IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 20, 2021, accepted May 26, 2021, date of publication June 2, 2021, date of current version June 9, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3085370

Service Placement for Latency Reduction in the Fog Using Application Profiles

KARIMA VELASQUEZ, DAVID PEREZ ABREU, MARILIA CURADO, AND EDMUNDO MONTEIRO, (Senior Member, IEEE)

Centre for Informatics and Systems, Department of Informatics Engineering, University of Coimbra, 3030-290 Coimbra, Portugal

Corresponding author: Karima Velasquez (kcastro@dei.uc.pt)

This work was supported in part by the European Regional Development Fund (FEDER), through the Regional Operational Programme of Lisbon (POR LISBOA 2020), in part by the Competitiveness and Internationalization Operational Programme (COMPETE 2020) of the Portugal 2020 Framework through the Project 5G with Nr. 024539 under Grant POCI-01-0247-FEDER-024539, in part by the National Funds through the Foundation for Science and Technology (FCT), I.P., through the Project Centre for Informatics and Systems of the University of Coimbra (CISUC) under Grant UID/CEC/00326/2020, and in part by the European Social Fund through the Regional Operational Program Centro 2020. The work of Karima Velasquez and David Perez Abreu was supported by the Portuguese Funding Institution, Foundation for Science and Technology (FCT), under the Ph.D. Grant SFRH/BD/119392/2016 and Grant SFRH/BD/117538/2016.

ABSTRACT The Cloud-Fog-Internet of Things continuum combines different paradigms to provide connectivity and ubiquity for end-users, while also granting low latency and low jitter to cope with different challenges, including the requirements of latency-sensitive applications, such as virtual/augmented reality and online gaming. This constitutes a complex and dynamic environment with heterogeneous resources that need to be managed or orchestrated, in order to accomplish application requirements for low latency. Common orchestration solutions make placement decisions based only on the resources of the underlying network and the application resource requests; however, using the profiles of applications to make placement decisions has the potential to enhance the final performance perceived by the end-users. This paper proposes the use of application profiles according to their popularity to guide their placement. To corroborate the effectiveness of the use of the profiles, two placement mechanisms are presented, one based on Genetic Algorithms and the other inspired on graph partitions. Simulation results show that it is possible to reduce the latency and jitter of applications via a service placement guided by the profiles. The mechanism based on graph partitions showed better results for all scenarios, followed closely by the Genetic Algorithm in the scenarios with lower load.

INDEX TERMS Popularity, profiles, latency, placement, fog.

I. INTRODUCTION

New applications and services enhance the usability and impact of physical world entities by the use of digital systems and infrastructures, like the Internet of Things (IoT). The fusion of the physical world with the digital one enables the evolution of applications and services, which encompasses a smart relation and connectivity between people, processes, data, and things [1]. In this context, applications (e.g., eHealth, augmented reality, smart traffic control) usually require real-time analytics, stream mining, and low latency that could not be fulfilled entirely in the IoT devices given their resource constraints. Thus, recent solutions lean on the Cloud and Fog paradigms to meet the requirements from applications and services.

The associate editor coordinating the review of this manuscript and approving it for publication was Chi-Tsun Cheng.

The Cloud computing model allows the delivery of computing services such as processing, storage, and networking on-demand according to end-user requirements. As an extension of the Cloud, the Fog brings computing services closer to the source of data generation (i.e., network edge) providing lower latency levels, mobility support, and location awareness [2], [3]. But since the Fog represents a vast environment filled with heterogeneous devices, it is necessary to design and develop smart placement mechanisms that optimize the resource usage while improving the performance of the applications, particularly for those that are latency-sensitive, such as online gaming, videoconferences, and virtual/augmented reality.

The placement mechanisms should work in unison with clever monitoring systems, to keep track of the current status of the applications and the underlying infrastructure, and to guarantee that the proper decisions can be made in order to

fulfill the Quality of Service (QoS) requirements from the Cloud/Fog to the IoT [4], and thus maintaining the Quality of Experience (QoE) for end-users [5]. It is necessary to gather fairly complete information regarding required resources and operational behavior of the system [6], such as applications resource demand characteristics under different workloads and impact of different infrastructure configurations on the QoS and QoE. With this approach, as well as by gathering data and monitoring the behavior of the components (i.e., services, applications, and users), it is possible to identify profiles to enable smart decision mechanisms from the orchestration point of view.

Applications can be profiled according to different factors, such as their resource requirements [7], [8] and their workload [9], [10]. Another possibility is using their popularity, measured by their request count, in order to prioritize them during their placement. The changes in the popularity can be grouped in *profiles* that can guide the placement process. Thus, more popular applications can be favored in the placement process, to ultimately benefit a larger number of end-users by offering them better performance, including lower latency.

Using the profiles of content and applications has been studied in the past [10], [11]; however, the Cloud to IoT ecosystem introduced new features that require a new approach regarding how to characterize the behavior of applications and their components in this context. Furthermore, using the popularity profiles of applications during their placement might affect their final performance and the response time of the overall system. Here, the following question arises: is it possible to improve latency by taking advantage of the profiles of applications, based on the correlation between popularity and Quality of Service?

To answer the previous question, in this article we propose two dynamic mechanisms for the placement of applications and their components (i.e., microservices) in the Cloud/Fog ecosystem, combining the profiles of applications and infrastructure metrics (e.g., propagation delay and resource consumption) aimed at reducing latency. We consider four types of popularity for application profiles over time: static, increasing, decreasing, and oscillating. The mechanisms use the profiles of the applications, which are updated using a time window, to decide where to deploy the components of an application in the substrate infrastructure. The first mechanism uses a genetic approach that combines resource usage and latency, weighted by the popularity of the applications, for decision making. The second one is a variation of an already introduced mechanism, called Popularity Ranked Placement [12], that utilizes a graph partition method to rank the infrastructure nodes where applications prioritized by popularity are going to be placed. Simulation results show that it is possible to reduce the latency and the jitter of popular applications while also reducing the latency of the overall system.

The contributions of this paper include: (1) the proposal of using application profiles, based on their popularity, for service placement; (2) the new version of our Popularity Ranked Placement algorithm [12], updated to support dynamic scenarios and the handling of applications via profiles; and (3) a service placement solution based on the Weighted Sum Genetic Algorithm for service placement via application profiles.

This paper is organized as follows. Section II offers a revision on related work on service placement in the Fog as well as using application profiles for service placement. Section III proposes the use of application profiles to orchestrate the Fog, to later introduce two service placement mechanisms that use the profiles of applications for their placement decisions. One mechanism is based on Genetic Algorithms (Subsection III-B) and the other is based on graph partitions, following the PageRank algorithm (Subsection III-C). Section IV describes the evaluation setup, so the results presented in this paper can be recreated. Section V outlines the experimental results and presents an analysis of the findings. Section VI concludes the paper, offering suggestions for future research paths to advance this work.

II. RELATED WORK

There have been many efforts on the field of service placement for Cloud environments, with more recent works focusing on the extended Cloud-Fog-IoT continuum. Relying on the Fog brings the advantage of reducing the latency by bringing the service closer to end-users, however, the Fog is a vast environment and placement in this ecosystem becomes an NP-hard problem that must be carefully handled.

The works analyzed in this section are grouped in two categories: works related to *service placement* and works related to using *application profiles*. Both categories are summarized in Table 1. Regarding the service placement, several strategies have been proposed to solve this problem in Cloud and Fog environments. Many proposals are solely based on Fog environments, with some exceptions considering Cloud/Fog environments [13]–[17].

It is relevant to notice the use of mathematical programming techniques, like Linear Programming (LP), ILP, and Mixed Integer Linear Programming (MILP) [15], [18], [19], in the design of models to find the optimal location for services, as well as the use of mathematical models [20]–[22]. GA is another technique commonly used for placement heuristics [14], [18], [23], [24]. An additional factor to consider is the metric guiding the placement process. Mostly, the metrics used are based on network aspects such as resource usage [13], [16], [18], [23], power consumption [21], [24], reliability [25], availability [26], QoS [17], [20], and time-related metrics such as response time [24], [27], delay [21] and latency [14], [15], [22], [23], [28], [29]. In the case of metrics that concern the applications and their services, QoE, which measures the quality as experienced by the user, is explored as metric [30]; however, no element to describe the applications and the behavior of their services is used during the placement

TABLE 1. Characteristics from related work.

