Francescomaria Faticanti; Francesco De Pellegrini; Domenico Siracusa; Daniele Santoro; Silvio Cretti, Throughput-Aware Partitioning and Placement of Applications in Fog Computing, IEEE Transactions on Network and Service Management, Volume: 17,Issue: 4, December 2020, pp. 2463-2450, DOI: 10.1109/TNSM.2020.3023011

The final published version is available online at: https://ieeexplore.ieee.org/document/9189841

Throughput-Aware Partitioning and Placement of Applications in Fog Computing

Francescomaria Faticanti, *Graduate Student Member, IEEE,* Francesco De Pellegrini, *Member, IEEE,* Domenico Siracusa, Daniele Santoro, and Silvio Cretti

***Abstract*—Fog computing promises to extend cloud computing to match emerging demands for low latency, location-awareness and dynamic computation. It thus brings data processing close to the edge of the network by leveraging on devices with different computational characteristics. However, the heterogeneity, the geographical distribution, and the data-intensive profiles of IoT deployments render the placement of fog applications a fundamental problem to guarantee target performance figures. This is a core challenge for fog computing providers to offer fog infrastructure as a service, while satisfying the requirements of this new class of microservices-based applications. In this article we root our analysis on the throughput requirements of the applications while exploiting offloading towards different regions. The resulting resource allocation problem is developed for a fog-native application architecture based on containerised microservice modules. An algorithmic solution is designed to optimise the placement of applications modules either in cloud or in fog. Finally, the overall solution consists of two cascaded algorithms. The first one performs a throughput-oriented partitioning of fog application modules. The second one rules the orchestration of applications over a region-based infrastructure. Extensive numerical experiments validate the performance of the overall scheme and confirm that it outperforms state-of-the-art solutions adapted to our context.**

***Index Terms*—Fog computing, IoT, applications partitioning, resource allocation, microservices.**

I. Introduction

THE SPREAD of IoT devices has led to the design of new extensions of cloud computing, since at scale the huge amount of data produced by those devices is becoming unmanageable with existing device-to-cloud paradigms. Sending all raw data generated by IoT devices to the remote cloud for processing, in fact, represents a key bottleneck due to the high latency and network congestion introduced. Fog computing has been proposed in order to mitigate these core problems [1]. The idea is to perform part of the overall computation at the edge of the network without sending raw data directly to the cloud. In this manner, the amount of data sent to the cloud is reduced, resulting in lower bandwidth consumption and reduced response time.

The current fog computing paradigm is based on a layered architecture, including a central cloud, a series of edge nodes and, finally, target objects which generate data and/or actuate. In contrast to cloud computing, where a set of homogeneous resources are concentrated in the same area, fog computing infrastructure typically consists of a set of heterogeneous and geographically distributed resources, possibly organized in fog regions with respect to the objects each region hosts. In this context, the usage of edge resources need to be optimised with respect to the specific architecture of fog applications to satisfy their required performance figures.

With the fast growth of IoT, microservices-based development has become the current practice in cloud and fog application design in order to guarantee availability and scalability [2], [3]. Microservices applications are composed by interdependent modules, such as for example, a graphical user interface, a Web repository or an image recognition module.

Satisfying the requirements of all the modules of an application translates in the selection and allocation of the right set of resources in the fog-infrastructure. But, while resource allocation in cloud computing is a well-known and complex problem – provably NP-hard in all cases of practical relevance [4] – in fog computing the problem has a specific structure due to geo-distribution of heterogeneous resources and location of IoT devices. Actually, in a typical fog scenario there exist different regions where the modules of a certain fog application can be deployed, as depicted in Figure 1. Thus, whilst several approaches have been proposed in the cloud literature, few of them account for the deployment among different regions. In fact, existing technologies fit the cloud scenario where the resources' pool is concentrated in the same area irrespective of the objects' location. For instance, current Kubernetes' placement algorithms [5] perform well in cloud, but they require new functional extension to discriminate the deployment of application components across multiple regions. Consequently, there is a lack of solution for application deployment that exploits the inter-operability between different regions. Actually, in our tests we verify that, while for the sake of implementation it is tempting to treat all the fog regions as a unique region ruled by off-the-shelf Kubernetes

Manuscript received March 10, 2020; revised July 2, 2020; accepted September 1, 2020. Date of publication September 9, 2020; date of current version December 9, 2020. This work has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement no. 815141 (DECENTER: Decentralised technologies for orchestrated Cloud-to-Edge intelligence). The associate editor coordinating the review of this article and approving it for publication was D. Pezaros. *(Corresponding author: Francescomaria Faticanti.)*

Francescomaria Faticanti is with the ICT, RiSING Group, Fondazione Bruno Kessler, 38123 Trento, Italy, and also with the Department of Information Engineering and Computer Science (DISI), University of Trento, 38122 Trento, Italy (e-mail: ffaticanti@fbk.eu).

Francesco De Pellegrini is with the Laboratoire Informatique d'Avignon, University of Avignon, 84140 Avignon, France.

Domenico Siracusa, Daniele Santoro, and Silvio Cretti are with the ICT, RiSING Group, Fondazione Bruno Kessler, 38123 Trento, Italy.

Digital Object Identifier 10.1109/TNSM.2020.3023011

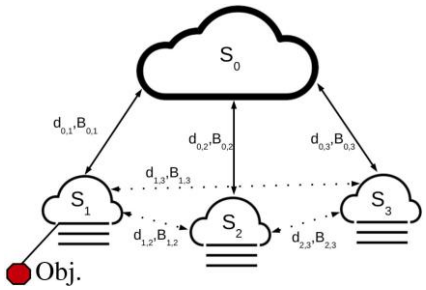

Fig. 1. Reference fog system architecture: application $u1$ requests data from an IoT object (marked red) located in area $S1$; the whole system is composed of geographical areas (regions) S_i, $i = 1, 2, 3$ connected to the central cloud S_0.

placement algorithms, this may severely limit the number of applications deployed on the infrastructure.

Main contribution: In this work we address the problem of deployment of batch of applications in a fog infrastructure covering different geo-distributed and heterogeneous regions, subject to the applications' requirements with respect to both computation and communication. We represent IoT applications' workflows by means of direct acyclic graphs (DAG), where microservices are vertices and the dependencies between the microservices represent graph edges.

The corresponding resources optimisation becomes a non linear mixed integer NP-hard problem. The problem combines a multicommodity flow and a graph embedding problem, for which we provide a negative result for the possibility to design tight polynomial-time approximation algorithms. Thus, we propose a cascade solution consisting of two main steps. First, at the application level, a preliminary partitioning for applications minimises the throughput footprint of fog-native applications. Second, a placement step accounts for the computational and communication demands and proximity requirements of applications.

The rest of this article is organised as follows. The next section describes the system model, i.e., the abstractions we adopt for the applications' architecture, the network infrastructure and applications' deployment configurations. In Section III we present the problem formulation, introducing the most general problem setting. The placement problem is addressed in Section IV. The problem is proved NP-hard by reduction from a multi-dimensional multiple-choice knapsack problem and a greedy algorithm is developed in Section V. Numerical results are reported in Section VI. Related works and existing container orchestration technology for fog computing are reported in Section VII. A concluding section ends this article.

II. System Model

The combination of a fog-native partitioning and a placement scheme follows the rationale that microservices of the same application, when deployed on the same fog region, generate negligible communication overhead; in fact, under standard containerisation technologies they can be deployed on the same pod [6], [7]. On the other hand, when two application modules are deployed in different region, the resources allocation balance must account for the communication overhead. A

TABLE I
Main Notation Used Throughout This Article

Symbol	*Meaning*
$\mathscr{K}$	set of regions $\lvert\mathscr{K}\rvert = K$
$\mathscr{U}$	set of applications to be deployed $\mathscr{U} = \cup_{i=1}^{K} U_i$, $\lvert\mathscr{U}\rvert = U$
S_k	set of server units in region k, with $\lvert S_i\rvert = n_i, \forall i \in \mathscr{K}$, $S_i = \{s_{i_1}, \ldots, s_{i_{n_i}}\}$
S_0	central cloud
U_k	set of applications requiring IoT data in region k
F_u	output samples per second required by application u
$\mathbf{C_{k_i}}$	memory, storage and processing capacity of the i-th server in region k: $\mathbf{C_{k_i}} = (C^M_{k_i}, C^S_{k_i}, C^P_{k_i})$
$x_{u,k,i} \in \{0,1\}$	boolean variable indicating the fog chunk of application u is placed on server unit i of region k
$x_{u,k}$	$x_{u,k} = \sum_{i \in S_k} x_{u,k,i}$
$x_{u,u}$	boolean variable indicating the fog chunk of application u is placed on the same region of the IoT object requested by the application u

rational choice is to group applications microservices according to their requirements: fog modules with strict latency constraints require installation on the edge, e.g., to support real-time processing of data streams from a local IoT device. Other microservices may need computational power not available on edge nodes, this is the case, e.g., of online machine learning algorithms. This fog-cloud dichotomy greatly simplifies the placement of such fog-native applications, since while the first group hosts microservices that could be or need to be deployed on the edge, the second one is the set of microservices without strict latency requirements and, in turn, possibly hard computational requirements; for the latter, cloud deployment is assumed apriori. This minimal partitioning results in the functional bi-partition shown in Figure 2.

A. Network Model

We consider a standard fog system architecture [8], [9] where the network architecture consists of a central cloud, which is assumed of unlimited computational capabilities – relatively to edge nodes – and a set of fog regions connected to the central cloud. Each fog region comprises servers with specific computational capabilities in terms of memory, CPU and storage. We consider a batch of applications to be deployed on such an infrastructure. Each application is described by a list of requirements in terms of memory, CPU, storage and bandwidth. The objective is to deploy the applications on the infrastructure in order to maximize a certain profit function while satisfying the resources constraints and applications' demands. Throughout this article the target performance figure is the number of concurrent fog-applications hosted simultaneously on the infrastructure; more general objective functions can be studied in the same framework and are left as part of future works. The problem is to find a set of mappings of applications onto fog-regions to maximize such objective function, where each such mapping engenders diverse resource occupation vectors and its own profit value.

