Energy- and Resource-Aware Graph-Based Microservices Placement in the Cloud-Fog-Edge Continuum

Imane Taleb[1], Jean-Loup Guillaume[1], and Benjamin Duthil[2]

[1] L3i, La Rochelle University, La Rochelle, France
[2] EIGSI, La Rochelle, France
{imane.taleb,jean-loup.guillaume}@univ-lr.fr

Abstract. The development of Cloud-Fog-Edge computing infrastructures in response to the rapid advance of IoT technologies requires applications to be positioned closer to users at the edge of the network. Characterised by a geographically distributed configuration with numerous heterogeneous nodes, these infrastructures face challenges such as node failures, mobility constraints, resource limitations and network congestion. To address these issues, the adoption of microservices-based application architectures has been encouraged. However, the interdependencies and function calls between services require careful optimisation, as each has unique resource requirements. In this paper, we propose a new model and heuristic for the placement of microservices in the Cloud-Fog-Edge continuum, based on community detection and a greedy algorithm to optimise energy use while taking into account the resource constraints and ensuring that response time is acceptable. This method aims to reduce energy consumption and network load, thereby improving the efficiency and sustainability of the infrastructure. Results have been compared with different scenarios and show that our approach can significantly reduce energy consumption and make efficient use of resources.

Keywords: Energy efficiency, Microservices placement, Sustainability, Cloud-Fog-Edge Continuum, Optimization, Graph partitioning.

1 Introduction

The development of smart technologies, particularly those associated with the Internet of Things (IoT), has played a central role in the ongoing transformation of the internet in response to user behaviour and emerging needs [1]. This evolution has led to the emergence of new applications, services and network infrastructures. Given the significant increase in data exchanges and the need for rapid response times, these applications need to be deployed as close as possible to users. The resulting reduction in latency and improved responsiveness of the services guarantee more efficient interaction and a better user experience. To meet these requirements, the Cloud-Fog-Edge continuum is emerging as a

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63749-0_17

key architectural solution, offering a strategic distribution of computing, storage resources and capacities closer to end users [2]. However, this infrastructure is also characterised by a geographically distributed configuration with numerous heterogeneous nodes facing challenges such as node mobility, failures, resource limitations and network congestion.

These constraints have encouraged the adoption of application architectures based on microservices, which are lightweight, flexible and modular modules, unlike traditional monolithic architectures. According to [3] microservices are defined as a series of small services that operate independently, offering modularity, ease of deployment and scalability. These are key advantages in heterogeneous and resource-constrained environments. However, microservices architectures involve dependencies and function calls between services, each of which has specific resource requirements and needs to be optimised. Inadequate positioning of microservices can cause high latency, over-consumption of energy and a higher environmental impact, increased costs and, more generally, impact on service availability and quality. This highlights the importance of adopting energy-efficient management and deployment strategies, particularly in the dynamic resource-constrained environments of the Cloud-Fog-Edge continuum.

Many researchers have focused on creating placement algorithms for distributed applications in Fog environments [4–8], focusing on metrics such as latency and quality of service. However, the specific placement of Fog applications based on microservices remains widely underexplored, particularly with regard to optimising energy consumption. For these reasons and since this is a complex multi-criteria optimisation problem, we propose a model and a heuristic for placing microservices in the Cloud-Fog-Edge continuum, based on community detection and a greedy algorithm. This approach aims to optimise energy consumption by minimising the communication distances between services, while taking into account node resource constraints and respecting response times. By strategically adapting microservices according to the interaction groups detected, we are able to reduce energy consumption and the load on the network.

This paper is structured as follows: Section 2 reviews recent research on microservice placement and load balancing. Section 3 describes the proposed model, and Section 4 details the heuristic and its constituent algorithms. Section 5 presents a comprehensive evaluation of the heuristic through various tests and analyses. Finally, the paper summarizes the main contributions in Section 6.