Service Placement				
Work	**Decision Factor**	**Approach**	**Environment**	**Evaluation**
Skarlat et al. [18]	Resource usage	ILP + GA	Fog	iFogSim
Wang et al. [19]	Cost	LP + Heuristic	Mobile micro-clouds	*Unspecified simulator*
Taneja and Davy [13]	Resource usage	Module mapping algorithm	Cloud/Fog	iFogSim
Lera et al. [26]	Availability	Graph partition	Fog	YAFS
Guerrero et al. [28]	Latency	Decentralized placement	Fog	iFogSim
Mahmud et al. [30]	QoE	Fuzzy logic	Fog	iFogSim
Brogi and Forti [20]	QoS	Mathematical model	Fog	FogTorch
Guerrero et al. [23]	Latency, service spread, resource usage	GA	Fog	Python
Paul Martin et al. [25]	Cost, reliability	Heuristic	Fog	iFogSim, Testbed
Venticinque and Amato [27]	Response time, resource usage	BET method	Fog	Testbed
Liu et al. [21]	Energy consumption, delay, cost	Mathematical model	Fog	*Unspecified simulator*
He et al. [29]	Computational cost, latency, BW	Benchmark	Fog	*Own simulator*
Shi et al. [14]	Latency	GA	Cloud/Fog	Matlab
Santos et al. [15]	Requests, migrations, latency	MILP	Cloud/Fog	CPLEX
Pang et al. [22]	Latency	Mathematical model, heuristic	Fog	*Unspecified simulator*
Natesha et al. [24]	Service time, cost, energy consumption	GA	Fog	Testbed
Nezami et al. [16]	Resource usage, cost	Decentralized multi-agent	Cloud/Fog	Testbed
Sami et al. [17]	QoS	Deep Reinforcement Learning	Cloud/Fog	Python, Tensorflow
Application Profiles				
Work	**Profile Factor**	**Approach**	**Environment**	**Evaluation**
Chowdhury et al. [7]	Resource requirement	Architecture proposal	Cloud/Fog	*Theoretical proposal*
Filip et al. [8]	Resource requirement	Mathematical model, heuristic	Cloud/Fog	CloudSim
Jindal et al. [31]	Resource requirement	Performance model	Cloud	Testbed
Han et al. [9]	Workload	Greedy heuristic	Cloud/Fog	Testbed
Yu et al. [32]	Saturation point	Map-table, Bayesian optimization	Cloud	Testbed
Ye et al. [10]	Workload	Mathematical model	Cloud	LINGO, Testbed
Khan et al. [6]	Resource requirement	Technology framework	NFV	Testbed

process, except one work [15] that explores the placement of applications according to their requests, but does not categorize the applications accordingly. For the validation, most works use simulation, varying the tool used, largely iFogSim [13], [18], [25], [28], [30], but also YAFS [26], FogTorch [20], Matlab [14], and Python [17], [23]. There is also evaluation via testbed using dockers and containers [16], [24].

From this review, it seems more relevant to consider the entire Cloud/Fog environment instead of focusing only on the Fog since this will result in a more complete solution for real scenarios. Also, there is a lack of works that consider the use of characteristics intrinsic to the applications and the behavior of their services to guide the placement process. Most of the works are focused on using network metrics during their selection process, and no profile of the applications is used during the placement. Using an application metric could take advantage of application profile analysis, since different applications might have different requirements and/or behavior. Even more, there is a possibility of using a combination of metrics during the placement process to help in the selection of the optimal location for a given service.

Concerning the use of application profiles, although this topic has been vastly addressed in the context of content placement [11], the use of profiles of applications to guide their placement has not yet been thoroughly investigated in the Cloud/Fog context.

From the selection of works highlighted in Table 1, some approaches are based on service requirements, so the service can be deployed in nodes with specific hardware characteristics [7], [8], [31]. Other proposals use profiling to determine when to scale services, after identifying workloads and saturation points [9], [32]. Empirical profiling for service placement has also been explored, although limited to placement of Virtual Machines into physical ones [10]. Likewise, there are proposals of orchestration architectures [7] that consider scheduling/planning modules based on application profiling, but limited to the study of Network Funtions Virtualization (NFV) [6]. Some works present the empirical evaluation of

their proposal using testbeds including technologies such as Kubernetes [9], [31], [32]. Thus, to the best of our knowledge, there are no proposals for using application profiles in the Fog based on application features (e.g., behavior) beyond their resource requirements.

As open issues regarding the service placement of applications composed of microservices, one possibility is studying the use of hybrid metrics, combining profiles of applications according to their behavior, and network-related metrics that can guide the placement process to an optimal solution. Both metrics can change during time, adjusting themselves to reflect the current conditions of the network and the demands of users, unlike other metrics used in related works, such as hop count [28] or service resource requirement [7]–[9], [31].

This work uses the popularity profile of the applications, indicated by their request count, as a metric to characterize them and prioritize their placement. Two placement mechanisms are proposed to validate the use of application profiles during the placement process; both mechanisms combine the profiles based on the popularity of the applications and the propagation delay of the network, in order to minimize the latency of the most popular applications, taking advantage of the entire Cloud/Fog landscape.

III. PLACEMENT VIA APPLICATION PROFILES

Placement in Cloud/Fog environments can have a crucial impact on reducing latency. Using characteristics from the applications to place has proven to be useful, prioritizing them according to a given criterion [12]. The popularity of the applications is a good metric to use, since prioritizing popular applications might favor a majority of users.

This section describes how the Orchestrator could take advantage of using application profiles, to later introduce two placement mechanisms based on prioritizing the placement of microservices that belong to popular applications to minimize their latency. To add dynamism to both proposals, and evaluate how they adapt to changing conditions on the network and on the requests from end-users, a time-window approach is used. This means the Orchestrator will adapt the placement locations at the beginning of each time window, considering the new conditions in the network and the application requests. Both placement mechanisms are described in the following subsections.

A. USING PROFILES TO ORCHESTRATE APPLICATIONS

As stated in Section I, the Orchestrator has to monitor the deployment infrastructure (i.e., nodes, links, services, and users) to make smart informed decisions that will affect the behavior of the system. An architecture for a Orchestrator was already introduced [4], and one of its main modules is dedicated to the execution of planning mechanisms in charge of the selection of optimal location to deploy the application components. To improve the performance of the planner mechanisms, it is essential to provide them with knowledge not only about the substrate network, but also of the entire system, including users and applications. The use of a behavior profile for this goal would provide the Orchestrator the possibility to respond to the changing conditions in the environment.

The devices in the Fog and IoT are notoriously resource constrained in comparison with the Cloud [18]. However, in these tiers end-users will benefit from lower latency. Thus, it is critical to determine the proper location in which to deploy the services. For this particular scenario, having knowledge of the behavior of application components would benefit the decision-making for the Orchestrator. This knowledge has to cover past behavior and current conditions, to be able to react to dynamic situations such as traffic load variations.

Although the use of profiles has been explored in the context of the Cloud, the adoption of the microservices architecture brings new challenges, given the fine granularity of the components that can even be shared between different applications. To the best of our knowledge, the use of behavior profiles in Cloud to IoT scenarios has not been thoroughly exploited so far from the academic point of view, because of the complexity of the landscape and the lack of datasets available for study to apply data analytics techniques, with some studies using application requirement profiles [7]–[9], [31]. Nevertheless, there have been some recent efforts to gather real-time data in this kind of environment that are still in an early stage [6].

Figure 1 describes the overall placement process proposed in this work, including the gathering of information about application behavior. Users place their requests on different microservice-based applications. The Orchestrator (based on the architecture presented in [4]) collects this information and uses it to generate smart informed decisions. Particularly, the *Planner Mechanisms*, in charge of determining the location to deploy the microservices, apply a categorization to the

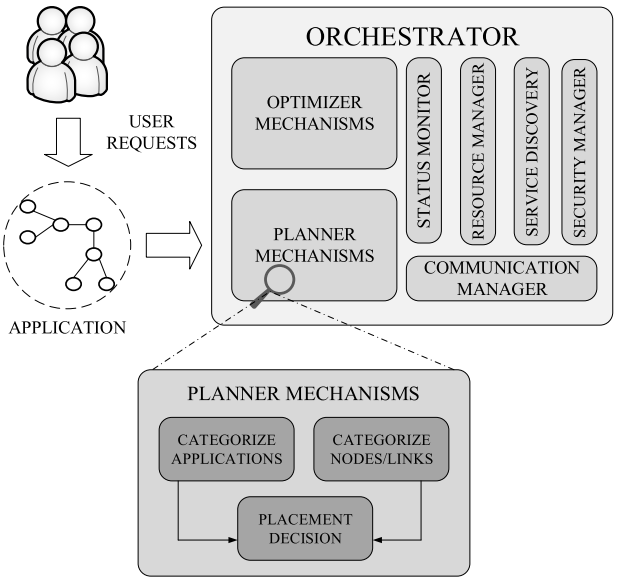


FIGURE 1. Overall Design of the Placement Process.

applications based on their popularity, and also a categorization of the substrate components of the deployment infrastructure (i.e., node resources and propagation delay) to make the placement decision. The Orchestrator will re-evaluate the placement decisions over time using a time window to update the values regarding the popularity of the applications, as well as the revised information about the deployment infrastructure.