More formally, we consider a fog system deployed over a set of geographic regions $K = \{1, \ldots, K\}$. Region k hosts a set S_k of edge servers or units. We denote s_{k_i}, with $i \in \{1, \ldots, n_k\}$ a specific edge unit deployed within the

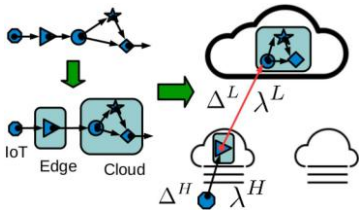

Fig. 2. Cascade of throughput-aware partitioning of a fog application into two chunks, and subsequent placement of the fog chunk onto an edge node.

k-th region; for the sake of notation we denote the central cloud as S_0. The resources of edge unit s_{k_i} are represented by capacity vector $\mathbf{C}_{k_i} = (C^M_{k_i}, C^P_{k_i}, C^S_{k_i})$. The first component of the capacity vector is the memory capacity, while the second component is the processing capacity; the third component denotes the storage capacity, i.e., the data volume that can be accommodated on the storage of the edge unit. The fog infrastructure can be described by an undirected weighted graph $G = (V, E)$ where $V = \{S_i \cup U_i\}_{i \in K}$ and $E \subseteq \{\{i,j\} | i,j \in V, i \neq j\}$. The weight of each edge $\{i,j\} \in E$ consists of the latency on the link d_{ij}, and the link bandwidth B_{ij}. Let $N(S_i) = \{S_j | \{j, i\} \in E\}$. Figure 1 reports a pictorial example of such fog infrastructure.

B. Application Model

The application model adopted in this article is based on the distributed data flow model [10], where distributed components of an IoT application perform different computational tasks. We model each application like a weighted DAG where the nodes represent the application modules and the edges between nodes represent the execution order of the modules. The weight of each edge represents the throughput generated by one microservice performing some computation and sending the results to another component downstream. More formally, each application u is defined by the weighted digraph (G_u, λ_u), where $G_u = (V_u, E_u)$ and $\lambda_u : E_u \rightarrow \mathbb{R}^+$, i.e., $\lambda_u(m,n)$ represents the maximum throughput that can be generated by the microservice m and sent to microservice n of the same application. In each fog region k we have a set of applications to be deployed, U_k. From here on out, we identify the application and the device from which data are requested with same symbol. We say that application u "belongs" to a given region because the IoT object is located there. Such region is denoted S_u for the sake of notation. Furthermore, we denote with $U = \cup_{i=1}^{K} U_i$ the set of all applications to be deployed on the infrastructure. A further assumption that we shall introduce is that each application is deployed on a set of servers defined by the neighbourhood of the region from which the data are generated, as in Figure 1. To this respect, we remark that the practice to ensure connectivity between containers is to interface them at the application layer, e.g., connecting them by http(s) protocol, so that multi-hop routing at the network layer is not viable for joint optimisation as it would introduce queuing and routing delay at each hop [11]. In the proposed system model, placement on neighbouring regions still attains load balancing while greatly simplifying system design. Nevertheless, in the numerical section we perform a thoroughly comparison of our reference system with other solutions violating such assumption.

III. Problem Formulation

In this section we describe the overall optimisation problem for the deployment of fog applications across multiple connected fog regions, corresponding to the reference architecture of Figure 1. First, we describe the original problem and describe its complexity and approximability. Afterwards, we introduce the problem split into the cascade of two sub-problems which we analyse in the rest of this article.

Main problem: Given a set of DAG-like applications U and a fog infrastructure $G = (V, E)$, deploy the maximum number of applications in order to maximize the provider's revenue while satisfying the applications' delay requirements and the infrastructure capacity.

The deployment of an application $u \in U$ is represented by a mapping between each module $v \in V_u$ and a server of the infrastructure such that all the applications requirements and infrastructure capacity constraints are satisfied. These constraints involve the CPU, memory, storage and bandwidth constraints for the infrastructure, and delay constraints for the application. This latter constraint involves the computation of the optimal throughput for each edge of the application $(v,w) \in E_u$ given the placement of application's nodes v and w. In order to compute such throughput values we should take into account the maximal path, in terms of latency, from the source to the destination node. This would result in having a non-linear constraint involving the binary variables for the placement of each application's module and the continuous variables for the computation of the optimal throughput for each application's edge. The mathematical formulation of the main problem can be found in the Appendix. Regarding its computational complexity, it can be observed that – by fixing throughput variables – it can be proved to be NP-hard by polynomial time reduction from a virtual network embedding problem, known to be strongly NP-hard. As discussed in the Appendix, such problem cannot be approximated by a polynomial-time algorithm. Hence, we have the following result:

Proposition 1: Unless P = NP, the application placement problem can not be approximated in polynomial time within a factor of $n^{\frac{1}{2}-E}$ for any $E > 0$, where $n := 1 + \sum_{i=1}^{K} n_i$.

Here, the term n represents the total number of servers available for microservices' deployment: we have n_i servers for each fog region $i \in K$, plus the cloud region which counts as a single server with infinite capacity.

A. Resolution Approach

The previous result rules out the possibility to devise efficient approximation algorithms to solve the main problem altogether, in all cases of practical interest. Thus, in order to render the original problem more tractable, we split it into the cascade of two sub-problems. In the first one, each application is partitioned in two chunks in order to minimise the maximum throughput between them, and imposing that all

the application's modules belonging to the same chunk will be deployed in the same region of the infrastructure. In this manner, the communication overhead between application's components and the number of binary and continuous variables for the placement and throughput computation are reduced. In the second sub-problem, once fixed the partition for each application solving the previous problem, a feasible deployment per application's chunk is found. The latter problem will be further transformed into an integer linear programming problem.

As introduced before, after the partitioning step, each fog application will consist of two chunks: the first one is the subset of application microservices apriori executed in cloud, whereas the second is the subset encompassing microservices that can be deployed either in fog or in cloud. By assumption, all applications are supposed to adhere to such a fog-oriented *partitioning*.[1] Of course, in general, any partition of microservices into two chunks will do. However, we shall consider both the case of throughput-agnostic and that of throughput-aware partitioning. With the former, the application partition is oblivious to the data flow across microservices of the same application. In the latter case, instead, we shall optimise partitioning based on the communication requirements of the microservice architecture. In both cases, anyhow, it is reasonable that a fog-native partitioning would be completely infrastructure-agnostic, thus accounting for requirements of a tagged application only.

Note that the partitioning phase can be realised by a static and offline algorithm processing each application individually. Depending on the business model, the infrastructure provider can offer the application owner with a cloud service to containerise her/his applications into a fog and a cloud chunk using the proposed partitioning algorithm. For the sake of simplicity, we have omitted all the aspects related to the orchestration of the registries where the images of the chunks are finally stored. In fact, multiple schemes can be envisaged, e.g., either a single cloud registry, or also several per region registries caching application chunk images. Once the registries have been populated with the chunk images of the applications concerned with target areas, the placement algorithm could leverage, e.g., the Kubernetes control plane to switch on the fog chunks of applications in the target regions. How to optimise the registry placement and the chunk image caching process is part of our current research effort and is left out of the scope of the present manuscript.

The cloud computing literature provides well-known solutions in a single data-center, such as, for instance, Kubernetes' scheduling algorithms [5]. Although these algorithms show good results in a single-region scenario, they are not designed to perform under throughput constraints among several fog regions. Our target is an algorithmic solution identifying a region for the deployment of each partitioned application. Focusing on the region selection only complies with current practice in containers' orchestration, since Kubernetes orchestration procedures and/or optimised versions can be later applied within each single region to find the best local resource allocation. Overall, the complete placement scheme proposed consists of the following logical phases which we describe next as described in Figure 2:

i. *fog application partitioning:* application packaging into a cloud and a fog chunk;

ii. *region selection:* selection of target regions in the neighbourhood of the objects that generate data consumed by fog applications;

iii. *deployment:* using an off-the-shelf orchestrator (e.g., Kubernetes) to perform allocation within a fog region cluster according to standard or optimised local placement rules.

B. Throughput-Aware Fog Partitioning

An efficient bipartition of the application graph can reduce monitoring and networking costs. In fact, for the sake of example, we can consider the case of video surveillance applications. We can expect one or more of the application microservices to require significant amount of computational power in order to execute, e.g., sophisticated face-recognition techniques. Such microservices will be normally part of the cloud module. On the other hand, since the throughput generated by video sensors represents a serious bottleneck, we can also expect some filtering microservice to pre-process the raw video stream on edge nodes, e.g., to extract just the frames relevant for the task from the raw videostream. Thus, in order to reduce upstream the fog-to-cloud throughput of such an application, it is indeed optimal to install, whenever possible, such a microservice in the fog chunk.

More in general, an efficient throughtput-aware partitioning shall split microservices such as to minimize the total throughput flowing from the first set of the microservices partition, the fog chunk, to the second set of the partition, the cloud chunk. Let the weighted DAG $G_u = (V_u , E_u)$ represent the target application u. The application partitioning problem can be reduced to the *Minimum k-cut* problem on graph G_u. The standard definition for a *k-cut* is set of edges whose removal leaves k connected components [12]. A minimum k-cut problem asks for a minimum weight k-cut. In our context, we are looking for a minimum 2-cut on G_u. The reason behind such a minimal cut is twofold. First, since we are not assuming a hierarchical structure for the infrastructure, only fog regions and the cloud can perform computational tasks. The IoT devices, indeed, are not usually able to perform significant operations. Furthermore, within the same region, delay and bandwidth constraints are negligible meanwhile they are significant between two different regions. Hence, splitting an application in more than two chunks would be equivalent to distribute the applications among more than two regions and this would bring additional communication overhead in terms of latency and routing implementation issues for the applications. Finally, two different fog regions may not be directly connected forcing the chunks of the applications to communicate through other regions. However, in this case we would lose what we have gained in terms of latency with fog computing. Secondly, if we wanted to partition an application in more than two chunks, we should solve a minimum k-cut

[1] Depending on the virtualisation technology, a fog-oriented application partitioning will produce two containers/pods or two VM images; however this is not relevant for the rest of our discussion.

problem with $k \geq 3$ and the partitioning problem would be NP-hard [12].

Given a source and a destination node, the *minimum 2-cut* problem is equivalent to the *maximum flow* problem for the *min-cut max-flow* theorem [13] which can be efficiently solved by a maximum flow algorithm such as the Ford-Fulkerson algorithm [14]. Hence, solving an *s-t* cut problem we can solve our application partitioning problem as expressed formally by Proposition 2 where APP-PARTITIONING is the application partitioning problem described above and MIN-CUT is the *minimum s-t cut* problem.