2 Background

Microservices placement methods mainly use meta-heuristics or graph-based approaches. Meta-heuristics encompass different approaches such as particle swarm optimisation (PSO) in its various forms, and different load balancing mechanisms to ensure efficient resource allocation. Graph-based methods are also used since they provide a systematic approach, using network topologies to refine placement decisions, offering a structured contrast to the adaptive and often probabilistic nature of meta-heuristic solutions.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63749-0_17

Pallewatta et al. [4] introduce a method focusing on resource balancing and service efficiency utilizing Multi-Objective PSO (MOPSO) for optimal microservice instance creation, coupled with Load-Balancing Request Routing (LBRR) and Load-Balancing Instance Placement (LBIP) for equitable service request allocation and CPU utilization. The authors [6] approach microservices placement as a bi-criteria optimization problem. They also use a PSO-based meta-heuristic to establish instances and identify the Pareto front, alongside techniques for evenly distributing requests akin to an unbounded bin-packing problem. Djemai et al. [9] formulated the placement problem as an optimization task, focusing on minimizing energy consumption and reducing delays in IoT applications. The approach uses a Discrete Particle Swarm Optimization (DPSO) algorithm, which operates with real-valued velocities to find optimal service locations. The physical setup is mapped as a graph, while IoT applications are represented as Directed Acyclic Graphs (DAGs). Mortazavi et al. [10] introduce a custom cuckoo search algorithm for service placement (CSA-SP) to optimize the positioning of services in fog nodes, aiming to minimize energy consumption while considering data transfer constraints and resource availability. The article [11] outlines a self-managing approach using the Whale Optimization Algorithm (WOA) for IoT services deployment across a three-tiered fog architecture, focusing on Quality of Service (QoS) for enhanced throughput.

Saboor et al. [12] propose a mathematical framework for the dynamic placement of container microservices based on rank matrix optimization, utilizing the stochastic matrix from microservices call graphs. They use eigenvector centrality to identify the most central microservices of an application, which will be grouped together in energy-efficient containers. Samani et al. [13] introduce a multilayer partitioning algorithm for Fog computing, tackling devices diversity by depicting resources as a four-layered graph, each reflecting similarities in network, CPU, memory, and storage among Fog nodes. Utilizing the Louvain algorithm [14], nodes are grouped by attributes, leading to a condensed graph of averaged clusters, thereby enabling application placement in similar resource environments. In [8] the authors propose a two-phase strategy for Fog computing: first, using community detection to map and partition nodes, placing applications based on deadlines; second, allocating services within Fog communities, using first-fit. Selimi et al. [15] propose a streamlined service deployment method for services in community micro-clouds, using a set number of clusters and network state information. Their Bandwidth and Availability-aware Service Placement algorithm involves node clustering with K-means, cluster head selection for optimal bandwidth, and dynamic service placement.

Authors in [7] introduce DECA, a method for energy and carbon-efficient placement in Edge Clouds, optimizing setup and migration to cut costs and emissions. Utilizing graph models and an A* algorithm, DECA considers geographical data and CO2 outputs, aiming for a balance between energy use and expense. This strategic placement enhances both environmental sustainability and operational efficiency. In [5] authors present a fully decentralized placement strategy designed for Fog-Edge environments, utilizing Markov approximation to

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63749-0_17

optimize communication among microservices, starting with random placement on Fog nodes and adjusting based on a Markov Chain representation.

In light of this state of the art, existing methods focus primarily on optimising availability, resource allocation, minimising delays and improving service quality. However, these approaches often neglect the critical dimension of energy efficiency, which is paramount given growing environmental concerns and the escalating costs associated with energy consumption in large-scale deployments. Our approach aims to fill this gap by specifically targeting the reduction of energy consumption in microservices placement. By incorporating energy-sensitive measures into our placement strategy, we aim not only to improve the efficiency of resource use and communication within the network, but also to significantly reduce the overall energy footprint of microservices deployment.

3 Model formulation

This section presents our model. We first describe our network topology and microservices applications graphs. Then we expand upon this model to discuss the microservices placement problem.

3.1 Network infrastructure

We propose a network model for the Cloud-Fog-Edge continuum. Each computing node is responsible for processing application placement requests. These nodes, spread across various locations, are heterogeneous in terms of geographical placement and available resources, including processing power and memory. The ability to adapt to these dynamic conditions is crucial to maintaining network functionality and ensuring continuous operation despite the inherent challenges posed by outages, node movement and varying levels of network congestion.

In our work, we consider three layers in the network architecture: Cloud layer, Fog layer and Edge/IoT layer. The devices of each level have specific characteristics and have resources to host and execute services.