To determine if the information about the behavior of the applications applied in the placement decisions impacts the final performance of the applications, particularly the latency perceived by end-users, we designed two placement mechanisms that use application profiles based on popularity. Four popularity profiles were used in this work:

- *fixed*, where the application does not vary the number of requests over time;
- *mixed*, where the application oscillates gaining or losing popularity over time;
- *up*, where the application continuously gains popularity, and;
- *down*, where the application repeatedly loses popularity.

These profiles cover different changes in the popularity of applications, showing a dynamic behavior over time. This could model viral communications using social networks or access to email applications during business hours (quickly adding more requests to later on decrease them), but at the same time applications with a more static behavior, like automatic weather updates.

The following subsections describe the placement mechanisms designed to validate if the use of popularity profiles influences the final performance of the applications.

B. GENETIC PLACEMENT GUIDED BY POPULARITY

The placement of applications in network nodes is an optimization problem that has already been addressed by using many optimization techniques, such as ILP [18], [19] and other mathematical models [20]. However, the Cloud-Fog-IoT poses a complex scenario, filled with heterogeneous devices and dynamic conditions, that make optimization solutions not suitable because of their lack of adaptability and high response times [33]. An alternative is using Genetic Algorithms, which are often viewed as function optimizers and have previously been used to solve service placement problems [14], [23], [24], [34], [35]. From the different variants of GAs and Evolutionary Algorithms (EAs) used for service placement in the Fog, the Weighted Sum Genetic Algorithm (WSGA) showed a reasonable compromise regarding the execution time and fitness values obtained, while also exhibiting lower convergence times [23] which led to the selection of this algorithm for this work.

WSGA consists of a transformation that normalizes the values of multiple objective functions to the unit interval and weights them to obtain a final result. The calculation used for the *fitness function* is shown in Equation 1, which is the result of the product of ω_i, the scaling factor; θ_i, the weight; and X_i, the value of the objective function. *numObj* represents the total of objective functions to be combined in this equation.

$$fitness = \sum_{i \in numObj} \omega_i \times \theta_i \times X_i \tag{1}$$

Table 2 lists the parameters and variables used by the GA heuristic implemented in this work. For variables ω_s and Ω_n the resource unit is used, where a resource unit is a vector holding the resources (i.e., CPU, memory, storage) of node $n \in N$. The cost matrix contains the cost (in terms of latency) to reach a node from a gateway ($gw \in GW$). $C_{n,gw}$ equals the propagation delay of the shortest path that connects $n \in N$ to $gw \in GW$. The instance matrix identifies the microservices that compose an application. $I_{a,s}$ equals 1 if microservice $s \in S$ belongs to application $a \in A$, and 0 otherwise. Finally, the placement matrix relates the request per application and the nodes that are selected as the location for the microservices. $P^{a,r}_{s,n} = 1$ if microservice $s \in S$ is located in node $n \in N$ to satisfy request $r \in R$ for application $a \in A$, and 0 otherwise.

TABLE 2. Parameters and variables for the GA heuristic.

Parameters	
Parameter	**Description**
S	Set of microservices to be placed
N	Set of nodes where the microservice can be executed
GW	Set of gateways
A	Set of applications. An application is composed by a set of microservices
R	Set of requests for all the applications
Q_a	Sum of requests for $a \in A$
Ω_n	Resource capacity for $n \in N$
ω_s	Resource requirement for $s \in S$
C	Cost matrix. An $\|N\| \times \|GW\|$ matrix
I	Instance matrix. An $\|A\| \times \|S\|$ matrix
Variables	
Variable	**Description**
P	Placement matrix. An $\|A\| \times \|R\| \times \|S\| \times \|N\|$ matrix

For the GA implemented in this work, *individuals* are modeled using the placement matrix P. For a solution to be valid, the *genes* or bits in P must reflect valid requests according to R, the proper microservices $s \in S$ that compose $a \in A$ according to the information from the instance matrix I, and must comply with feasibility restrictions of node capacities imposed by Ω.

The mutation is carried out by keeping fixed the information regarding the requests ($r \in R$), applications ($a \in A$), and microservices ($s \in S$), and altering the information of the nodes $n \in N$. This way, the same services placed in a previous solution are moved to different locations, generating a different solution. If a solution is invalid, a fitness value of

infinite is assigned, so it is not passed to the next generation. This way, there is no need to incorporate an additional method into the algorithm to validate the solutions.

Algorithm 1 Weighted Sum Genetic Algorithm

```
   Result: solution
1  P_t ← generateRandomPopulation(popSize)
2  objValues ← getObjValues(Pt)
3  fitness ← ws(objValues, ω, θ)
4  foreach i in generations do
5  |  P_off ← ∅
6  |  foreach j in popSize do
7  |  |  parent1 ← selectParent(P_t, fitness)
8  |  |  parent2 ← selectParent(P_t, fitness)
9  |  |  child1,child2 ← crossover(parent1, parent2)
10 |  |  if random() ≤ mutationProb then
11 |  |  |  mutate(child1, child2, numGenes)
12 |  |  end
13 |  |  P_off ← P_off ∪ child1, child2
14 |  end
15 |  objValues ← getObjValues(P_off)
16 |  fitnessOff ← ws(objValues, ω, θ)
17 |  fitness ← fitness ∪ fitnessOff
18 |  P_off ← P_off ∪ P_t
19 |  P_off ← order(P_off, fitness)
20 |  P_t ← P_off[1..popSize]
21 end
22 solution ← min (P_t, fitness)
23 return solution
```

Algorithm 1 shows the basic procedure of this GA. It starts at line 1 by randomly generating the size of the population (i.e., *popSize*) in order to create the first generation of solutions. The objective function values are calculated for the first generation (line 2), and the fitness value is calculated (line 3). Then, at line 4, for each generation, it initializes the offspring population as the empty set, as seen on line 5.

On each generation, and for each individual in the population, the parents are selected from a binary tournament selection operator (lines 7 and 8); this is, choosing the individual with the best fitness from a subset of the population. After selecting the parents, the children are created by applying a crossover operator (line 9) and then mutating them with a probability of *mutationProb*, as seen in line 11. The crossover consists of mixing a portion of the first solution (i.e., *parent1*) with the remaining portion of the second solution (i.e., *parent2*). The solutions of the children are mutated with a uniformly distributed probability, as shown in line 10. The mutation selects a node from the solution and changes it for a different node (i.e., for a given microservice of a given application and a given request, the location is updated). The number of nodes to update (i.e., mutate) in the children solutions is based on the total number of microservices in the solution, indicated by the *numGenes* parameter in line 11.

In line 13, the newly created children join the new population (offspring). The fitness value is calculated in line 15 as described by Equation 2 and Equation 3. Both objectives are combined with Equation 1 in line 16. The fitness values are combined in line 17, and all the population members are joined in line 18. All the elements of the population (including the newly created children) are sorted by their fitness value in line 19. Only the best *popSize* elements of the population will survive for the next generation (line 20). Invalid solutions (i.e., those that do not respect feasibility restrictions of node capacities imposed by Ω) will be assigned *infinite* as fitness value and thus will be discarded. After iterating for *generation* times, the returned solution will be the one with the best fitness value, shown in line 23.

Latency and resource usage are the two different objectives combined for the fitness value, using Equation 1. Latency is calculated using the propagation delay of the links in the network topology (other latency sources, e.g., processing delay, are out of the scope of this work), and the resource usage is calculated using a resource unit [36]. The goal is minimizing both objectives prioritizing the latency, using the weight factor (i.e., θ) for this purpose.

