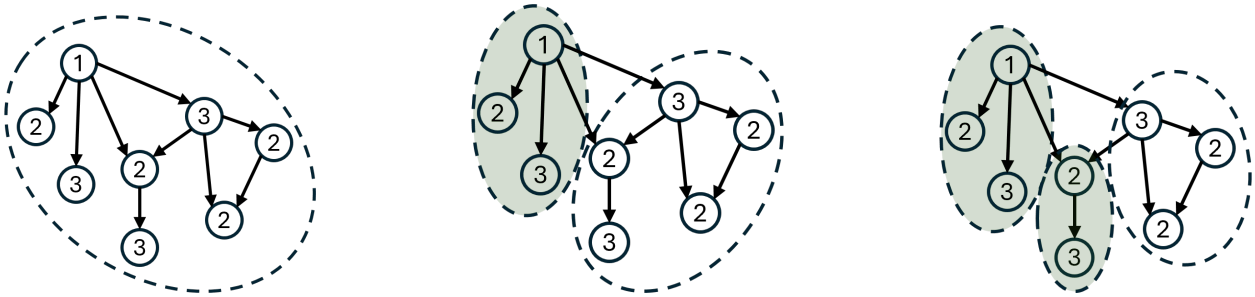
4.1.4 Mutualization of Microservices

Until now, each MFP has been placed independently. However, it is common for microservices to belong to more than one MFP (see Figure 1). To optimize resource utilization, we introduce a mutualization mechanism that reuses already deployed microservices whenever possible (see Algorithm 1).

Assume that a microservice m from MFP_1 has been deployed on a network node n and is used by a set of users U_1. We now want to deploy MFP_2, which also includes m and is requested by a set

[†] We use the Kernighan-Lin (KL) algorithm [28] to partition the MFP into two equally sized parts while minimizing the number of edges between them.

of users U_2. Since m is already placed, the first step is to verify if n has sufficient capacity to handle the requests from MFP_2. However, n may be distant from U_2, and the decision to reuse the existing deployment of m also depends on the distance between the users' communities. If the distance between the communities C_1 of the users of U_1 and C_2 of the users of U_2 is below a predefined threshold τ (i.e., if $d(C_1, C_2) \leq \tau$), mutualization is prioritized; otherwise, we duplicate m and place it close to U_2. Mutualized microservices reduce duplication and energy use.



(a) Node capacity: 11 **(b)** Node capacity: 5 **(c)** Node capacity: 0

Figure 2. Illustration of the placement of an MFP with 8 microservices requiring a total of 18 resources on a network node with a capacity of 11. Each node represents a microservice and its resource requirements. (2a) The target network node has an insufficient capacity for the full MFP. (2b) The MFP is bipartitioned into two sub-MFPs requiring 6 and 12 resources, respectively. The largest sub-MFP that fits (6 resources) is placed on the node, leaving 5 capacity. (2c) The remaining microservices are further bipartitioned into sub-MFPs requiring 5 and 7 resources. The first one fits (5 resources) and is placed, leaving 0 capacity. The remaining microservices will be placed on another node.

Algorithm 1 Heuristic Method for MFP Placement

Input Set of requests (user, MFP), Infrastructure Graph
Output Placement, Updated Resources

for each MFP M in the set of requests **do**
  $U \leftarrow \{u \in \text{Users} : (u, M) \in \text{requests}\}$
  $LC(U) \leftarrow \{community(u) : u \in U\}$
  $I \leftarrow$ Intersection($LC(U)$)
  $GeoScore \leftarrow$ GeometricScore(U)
  for each microservice $m \in M$ **do**
    if an instance of m has already been placed **then**
      Place m using Mutualization/Duplication and Update resources
  while M is not fully placed **do**
    if $I \neq \emptyset$ **then**
      $n \leftarrow$ Highest fitness node from I
    else
      $n \leftarrow$ Highest scored node from $GeoScore$
    Place as much as M on n using KL decomposition
    Update resources and fitness and Remove n from I and $GeoScore$
return Placement, Updated Resources

4.2 Dynamic Management of User Requests

To adapt the model for dynamic scenarios where user requests for an already placed MFP arrive progressively, we propose a two-fold strategy. If the local community of the new user intersects with those of users already linked to the MFP, we capitalize on the current setup and connect the

new user to the existing placement. When no intersection exists between the local community of the new user and those associated with the MFP, the system evaluates two options:

- Duplicating the MFP, which incurs an energy cost due to the hosting of the MFP on a new node.
- Keeping the MFP in its current location, which results in communication overhead that is determined by the distance between the new user and the MFP.