- Cloud layer: refers to data centers that possess high-performance computing resources and resources high enough to be considered unlimited.
- Fog layer: is situated between the Cloud and the end-users. These nodes offer computational and storage services in close proximity to the end-users.
- Edge layer, or client layer: corresponds to a set of user devices from which placement requests originate.

We model the physical topology of Cloud-Fog-Edge architecture as a connected undirected graph $G_P = (V_P, E_P)$, where vertices represent physical execution nodes and edges are network links between these devices. We denote V_P the set of physical nodes. Each of these nodes $n_i \in V_P$ has the following characteristics: a speed of processing capacity cpu_i in MIPS, a memory size ram_i in GB and a power consumption ranging from p_i^{idle} when the device in on but not in use to p_i^{max} when used to the maximum, with a linear increase between

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63749-0_17

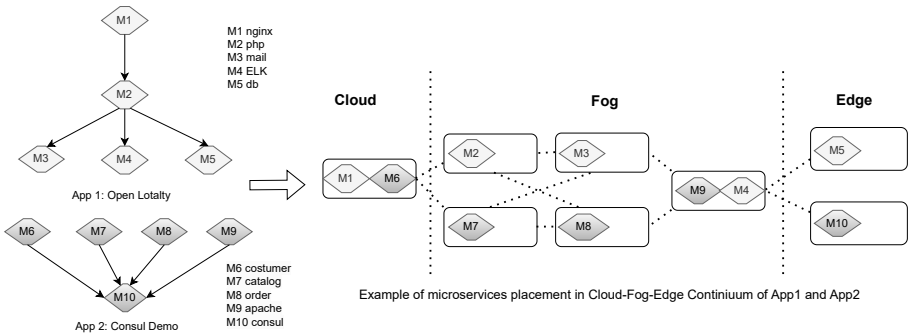


Fig. 1: Deployment and placement example of two microservices applications provided from the dataset [16].

these two values. Each physical link $l = \{n_i, n_j\} \in E_P$ is identified by the two nodes n_i and n_j it connects and is characterized by a bandwidth $bw_l = bw_{i,j}$ and a propagation delay $Pr_l = Pr_{i,j}$. Following this, we can build a logical link l'. We assume that if a node can communicate with another non adjacent node in the network, we can add a logical link between these two nodes. This link represents the shortest path P in terms of propagation delay composed of a sequence of physical links between the two nodes. $bw_{l'} = min_{l \in P}(bw_l)$ and $Pr_{l'} = \sum_{l \in P}(Pr_l)$, i.e. the minimum bandwidth on the path and the sum of the propagation delay on the path. Therefore, the network connections represent a complete graph with the consideration of physical and logical links.

3.2 Microservices applications and function paths

In our research, applications are a collection of microservices. Each of them is implemented as a standalone container, with specified resource needs. These microservices communicate, inter operate with each other and exchange data via function calls (API) to accomplish a specific application task. A micro-services application can be modeled as a directed acyclic graph (DAG) $G_S = (V_S, E_S)$ where the nodes represent the services $V_S = \{s_1, s_2, ..., s_t)$ and the edges are the dependencies and the function calls between the services. Each service $s_k \in V_S$ requires some resources consumption: a requested CPU cpu_k in million instructions and a requested RAM ram_k in GB. Each directed link $(s_k, s_l) \in E_S$ that connects s_k to s_l represents the request need and the data dependencies between these two microservices. It is characterised by a message size $data_{k,l}$.

We define three distinct categories of services, depending on the origin of the service request: firstly, sensors and actuators, for which requests come only from users, these components act as source nodes within the DAG application. The second category comprises internal services, responsible for processing operations

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63749-0_17

requested exclusively by other microservices. The third category represents the final microservices, which constitute the sinks in the DAG.

Within the described microservices graph framework, we introduce the concept of Microservice Function Paths, or MFP, denoting an organized sequence of dependencies designed to execute a particular task. Essentially, an MFP encapsulates a microservice source and all its direct and indirect successors, extending down to the sinks. An MFP therefore consists of one and only one source, but may have several sinks as shown in Figure 2. This graph has 4 sources and is therefore made up of 4 MFP. In microservices architecture, source nodes serve as the main entry points for user requests. The descendants of these nodes, and their connections, represent the dependency network and the trajectory of function calls throughout the system. We design the process in such a way that each action is initiated by a source node and traverses the network, ultimately ending at terminal nodes, which correspond to the final microservices where transactions or processes are completed.