To evaluate the latency (*lat* in Equation 2), information about the propagation delay (i.e., matrix C) is added for the services that belong to application $a \in A$ (i.e., matrix I) of the placed applications (i.e., matrix P). This value is weighted according to the popularity of the application, as stated by Equation 2; where Q_a is the number of requests for application $a \in A$ measuring its popularity. The objective function value is calculated as the product of the different variables, as described in Equation 2.

$$lat = \sum_{a\in A}\sum_{r\in R}\sum_{s\in S}\sum_{n\in N}\sum_{gw\in GW}\frac{1}{Q_a}\times\left[P_{s,n}^{a,r}\times I_{a,s}\times C_{n,gw}\right] \tag{2}$$

Regarding the second objective function, resource usage (*ru* in Equation 3), it is calculated by adding the resources used by the node, as shown in Equation 3; calculated as the product of ω_s, the number of resources required by service $s \in S$; the placement matrix P; the instance matrix I; and the cost matrix C. Ω_n denotes the amount of resources of node $n \in N$. Equation 3 depicts the free resources of nodes $n \in N$.

$$ru = 1 - \frac{\sum_{a\in A}\sum_{r\in R}\sum_{s\in S}\sum_{n\in N}\left[P_{s,n}^{a,r}\times I_{a,s}\times\omega_s\right]}{\sum_{n\in N}\Omega_n} \tag{3}$$

Both objectives are combined in Equation 1 and minimized in Algorithm 1 (see line 22).

According to the conditions of the scenario (i.e., size of the topology, traffic load), the number of generations needed to reach an optimal solution increases, impacting the computation resources and execution time needed. An alternative heuristic for more complex scenarios is presented in the following section.

C. POPULARITY RANKED PLACEMENT

A heuristic based on the PageRank algorithm, called Popularity Ranked Placement (PRP), was already presented [12]. Adaptations to the basic version of the heuristic were performed to extend its functionalities by allowing the use of application profiles and the support of dynamic scenarios. The new version of PRP is described in this subsection.

Algorithm 2 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
```

The PageRank algorithm was originally introduced to rank web pages in a search engine [37], and the idea is to rank the nodes in a graph via probability propagation. In PRP, the PageRank algorithm is used to rank the nodes in the network topology in order to build *communities* where the placement will take place. The idea behind this approach is to avoid the saturation of the gateways by sharing the load among the nodes with lower propagation delay connecting routes, which are to be grouped in the community of the gateway. Each gateway will have a community and the requests placed in a gateway will be deployed (if possible) in its community. The community building process is presented in Algorithm 2.

The topology is initialized in line 1. The nodes are ranked using the PageRank algorithm [37] using the propagation delay (*PD*) of the links as a weight metric for the probability propagation calculation [12], as seen in line 2. This means the nodes with highest rank will be those connected via links with the lowest propagation delay whether they are neighbors or not. Those nodes with a transition probability higher than a threshold (line 7) will belong to the community of the gateway. The nodes that did not become part of any community will be grouped in a community, called the *non-community* nodes, initialized in line 4 and updated in line 16.

After all the communities are built, it is time to deploy the microservices as determined by Algorithm 3. The same *Placement matrix* variable described in Table 2 is used to guide the placement process, as in Algorithm 1. The communities are created in line 3, using the previously described *Build Communities* algorithm (Algorithm 2).

Algorithm 3 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 ← rankAppsByProfile();
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
```

The applications to deploy are ranked using their popularity profiles in line 5. The first option is to place the microservices of the applications in the community of the gateway where the request was launched. An expanding window is used to distribute the microservices along the nodes inside the community, trying to take advantage of the best ranked nodes without saturating them. If the community window does not have enough resources to host the microservice (line 8) the window will expand (line 10). If there are enough resources within the community window, the microservice is deployed (line 13); in the case that the community window runs out of resources, the deployment will be carried out in the non-community nodes (line 16). If all previous attempts fail, a Cloud deployment will be carried out (line 18). The placement decision will be performed by the Orchestrator on each placement time period.

Simulations experiments were carried out to validate the performance of the placement mechanisms and to evaluate the impact of using the proposed popularity profiles. The experimental setup used is described in the following section.

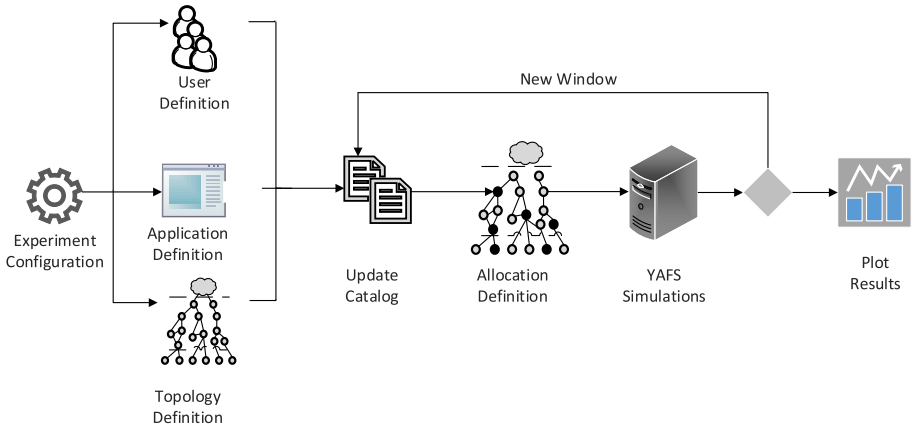


FIGURE 2. Experimental Methodology.

IV. EXPERIMENTAL SETUP

Yet Another Fog Simulator (YAFS) [36] was selected as simulation tool because of its strong support for Fog features and high granularity of result reports [38]. 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 was used to run the experiments. The PC was running Microsoft Windows 10 Pro (Build 18363) operating system and Python 2.7.16 for YAFS. Figure 2 shows the methodology followed during the evaluation process.

The experiment configuration was composed of three files: (1) the definition of the request by users, which specifies the profiles; (2) the definition of the applications, their microservices, and how they communicate; and (3) the topology of the underlying network where the applications will be deployed. This information is loaded in the catalog, which is used by the Orchestrator to determine the profiles of the applications according to their changing popularity. The *Planner Mechanisms* (as depicted in Figure 1) will make the placement decisions generating the allocation definition file for the current time window, and this information is provided to the YAFS simulation engine which will generate the results for each time window. This process is repeated for the different time windows, updating the catalog and the placement decisions accordingly. After all the time windows are executed, the final results are collected and plotted.

The network topology was built using the complex network theory, following a random Barabasi-Albert network model. 100 nodes were deployed in the Fog, and an additional node represents the centralized Cloud for a total of 101 nodes. The Cloud node was the one connected to the Fog with the highest betweenness centrality. In the Fog, the nodes with lower betweenness centrality were delegated as gateways, representing the nodes at the edge of the network.

The applications were randomly generated following the *Growing Network* graph structure, in which the vertices are added one by one, with an edge to the last added vertex [39]. Finally, two vertices (except the source) are randomly selected to generate an information flow towards the source vertex. This allows modeling applications that collect data and send an automated answer, which covers most of the applications typically used in Cloud-Fog-IoT continuum scenarios (e.g., sensing/actuating, eHealth, virtual/augmented reality).

The configuration parameters for the experiments are summarized in Table 3, for the network links, Fog nodes, GWs, applications, and microservices. Values similar to these were used in previous work [23], [26]. The requirements for each application microservice are measured using the YAFS resource unit [36], which is a vector that contains the capacity of different computational elements (e.g., memory, CPU, hard disk). For the GA, since the main goal in this work is to minimize the latency, this objective value received a weight of 0.9 and the resource usage only 0.1. Preliminary tests were performed to determine the number of generations to use; the resulting trend allowed us to determine that around 400 generations the fitness value converges, halting the evolution process for all the scenarios. A similar number of generations was also successfully used for WSGA in previous work [23].

Regarding the network load, three different scenarios were modelled, in order to evaluate the performance of the placement mechanisms with varying conditions: (1) *small*: five different applications; (2) *medium*: ten different applications; and (3) *large*: fifteen different applications. Each application has at least one request and follows one of the four application profiles described in Subsection III-A: *fixed, mixed, up,* and *down*, as listed in Table 4. Each application gets one of the four profiles using a uniform distribution. Meanwhile, the increase/decrease of the popularity is set as a percentage of the previous value for each time window, in this case 25%, as stated in Table 3. The number of requests was also determined using a uniform distribution. These profiles allow to model the different behavior of applications regarding their popularity. For instance, the viral transfer of content via social media applications that can gain or lose popularity over time,

TABLE 3. Simulation parameters.

Element	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	Microservices (number)	2 - 8
	Resources (units)	1 - 5
	Execution (instr/req)	20000 - 60000
	Message size (bytes)	1500000 - 4500000
	Popularity increase/decrease	0.25
Orchestrator	Time window	10000 sim. time units
Genetic algorithm	Population size	100
	Generations	400
	Mutation probability	0.25
	Num. indiv. to mutate	10% microservices
	Tournament size	2
	Weight for latency obj.	0.9
	Weight for resource usage obj.	0.1

TABLE 4. Application profiles by popularity.

