A Rank-based Mechanism for Service Placement in the Fog

Karima Velasquez, David Perez Abreu, Luís Paquete, Marilia Curado, and Edmundo Monteiro
Centre for Informatics and Systems
Department of Informatics Engineering
University of Coimbra
Polo II, Pinhal de Marrocos, Coimbra 3030-290, Portugal
Email: {kcastro,dabreu,paquete,marilia,edmundo}@dei.uc.pt

***Abstract*—As communications evolve to give space to new applications, such as augmented reality and virtual reality, new paradigms arise to provide essential characteristics like lower latency levels, mobility support, and location awareness. Such is the case of Fog computing, which extends from the well-known Cloud computing paradigm by bringing processing, communications, and storage capabilities to the edge of the network. By offering these novel features, also new challenges emerge that call for the design and implementation of orchestration mechanisms to deal with resource management. One of these mechanisms is related to the service placement, which consists in the selection of the appropriate execution node for the applications according to a specific optimization objective. In this paper, an Integer Linear Programming model for service placement aimed at latency reduction of popular applications is proposed. Furthermore, a heuristic based on the PageRank algorithm, called Popularity Ranked Placement, is also introduced. Simulation results show that the heuristic has lower execution times and is able to better balance the load in the network nodes, while being close to the ILP-based solution latency levels.**

I. Introduction

The global IP traffic is expected to triple by 2022 [1], and 82% of this traffic will correspond to IP video traffic. Virtual Reality and Augmented Reality are predicted to increase their traffic 12 times, and Internet video-to-TV will increase three times [1]. With this massive amount of data, a rapid exhaustion of the infrastructure resources, and, therefore, a decrease in the quality of the communications is foreseeable. Given the predominance of streaming video in the predictions of the IP traffic, latency becomes one key aspect to consider to meet delivery constraints of applications.

To improve latency levels it is necessary to design and implement smart service placement mechanisms that select the optimal location among the different possibilities in order to enhance the QoS (Quality of Service).

The decentralization of the Cloud, by bringing the services and applications towards the end-user IoT (Internet of Things) devices and near-user edge devices in order to provide lower latency levels, mobility support, and location awareness, led to the advances in newer paradigms such as Fog computing [2]. Fog computing extends the Cloud paradigm, being located between the dense IoT environment and the Cloud. The Fog nodes (e.g., gateways, switches, servers) can be organized into clusters/communities, supporting federation [3], providing a higher level of organization in a complex and dense environment.

Given its distributed and heterogeneous nature, the Fog requires new mechanisms to automate the management of the resources. The design of a hierarchical architecture that incorporates a Cloud-based repository for the services and a Fog-based deployment was already proposed [4][5]. In such solutions, an orchestrator is assigned with the task of managing the resources of the network efficiently. Nevertheless, in more decentralized scenarios, like the Fog, more complex managing approaches are required. Within the many tasks of the orchestrator, there is a set of *Planning Mechanisms* that must be implemented, including those related to *Service Placement*.

Only bringing the service instances to the edge of the network is not enough, since the perimeter of the Cloud can be broad, and in a dense environment such as the Fog, there could be multiple choices to place the service instances. Thus, it is relevant to select a metric that allows to guide the placement process. The popularity of the applications (and ultimately, of the services) seems to be a good option since it would lead the placement of the most popular applications towards the locations where the most data and service-hungry users are stationed, i.e., at the edge of the network. The popularity can be measured by the number of requests for a given application.

On the other hand, popularity as a metric is not enough to guarantee the QoS requirements for the applications at the communication infrastructure level, since it does not take into consideration the current status of the network. Thus, combining the popularity of the applications with a network metric, such as the propagation delay, will lead to a context-aware orchestrator regarding the service placement.

This paper presents an ILP (Integer Linear Programming) formulation to find the optimal solution for service placement based on popularity aimed at the reduction of latency in Fog environments. Furthermore, a heuristic called PRP (Popularity Ranked Placement) based on the PageRank algorithm [14] is proposed to offer an option close in quality to the optimal solution, while being significantly less time-consuming. Both approaches are evaluated via simulation.

The paper is structured as follows. Section II presents a

ISBN 978-3-903176-28-7

TABLE I
CHARACTERISTICS FROM RELATED WORK

Work	Optimization Goal	Approach	Environment	Simulator
Skarlat et al. [6]	Maximize resource usage	ILP	Fog	iFogSim
Skarlat et al. [7]	Maximize resource usage	ILP + Genetic algorithm	Fog	iFogSim
Wang et al. [8]	Minimize cost	Linear programming + Heuristic	Mobile micro-clouds	*Unspecified*
Taneja and Davy [9]	Maximize resource utilization	Module mapping algorithm	Cloud/Fog	iFogSim
Lera et al. [10]	Maximize availability	2-phase placement / communities	Fog	YAFS
Guerrero et al. [11]	Minimize distance	Decentralized placement	Fog	iFogSim
Mahmud et al. [12]	Maximize QoE	Fuzzy logic	Fog	iFogSim
Brogi and Forti [13]	Maximize QoS	Mathematical model	Fog	FogTorch

selection of related work on service placement. Section III depicts the ILP model regarding service placement for latency reduction. Section IV defines a heuristic based on the PageRank algorithm. Section V depicts the evaluation process and setup, while Section VI presents the results and their respective analysis. Finally, Section VII concludes the paper and shows the road-map for future work.

II. RELATED WORK

Although the placement problem has been vastly addressed in the past for Cloud environments [15][16][17][18], new strategies are required for the Cloud/Fog continuum [19].

