# Cutting Throughput with the Edge: App-Aware Placement in Fog Computing

Francescomaria Faticanti*,‡, Francesco De Pellegrini*,◇, Domenico Siracusa*, Daniele Santoro* and Silvio Cretti*

***Abstract*—Fog computing extends cloud computing technology to the edge of the infrastructure to support dynamic computation for IoT applications. Reduced latency and location awareness in objects' data access is attained by displacing workloads from the central cloud to edge devices. Doing so, it reduces raw data transfers from target objects to the central cloud, thus overcoming communication bottlenecks. This is a key step towards the pervasive uptake of next generation IoT-based services.**

**In this work we study efficient orchestration of applications in fog computing, where a fog application is the cascade of a cloud module and a fog module. The problem results into a mixed integer non linear optimisation. It involves multiple constraints due to computation and communication demands of fog applications, available infrastructure resources and it accounts also the location of target IoT objects. We show that it is possible to reduce the complexity of the original problem with a related placement formulation, which is further solved using a greedy algorithm. This algorithm is the core placement logic of FogAtlas, a fog computing platform based on existing virtualization technologies. Extensive numerical results validate the model and the scalability of the proposed algorithm, showing performance close to the optimal solution with respect to the number of served applications.**

***Index Terms*—fog computing, microservice, resources allocation, placement**

## I. Introduction

Fog computing adopts cloud technology to move computation to the edge. It promises to solve the core problem of data explosion in the IoT domain [1]. Instead of performing raw data transfer to the cloud, in fact, data flows generated from objects, i.e., IoT devices, can be intercepted to extract information at the edge of the network. This architectural choice prevents massive, diffused and continuous raw data injection into the communication infrastructure, avoiding severe communication congestion [2]. Furthermore, compared to customary cloud-based IoT deployments, proximity to mobile or sensing devices lowers round-trip-time between target objects and backends of processing applications [3].

Further incentives in the development of fog computing solutions include the standardization of IoT deployments, typically encompassing several different technologies. For instance, this can ease management and maintenance of IoT services in industrial networks [4]. Local computing can also overcome privacy issues by confining raw data within specific geographical regions [5]. The fog orchestrator studied in this paper is part of FogAtlas, a platform designed to perform efficient deployment of fog computing applications according to the above guidelines [6, 7].

Edge resources optimization in this context is key. Indeed, compared to standard cloud technologies, which are typically utilized in overprovisioned datacenters, edge infrastructure owners can hardly count on systematic overprovisioning. In fact, last mile connections are traditionally owned by telecommunication operators, while edge computing clusters are relatively small compared to cloud datacenters. In turn, the scarce storage, memory and processing capabilities of edge units need to be optimized to meet customers' demands [8, 9]. This appears a challenging engineering problem to attain premier fog service provision sustaining localised data processing and low round-trip time.

In practice, the current paradigm of fog computing is based on a layered architecture, including a central cloud, a series of edge units, wireless gateways, and, finally, target objects which generate data and possibly actuate. Hence, virtual machines or containers can run either in the central cloud, or over edge units, depending on the requirements of IoT-based applications. Actually, the current practice in cloud application design tends to favour microservice-based development, both for reasons of availability and of scalability. Microservice applications are composed by coupled modules, such as a graphical user interface, a user repository, a web server, an image recognition module, a monitoring application, etc. Once interconnected using a specific communication and computing pattern, the microservice architecture delivers the intended functionality while preserving scalability, minimality and cohesiveness of the application [10].

To this respect, it is natural to assume that fog-native applications should adhere to the modular microservice paradigm. The simplest possible containerization of a fog application would actually split computing functionalities using just two containerized modules. First, a module typically comprising one or more microservices not directly involved in objects' data computation. Such module, i.e., the *cloud module*, should reside in the central cloud, possibly inside a pod or a virtual machine hosting the related microservices. The second module, namely the *fog module*, includes functionalities of processing of target objects' data. We assume it is hosted on a single container installed in the cloud or on edge nodes, depending on the placement operated by the fog orchestrator. Such a minimal containerization adheres to the current practice in cloud computing, since by colocating edge modules within

*Fondazione Bruno Kessler, via Sommarive, 18 I-38123 Povo, Trento, Italy, ‡University of Trento, via Sommarive, Trento, Italy, ◇University of Avignon, 339 Chemin des Meinajaries 84140, Avignon, France

same pods one would reduce monitoring and networking operations.

For the sake of concreteness, we shall often refer to an example of application which has been deployed on the FogAtlas platform [7]. It is a two-module video application where the fog module can be dispatched on an edge server close to a target video-camera to perform remote image pre-processing for plate recognition. Actually, stream mining is emerging as a reference fog-computing [11]. In such data-intensive service, processing directly on edge nodes permits large bandwidth savings. In particular, a raw video stream can be filtered through an image detection algorithm so that only tagged frames need to be forwarded to the cloud. Finally, only a small fraction of information is transferred towards the central cloud.