Profile	Description
Fixed	The same amount of requests is maintained
Mixed	The amount of requests oscillates, increasing or decreasing in each time window according to a rate
Up	The amount of requests is increased according to a rate
Down	The amount of requests is decreased according to a rate

or an eHealth application (e.g., to control the medication of patients) in the vicinity of a hospital, that is constantly being accessed with a fairly fixed popularity.

Ten time windows were used for each simulation, with a duration of 10000 simulation time units. All the scenario setup, the source code, and additional material (i.e., plots, charts), as well as results for all the scenarios, are available via a GitLab repository.[1] The two proposed mechanisms, PRP and GA, are compared to a baseline, namely a greedy First Fit heuristic, as it was done in similar works for evaluation purposes [12], [18], [40]. In the case of the First Fit (FF) algorithm, the nodes were organized from lowest to highest, according to their available resources. This way, the nodes with fewer resources are prioritized. These nodes usually correspond with the nodes deployed at the edge of the network. Thirty simulations were executed to mitigate the statistical error, including 95% confidence intervals in the plots. The simulation results are presented in the following section.

V. RESULTS AND ANALYSIS

The performance of PRP and the genetic approach GA, and the impact of using the popularity profiles, are evaluated in this section.

Figure 3 shows the total latency per scenario. PRP always showed the lowest latency, followed by GA and FF. As the load grows, so does the latency for all the mechanisms,

[1]https://git.dei.uc.pt/kcastro/appProfiling.git

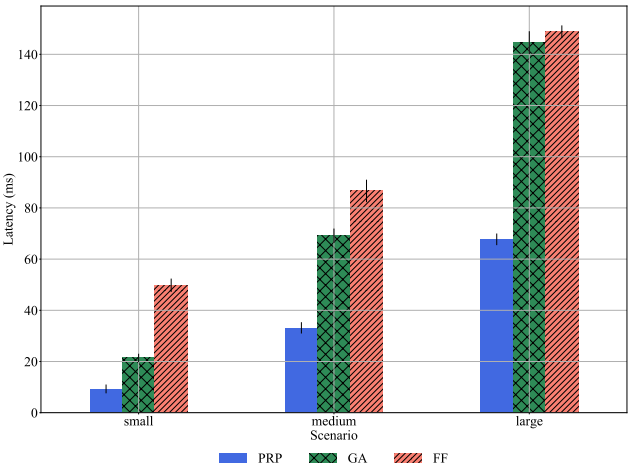
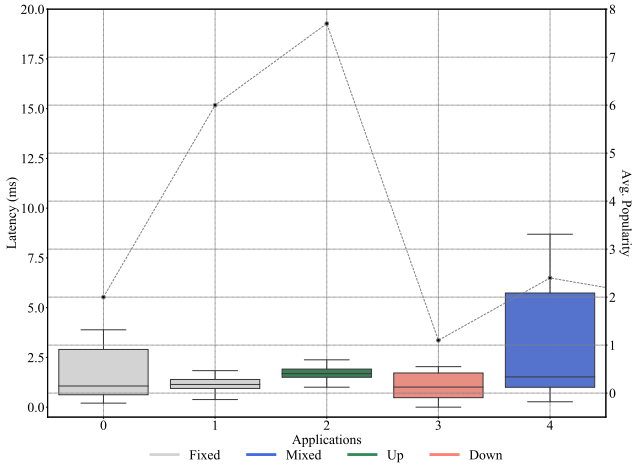


FIGURE 3. Total Latency by Scenarios.

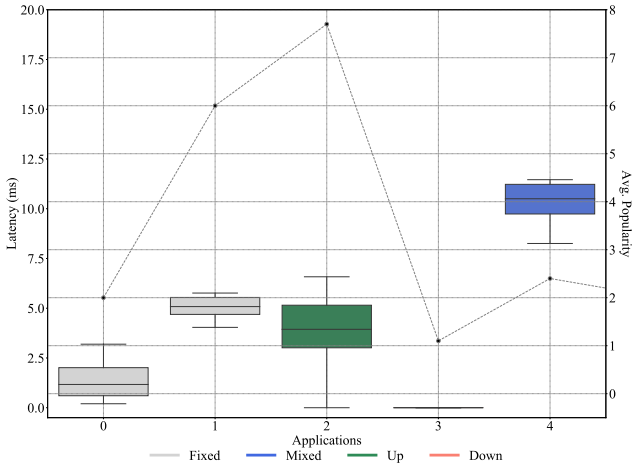
as expected. The traffic increases, saturating the nodes and communication links, influencing the overall latency of the system. For the smallest load (i.e., *small* scenario), GA showed an exceeding latency of 2 times the values reported by PRP, while FF showed a surmount of around 5 times over PRP. This breach shrinks as the load grows, as seen in the *large* scenario, where GA is only slightly better than FF. This is because as the load grows, the feasibility condition to validate solutions generated by GA (i.e., not exceeding the capacities of the nodes) was more difficult to reach via the mutations introduced by the algorithm, generating solutions with elevated fitness values that were discarded for the following generations, thus evolving slower. More generations are going to be needed as the load grows to attain better results, impacting the execution time.

The following plots also show the latency but with a finer granularity level. The latency is shown by mechanism and by application for the small, medium, and large scenarios. The mean value of the boxplot determines the average latency of the application (see left Y axis). In contrast, the boxplot itself reflects the variation of the latency values, i.e., the jitter experienced by the application, which can be calculated due to the dynamism in the scenario. Different colors are used to code the profile to which the application belongs (see Table 4), and finally the dashed line (see right Y axis) shows the average popularity (i.e., number of requests) of the application during the entire simulation (i.e., along all the time windows).

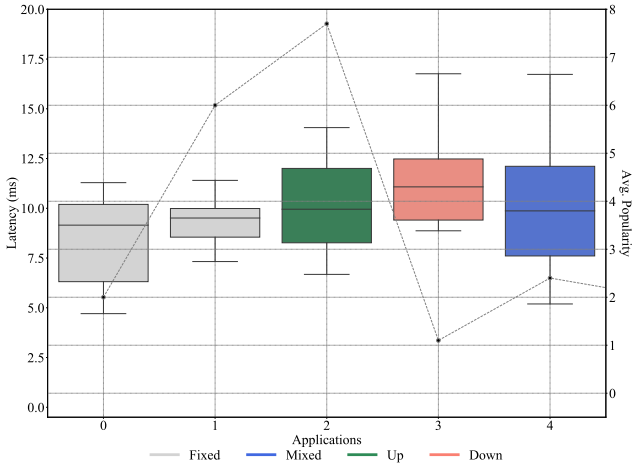
Figure 4 shows the results for the small scenario for PRP (Figure 4a), GA (Figure 4b), and FF (Figure 4c). The first observation that arises is that PRP showed the lowest latency values, followed by GA and FF, confirming the results shown in Figure 3. On the other hand, the smallest jitter was shown by GA, followed closely by PRP and then by FF. However, for PRP, it is noticeable that with higher popularity, the latency variation (i.e., jitter) is lower, giving a clearer advantage to the most popular applications. This means that PRP showed a better treatment of the applications according to their profile.