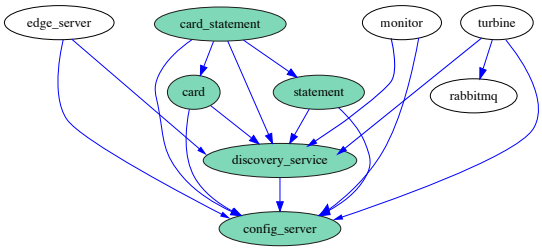


Fig. 2: Example of microservices dependencies graph of an IT blog called the card statement application taken from the dataset [16]. One MFP is highlighted.

3.3 Energy efficiency placement problem

In the context of Cloud-Fog-Edge computing, effective management of energy is crucial to maintain an optimal use of resources. Our goal is to place microservices in the network graph in a way that minimises energy consumption. We consider that the total energy consumption is the sum of energy from node execution and energy from network communication. Energy used by node n_i, see Equation (1), is composed of a fixed term p_i^{idle} if there is at least one microservice placed on n_i plus the fraction of CPU used by each microservice s_k placed on node n_i. Similar to the works of [9–11], we consider a linear correlation between resources utilisation and energy consumption and therefore each microservice use a fraction of $\left(p_i^{\max} - p_i^{\text{idle}}\right)$. Similarly, a communication between two microservices s_k and s_l uses some energy on both end nodes n_i and n_j involved for the total duration (transmission time plus propagation delay Pr) of the transmission, see Equation (2). If two microservices are placed on the same physical node there is

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63749-0_17

no communication (except memory transfers that have very low consumption) and we therefore consider the energy to be zero. Also, if two microservices are not placed on physically adjacent nodes but use a logical link, all intermediate nodes are switched on even if they are not used for computation.

$$\begin{cases} \text{Minimize } E_{tot} = \sum\limits_{n_i \in V_P} E_{node}(n_i) + \sum\limits_{(n_i,n_j) \in V_P^2} E_{com}(i,j) \\ \text{such that all capacity constraints are respected} \end{cases}$$

$$E_{node}(n_i) = \overbrace{p_i^{idle}\delta(\_,n_i)}^{\text{Node } n_i \text{ is on}} + \sum_{s_k \in V_S} \delta(s_k,n_i) \underbrace{\frac{cpu_k}{cpu_i}\left(p_i^{\max} - p_i^{\text{idle}}\right)}_{\text{Fraction of CPU for MS } s_k} \tag{1}$$

$$E_{\text{com}}(n_i,n_j) = \sum_{(s_k,s_l) \in V_S^2} \delta(s_k,n_i)\delta(s_l,n_j) \overbrace{\left(\frac{data_{k,l}}{bw_{i,j}} + Pr_{i,j}\right)}^{\text{Transmission time}} \underbrace{\left[p_i^{\max} + p_j^{\max} - p_i^{\text{idle}} - p_j^{\text{idle}}\right]}_{\text{Energy used on both ends}} \tag{2}$$

Where $\delta(s_k,n_i)$ is equal to 1 if microservice s_k is placed on node n_i and 0 otherwise, and $\delta(\_,n_i)$ is equal to 1 if there is at least one microservice s_k such that $\delta(s_k,n_i) = 1$. Note that the communication energy is only described for physical links but the same applies for all the nodes alongside a logical link.

We focus solely on RAM as the primary constraint for our analysis. However, it is important to note that other constraints, such as storage limitations or other resources, could also be added in the microservices constraints:

$$\sum_{s_k \in V_S} ram_k.\delta(s_k,n_i) \leq ram_i, \quad \forall n_i \in V_P \tag{3}$$

3.4 Minimum execution time of an MFP

The execution time of an MFP is a composite of two primary components: the processing time at each node and the time taken to traverse the network's edges when microservices on different network nodes must communicate with each other. If there is a direct or indirect dependency between two microservices, they cannot be run at the same time since one needs the other and conversely if there is no dependency then microservices can be executed in parallel.