The problem of application placement, even for two-module based fog-applications, is proved to be NP-hard in our preliminary work [12]. Hence, the main objective of this work is to describe an efficient placement of fog application modules either on the edge or in the cloud for a batch of concurrent fog-based applications. In order to determine such a placement, constraints on computational and bandwidth requirements have to be factored in. Furthermore, specific constraints depend on the *locality*, of objects which are the target sources of fog applications. We remark that, compared to standard cloud computing orchestration, the presence of concurrent requests for data originating at a specific locations may generate hotspot conditions. An orchestration mechanism able to offload applications to neighboring regions of a hotspot proves an effective tool to increase system performance.

We shall introduce first the general placement problem and then describe our algorithmic solution. Our main contribution is an optimization framework for the orchestration of concurrent fog applications across multiple fog regions. Our scheme accounts for object's location and the constraints on both network and computing resources. Finally, it reacts to hotspot conditions by performing offloading to neighboring regions.

The rest of the paper is organized as follows. The next section reports on related works and how existing container orchestration technology can apply to fog computing. In Sec. III we describe the system model, including the abstractions we adopt for the applications' architecture, the network infrastructure and applications' deployment configurations. In Sec. IV we present the problem formulation, introducing the most general problem setting. The placement problem is addressed in Sec. V by proposing a greedy algorithm for its resolution given the NP-hardness of the problem. Numerical results are reported in Sec. VI. A concluding section ends the paper.

## II. Related Work

Efficient service deployment lies at the core of cloud computing research [13, 14]. In fog computing, the presence of remote, heterogeneous devices on edge nodes requires novel schemes to match QoS requirements and optimize network usage [9, 15].

As described in [16], cloud software design privileges modular software structures: applications may have multiple components known as microservices. In [15], microservice fog applications are represented like DAGs (Directed Acyclic Graphs), where graph nodes represent an application's modules, and edges between nodes represent dependencies between them. In this paper, we use weights to represent the throughput generated by outbound module interfaces. The general application deployment problem is a graph embedding problem [17], a standard NP-hard problem. In our context, even for two-module based fog-applications, such problem is proved to remain NP-hard.

Authors of [18] focused on the provision of QoS constrained, eligible deployments for applications. The problem is stated to be NP-hard with a reduction from the subgraph isomorphism problem. Preprocessing plus backtracking determines the final eligible deployment restricting the search space. But, no performance target is optimized.

In [8], application provisioning is studied from the perspective of the network infrastructure. A fully polynomial-time approximation scheme is derived for single and multiple application deployment, showing large QoS performance improvement with respect to applications' bandwidth and delay figures. However, computational requirements are not accounted for.

In [19], a general technique to minimize execution time of IoT applications is proposed. IoT applications are modeled as dataflow graphs where nodes are specific computational operators and edges between nodes express a dependency between them. 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. Our objective is different since we aim at maximizing the number of deployed applications.

Authors of [20] proposed a fog computing platform for deployment of applications on an infrastructure composed of datacenters and edge devices. The related placement problem is an instance of the Knapsack problem, solved via a heuristic algorithm. The work does not consider the cumulative applications' bandwidth demand across the infrastructure.

Taneja et al. [9] 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.

With respect to the container technologies discussed in this work, the de-facto standard for container orchestration is Kubernetes [21]. 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

Table I: Main notation used throughout the paper

| *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 $\lvert S_i\rvert = n_i$ |
| $U_k$ | set of applications requiring IoT data in region $k$ |
| $\lambda_u^H/\lambda_u^L$ | high/low throughput required by application $u$ |
| $\Delta_u^H/\Delta_u^L$ | large/small data unit of application $u$ |
| $F_u$ | output samples per second required by application $u$ |

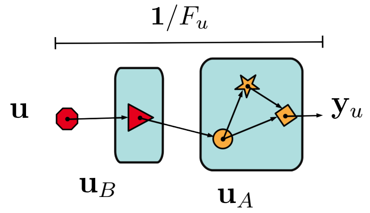

Figure 1: The modules cascade outputs a result $y_u$ every $1/F_u$ sec; $\mathbf{u}_A$ is the cloud module, whereas $\mathbf{u}_B$ is the fog module.

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.