(a) PRP

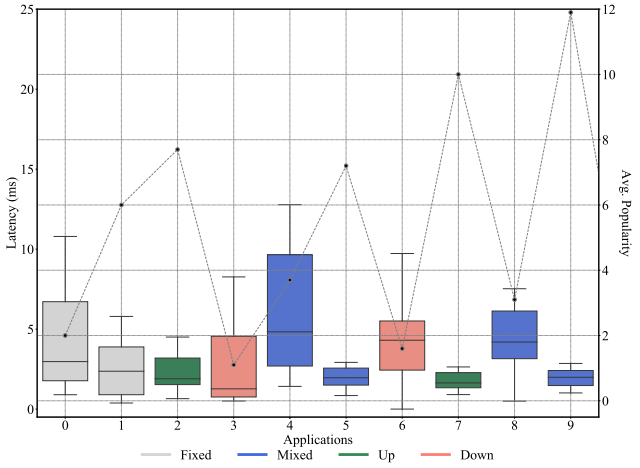


(b) GA

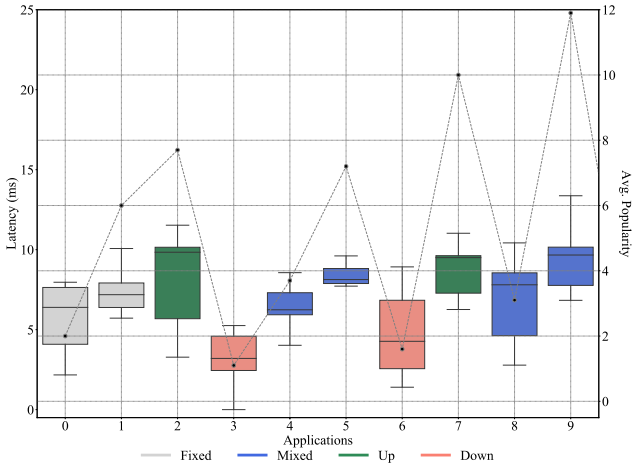


(c) FF

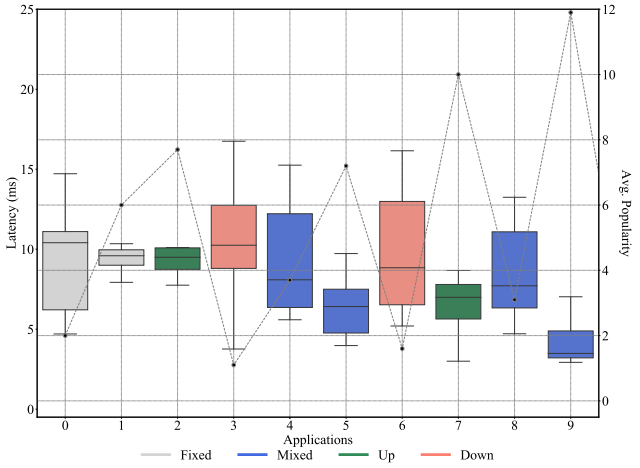
FIGURE 4. Latency and Jitter per Application - Small Scenario.



(a) PRP



(b) GA



(c) FF

FIGURE 5. Latency and Jitter per Application - Medium Scenario.

FF showed no difference in the treatment of the applications regarding their popularity.

Figure 5 shows the results for the medium scenario for PRP (Figure 5a), GA (Figure 5b), and FF (Figure 5c). Notice the different maximum sizes on the left Y axis in comparison with Figure 4. This is due to the increase in traffic load by doubling the amount of applications in the medium scenario, having a direct impact on the latency of the system. The average popularity size (right Y axis) also has a higher maximum

value for the medium scenario, since Application 9 (the one with highest popularity) was not present in the small scenario.

In this case, for PRP it is noticeable that applications with a fixed popularity profile have similar latency although slightly different jitter when their popularity is lower (see Application 0) or higher (see Application 1), favoring the later. Applications with an increasing popularity profile (see Applications 2 and 7) show similar latency and jitter, which are also lower to those displayed by applications with other profiles.

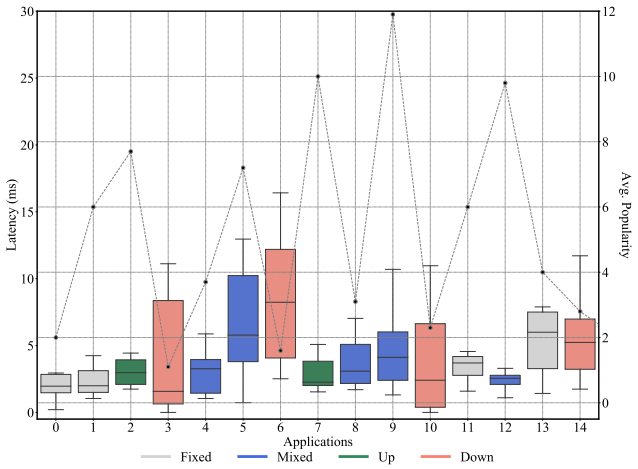
For the applications that have oscillating popularity values the jitter is more erratic, since for some time windows they were gaining popularity and for others they were losing popularity, thus having less stable results. Finally, for applications that constantly lost popularity (see Applications 3 and 6) the jitter is higher. The advantages of having a lower jitter as the application has more popularity are less evident with GA, although the mixed behavior for oscillating applications is also present. Still, GA outperformed FF in both latency and jitter. FF showed the highest latency and jitter values, as well as fewer discrepancies among applications regarding their popularity levels and profiles.

Figure 6 shows the results for the large scenario for PRP (Figure 6a), GA (Figure 6b), and FF (Figure 6c), respectively. Again, there is an increase in the maximum size displayed in the left *Y* axis due to the increase in the traffic that affects the overall latency and jitter of the applications.

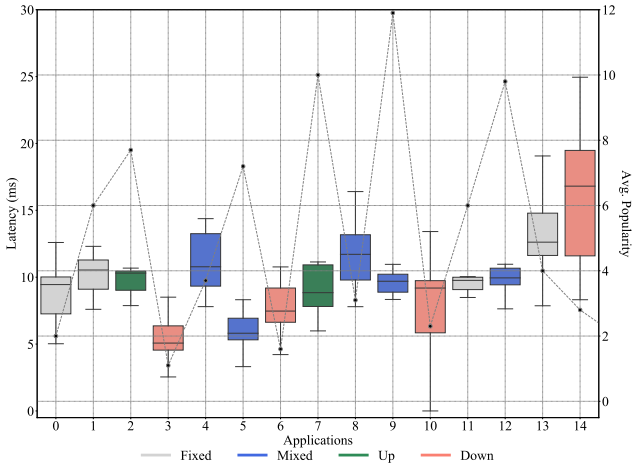
The first observation that arises is that PRP showed the lowest latency values, followed by GA and FF, as in the previous scenarios, although the gap between PRP and GA increases as the traffic load grows. On the other hand, the smallest jitter was shown by GA, followed closely by PRP and then by FF. For PRP, it is noticeable that with higher popularity, the latency variation (i.e., jitter) is lower, giving a clearer advantage to the most popular applications. This means that PRP showed a better treatment of the applications according to their profile. FF showed no difference in the treatment of the applications regarding their popularity.

In the large scenario, and as depicted in Figure 3, PRP shows the most significant reduction of latency for the overall system, with GA showing results slightly better than FF. This confirms that as the scenario increases its complexity and the load raises, GA becomes an unviable solution since it is computationally heavier without having a significant impact on the reduction of latency. Nonetheless, GA was still able to reduce the jitter of the overall system even in the scenario with the heaviest load. Latency and jitter were reduced for the overall system, but particularly for the applications with growing (*up*) and static (*fixed*) popularity profiles.

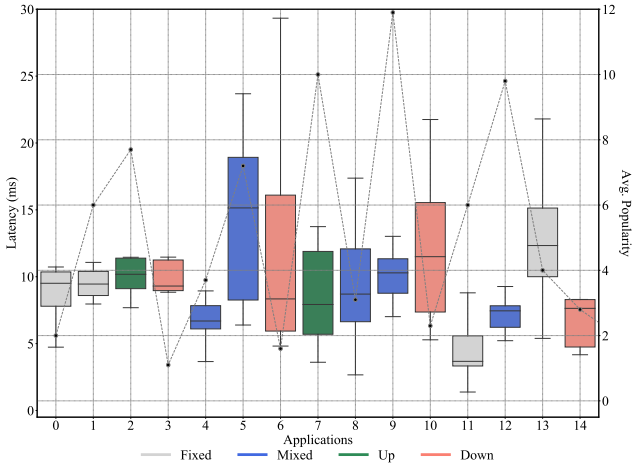
For all the scenarios, with PRP and GA it is noticeable that as the popularity grows or stays fixed in a high value, the latency tends to remain stable and low, with smaller jitter values. On the other hand, as the popularity decreases or oscillates, the latency shows significant variations, increasing the jitter. FF showed less differentiating behavior regarding the popularity of the applications, as well as displaying the highest latency and jitter values. The reduction of jitter can be particularly beneficial for certain types of applications, such as real-time communications, virtual reality, Cloud games, and videoconferences.



(a) PRP



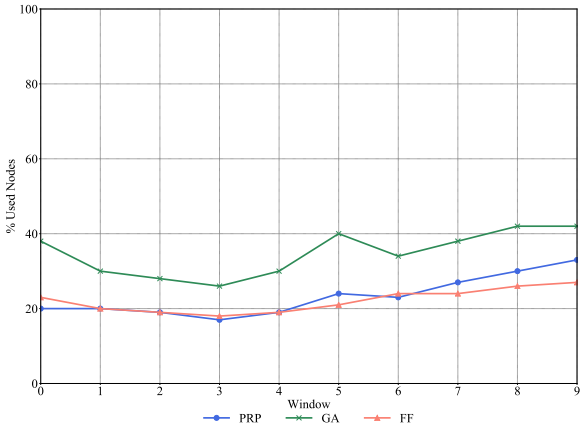
(b) GA



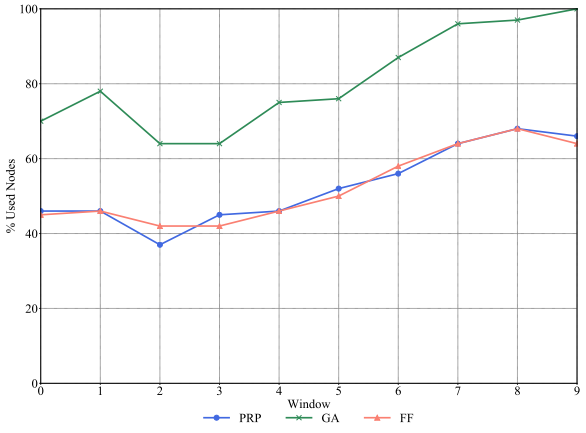
(c) FF

FIGURE 6. Latency and Jitter per Application - Large Scenario.