Proposition 2: APP-PARTITIONING $\leq_p$ MIN-CUT.

Proof: We can show the polynomial reduction by taking an application DAG and selecting a source and a destination node (since we have a DAG structure there exists at least one node with no incoming edges, the source, and one node with no outgoing edges, the destination). Additionally, we set the maximum throughput of each application's edge as the maximum capacity of the graph. In this manner, by solving the min-cut problem on such graph we have a 2-cut partitioning of our application graph. ■

Incidentally, whenever a single source and sink node pair cannot be identified in the application DAG, it is always possible to add a virtual source (sink) node with large capacity and apply the algorithm to the resulting DAG.

The partitioning algorithm is described in Algorithm 1. The input is represented by the application graph and a subset C of application's microservices to be deployed in cloud by default. The algorithm adds a dummy target sink node t to the application graph; t is connected to each node in C using a dummy high-throughput edge. First, the algorithm establishes a source node through a topological sort of the application's DAG. Second, the Ford-Fulkerson's algorithm, applied to the augmented graph, returns the partition separating s and t with a cut of minimum capacity. All the nodes still reachable from the source node in the residual graph will be included in the `Edge` set, while the others will go to the `Cloud` set. We observe that the high throughput of the edges between nodes in C and node t grants that all nodes in C will belong to the `Cloud` set, as stated by Lemma 1.

Lemma 1: For each node $v \in V_u \setminus \{s\}$, if $v \in C$ then $v \in$ `Cloud` when Algorithm 1 terminates.

Proof: Let assume, by contradiction, that for some $v \in C$ it holds $v \notin$ `Cloud`. Since, by assumption, $v \neq s$, there exists an in-going edge to v, (v^*, v). If $v^* \notin C$, then $\lambda_u(v^*, v) < \lambda_u(v, t)$, contradicting the cut minimality found by the Algorithm 1. If $v^* \in C$ we can repeat the previous argument until we find the first node $v^{**} \notin C$, in the worst case until reaching node s. ■

Complexity: The complexity of Algorithm 1 is dominated by the time complexity of the Ford-Fulkerson's algorithm which is $O(|E_u| \cdot \lambda^*)$ for each application u, where λ^* is the maximum flow in the application network. We can improve the runtime through the Edmonds-Karp implementation of the algorithm [15], [16] obtaining a complexity of $O(|V_u| \cdot |E_u|^2)$ totally independent of the maximum flow of the application network.

Algorithm 1: Application Partitioning Algorithm

Input: Application graph $G_u = (V_u, E_u)$, $C \subseteq V_u$
Output : 2-cut partition of the application u

```
Cloud ← ∅;
Edge ← ∅;
V_u ← V_u ∪ {t};
for v ∈ C do
    E_u ← E_u ∪ {(v, t)};
    λ_u(v, t) ← +∞;
s ← first node in a topological order of G_u;
G_u* ← Max_Flow(G_u, s, t, λ_u);
for w reachable from s in G_u* do
    Edge ← Edge ∪ {w};
Cloud ← V_u ∖ Edge;
return (Edge, Cloud)
```

1) Throughput-Intensive Applications (Notation): From here on out, for the sake of brevity, for each application $u \in U$, we identify with u_A and u_B the `Cloud` and the `Edge` chunks, respectively. Finally, let denote c_u^M, c_u^S, c_u^P the total resource requirements, in terms of memory, storage and processing capacity, respectively, of u_B; with compact notation we denote $\mathbf{c_u} = (c_u^M, c_u^S, c_u^P)$.

While the proposed framework does apply to other classes of applications, e.g., memory-intensive or CPU intensive applications, we shall focus on optimising the allocation of resources for *throughput-intensive* fog applications. They are a critical class of IoT applications which includes, e.g., streaming mining applications [17], where processing directly on fog nodes leads to huge bandwidth savings. In this context, for instance, a raw video stream can be filtered to extract only the relevant parts of specific frames, e.g., those containing a face which need to be sent to the cloud to perform recognition. Hence, after the partitioning step showed in Figure 2, two different data units may be transmitted. We call, for each application u, Δ_u^H the data unit transmitted directly from the IoT sensor (e.g., video camera frames) to the Fog chunk. Δ_u^L is the data unit transmitted from the Fog chunk to the Cloud chunk once a processing step has been performed (e.g., face images).

Delay Constraints: These two different data units are sent at different rate; we denote λ_u^H as the raw throughput generated by the IoT device and λ_u^L for the throughput generated towards the Cloud chunk once the Fog chunk of the application has processed the stream. In this case, typically $\lambda_u^H \geq \lambda_u^L$. In the fog resource allocation problem described in the next section the throughput λ_u^H and λ_u^L are decision variables which can be optimise in order to meet the applications' delay requirements. In fact, we can assume that each application u has to output every $1/F_u$ seconds a result like a positive or negative face recognition match. The cloud chunk is installed in the central cloud S_0. We can hence consider the whole processing chain involved by the two-chunks and the related data transmission delay. We should also include the processing delay d_u of application u (if deployed back to back to the IoT object), plus the communication delay d_{uj}, which is the additional delay to retrieve data from region where the sensor belongs, when the

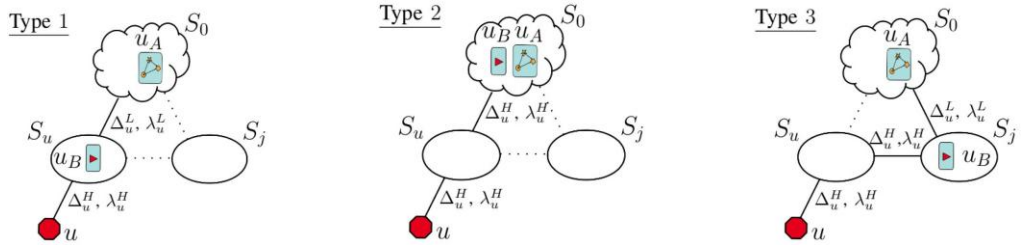

Fig. 3. Configurations types for the deployment of the `Edge` module u_B; `Cloud` module u_A is always installed in cloud.

Fog chunk is installed in region j. In the resource allocation problem we need to consider the processing and transferring time. Actually, the processing time for each information unit depends on the throughput between application chunks. Any application placement has to guarantee the application to process an information unit Δ_u in $\frac{1}{F_u}$ seconds. Thus, the allocation of such throughput depends on the application deployment configurations. Since the `Cloud` chunk is always installed on the central cloud, the three basic fog configurations to deploy application u are as in Figure 3:

Type 1: Fog chunk deployed on S_u; higher throughput λ_u^H flows between IoT object u and region S_u, with IoT data unit Δ_u^H. Δ_u^L served between S_u and S_0 with low throughput λ_u^L;

Type 2: Fog chunk deployed on central cloud S_0; IoT data Δ_u^H is served between S_u and S_0 with high throughput λ_u^H;

Type 3: Fog chunk deployed on a neighbouring fog region $S_j \neq S_u$; lower throughput required between S_j and central cloud S_0. However, the IoT data $\Delta_u = \Delta_u^H$ is served between S_u and S_j with high throughput λ_u^H.

C. Fog Resource Allocation Problem

The objective of the edge-infrastructure owner is to maximize the revenue obtained by provisioning her fog infrastructure to application tenants. By using the traditional scheme of pay per use, it can settle a cost in order to deploy an application. A tenant owning application u pays $f_{u,k} > 0$ euros per application installed in region k.

The objective is thus to host containerized fog applications such in a way to maximize the owner revenue, while satisfying the applications' requirements. We can obtain the optimal reward for a given batch of applications.

Decision variables $x_{u,k,i}$ are boolean variables indicating the placement of the fog chunk of application u on the i-th server of region k. Further, decision variables $\lambda_u^H, \lambda_u^L \in \mathbb{R}^+$ represent the optimal throughput in the large and small data unit transfer mode for application u, respectively. Indeed, the throughput declared for each application's link in the partitioning step is an upper bound on the actual throughput generated on each link of the application. The optimal allocation policy using a mixed integer non linear program (MINLP) writes:

$$\text{maximize:} \quad \sum_{(u,k) \in U \times K} f_{u,k}\, x_{u,k} \tag{1}$$

$$\text{subject to:} \quad \sum_{u \in U} \mathbf{c_u}\, x_{u,k,i} \leq \mathbf{C_{k_i}}, \quad \forall k \in K,\ \forall i \in S_k \tag{2}$$

$$\sum_{u \in U_k} x_{u,k}\, \lambda_u^L + x_{u,0}\, \lambda_u^H + \sum_{j \in N(S_k)} \sum_{v \in U_j} x_{v,j}\, \lambda_v^L \leq B_{k0}, \quad \forall k \in K \setminus \{0\} \tag{3}$$

$$\sum_{u \in U_k} x_{u,j}\, \lambda_u^H + \sum_{u \in U_j} x_{u,k}\, \lambda_u^H \leq B_{kj}, \quad \forall j,k \in E, j,k \neq 0 \tag{4}$$

$$\left(d_u + \frac{\Delta_u^H}{B_u} + d_{uj} + \frac{\Delta_u^H}{\lambda_u^H} + \frac{\Delta_u^L}{\lambda_u^L}\right) x_{u,j} + \left(d_{u0} + \frac{\Delta_u^H}{\lambda_u^H}\right) x_{u,0} + \left(d_{u0} + \frac{\Delta_u^L}{\lambda_u^L}\right) x_{u,u} \leq \frac{1}{F_u} \quad \forall u \in U, \forall j \in N(S_u) \tag{5}$$

$$\sum_{k \in K} x_{u,k} \leq 1 \quad \forall u \in U \tag{6}$$

$$\sum_{k \in K \setminus \{N(u) \cup \{u\}\}} x_{u,k} \leq 0 \quad \forall u \in U \tag{7}$$

$$x_{u,k,i} \in \{0,1\} \quad \forall (u,k) \in U \times K \quad \forall i \in S_k \tag{8}$$

$$\lambda_u^H, \lambda_u^L \in \mathbb{R}^+ \tag{9}$$

where we let $x_{u,k} = \sum_{i \in S_k} x_{u,k,i} \quad \forall (u,k) \in U \times K$ for notation's sake. The objective function is the revenue gained by the infrastructure owner. The constraint (2) is meant component-wise: it bounds the resources utilization on fog servers in terms of memory, processing and storage capacity, respectively. Also, (3) and (4) bound the throughput generated by applications with respect to links' capacity. Equation (3) accounts for all traffic from region k to the central cloud, whereas (4) accounts for the throughput across adjacent regions as in Figure 3c. By constraint (5), the total transmission and computing time needs to be smaller than the service rate of the application. We assume that, according to (6), each application has at most one deployment region, since, given the limited resources, it is not always possible to deploy all the applications. In particular, (7) indicates that each application can be deployed only on neighbour regions or on its original region.

