This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI 10.1016/j.comcom.2023.02.012, Elsevier, Computer Communication

Locality-aware Deployment of Application Microservices for Multi-Domain Fog Computing*

Francescomaria Faticanti[a,b], Marco Savi[c], Francesco De Pellegrini[d], Domenico Siracusa[a]

[a] *Fondazione Bruno Kessler, Via Sommarive, 18, Povo, Trento, 38123, Italy*
[b] *INRIA, 2004 route des Lucioles, BP 93, Sophia Antipolis, 06902, France*
[c] *University of Milano-Bicocca, Department of Informatics, Systems and Communication, Viale Sarca, 336, Milano, 20125, Italy*
[d] *University of Avignon, Laboratoire d'Informatique d'Avignon, Université d'Avignon, Avignon, 84140, France*

Abstract

In fog computing customers' microservices may demand access to connected objects, data sources and computing resources outside the domain of their fog provider In practice, the locality of connected objects renders mandatory a multi-domain approach in order to broaden the scope of resources available to a single-domain fog provider. We consider a scenario where assets from other domains can be leased across a federation of cloud-fog infrastructures. In this context, a fog provider aims to minimize the quantity of external resources to be rented to satisfy the applications' demands while meeting their requirements. We first introduce a general framework for the deployment of applications across multiple domains owned by multiple cloud-fog providers. Hence, the resource allocation problem is formulated in the form of an integer linear program. We provide a novel heuristic method that explores the resource assignment space in a breadth-first fashion to ensure that locality constraints are met. Extensive numerical results evaluate deployment costs and feasibility of the proposed solution demonstrating that it outperforms the standard approaches adopted in the literature.

Keywords: Fog Computing, Microservices, Federation, Resources Allocation, Virtual Network Embedding, Locality Constraints

1. Introduction

Fog computing is an innovative paradigm conceived to bridge the gap between cloud and IoT domains, ensuring the elaboration of data, when needed, at the network's edge, i.e., close to where it is produced [2]. This paradigm brings many benefits such as service latency reduction [3], privacy enhancement [4] and bandwidth savings [5]. However, it relies on the adoption of well-designed application architectures to leverage concepts and functionalities of cloud computing for the best users' experience.

In fact, in the last decade the design and development of cloud-native applications has evolved towards new architectural paradigms, including the adoption of *microservice-oriented architectures* [6, 7]. A microservice-oriented application is designed as a set of inter-related loosely-coupled modules/components, namely, the microservices. Here we use the terms "microservice" and "module" interchangeably. Each application microservice performs specific computations on input data and forwards obtained results to other modules for additional processing. One of the great advantages of such an approach is that each module can be containerized independently from each other. Finally, this architecture can implement the same intended functionalities of a traditional monolithic application, but attains a much higher degree of scalability, reliability, flexibility and even higher privacy guarantees [8]. As fog computing relies on computational resources that are geographically dispersed, the microservice-oriented architecture appears indeed the most natural choice for the design and development of fog-native applications. In this way, any fog application can be split into basic components to be executed each at the edge or in the cloud, depending on the requirements of the application and of its constituting components.

The evolution of fog computing has gained momentum in the last few years, when some fog-like solutions have been commercialized [9][10]. However, all existing fog computing platforms have been designed to work within a single administrative domain. Hence, they require exclusive ownership of resources spanning from cloud to things. This, unfortunately, is hardly suitable for the deployment of complex fog-native applications, mainly due to the wide geographical diffusion and heterogeneity of fog computing resources, including their ownership. In fact, fog-native applications come – by their own nature – with some hard-constraining *locality* requirements. It is worth noting that loose locality requirements do exist in cloud computing, both for delay/performance reasons and for national legislative aspects related to citizens' data ownership. However, while in cloud computing locality promotes the coverage of large regional sites, in fog computing locality is dictated by the specific geographical location where some processing task

*This work has received funding from the EU H2020 R&I Programme under Grant Agreement no. 815141 (DECENTER) and EU Horizon Europe R&I Programme under Grant Agreement no. 101070473 (FLUIDOS). The work of F. De Pellegrini has been partially supported by the French National Research Agency (ANR) within the framework of the PARFAIT project (ANR-21-CE25-0013). A preliminary version of this paper has been presented at COINS 2020 [1].

must be executed. This in turn imposes a fine-grained set of spatial constraints. As a matter of example, an application may need to keep a specific module in charge of processing sensitive data in a given location at the edge, e.g., a video stream monitoring people moving within a private area, administrated by a specific *fog provider* (i.e., owner of fog resources). This indeed enhances the privacy level of the application since data are not to be moved. In addition, the same application may require to integrate additional data which can only be captured by a specific peculiar resource at the edge (e.g., a camera covering a critical angle of vision), and whose access is offered by another specific fog provider.

This paper contributes to the research discussion on how to define a *multi-domain federated fog ecosystem*: fog providers can stipulate contracts among them to rent additional resources – which would not be accessible otherwise – and ensure smooth execution of their customers' applications. Previous works [11][12] have already advocated the technical feasibility of such approach by means of an interface to a resource and/or a platform. In this work we provide a different angle by considering the perspective of a fog provider and ask the question: *which resources should I rent to ensure that the requirements posed by a given set of applications with locality needs* – that I have to deploy – *are met in the federated fog while minimising external resource usage?*

In this context, fog providers are assumed to rely on a resource brokerage platform by which they can reserve the potential rentable resources owned by other providers. Such resources are identified using matchmaking mechanism [13] based on application modules' locality needs and, possibly, on other requirements of non-functional type. Given such a set of resources and self-owned resources, the objective is to determine an application deployment that minimises the resource rental cost.

Formally the *federated fog application deployment* problem is formulated by means of integer linear programming (ILP). The goal is to allocate the needed federated fog resources while satisfying any application's requirement and using self-owned resources as much as possible. The *deployment* of applications is defined as a *map* that associates applications' microservices and their exchanged network traffic flows to the available fog resources. The general problem of finding an optimal such map corresponds to an instance of the *virtual network embedding (VNE) problem*, which is known NP-hard [14][15].

To the best of the authors' knowledge, the literature on fog computing has not studied so far the problem of specializing VNE-like solutions to federated fog systems. In this context, the microservice structure of fog computing applications requires at the same time to comply with locality constraints on the placement of micro service modules and possibly lease resources not yet covered in house. To this aim, we propose a scalable heuristic algorithm to solve the application deployment problem by prioritizing modules with locality needs. It adopts a region-based, topological sorting of applications to be deployed and applies a Breadth-First Search approach. The rationale is that, since fog resources consumed by modules with locality constraints are precious, they should not be consumed in advance by modules not locality-constrained, a shortcoming which occurs with standard VNE heuristics based on iterative application placement [14].

We validate our solution with respect to the optimal deployment and two state-of-the-art approaches, one based on Depth-First Search and the other on Breadth-First Search (BFS) according to applications' topological sort [1]. This permits to consider the role of locality constraints – in terms of deployment cost and feasibility percentage – while retaining flexibility in the allocation of non locally-constrained microservices. Experimental results show that our proposal outperforms existing approaches especially under multiple locality constraints, leading to deployment costs and feasibility rates much closer to the optimum.

The rest of this paper is organized as follows. The next section describes the reference scenario and the system model. The reference resources allocation problem is reported in Sec. 4. Sec. 5 introduces the algorithmic solution, whereas its performance evaluation is detailed in Sec. 6. Related works are reported in Sec. 2 and a closing section ends the paper.

2. Related work

In this section we recall the related work with respect to service (or application) deployment in federated cloud and to virtual network embedding.

2.1. Service deployment in federated cloud

Service deployment mechanisms have been widely studied in the literature related to cloud federation [16][17]. Various dynamic and adaptive algorithms for resource allocation have been proposed. For instance, a distributed and adaptive approach for service placement on a heterogeneous cloud federation is presented in [18]. In [19] a multi-objective optimisation problem is formulated for resource allocation while addressing variations in applications' behaviour. Although these works describe crucial problems for an appropriate applications' deployment in a cloud federation environment, in that context locality constraints coming from inherent needs of IoT applications are not taken into account as applications' requirements. However, these constraints are the ones that mostly distinguish a federated fog from a federated cloud ecosystem, besides a higher resources heterogeneity. Conversely, our approach accounts explicitly for locality constraints when selecting resources for the deployment of fog applications. Furthermore, all the aforementioned works consider a monolithic structure for applications, whereas a microservice-oriented architecture represents a more promising paradigm for the design of future-proof fog-native applications [20].

