

# Availability-aware Service Placement Policy in Fog Computing Based on Graph Partitions

Isaac Lera, Carlos Guerrero, and Carlos Juiz, *Senior Member, IEEE*

**Abstract**—This paper presents a policy for service placement of fog applications inspired on complex networks and graph theory. We propose a twofold partition process based on communities for the partition of the fog devices and based on transitive closures for the application services partition. The allocation of the services is performed sequentially by, firstly, mapping applications to device communities and, secondly, mapping service transitive closures to fog devices in the community. The underlying idea is to place as many inter-related services as possible in the most nearby devices to the users. The optimization objectives are the availability of the applications and the Quality of Service (QoS) of the system, measured as the number of requests that are executed before the application deadlines. We compared our solution with an Integer Linear Programming approach, and the simulation results showed that our proposal obtains higher QoS and availability when fails in the nodes are considered.

**Index Terms**—Fog computing, Service placement, Service availability, Performance optimization, Complex network communities, Graph transitive closures.

## 1 INTRODUCTION

FOG computing has emerged as a suitable solution for the increase of application execution time and network usage that Internet of Things applications based on cloud services generate. This paradigm establishes that the in-network devices are provided with computational and storage capacities, and it enables them to allocate or execute services of the IoT applications that are commonly executed in the cloud provider [1]. By this, the application services are placed closer to the users (or IoT) devices and, consequently, the network latency between users and services and the network usage are reduced. Nevertheless, the limited capacities of the in-network devices, also known as fog devices in this domain, make the definition of management policies even more necessary than in other distributed system such as cloud computing.

The objective of our work is to study an application service placement policy to maximize service availability in case of failures. The placement consist on the selection of the most suitable fog devices to map service instances. We consider that the IoT applications are defined as a set of interrelated services that are initially and permanently deployed on the cloud provider, but that they can be horizontally scaled by creating new stateless instances in the fog devices. We also consider that the users of our domain are unalterable connected to a same gateway or access point, i.e., we consider that our users are IoT devices such as sensors or actuators, instead of considering mobility patterns, as for example in the case of mobile users.

We propose a two phases policy that is addressed to optimize the service availability, in terms of reachability of the services from the IoT devices, and the deadline satisfaction ratios, in terms of the percentage of requests that obtain the application responses before their deadlines.

In the first phase, the policy maps applications (the complete set of interrelated services) to a set of well-connected devices to guarantee the availability of the application for the users connected to that set. We propose to use the community structure of the fog devices for the generation of the partitions of those devices. Once that an application is mapped to a fog community, a second allocation process is performed, by mapping the services of the application to the fog devices in the community. This second phase addresses the optimization of the response time by prioritizing the allocation of interrelated services in the same fog device. We propose to partition the services of an application by using the transitive closure of a service to determine the services to be placed together in the same device.

Fog service placement problem has been addressed in previous researches, even considering community-based approaches [2], but we address some features that have not been previously considered, and the novel contributions of our approach are:

- The combination of the use of complex network communities for the device partition and service transitive closures for the application partition, that has not been used simultaneously in previous studies.
- The optimization of both the application deadline satisfaction, considered in some previous studies, and the application availability, not included in previous studies, and their evolution along the simulation.
- An experimental validation that includes dynamic fails of the infrastructure along the simulation.

## 2 RELATED WORK

The problem of the optimization of service placement in a fog architecture has been previously addressed from several different perspectives, by considering algorithm proposals such as genetic algorithms [3], [4], Montecarlo methods [5], distributed solutions [6], Petri Nets [7], Markov

• The authors are with the Computer Science Department, Balearic Islands University, Palma, SPAIN, E07122.  
Corresponding author: Carlos Guerrero E-mail: carlos.guerrero@uib.es

processes [8], and being linear programming one of the most common solutions [9], [10], [11], [12], [13], [14].

Nevertheless, there is still room for improvement and some research challenges have not been still covered. For example, most of the previous solutions have included the optimization of response time, power consumption, cost, or network usage. But to the best of our knowledge, they have not studied the availability and the influence of failures in the infrastructure.