The decision variables are the binary variables for the placement and the continuous variables for the throughput. Prob. 1–9 is a combination of a placement and a multi-commodity flow problem. For the sake of tractability, in the

next section we offer a transformation to a pure placement problem.

IV. Pure Placement Problem

The aforementioned transformation is attained by fixing the continuous decision variables of the MINLP, i.e., λ_u^L and λ_u^H. To do so, it is sufficient, for each application $u \in U$, to fix the minimum throughput required in order to deliver the output at target rate F_u, given its configuration type and deployment region.

Type 1: Processing each information unit and providing an output result should happen at rate $\frac{1}{F_u}$; by accounting for all processing and communication delay we write

$$d_u + d_{uo} + \frac{\Delta_u^H}{B_u} + \frac{\Delta_u^L}{\lambda_u^L} \leq \frac{1}{F_u} \tag{10}$$

which can be solved for equality in λ_u^L;

Type 2: For each application u, we have

$$d_u + d_{uo} + \frac{\Delta_u^H}{B_u} + \frac{\Delta_u^H}{\lambda_u^H} \leq \frac{1}{F_u} \tag{11}$$

In this case we are solving for λ_u^H; we observe that it must hold indeed $\lambda_u^H \geq \lambda_u^L$.

Type 3: If u_B is deployed in a region neighbor of the original region of u, it holds

$$d_u + d_{uj} + d_{jo} + \frac{\Delta_u^H}{B_u} + \frac{\Delta_u^H}{\lambda_u^H} + \frac{\Delta_u^L}{\lambda_u^L} \leq \frac{1}{F_u} \tag{12}$$

In this case, in order to have a unique solution in the minimum throughout, we impose additional constraints, namely we restrict to the set of solutions such that

$$\frac{\lambda_u^H}{\lambda_u^L} = \frac{\Delta_u^H}{\Delta_u^L} \tag{13}$$

Once we performed the above identification, the original problem becomes:

$$\text{maximize:} \sum_{(u,k) \in U \times K} f_{u,k}\, x_{u,k} \tag{14}$$

$$\text{subject to:} \sum_{u \in U} c_u\, x_{u,k,i} \leq C_{k_i}, \quad \forall k \in K,\ \forall i \in S_k \tag{15}$$

$$\sum_{u \in U_k} x_{u,k}\, \lambda_u^L + x_{u,o}\, \lambda_u^H + \sum_{j \in N(S_k)} \sum_{v \in U_j} x_{v,j} \lambda_v^L \leq B_{ko}, \quad \forall k \in K \setminus \{0\} \tag{16}$$

$$\sum_{u \in U_k} x_{u,j}\, \lambda_u^H + \sum_{u \in U_j} x_{u,k}\, \lambda_u^H \leq B_{kj}, \quad \forall j,k \in E, j,k \neq 0 \tag{17}$$

$$\sum_{k \in K} x_{u,k} \leq 1 \quad \forall u \in U \tag{18}$$

$$\sum_{k \in K \setminus (N(u) \cup \{u\})} x_{u,k} \leq 0 \quad \forall u \in U \tag{19}$$

$$x_{u,k,i} \in \{0,1\}, \quad \forall (u,k) \in U \times K, \quad \forall i \in S_k. \tag{20}$$

A. Complexity

Problem (14)–(20) has formulation similar to a specific Knapsack problem, namely the Multidimensional Multiple-Choice Knapsack (MMCK) problem. This latter problem is one of the most complex versions of the Knapsack Problem's family. In this version of the KP, there are different groups of items with the constraint that exactly one item for each group must be picked [18].

In the multidimensional multiple-choice Knapsack problem there are m resource types and the amount of available resources is given by vector $C = (C^1, \ldots, C^m)$. Also, there are n disjoint classes of items, J_i $i = 1, \ldots, n$, where each class J_i has r_i items. Each item $j \in J_i$ has profit value $v_{ij} \geq 0$ and weight vector $W_{ij} = (w_{ij}^1, \ldots, w_{ij}^m) \geq 0$, where each weight component $w_{ij} \geq 0$, $k = 1, \ldots, m$, is the occupation of resource k of item $j \in J_i$.

The objective is to pick exactly one item from each class in order to maximize the total profit value of the pick, subject to resource constraints:

$$\text{maximize:} \quad Z = \sum_{i=1}^{n} \sum_{j=1}^{r_i} v_{ij}\, x_{ij} \tag{21}$$

$$\text{subject to:} \sum_{i=1}^{n} \sum_{j=1}^{r_i} w_{ij}^k\, x_{ij} \leq C^k, \quad 1 \leq k \leq m \tag{22}$$

$$\sum_{j=1}^{r_i} x_{ij} = 1, \quad 1 \leq i \leq n \tag{23}$$

$$x_{ij} \in \{0,1\}, \quad \forall 1 \leq i \leq n,\ 1 \leq j \leq r_i. \tag{24}$$

The pure placement problem and the MMCK have similar formulations. The next result proves NP-hardness of our problem by reduction from the MMCK problem.

Proposition 3: Problem (14) is *NP*-hard.

Proof: We prove the NP-hardness by reduction from a specific case of the MMCK problem. In particular, we consider the case where each class has the same number of elements r [19]. Furthermore, the NP-hardness can be easily proved by reduction from the MMCK problem with $m = 1$, since starting from $m = 1$ the problem is already known to be NP-hard (for $m = 1$ the problem is known as Multiple Choice Knapsack). Hence, for every instance of a MMCK with n classes consisting of r elements and 1 dimension, we can reduce it to an instance of our problem. Indeed, it is sufficient to consider an instance of (14)–(20) with n applications and a single region with $r - 1$ servers with infinite capacity. Each application has three possible configurations: deployed in cloud, deployed in fog or not deployed. In this manner, each application defines a class of configurations consisting of r elements. Each element's weight can be mapped to the throughput generated by each application towards the cloud. In this manner, the only one capacity constraint is the bandwidth constraint on the fog-cloud link. ■

Several algorithmic solutions have been proposed in the literature for the MMCK problem [19], [20]. However, such heuristic solutions cannot guarantee approximation bounds on the produced allocation. Actually, to the best of the authors' knowledge, no approximation algorithm is available to date

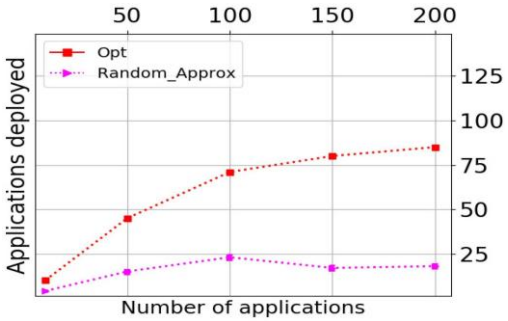

Fig. 4. Optimal solution vs. Rounding.

for this specific Knapsack problem with a variable number of dimensions. In [21], authors designed a polynomial time approximation scheme (PTAS) for the MMCK problem when the number of knapsack dimensions is $O(1)$. In our case, instead, the number of dimensions varies with the number of applications and regions.

One of the standard techniques to obtain approximation schemes for binary integer NP-hard problems is to relax first the integrality of choice variables thus obtaining a linear program and then round the so-obtained continuous solution of the linear program to an integer solution of the original one [22]. One technique is to consider the continuous relaxed variable as the probability to pick the tagged choice variable (setting that variable equal to 1). Typically, a difficult challenge for such a relaxation algorithm is to guarantee that the obtained integer solution is a feasible one for the initial integer problem. In our case, the rounding technique does not represent a good strategy to obtain a valid approximation scheme. As shown in Figure 4, the approximated feasible solution suffers a significant optimality gap with respect to the optimal one. The figure reports the best feasible solution obtained by the applying the rounding technique to the relaxed version of problem (14). One of the main issues is represented by the constraint (18), which misleads the relaxation procedure to pick solutions quite far from optimal. Indeed, by relaxing this constraint, more than one variable can assume a non-zero value. In this case, we need to choose which one to pick entirely. In our case we considered a probability distribution defined for each application on the set of its possible deployments. However, as confirmed by Figure 4, this leads to a very conservative approximation factor since the most probable variables to be picked are may be arbitrarily far from the ones chosen by the optimal integer solution.

Finally, in Table II, we have resumed the complexity results for the problems tackled in this article. The main problem is strongly NP-hard since it induces a VNE problem; furthermore, it does not admit any tight polynomial-time approximation scheme. The application partitioning problem has a polynomial time algorithm as per the reduction showed in Proposition 2; it becomes NP-hard as the number of chunks is greater or equal to 3. Finally, the Fog Resource Allocation problem is formulated as a MINLP problem: by fixing the continuous throughput variables it becomes the Pure Placement problem which is NP-hard by reduction from the MMCK problem.

TABLE II
COMPLEXITY OF ALL THE SUB-PROBLEMS

Problem	*Complexity*
Main Problem	Strongly NP-hard, no PTAS better than $n^{\frac{1}{2}-\epsilon}$
App-partitioning	Poly for $k = 2$, NP-hard for $k \geq 3$
Fog Resource Allocation	NP-hard
Pure Placement	NP-hard, No FPTAS

Algorithm 2: Fog Placement Algorithm (FPA)

Input: $G = (V, E)$, U
Output : Container placement for each $u \in U$

while $U \neq \emptyset$ **do**
  $place \leftarrow \{\}$;
  for $i = 1, \ldots, K$ **do**
    for $u \in U_i$ **do**
      $A \leftarrow \emptyset$;
      if *verify*(S_i, u) **then**
        $A \leftarrow A \cup \{S_i\}$;
      for $S \in N(S_i)$ **do**
        if *verify*(S, u) **then**
          $A \leftarrow A \cup \{S\}$;
      if $A \neq \emptyset$ **then**
        $place[u] \leftarrow select(A, u)$;
      else
        $place[u] \leftarrow \emptyset$;
  // select the application to be deployed
  $u^* \leftarrow min\_resource\_region(place)$;
  deploy$(u^*, place[u^*])$;
  // Update G
  update$(G, S_{place[u^*]}, S_{u^*}, u^*)$;
  $U \leftarrow U \setminus \{u^*\}$

V. FOG PLACEMENT ALGORITHM

Hereafter, we describe FPA, a greedy solution for (14) which is meant to provide practically viable solution for the placement problem described above. The key intuition of the algorithm is to pick an aggregated view of fog regions' resources utilisation, thus permitting to measure the effect of placement as a pseudo-gradient descent in the space of occupied resources, while treating the alternatives for the deployment of same application as different yet exclusive instances.