This new architecture introduces new dimensions when resource allocation is performed for applications designed using the microservice paradigm [21]. Indeed, once all the microservices of a given application have been mapped to specific nodes of the infrastructure, a communication path between all these nodes must be ensured for the application to work properly. As said, most of the existing works considering resource allocation problems for the deployment of applications do not

consider such a structure for the applications, and do not consider placement and communication issues jointly. In our work we consider microservice-oriented applications that should be deployed on a distributed and heterogeneous (in terms of resources) fog computing infrastructure with locality constraints. This represents a key difference with respect to the federated cloud since the generated data comes from a specific geographical locations. Traditional use cases of this sort are: i) *video-analytics* applications typically consisting of a pipeline of computational modules that process video streams coming from mobile or fixed video-cameras [22]; ii) *Federated Learning* applications where some training modules should be executed locally to guarantee data-privacy requirements [23]; iii) *industrial applications* owned by companies that do not want to share their data presenting some application modules that should not be executed outside of the companies' domains [24]. The placement problem of computational modules of such applications mostly results to be NP-hard given the heterogeneity of both the computational requirements and the available resources.

In [25], one of the few works dealing with a placement strategy for microservice-oriented IoT applications is presented. The work considers a graph-based structure for applications that have to be deployed on a hierarchical fog infrastructure. The main differences with respect to our work are (i) the set of applications to be deployed and (ii) the type of infrastructure. For the former, a single application type is considered, whilst, in our case, a batch of applications is always deployed. Regarding the latter, we consider a federation of fog domains, while [25] considers a simple hierarchical infrastructure. To the best of the authors' knowledge, our work is the first considering the problem of application's deployment in a federated fog environment, where a cost for resource rental is applied for resources located outside of the owned domain. We believe that this scenario is the most plausible for future developments of fog environments, given the heterogeneity and exiguity of computational and networking resources ad the edge.

2.2. Virtual Network Embedding

Several works in the literature considered a microservice-oriented and modelled applications as Directed Acyclic Graphs (DAGs) [26][27]. With this kind of structure, the application deployment can be assimilated to a Virtual Network Embedding problem, which is NP-hard. Furthermore, it has been proven that VNE-like problems result hard to be approximated even with locality constraints [28]. Hence, several heuristic [29][30][31], and metaheuristic [32][33] methods have been proposed in the literature, especially with respect to the relevant problem of Virtual Network Function (VNF) placement [14] and for the deployment of requests of a single application (or VNF) at a time. Indeed, the most common procedure to solve these problems is to greedily deploy one VNF at a time. However, when dealing with locality constraints, the deployment should be performed considering the whole batch of applications at once. In this context, we proved that a Breadth-First Search (BFS) visit driven by the applications' region-based partitioning significantly reduces the deployment costs and ensures better feasibility percentages with respect to existing solutions.

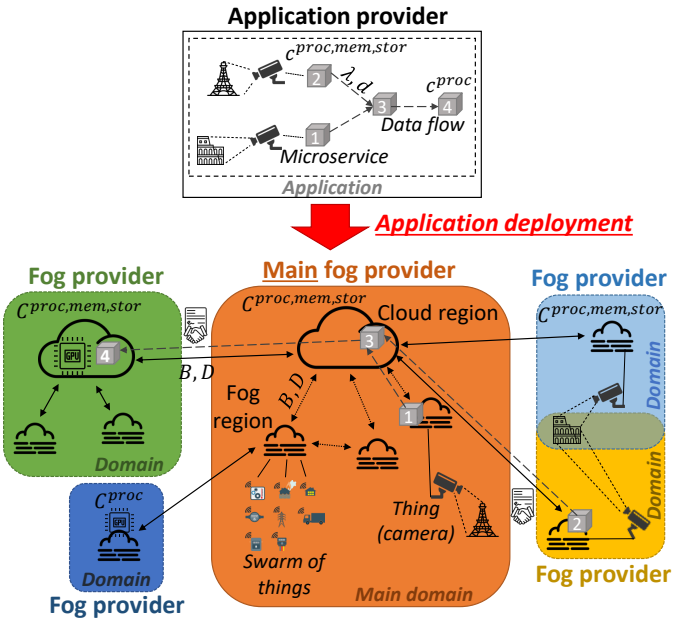

Figure 1: Multi-domain federated fog computing: an application deployment

When the locality constraint for each fog application is to have their first microservice deployed on a well-defined fog region, a BFS visit for resources allocation proves indeed more performing than standard VNE approaches [1]. In this work we consider the general case when the applications' deployment includes the possibility of multiple locality constraints per application in any of the possible fog regions. In such a general scenario, this placement algorithm outperforms standard VNE approaches and the previously-defined BFS approach.

3. System Model

3.1. Scenario and involved stakeholders

The high-level scenario considered in this work is depicted in Figure 1. We envision two main stakeholders:

- **Application provider:** it develops *microservice-oriented application*, used by its customer(s) to execute some specific tasks. A reference example to this respect may be an application monitoring the vehicular access to a restricted traffic area. The customer, e.g., a municipality, needs an application to perform the following chain of tasks: access a camera covering the said area (task 1), capture plates' number images for the vehicles entering the area (task 2), convert the plate number image into a plate number text (task 3), store the plate number into a database (task 4) and match it with authorized plate numbers (task 5). An application of this kind suits well the microservice-oriented architecture since each of the above tasks can run on a different microservice module. Furthermore, placing the video stream computation (task 2) close to the camera improves privacy and drastically reduces the application bandwidth consumption.
- **Fog provider:** it runs the *fog infrastructure* able to host microservice-oriented applications. The fog infrastructure may be geographically distributed when it spans both cloud and edge

infrastructures. Or, it may be limited to a specific geographical area when it comprises only an edge infrastructure. The fog infrastructure (i) provides computational, memory and storage capacity and (ii) permits to access things present on a target area and owned by the fog provider, if any (e.g., IoT sensors or video cameras). In the rest of the discussion we call *fog domain* or, shortly, *domain*, any infrastructure that is owned by a fog provider.

As introduced before, in the scenario we study we take the viewpoint of a fog provider – from now on called *main fog provider*. Her objective is to deploy different applications on its fog infrastructure, called *main domain*. However, depending on the application demands, the fog provider in some cases may not be able to meet the applications' requirements based on its own fog infrastructure alone. This can be due to *resource scarcity* within the main domain (e.g., not enough processing capacity is available to deploy *all* the applications) or – a very specific feature of fog computing paradigm – because of *locality constraints*, which may be of two types:

- *Geographical:* an application requires some storage or processing in a specific geographical location, e.g. for privacy reasons. This is typically related to the elaboration of data gathered by specific sensors. A customary example is video streaming capture and elaboration from a camera covering a specific area but owned by another fog provider.
- *Functional:* the main fog provider cannot secure a specific set of resources required by an application either because they are placed in a different domain or because the available resources are not sufficient for the application execution. For instance, an application may require a set of GPUs for fast processing, but the set of GPUs available to the main fog provider may satisfy such demand only partially.

In this work we assume that the main fog provider is part of a multi-domain federated fog computing ecosystem. She can hence rely on a federated infrastructure in order to fulfill the applications' requirements, including locality and otherwise exceeding own capacity (see Fig. 1).

3.2. Management and orchestration architecture

We now provide some hints on how a complex multi-domain federated fog computing ecosystem as the one shown in Fig. 1 could be managed. Our vision adheres to the vision described in [12], whose simplified management and orchestration architecture is shown in Fig. 2.

Any fog provider implements some local and distributed *Orchestration & Resource Exchange* logic to automatically (i) orchestrate services/resources for applications to be deployed and (ii) rent/lend some resources from/to other providers, so that they can be directly managed by the buyer. Resources available for rental are published on a distributed ledger (e.g. blockchain), called *Resource Exchange Broker* (REB).

In addition, fog providers implement some *Resource Selection* functionalities, which are in charge of (i) selecting *potential* resources to be acquired (e.g. those that roughly match geographical and/or functional locality constraints) and (ii) finally choosing what resources to acquire while minimizing acquisition costs. Sub-functionality (i) can be guaranteed by a *matching* algorithm operating on all the resources published on the REB, which is not subject of further study in this paper. Instead, our proposed *resource allocation* algorithms (see Sec. 5) are designed to implement sub-functionality (ii).