## III. System Model

We consider a fog system deployed over a set of geographic regions $\mathscr{K} = \{1, \dots, K\}$. Fog region $k$ hosts a set $S_k$ of edge servers or units. We denote $s_{k_i}$, with $i \in \{1, \dots, n_k\}$, a specific edge unit deployed within the $k$-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 $\mathbf{C}_{k_i} = (C_{k_i}^M, C_{k_i}^P, C_{k_i}^S)$. The first component of the capacity vector is the memory capacity. The second component is the processing capacity, which determines the maximum load which can be sustained on the edge unit. Finally, the third component denotes the storage capacity, i.e., the data volume that can be accommodated on the storage of the edge unit. We assume that the storage of a containerized application is handled on the same unit where the container is deployed, with the aim to reduce the communication costs.

In region $k$, objects serve data required by a set of applications $U_k$. From here on out, we identify the application and the device from which data are requested with same symbol. The extension of the following optimization framework in the case of multiple requests for same IoT device is immediate, by considering virtual replicas of a tagged IoT device. 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. We leave access of applications to IoT objects of different regions for future works.

*Network Architecture.* The fog system can be described by a weighted graph $G = (V, E)$ where $V = \{S_i\}_{i \in \mathscr{K}} \cup \{S_0\}$ and $E \subseteq \binom{V}{2}$. The weight of each edge $\{i, j\} \in E$ consists of the delay, $d_{ij}$, of the link and the bandwidth of the link $B_{ij}$. Let $\mathscr{N}(S_i) = \{S_j | \{j, i\} \in E\}$.

*Application Architecture.* As depicted in Fig. 2, an application $u \in \mathscr{U}$ consists of two modules: the fog module $u_A$ and the cloud module $u_B$. In order to render the notion of computing and communication constraints more concrete, we refer to a benchmark application for face recognition in a video stream. As introduced before, modules for processing IoT data streams – face detection processing over the sequence of video frames in our example – are containerized in $u_B$. They can be deployed in the central cloud $S_0$ or on the edge, i.e., in regions $S_i, i = 1, 2, 3$. Conversely, cloud module $u_A$ contains all remaining logic, including, e.g., alarm generation in case a positive match is returned. The application has to output every $1/F_u$ seconds a result $y_u$ – in this case a positive or negative face recognition match. $u_A$ is installed in the central cloud $S_0$. We can hence consider the whole processing chain involved by the two-containers 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 to $u_A$, when $u_B$ is installed in region $j$.

The IoT source – in the example a videocamera – generates information units – video frames – of size $\Delta_u$, which are served at rate $B_u$ bit/s. We denote $\Delta_u^H = \Delta_u$. Conversely, $u_B$ transfers smaller information unit $\Delta_u^L$ to $u_A$.

Finally, we denote $c_u^M$, $c_u^S$, $c_u^P$ the resource requirements of of application $u$, in terms of memory, storage and processing capacity, respectively, of $u_B$; with compact notation we denote $\mathbf{c_u} = (c_u^M, c_u^S, c_u^P)$.

In the placement problem we need to consider the processing and transferring time. Actually, the processing time for each information unit depends on the throughput between application modules. Any application placement has to guarantee that the application to process an information unit $\Delta_u$ in $\frac{1}{F_u}$ seconds. Thus, the allocation of such throughput depends on the application deployment configurations. Since $u_A$ is always installed on the central cloud, the three basic fog configurations to deploy application $u$ are as in Fig. 2:

*Type 1:* $u_B$ deployed on $S_u$; higher throughput $\lambda_u^H$ flows between IoT object $u$ and region $S_u$, with IoT data unit $\Delta_u = \Delta_u^H$. $\Delta_u^L$ is served between $S_u$ and $S_0$ with low throughput $\lambda_u^L$;

*Type 2:* $u_B$ deployed on central cloud $S_0$; the IoT data $\Delta_u = \Delta_u^H$ is served between $S_u$ and $S_0$ with high throughput $\lambda_u^H$;

*Type 3:* $u_B$ deployed on a neighboring 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 $S_0$ with high throughput $\lambda_u^H$.

## IV. Problem formulation

The resource allocation problem is tackled from the perspective of the edge-infrastructure owner. Her aim is to maximize the revenue obtained in the provision of her fog infrastructure to application tenants. In fact, she settles a cost in order to