Skarlat et al. [6] analyze the placement of IoT services in the Fog, according to their QoS requirements. They define an ILP model aimed at maximizing the utilization of the Fog landscape, while also keeping the resource usage constraints. Evaluation is carried out using iFogSim [20]. On another work, Skarlat et al. [7] present a heuristic based on a genetic algorithm to solve the service placement problem. The main purpose of this work is to maximize the number of services placed, and the usage of Fog devices. Once again, iFogSim is used as an evaluation tool. The proposed solutions are compared with a First Fit greedy approach. The genetic algorithm provided lower deployment delay by exploiting more Cloud resources.

Wang et al. [8] introduce an optimization approach and a heuristic for online scenarios. Both approaches are evaluated using simulations. Their proposal is $O(1)$ competitive for a broad family of cost functions, meaning its competitive ratio is given by a constant. Taneja and Davy [9] describe a Module Mapping algorithm aimed at the optimization of resource utilization for Cloud/Fog scenarios. The solution is compared with placing the applications entirely in the Cloud, and it is carried out by simulation using iFogSim. Response times and network usage are reduced when using the Fog compared with a Cloud-only approach.

Lera et al. [10] present a solution for service placement aimed at improving the availability by placing as many interrelated services as possible within the proximity of the user. The proposal is compared with an ILP approach by means of simulation using YAFS [21], and showed improved QoS and availability. Guerrero et al. [11] propose to place popular services closer (according to the hop-count) to the users. The decision is made in a decentralized fashion by each of the devices. Simulation (using iFogSim) showed that more popular applications had lower latency while less popular applications were affected by a larger delay.

Mahmud et al. [12] use fuzzy logic to place applications in the Fog, with the objective of maximizing the QoE (Quality of Experience). Experimental evaluation is performed using iFogSim. Results show a reduced deployment time and improved QoE. Brogi and Forti [13] propose a model for the deployment of IoT applications in the Fog. Authors also introduce a tool, FogTorch, in which the model is prototyped.

The works reviewed in this section are summarized in Table I. It is noticeable that all works use simulation for the validation, varying the tool used (mostly iFogSim, but also YAFS and FogTorch). It is also relevant to notice the use of mathematical programming techniques, like ILP, in the design of models to find the optimal location for services.

The work presented in this paper uses two metrics to guide the service placement process: (1) the number of requests of the applications, as an upper-level metric to measure the popularity of the applications, and (2) the propagation delay as a network-level indicator of the status of the communication links. Both metrics can change during time, adjusting themselves to reflect the current conditions of the network and the users' demands, unlike other metrics used in related works, such as hop count. Thus, the proposed mechanisms can be executed in different time windows to re-adjust the placement of the existing services and to satisfy the new users' demands. The service placement mechanisms proposed are described in the following sections.

III. AN ILP MODEL FOR SERVICE PLACEMENT

This section introduces an ILP model to maximize the placement of popular applications while minimizing the latency for final users in Fog environments.

A lexicographic formulation of the optimization problem is considered in this article. First, with the goal of serving the largest number of users, the problem consists of maximizing the selection of accepted requests, given the popularity of the applications. Then, the second problem consists of placing the services that belong to an application in the Fog nodes that

minimize latency, given the selection of the requests obtained in the first problem. The constraints from the first problem are kept for the second in order to ensure feasibility. The goal is to improve the QoS; hence the overall solution of solving both problems should provide a better QoS to the maximum amount of users.

A. Parameters and Variables

Table II summarizes the parameters and variables for the model. Regarding the parameters, the set of entry points from where requests are generated is labeled as GW; these correspond to the gateways from where clients access the Fog environment. The instance matrix I reflects the micro-services that compose the applications. This means that an application can be built by the combination of a set of services. $I_{a,s} = 1$ if service $s \in S$ belongs to application $a \in A$, and 0 otherwise. The cost matrix C contains the propagation delay (as a component of latency) from the shortest path that connects each pair of nodes $n \in N$ and gateways $gw \in GW$.

Concerning the variables, the acceptance matrix, K is a binary matrix that indicates which requests are accepted for each application. $K_{a,r} = 1$ if the *r-th* request for application $a \in A$ is accepted, and 0 otherwise. P represents the placement matrix, indicating the final location for the services. The matrix relates the request per applications, and the nodes where the services belonging to those applications are finally placed; $P^{a,r}_{s,n} = 1$ indicates that service $s \in S$ is executed on node $n \in N$ to satisfy request $r \in R$ for application $a \in A$, and $P^{a,r}_{s,n} = 0$ otherwise.

B. Maximizing the Placement of Popular Applications

The first step in the optimization solution is to select the applications with the highest amount of requests, which are considered as the most popular. Eq. (1) depicts the main objective function of the ILP model. Since the main objective is to maximize the selection of popular applications, the set of requests per application is used to determine which applications have higher priority in the selection.

$$\max \sum_{a \in A} Q_a \times \sum_{r \in R} K_{a,r} \tag{1}$$

The first constraint in the model, shown in Equation (2), is meant to ensure that the selection of applications is only carried out when the request can be fulfilled entirely. The constraint in Eq. (3) guarantees that the services that belong to an application selected are executed in only one server. Eq. (4) forces that all the services belonging to an application can be executed before selecting it.

$$P^{a,r}_{s,n} \quad \leq \quad K_{a,r} \quad \forall a \in A, r \in R, s \in S, n \in N \tag{2}$$

$$\sum_{n \in N} P^{a,r}_{s,n} \quad \leq \quad 1 \quad \forall a \in A, r \in R, s \in S \tag{3}$$