The system then selects the optimal action by balancing the duplication energy costs and the communication overhead. This adaptive strategy efficiently manages energy consumption, accommodates evolving user demands, and ensures system flexibility.

5 Evaluation and Results

To evaluate the performance of our approach under various scenarios, we ran experiments on a computer equipped with an Intel(R) Core (TM) i7-9850H CPU (2.60 GHz), 32GB of RAM, and Windows 10 Professional Education. The simulations were implemented in Python 3.9.13 using the NetworkX library for the graph algorithms. We thoroughly evaluated our approach to verify that (1) each MFP is fully deployed, (2) energy usage is minimized, (3) resource constraints are upheld, and finally (4) actual execution time stays close to the optimal value.

5.1 Evaluation Scenarios and Experimental Setup

The evaluation is designed to assess the performance of our heuristic under realistic scenarios and quantify the impact of successive improvements. Specifically, we compare four variants with the same objective function:

- **Global community** approach: This solution, proposed in [6], uses global community detection independent of the user location.
- **Forced duplication** approach: This strategy deploys each MFP completely independently and never considers mutualization.
- **Forced mutualization** approach: This variant mutualizes microservices whenever possible, eliminating redundant deployments at the risk of higher communication overhead.
- **Mutualization with duplication** approach: Our proposal that reuses already deployed microservices whenever possible and duplicates microservices to avoid excessive communication costs.

5.1.1 Network

Following the methodology outlined in [6] and [17], we used the Internet Zoo topology library dataset [29], a collection of real-world network topologies. We selected the topology of OTEGLOBE, a European operator providing telecommunications services to network operators. We focus on the largest connected component, retaining 81 nodes, and partition the topology into three types of nodes: (1) the most central node (based on its betweenness centrality) represents the Cloud node, (2) half of the remaining nodes (with the highest betweenness) are classified as Fog nodes, (3) the remaining are classified as Edge nodes. A more thorough analysis could be conducted on different topologies, including larger ones.

To simulate the characteristics of Cloud-Fog-Edge devices (e.g., number of cores, CPU speed, memory), we used a uniform random distribution, consistent with similar studies [5], [30]. Bandwidth and latency configurations in the Fog network are defined based on methodologies from previous research [5], [15]. Additionally, weights are assigned to the topology graph based on maximum and minimum electrical power values. Parameters are described in Table 1.

Users are placed on randomly selected Edge nodes. Each user can request between 1 and 5 MFPs, and each MFP is requested by at least two distinct users to ensure that no MFP is trivially tied to

a single user. To simulate a realistic request environment, the number of requests for each MFP follows a power-law distribution (with an exponent of 1.8), modeling access patterns in which a few microservices are highly requested while most are rarely used.

Table 1. Parameters (network and microservices) used in the experiments

Parameter (unit)	Value or range
Network	Zoo Topology Dataset OTEGLOBE [29]
Number of nodes/links	81 / 103
Type of nodes	1 Cloud, 40 Fog, 40 Edge
CPU (GIPS)	[250-500] (Cloud), [1.5-4] (Fog), [1-1.5] (Edge)
Ram (GB)	[100-250] (Cloud), [2.5-5] (Fog), [1-2] (Edge)
$p_{idle} - p_{max}$ (J)	145-320 (Cloud), 45-169 (Fog), 30-90 (Edge)
Bandwidth (KB/ms)	75
Propagation delay (ms)	1
Applications	Microservices dataset [19]
Microservices number	[5-25] depending on the application
CPU (MI)	[300-800]
Ram (MB)	[100-600]
Message size (KB)	[1500-4500]
Users	Request between 1 and 5 MFPs
Distribution	Power law (k), k = 1.8

5.1.2 Microservices Applications

We use the microservices dependency graph dataset from [19]. It contains 20 distinct application architectures, each built with 5 to 25 microservices. Dependencies between microservices are represented as DAGs. We use a uniform random distribution to assign values for various computing requirements, such as the CPU, the memory size, and the message size between microservices (see Table 1). To create a more realistic and challenging placement scenario, we duplicate the MFPs obtained from these applications.