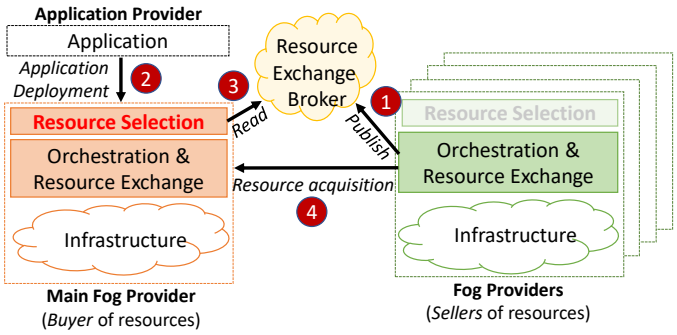

Figure 2: Management and orchestration architecture [12]

Figure 2 also reports the expected workflow: (1) fog providers publish resources on the REB, (2) an application deployment is requested by the application provider to the main fog provider, (3) the main fog provider reads the REB to record all the potential resources that can be acquired, (4) upon selection of adequate resources, the main fog provider stipulates the required smart contracts and acquires them. More details on the workflow and on how it can be technically guaranteed can be found in [12], while implications from a business perspective deserve further studies.

In the following, we introduce the model for (i) the multi-domain federated fog computing infrastructure and (ii) the fog applications.

3.3. Multi-domain federated fog infrastructure

An overall example of the reference fog infrastructure is reported in Figure 1. It is formed by multiple geographically spread fog or cloud *regions* which belong to different fog domains, including the main one. We identify with the term "domain" a set of fog or cloud regions belonging to the same owner. The core difference between cloud and fog regions lies in their processing, memory and storage capacities. Internally, each region encompasses several *hosts*. An host represents the atomic unit to deploy a microservice. Within each region, our assumption is that hosts are interconnected via high-performance links so that latency effects can be neglected and communication capacity is never a bottleneck. Conversely, different regions (either within the same domain or belonging to different domains of the federation) are interconnected by means of best-effort network connections.

Overall, the multi-domain infrastructure is modelled by a weighted graph $G_I = (V_I, E_I)$. In particular, V_I is the set of regions and the set of links $E_I \subseteq V_I \times V_I$. The nodes weights defined by weight function $w : V_I \rightarrow \mathbb{R}^+$ represent the cost to deploy a microservice in a specific region of the infrastructure. The set of domains is defined as $\mathcal{I} = \{I_1, \ldots, I_D\}$, where $I_d \subseteq V_I, \forall d \in [1, D]$, $\bigcup_{d=1}^{D} I_d = V_I$ and $I_d \cap I_{d'} = \emptyset, \forall d, d' \in [1, D]$ with $d \neq d'$. When we refer to nodes and edges/links composing the infrastructure we call them *physical nodes* and

physical edges/links. Each physical link $(u, v) \in E_I$ is characterized by the pair $(\Delta_{u,v}, B_{u,v})$, that is its latency and bandwidth, respectively.

3.4. Microservice-oriented fog applications

We define $\mathcal{A}$ as the set of applications to be deployed onto the infrastructure: it is the input to the virtual network embedding problem formulated in Sec. 4. Each application $A \in \mathcal{A}$ is represented by a weighted DAG $G_A = (V_A, E_A)$. Set V_A is the set of microservices (or components, or modules) composing the application, i.e., *virtual nodes*. Also, E_A represents the set of microservices dependencies, i.e., the *virtual edges/links*. Each directed edge (u_A, v_A) of an application A is characterised by (i) the maximum throughput on that link, $\lambda_A(u_A, v_A)$ and (ii) the maximum tolerated delay on the same link, $\delta_A(u_A, v_A)$. Each node v_A of an application A has some computational requirements in terms of processing, memory and storage, $c^r_{v_A}$, where $r \in \{proc, mem, stor\}$.

In addition, each application comes with a set of *locality constraints* that represent the regions where some microservices must be deployed, e.g. because data from specific devices/sensors belonging to that regions must be acquired or because some peculiar local resources need to be consumed. This is modelled by introducing a set $L_A \subseteq V_A$ for each application $A \in \mathcal{A}$. Such a set contains all the microservices that need to acquire data from a given device (geographical locality) or to use peculiar resources (functional locality) from a given region: hence, they must be placed in the specified region. Function $r_A : L_A \to V_I$ specifies for each node in L_A the region candidate for its deployment.

Table 1: Main notation.

Symbol	*Meaning*
$G_I = (V_I, E_I)$	Infrastructure network graph: V_I physical nodes (regions), E_I physical edges (network connections)
$\mathcal{A}$	Set of applications to be deployed on G_I
$G_A = (V_A, E_A)$	Graph for application $A \in \mathcal{A}$: V_A virtual nodes (modules) and E_A are virtual edges (data flows)
$L_A \subseteq V_A$	Subset of virtual nodes to be deployed on a specific physical node of the infrastructure
$r_A : L_A \to V_I$	Maps virtual node v_A to a given physical node (locality constraint)
$c^r_{v_A}$	Resource requirements of virtual node $v_A \in V_A$, with $r \in \{proc, mem, stor\}$
$\lambda_A(u_A, v_A)$	Max. throughput on edge $(u_A, v_A) \in E_A$
$\delta_A(u_A, v_A)$	Max. tolerated delay on edge $(u_A, v_A) \in E_A$
S_v	Set of available hosts in physical node $v \in V_I$
$C^r_{v,i}$	Resource capacity of the i-th host in physical node $v \in V_I$, with $r \in \{proc, mem, stor\}$
$\Delta_{u,v}$	Latency on physical link $(u, v) \in E_I$
$B_{u,v}$	Capacity of physical link $(u, v) \in E_I$
$w(v)$	Cost of physical node $v \in V_I$
P	Set of computed paths p between any couple of physical nodes
$P_{u,v} \subseteq P$	Set of computed paths between $v \in V_I$ and $u \in V_I$
s_p, t_p	First and last node of path $p \in P$

4. Problem Formulation

This sections reports and describes the ILP formulation of the resource allocation problem under locality constraints.

Decision variables. We employ two decision variables:
1) Binary variable $x^{A,v_A}_{v,i}$, which assumes value 1 if module $v_A \in V_A$ of application $A \in \mathcal{A}$ is assigned to host $i \in S_v$ of node $v \in V_I$ and is zero otherwise.
2) Binary variable $y^{(u_A,v_A)}_p$, which assumes value 1 if virtual link (u_A, v_A) is mapped to path p and is zero otherwise.

Objective Function. We assume that the main fog provider owns a tagged subset of the infrastructure nodes. Hence, we tackle the minimization of the total *deployment cost* in order to satisfy the application requests issued to the main provider. The cost of deploying a virtual node onto a physical node v of the fog infrastructure is represented by weight w. The cost of such deployment is assumed larger when it is performed onto a node owned by other fog providers, because their resources have to be rented. A weight function is thus defined per physical node, namely $w : V_I \to \mathbb{R}^+$. Finally, we can write the objective function as:

$$\sum_{A\in\mathcal{A}} \sum_{v_A\in V_A} \sum_{v\in V_I} \sum_{i\in S_v} w(v)\, x^{A,v_A}_{v,i}, \tag{1}$$

Recall that $x^{A,v_A}_{v,i}$ is one if microservice v_A of application A has been placed on server i of the region v, and it is zero otherwise.

Constraints. The first set of constraints are integrality constraints in the form:

$$\sum_{v\in V_I} \sum_{i\in S_v} x^{A,v_A}_{v,i} = 1, \quad \forall A \in \mathcal{A}, \forall v_A \in V_A. \tag{2}$$

They describe the fact that all application modules must be deployed exactly once.

Second, we account the for limited host capacity with the following constraints:

$$\sum_{A\in\mathcal{A}} \sum_{v_A\in V_A} c^{res}_{v_A} x^{A,v_A}_{v,i} \le C^r_{v,i} \tag{3}$$

where in (3) it holds $v \in V_I$, $i \in S_v$, and resource $r \in \{\text{proc, mem, stor}\}$.

Third, we consider the constraints for the capacity of virtual and physical links. A first set of constraints tie together allocation variables for nodes and virtual links:

$$\sum_{p\in P_{u,v}} y^{(u_A,v_A)}_p \le \sum_{i\in S_u} x^{A,u_A}_{u,i}, \tag{4}$$

$$\sum_{p\in P_{u,v}} y^{(u_A,v_A)}_p \le \sum_{j\in S_v} x^{A,v_A}_{v,j}, \tag{5}$$

$$\sum_{p\in P_{u,v}} y^{(u_A,v_A)}_p \ge \sum_{i\in S_u} x^{A,u_A}_{u,i} + \sum_{j\in S_v} x^{A,v_A}_{v,j} - 1, \tag{6}$$

where $(u, v) \in V_I^2$, $A \in \mathcal{A}$, and virtual link $(u_A, v_A) \in E_A$.