The use of the community relationship of the devices of a distributed system for the optimization of the resource management was initially proposed by Filiposka et al. [15], and they applied it in the optimization of the allocation of virtual machines in a datacenter to optimize the hop distances between related virtual machines. In the field of fog computing, the use of other topological features of graphs and complex network was proposed at a later stage, such as centrality indexes for the static definition of fog colonies [16] or the placement of data in fog devices [17].

The idea of organizing the complex structure of a fog architecture have been applied in several studies, where the authors defined these static infrastructure organizations as fog colonies [4], micro-clouds [18], Foglets [19], or fog domains [20]. For example, Skarlat et al. [4] defined a twofold distributed placement policy that first considered if a service should be allocated in a fog colony or migrated to the neighbor colony. Once that the colony was chosen, the control node of the colony decided the device that allocated the service. In all those studies, the partition of the fog devices was static and unique for all the applications.

On the contrary, Filiposka, Mishev and Gilly proposed a virtual partition of the devices that is specific for each application and it is dynamically established by the conditions of the system. They implemented an evolution of the proposal in [15] for the case of allocation of virtual machines (VM) into fog devices [2]. They considered that the fog services were encapsulated in one VM and they proposed a two phases optimization process, where in the first step the VM is mapped to a device community, and in the second step, the VM is allocated in any of the devices in the community with a traditional optimization technique. This is probably the most similar work to our proposal in terms of the optimization algorithm, but with a different optimization objective. Their objective was to propose a run-time algorithm for the migration of the VM as mobile user of the applications move through different access points to reduce the average service delay.

The main differences of the work of Filiposka et al. with our proposal are: first, we study the suitability of the community relationships to improve service availability instead of the migration of VMs due to the user mobility; second, we consider a more complex structure of the applications because we defined them as a set of interrelated services that can be allocated in different devices, while they defined the applications as a single encapsulating element, the VM; third, we also study the use of a graph partitioning approach, the transitive closure of the services, for the allocation of the services inside the communities to also benefit the placement of the most interrelated services in the same devices to reduce the network delays between interrelated services.

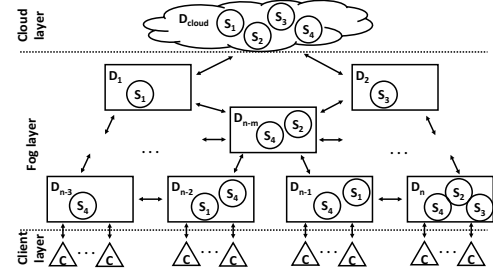


Fig. 1. Fog computing architecture.

### 3 PROBLEM STATEMENT

A general fog computing architecture is represented in Fig. 1 where three layers can be identified: cloud layer, fog layer and client layer. Three types of devices can be differentiated: a device for the cloud provider of the cloud layer; the gateways, that are the access points for the clients; the fog devices, the network devices between the cloud provider and the gateways. All the devices have resources to allocate and execute services.

The fog infrastructure can be modeled as a graph where the nodes are the devices and the edges the direct network links between devices. We identify those devices as  $D_i$ , considering two special cases for the cloud provider ( $D_i^{cloud}$ ) and the gateways ( $D_i^{gtw}$ ). The devices are defined by the available capacity of their resources  $AR_{D_i}$ , that is a vector which contains the capacities of each physical component. For the sake of simplicity, we have considered a scalar value, but it could easily be extended by including as many elements as necessary. We suppose unlimited resources for the specific case of the cloud provider,  $AR_{D_i^{cloud}} = \infty$ . The devices are also defined by the processing speed  $IPT_{D_i}$ , measured in term of instructions per unit of time. The network links are identified by the two connected nodes  $NL_{D_i, D_j}$ , and we consider that it is a bidirectional communication,  $NL_{D_i, D_j} = NL_{D_j, D_i}$ . The network links are defined by the propagation delay,  $PR_{NL_{D_i, D_j}}$ , and the network bandwidth,  $BW_{NL_{D_i, D_j}}$ . Thus, the network delay,  $ND_{NL_{D_i, D_j}}$ , for the transmission of a packet between two connected devices is calculated as:

$$ND_{NL_{D_i, D_j}} = PR_{NL_{D_i, D_j}} + \frac{size}{BW_{NL_{D_i, D_j}}} \quad (1)$$

where *size* is the size of the packet to be transmitted.