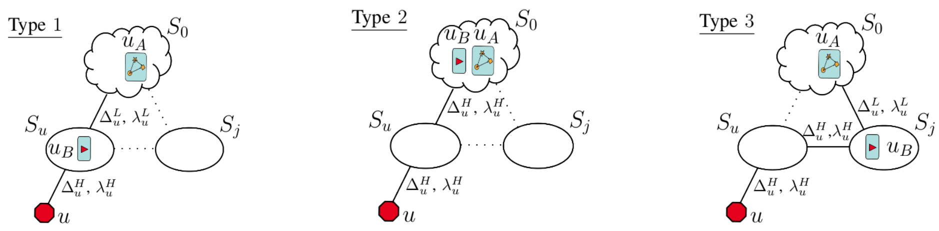
Type 1
$S_0$
$u_A$
$S_u$
$S_j$
$u_B$
$\Delta_u^L, \lambda_u^L$
$\Delta_u^H, \lambda_u^H$
$u$
Type 2
$S_0$
$u_B$ $u_A$
$\Delta_u^H, \lambda_u^H$
$S_u$
$S_j$
$\Delta_u^H, \lambda_u^H$
$u$
Type 3
$S_0$
$u_A$
$\Delta_u^L, \lambda_u^L$
$S_u$
$\Delta_u^H, \lambda_u^H$
$S_j$
$u_B$
$\Delta_u^H, \lambda_u^H$
$u$

Figure 2: The three configurations types for the deployment of the fog module $u_B$; cloud module $u_A$ is always installed in cloud.

deploy an application using the traditional scheme of pay per use. A tenant owning application $u$ pays $f_{u,k} > 0$ euros per container installed in region $k$.

The objective is to schedule the 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 set of application requests. Hence, the following formulation provides an upper bound on the average reward that can be attained with perfect information.

Decision variables $x_{u,k,i}$ are boolean variables indicating the placement of the application $u$ on the $i$-th server of region $k$. Further, decision variables $\lambda_u^H, \lambda_u^L \in \mathbb{R}^+$ represent throughput in the large and small data unit transfer mode of application $u$, respectively. The optimal allocation policy using a mixed integer non linear program (MINLP) writes:

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

subject to:

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

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

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

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

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

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

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

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

where we let $x_{u,k} = \sum_{i\in S_k} x_{u,k,i} \quad \forall (u,k) \in \mathscr{U}\times\mathscr{K}$ 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. (3) accounts for all traffic from region $k$ to the central cloud, whereas (4) accounts for the throughput across adjacent regions as in Figure 2c. 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. In particular, (7) indicates that each application can be deployed only on neighbor 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 reduction to a pure placement problem, which is proved to be NP-hard by reduction from the $m$-dimensional knapsack problem.

## V. Pure placement problem

The reduction is attained by fixing the continuous decision variables of the MINLP, i.e., $\lambda_u^L$ and $\lambda_u^H$. To do so, we fix the minimum throughput required for each application $u\in\mathscr{U}$ to deliver the output at target rate $F_u$, given the configuration type and the deployment region for $u_B$.

*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

$$d_u + d_{u0} + \frac{\Delta_u^H}{B_u} + \frac{\Delta_u^L}{\lambda_u^L} \leq \frac{1}{F_u} \tag{10}$$

which can be solved for equality in $\lambda_u^L$;

*Type. 2:* For each application $u$, we have

$$d_u + d_{u0} + \frac{\Delta_u^H}{B_u} + \frac{\Delta_u^H}{\lambda_u^H} \leq \frac{1}{F_u} \tag{11}$$

In this case we are solving for $\lambda_u^H$; 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

$$d_u + d_{uj} + d_{j0} + \frac{\Delta_u^H}{B_u} + \frac{\Delta_u^H}{\lambda_u^H} + \frac{\Delta_u^L}{\lambda_u^L} \leq \frac{1}{F_u} \tag{12}$$

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 an Integer Linear Programming problem which is shown to be NP-hard in [12] by reduction from the well-known multidimensional knapsack problem.

### A. Placement algorithm

Given the NP-hardness of the pure placement problem, we propose a greedy solution called FPA. In Algorithm 1 we report the pseudocode of the algorithm.

**Algorithm 1:** Fog Placement Algorithm (FPA)

```
Input: $G = (V, E)$, $\mathscr{U}$
Output : Container placement for each $u \in \mathscr{U}$
1  while $\mathscr{U} \neq \emptyset$ do
2      $place \leftarrow \{\}$;
3      for $i = 1, \dots, K$ do
4          for $u \in U_i$ do
5              $\mathscr{A} \leftarrow \emptyset$;
6              if $verify(S_i, u)$ then
7                  $\mathscr{A} \leftarrow \mathscr{A} \cup \{S_i\}$;
8              for $S \in \mathscr{N}(S_i)$ do
9                  if $verify(S, u)$ then
10                     $\mathscr{A} \leftarrow \mathscr{A} \cup \{S\}$;
11             if $\mathscr{A} \neq \emptyset$ then
12                 $place[u] \leftarrow select(\mathscr{A}$,u);
13             else
14                 $place[u] \leftarrow \emptyset$;
       // select the application to be deployed
15     $u^* \leftarrow min\_resource\_region(place)$;
16     deploy($u^*, place[u^*]$);
       // Update $G$
17     update($G, S_{place[u^*]}, S_{u^*}, u^*$);
18     $\mathscr{U} \leftarrow \mathscr{U} \setminus \{u^*\}$
```

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 $\mathscr{A}$ of admissible regions for the deployment of module $u_B$ container. 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 some 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.