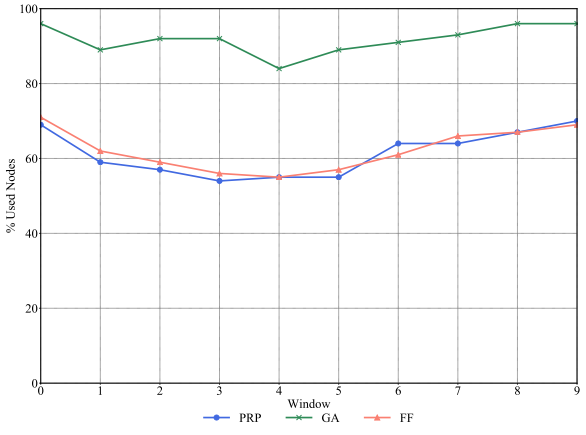
Figure 7 depicts the resource usage in the network. The *Y* axis shows the percentage of nodes used (100 Fog nodes and 1 Cloud node for a total of 101 nodes in the topology) during each time window, indicated in the *X* axis. Figure 7a shows the results for the small scenario, Figure 7b for the medium scenario, and Figure 7c for the large scenario.



(a) Small Scenario



(b) Medium Scenario



(c) Large Scenario

FIGURE 7. Nodes Usage by Scenario.

For all the scenarios, PRP and FF exhibit similar results. FF organizes all the nodes by resources (i.e., from fewer resources to more resources) and fills each node with incoming microservices. PRP also tries to distribute the microservices within each community, prioritizing the nodes with lower latency. Thus, the prioritized nodes on each mechanism will fill out sooner than less favored nodes, using fewer nodes in the network. On the other hand, for GA, since the resource usage was included as a secondary objective within the fitness function, this mechanism will promote leaving unused space on each node, spreading the microservices and leading to more used nodes, as depicted for all the scenarios in Figure 7, although the gap is more notorious in the medium (Figure 7b) and large (Figure 7c) scenarios. This behavior for GA will lead to more energy consumption since more nodes are going to be used; however, by spreading the microservices into more nodes, fewer microservices would be affected in the case of node failure. Evaluation of node failure is out of the scope of this work.

Finally, Table 5 shows the execution time, in seconds, for the algorithms in all the scenarios. Considering that FF has the more straight-forward logic, it has the lowest execution time, followed relatively close by PRP. On the other hand, GA shows a significantly higher execution time, particularly for the larger scenarios. Thus, GA might not be suited for more complex and dense scenarios.

TABLE 5. Execution time (in seconds).

Scenario	PRP	GA	FF
Small	0.020	17.224	0.010
Medium	0.031	42.157	0.019
Large	0.042	99.357	0.031

The mechanisms shown in this section (especially PRP) could use different profiles based on another criterion instead of popularity but on other relevant factors, for instance, how sensitive the applications are to the jitter. These types of profiles could prioritize jitter-sensitive applications, offering lower jitter values while maintaining low latency levels for the overall system.

VI. CONCLUSION

The complexity that derives from the combination of paradigms such as the Cloud, Fog, and IoT requires the revision of the Orchestration mechanisms to manage this environment. Among the tasks that have to be revised are those related to service placement. Applying an analysis of the applications to place provides service placement mechanisms with additional information that optimizes their performance. Using the popularity profiles of the applications, measured by their requests count, is a possibility to categorize the applications for their placement. Two mechanisms for service placement, one based on Genetic Algorithms and one based on the PageRank algorithm are proposed. Popularity-based application profiles and the propagation delay were used to guide the placement process.

The mechanisms are evaluated under dynamic conditions via simulations using YAFS, and a greedy FF heuristic was used as a baseline for evaluation purposes. The experimental evaluation showed that PRP outperformed GA in every scenario, which in turn outperformed FF. The advantages of using GA are diminished as the load increases as more generations are needed to reach better results, which implies higher execution times and computational resources. This was also observed in the execution times, for which GA required a significantly higher execution time than PRP and FF.

An advantage of using the popularity-based profiles was observed in the results, as more popular applications suffered lower latency and lower jitter than less popular applications, while the latency of the entire system was also reduced. A profiling system that organizes applications from more sensitive to less sensitive to jitter could be used combined with their popularity to benefit these applications. Results regarding node usage showed that PRP and FF used significantly fewer nodes than GA. Thus, GA might incur in greater energy consumption but have fewer microservices affected in case of node failure.

The mechanisms proposed, particularly PRP for scenarios with heavier load, could be used along with monitoring systems that collect relevant data about the behavior of applications and their services, to gather datasets that can be processed in order to extract key features allowing the identification of profiles that model how applications and services interact among them.

As future work, it is possible to explore more profiling criteria based on different factors, for instance, categorizing the microservices that generate more profit for the service provider, the modules that consume more energy, or those more prone to failure. Furthermore, it would be interesting to evaluate the performance of the placement mechanisms in a real testbed scenario.

REFERENCES

[1] E. Baccarelli, P. G. V. Naranjo, M. Scarpiniti, M. Shojafar, and J. H. Abawajy, "Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study," *IEEE Access*, vol. 5, no. 1, pp. 9882–9910, May 2017.

[2] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Comput. Commun. Rev.*, 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," Nat. Inst. Standard Technol., Gaithersburg, MD, USA, Tech. Rep. NIST 500-325, Mar. 2018.

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

[5] H. Nashaat, E. Ahmed, and R. Rizk, "IoT application placement algorithm based on multi-dimensional QoE prioritization model in fog computing environment," *IEEE Access*, vol. 8, no. 1, pp. 111253–111264, Jun. 2020.

[6] M. G. Khan, J. Taheri, A. Kassler, and M. Darula, "Automated analysis and profiling of virtual network functions: The NFV-inspector approach," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*. Verona, Italy: IEEE, Nov. 2018, pp. 1–2.

[7] S. R. Chowdhury, M. A. Salahuddin, N. Limam, and R. Boutaba, "Re-architecting NFV ecosystem with microservices: State of the art and research challenges," *IEEE Netw.*, vol. 33, no. 3, pp. 168–176, May 2019.

[8] I.-D. Filip, F. Pop, C. Serbanescu, and C. Choi, "Microservices scheduling model over heterogeneous cloud-edge environments as support for IoT applications," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2672–2681, Aug. 2018.

[9] J. Han, Y. Hong, and J. Kim, "Refining microservices placement employing workload profiling over multiple kubernetes clusters," *IEEE Access*, vol. 8, no. 1, pp. 192543–192556, Oct. 2020.

[10] K. Ye, H. Shen, Y. Wang, and C. Xu, "Multi-tier workload consolidations in the cloud: Profiling, modeling and optimization," *IEEE Trans. Cloud Comput.*, early access, Feb. 24, 2020, doi: 10.1109/TCC.2020.2975788.

[11] A. Passarella, "A survey on content-centric technologies for the current Internet: CDN and P2P solutions," *Comput. Commun.*, vol. 35, no. 1, pp. 1–32, Jan. 2012.

[12] K. Velasquez, D. P. Abreu, L. Paquete, M. Curado, and E. Monteiro, "A rank-based mechanism for service placement in the fog," in *2020 IFIP Networking*. Paris, France: IEEE, Jun. 2020, pp. 64–72.

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

[14] C. Shi, Z. Ren, K. Yang, C. Chen, H. Zhang, Y. Xiao, and X. Hou, "Ultra-low latency cloud-fog computing for industrial Internet of Things," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*. Barcelona, Spain: IEEE, Apr. 2018, pp. 1–6.

[15] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards end-to-end resource provisioning in fog computing over low power wide area networks," *J. Netw. Comput. Appl.*, vol. 175, Feb. 2021, Art. no. 102915.