The applications in our problem domain follow a microservice based development pattern, that is increasingly being used in IoT applications [21], [22], [23]. This type of applications are modeled as a set of small and stateless services that interoperate between them to accomplish a complex task [24]. Thus, the services can be easily scale up, by downloading the encapsulating element and executing it, or scale down, by just stopping and removing instances of the service. We assume that there is at least one instance of each service running in the cloud provider ( $D_i^{cloud}$ ).

We model each application  $APP_x$  as a directed graph, where the nodes are the services and the edges are the request messages between the services. We identify the services as  $S_u$  and they are defined by the resource consumption generated in the device that allocates the service,

$CR_{S_u}$ . As in the case of the available resources in a device, the resource consumption is generally defined as a vector which measures the consumption of each physical component, but we have considered a scalar value for a simpler definition of the problem. Services are executed when a request message is received. We classify the services in two types depending on the origin of the service request: the entry-point service  $S_u^{ep}$ , the origins of the request messages that arrive to those services are users  $US_a$  or IoT devices (sensors typically)  $ID_b$ ; the intra-services  $S_u^{intra}$ , that are only requested by other services. An intra-service can be requested for several different services and the entry-point service can be requested for several users or IoT devices. But, we suppose that there is only one entry-point service for each application.

The task performed by a service is different depending on the requester, so the execution generated by a request not only depends on the service but also on the requester, i.e. the request message. The request messages are identified by the origin and target services,  $MS_{S_u, S_v}$ , and they are modeled as unidirectional edges,  $MS_{S_u, S_v} \neq MS_{S_v, S_u}$ . The requests generated by the users or the IoT services, i.e. the requests to the entry-point services, are only identified by the target entry-point service  $MS_{\emptyset, S_u}$ .

The request messages are defined by the size of the request message  $SZ_{MS_{S_u, S_v}}$ , that determines the transmission time of the service request, and the execution load that the target service will generate in the device, defined by the number of instructions to be executed,  $EI_{MS_{S_u, S_v}}$ .

We assume that there is at least one instance of each service in the cloud provider. But those services can be horizontally scaled by deploying new instances in the fog devices. By this, the workload can be distributed between instances and the network delay from the user to the service is reduced. We define a placement matrix,  $P$ , of size  $|S_u| \times |D_i|$ , number of services per number of fog devices, where a element  $p_{ui}$  is equal 1 if service  $S_u$  is deployed in device  $D_i$ , and 0 otherwise.

The placement of the services are constrained by the device resource capacity. The resources consumed by the allocated services should not exceed the available resources in the device:

$$\sum_{u=1}^{|S_u|} (p_{ui} \times CR_{S_u}) \leq AR_{D_i}, \forall D_i \quad (2)$$

Our optimization objectives are to increase the application deadline satisfaction ratio, and the application availability as the devices or the network links fail.

We define the deadline satisfaction ratio as the percentage of application requests that are processed before the application deadline. Consequently, the applications in the system,  $APP_x$ , need to be defined by their deadlines,  $DL_{APP_x}$ . The user perceived response time,  $RT_{RQ_{US_a, APP_x}^n}$ , is the metric that measures the time between a specific application request is sent by the user ( $RQ_{US_a, APP_x}^n$ ) and all the application services finish their execution. It includes the network delay of the request between services and the response times (execution and waiting time) of the services.

The equation for the deadline satisfaction ratio is:

$$deadline(US_a, APP_x) = \frac{|RT_{RQ_{US_a, APP_x}^n} < DL_{APP_x}|}{|RQ_{US_a, APP_x}^n|} \quad (3)$$

where  $|RQ_{US_a, APP_x}^n|$  is the number of times that a request for  $APP_x$  is sent from user  $US_a$ , and  $|RT_{RQ_{US_a, APP_x}^n} < DL_{APP_x}|$  is the number of those requests that satisfied the application deadline. This metric can be generalized by considering the request to an application from any user,  $deadline(APP_x)$ , or the ratio for all the applications and users in the system,  $deadline(system)$ .