5.2 Results Analysis

To evaluate and compare the performance of our heuristic across different scenarios, we use several key metrics. First, we measure the energy consumption of the deployment, including both intra-node energy usage and inter-node communication energy. We also examine the total number of nodes and links used for the placement as an indicator of the dispersion of microservices. To assess whether our placement strategy meets performance requirements, we compare the actual execution time of the MFPs with their optimal execution time. We further analyze the parameters influencing our heuristic, focusing particularly on the impact of the local community detection algorithm used and the parameter τ, which balances mutualization and duplication.

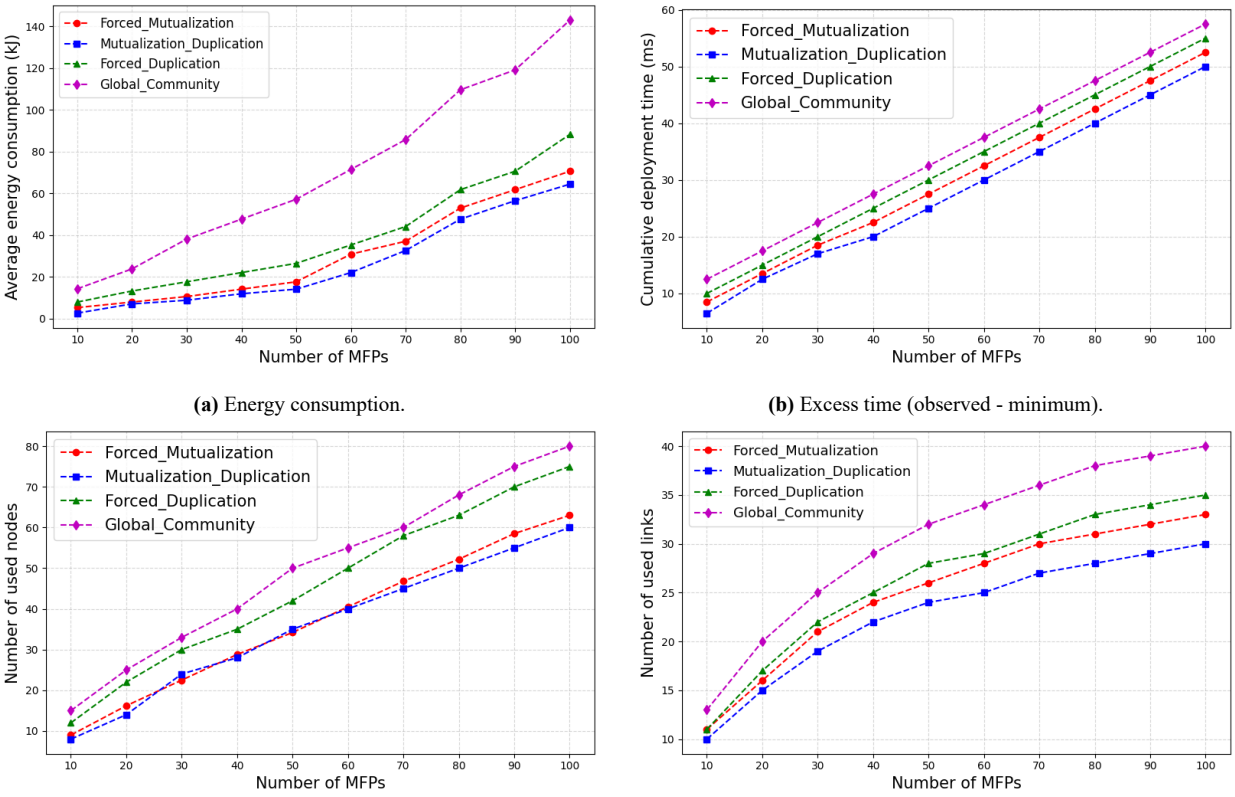
5.2.1 Energy Consumption of the Deployment

Figure 3a shows the clear advantage of the *Mutualization with duplication* approach, which achieves the lowest cumulative energy consumption (**63 kJ** for 100 MFPs), corresponding to a **50% reduction** compared to the *Global community* approach (*140 kJ*). The *Forced mutualization* and *Forced duplication* approaches, with an energy consumption of *70 kJ* and *90 kJ*, respectively, perform slightly worse due to the absence of duplication/mutualization, which increases communication costs. Finally, the *Global community* approach does not consider the

users, so the placement might be suitable in scenarios where placement must occur before user requests and cannot be adapted afterward. However, when user dynamics are taken into account, these results highlight the substantial energy savings enabled by local optimization with mutualization and duplication.