TABLE II
PARAMETERS AND VARIABLES FOR THE ILP MODEL

Parameters	
Parameter	**Description**
S	Set of services to be placed
N	Set of nodes where the service can be executed
GW	Set of gateways at the edge of the Fog
A	Set of applications. An application is composed by a set of services
R	Set of requests for all the applications
Q_a	Sum of requests for $a \in A$
Ω_n	CPU capacity for $n \in N$ (GHz)
Φ_n	Memory capacity for $n \in N$ (GB)
ω_s	CPU requirement for $s \in S$ (GHz)
φ_s	Memory requirement for $s \in S$ (GB)
I	Instance matrix. An $\|A\| \times \|S\|$ matrix
C	Cost matrix. An $\|N\| \times \|GW\|$ matrix
Variables	
Variable	**Description**
K	Acceptance matrix. An $\|A\| \times \|R\|$ matrix
P	Placement matrix. An $\|A\| \times \|R\| \times \|S\| \times \|N\|$ matrix

$$K_{a,r} \times I_{a,s} \quad = \quad \sum_{n \in N} P^{a,r}_{s,n} \quad \forall a \in A, r \in R, s \in S \tag{4}$$

Memory and CPU restrictions are also imposed as constraints to the model, to enforce resource limits in the execution nodes. Eq. (5) describes the CPU constraint and Eq. (6) does the same for the memory constraint. For both equations, the sum of processing requirements from all services can not surpass the available resources in the nodes.

$$\sum_{a \in A} \sum_{r \in R} \sum_{s \in S} P^{a,r}_{s,n} \times \omega_s \quad \leq \quad \Omega_n \quad \forall n \in N \tag{5}$$

$$\sum_{a \in A} \sum_{r \in R} \sum_{s \in S} P^{a,r}_{s,n} \times \varphi_s \quad \leq \quad \Phi_n \quad \forall n \in N \tag{6}$$

While the resources of the nodes are usually fixed and well known, an estimate is used for the resource requirements of the services (i.e., CPU and memory). Service profiles can be created by gathering information from previous executions. In this work, the services profiles are generated using a bounded random approach (see Section V).

C. Minimizing the Latency

The second optimization goal is to reduce the latency of the most popular applications, selected on the first optimization problem. Eq. (7) depicts the formulation. The main idea is to minimize cost, represented by the propagation delay of the links towards a node ($C_{n,gw}$). The cost can be changed to another gauge, e.g., energy consumption, to model a different requirement.

$$\min \sum_{a \in A} \sum_{r \in R} \sum_{n \in N} \sum_{s \in S} \sum_{gw \in GW} P^{a,r}_{s,n} \times C_{n,gw} \tag{7}$$

In order to guarantee that the popular applications are still selected (i.e., maintaining the results from the first optimization), the acceptance matrix resulting from the first optimization problem is now a parameter matrix and used as input for the placement process in the second optimization problem. Therefore, during this step of the optimization, the services are placed in the most convenient nodes in the Fog, aiming at the reduction of the latency of the services; see Eq. (8).

$$\sum_{s \in S} K_{a,r} \times I_{a,s} \quad = \quad \sum_{s \in S} P_{s,n}^{a,r} \quad \forall a \in A, r \in R, n \in N \tag{8}$$

Constraints from the first optimization problem (i.e., Eq. (2) through (6)) are kept in this step to guarantee feasibility. In the following section, an alternative heuristic based on the PageRank algorithm is proposed.

IV. A Heuristic based on PageRank

Although optimization solutions are often used for offline environments and as a theoretical threshold, they are not usually applied in online or more realistic scenarios given their lack of adaptability and also their high response times [22]. Furthermore, by optimizing the selection of requests that come from the gateways, the procedure will most likely place the majority of the applications in the same Fog nodes, thus potentially creating an overload in the same nodes and links in case of heavy load. To overcome these issues, an alternative heuristic, based on the PageRank algorithm, is presented in this section.

First, the original PageRank algorithm is introduced to, later on, describe how it is adapted for the PRP (Popularity Ranked Placement) heuristic.

A. The PageRank Algorithm

The PageRank algorithm was first introduced to rank web pages in a search engine, and it was based on a summation derived from bibliometrics research (i.e., the analysis of the citation structure among academic papers) [14]. The idea behind the PageRank algorithm is to rank the nodes in a graph via probability propagation. To understand how it works, an example is provided in Fig. 1. Consider the graph $G = (V, E)$, where $V = \{A, B, C, D\}$ and $E = \{(A, B), (A, C), (B, D), (C, A), (C, B), (C, D), (D, C)\}$. At the beginning of the algorithm, it is assumed that all the nodes in set V have the same rank $r = \frac{1}{n}$, where n denotes the cardinality of set V.

Nodes are ranked following an iterative approach, according to Eq. (9) where B_{n_i} is the set of nodes pointing into node n_i and $deg^+(n_j)$ is the number of out-links from node n_j. Let $r_{k+1}(n_i)$ be the rank of node n_i at iteration $k+1$. The ranking process begins with $r_0(n_i) = \frac{1}{n}$ for all nodes and is repeated until the PageRank scores converge to final stable values [23]. The values obtained after applying the first three iterations of Eq. (9) to the graph depicted in Fig. 1 are shown in Table III.