Our second objective, the application availability, is defined as the ratio of users that are able to reach all the services of the applications they request for a given point in time. In a hypothetic case, where any of the elements in the system fails, the service availability would be 1.0. But the devices or the network links can fall down, breaking the shortest paths between the users and the application services. At best, this only would generate an increase in the network delay due to the requests would be routed by a longer path, damaging the deadline satisfaction ratios. But it could even result in making the user impossible to reach all the application services, damaging the service availability ratio. The equation for the service availability ratios is:

$$availability(APP_x) = \frac{|US_a, g.t. \exists \text{ path } US_a \text{ to } APP_x|}{|US_a, g.t. US_a \text{ requests } APP_x|} \quad (4)$$

In summary, our domain problem is addressed to find  $P, p_{ui} \forall S_u, D_i$  by minimizing  $deadline(US_a, APP_x) \wedge (1 - availability(APP_x)) \forall US_a, APP_x$  subject to the constraint in Eq.(2).

## 4 TWO PHASES PARTITION-BASED OPTIMIZATION PROPOSAL

Our optimization algorithm is based on a two phases placement process with a first mapping of applications in fog communities and a second phase which allocates the services of an application in the devices of a fog community. We partition the fog devices using the community relationship of the complex network that models the network infrastructure of the system. The application services are partitioned considering the transitive closures of the nodes that represent the services in the application graph.

We study if the community relationships of the fog devices is a good indicator to detect device sets that guarantee the availability of the services and the reachability of the devices when device and network links failures are considered. Additionally, we also study if the transitive closure of a service is a good indicator to decide the services that are allocated in the same device to avoid network communications overheads.

### 4.1 Community-based Fog Devices Partition

The first phase of our optimization algorithm deals with the mapping between applications (a set of interrelated services) and a device partitioning. We propose to partition the devices with the use of the community relationship between them. This phase of our optimization algorithm is based

on the previous work of Filiposka, Mishev and Gilly, where they studied and validated community-based algorithms for placement optimization in cloud computing [15] and in fog computing [2].

The community structure is a topological feature of graphs that determines the sets of nodes which are better connected between them than with the rest of the network. The most popular community detection method is the one proposed by Girvan and Newman [25], which detects communities by progressively removing edges from the original graph. The algorithm removes the edges with the highest betweenness centrality, at each step. Betweenness centrality of an edge is the sum of the fraction of the shortest paths that pass through the edge. Therefore, a community, that is organized with two regions that are mainly communicated by only one edge, is split into two new communities in each algorithm iteration.

Under the conditions of our domain problem, a device community can be understood as a set of devices that are well connected between them, with alternatives communication paths, and that the shortest paths between devices are evenly distributed between the topology. Consequently, a fail in an edge inside the community will have a lower influence in the communication paths between devices than a fail in the edges that connect the communities. This lower influence means that the fails inside the communities will not generate isolated regions in the topology neither an important increase in the communication delays.

The Girvan-Newman method iteratively determines the communities and the dendrogram, the tree structure of the communities, can be built. We characterized those communities with its depth in the dendrogram. We define this depth as the iteration in which the community was obtained. The higher the depth value is, the better communicated the device community is. Consequently, from the point of view of the availability, it is better to place the applications in device communities with higher depth values, since the devices inside those communities are better communicated between them than the devices in communities with lower depths values [26].

For example, consider the fog infrastructure in Fig. 2. The network link  $NL_{D_c, D_f}$  is the one with the highest edge betweenness centrality since it is passed through the highest number of shortest paths. If we iterate the Girvan-Newman method over this example, communities 2 and 3 have higher depth values than community 1 since they are obtained when  $NL_{D_c, D_f}$  is removed in the next iteration of the community generation algorithm. Consider also that we deploy an application with services  $S_i$  and  $S_j$  in community 1, allocating  $S_i$  in  $D_a$  and  $S_j$  in  $D_h$ , and that the user that requests the application is connected to device  $D_b$ . Under those conditions, a fail in  $NL_{D_c, D_f}$  would make impossible to finish the execution of the application since their services are unreachable. On the contrary, if we deploy the application in community 2, any fail in a edge would not make impossible to execute the application. Finally consider that a second user is connected to device  $D_h$ . The best alternative, from the point of view of the availability, would be to horizontally scale up by deploying the same application twice in both communities 2 and 3, than only once in any of them.