The *select* procedure is reported in Algo. 2: *select* first calculates, for all the admissible regions for the deployment of an application $u$, a gradient $\bar{v}_S$ ($\forall S \in \mathscr{A}$). Its components are calculated at lines 1, 2, 3, 7-8, 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 gradient (line 16). Once a feasible region is selected for each application, the algorithm 1 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 gradient (computed through

**Algorithm 2:** *Select* procedure

```
Input: $\mathscr{A}$, set of admissible regions for the deployment of the module
       $u_B$
Output : A region for the deployment
// Build a gradient vector for each region in $\mathscr{A}$
1  for $S \in \mathscr{A}$ do
2      $v_m \leftarrow \frac{c_u^M}{residual\_mem(S)}$;
3      $v_p \leftarrow \frac{c_u^P}{residual\_proc(S)}$;
4      $v_s \leftarrow \frac{c_u^S}{residual\_stor(S)}$;
5      if $S \neq S_u$ then
6          if $S \in \mathscr{N}(S_u)$ then
               // Case 3
7              $b_1 \leftarrow \frac{\lambda_u^H}{residual\_band(\{u,S\})}$;
8              $b_2 \leftarrow \frac{\lambda_u^L}{residual\_band(\{S,0\})}$;
9              $\bar{v}_S \leftarrow (v_m, v_p, v_s, b_1, b_2)$;
10         else
               // $S = S_0$, case 2
11             $b_1 \leftarrow \frac{\lambda_u^H}{residual\_band(\{0,u\})}$;
12             $\bar{v}_S \leftarrow (v_m, v_p, v_s, b_1, 0)$;
13     else
           // Case 1
14         $b_1 \leftarrow \frac{\lambda_u^L}{residual\_band(\{0,u\})}$;
15         $\bar{v}_S \leftarrow (v_m, v_p, v_s, b_1, 0)$;
16 return $\arg\min_{S \in \mathscr{A}} \{\|\bar{v}_S\|^2\}$
```

the *select* procedure) is associated. Subsequently, once the algorithm has selected the application to be deployed, it updates the computational capacities of the server hosting the module of that application. Afterwards, the algorithm updates the graph structure decreasing the bandwidth of the links that connected the regions selected for the deployment (line 17). It iterates until all applications have been considered.

*Complexity.* Now we look at the complexity of FPA. The procedures *verify*, *updateServer* and *update* have constant time complexity. The procedure *select* computes a vector for each eligible region in the set $\mathscr{A}$. In the worst case, the cardinality of $\mathscr{A}$ is at most $K - 1$. Hence, the complexity of the *select* procedure is $O(K)$. The cardinality of $\mathscr{U}$ is $U$, and the maximum cardinality of a neighborhood 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

First, we describe the setup of the tested scenarios. Where not otherwise specified, we intend the infrastructure owner to maximize the number of deployed applications, i.e., $f_{u,k} \equiv 1$.

*Network topology:* the fog infrastructure is modeled with an undirected network graph with a fixed number of fog regions $K = 7$. The central cloud and regions form a star topology of cloud-to-fog connections, namely cloud-links. For every topology realization, crosslinks among regions are added according to an Erdős-Rényi random graph model, where a link exists between two regions with probability $q$. Finally,

Table II: Distribution of the application requirements of CPU, memory, storage and throughput.

| *Requirement* | *Mean Value* ($u_0$) | *Range* ($u \in \mathscr{U}$) |
|---|---|---|
| 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 |
| Low throughput ($\Delta_u^L$) | 1.5 Mbps | $[1, 2]$ Mbps |
| High throughput ($\Delta_u^H$) | 4.25 Mbps | $[3.5, 5]$ Mbps |

Table III: 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 |

each link in the resulting network is assigned a bandwidth of 15 Mbps, both for cloud-links and crosslinks.

*Application Batch Generation:* a batch of fog applications is generated for each experiment; we considered $U = \{10, 50, 100, 150\}$. The demands of each application of the batch for CPU, storage, memory and throughput are uniform independent random variables. The mean value of such variables is dictated by the nominal value we measured for our benchmark application. That application, as recalled before, is a plate-recognition application packaged as a two-modules microservice. The fog microservice module can process the video stream either in the cloud or on a fog node. The resulting distribution values for the application batches are enlisted in Tab. II; symbol $u_0$ refers to the nominal values we measured on FogAtlas for the plate recognition app.