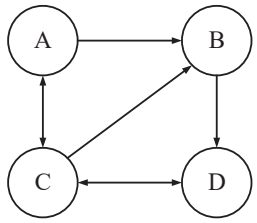

Fig. 1. A graph example to illustrate the PageRank algorithm

$$r_{k+1}(n_i) \quad = \quad \sum_{n_j \in B_{n_i}} \frac{r_k(n_j)}{deg^+(n_j)} \tag{9}$$

The implementation of the PageRank algorithm could be sped-up using matrices [23]. The idea is to use an $N \times N$ matrix H in combination with a $1 \times N$ row vector π^T, which holds the PageRank value of all nodes in each iteration. The matrix H is a row normalized hyperlink matrix with $H_{ij} = \frac{1}{deg^+(n_j)}$ if there is a link from node n_i to node n_j, and 0, otherwise. Thus, nonzero elements in H represent the transition probability from one node to another. The corresponding H matrix for the graph depicted in Fig. 1 is shown in Eq. (10).

$$H \quad = \quad \begin{pmatrix} 0 & 0 & \frac{1}{3} & 0 \\ \frac{1}{2} & 0 & \frac{1}{3} & 0 \\ \frac{1}{2} & 0 & 0 & 1 \\ 0 & 1 & \frac{1}{3} & 0 \end{pmatrix} \tag{10}$$

The nonzero elements of row i in H correspond to the out-link nodes of node i; meanwhile, the nonzero elements of column i denote the in-link nodes of node i. Regarding the rank of the nodes, the row vector $\pi^{(k)T}$ represents the PageRank vector at the *k-th* iteration of the algorithm. Consequently, Eq. (9) can be written using a matrix notation as shown in Eq. (11).

$$\pi^{(k+1)T} \quad = \quad \pi^{(k)T} H \tag{11}$$

From Eq. (11), the following observations arise [23]: (1) each iteration requires one vector-matrix multiplication, considering that H is a very sparse matrix, the vector-matrix operation is reduced to $O(n)$ computational effort; (2) the iterative method used is a linear stationary process performed applying the power method to matrix H; and (3) H could be seen as a stochastic transition probability for a *Markov* chain, where the dangling nodes of the graph create 0 rows in the matrix and the other rows belong to the non-dangling node keeping stochastic values. Consequently, H is considered a sub-stochastic matrix.

In this work, the PageRank algorithm described above is adapted to rank the nodes in a network topology using the PD (Propagation Delay) of the links to weight the probability transition between nodes. A heuristic based on this ranking process is described below.

TABLE III
PAGERANK VALUES PER ITERATIONS

Iteration 0	Iteration 1	Iteration 2	PageRank
$r_0(A) = \frac{1}{4}$	$r_1(A) = \frac{1}{12}$	$r_2(A) = \frac{1.5}{12}$	1
$r_0(B) = \frac{1}{4}$	$r_1(B) = \frac{2.5}{12}$	$r_2(B) = \frac{2}{12}$	2
$r_0(C) = \frac{1}{4}$	$r_1(C) = \frac{4.5}{12}$	$r_2(A) = \frac{4.5}{12}$	4
$r_0(D) = \frac{1}{4}$	$r_1(D) = \frac{4}{12}$	$r_2(A) = \frac{4}{12}$	3

B. Ranking nodes to create Communities

To avoid the potential issue concerning the overload of the nodes close to the gateways and the gateways themselves, an alternative is to create *communities* that can share the load of the gateways while keeping the applications deployed in their proximity, as explored in other works [7][10]. However, only selecting neighboring nodes to share the load could be restrictive since there is a chance that a non-neighbor is better qualified than the neighbors (e.g., has lower latency connectivity). Thus, ranking the nodes according to the probability in which they would communicate (e.g., be chosen in the shortest path) represents a better selection criterion to build these sharing groups or communities. In this work, the ranking process described above is used to create those communities.

The heuristic takes advantage of the low latency levels in the Fog, by deploying applications' modules through the Fog infrastructure and near to the final users whenever possible. When the Fog runs out of resources, deployment in the Cloud will be carried out. The ranking of the nodes is created according to the procedure described in Subsection IV-A. For a weighted graph, the probability of moving from one node to the next will not only depend on the presence of a link but on a metric based on the weights of the edges. In this work, the PD is used as a metric of the link latency. Thus, nodes connected through a link with a lower propagation delay will have a higher probability to connect since they will be part of the shortest path towards the destination node.

The community-building process is described in Algorithm 1. The first step is to get the topology graph (line 1), and to remove the Cloud node since it will not be part of any community (i.e., only being used when there are no resources available in the Fog), and also transforming it as a complete directed graph. Following this, the nodes are ranked using the process described in Subsection IV-A. The PD is used to weight the transition probability from one node to another in the graph (line 2).

The communities are built by selecting all the nodes with a transition probability higher than a threshold (line 7). For this work, the threshold was defined in 0.1, to create bigger communities, but the value could be adapted following a different criterion. After this, a community is created by combining this initial community with all the communities of its initial members (line 14).

All the Fog nodes that were not assigned to any community during this process are grouped in the final step, as shown in line 16. The results of this process include: (1) a structure with the communities for all nodes; (2) a structure with the nodes that do not belong to any community; and (3) the average size of the initial communities (see line 17).

Algorithm 1: Build Communities

```
   Result: Communities for all nodes in the infrastructure
1  topology ← getTopology()
2  prank ← PageRank(topology, weight=PD)
3  communities ← ∅
4  non_communities ← ∅
5  foreach node in topology do
6      foreach element in prank do
7          if prank[element] ⩾ threshold then
8              Add element to communities[node];
9          end
10     end
11 end
12 avg_size ← getCommunitySize(communities)
13 foreach node in topology do
14     mergeCommunities(node, communities);
15 end
16 non_communities← prank − communities;
17 return communities, non_communities, avg_size
```

Once the communities are built, the deployment process begins, as explained in the following subsection.