FPA operates an iterative application deployment. At each step, for each region and for each application u which belongs to that region, it selects the set A of admissible regions for the deployment of chunk u_B. Such set includes all the regions satisfying the computational and throughput requirements of a tagged application. Preliminarily, a feasibility check is performed through a *verify* procedure: given a region and application's requirement, it verifies whether exists at least one server in the region to host u_B, i.e., if the server has enough space in terms of CPU, memory and storage. Further, throughput requirements are verified against each configuration type for each application, by ensuring that the residual bandwidth of involved links satisfies the minimum throughput requirement corresponding to the tagged configuration type.

Algorithm 3: *Select* Procedure

Input: A, set of admissible regions for the deployment of chunk u_B
Output : A region for the deployment

```
// Build a pseudo-gradient vector for each region in A
for S ∈ A do
    v_m ← c_u^M / residual_mem(S);
    v_p ← c_u^P / residual_proc(S);
    v_s ← c_u^S / residual_stor(S);
    if S ≠ S_u then
        if S ∈ N(S_u) then
            // Case 3
            b_1 ← λ_u^H / residual_band({u,S});
            b_2 ← λ_u^L / residual_band({S,o});
            v̄_S ← (v_m, v_p, v_s, b_1, b_2);
        else
            // S = S_o, case 2
            b_1 ← λ_u^H / residual_band({o,u});
            v̄_S ← (v_m, v_p, v_s, b_1, o);
    else
        // Case 1
        b_1 ← λ_u^L / residual_band({o,u});
        v̄_S ← (v_m, v_p, v_s, b_1, o);
return arg min_{S∈A} {|v̄_S|^2}
```

The *select* procedure is reported in Algo. 3: *select* first calculates, for all the admissible regions for the deployment of an application u, a pseudo-gradient $\bar{v}_S$ ($\forall S \in A$). Its components are calculated at lines 2, 3, 4, 7–9, 11, and 14, respectively, by estimating the normalized decrease of each resource type in case of deployment with tagged configuration. The output is the region with the minimum pseudo-gradient (line 16). Once a feasible region is selected for each application, the algorithm 2 chooses the application to be deployed first. This step is executed by the *min_resource_region* procedure. It takes the *place* map as input and returns the application to which the region with the minimum pseudo-gradient is associated, as computed per the *select* procedure.

Subsequently, once the algorithm has selected the application to be deployed, it updates the computational capacities of the server hosting the chunk of that application.

Afterwards, the algorithm updates the graph structure decreasing the available bandwidth of links connecting regions selected for the deployment (line 17). It iterates until all applications have been considered.

Finally, we observe that, while the presentation of FPA is performed in the case of a batch of applications all available at the same time, the formulation can be easily adapted to the online case, where applications to be deployed arrive and depart, by simply including the release of resources to the update procedures.

Complexity: The computational complexity of the FPA algorithm is derived by noting that procedures *verify*, *updateServer* and *update* have constant time complexity. Procedure *select* computes a vector for each eligible region in the set A. In the worst case, the cardinality of A is at most $K - 1$. Hence, the complexity of the *select* procedure is $O(K)$. The cardinality of U is U, and the maximum cardinality of a neighbourhood of a certain region is $O(K)$ in the worst case. The cardinality of the set of applications to be ranked is $O(U)$ at each step. Finally, the complexity of FPA is $O(U^2 \cdot K^2 + U^2)$ in the worst case.

VI. Numerical Results

In this section we evaluate the performance of the combined scheme using throughput-aware application partitioning and placement. We compare the performance with the case of throughput-agnostic application partitioning, and with the case of placement driven by virtual network embedding [23].

In order to understand the average behaviour of the produced solutions, we examined the performance of the algorithms on a number of randomly-generated graphs. The network infrastructure is modelled with an undirected graph connecting a central cloud to a fixed number K of fog regions, where $K = 10$ in our experiments. Thus, the central cloud and fog regions form a star topology of cloud-to-fog connections, namely cloud-links. For every topology realization, links between two fog regions are added according to an Erdo˝s-Rényi random graph model, where a link exists between two regions with probability q. Finally, each link in the resulting network is assigned a bandwidth of 15 Mbps, both for cloud-links and fog-links.

A batch of fog applications is generated for each experiment; we considered $U = \{80, 90, 100, 150, 200\}$ for the applications cut evaluation, $U = \{60, 70, 80, 90, 100\}$ for the comparison with network embedding approaches and $U = \{10, 50, 100, 150, 200\}$ for the comparison with the standard Kubernetes placement described in Section VI-A3. The demands of each application of the batch for CPU, storage, memory and throughput are modelled as uniform independent random variables. The mean value of such variables is dictated by the nominal value we measured on a benchmark application, that is a plate-recognition application packaged as a two-modules microservice. The fog microservice module can process the video stream either in cloud or on a fog node. The resulting distribution of the key parameters for the applications microservices are enlisted in Table III; symbol u_o refers to the nominal values we measured on a proprietary fog platform for the plate recognition app [24], [25]. Each application is generated as a DAG with a number of nodes sampled from the set $\{5, \ldots, 20\}$.

Finally, the probability that an application belongs to region $1 \le k \le K$ follows a truncated Pareto distribution of parameter α, i.e., $\mathbb{P}\{R_u > k\} = k^{-\alpha}/\gamma$, where R_u is the random variable representing the index of the region assigned to the application u and normalization constant $\gamma = \sum_{h=1}^{K} h^{-\alpha}$.

In the proposed scenario, the servers available within each region belong to three classes, depending on the resources they

TABLE III
DISTRIBUTION OF THE APPLICATIONS' MICROSERVICES REQUIREMENTS OF CPU, MEMORY, STORAGE AND THROUGHPUT

Requirement	*Mean Value* (u_0)	*Range* ($u \in \mathcal{U}$)
CPU (c_u^P)	1250 MIPS	$[500, 2000]$ MIPS
Memory (c_u^M)	1.2 Gbytes	$[0.5, 2]$ Gbytes
Storage (c_u^S)	3.5 Gbytes	$[1, 8]$ Gbytes
Throughput ($\lambda_u(m,n)$)	3 Mbps	$[1, 5]$ Mbps

TABLE IV
CHARACTERISTICS OF THE THREE CLASSES OF FOG SERVERS: LOW, MEDIUM AND HIGH

Type	*CPU (MIPS)*	*Memory (GB)*	*Storage (GB)*
Low	5000	2	60
Medium	15000	8	80
High	44000	16	120

are equipped with, namely *low*, *medium* and *high* class. The computational characteristics are listed in Table IV where CPU performances are described in terms of Million Instructions Per Second (MIPS) representing the processor's speed of a server. This standard measure [26] is relevant in a heterogeneous environment as fog computing, since all the servers in a specific fog region can mount different cores and different processors. Hence, given this heterogeneity, this measure represents a basic universal metric for processors' performances, used in other fog simulators [27], that is independent from the specific model of the processor. The number of servers per region is determined at each experiment sample as follows. Each region is meant to satisfy same fraction of the expected aggregated demand. More precisely, each region is equipped with aggregated resource vector $(1+\beta)\frac{1}{K}\sum_{u} c_{u_0}$. The parameter β is a slack parameter tuning the probability that resources available in a fog region are underprovisioned/overprovisioned compared to the aggregated demand. Finally, the servers' population of the tagged region is generated by allocating iteratively servers of different types at random until the region resource budget is exhausted.

Well-known Fog simulators and emulators presented in the literature [27], [28] do not support natively scenarios with multiple fog regions. Hence, we developed a Python-based simulator for the evaluation of the above algorithms. The Gurobi solver has been used to solve the optimal placement problem (OPT). Experiments have been conducted on an Ubuntu Linux server with 32 core AMD Opteron 1.4GHz CPU and 64GB of memory. Each data point depicted is the result of an average over 30 instances where the network infrastructure is fixed and the application and servers distributions change. All the results are averaged with the corresponding 95% confidence interval.

A. Reference Algorithms

We compared our proposed scheme with two benchmark solutions. More precisely, we evaluate the application splitting and the deployment (FPA) algorithms. For the former one we make a comparison with a throughput-agnostic splitting algorithm. For the latter we implemented a state of the art virtual network embedding algorithm presented in [23] and two variants of the Kubernetes scheduler [5].

1) Throughput-Agnostic Partitioning: For the evaluation of the partitioning algorithm we compare it with a throughput-agnostic method that, given an application DAG as input, performs a random cut of the application graph. In detail, the algorithm takes in input the application graph and perform a visit of the DAG. Once an order of the application nodes is induced by the visit, a random cut is chosen on the so-defined order without considering the total throughput on the chosen cut.

2) Network Embedding Algorithm: The general problem of deployment of a DAG-like fog-application can be seen as a network embedding problem, we compare our solution with a standard algorithm. The general procedure [23] for the embedding algorithm consists of three main steps to be executed sequentially for each application.

1) *Topological sorting*. A topological sort is obtained by simply performing a modified visit of the application DAG. The DAG-like structure of the application ensures that there exists at least one possible order of the application nodes.

2) *Application's nodes mapping*. This procedure selects, for each node, a set of possible regions where the node can be deployed. Once this set is defined, one region is selected according to a fixed priority function. In our implementation, we select the region S_i that maximizes

$$res_{CPU}(S_i) \prod_{j \in N(S_i)} res_{BW}(S_i, S_j), \qquad (25)$$

where $res_{BW}(S_i, S_j)$ and $res_{CPU}(S_i)$ indicate the residual bandwidth of physical link $\{S_i, S_j\}$ and the residual CPU capacity in region S_i, respectively. If any of the application's node has not eligible region, the application is not deployed.