Constraints (4), (5) and (6) grant that there is a unique physical path implementing a virtual link, i.e., connecting two virtual nodes.

The following are constraints on the bandwidth capacity for all physical links $(u, v) \in E_I$:

$$\sum_{A \in \mathcal{A}} \sum_{(u_A, v_A) \in E_A} \sum_{p \in P : (u,v) \in p} \lambda_A(u_A, v_A)\, y_p^{(u_A, v_A)} \leq B_{u,v}, \tag{7}$$

and the applications' delay constraints, namely:

$$y_p^{(u_A, v_A)} \sum_{(u,v) \in p} \Delta_{u,v} \leq \delta_A(u_A, v_A), \tag{8}$$

where $A \in \mathcal{A}$, $(u_A, v_A) \in E_A$, and $p \in P$.

Last, but not least we impose locality constraints forcing a target subset of application modules on specific regions to comply either to geographical or functional constraints:

$$\sum_{i \in S_{r_A(v_A)}} x^{A, v_A}_{r_A(v_A), i} = 1, \quad \forall A \in \mathcal{A}, \forall v_A \in L_A. \tag{9}$$

5. Heuristic Algorithms

The Virtual Network Embedding (VNE) problem formulated in Section 4 is a well-known NP-hard problem. For this reason, it is important to look for efficient algorithms specialized for the considered scenario, whose application deployment solutions ensure a low deployment cost while guaranteeing acceptable computational time.

The general VNE problem has been deeply studied in the literature related to Software-Defined Networking and, given its NP-hardness, several heuristic solutions have been proposed [29]. Here we present the general idea adopted by most of the state-of-the-art heuristic solutions, which is based on a Depth-First-Search (*DFS*) exploration of the search space for applications deployment. Starting from this general approach we then propose a solution that is specifically tailored to the requirements of our problem, where locality constraints play a fundamental role, which is based instead on a Breadth-First Search (BFS) method. The BFS-based proposal described in this section, which we call *iBFS* (improved BFS), is an enhanced version of the BFS algorithm described in [1] (simply called *BFS*) and solves its main shortcomings. The reader should refer to [1] for details on how our previous BFS algorithm works.

5.1. Existing Approach: Depth-First Search

Several heuristic methods for VNE take a sorted lists of the deployment requests as input and return a map between each request and a subset of hosts and edges of the infrastructure. We consider an adaptation of the general approach presented in [34] as our main reference algorithm related to the DFS-based method.

For any application belonging to the input batch the following three steps are executed:

1. *Topological sorting of the application graph*: since the application graph is a DAG, there exists at least one topological order of its nodes, which can be obtained by performing a visit of the graph.
2. *Virtual node mapping*: for each application node a physical host is selected from a set of possible placements.
3. *Virtual link mapping*: for each application edge a physical path between the hosts where the vertices of the virtual edge have been mapped in the previous step is found.

Given a DAG, a topological sort of its nodes ensures that for each couple of vertices $u_A, v_A \in V_A$, if $(u_A, v_A) \in E_A$ then u_A precedes v_A in the topological sorting. In this way, the node mapping procedure is performed following the order defined by the topological sorting. Given an application graph, the node mapping establishes, for each module of the application, an admissible set of regions where to deploy the module. If the admissible set is not empty, a single region is selected for the deployment according to a given priority function. In this case, the procedure selects the region $v \in V_I$ that maximises the following formula:

$$res_{CPU}(v) \left\{ \sum_{u | (u,v) \in E_I} res_{BW}(u, v) + \sum_{u | (v,u) \in E_I} res_{BW}(v, u) \right\}, \tag{10}$$

where $res_{BW}(u, v)$ and $res_{CPU}(v)$ represent the residual bandwidth of the physical link (u, v) and the overall residual CPU capacity in region v, respectively [34]. If it happens that the set of admissible regions is empty for at least one module of the application, it means that such an application cannot be deployed. Conversely, if any module of the application is mapped to an infrastructure's region, then the procedure continues and the link mapping procedure is performed. According to this step, each application edge is mapped to the least congested path connecting the two regions where the vertices of the edge have been mapped in the previous step. The selected path must satisfy both bandwidth and latency requirements of the application's virtual edge. If all the virtual edges of the applications have been mapped to physical paths, then the map of the application onto the infrastructure is completed. This procedure is executed for all the applications in the batch.

5.2. Our Proposal: Improved Breadth-First Search Approach

The procedure described above deploys one application at a time. The main idea of our novel proposed approach is instead to deploy the batch of applications in a breadth-first fashion. Indeed, at each deployment iteration we consider all the applications: a subset of modules of each application is selected, as determined by a *region-based sorting* (and partitioning) of the application's graph, which considers the regions towards which locality constraints are set. Every time any subset of the application's modules is mapped to a set of hosts in the considered region, it is popped up from the stack of the region-based sorted modules. In this manner, applications' locality constraints can be better matched. In fact, a depth-first greedy procedure tends to quickly saturate all the resources of a regions with low deployment cost for the deployment of some applications, without considering that other applications, still waiting for their

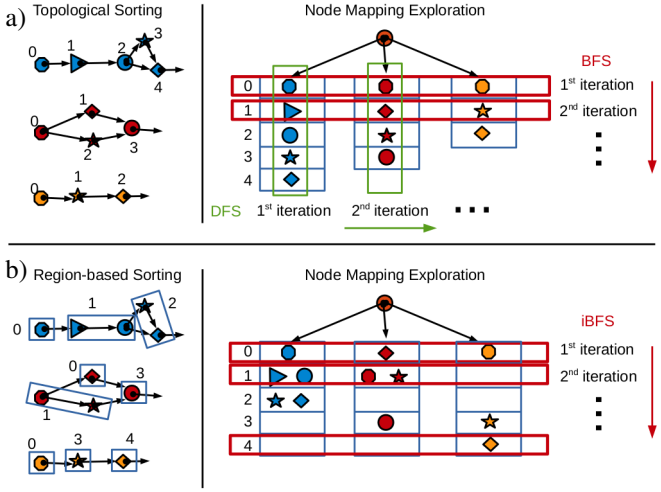

Figure 3: a) DFS vs. BFS [1] approaches after the topological sorting step; b) iBFS approach after region-based sorting.

deployment, may have a hard locality constraint on that region. This typically makes the deployment of the whole batch unfeasible, as we will better show in Sec. 6.

Unlike the solution proposed in [1], here we propose to initially partition each application on the basis of its locality constraints. As seen in the experimental section, this is key to reduce the number of infeasible application instances, i.e., those which cannot satisfy locality constraints. The *BFS* approach proposed in [1], in fact, explores each application following only its topological sorting. As for a depth-first search approach, this may lead to situations where the resources of a region are saturated preventing the deployment of other modules whose locality constraints reside on that region. Prioritizing those constraints in the partitioning phase ensures that they are satisfied if there are enough resources left on the target regions.

Our proposed algorithm consists of three main steps:

1. *Sorting of the batch of applications*: all the application are sorted based on their total bandwidth consumption.
2. *Partitioning of each application graph*: each application is partitioned based on the locality constraints defined on each microservice of that application. This step defines the *region-based sorting* of applications.
3. *Virtual node and link mapping*: by iterating over all the applications in a breadth-first manner, this step explores all of them jointly and level by level. At each iteration, all nodes' subsets on top of the applications' stacks (as specified by the region-based sorting of step 2) are popped up and a node mapping is performed, for each node in the subset, following the order established in step 2. Each module is mapped to a host of the infrastructure, and link mapping is performed together with node mapping.

Figure 3a) shows a high-level view on how basic DFS and BFS approaches work [1]. Both rely on a topological sorting of applications' graphs, and they differ on how the node mapping is performed through the different iterations. Figure 3b) shows instead how iBFS works. The main differences with BFS rely on both initial applications partitioning and on applications' modules deployment. In iBFS each application is partitioned with respect to locality constraints and all the sets of the partition are sorted according to the indexes of the regions towards which the locality constraints are defined. A last set of the partition includes all the modules of the application that have not locality constraints. For simplicity, Fig. 3b) shows a case where any module of any application is subject to a locality constraint. On the nodes mapping side, unlike the previous BFS version, iBFS can deploy more than one module, for each application, in each iteration. Indeed, the new algorithm deploys all the virtual nodes belonging to the set on top of the application's stack obtained by the initial partitioning procedure, which all have a locality constraint set towards the same region.