Finally, the probability that an application belongs to region $k \in \{1, \dots, K\}$ follows a truncated Pareto distribution of parameter $\alpha$, 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 $\gamma = \sum_{h=1}^{K} h^{-\alpha}$.

*Fog Server Classes:* the servers available within each region belong to three classes, depending on the resources they are equipped with, namely *low*, *medium* and *high* class. The computational characteristics are listed in Table III. The number of servers per region is determined per realization 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 $(1 + \beta)\frac{U}{K}\mathbf{c}_{\mathbf{u_0}}$. The parameter $\beta$ is a slack parameter tuning the probability that fog resources are underprovisioned/overprovisioned compared to the aggregated demand. Finally, the servers' population of the tagged region is determined by allocating servers of random type until the region resource budget is exhausted.

Experiments have been conducted on an Ubuntu Linux server with 32 core AMD Opteron(tm) 1.4GHz CPU and 64GB of memory.

## A. Reference Algorithms

In order to make a comparison with other state-of-the-art deployment strategies, we have implemented two variations of the basic Kubernetes scheduling algorithm [22]. Kubernetes' scheduling logic is composed of three main steps [23]:

- *Filtering*: this phase selects a set of admissible servers that can host the application module; this is a feasibility control step common to our FPA algorithm.
- *Ranking*: once a list of possible servers has been established, the second step is to choose the best server for the given application module. There are several existing metrics to produce the rank of servers based on the use of Kubernetes built-in priority functions. After checking the existing options, we decided that, in a fog setting, the most fair comparison with our solution is indeed provided by the *LeastRequestedPriority* function. In this case, in fact, a server node is ranked based on the fraction of the node resources that would be free after the deployment of the application module.
- *Deployment*: once all the servers have been ranked, the best node with respect to the selected priority function is chosen and the application module is deployed on the selected server. Also, the network graph is updated consequently.

We observe that, despite being designed as native cloud solution, Kubernetes offers also a network-agnostic, ready-to-use solution for Fog computing. Nevertheless, it requires some adaptation in order to handle a multi-region scenario. In our numerical evaluation we have considered two main variants:

1) 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 modules to be deployed. We observe that in this approach, only deployments of type 1 and type 2 are possible.
2) 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 module across all fog regions.

In both cases, fairness considerations suggest to assume that the basic container orchestration is able to account also for network metrics[1]. In fact, the baseline feasibility check requires to assess the bandwidth availability between either two neighboring fog regions or between the cloud and the region where the target server lies.

## B. Experimental Results

In Fig. 3a and b we have depicted the number of deployed applications at the increase of the batch size. The graphs report the results averaged on 30 instances with 95% confidence interval for two scenarios with parameters $\beta = -0.2, q = 1/3$ and $\beta = 0.5, q = 1/2$, respectively. The red line represents the optimal solution obtained by the Gurobi solver [24], the blue line FPA, whereas the green and the magenta ones represent the first (Kub) and the second (Kub1) variant of the Kubernetes algorithms, respectively. In the second scenario all algorithms

[1]This feature has been actually implemented in our Kubernetes-based fog platform [6, 7]; the details of the implementation are out of the scope of the current work.