To formalize this, we use the notion of depth in directed acyclic graph. In a DAG, the depth of a node measures its distance from the source nodes (i.e., nodes without predecessors) which are at depth 0. A node u is at depth k if the longest path from a source node to that node u has k edges. For example, the microservices in Figure 2 are organized according to their depth levels. In this way, each depth represents an additional step in the graph's sequence of dependencies. With this definition, two microservices at the same depth can be run in parallel (no dependency at all) and two microservices that are not at the

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63749-0_17

same depth can be interdependent and must not be run in parallel. In practice, two microservices can be independent even with a different depth if they are on different paths of the DAG. This notion of depth is extended to DAGs and MFP by considering it to be the maximum depth of their nodes.

From this, we define the minimum time execution of any MFP as its depth. Indeed, we consider that if the entire MFP is on the same server, i.e. without any network transmission, then all microservices on the longest path from a source to a sink will have to be executed one after the other. If microservices are not placed in the same network node, their execution will generate additional network communications and the overall execution time will increase.

It should be noted that, in practice, when a microservice s_k queries another microservice s_l, s_k will perform calculations, send a request to s_l, which will in turn perform calculations and possibly query other services, then s_l will send back a response to s_k, which will perform final calculations. The return time will just double the transmission times, but in a homogeneous way. We therefore disregard it and consider that there is no response.

4 Proposed solution for energy efficiency microservices placement

To solve the placement problem, we propose a heuristic placement approach based on community detection that aims to optimize the allocation of applications in communities of devices by considering available resources and their requirements. In a nutshell, our approach consists of two main phases: firstly, we identify a community of network nodes in which to place the application, and secondly, we select the network nodes within the selected community.

4.1 Community based selection

In a graph, communities are groups of nodes that are densely connected. That means, there is a lot of edges within a community, and few or less edges between communities. In our context, a community therefore represents a group of network nodes which are strongly connected: if communications between microservices are required, it would be preferable to place them in an area where communication possibilities are numerous. We make the assumption that the communication between microservices that are not placed in the same network nodes consumes the most energy and that the power electric cost is lower within communities compared to outside communities. Once the network communities are determined, we select all the MFP from the microservices dependency graph. We aim to allocate entire MFP within a single community to minimize inter-community exchanges.

The first phase therefore starts by calculating communities of devices using Louvain algorithm [14] which is fast and gives good results. We assign a score to each community, determined by its available resources and its size. We identify communities that are both resource-rich and have a large number of nodes. The

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63749-0_17

score gives priority to communities that are both resource-rich and large in size, in which we begin to place the largest MFP as follows:

$$\text{Community\_score} = \text{total\_cpu} \times \text{total\_ram} \times \text{num\_Node} \tag{4}$$

These communities are then ranked in ascending order according to their respective scores. Similarly, the applications are assigned scores based on their resource requirements and sorted in descending order as follows :

$$\text{MFP\_score} = \text{total\_cpu} \times \text{total\_ram} \times \text{num\_MS}. \tag{5}$$

The algorithm iterates through the MFP with the aim of placing the largest MFP in the smallest community that can fully accommodate it. If a community meets the requirements, the MFP is placed in that community, and the resource usages are updated accordingly. However, if no community has enough resources, the algorithm divides the MFP, by placing as many as possible of the partitions in the community with the highest score and the remaining partitions in another adjacent community identified by the shortest distance between them i.e the lowest edge weight between two pairs of nodes belonging to the two communities. This iterative process ensures efficient allocation of applications in device communities, considering both available resources and application requirements.

Algorithm 1 Community detection based placement algorithm for MFP

Input Microservices applications DAG, Network graph
Output placement communities for all the applications

$C \leftarrow$ calculate device communities and assign a score to each
$OC \leftarrow$ order communities C in ascending score
$MFP \leftarrow$ extract the MFP from the DAGs App and assign a score to each
$OMFP \leftarrow$ order applications by descending score
$MFPlacement \leftarrow \emptyset$
for MF in OMFP **do**
    for Comm in OC **do**
        if Comm has enough resources **then**
            MFPlacement(Comm,MF)
            updateResourceUsages(Comm,MF)
            RecalculateScore(Comm)
            ReorderCommunitiesByScores(C)
        end if
    end for
    Divide MFP, place partitions in highest-scored and the remainder in adjacent communities
end for