More in detail, the virtual node mapping (step 3) selects a set of admissible regions accounting for the requirements and the locality constraints of each application's module. If a locality constraint exists for a module, only one region is admissible. Once the set is defined, the algorithm greedily selects the regions leading to the lowest *deployment cost*. If many deployment options have the same cost, the algorithm selects the deployment option that presents the smaller relative increment in nodes' resource occupation (in terms of CPU, memory, storage and bandwidth). Finally, for link mapping, each time an application node v_A is mapped to a region R, the algorithm extrapolates the list of all the regions assigned to the already-deployed neighbours of v_A and selects the least-congested paths between them and R.

5.2.1. *Algorithm description and pseudocode*

The pseudocode of the algorithm is reported in Algorithm 1. `iBFS Deployment` first sorts all the applications by their total bandwidth requirement computed as the sum of the throughput requirements on each applications' virtual link (line 2).

Afterwards, for each application A in the defined order, the algorithm performs a *region-based sorting*. First, it partitions A according to the locality constraints defined on the modules. Each set of the partition is associated to a specific region $u \in V_I$. In this manner, the set associated to region u contains all the application's modules that have a locality constraint specified on region u. The cardinality of such a partition is $|V_I| + 1$, i.e., the total number of regions plus the set of all modules without locality constraints. Then, the partition of each application is sorted according to increasing regions' indexes, including the subset of modules with no constraints as the last element of the sorting. Each partition of application A is represented as a stack, $S_{\mathcal{A}}[A]$, where the first set of the sorting is on the top of the stack (line 6).

The algorithm then puts all the nodes with a constraint on the first region in the queue Q_A (lines 7-8). In this manner, all the regions are explored level-by-level according to the regions' indexes. The applications are visited in a breadth-first fashion (line 9), i.e., at each level u, and for each application A, all the nodes of A having a locality constraint in region u are deployed. Once all the modules of all the applications for a given level are deployed, the next level is considered. Then, until the queue is not empty, the algorithm extracts the next set from the queue

Algorithm 1: iBFS Deployment

Input: $G_I = (V_I, E_I)$: graph of the infrastructure; $\mathcal{A}$: set of applications to be deployed on G_I; L_A: set of locality constraints for each $A \in \mathcal{A}$

Output: Feasible deployment $P_{\mathcal{A}}$ for each $A \in \mathcal{A}$

```
1  BW_A ← sort_by_BW(A);
2  Q_A ← ∅;                                  // Empty queue
3  S_A ← ∅;                                  // List of stacks
4  P_A ← ∅;
   // Initialise Q_A
5  for A ∈ BW_A do
6      S_A[A] ← region_based_sorting(A, L_A);
7      C_A ← S_A[A].pop();
8      Q_A.enqueue(C_A);
9  while Q_A ≠ ∅ do
10     B_A ← Q_A.dequeue();
11     for u_A ∈ B_A do
12         P_A[A][u_A] ← node_mapping(G_I, u_A);
13         link_mapping(G_I, P_A, u_A);
14     C_A ← S_A[A].pop();
15     Q_A.enqueue(C_A);
16 return P_A
```

(line 10) and it performs the node and link mapping procedures for each node in the set (lines 12-13). Note that every time a set is extracted from the queue, that set is at the top of one of the stacks representing the region-based orders of the applications. Once a deployment for an application set is performed, the next set in the region-based sorting of the application is put in the queue (lines 14-15).

Link mapping is performed, once an application node u_A is assigned to a region u, by selecting the admissible less-congested path between the selected region u and all the regions assigned to the already-deployed neighbours of u_A.

5.2.2. Computational Complexity

The total computational complexity of Algorithm 1 is $O(\sum_{A\in\mathcal{A}} |V_A|\,(|V_I|^2 \log|V_I| + |E_I||V_I|))$ in the worst case, which is polynomial in the input size. The complexity is dominated by the `node_mapping` and `link_mapping` operations. Indeed, in the former, for each module, the algorithm can explore each region of the network in the worst case. For the latter, Dijkstra's algorithm can be applied to obtain the shortest path between the placement region of a module all its neighbour modules in the application DAG. The resulting complexity is $O(|V_I| \log|V_I| + |E_I||V_I|)$ for each iteration of the inner `for` loop (lines 12-13).

6. Performance Evaluation

This section describes the set of numerical experiments performed to validate the investigated algorithms on a multi-domain federated infrastructure with locality constraints.

With our experiments we want to (i) prove the negative impact of a Depth-First Search approach for the deployment of applications on either the *feasibility* of the solution or on its *optimality*; (ii) show the good trade-off between feasibility and optimality as offered by the improved Breadth-First Search approach proposed in this paper. The *feasibility* is measured as the percentage of the instances that admit a feasible solution, i.e., meet all the constraints of the optimisation problem described in Section 4, over all the generated instances.

The iBFS approach is tested in three different cases. In the first one we compare iBFS with the other strategies, including the BFS approach proposed in [1]; for the sake of comparison, we consider the scenario reported in the same paper, i.e., when only one locality constraint is specified for the first module of each application, which must be mandatorily placed in the main domain. The scenario is called here *Original Case*.

In the second case each module of each application is subject to a locality constraint with assigned probability p. We call this *Random Case*. In this case, we highlight the shortcomings of *BFS* that are solved by *iBFS*.

The last case specifies two locality constraints for each application. The first constraint requires that the first module of each application is deployed on a specific domain external to the main one. The second constraint requires that the last module of each application must be deployed on the main domain. We call this *Realistic Case*, as the combination of these two locality constraints appears to be the expected configuration for many application. For instance, a video analytics application requires access to an external camera and the final processing results are to be stored in the main domain.

6.1. Simulation settings

We describe how we have generated the test network topologies and the batch of applications to be deployed. We also report information on the experimental adopted tools.

6.1.1. Network topology

The fog infrastructure is modelled as a directed network graph with $|V_I| \in \{3, 6, 10, 15, 20\}$ regions and $D = 3$ domains. Concerning the main fog domain, we include a main cloud region that is connected to fog regions by means of a star topology where links between the cloud region and the fog regions are generated with probability $pr = 1$. For each random topology realisation, links among different fog regions are added according to an Erdős-Rényi random graph model, where the probability of having a link between two region is $pr = 0.5$. his models the common scenario where a generic fog region can always have access to the cloud meanwhile the direct connection to a different fog regions may not be guaranteed. The randomness on the generation of edges tries to simulate the uncertainty and the instability of connection typical of a fog scenario. This reflects a wide range of scenarios going from the urban one where the fog regions are easily reachable from any location to a more geographical distributed scenario where fog region are miles away far from each other. The resulting topology for the main fog domain is similar to that shown in Fig. 1. Eventually, each

Table 2: Applications' microservices requirements [15].

Requirement	*Mean Value*	*Range*
CPU ($c_{v_A}^{cpu}$)	1250 MIPS	[500, 2000] MIPS
Memory ($c_{v_A}^{mem}$)	1.2 Gbytes	[0.5, 2] Gbytes
Storage ($c_{v_A}^{stor}$)	3.5 Gbytes	[1, 8] Gbytes
Throughput (λ_A)	3 Mbps	[1, 5] Mbps
Delay (d_A)	262.5 ms	[25, 500] ms

link in the obtained topology is assigned to an average bandwidth $B = 60$ Mbps and an average delay $\Delta = 10$ ms [15].

We defined three main classes of available hosts, in each region, based on the resources they are equipped with. CPU resources are measured in terms of Million Instructions Per Second (MIPS). The classes are called *low* (CPU: 5000 MIPS, memory: 2 GB, storage: 60 GB), *medium* (CPU: 15000 MIPS, memory: 8 GB, storage: 80 GB) and *high* (CPU: 44000 MIPS, memory: 16 GB, storage: 120 GB). The main cloud region has instead infinite resources. In order to assign computational resources to each region, the aggregated demand of the batch of applications to be deployed (in terms of CPU, memory and storage) is equally split among all the available regions, excluding the cloud region of the main provider. Then, the set of hosts of each given region is generated by iteratively and randomly choosing hosts of different types up to the point where the aggregated demand fraction assigned to that region is satisfied. This procedure ensures that the infrastructure has enough CPU, memory and storage resources to deploy all the applications in the fog regions, and that any infeasibility is only caused by bandwidth scarcity, placement constraints and placement decisions.

6.1.2. Application batch

For each experiment a batch of applications $\mathcal{A}$ is generated with $|\mathcal{A}| = \{10, 15, 20, 25, 30\}$. The requirements of each applications' module in terms of CPU, storage, memory and throughput are uniform independent random variables with distribution values for each microservice as reported in Table 2. Each application is generated as a DAG by ordering all the nodes and adding an edge only between predecessors and successors in the defined order.