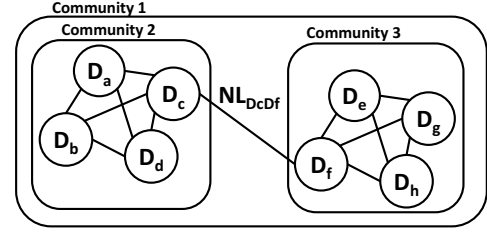


Fig. 2. Example of fog device communities.

This example shows that, in an unrealistic situation with unlimited resources in all the devices, the best option would be to deploy an instance of the application for each client that requests it and this deployment would be placed in the community with the highest depth value that includes the device where the client is connected to. But this cannot be performed due to the limited resources in the devices of a community. Moreover, if we note that the higher the depth value of the community, i.e., the smaller the number of devices in the community, the communities with the highest values are the ones formed by only one device. Consequently, it is necessary to prioritize the allocation of the applications in the communities. We propose to use a greedy algorithm for this prioritization, more concretely, the First-Fit Decreasing algorithm [27].

Our optimization algorithm deals, in this first step, with the placement of applications in device communities using a First-Fit Decreasing approach. The priority criteria for ordering the applications is their execution deadlines, by prioritizing the applications with shortest deadlines. The algorithm starts checking the allocation of the application from the device communities with highest depth to the ones with the lowest, and the application is allocated in the first community with enough resources to allocate all the services of the application. If after checking all the communities, the application has not been allocated, this will be available only in the cloud provider. The process for the same application is repeated as many times as the number of users in the system that request this application. Algorithm 1 shows the pseudo-code of our proposal. The algorithm goes through the applications (in ascending deadline order), the users that request them and the communities (in descending depth order), trying to allocate the services of the application in the devices in the community.

In this first step, we map the applications in communities, but the map of services remains to be defined. We separate the process in two steps because we mainly focus the first one (mapping applications to communities) on increasing the application availability, and the second one (mapping services of an application to devices in a device community) on the application deadlines. This second step is performed by the function *placeServicesInDevices()*, in line 15, and its details are explained in Section 4.2 and Algorithm 2.

Our algorithm checks if an application has been previously placed in a community (line 11), and if not, it delegates the decision to place the application to the community to the algorithm which checks if the application services fit into the device community (Algorithm 2).

### Algorithm 1 Device community-based application allocation

```

1:  $\mathbb{C} \leftarrow$  calculate device communities
2:  $\mathbb{IC} \leftarrow$  order communities  $\mathbb{C}$  by descending depth
3:  $\mathbb{A} \leftarrow$  order applications by ascending deadline
4:  $appPlacement \leftarrow \emptyset$ 
5: for  $app$  in  $\mathbb{A}$  do
6:    $\mathbb{U} \leftarrow$  get users requesting application  $app$ 
7:   for  $user$  in  $\mathbb{U}$  do
8:      $dev \leftarrow$  get device where  $user$  is connected
9:     for  $infCom$  in  $\mathbb{IC}$  do
10:      if  $dev \in infCom$  then
11:        if  $infCom \in appPlacement[app]$  then
12:          "application  $app$  already placed in community  $infCom$ "
13:          break
14:        else
15:          if  $placeServicesInDevices(app, infCom)$  then
16:             $appPlacement[app].append(infCom)$ 
17:            update resource usages in  $infCom$ 
18:            "placed application  $app$  in community  $infCom$ "
19:            break

```

## 4.2 Transitive Closure-based Application Partition

Once that the mapping of a given application into a candidate community of devices is performed by the first phase of the optimization algorithm, the second phase deals with the allocation of the services of the application into the devices in the community. We first partition the applications into sets of services, and it is checked if each of those service sets can be placed in just one device. If not, smaller sets are considered. The partition of the service into sets is based on our previous work [6], where we studied and validated the use of a distributed placement algorithm where the service sets are created by considering the transitive closure of the services in the application graph.