C. Popularity Ranked Placement

The heuristic proposed, called PRP (*Popularity Ranked Placement*) uses the communities described in the previous subsection for the placement process, and is presented in Algorithm 2. After creating the communities (line 3), the applications to deploy are ranked according to their requests (i.e., to prioritize popular applications, as in the ILP model described in Section III), as shown in line 5. Also, the nodes in each community are ranked according to the probabilities calculated in the previous step.

The first option is to deploy the applications' modules within the community of the GW from which the application's request was launched. The search space inside the community is limited by an expanding window, which size is set according to the average size of the initial communities (line 6). The expanding window technique allows to minimize the internal fragmentation of resources inside of the nodes while allowing to place a less demanding application's module in the already explored nodes of the community. The searching process within the windows follows a RR (Round Robin) approach until no more resources are available to host the application's module. In the case that the deployment succeeds, the RR pointer is updated accordingly.

When the community window runs out of resources to deploy the application's module, the size of the expanding window is augmented by the average size of the initial communities, until the size of the community is reached, and a new search process to find the hosting node is carried out

Algorithm 2: Popularity Ranked Placement

Result: Placement of applications' modules

```
1  placement_matrix ← ∅
2  topology ← getTopology();
3  community, non_community ← Build Communities()
4  reqs ← getRequests();
5  apps ← rankApps();
6  expWin ← getAvgCommunitySize();
7  foreach req in reqs do
8      while req > max(WindowNodeCapacity) ∧
9      ¬ ReachCommunitySize do
10         Expand expWin;
11     end
12     if req ≤ max(WindowNodeCapacity) then
13         deploy(placement_matrix, community);
14     else
15         if req ≤ max(NonComNodeCapacity) then
16             deploy(placement_matrix, non_community);
17         else
18             deploy(placement_matrix, Cloud);
19         end
20     end
21 end
22 return placement_matrix
```

(lines 8 - 11). The process is also illustrated in Fig. 2. The community for node *A* is shown in Fig. 2(a). The initial window size in this example is 3 (i.e., the average size of the initial communities). Once the subset contained inside the expanded window runs out of resources (there is no node that can host the application's module to deploy), the window's size is augmented by its original size, 3 in this case, as depicted in Fig. 2(b). This process is repeated until finding a node with enough resources to fit the application's module to deploy, or until the community size is reached, as seen in Fig. 2(c).

If the expanding window reaches its maximum size (i.e., the community size) and there are not enough resources to satisfy the request, a new search process is initiated in the non-community nodes (also a resulting structure from the previous step, see Subsection IV-B) following a First Fit approach (lines 15 - 16). Thus, Fog nodes are prioritized for placement before trying to deploy the application's modules in the Cloud. If all previous attempts fail, the application's module is deployed in the Cloud (line 18).

Simulation experiments were performed to validate this proposal. The evaluation setup is described in the next section.

V. Evaluation

The validation was performed using the YAFS simulator [21] because of its strong support of Fog critical features and high granularity of reported results [24]. The experiments were conducted on a PC with 32GB 2400MHz DDR4 RAM and 2.80GHz Intel Core i7-7700HQ with 4 cores and 8 threads (2 threads per core) processor. The PC was running Microsoft Windows 10 Pro (Build 18363) operating system. Python 2.7.16 was used for YAFS. For the ILP model, the IBM CPLEX Optimizer version 12.9 was used [25].

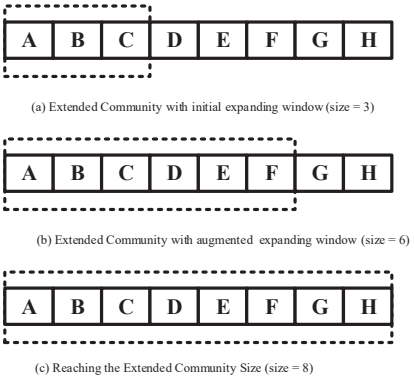

Fig. 2. Expanding Window

Regarding the network topology, a graph was generated according to the complex network theory, following a random Barabasi-Albert network [26]. 50 Fog nodes comprise the network, and an additional node was added to represent the centralized Cloud. This node is connected to the Fog nodes with the highest betweenness centrality in the graph. The nodes with lowest betweenness centrality were appointed as GWs, representing the nodes at the edge of the network.

Applications were randomly generated following a *Growing Network* graph structure. This means a vertice is added one at a time with an edge to the last added vertice. After the graph is complete, two vertices are randomly selected (excluding the source) to generate an information flow to the source vertice. This allows simulating typical Fog/IoT delay-sensitive applications that can be deployed at Cloud/Fog environments. For instance, an augmented reality application that can capture location-related information, process it, and send a reply to the user; or an eHealth application where some medical values are captured from the patient, processed and/or stored for later analysis, and sent to the health specialist for evaluation, or to regulate a patient's medication.

TABLE IV
Parameters values for experiments

	Parameter	Value (min - max)
Network	Propagation Delay (ms)	2 - 10
	Bandwidth (bytes/ms)	75000
Fog	Resources (units)	10 - 25
	Speed (instr/ms)	500 - 1000
GW	Request rate (1/ms)	1/1000 - 1/200
	Popularity (prob)	0.25
Application	Services (number)	2 - 8
	Resources (units)	1 - 5
	Execution (instr/req)	20000 - 60000
	Message size (bytes)	1500000 - 4500000

The parameters were set according to the values displayed in Table IV, for each network link, Fog node, GW, and application. Similar values have been previously used in re-

lated work [10]. The application demands are measured using the YAFS' resources unit, defined as a vector containing the capacity of different computational resources (e.g., number of cores for CPU, GB for memory, or TB for the hard disk).