3) *Application's links mapping*. If a mapping for all the modules of the application is found, the next step is to find a path between each couple of regions where two application nodes are mapped in the previous step. Given two regions assigned to two application's nodes, we compute all the paths between these two regions and we take the first path with enough bandwidth capacity for the application's link. If all the application's links are mapped to a set of paths of the network infrastructure, the application is finally deployed on the infrastructure.

Before applying the embedding procedure for each application, a deployment priority should be established among them. Hence, we sort the batch of applications on the basis of their total throughput demand. In this manner, the next application chosen for the deployment is the one with the minimum total throughput demand.

Remark: In the experimental section, we shall consider two variants of the VNE approach. The first is the one where embedding is performed only to regions neighbouring the original one for the target fog application u. The second one, VNE-MultiHop, assumes routing can be performed across all regions. The VNE-MultiHop variant may be at advantage by performing load balancing more efficiently across the whole deployment, it relies on a strong technological assumption in

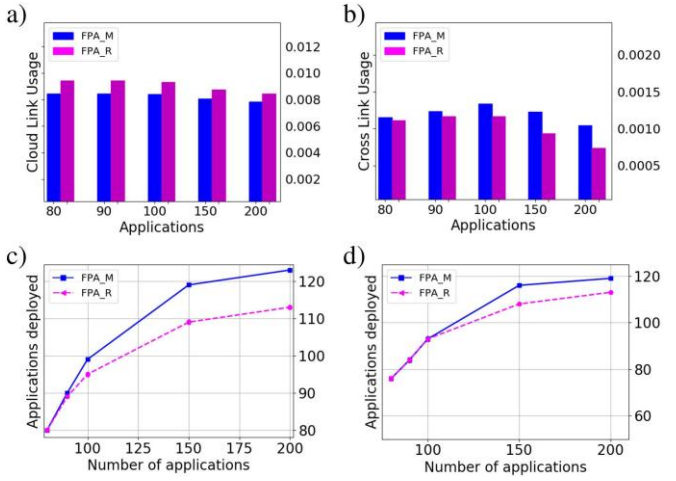

Fig. 5. Comparison of throughput-aware vs. throughput-agnostic partitioning. a) cloud links usage, $q = 1/2$, $\beta = 0.5$; b) cross links usage, $q = 1/2$, $\beta = 0.5$; c) number of deployed applications for $q = 1/2$, $\beta = 0.5$; d) number of deployed applications for $q = 1/2$, $\beta = -0.5$.

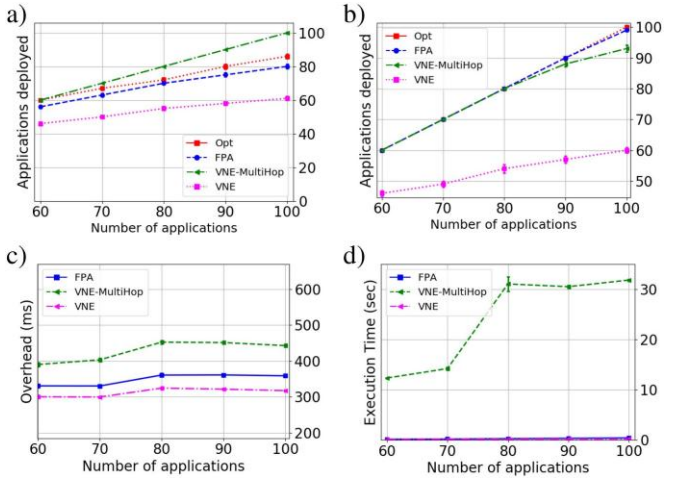

Fig. 6. a) OPT vs. FPA vs. VNE in terms of number of applications deployed, $q = 1/2$, $\beta = 0.5$; b) OPT vs. FPA vs. VNE in terms of number of applications deployed, $q = 1/2$, $\beta = -0.9$; c) FPA vs. VNE in terms of average total delay overhead, $q = 1/2$, $\beta = -0.9$; d) Average execution time for FPA and VNE, $q = 1/2$, $\beta = -0.9$.

that it requires joint orchestration of network and application layers; conversely all other solutions envisioned in this article can be fully implemented at the application layer.

3) Kubernetes: Kubernetes scheduling algorithms consist of three main components: the *filtering* step where a set of servers is selected for each pod; the *ranking* which selects, on the basis of a specific priority function, the best server for the deployment among the ones previously selected; finally, the *deployment* step is dedicated to the final deployment of the pods on the servers selected on the previous step. In our case we selected *LeastRequestedPriority* as the priority function for the second step. In this manner each server node is ranked based on the fraction of the node resources that would be free after the deployment of the selected pod.

For the sake of a comparison with our approach, we adapted the Kubernetes scheduler to a multi-region scenario implementing two variants of the aforementioned procedure: the first one runs the basic Kubernetes algorithm in every single fog region; each region is hence thought as a separate cloud where a fog server is chosen to host the application chunks to be deployed. We observe that in this approach, only deployments of type 1 and type 2 are possible. The second approach is to consider the whole fog deployment as a unique region. Hence, in the filtering step, Kubernetes shall select all the servers able to host a tagged chunk across all fog regions.

B. Experimental Results

1) Application Partitioning: In Figure 5 we evaluate the effect of the fog partitioning algorithm. Figure 5a reports on the fraction of cloud-link usage for each application deployed on the network infrastructure both for the throughput-aware application splitting (FPA_M) and for the throughput-agnostic (FPA_R) [29]. It is clear from the figure that with the min-cut splitting the cloud-link usage is almost constant as the size of applications batch increases. Furthermore, the cloud-link usage of the throughput-aware partitioning is always less than the random cut, as expected. The cloud-link usage of the throughput-agnostic decreases as the size of application batches decreases for the applications deployment. Indeed, to increase the number of applications deployed FPA tries to save bandwidth towards the cloud as shown in [29]. Figure 5b represents the amount of fog-to-fog links usage for both the two approaches. The throughput-aware partitioning exploits a larger number of cross-links: as a consequence it can deploy a larger number of applications compared with the throughput-agnostic solution, as confirmed by Figure 5c.

Figure 5c shows the effectiveness of the proposed application cut in terms of applications placement in an overprovisioning situation in terms of available computational resources. Indeed, with a minimum cut for the application splitting, FPA_M is able to deploy almost the totality of the applications' requests until $|U| = 100$. This is also the reason why the cross-link usage for the throughput-aware approach keeps increasing steadily until $|U| = 100$, as shown in Figure 5b. In fact, FPA_R suffers from the non-optimised splitting of the applications, especially i scenarios, where bandwidth saturation represents a bottleneck for the applications deployment [29]. Thus, we conclude that optimised application splitting causes a significant increase of the infrastructure capacity to host fog applications. Figure 5d shows that the relative gain of the min-cut approach under tighter constraints on available resources remains significant versus throughput-agnostic approaches.

2) Network Embedding: For the VNE algorithm we implemented two variants: the standard one (VNE), adapted to our scenario, allows the deployment of microservices only on regions neighbouring the target one; the extended one (VNE-MultiHop), conversely, allows paths across region nodes when mapping links of the application graph during the embedding procedure.

Figure 6a) reports on the number of applications deployed across the infrastructure in an underprovisioning scenario. We can observe that, as expected, VNE-MultiHop can deploy all

the applications since we remove the constraint of using only one-hop links for the applications mappings. The difference between OPT and FPA is reduced and, on the other hand, the VNE presents a significant loss. The VNE algorithm, indeed, tries to deploy all the applications towards the cloud until the bandwidth between the original regions and the cloud is exhausted, confirming the bandwidth towards the cloud to be a real bottleneck for the applications deployment problem. The VNE-MultiHop, instead, escape from this problem by allowing the applications' links mapping among the paths that go from the original region of the applications and the cloud. Indeed, given the metric (25), the best resulting region for each application deployment will be the cloud (for the unlimited computational power). However, as highlighted in Figure 6c), this approach can lead to a significant time overhead. In Figure 6b) reports we have the same results when the fog has enough computational resources (overprovisioning). In this case we can see a little loss from the VNE-MultiHop still due to the metric (25) for the region selection. The VNE-MultiHop continues to select the cloud without considering the quantity of resources available in Fog. This can easily leads to a bandwidth saturation towards the cloud.

Figure 6c) validates our choice of deployment using two hop schemes when offloading to neighbouring regions in Type 3 configurations. Indeed a multi-hop approach such as VNE-MultiHop may incur in a significant latency overhead due to the multi-hop path traversing several links to connect two different regions. On the other hand FPA and VNE have a small difference in delay, confirming that it is the multi-hop approach to introduce a significant latency overhead.

Finally, in 6d) we show the execution time of the three algorithms. It is clear from the figure that VNE and FPA have comparable and scalable execution times. The VNE-multiHop presents highest execution times given by the computation of the paths between all pairs of region nodes where two application nodes are mapped.

3) Kubernetes: Finally, in Figure 7a) and b) we have compared our solution with the two Kubernetes algorithm's variants: Kub and Kub1, respectively. In the second scenario all algorithms tend to deploy a larger number of applications than in the first one. This is expected since the latter both has more computational resources and more connected regions. In both figures we can observe that FPA performs close to the optimal solution. The poor performance of the Kub algorithm indicates that offloading towards neighbourhood fog regions is key to efficient fog resource allocation. Also, as the number of applications increases, the gap between FPA and Kub1 broadens. The reason can be ascribed to two key difference between FPA and Kub1. First, the deployment order of applications in FPA matches remaining resources at each step, by choosing the application with minimum resources consumption pseudo-gradient. In Kub1, conversely, applications are deployed in a predefined order. Second, for Kub1 neglects crosslinks bandwidth utilization, it leads quickly to bandwidth resources consumption. On the other hand, FPA's better performance is due to the fact that it accounts for bandwidth occupation of both cloud-links and crosslinks. This confirms the key role of accounting for bandwidth consumption on