[16] Z. Nezami, K. Zamanifar, K. Djemame, and E. Pournaras, "Decentralized edge-to-cloud load balancing: Service placement for the Internet of Things," *IEEE Access*, vol. 9, no. 1, pp. 64983–65000, Apr. 2021.

[17] H. Sami, A. Mourad, H. Otrok, and J. Bentahar, "Demand-driven deep reinforcement learning for scalable fog and service placement," *IEEE Trans. Services Comput.*, early access, Apr. 27, 2021, doi: 10.1109/TSC.2021.3075988.

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

[19] 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 Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1002–1016, Apr. 2017.

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

[21] L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi, "Multiobjective optimization for computation offloading in fog computing," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 283–294, Feb. 2018.

[22] A.-C. Pang, W.-H. Chung, T.-C. Chiu, and J. Zhang, "Latency-driven cooperative task computing in multi-user fog-radio access networks," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*. Atlanta, GA, USA: IEEE, Jun. 2017, pp. 615–624.

[23] C. Guerrero, I. Lera, and C. Juiz, "Evaluation and efficiency comparison of evolutionary algorithms for service placement optimization in fog architectures," *Future Gener. Comput. Syst.*, vol. 97, no. 1, pp. 131–144, Aug. 2019.

[24] B. V. Natesha and R. M. R. Guddeti, "Adopting elitism-based genetic algorithm for minimizing multi-objective problems of IoT service placement in fog computing environment," *J. Netw. Comput. Appl.*, vol. 178, no. 1, Mar. 2021, Art. no. 102972.

[25] J. P. Martin, A. Kandasamy, and K. Chandrasekaran, "CREW: Cost and reliability aware Eagle-Whale optimiser for service placement in fog," *Softw., Pract. Exper.*, vol. 50, no. 12, pp. 2337–2360, Sep. 2020.

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

[27] S. Venticinque and A. Amato, "A methodology for deployment of IoT application in fog," *J. Ambient Intell. Humanized Comput.*, vol. 10, no. 5, pp. 1955–1976, May 2019.

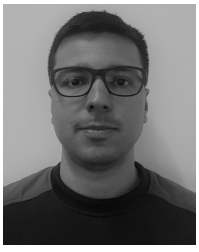
[28] C. Guerrero, I. Lera, and C. Juiz, "A lightweight decentralized service placement policy for performance optimization in fog computing," *J. Ambient Intell. Humanized Comput.*, vol. 10, no. 6, pp. 2435–2452, Jun. 2019.

[29] J. He, J. Wei, K. Chen, Z. Tang, Y. Zhou, and Y. Zhang, "Multitier fog computing with large-scale IoT data analytics for smart cities," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 677–686, Apr. 2018.

[30] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Quality of experience (QoE)-aware placement of applications in fog computing environments," *J. Parallel Distrib. Comput.*, vol. 132, no. 1, pp. 190–203, Oct. 2019.
[31] A. Jindal, V. Podolskiy, and M. Gerndt, "Performance modeling for cloud microservice applications," in *Proc. ACM/SPEC Int. Conf. Perform. Eng.* New York, NY, USA: ACM, Apr. 2019, pp. 25–32.
[32] G. Yu, P. Chen, and Z. Zheng, "Microscaler: Cost-effective scaling for microservice applications in the cloud with an online learning approach," *IEEE Trans. Cloud Comput.*, early access, Apr. 6, 2020, doi: 10.1109/TCC.2020.2985352.
[33] K. Y. Lee and M. A. El-Sharkawi, *Modern Heuristic Optimization Techniques: Theory and Applications to Power Systems*. Hoboken, NJ, USA: Wiley, 2008.
[34] H. Moens, B. Hanssens, B. Dhoedt, and F. De Turck, "Hierarchical network-aware placement of service oriented applications in clouds," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*. Krakow, Poland: IEEE, May 2014, pp. 1–8.
[35] S. Khebbache, M. Hadji, and D. Zeghlache, "A multi-objective non-dominated sorting genetic algorithm for VNF chains placement," in *Proc. 15th IEEE Annu. Consum. Commun. Netw. Conf. (CCNC)*. Las Vegas, NV, USA: IEEE, Jan. 2018, pp. 1–4.
[36] I. Lera, C. Guerrero, and C. Juiz, "YAFS: A simulator for IoT scenarios in fog computing," *IEEE Access*, vol. 7, no. 1, pp. 91745–91758, Jul. 2019.
[37] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the Web," Stanford InfoLab, Stanford, CA, USA, Tech. Rep. 1999-66, Nov. 1999.
[38] A. Markus and A. Kertesz, "A survey and taxonomy of simulation environments modelling fog computing," *Simul. Model. Pract. Theory*, vol. 101, no. 1, May 2020, Art. no. 102042.
[39] B. Yao, X. Liu, W. Zhang, X. Chen, and M. Yao, "Nested growing network models for researching the Internet of Things," in *Proc. IEEE 7th Joint Int. Inf. Technol. Artif. Intell. Conf.* Chongqing, China: IEEE, Dec. 2014, pp. 450–454.
[40] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards QoS-aware fog service placement," in *Proc. IEEE 1st Int. Conf. Fog Edge Comput. (ICFEC)*. Madrid, Spain: IEEE, May 2017, pp. 89–96.

KARIMA VELASQUEZ received the B.S. and M.S. degrees in computer science from the Central University of Venezuela, in 2005 and 2013, respectively. She is currently pursuing the Ph.D. degree with the University of Coimbra, Portugal.

From 2006 to 2014, she worked as a Researcher with the Laboratory of Computer Networks, Central University of Venezuela. She also works with the Centre for Informatics and Systems, Laboratory of Communications and Telematics. She has published several conference papers and journal articles. Her current research interests include fog and cloud computing, network performance, and quality of service.



DAVID PEREZ ABREU received the B.S. and M.S. degrees in computer science from the Central University of Venezuela, in 2005 and 2013, respectively. He is currently pursuing the Ph.D. degree with the University of Coimbra, Portugal.

From 2006 to 2014, he worked as a Researcher with the Laboratory for Mobile and Wireless Networks, Central University of Venezuela. He also works with the Laboratory of Communications and Telematics, University of Coimbra. He has published several conference papers and journal articles. His research interests include operating systems, virtualization, cloud computing, resilience, and the Internet of Things.

MARILIA CURADO received the Ph.D. degree in computer engineering from the Department of Informatics Engineering (DEI), University of Coimbra, Portugal, in 2005.

She is currently a Full Professor with the Department of Informatics Engineering (DEI), University of Coimbra. She is also the Director of the Laboratory for Informatics and Systems, Pedro Nunes Institute. She has participated in a large number of national and international projects, in particular the European FP6 EuQoS, CONTENT, and WEIRD projects and in the European projects FP7 MICIE, GINSENG, and COCKPIT-CI. She is involved in the H2020 ATENA and POSEIDON projects, being the Principal Investigator of the EUREKA Eurostars OUTERMOST and FCT DenseNet projects, and the Coordinator of the UC Team in the PT2020 MobiWise and Mobilizer 5G projects. Her research interests include resilience and quality of service in 5G networks, the Internet of Things, and communications in the cloud.

Dr. Curado is a member of the Editorial Board of *Computer Networks* (Elsevier), *Computer Communications* (Elsevier), *Transactions on Emerging Telecommunications Technologies* (Wiley), and *Internet Technology Letters* (Wiley), and has been involved in the scientific organization and coordination of several international conferences, such as WoWMoM, IM, ACM SAC, IoT-SoS, WMNC, and CloudNet.

EDMUNDO MONTEIRO (Senior Member, IEEE) graduated in electrical engineering (informatics specialty) from the University of Coimbra, in 1984. He received the Ph.D. degree in informatics engineering (computer communications) and the Habilitation degree in informatics engineering from the University of Coimbra, in 1996 and 2007, respectively.

He is currently a Full Professor with the Department of Informatics Engineering (DEI), University of Coimbra (UC), Portugal. He is also a Senior Member of the Research Centre for Informatics and Systems of the University of Coimbra (CISUC). He has more than 25 years of research and industry experience in the field of computer communications, wireless technologies, quality of service and experience, network management, and computer security. He participated in many Portuguese and European research projects and initiatives. His publications include six books (authored and edited), several book chapters and journal publications, and over 200 papers in national and international refereed conferences. He has coauthored nine international patents.

Dr. Monteiro is a member of the Ordem dos Engenheiros (the Portuguese Engineering Association), and a Senior Member of the IEEE Communication and Computer Societies and the ACM SIGCOMM. He is also a member of the Editorial Board of *Wireless Networks* (Springer) journals, and involved in the organization of many national and international conferences and workshops.