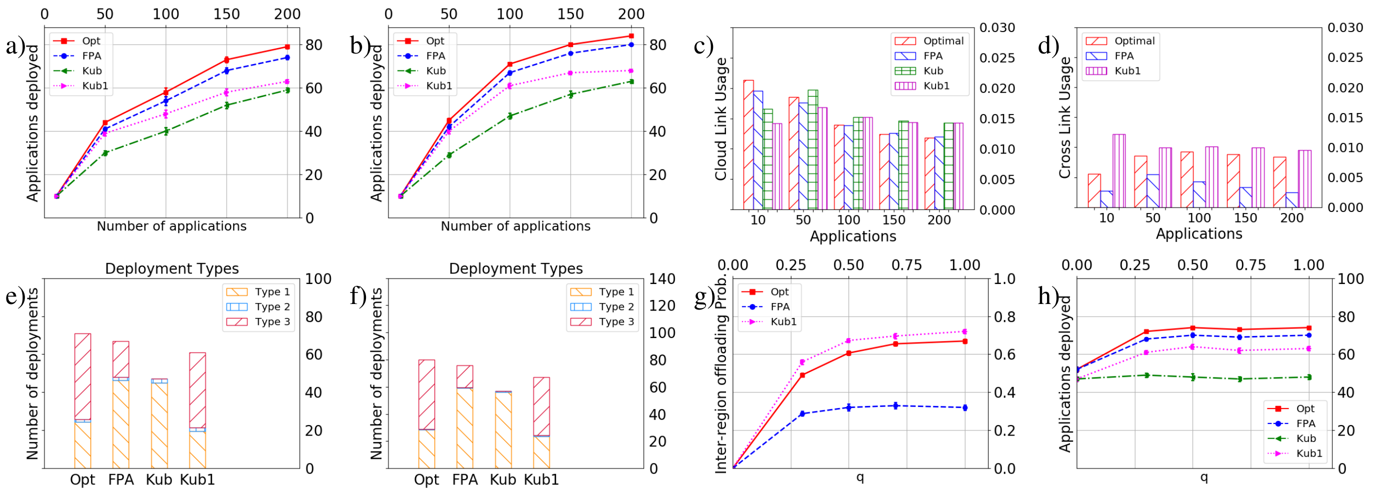

Figure 3: a/b) Number of deployed applications: a) $q = 1/3$, $\beta = -0.2$; b) $q = 1/2$, $\beta = 0.5$; c/d) Average link usage (settings as in b): c) cloud-links and d) crosslinks; e) Configuration types distribution for a typical solution instance with $U = 100$, $q = 0.5$ and $\beta = 0.5$; f) Configuration types distribution for a typical solution instance with $U = 150$, $q = 0.5$ and $\beta = 0.5$; g) Probability of inter-region offloading varying $q$, $U = 100$; h) Number of deployed applications varying $q$, $U = 100$;

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 gradient. In Kub1, conversely, applications are deployed in a predefined order. Second, because Kub1 neglects crosslinks bandwidth utilization, it experiences faster bandwidth resources consumption. On the other hand, FPA's better performance is due to the fact that it accounts for both the bandwidth occupation of both cloud-links and crosslinks.

Fig. 3c reports the average cloud-link usage per deployed application. As the number of applications increases, the average bandwidth consumption decreases for both OPT and FPA. In fact, they both tend to deploy applications on the fog regions thus saving cloud-link bandwidth; the figure shows that FPA behaves similarly to OPT in terms of cloud-links bandwidth consumption. Furthermore, Fig. 3d shows the average cross-link usage by each deployed application. We observe that the optimal solution makes relatively larger usage of crosslinks compared to FPA; Kub1 on the contrary makes intense usage of cross-link bandwidth.

Fig. 3e and Fig. 3f provide some further insight into the structure of the produced solutions. There, we have reported the number of deployments of each type produced by different algorithms. 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 (Fig. 3f).

Fig. 3g and Fig. 3h highlight the effect of inter-area connectivity. We observe that the number of applications deployed increases sharply when $q = 1/3$. Starting at this level of connectivity, in fact, links between crowded hot-spot fog regions and lightly loaded ones become more probable. Inferior performance of Kubernetes placement algorithms is expected, since they do not account for communication capacity in order to perform deployment decisions. Thus, even though Kub1 solutions resort heavily to type 3 configurations, the total number of deployed applications performance is much lower than FPA.

Finally, a direct inspection of the structure of the OPT solutions has revealed not intuitive deployment choices, as, for example, the occurrence of simultaneous offloading between two crowded fog regions, a behaviour which is not reproduced neither by Kub1 or FPA. In fact, the gain obtained by FPA is due to the ability to able to offloads the applications from crowded regions to the lightly-loaded ones, accounting for communication bottlenecks as well as utilisation of computation resources. This causes major bandwidth savings as the number of applications increases.

FPA has been integrated into the orchestrator of FogAtlas [7]. FogAtlas is a fog platform derived from several extensions of the early platform described in [6]. It handles microservice deployment and workload placement over a distributed fog infrastructure encompassing a cloud region and one or more fog regions. Actually, the orchestration mechanism is adapted to a region-oriented architecture: this is a key difference with respect to existing technologies for resources

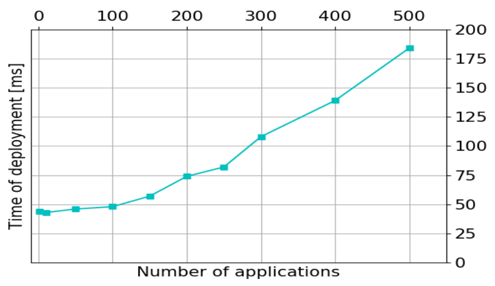

Figure 4: Orchestration delay

orchestration in a centralized-cloud environment. FogAtlas handles the orchestration among regions, and delegates intra-region orchestration to standard OpenStack [25] and Kubernetes [22] controllers. Also, it needs system-wide monitoring and inventory control both at the level of computing and of network resources (the interested reader can refer to [7] for further details).

We aim at ensuring that FPA does not impact severely the deployment time under current standard technology. Hence, we decided to test the scalability of the FPA-based orchestration via our FogAtlas platform. To this aim, Fig. 4 reports on tests performed on the orchestration delay, defined as the time needed from the instant when the batch of applications is offered to the orchestrator until the placement is calculated. As we can see, the expected time complexity is moderately super-linear, confirming the scalability of FPA to large application batch sizes.