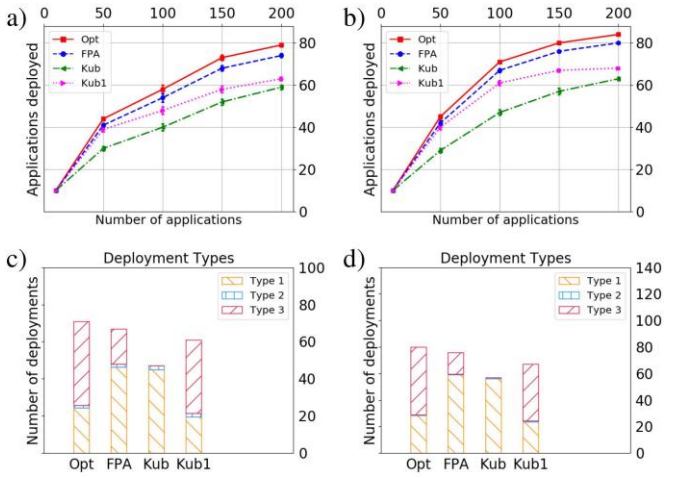

Fig. 7. Number of deployed applications with respect to Kubernetes algorithms: a) $q = 1/3$, $\beta = -0.2$; b) $q = 1/2$, $\beta = 0.5$; c) Configuration types distribution for a typical solution instance with $U = 100$, $q = 0.5$ and $\beta = 0.5$; d) Configuration types distribution for a typical solution instance with $U = 150$, $q = 0.5$ and $\beta = 0.5$.

both fog-links kinds of network links for the final applications deployment, in order to avoid early bottleneck formation. Figure 7c and Figure 7d provide further insights into the structure of the produced solutions. There, we have reported the number of deployments of each type produced by different algorithms in a throughput-dominated scenario. The OPT and Kub1 solutions prioritize type 3 configurations over type 1 configurations, while the opposite is seen to occur for FPA. Overall, as expected, deployments on fog, i.e., type 1 and type 3, are more frequent than type 2 configurations, since they save bandwidth on cloud-links. Actually, for a batch of 150 applications the number of type 2 deployments becomes negligible (Figure 5f). These results highlight the importance of the fog side in the applications deployment even when we have an overprovisioning situation at the edge.

VII. Related Work

In cloud and mobile cloud computing, the problem of microservices applications deployment has been thoroughly studied. As described in [30], cloud software design privileges modular software structures, where applications are composed by multiple coupled components known as microservices. In [31], applications are assumed to have a microservice architecture: the authors proposed a distributed mechanism for microservices scheduling at the edge. The objective is minimise the service latency for all the applications to be deployed. However, applications are simply represented as sets of independent microservices. In [32], instead, microservice fog applications are represented as DAGs, where graph nodes represent an application's modules, and edges between nodes represent dependencies between them. With such an application structure, the general application deployment problem bears several similarities with the graph embedding problem [33], [34], a well-known NP-hard problem. In our context, even for two-module containerised fog-applications, such problem is proved to still be NP-hard. In [35], a DAG

structure for IoT applications similar to the one used in this article is considered. But the aim of the work is different from ours, because the objective there is to perform communication load balancing among the microservices of a single application via their replication across the infrastructure. Furthermore, the infrastructure model is characterised only by the nodes that host some replicas of the application's microservices.

In [36], application provisioning is studied from the perspective of the network infrastructure. A fully polynomial-time approximation scheme is derived for single and multiple applications deployment, showing large QoS performance improvement with respect to applications' bandwidth and delay figures. However, the applications are represented as a single module communicating with a set of IoT devices generating data. For this reason, authors focused mostly on applications' networking requirements.

Taneja and Davy [10] defined a placement algorithm by mapping the directed acyclic graph of the modules of an IoT-based application into fog and cloud nodes. Numerical results show performance gains in terms of latency, energy and bandwidth constraints, compared to edge-agnostic placement schemes. Our work, conversely, develops an optimization framework able to account for both traffic and computing demands of a whole batch of applications to be deployed over multiple regions.

The DAG-like applications deployment on an infrastructure network reminds the *Virtual Network Embedding* problem, a well-known NP-hard problem deeply studied especially in the topic of Virtual Network Function (VNF) placement [33]. Many heuristic solutions have been proposed in the literature [37] since the general problem presents strong inapproximability results [38]. The main difference with respect to our context is that, usually only CPU constraints are taken into account in VNF problems. Conversely, in a fog scenario, where resources are limited and heterogeneous, all the requirements (CPU, RAM and Storage) of each microservice should be considered. Furthermore, the network infrastructure where VNF applications are deployed is usually not partitioned into regions.

Partitioning the applications' computation process represents a promising technique to guarantee high performance in mobile cloud or edge computing. The authors of [39] studied the problem of computation partitioning with the aim of maximizing the application throughput in processing the streaming data. A genetic algorithm is able to find the best partition in between the cloud and the mobile at runtime. In [40], a general technique to minimize execution time of IoT applications is proposed. The model introduced takes into account computation and communication delays. By reduction to the Matrix Chain Ordering Problem, an algorithm is provided in order to solve the optimization problem via dynamic programming, with time-complexity log-linear with respect to the number of operators of the application. With respect to such solutions, we have a different objective, that is to maximize the infrastructure owner's revenue by combining efficient applications' partitioning while avoiding network bottleneck.

One of the novelties of this work is the multi-regions scenario for the applications partitioning and orchestration. With respect to the container technologies discussed in this work, the de-facto standard for container orchestration is Kubernetes [6] even if new technology solutions for the edge are being proposed [41]. As described in Section VI-A3, resource allocation in Kubernetes proceeds by first enlisting servers able to host a target application module in a container pod. In the native cloud version, the actual container deployment is performed agnostic of the notion of fog-region and agnostic of network conditions. Our orchestration logic is able to addresses also the locality of object demands and their cumulative effect onto the communication infrastructure.

VIII. Conclusion

In this article, we have introduced a joint partitioning and optimization framework for throughput-intensive applications. We showed that a smart cut of the applications' computation flow in between cloud and fog is beneficial to cope with the applications' performance requirements while improving the revenue figures of the infrastructure owner. The scheme for the resource allocation combines a multi-commodity flow and a placement problem, but can be reduced to a Knapsack problem by introducing throughput proportionality and considering only the placement formulation. A greedy algorithm, FPA, is able to perform efficiently with respect to the optimal solution by placing partitioned applications using a pseudo-gradient approach. The numerical experiments confirm the scalability properties of the proposed fog orchestration scheme and the efficiency in terms of infrastructure owner's revenue and additional communication overhead compared to the most existing solutions in the literature.

Appendix

A. Main Problem Formulation

In this section we provide a formal description of the main problem presented in this article.

Variables: We introduce a binary variable for the placement of each applications' microservice on a single server of the infrastructure:

$$x_{k,s}^{u,m} = \begin{cases} 1, & \text{if microservice } m \text{ of application } u \\ & \text{is placed on server } s \text{ of region } k \\ 0, & \text{otherwise} \end{cases}$$

$\forall u \in U$, $\forall m \in V_u$, $\forall k \in K \cup \{0\}$, $\forall s \in S_k$.

For the applications' links mapping to the physical paths of the infrastructure we introduce a second kind of binary variables:

$$y_p^{u,(m,n)} = \begin{cases} 1, & \text{if } (m,n) \in E_u \text{ is mapped to } p \in P \\ 0, & \text{otherwise} \end{cases}$$

$\forall u \in U$, $\forall (m,n) \in E_u$, $\forall p \in P$, where P is the set of all paths between nodes in the infrastructure graph G.

Furthermore, since the objective of the problem is to maximize the number of applications entirely deployed on the infrastructure (i.e., maximizing the revenue), we introduce a third binary variable to indicate whether an application is entirely deployed on the infrastructure:

$$z_u = \begin{cases} 1, & \text{if application } u \in U \text{ is entirely deployed} \\ 0, & \text{otherwise} \end{cases}$$

$\forall u \in U$. Hence, z_u will be set to 1 if and only if all the microservices and links of application u are deployed on the infrastructure.

Finally, we have continuous variables to control the optimal throughput generated on applications' links in order to satisfy the delay constraints:

$$\lambda_u(m, n) \in \mathbb{R}^+, \qquad \forall u \in U, \forall (m.n) \in E_u.$$

Constraints: First, we have resource capacity constraints for each fog server of the infrastructure:

$$\sum_{u \in U} \sum_{m \in V_u} c_m x_{k,s}^{u,m} \leq C_{k,s} \qquad \forall k \in K, \forall s \in S_k. \tag{26}$$

Then we have all the constraints related to the applications' links mapping:

$$\sum_{p \in P_{k,k^*}} y_p^{u,(m,n)} = \sum_{s \in S_k} x_{k,s}^{u,m} \wedge \sum_{s \in S_{k^*}} x_{k^*,s}^{u,n}, \tag{27}$$

$\forall (k, k^*) \in V \times V$, $\forall u \in U$, and $\forall (m, n) \in E_u$. Constraint (27) ensures that a unique physical path of the network infrastructure is used by an application link whenever the application nodes connected by such link are deployed on the extreme nodes of the physical path (the symbol $\wedge$ represents the Boolean *"and"* operator).

We have to add constraints to guarantee a unique placement for each component of each application (microservices and links):

$$\sum_{m \in V_u} \sum_{k \in V} \sum_{s \in S_k} x_{k,s}^{u,m} = |V_u| \vee \sum_{m \in V_u} \sum_{k \in K} \sum_{s \in S_k} x_{k,s}^{u,m} = 0, \quad \forall u \in U, \tag{28}$$

$$\sum_{k \in V} \sum_{s \in S_k} x_{k,s}^{u,m} \leq 1, \qquad \forall u \in U, \forall m \in V_u. \tag{29}$$

$$\sum_{p \in P} y_p^{u,(m,n)} \leq 1, \qquad \forall u \in U, \forall (m,n) \in E_u. \tag{30}$$

Constraint (28) guarantees that either all the applications' microservices are deployed or no one of them is deployed (the symbol $\vee$ is the Boolean "*or*" operator). The same thing must be guaranteed for the applications' links:

$$\sum_{(m,n) \in E_u} \sum_{p \in P} y_p^{u,(m,n)} = |E_u| \vee \sum_{(m,n) \in E_u} \sum_{p \in P} y_p^{u,(m,n)} = 0. \tag{31}$$

An additional constraint should be added to guarantee a complete deployment for each application. If all the microservices of an application are deployed then all the application's links must be mapped to a physical path and vice versa.

$$\sum_{m \in V_u} \sum_{k \in V} \sum_{s \in S_k} x_{k,s}^{u,m} = |V_u| \Leftrightarrow \sum_{(m,n) \in E_u} \sum_{p \in P} y_p^{u,(m,n)} = |E_u|. \tag{32}$$

The following are constraints on the bandwidth capacity for each physical link $(k, k^*) \in E$:

$$\sum_{u \in U} \sum_{(m,n) \in E_u} \sum_{p \in P : (k,k^*) \in p} \lambda_u(m, n)\, y_p^{u,(m,n)} \leq B_{kk^*}. \tag{33}$$

Furthermore, we have constraints that bind x and y variables to z for each application $u \in U$:

$$z_u = \left\lfloor \frac{\sum_{m \in V_u} \sum_{k \in K} \sum_{s \in S_k} x_{k,s}^{u,m}}{|V_u|} \wedge \left\lfloor \frac{\sum_{(m,n) \in E_u} \sum_{p \in P} y_p^{u,(m,n)}}{|E_u|} \right\rfloor \right\rfloor. \tag{34}$$

Indeed, for each application $u \in U$, the decision variable z_u is set to one if and only if all the application's modules and links are deployed.