6.1.3. Tools

The most common fog simulators in the literature [35] do not offer support for scenarios with multiple domains and fog regions yet. Hence, for the evaluation of the proposed algorithms, we developed a Python-based simulator from scratch. For the resolution of the optimal placement ILP problem (*OPT*) we used the Gurobi solver [36]. Each data point in the reported graphs is the obtained average value over 30 randomized instances, where the network infrastructure does not change while the batch of applications and host distribution are randomly generated as described above. All the points are reported with their corresponding 95% confidence interval.

The optimisation problem is solved from the *main* fog provider's perspective. Her domain consists of *one cloud* and

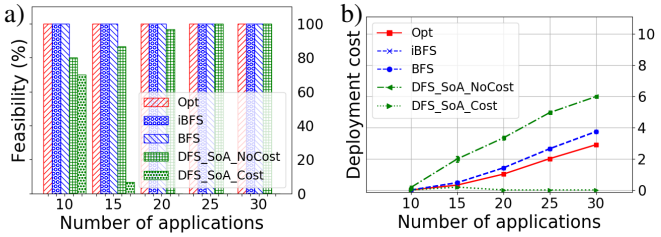

Figure 4: Feasibility-optimality tradeoff for the Original Case with $V_I = 3$. a) Feasibility percentage; b) Total deployment cost for each application.

one fog region while the other regions are distributed among the remaining *external* providers. The cost for all the deployment outside the main domain is $w = 1$ while the cost for the deployments inside the main domain is set to zero, i.e., $w = 0$.

6.2. Numerical results

6.2.1. Original Case

Figure 4 evaluates the optimality-feasibility tradeoff of the different solutions in a scenario with $V_I = 3$. In this case we imposed the constraint where the first module of each application must be deployed on main domain's fog region. Figures 4a) and 4b) report on feasibility and optimality of the proposed solution, namely *iBFS*, against two variants of the state-of-the-art DFS approach and the initial BFS approach presented in [1], namely *BFS*. *DFS_SoA_NoCost* does not take into account the *deployment cost optimization*, hence, its objective is just the maximisation of the number of deployed applications. Conversely, the objective of *DFS_SoA_Cost* is the minimisation of the deployment cost, as defined in (1). From Figure 4a) it can be noticed the high feasibility of *OPT*, *iBFS*, *BFS* and *DFS_SoA_NoCost* for all the sizes of the application's batch. Indeed, these approaches can deploy the complete batch of applications in all the randomised instances, reaching a feasibility of 100%. With the term *feasibility* we indicate the percentage of feasible instances among all the generated ones.

Furthermore, from Figure 4b) we can notice the closeness of *BFS* and *iBFS* to the optimal solution (*OPT*). On the other hand, although a good feasibility percentage, *DFS_SoA_NoCost* presents a high deployment cost. Conversely, *DFS_SoA_Cost* presents a low deployment cost, but its feasibility percentage drops dramatically with respect to *DFS_SoA_NoCost*. This is reasonable as the DFS approach is greedy by its inherent nature. Indeed, by combining DFS with cost minimization for resources allocation, the regions in the main domain with lowest cost are prioritized to host the applications' modules. In this way, lower cost resources are saturated soon, thus precluding the possibility to satisfy the locality constraints for the applications that have not been deployed yet. The same rationale explains the higher performance figures obtained with a BFS microservice placement: instead of performing the full placement of an application at each step – with all its microservices, both *BFS* and *iBFS* consider the deployment of a few modules of each application at each iteration. Thus feasibility percent-

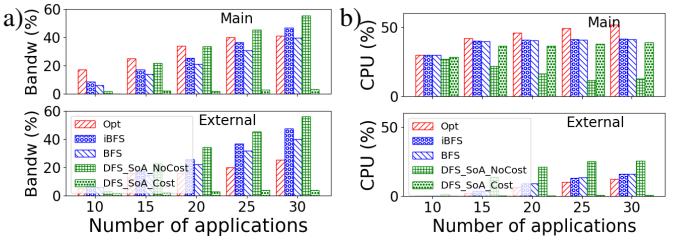

Figure 5: Bandwidth and CPU usage for the Original Case with $V_I = 3$. a) Percentage of bandwidth usage in the main and in external domains; b) CPU usage in the main and in external domains.

ages improve, since this helps to meet locality constraints, and yet it leads to a consistent reduction of the deployment cost.

To summarize, Figure 4 confirms that BFS-based solutions can guarantee a better trade-off between optimality and feasibility if compared to both DFS variants.

In Figure 5a) we evaluate the bandwidth consumed by the proposed solutions inside and outside the main domain. Given the greediness of *DFS_SoA_Cost*, which leads to a quick saturation of low cost resources (i.e., the ones in the main domain), its bandwidth consumption results almost constant and low in both the main and external domains. On the other hand, *OPT*, *BFS* and *iBFS* present a similar trend as the size of application batch increases. The *OPT* tends to consume slightly more bandwidth than *iBFS* and *BFS* on the link between the main cloud and the main fog region since the optimal solution deploys more applications in the main domain. The opposite happens with respect to external domains, confirming that BFS-based solutions tend to exploit the main domain slightly less than *OPT*. Finally, *DFS_SoA_NoCost* presents a similar trend on both main and external domains. This happens because eq. (10) is always maximized, irrespective of main or external domains. The presented results confirm the same behaviour of our new improved approach, *iBFS*, with respect to the original approach [1].

Figure 5b) reports on the CPU usage (in percentage) of all the solutions inside (upper figure) and outside (lower figure) the main domain as the batch size increases. As expected and confirmed by Figure 4b), the CPU consumption of *OPT* is slightly higher than the consumption of *BFS* and *iBFS* in the main domain given the higher number of applications deployed there. *DFS_SoA_Cost*, due to its greediness, presents higher CPU consumption in the main domain (low cost resources) and very low consumption in external domains. Conversely, *DFS_SoA_NoCost* shows an opposite trend with an increasing CPU usage in external domains as the applications' batch size increases. For the sake of conciseness, memory and storage consumption are not reported in this section since they present a similar behaviour as CPU usage.

Finally, in Table 3 we report on the average experienced execution time for all the considered solutions. For the computation of optimal solutions, the solver has been stopped after 5 minutes if no final solution has been found in this time range. Looking at the time of all the heuristic solutions, their scalability is glaring with respect to *OPT*. Given the high infeasibility

Table 3: Execution time (sec) with increasing the number of applications

$\lvert\mathcal{A}\rvert$	*OPT*	*iBFS*	*BFS*	*DFS_SoA_NoCost*	*DFS_SoA_Cost*
10	3.58	0.03	0.03	0.02	0.02
15	26.90	0.06	0.05	0.05	0.03
20	40.30	0.08	0.07	0.07	0.03
25	60.72	0.10	0.09	0.09	0.03
30	79.10	0.13	0.13	0.12	0.04

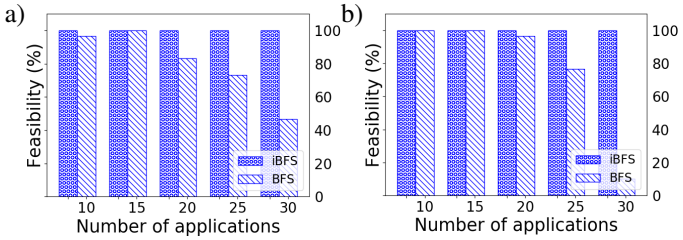

Figure 6: Comparison between iBFS and BFS in terms of feasibility with multiple locality constraints defined on applications' modules (Random Case). a) $|V_I| = 3$ and $p = 0.2$; b) $|V_I| = 3$ and $p = 0.8$.

rate of *DFS_SoA_Cost*, it presents a lower execution time as its runs are stopped as soon as the algorithm finds the first application that cannot be deployed, which is the condition that leads to the infeasibility of the whole deployment. In our tests, all heuristic methods possess linear time complexity for moderate batch sizes.

6.2.2. Random Case

In the Original Case, *iBFS* and *BFS* lead to the same feasibility and deployment cost. In order to highlight differences between these two approaches when multiple constraints are defined on applications' modules, we perform a comparison on an infrastructure with $|V_I| = 3$, while varying the probability p of having a locality constraint for each module of each application. If an application's module presents a locality constraint, with probability $q = 0.5$ the constraint will be the placement on the fog region of the main domain. Figure 6 shows the difference in terms of feasibility between the two approaches for two different values of p: a) $p = 0.2$, b) $p = 0.8$. In both cases we can see that for *BFS* the feasibility percentage similarly decreases as the size of applications' batch increases. This is due to the different initial sorting of application modules with respect to *iBFS*. Indeed, the *BFS* approach does not prioritize locality, i.e., it deploys microservices based on the initial topological sorting of applications. This can lead to an early saturation of a certain region preventing that other locality constraints of other applications and on the same region are met. Conversely the *iBFS* approach, with its initial region-based sorting and partitioning, ensures that all the locality constraints are met.