5.2.2 Total Number of Nodes and Links Used for the Placement

Figures 3c and 3d show that approaches with and without duplication exhibit similar trends, consistently using fewer nodes and links compared to the other approaches. This similarity can be attributed to their use of mutualization, which enables efficient grouping of microservices to minimize node usage. However, the *Mutualization with duplication* approach performs slightly better when the number of MFPs placed increases, as controlled duplication adds flexibility to placement, further optimizing resource utilization. In contrast, the *Forced duplication* approach uses significantly more nodes, as it neither employs mutualization nor duplication, resulting in less efficient microservices reuse. The most efficient solution (*Mutualization with duplication* approach) uses 25% fewer nodes than the *Global community* approach and 20% fewer nodes than the basic *Forced duplication* heuristic after the placement of 100 MFPs. Regarding the number of links used in Figure 3d, the trend is similar to that observed for nodes: at 100 MFPs, the *Mutualization with duplication* approach strategy achieves approximately a 25% reduction compared to *Global community*, confirming its efficiency.



(a) Energy consumption.

(b) Excess time (observed - minimum).

(c) Total number of Nodes used.

(d) Total number of Links used.

Figure 3. Comparison of energy consumption, execution time, and number of nodes and links used as a function of the number of MFPs, for the different microservices placement scenarios.

5.2.3 MFPs Execution Time

The execution time of an MFP depends on two key factors: the processing time at individual nodes and the communication delays between them. Microservices with dependencies execute

sequentially, whereas independent ones can run in parallel. In an ideal scenario where all microservices are placed on a single node, the execution time of an MFP is determined by the longest path in the DAG (i.e., the depth of the DAG, as detailed in Section 3.1), as microservices on that path must execute sequentially. However, when microservices are distributed across multiple nodes, additional communication delays are introduced for data transmission between nodes. Note that response times are omitted for simplicity, as they uniformly double transmission times and thus do not affect relative performance comparisons.

Figure 3b shows the difference between the actual execution time and the theoretical minimum execution time across different approaches. All approaches yield similar results, though the *Mutualization with duplication* approach performs slightly better, with a gap of **50 units** at 100 MFPs, remaining closest to the theoretical minimum. This suggests that even with mutualization and duplication, the overall impact on latency remains minimal.

5.3 Impact of the Parameters

Table 2a illustrates the cumulative rates of intersection between user communities identified by various algorithms as the number of MFPs increases. Among the tested methods, CO consistently outperforms the others, demonstrating significant and sustained growth in intersection rates and reaching up to 55 intersections per 100 MFP compared to PPR, KD, and FP (discussed in Section 4.1.1) with 49, 20, and 12 intersections respectively. When no intersection is found, our heuristic relies on the fallback mechanism, which is less efficient. Although energy consumption histograms are not shown, the intersection size directly translates to more energy-efficient placements as PPR, KD, and FP result in energy consumption increases of 2.5%, 18% and 33%, respectively.

We also tested several thresholds of τ to analyze its influence on the overall energy efficiency of the placement (see Table 2b). $\tau = 5$ emerged as the optimal value for balancing duplication and mutualization. When $\tau < 5$, duplication becomes more frequent, leading to increased energy consumption due to the deployment of additional microservice instances. Conversely, for $\tau > 5$, mutualization becomes more dominant, reusing already placed microservices. This reduces resource duplication but increases communication energy because of the greater distance between users and shared microservices. We use $\tau = 5$ in our experiments.

Table 2. (a) Number of nodes in the intersection depending on the proximity measure used.
(b) global energy (in 10^2 KJ) as a function of the parameter τ. Both with a varying number of MPFs placed.

Number of MFP	20	40	60	80	100
Personalized PageRank	8	19	28	41	49
Carryover Opinion	10	22	32	45	55
KernelDiff	3	9	13	18	20
FlowPro	3	5	6	8	12

(a)

Number of MFP	20	40	60	80	100
$\tau = 3$	100	170	360	610	800
$\tau = 5$	80	135	250	540	730
$\tau = 7$	100	160	350	600	790

(b)

6 Conclusion and Future Work

In this study, we presented a user-centric, energy-aware microservice placement strategy that combines local community detection with energy optimization. Our approach successfully bridges the often-overlooked trade-off between minimizing latency and reducing energy consumption – two critical yet frequently isolated objectives in existing literature. By leveraging local communities, we strategically position microservices near their users, boosting both system performance and energy efficiency.