The network load was varied to evaluate the performance of the proposals under different conditions. Four scenarios were defined: (1) *tiny*: 5 different applications, (2) *small*: 10 applications, (3) *medium*: 15 applications, and (4) *large*: 20 applications. Each of the applications has at least one request. To simulate the popularity of the applications, the number of requests was determined using a uniform distribution. All the scenario setup, as well as the source code, is available via a GitLab repository [27].

The ILP solution and the proposed heuristic (PRP) are compared with the well known FF (First Fit) algorithm, as it was used in other works for evaluation purposes [6][7]. For the FF algorithm, the nodes were organized according to their resources, from lowest to highest, to prioritize nodes with less resources that usually are deployed at the edge of the network, closer to the users. All solutions were executed once before the simulations, i.e., a static service placement is considered, and 30 simulations were executed using these static placements to mitigate the statistical error. 95% confidence intervals are included in the plots.

VI. Results and Analysis

The performance of each placement method (ILP model, PRP, and FF) is presented in this section. The first metric to evaluate is the latency. The latency is calculated as the sum of the transmission times among the application's modules [21]. Fig. 3 shows the results, grouped by scenario and displayed in seconds.

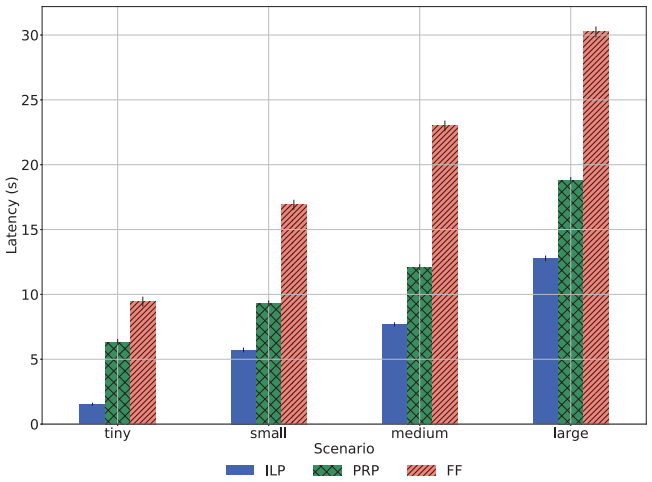

Fig. 3. Total Latency by Scenarios

From Fig. 3, it is possible to note that the best results correspond to the ILP optimization model, as expected. As the load grows, the latency increases, which is also expected. For the smallest load (i.e., *tiny* scenario), PRP shows an exceeding latency of about 3 times the values obtained with the ILP placement, while FF shows an extra of about 5 times regarding the ILP latency. These discrepancies are reduced in higher load scenarios. Nevertheless, while the breach is reduced between ILP and PRP, it is more prominent for FF. For instance, in the *medium* scenario, FF shows an increase of 3 times respecting the ILP approach, while PRP only shows 1.5 times more latency. Grouping the nodes in communities (PRP) showed better results than performing a linear search (FF). As the scenario got more complex, the improvement is more noticeable; in general, the results of PRP are closer to the optimal than FF.

Fig. 4 depicts the network transmission, including the application messages forwarded and the average network buffer occupancy (i.e., the average amount of messages kept in node's network buffers waiting for link availability). From these results, it is important to point out that the ILP method forwarded the lowest amount of messages, thus generating less traffic and, ultimately, less congestion. PRP values remained close to the theoretical optimum (ILP), especially for the smaller scenarios. As the load increases, so does the number of messages forwarded. FF showed to be the least efficient method; the nodes with the lower amount of resources are selected first, saturating the network buffers as the load increases. For the application messages, since FF does not take into consideration the interaction between nodes, modules from the same application can be placed in distant nodes (i.e., several hops); thus, the amount of messages forwarded is larger.

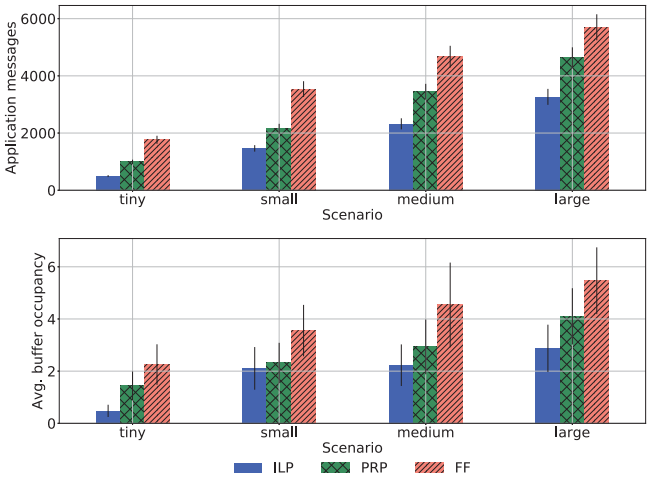

Fig. 4. Network Transmission

Fig. 5 illustrates the number of nodes used by each method. This means the number of nodes where applications' modules were placed; the figure also portraits the number of modules placed on the busiest node; this is the node where more modules were deployed. Since the placement was statically performed before the simulations, there are no variations in the results and thus no confidence intervals shown.