The transitive closure of a directed graph indicates the nodes that are reachable for each of the nodes in the graph. If a vertex  $j$  is reachable by a vertex  $i$  means that there is a path from  $i$  to  $j$ . The reachability matrix of a graph is called the transitive closure of the graph, and the set of reachable nodes for a given node is called the transitive closure of a node [28].

Under the conditions of our domain problem, the transitive closure of a node can be understood as the set of services that are requested for the execution of the given service, i.e., the outgoing requests generated by a service when it receives an incoming request. If we are interested in reducing the response time of the application execution, the services of the transitive closure should be allocated in the same device to reduce the communication delays between them, since the network delay is 0.0 for request messages inside the same device. Moreover, the best case is when all the services of an application are allocated in the same device, but this is limited by the resource constraint (Equation 2).

We also propose a First-Fit algorithm for this second phase (Algorithm 2), which orders the sets of services from the ones with the biggest sizes (only one transitive closure with all the services) to the smallest sets of services (the transitive closures with only one node or with the loops in the service flow), and tries to place those sets of services into a same device. The devices are ordered by a fitness value which is the theoretical user perceived response time. This value is obtained by adding the network latency between the device and the user and the execution time of all the

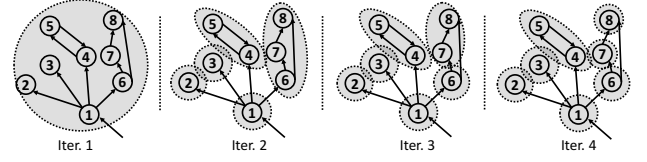


Fig. 3. Example of service transitive closures.

### Algorithm 2 Transitive closure-based service allocation

```

1: function  $PLACE\_SERVICES\_IN\_DEVICES$ 
2:    $TC \leftarrow$  generate transitive closure partitions for  $app$ 
3:    $\mathbb{D} \leftarrow$  order devices in  $infCom$  by response time
4:    $SP \leftarrow \emptyset$  /* Services already placed */
5:    $servPlacement \leftarrow \emptyset$ 
6:   for  $dev$  in  $\mathbb{D}$  do
7:     for  $appPartition$  in  $TC$  do
8:       for  $closure$  in  $appPartition$  do
9:         if ( $closure$  not in  $SP$ ) and ( $closure$  fits in  $dev$ ) then
10:           $SP = SP \cup closure$ 
11:          for  $service$  in  $closure$  do
12:             $servPlacement[service] = dev$ 
13:            update resource usages in  $dev$ 
14:          if  $SP == app$  then
15:            return True,  $servPlacement$ 
16:   return False,  $\emptyset$ 

```

services in the device. This prioritizes the devices that are both closer to the users and faster in the execution. By this, the second step of the algorithm optimizes the user perceived response time, and, consequently, improves the deadline satisfaction ratio.

Initially, Algorithm 2 goes through the devices ordered by the fitness value, and tries to allocate as much services as possible in the devices with the highest values. For the first device, it first tries to allocate all the services of the application. If they do not fit, the service set is split in several sets, one for each entry-point service and one additional set for the transitive closures of each of its neighbor services of the entry-point one, and it checks if any of those new sets fits in the first device. This is recursively repeated for each transitive closure set that contains services not previously allocated. Fig. 3 shows an example of how the transitive closure of the services is generated along the iterations of the algorithm that partition the services of the application.

Once that all the service sets have been evaluated to be placed in the first device, this process is sequentially repeated for all the devices for the unallocated services. If after considering all the devices, there are still unallocated services, the mapping of the application in the current device community is rejected. Consequently, the first phase of the algorithm has to consider a greater community for the placement.

## 5 EXPERIMENTAL EVALUATION

We defined random characteristics for the elements of our simulation experiments. We modeled the parameters of the elements in the domain with uniform distributions and the minimum and maximum values are shown in Table 1.