While our method demonstrated substantial energy savings, the model that we used is still fairly simple. For instance, we did not use latency as a criterion for placement. Doing so would allow for service-level agreement constraints, ensuring that critical services are located closer to users. We also evaluated our model under partially dynamic network conditions. Although MFPs are placed incrementally, the setup does not fully reflect the complexity of real-world systems, where users may arrive unpredictably, shift behavior, or release resources at any time. Future work could build on this by exploring fully dynamic environments, accounting for varying user presence, fluctuating network loads, and resource availability. Additionally, incorporating awareness of the energy sources powering servers – prioritizing low-carbon or renewable energy – could further reduce the environmental impact of deployments. Our model is designed with adaptability in mind and can easily integrate such sustainability enhancements.

References

[1] M. Somayya, R. Ramaswamy, and T. Siddharth, "Internet of Things (IoT): A literature review," *Journal of Computer and Communications*, vol. 3, no. 05, p. 164, 2015. Available: https://doi.org/10.4236/jcc.2015.35021

[2] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, and O. Rana, "The internet of things, fog and cloud continuum: Integration and challenges," *Internet of Things*, vol. 3, pp. 134–155, 2018. Available: https://doi.org/10.1016/j.iot.2018.09.005

[3] S. Pallewatta, V. Kostakos, and R. Buyya, "Placement of microservices-based IoT applications in fog computing: A taxonomy and future directions," *ACM Computing Surveys*, vol. 55, no. 14s, pp. 1–43, 2023. Available: https://doi.org/10.1145/3592598

[4] I. Taleb, J.-L. Guillaume, and B. Duthil, "A Survey on Services Placement Algorithms in Integrated Cloud-Fog / Edge Computing," *ACM Computing Surveys*, vol. 57, no. 11, Jun. 2025. Available: https://doi.org/10.1145/3729214

[5] Z. N. Samani, N. Saurabh, and R. Prodan, "Multilayer Resource-aware Partitioning for Fog Application Placement," in *2021 IEEE 5th International Conference on Fog and Edge Computing (ICFEC)*. IEEE, 2021, pp. 9–18. Available: https://doi.org/10.1109/icfec51620.2021.00010

[6] I. Taleb, J.-L. Guillaume, and B. Duthil, "Energy-and Resource-Aware Graph-Based Microservices Placement in the Cloud-Fog-Edge Continuum," in *International Conference on Computational Science*. Springer, 2024, pp. 240–255. Available: https://doi.org/10.1007/978-3-031-63749-0_17

[7] I. Lera, C. Guerrero, and C. Juiz, "Availability-aware service placement policy in fog computing based on graph partitions," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3641–3651, 2019. Available: https://doi.org/10.1109/jiot.2018.2889511

[8] A. Saboor, A. K. Mahmood, A. H. Omar, M. F. Hassan, S. N. M. Shah, and A. Ahmadian, "Enabling rank-based distribution of microservices among containers for green cloud computing environment," *Peer-to-Peer Networking and Applications*, vol. 15, no. 1, pp. 77–91, 2022. Available: https://doi.org/10.1007/s12083-021-01218-y

[9] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008. Available: https://doi.org/10.1088/1742-5468/2008/10/p10008

[10] M. Selimi, L. Cerdà-Alabern, M. Sánchez-Artigas, F. Freitag, and L. Veiga, "Practical service placement approach for microservices architecture," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2017, pp. 401–410. Available: https://doi.org/10.1109/ccgrid.2017.28

[11] Y. Wang, C. Zhao, S. Yang, X. Ren, L. Wang, P. Zhao, and X. Yang, "MPCSM: Microservice placement for edge-cloud collaborative smart manufacturing," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 9, pp. 5898–5908, 2020. Available: https://doi.org/10.1109/tii.2020.3036406

[12] E. Ahvar, S. Ahvar, Z. A. Mann, N. Crespi, R. Glitho, and J. Garcia-Alfaro, "DECA: A Dynamic Energy Cost and Carbon Emission-Efficient Application Placement Method for Edge Clouds," *IEEE Access*, vol. 9, pp. 70 192–70 213, 2021. Available: https://doi.org/10.1109/access.2021.3075973

[13] S. Khan and S. Khan, "Latency aware graph-based microservice placement in the edge-cloud continuum," *Cluster Computing*, vol. 28, no. 2, p. 88, 2025. Available: https://doi.org/10.1007/s10586-024-04824-6