After showing that *iBFS* outperform *BFS*, from now on we only focus on *iBFS* and further evaluate the impact of locality constraints on the applications' deployment. We compare *iBFS* with *OPT* and *DFS_SoA* while varying the probability $p \in \{0.2, 0.5, 0.8\}$ and $q = 0.5$. Figure 7 reports on the average deployment cost, as the probability p increases, for two differ-

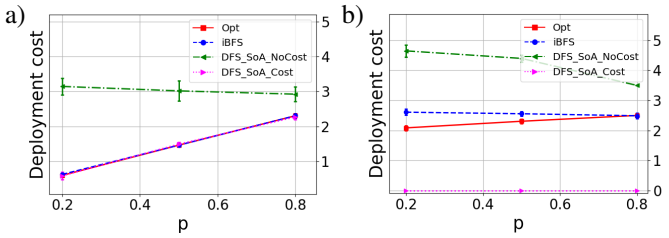

Figure 7: Deployment cost while varying the probability of having a locality constraint on each application's module (Random Case). a) $|V_I| = 3$ and $|\mathcal{A}| = 25$; b) $|V_I| = 6$ and $|\mathcal{A}| = 25$.

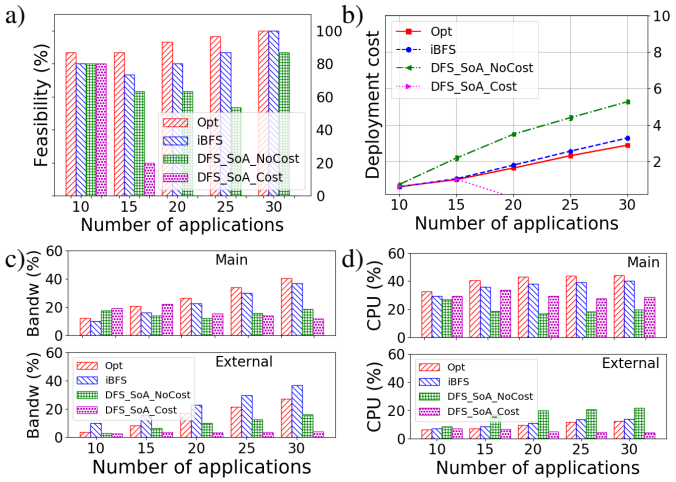

Figure 8: Random Case with $|V_I| = 6$ and $p = 0.5$. a) Feasibility percentage; b) Average deployment cost for each application; c) Percentage of bandwidth usage within the main domain and in external domains; d) CPU usage in the main domain and in external domains.

ent infrastructure sizes: $|V_I| = 3$ for Figure 7a), and $|V_I| = 6$ for Figure 7b). For $|V_I| = 3$ there is a small difference between the optimum, the *DFS_SoA_Cost* and the *iBFS* approaches. On the other hand, for $|V_I| = 6$ the difference between the approaches becomes more relevant. In both figures *iBFS* is always close to the optimal solution, especially when p is larger. Moreover, as the size of the infrastructure increases, a greedy depth-first approach leads to higher costs, in case of *DFS_SoA_NoCost*, and to higher infeasibility percentages, in case of *DFS_SoA_Cost*. This confirms the *iBFS* effectiveness in exploring the best trade-off between optimality and feasibility in the presence of locality constraints.

Further, Figure 8 shows the comparison of the proposed solutions with $|V_I| = 6$ and $p = 0.5$. From Figure 8a) we can notice the low feasibility of the *DFS_SoA_Cost* approach. Such a low feasibility is due to the combination of the greediness of the algorithm and the relevant number of locality constraints for each application. Figure 8b) confirms the closeness of *iBFS* to the optimal solution in terms of deployment cost. The apparently good performance of *DFS_SoA_Cost* is due to its high infeasibility. Reasonably, the *DFS_SoA_NoCost* presents a good level of feasibility and higher deployment cost for its cost-agnostic

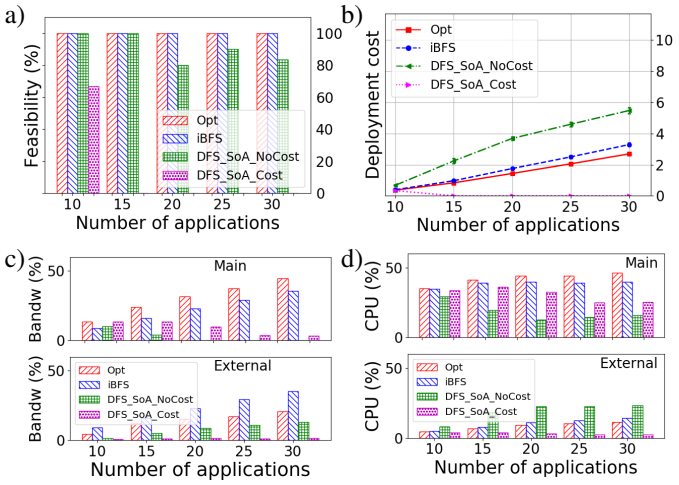

Figure 9: Realistic Case with $|V_I| = 6$. a) Feasibility percentage; b) Average deployment cost for each application; c) Percentage of bandwidth usage within the main and in external domains; d) CPU usage in the main and in external domains.

nature. Figure 8c) and Figure 8d) show the networking and CPU resource consumption, respectively. Given their lower deployment cost, the resource consumption for both the optimal solution and *iBFS* is greater in the main domain than in external domains. Due to its greedy nature, also *DFS_SoA_Cost* consumes more internal resources than external; conversely, for *DFS_SoA_NoCost* we have not identified such a regular placement pattern.

6.2.3. *Realistic Case*

To simulate a more realistic case, we evaluated our approach imposing locality constraints only on the first and the last module of each application. For the first module, the locality constraint is set on a randomly-selected fog region external to the main domain. For the last module, the locality constraint is imposed on the fog region of the main domain. Indeed, this simulates the situation where an application needs for an external device, like a sensor or a specific camera, located in an external domain, and the final results of the computing process need to be stored locally in the main domain (e.g. to guarantee privacy).

Figure 9 confirms, even in this case, the similar behaviour of *iBFS* and the optimal solution in terms of feasibility, average deployment cost, bandwidth and CPU utilisation. Such a Realistic Case highlights the greediness of a typical approach such as *DFS_SoA_Cost*, which does not treat locality constraints smartly. In particular, given the low bandwidth consumption, it is apparent that *DFS_SoA_Cost* tends to saturate all the resources of the fog region of the main domain by placing there modules not subject to locality constraints. Thus, it attains low feasibility figures because locality constraints set on the last module of applications becomes soon bottleneck.

Finally, Figure 10 reports on the feasibility and the cost of compared algorithms as the number of regions, i.e., $|V_I|$ increases. In Figure 10 we do not report the optimal solution

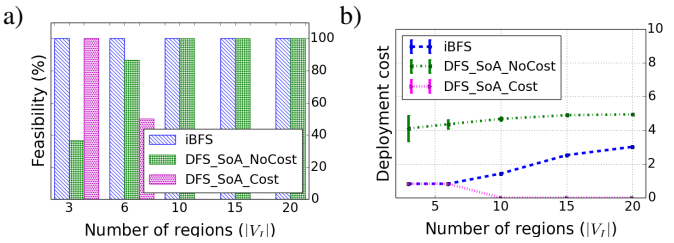

Figure 10: Realistic case with increasing number of regions. a) Feasibility percentage; b) Average deployment cost for each application.

Table 4: Execution time (sec) with increasing number of regions with $|\mathcal{A}| = 25$.

$\|V_I\|$	*iBFS*	*DFS_SoA_NoCost*	*DFS_SoA_Cost*
3	0.14	0.09	0.09
6	0.16	0.08	0.07
10	0.22	0.10	0.10
15	0.31	0.15	0.16
20	0.50	0.31	0.33

given the high execution time presented by the solver to compute it. The comparison is performed in the real scenario where all applications present one locality constraint on the first and on the last module of their topological orders. iBFS achieves 100% of feasibility for all the sizes of the physical networks and presents the lowest deployment cost as the number of regions increases. Even when more resources are available with a higher number of regions *DFS_SoA_Cost* has a low percentage of feasibility highlight the need to perform a breadth-first search in the state space for the deployment of applications presenting these particular locality constraints. The scalability of the compared approaches is confirmed by Table 4.