Finally, we have delay constraints for each application:

$$\max_{p_u \in P_u} \left\{ \sum_{(m,n) \in p_u} \sum_{p \in P} \left(\frac{\Delta_{m,n}^u}{\lambda_u(m, n)} + y_p^{u,(m,n)} \right) d_{k,k^*} \right\} \leq \frac{1}{F_u}, \tag{35}$$

where P_u represents the set of all directed paths between the source and the destination node of the application u, and $\Delta_{m,n}^u$ is the data transmitted from the module m to module n of the application u.

Objective: The objective function is the revenue of the infrastructure's owner based on the number of applications entirely deployed:

$$\text{maximize} \sum_{u \in U} f_u z_u. \tag{36}$$

B. Complexity and Approximability of the Main Problem

As it can be noticed from the formulation, the main problem may resemble a Virtual Network Embedding (VNE) problem with in addition the decision variables and constraints for the throughput on the applications DAGs edges. Overall, the main problem appears non-linear. Furthermore, the result in [38] shows that the VNE problem is strongly NP-hard with inapproximability results obtained by reduction from the Maximum Stable Set Problem (MSSP). Thus, unless P = NP, no polynomial time approximation scheme can be found within a factor of $n^{1-\varepsilon}$ for any $\varepsilon > 0$. The formal proof of the result in Proposition 1 follows.

Proof: The proof holds by reduction from the VNE problem, as defined in [38], to the main problem. Let assume that the throughput variables on the applications' edges are fixed. Given the substrate graph $G^o = (V^o, E^o)$ of VNE problem and the set of requests R, where each $r \in R$ is a graph $G^r = (V^r, E^r)$, we map each node in V^o to a server of the fog infrastructure $G = (V, E)$ and we map all each request $r \in R$ to an application with V^r microservices and E^r edges. The nodes and edges capacities of G^o are mapped to servers capacities and to the bandwidth capacities of the links in G, respectively. The demand for each node and the traffic demand of each edge in G^r are mapped to microservices requests and edges' throughputs of each application. Exploiting the transitivity of polynomial reduction, [38, Corollary 3.3] applies to our problem. ■

REFERENCES

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.

[2] Y. Gan and C. Delimitrou, "The architectural implications of cloud microservices," *IEEE Comput. Archit. Lett.*, vol. 17, no. 2, pp. 155–158, Jul.–Dec. 2018.

[3] E. Wolff, *Microservices: Flexible Software Architecture*. Boston, MA, USA: Addison-Wesley Prof., 2016.

[4] L.-N. Ni, J.-Q. Zhang, C.-G. Yan, and C.-J. Jiang, "A heuristic algorithm for task scheduling based on mean load on grid," *J. Comput. Sci. Technol.*, vol. 21, no. 4, pp. 559–564, 2006.

[5] *Kubernetes Scheduler*. Accessed: 2020. [Online]. Available: https://github.com/kubernetes/

[6] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," *Commun. ACM*, vol. 59, no. 5, pp. 1837–1852, May 2016.

[7] *Kubernetes*. Accessed: 2020. [Online]. Available: http://kubernetes.io/

[8] C. Pahl and B. Lee, "Containers and clusters for edge cloud architectures—A technology review," in *Proc. IEEE 3rd Int. Conf. Future Internet Things Cloud*, 2015, pp. 379–386.

[9] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Comput.*, vol. 4, no. 2, pp. 26–35, Mar./Apr. 2017.

[10] M. Taneja and A. Davy, "Resource aware placement of IoT application modules in fog-cloud computing paradigm," in *Proc. IFIP/IEEE Symp. Integr. Netw. Serv. Manag. (IM)*, Lisbon, Portugal, 2017, pp. 1222–1228.

[11] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017.

[12] V. V. Vazirani, *Approximation Algorithms*. Heidelberg, Germany: Springer, 2013.

[13] G. Dantzig and D. R. Fulkerson, "On the max flow min cut theorem of networks," *Linear Inequalities and Related Systems* (Annals of Mathematics Studies No. 38). Princeton, NJ, USA: Princeton Univ. Press, 1956, pp. 215–221.

[14] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," in *Classic Papers in Combinatorics*. Boston, MA, USA: Springer, 2009, pp. 243–248.

[15] E. A. Dinic, "Algorithm for solution of a problem of maximum flow in networks with power estimation," *Soviet Math. Doklady*, vol. 11, no. 5, pp. 1277–1280, 1970.

[16] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM*, vol. 19, no. 2, pp. 248–264, 1972.

[17] L. Canzian and M. V. D. Schaar, "Real-time stream mining: Online knowledge extraction using classifier networks," *IEEE Netw.*, vol. 29, no. 5, pp. 10–16, Sep./Oct. 2015.

[18] M. M. Akbar, M. S. Rahman, M. Kaykobad, E. G. Manning, and G. C. Shoja, "Solving the multidimensional multiple-choice knapsack problem by constructing convex hulls," *Comput. Oper. Res.*, vol. 33, no. 5, pp. 1259–1273, 2006.

[19] B. Han, J. Leblet, and G. Simon, "Hard multidimensional multiple choice knapsack problems, an empirical study," *Comput. Oper. Res.*, vol. 37, no. 1, pp. 172–181, 2010.

[20] M. Hifi, M. Michrafy, and A. Sbihi, "Heuristic algorithms for the multiple-choice multidimensional knapsack problem," *J. Oper. Res. Soc.*, vol. 55, no. 12, pp. 1323–1332, 2004.

[21] B. Patt-Shamir and D. Rawitz, "Vector bin packing with multiple-choice," *Discrete Appl. Math.*, vol. 160, nos. 10–11, pp. 1591–1600, 2012.

[22] B. Korte and J. Vygen, *Approximation Algorithms*. Heidelberg, Germany: Springer, 2012.

[23] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17–29, 2008.

[24] D. Santoro, D. Zozin, D. Pizzolli, F. De Pellegrini, and S. Cretti, "Foggy: A platform for workload orchestration in a fog computing environment," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Hong Kong, China, Dec. 2017, pp. 231–234.

[25] *FogAtlas*. Accessed: 2020. [Online]. Available: https://fogatlas.fbk.eu/

[26] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *Proc. Int. Conf. High Perform. Comput. Simulat.*, Leipzig, Germany, 2009, pp. 1–11.

[27] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments," *Softw. Pract. Exp.*, vol. 47, no. 9, pp. 1275–1296, 2017.

[28] R. Mayer, L. Graser, H. Gupta, E. Saurez, and U. Ramachandran, "EmuFog: Extensible and scalable emulation of large-scale fog computing infrastructures," in *Proc. IEEE Fog World Congr. (FWC)*, Santa Clara, CA, USA, 2017, pp. 1–6.

[29] F. Faticanti, F. De Pellegrini, D. Siracusa, D. Santoro, and S. Cretti, "Cutting throughput with the edge: App-aware placement in fog computing," in *Proc. 6th IEEE Int. Conf. Cyber Security Cloud Comput. (CSCloud) 5th IEEE Int. Conf. Edge Comput. Scalable Cloud (EdgeCom)*, Paris, France, 2019, pp. 196–203.

[30] J.-P. Arcangeli, R. Boujbel, and S. Leriche, "Automatic deployment of distributed software systems: Definitions and state of the art," *J. Syst. Softw.*, vol. 103, pp. 198–218, May 2015.

[31] A. Samanta, Y. Li, and F. Esposito, "Battle of microservices: Towards latency-optimal heuristic scheduling for edge computing," in *Proc. IEEE Conf. Netw. Softwa. (NetSoft)*, Paris, France, 2019, pp. 223–227.

[32] N. K. Giang, M. Blackstock, R. Lea, and V. C. Leung, "Developing IoT applications in the fog: A distributed dataflow approach," in *Proc. 5th Int. Conf. Internet Things (IOT)*, Seoul, South Korea, 2015, pp. 155–162.

[33] X. Cheng *et al.*, "Virtual network embedding through topology-aware node ranking," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 2, pp. 38–47, 2011.

[34] A. Brogi, S. Forti, and A. Ibrahim, "How to best deploy your fog applications, probably," in *Proc. IEEE 1st Int. Conf. Fog Edge Comput. (ICFEC)*, Madrid, Spain, 2017, pp. 105–114.

[35] R. Yu, V. T. Kilari, G. Xue, and D. Yang, "Load balancing for interdependent IoT microservices," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Paris, France, 2019, pp. 298–306.

[36] R. Yu, G. Xue, and X. Zhang, "Application provisioning in fog computing-enabled Internet-of-Things: A network perspective," in *Proc. INFOCOM Conf. Comput. Commun.*, Honolulu, HI, USA, 2018, pp. 783–791.

[37] H. Cao, H. Hu, Z. Qu, and L. Yang, "Heuristic solutions of virtual network embedding: A survey," *China Commun.*, vol. 15, no. 3, pp. 186–219, Mar. 2018.

[38] E. Amaldi, S. Coniglio, A. M. Koster, and M. Tieves, "On the computational complexity of the virtual network embedding problem," *Electron. Notes Discrete Math.*, vol. 52, pp. 213–220, Jun. 2016.

[39] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 4, pp. 23–32, 2013.

[40] T. Elgamal, A. Sandur, P. Nguyen, K. Nahrstedt, and G. Agha, "DROPLET: Distributed operator placement for IoT applications spanning edge and cloud resources," in *Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*, San Francisco, CA, USA, 2018, pp. 1–8.

[41] *KubeEdge: An Open Platform to Enable Edge Computing*. Accessed: 2020. [Online]. Available: https://kubeedge.io/