[14] P. Kayal and J. Liebeherr, "Autonomic service placement in fog computing," in *2019 IEEE 20th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2019, pp. 1–9. Available: https://doi.org/10.1109/wowmom.2019.8792989

[15] T. Djemai, P. Stolf, T. Monteil, and J.-M. Pierson, "A discrete particle swarm optimization approach for energy-efficient IoT services placement over fog infrastructures," in *18th International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE, 2019, pp. 32–40. Available: https://doi.org/10.1109/ispdc.2019.00020

[16] S. Pallewatta, V. Kostakos, and R. Buyya, "Microservices-based IoT application placement within heterogeneous and resource constrained fog computing environments," in *Proceedings of the 12th IEEE/ACM Intl Conference on Utility and Cloud Computing*, 2019, pp. 71–81. Available: https://doi.org/10.1145/3344341.3368800

[17] A. M. Maia and Y. Ghamri-Doudane, "A Deep Reinforcement Learning Approach for the Placement of Scalable Microservices in the Edge-to-Cloud Continuum," in *GLOBECOM 2023-2023 IEEE Global Communications Conference*. IEEE, 2023, pp. 479–485. Available: https://doi.org/10.1109/globecom54140.2023.10437143

[18] X. Lin, D. Guo, Y. Shen, G. Tang, and B. Ren, "DAG-SFC: Minimize the embedding cost of SFC with parallel VNFs," in *Proceedings of the 47th International Conference on Parallel Processing*, 2018, pp. 1–10. Available: https://doi.org/10.1145/3225058.3225111

[19] Rahman, I. Mohammad, P. Sebastiano, and T. Davide, "A curated Dataset of Microservices-Based Systems," in *Joint Proceedings of the Summer School on Software Maintenance and Evolution*. CEUR-WS, September 2019. Available: https://doi.org/10.48550/arXiv.1909.03249

[20] M. Ghobaei-Arani and A. Shahidinejad, "A cost-efficient IoT service placement approach using whale optimization algorithm in fog computing environment," *Expert Systems with Applications*, vol. 200, p. 117012, 2022. Available: https://doi.org/10.1016/j.eswa.2022.117012

[21] M. G. Mortazavi, M. H. Shirvani, and A. Dana, "A discrete cuckoo search algorithm for reliability-aware energy-efficient iot applications multi-service deployment in fog environment," in *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*. IEEE, 2022, pp. 1–6. Available: https://doi.org/10.1109/icecet55527.2022.9873056

[22] V. Varadarajan, T. Kooburat, B. Farley, T. Ristenpart, and M. M. Swift, "Resource-freeing attacks: improve your cloud performance (at your neighbor's expense)," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 281–292. Available: https://doi.org/10.1145/2382196.2382228

[23] G. Baltsou, K. Christopoulos, and K. Tsichlas, "Local community detection: A survey," *IEEE Access*, vol. 10, pp. 110 701–110 726, 2022. Available: https://doi.org/10.1109/access.2022.3213980

[24] R. Andersen, F. Chung, and K. Lang, "Local graph partitioning using pagerank vectors," in *2006 47th annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE, 2006, pp. 475–486. Available: https://doi.org/10.1109/focs.2006.44

[25] M. Danisch, J.-L. Guillaume, and B. Le Grand, "Multi-ego-centered communities in practice," *Social Network Analysis and Mining*, vol. 4, pp. 1–10, 2014. Available: https://doi.org/10.1007/s13278-014-0180-x

[26] K. Kloster and D. F. Gleich, "Heat kernel based community detection," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014, pp. 1386–1395. Available: https://doi.org/10.1145/2623330.2623706

[27] C. Panagiotakis, H. Papadakis, and P. Fragopoulou, "Local community detection via flow propagation," in *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, 2015, pp. 81–88. Available: https://doi.org/10.1145/2808797.2808892

[28] B. Hendrickson and R. W. Leland, "A Multi-Level Algorithm for Partitioning Graphs," in *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*. Association for Computing Machinery, 1995, p. 28–es. Available: https://doi.org/10.1145/224170.224228

[29] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011. Available: https://doi.org/10.1109/jsac.2011.111002

[30] C. Guerrero, I. Lera, and C. Juiz, "A lightweight decentralized service placement policy for performance optimization in fog computing," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 6, pp. 2435–2452, 2019. Available: https://doi.org/10.1007/s12652-018-0914-0