4.2 Node inside community selection

The second phase partitions MFP using the Kernighan-Lin algorithm [17] to obtain clusters of microservices. The algorithm is a graph partitioning method

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63749-0_17

used to divide a graph into two parts while minimizing the number of edges between the two parts. It is a bipartitioning algorithm that offers granular control and predictability over partitioning, enabling a minimal but efficient division, in alignment with our goal of grouping as many microservices as possible. We start by dividing the MFP into two partitions and then applying the algorithm to each partition until we have 1 node per partition to obtain a complete (binary) dendrogram. We then calculate a fitness function for each node within the community, considering both available resources and the node betweenness centrality (node_BetCent), a measure that quantifies the number of times a node acts as a bridge on the shortest path between two other nodes. We assign a factor of 0.7 to available resources and a factor of 0.3 to intermediate centrality as we consider that available resources are more critical than intermediate centrality. More in-depth study should be performed to evaluate the impact of these 0.7/0.3 factors. In the context of microservices placement networks, leveraging this measure helps in identifying strategic nodes, thereby optimizing energy consumption through more efficient communication paths and reduced transmission distances:

$$Node\_\text{fit} = 0.7 \times \left(\frac{\text{MFP\_resources\_requirement}}{\text{node\_available\_resources}} \right) + 0.3 \times (\text{node\_BetCent}) \quad (6)$$

We then select the node with the highest fitness and we place as much microservices as possible on this node, respecting the dendrogram (i.e. placing MS in order one by one). Once the node is full, the other MS are placed on the next node and so on. This method ensures that microservices are distributed in a manner that optimizes resource utilization and maintains the logical grouping determined by the KL algorithm.

Algorithm 2 Node inside community-based application placement

```
KLMFP ← generate the KL partitions
ON ← order devices in community by the fitness
for MFPpartition in KLSFC do
    for partition in MFPpartition do
        for each N in ON do
            if N has enough resources then
                for each service-id in partition do
                    if service-id not already placed then
                        SelectNode(N, service-id)
                        Update resource usages(N)
                        ReorderNodesByFitness(ON)
                    else
                        Continue to next service-id if already placed
                    end if
                end for
            else
                Continue to next node if insufficient resources
```


ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63749-0_17

5 Evaluation and results

Direct comparison of our heuristic with existing state-of-the-art approaches presents difficulties, mainly because few studies focus on microservices placement taking into account specific architectural dependencies and inter-microservices communications. Furthermore, the existing literature does not address the energy efficiency aspect of microservices placement.Therefore, to evaluate the effectiveness of our microservices placement heuristic, we carried out extensive validations to guarantee that (1) all the microservices of each MFP are deployed, (2) energy consumption is reduced, (3) resources consumption are respected, and finally (4) deployment time does not exceed twice the minimum execution time. To evaluate our heuristic approach, we developed three baseline scenarios:

- **No overload** where we use the same heuristic but limit network node resource usage at 70 %. This prevent overloading and ensure redundancy.
- **Without community** where we do not make use of communities for MFP placement. Nevertheless, the algorithm assigns scores to nodes based on resources and attempts to rank MFP from largest to smallest, similar to the proposed heuristic.
- **Random** implements random placement, this algorithm starts by selecting a node at random and places as many MFP as possible in that vertex. Then another node is chosen, again at random, and this process is repeated until all the MFP have been placed.

All our tests were carried out on an Intel(R) Core(TM) i7-9850H CPU @ 2.60GHz 2.59 GHz computer, with 32 GB of RAM running on the Windows 10 Professional Education system. Simulations were performed using Python 3.9.13. We used the Networkx library for the graph algorithms used in our heuristic.