7. Conclusions

This paper considered the problem of fog applications deployment in a federated cloud-fog environment under locality constraints. Solving the problem of initial resource selection plays a fundamental role, and solutions should be tailored to the reduction of deployment costs while satisfying all the applications' requirement – including those related to locality – and accommodating as many concurrent requests as possible. By considering a microservice paradigm for fog-native applications, a Virtual Network Embedding problem has to be solved, which is known to be NP-hard. Standard VNE heuristic solutions need to explore the best trade-off between feasibility, cost-efficiency and scalability. But, typically, they do not account for the specific locality requirements of fog computing applications. This work presented a novel approach for the initial deployment of a batch of fog-native applications based on a breadth-first visit of all the applications' constituting graphs. The proposed algorithm was designed to prioritize the deployment of applications' microservices with specific locality needs, to ensure that they are guaranteed and precious resources are not saturated by other unconstrained microservices. It was shown to provide a near-optimal performance and yet excellent feasibility rate, outperforming both standard depth-first greedy heuristics and a non-locality-aware breadth-first strategy. Future works on the topic will be devoted to the design of service-exchanging mechanisms between different domains, paving the way to new deployment and orchestration strategies of applications in fog computing.

References

[1] F. Faticanti, M. Savi, F. De Pellegrini, P. Kochovski, V. Stankovski, D. Siracusa, Deployment of Application Microservices in Multi-Domain Federated Fog Environments, in: International Conference on Omni-layer Intelligent Systems (COINS), 2020.

[2] A. V. Dastjerdi, R. Buyya, Fog Computing: Helping the Internet of Things Realize Its Potential, IEEE Computer 49 (8) (2016) 112–116.

[3] A. Yousefpour, G. Ishigaki, J. P. Jue, Fog Computing: Towards Minimizing Delay in the Internet of Things, in: IEEE International Conference on Edge Computing (EDGE), 2017.

[4] Y. Guan, J. Shao, G. Wei, M. Xie, Data Security and Privacy in Fog Computing, IEEE Network 32 (5) (2018) 106–111.

[5] C. C. Byers, Architectural Imperatives for Fog Computing: Use cases, Requirements, and Architectural Techniques for Fog-Enabled IoT Networks, IEEE Communications Mag. 55 (8) (2017) 14–20.

[6] I. Nadareishvili, R. Mitra, M. McLarty, et al., Microservice architecture: aligning principles, practices, and culture, in: O'Reilly Media Inc., 2016.

[7] S. Pallewatta, V. Kostakos, R. Buyya, Microservices-based iot applications scheduling in edge and fog computing: A taxonomy and future directions, arXiv preprint arXiv:2207.05399 (2022).

[8] Y. Gan, C. Delimitrou, The Architectural Implications of Cloud Microservices, IEEE Computer Architecture Letters 17 (2) (2018) 155–158.

[9] AWS IoT Greengrass, `https://aws.amazon.com/greengrass/`.

[10] Azure IoT Edge, `https://shorturl.at/egpEJ`.

[11] E. Carlini, M. Coppola, P. Dazzi, et al., BASMATI: Cloud Brokerage Across Borders for Mobile Users and Applications, in: Springer Advances in Service-Oriented and Cloud Computing Workshop, 2018.

[12] M. Savi, D. Santoro, K. Di Meo, et al., A Blockchain-based Brokerage Platform for Fog Computing Resource Federation, in: Conference on Innovation in Clouds, Internet and Networks (ICIN), 2020.

[13] X. Li, H. Ma, F. Zhou, X. Gui, Service Operator-Aware Trust Scheme for Resource Matchmaking across Multiple Clouds, IEEE Transactions on Parallel and Distributed Systems 26 (5) (2015) 1419–1429.

[14] X. Cheng, S. Su, Z. Zhang, et al., Virtual Network Embedding through Topology-aware Node Ranking, ACM SIGCOMM Computer Communication Review 41 (2) (2011) 38–47.

[15] A. Brogi, S. Forti, A. Ibrahim, How to Best Deploy Your Fog Applications, Probably, in: IEEE International Conference on Fog and Edge Computing (ICFEC), 2017.

[16] B. Rochwerger, D. Breitgand, E. Levy, et al., The Reservoir Model and Architecture for Open Federated Cloud Computing, IBM Journal of Research and Development 53 (4) (2009) 1–4.

[17] A. J. Ferrer, F. Hernandez, J. Tordsson, et al., OPTIMIS: A Holistic Approach to Cloud Service Provisioning, Elsevier Future Generation Computer Systems 28 (1) (2012) 66–77.

[18] E. Carlini, M. Coppola, P. Dazzi, M. Mordacchini, et al., Self-optimising Decentralised Service Placement in Heterogeneous Cloud Federation, in: IEEE International Conference on Self-adaptive and Self-organizing Systems (SASO), 2016.

[19] R. G. Aryal, J. Altmann, Dynamic Application Deployment in Federations of Clouds and Edge Resources using a Multiobjective Optimization AI Algorithm, in: IEEE International Conference on Fog and Mobile Edge Computing (FMEC), 2018.

[20] A. Samanta, Y. Li, F. Esposito, Battle of microservices: Towards latency-optimal heuristic scheduling for edge computing, in: 2019 IEEE Conference on Network Softwarization (NetSoft), IEEE, 2019, pp. 223–227.

[21] Q. Li, B. Li, P. Mercati, R. Illikkal, C. Tai, M. Kishinevsky, C. Kozyrakis, Rambo: resource allocation for microservices using bayesian optimization, IEEE Computer Architecture Letters 20 (1) (2021) 46–49.

[22] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, M. Philipose, Videoedge: Processing camera streams using hi-

erarchical clusters, in: 2018 IEEE/ACM Symposium on Edge Computing (SEC), IEEE, 2018, pp. 115–131.
[23] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al., Advances and open problems in federated learning, Foundations and Trends® in Machine Learning 14 (1–2) (2021) 1–210.
[24] N. Bugshan, I. Khalil, N. Moustafa, M. S. Rahman, Privacy-preserving microservices in industrial internet of things driven smart applications, IEEE Internet of Things Journal (2021).
[25] S. Pallewatta, V. Kostakos, 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, 2019, pp. 71–81.
[26] R. Yu, V. T. Kilari, G. Xue, D. Yang, Load balancing for interdependent iot microservices, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 298–306.
[27] N. K. Giang, M. Blackstock, R. Lea, V. C. Leung, Developing iot applications in the fog: A distributed dataflow approach, in: 2015 5th International Conference on the Internet of Things (IOT), IEEE, 2015, pp. 155–162.
[28] E. Amaldi, S. Coniglio, A. M. Koster, M. Tieves, On the computational complexity of the virtual network embedding problem, Electronic Notes in Discrete Mathematics 52 (2016) 213–220.
[29] H. Cao, H. Hu, Z. Qu, et al., Heuristic Solutions of Virtual Network Embedding: A Survey, China Communications 15 (3) (2018) 186–219.
[30] J. Lischka, H. Karl, A virtual network mapping algorithm based on subgraph isomorphism detection, in: Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures, 2009, pp. 81–88.
[31] N. M. K. Chowdhury, M. R. Rahman, R. Boutaba, Virtual network embedding with coordinated node and link mapping, in: IEEE INFOCOM 2009, IEEE, 2009, pp. 783–791.
[32] I. Fajjari, N. Aitsaadi, G. Pujolle, H. Zimmermann, Vne-ac: Virtual network embedding algorithm based on ant colony metaheuristic, in: 2011 IEEE international conference on communications (ICC), IEEE, 2011, pp. 1–6.
[33] Z. Zhang, X. Cheng, S. Su, Y. Wang, K. Shuang, Y. Luo, A unified enhanced particle swarm optimization-based virtual network embedding algorithm, International Journal of Communication Systems 26 (8) (2013) 1054–1073.
[34] M. Yu, Y. Yi, J. Rexford, et al., Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration, ACM SIGCOMM Computer Communication Review 38 (2) (2008) 17–29.
[35] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, et al., iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in the Internet of Things, Edge and Fog Computing Environments, Wiley Software: Practice and Experience 47 (9) (2017) 1275–1296.
[36] Gurobi Optimization, LLC, Gurobi optimizer reference manual (2021). URL `http://www.gurobi.com`