The ILP model led to the concentration of the placement of modules in the nodes closer to the request points (i.e., the GWs), saturating these nodes, and therefore creating a new

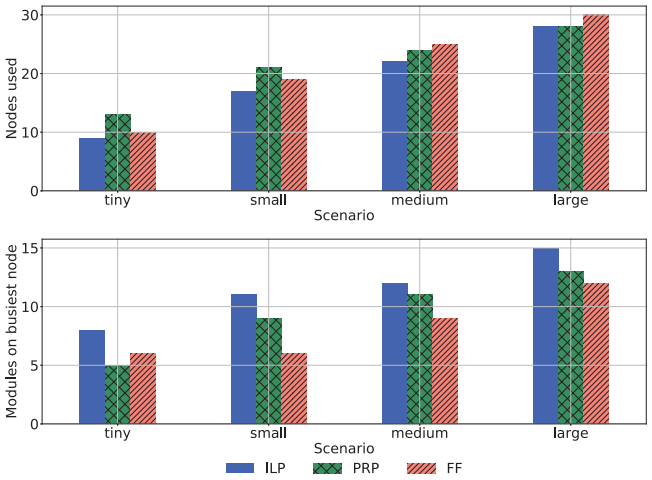

Fig. 5. Module Placement Metrics

issue in the form of congestion of the nodes and the links connected to said nodes. As the load increases, the advantages of the optimization are missed by the congestion created. On the other hand, PRP uses more nodes for the two smallest scenarios, thus having impact on the energy consumption. For the two larger scenarios, the number of nodes used tends to stabilize in all the placement approaches.

Fig. 6 depicts a comparison of the latency obtained by each application considering the amount of requests it has. For space constraints, only the results corresponding to the *large* scenario are displayed; similar results were obtained for the other scenarios.

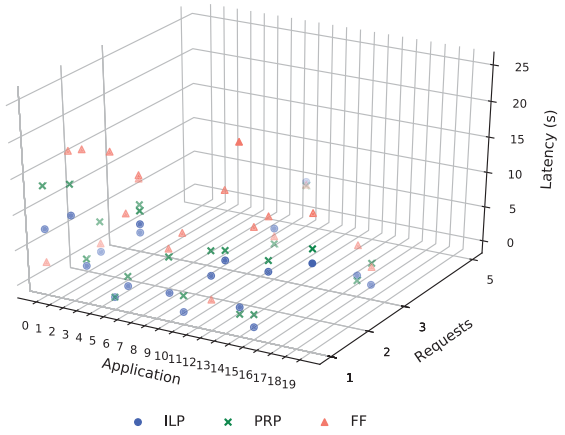

Fig. 6. Latency by Application - Large Scenario

Overall, the ILP model obtained the best results, while FF got the worst. It is noteworthy that as the application has more requests (i.e., more popularity), it shows lower latency for ILP and PRP. Furthermore, ILP and PRP results are relatively close, with ILP showing the lowest latency values. It is clear that there are three tiers regarding the latency response per application, being the lowest the one corresponding to the ILP model, the following to PRP, and the highest to FF.

Finally, Table V shows the execution time, in seconds, for the placement methods, by scenario. Since FF has the most straightforward logic, it has also the lowest execution times, followed closely by PRP. The times for ILP are significantly higher since this method evaluates all possible solutions in order to find the optimal result. The times obtained by the ILP model could not be suited for more complex and dense realistic scenarios.

TABLE V
EXECUTION TIME (IN SECONDS)

Scenario	ILP	PRP	FF
Tiny	19.191	0.017	0.005
Small	47.382	0.019	0.007
Medium	104.983	0.022	0.011
Large	313.588	0.031	0.0019

In general, the ILP approach got the best results regarding latency, but the worst on execution time and in node overload. The latency values obtained with PRP are close to the ones reported by ILP while getting significantly lower execution times. Furthermore, since the PageRank can be calculated dynamically, it is possible to apply this solution in case of variations in the network infrastructure. Finally, by creating communities, the load is spread among different nodes, which could lead to other issues such as higher energy consumption, that are not being considered in this work. This spread also means lower congestion levels in both the Fog nodes and communication links for PRP.

VII. CONCLUSIONS

Applications like augmented reality and virtual reality, as well as video over IP, constitute the more significant portion of IP traffic for the upcoming years. These types of applications have special requirements, including low latency. The Fog computing paradigm arises to overcome some of the new requirements, but also brings new challenges in the form of the need to design and develop novel orchestration mechanisms to take the full advantages that the Fog has to offer; for instance, new provisioning mechanisms, including service placement, have to be developed.

An ILP model for service placement, aimed at maximizing the placement of popular applications, while minimizing their latency is proposed. Moreover, a heuristic based on the PageRank algorithm, called Popularity Ranked Placement, is also introduced. PRP ranks the applications according to their requests as a measure of their popularity, and also ranks the nodes in the network topology to create communities with the nodes with the highest transition probability; this way, the placement load is divided among the nodes within the community, avoiding congestion while also maintaining low latency levels. An expanding window controls the placement among the nodes in the community.

The performance of both the ILP model and the PRP solution were tested using YAFS, and are also compared with the well known FF approach. Simulation results show that while the ILP model had the lowest latency for all the scenarios, it also had the highest concentration of placement in the same nodes, thus generating congestion in the processing nodes and for the communication links. PRP kept the latency at low levels, close to the ILP results, but the load was balanced between the nodes in the communities. Moreover, PRP showed significantly lower execution times than the ILP model, which makes it more suitable for more complex and dense scenarios, and is the option that should be deployed in practical realistic situations.

Future work includes studying the impact of the size of the expanding window and the size of the communities for different scenarios; analyzing its behavior under non-static environments; and combining different optimization objectives besides latency, for instance, applications' availability requirements.

ACKNOWLEDGMENT

Karima Velasquez and David Perez Abreu wish to acknowledge the Portuguese funding institution FCT - Foundation for Science and Technology for supporting their research under the Ph.D. grants SFRH/BD/119392/2016 and SFRH/BD/117538/2016 respectively.