5.1 Experimental setup

Network We used the topology of Oteglobe, obtained from The Internet Zoo topology library [18]. Oteglobe is a European operator known for providing telecommunications services to network operators. We divided this topology into three sets of nodes to create a single Cloud node, which represents the most central node determined by its betweenness centrality, this measure quantifies the centrality of a node as a key bridge in the shortest paths between two other nodes. Of the remaining nodes, 50% are designated as Fog nodes (nodes with the highest betweenness except the Cloud one). Finally, the remaining nodes are classified as Edge nodes. This partitioning strategy is aimed at modeling a diverse network infrastructure. For the simulation of resource characteristics (such as the number of cores, CPU speed and memory) of each Fog device, we employed a uniform random distribution similar to [13,19]. In addition, the bandwidth and latency configurations in the Fog network were defined in accordance with the methodologies used in previous studies [9, 13]. We have additionally assigned weights to the topology graph based on the maximum and minimum electrical power values. All the specific details are provided in Table 1.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63749-0_17

Parameter (unit)	Value or range
Network	Zoo Topology Dataset Oteglobe [18]
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)
$p_{idle}(J)$	145 (Cloud), 45 (Fog), 30 (Edge)
$p_{max}(J)$	320 (Cloud), 169 (Fog), 90 (Edge)
Bandwidth (KB/ms)	75
Link Propagation delay (ms)	1
Applications	Microservices dataset [16]
Microservices number	[5-25] depending on the application
CPU (MI)	[300-800]
Ram (MB)	[100-600]
Message size (KB)	[1500-4500]

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

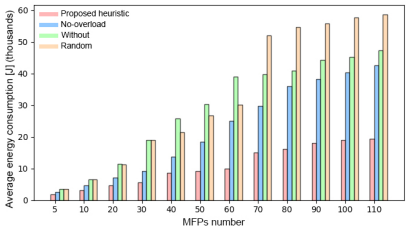
microservices applications We used the microservices dependency graph dataset taken from [16]. This dataset contains 20 distinct application architectures, each application consisting of a variable number of microservices ranging from 5 to 25. Each is represented by an acyclic directed graph where the nodes represent the microservices and the edges the function calls and dependencies between them. We used a uniform random distribution to assign values for various computing resources (such as cpu_i speed, memory size ram_i and also to specify the message size $data_{i,j}$ between microservices. To obtain a realistic and more challenging placement environment, we have duplicated the MFP obtained from these applications.

5.2 Results analysis

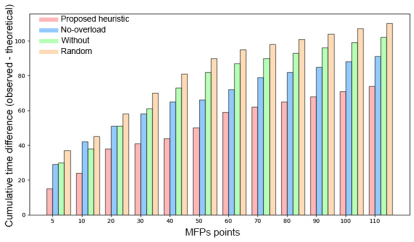
To analyse and evaluate the performance of our heuristic and compare it with the different scenarios mentioned above, we have used a number of metrics: (1) the energy consumption of the placement by measuring intra-nodes energy in nodes and inter-nodes communication energy; (2) the difference between the observed execution time and the minimal execution time; (3) the number of nodes used for the deployment of all MFP, considering that a node is used if at least one microservice is placed on it or if it is used for transmission; and (4) the number of links used as a measure of the dispersion of microservices.

Our heuristic has a clear advantage in terms of optimizing energy consumption over the other scenarios, as our evaluation shows in Figure 3a. Our approach reduces energy consumption in comparison with the strategy that limits node utilization to only 70% of capacity. This constraint restricts the efficient use of resources, since we are forced to place microservices in a higher number of nodes, generating more communications and therefore more energy between

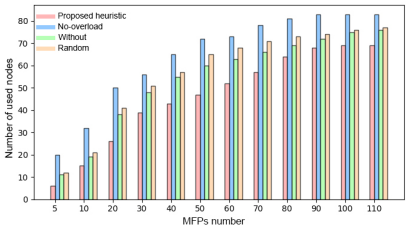
ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63749-0_17



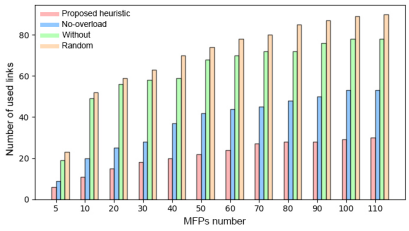
(a) Energy consumption.



(b) Excess time (observed - minimum).



(c) Total number of Nodes used.



(d) Total number of Links used.