## VII. Conclusions

In this paper, we have introduced an optimization framework for microservice fog applications. Different configurations are used to deploy fog computation modules either to the edge or in cloud. The problem combines a multi-commodity flow and a placement problem, but can be reduced to an NP-hard problem, proved by reduction from the multidimensional knapsack problem, by introducing throughput proportionality. We have proposed a greedy algorithm, namely FPA, which performs efficiently with respect to the optimal solution by placing applications using a gradient approach.

We have tested numerically our framework under a realistic dimensioning derived from our FogAtlas platform. The numerical experiments confirm the scalability properties of the proposed fog orchestration technique. Furthermore, we have inspected the resulting crosslink and cloud-link utilization patterns. We have observed that, while cloud-link capacity represents a real bottleneck for the applications' throughput, inter-region orchestration is key to save cloud-link bandwidth. This is especially true in hot-spot scenarios with unbalanced distribution of target objects for fog applications. Nevertheless, inter-region offloading must be performed taking into account the bandwidth consumption of both cloud-links and crosslinks to avoid bandwidth bottlenecks possibly impairing the applications' throughput. In future works, we shall explore the impact of availability requirements, which at present have not been accounted for in the orchestration process.

## VIII. Acknowledgements

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), and from the EU ROMA project under grant agreement 754514.

## References

[1] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497 – 1516, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870512000674

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[3] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, Dec 2016.

[4] G. Li, J. Wu, J. Li, K. Wang, and T. Ye, "Service popularity-based smart resources partitioning for fog computing-enabled industrial Internet of Things," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2018.

[5] Y. Guan, J. Shao, G. Wei, and M. Xie, "Data security and privacy in fog computing," *IEEE Network*, pp. 1–6, 2018.

[6] 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. of IEEE CloudCom*, Dec 2017, pp. 231–234.

[7] FogAtlas. https://fogatlas.fbk.eu/.

[8] R. Yu, G. Xue, and X. Zhang, "Application provisioning in Fog Computing-enabled Internet-of-Things: a network perspective," in *Proc. of INFOCOM*, 2018.

[9] M. Taneja and A. Davy, "Resource aware placement of iot application modules in fog-cloud computing paradigm," in *Proc. of IFIP/IEEE IM*, 2017, pp. 1222–1228.

[10] Y. Gan and C. Delimitrou, "The architectural implications of cloud microservices," *IEEE Computer Architecture Letters*, vol. 17, no. 2, pp. 155–158, July 2018.

[11] L. Canzian and M. V. D. Schaar, "Real-time stream mining: online knowledge extraction using classifier networks," *IEEE Network*, vol. 29, no. 5, pp. 10–16, Sept. 2015.

[12] F. Faticanti, F. De Pellegrini, D. Siracusa, D. Santoro, and S. Cretti, "Cutting throughput on the edge: App-aware placement in fog computing," *arXiv preprint arXiv:1810.04442*, 2018.

[13] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Almost optimal virtual machine placement for traffic intense data centers," in *Proc. of IEEE INFOCOM*, 2013, pp. 355–359.

[14] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *Proc. of IEEE INFOCOM*, vol. 12, 2012, pp. 2876–2880.

[15] N. K. Giang, M. Blackstock, R. Lea, and V. C. Leung, "Developing iot applications in the fog: a distributed dataflow approach," in *Internet of Things (IOT), 2015 5th International Conference on the*. IEEE, 2015, pp. 155–162.

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

[17] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual network embedding through topology-aware node ranking," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 2, pp. 38–47, 2011.

[18] A. Brogi, S. Forti, and A. Ibrahim, "How to best deploy your Fog applications, probably," in *Proc. of IEEE ICFEC*, 2017, pp. 105–114.

[19] T. Elgamal, A. Sandur, P. Nguyen, K. Nahrstedt, and G. Agha, "Droplet: Distributed operator placement for iot applications spanning edge and cloud resources," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 1–8.

[20] H.-J. Hong, P.-H. Tsai, and C.-H. Hsu, "Dynamic module deployment in a fog computing platform," in *Network Operations and Management Symposium (APNOMS), 2016 18th Asia-Pacific*. IEEE, 2016, pp. 1–6.

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

[22] Kubernetes. http://kubernetes.io/.

[23] Kubernetes scheduler. https://github.com/kubernetes/community/blob/master/contributors/devel/scheduler.md.

[24] Gurobi Optimization, LLC, "Gurobi optimizer reference manual," 2018. [Online]. Available: http://www.gurobi.com

[25] Openstack placement API service. https://docs.openstack.org/nova/latest/user/placement.html.