The work presented in this paper was partially carried out in the scope of the projects: ≪MobiWise: From mobile sensing to mobility advising≫ (P2020 SAICTPAC/0011/2015), cofinanced by COMPETE 2020, Portugal 2020 - Operational Program for Competitiveness and Internationalization (POCI), European Union's ERDF (European Regional Development Fund).

This work is funded by national funds through the FCT - Foundation for Science and Technology, I.P., within the scope of the project CISUC - UID/CEC/00326/2020 and by European Social Fund, through the Regional Operational Program Centro 2020

REFERENCES

[1] Cisco, "Cisco Visual Networking Index: Forecast and Trends, 2017–2022," Cisco, White paper, Feb 2019.

[2] L. M. Vaquero and L. Rodero-Merino, "Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, Oct 2014.

[3] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. S. Goren, and C. Mahmoudi, "Fog computing conceptual model," National Institute of Standard and Technology, Tech. Rep. NIST Special Publication 500-325, Mar 2018.

[4] K. Velasquez, D. Perez Abreu, M. Curado, and E. Monteiro, "Service Placement for Latency Reduction in the Internet of Things," *Annals of Telecommunications*, vol. 72, no. 1, pp. 105–115, Feb 2017.

[5] K. Velasquez, D. P. Abreu, D. Gonçalves, L. Bittencourt, M. Curado, E. Monteiro, and E. Madeira, "Service orchestration in fog environments," in *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*. Prague, Czech Republic: IEEE, Aug 2017, pp. 329–336.

[6] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards qos-aware fog service placement," in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*. Madrid, Spain: IEEE, May 2017, pp. 89–96.

[7] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized iot service placement in the fog," *Service Oriented Computing and Applications*, vol. 11, no. 4, pp. 427–443, Dec 2017.

[8] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1002–1016, April 2017.

[9] M. Taneja and A. Davy, "Resource aware placement of iot application modules in fog-cloud computing paradigm," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. Lisbon, Portugal: IEEE, May 2017, pp. 1222–1228.

[10] I. Lera, C. Guerrero, and C. Juiz, "Availability-aware service placement policy in fog computing based on graph partitions," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3641–3651, April 2019.

[11] C. Guerrero, I. Lera, and C. Juiz, "A lightweight decentralized service placement policy for performance optimization in fog computing," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 6, pp. 2435–2452, Jun 2019.

[12] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Quality of experience (qoe)-aware placement of applications in fog computing environments," *Journal of Parallel and Distributed Computing*, vol. 132, no. 1, pp. 190 – 203, Oct 2019.

[13] A. Brogi and S. Forti, "Qos-aware deployment of iot applications through the fog," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1185–1192, Oct 2017.

[14] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep. 1999-66, November 1999.

[15] H. Moens, B. Hanssens, B. Dhoedt, and F. De Turck, "Hierarchical network-aware placement of service oriented applications in clouds," in *IEEE/IFIP NOMS 2014 - IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World*. Krakow, Poland: IEEE, May 2014, pp. 1–8.

[16] D. Espling, L. Larsson, W. Li, J. Tordsson, and E. Elmroth, "Modeling and Placement of Cloud Services with Internal Structure," *IEEE Transactions on Cloud Computing*, vol. 4, no. 4, pp. 429–439, Oct 2016.

[17] R. B. Milad Ghaznavi, Aimal Khan, Nashid Shahriar, Khalid Alsubhi, Reaz Ahmed, "Elastic Virtual Network Function Placement (EVNFP)," in *4th IEEE Conference on Cloud Networking (CloudNet) 2015*, no. Oct. Niagara Falls, Canada: IEEE, Oct 2015, pp. 255–260.

[18] M. Steiner, B. G. Gaglianello, V. Gurbani, V. Hilt, W. Roome, M. Scharf, and T. Voith, "Network-aware service placement in a distributed cloud environment," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. Helsinki, Finland: ACM, Aug 2012, pp. 73–74.

[19] V. Souza, X. Masip-Bruin, E. Marín-Tordera, S. Sànchez-López, J. Garcia, G. Ren, A. Jukan, and A. J. Ferrer, "Towards a proper service placement in combined fog-to-cloud (f2c) architectures," *Future Generation Computer Systems*, vol. 87, no. 1, pp. 1–15, Oct 2018.

[20] 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," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, Jun 2017.

[21] I. Lera, C. Guerrero, and C. Juiz, "Yafs: A simulator for iot scenarios in fog computing," *IEEE Access*, vol. 7, no. 1, pp. 91 745–91 758, Jul 2019.

[22] K. Y. Lee and M. A. El-Sharkawi, *Modern heuristic optimization techniques: theory and applications to power systems*. John Wiley & Sons, Ltd, 2008.

[23] A. N. Langville and C. D. Meyer, *Google's PageRank and beyond: The science of search engine rankings*. Princeton University Press, 2012.

[24] D. P. Abreu, K. Velasquez, M. Curado, and E. Monteiro, "A comparative analysis of simulators for the cloud to fog continuum," *Simulation Modelling Practice and Theory*, p. 102029, Nov 2019.

[25] IBM, "CPLEX Optimizer," https://www.ibm.com/analytics/cplex-optimizer, 2019, accessed: 2019-12-20.

[26] M. Jalili and M. Perc, "Information cascades in complex networks," *Journal of Complex Networks*, vol. 5, no. 5, pp. 665–693, Jul 2017.

[27] K. Velasquez, D. Perez Abreu, L. Paquete, M. Curado, and E. Monteiro, "Rank-based Service Placement - GitLab Repository," https://git.dei.uc.pt/kcastro/rankPlacement.git, 2019, accessed: 2019-12-20.