Fig. 3: Comparison of the different microservice placement scenarios.

nodes. The comparison becomes even broader if we compare it to the version with a similar placement, but without the communities. This version, lacking community-based strategic allocation, does not place microservices that often communicate closely. Finally, the random strategy is not at all effective as expected. This disparity underlines the effectiveness of the heuristic we propose, not only in exploiting the full potential of the nodes, but also in ensuring a more coherent community-oriented deployment of services, which significantly improves the overall performance and energy consumption of the system.

Observed time of MFP execution We compared the observed execution time obtained through the placement with the minimum execution times that depends on the depth of the MFP. Figure 3b shows the cumulative time difference between the observed values and the minimum values (where each MFP is placed on a single node), This allows us to assess the extent to which the deployment times observed correspond to theoretical expectations. The proposed heuristic shows lower cumulative time differences than the other methods, indicating closer alignment with theoretical expectations. By contrast, the "No Overload" approach shows slightly higher time differences, followed by the "Without Communities" and "Random" approaches, suggesting that these approaches deviate more from expected deployment times.

Nodes and links number The analysis demonstrates that our method surpasses other approaches in efficiency by utilizing fewer links and nodes as the

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63749-0_17

quantity of MFP to be placed increases (see Figures 3c and 3d). This indicates a more effective optimization of communications and resource distribution. While the "No overload" strategy consumes more network nodes, its efficiency in link usage remains high due to strategic node selection within the same community, minimizing distance despite underutilized capacities because of imposed limitations. Conversely, the "Without community" and "Random" strategies, despite their relatively efficient employment of network nodes, exhibit significant inefficiencies in network communications. The "Random" strategy, in particular, is noted for its potential to select nodes that are exceedingly distant for placing microservices of the same MFP, leading to suboptimal communication paths.

6 Conclusion and future work

In this study, we developed a heuristic for the placement of microservices in IoT environments, based on community detection to optimise energy consumption in heterogeneous Cloud-Fog-Edge nodes with limited resources. Our proposed heuristic partitions the network into communities to identify nearby, strongly connected network nodes in order to prioritize the placement of MFP within these communities. We then used a best-fit approach to place the largest MFP in the smallest communities that could accommodate them. Our heuristic provides better results in terms of node and link utilization, and consequently a lower energy consumption than three other strategies.

In our future research, we aim to refine the placement of microservices in the Cloud-Fog-Edge continuum by including user nodes in the network topology, improving the management of microservice instances for energy efficiency, and implementing dynamic placement strategies that respond to real-time user needs and resource changes. In addition, we plan to integrate a wider range of environmental metrics, such as embodied energy and greenhouse gas emissions, to assess and optimise the environmental footprint of our deployment strategies. These efforts aim to optimise content delivery and minimise energy consumption, helping to create more sustainable and efficient IT environments.

References

1. S. Madakam, V. Lake, V. Lake, V. Lake, *et al.*, "Internet of things (iot): A literature review," *Journal of Computer and Communications*, vol. 3, no. 05, p. 164, 2015.
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.
3. J. Lewis and M. Fowler, "Microservices: a definition of this new architectural term," *MartinFowler. com*, vol. 25, no. 14-26, p. 12, 2014.
4. 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 International Conference on Utility and Cloud Computing*, pp. 71–81, 2019.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63749-0_17

5. 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)*, pp. 1–9, IEEE, 2019.
6. Y. Yu, J. Yang, C. Guo, H. Zheng, and J. He, "Joint optimization of service request routing and instance placement in the microservice system," *Journal of Network and Computer Applications*, vol. 147, p. 102441, 2019.
7. E. Ahvar, S. Ahvar, Z. 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. 70192–70213, 2021.
8. 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, 2018.
9. 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 *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*, pp. 32–40, IEEE, 2019.
10. 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)*, pp. 1–6, IEEE, 2022.
11. 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.
12. 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.
13. 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)*, pp. 9–18, IEEE, 2021.
14. 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.
15. 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)*, pp. 401–410, IEEE, 2017.
16. D. Rahman, MI.and Taibi, "A curated dataset of microservices-based systems," in *Joint Proceedings of the Summer School on Software Maintenance and Evolution*, CEUR-WS, September 2019.
17. B. Hendrickson, R. W. Leland, *et al.*, "A multi-level algorithm for partitioning graphs.," *SC*, vol. 95, no. 28, pp. 1–14, 1995.
18. 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.
19. 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.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63749-0_17