The service applications were generated randomly following a growing network (GN) graph structure. GN graphs are built by adding nodes one at a time with a link to one previously added node. The network infrastructure was created as a random Barabasi-Albert network with 100 nodes

TABLE 1  
Values of the parameters for the experiment characterization

Parameter		min.–max.
<b>Network</b>		
Propagation time (ms)	$PR_{NLD_i,D_j}$	5
Bandwidth (bytes/ms)	$BW_{NLD_i,D_j}$	75000
<b>Fog device</b>		
Resources (res. units)	$AR_{D_i}$	10–25
Speed (Intrs/ms)	$IP_{TD_i}$	100–1000
<b>Application</b>		
Deadline (ms)	$DL_{APP_x}$	300–50000
Services (number)		2–10
Resources (res. units)	$CR_{S_u}$	1–6
Execution (Intrs/req)	$EI_{MS_{S_u},S_v}$	20000–60000
Message size (bytes)	$SZ_{MS_{S_u},S_v}$	1500000–4500000
<b>IoT device</b>		
Request rate (1/ms)		1/1000–1/200
Popularity (prob.)		0.25

devices. Betweenness centrality index is a topological metric that measures the number of shortest path that goes through a device. The gateway devices were selected from the nodes placed in the edges of the network, i.e., the nodes with the smallest betweenness centrality indices. Betweenness centrality index is a topological metric that measures the number of shortest path that goes through a device. We selected the 25% of devices with the lowest centrality value to behave as gateways (25 gateways). The number and the applications requested from the IoT devices connected to the gateways were determined with a popularity distribution modeled with an uniform distribution.

The random experimental scenario finally resulted on 20 applications with 106 services, that totally needed 360 resource units and the fog devices were able to offer up to 1874 resources units. 70 IoT devices (or users) were deployed and they generated an application request each 1/557 ms in average.

We compared the results of our proposal with the ones obtained from the implementation of an integer linear programming (ILP) service allocation optimizer. As we mention in Section 2, ILP solutions are the most numerous in fog service placement optimization.

The experiments were executed using the YAFS simulator that we had previously developed for other research works. This simulator is able to include graph-based network topologies and pluggable fog service placement policies, apart from other features that, to the best of our knowledge, are not provided by other fog simulators, such as node failures, or dynamic service placement and routing. The simulator is open source and it can be downloaded from its code repository [29].

The experiment results are presented and analyzed in two separated sections. Section 5.1 includes the analysis of the results obtained with the YAFS simulator. Those results compare the user perceived response time and the availability of the applications for the IoT devices. In Section 5.2, it is presented an analysis of the service placement obtained with both optimization policies (our proposal and the ILP

one).

## 5.1 Simulation Results

A first simulation scenario included fails in the fog devices to study the availability of the services when the nodes are getting down. The simulation included random and permanent fails in the nodes, starting with all the devices (100 nodes) alive, and finishing the simulation with fails in all of them. The fails were generated uniformly along the simulation. The results of this simulation are presented in Fig. 4 and shows the QoS in terms of the total number of requests that are executed satisfying the application deadline. The reason because a request does not satisfied the deadline can be both due to the response time is higher than the deadline or due to none device with the services of the requested application are reachable from the IoT device due to all the paths between them have failed devices. Three data series are represented in Fig. 4: one for the total number of requests that are sent from the IoT devices (labeled with *Total num. of requests*), one for the number of requests that are executed before the deadline when the placement of our solution is considered (labeled with *Partition*); and the number of requests that satisfied the deadline with the ILP policy (labeled with *ILP*).

It is observed that our approach results in a higher number of satisfied requests, mainly during the first half of the simulation (up to 50 failed devices). In the second part of the simulation, improvements in the QoS are also observed but these are less significant in regard with the ILP.

For the sake of a deeper analysis of the availability, it has been also measured in terms of the number of IoT devices that are able to request their applications thank to that all the services they need are reachable with network paths without failed devices. This is represented in Fig. 5, where the y-axis are the number of IoT devices that are able to request their applications, and the x-axis the number of devices that have failed. The figure also includes the hypothetical and impossible case, due to the resource limit constraint, of allocating all the services in the gateways (labeled as *All in gtw.*). This is the best case and is useful to compare the solutions with the best upper bound. These results confirm that our proposal is able to increase the availability of the system when fails happen in the fog devices.

A second simulation scenario did not included fails in the fog devices and was used to study the user perceived response time of the applications. These response times were measured as the time between the user request was generated in the IoT device and all the application services finished. The results were measured independently for each pair application-IoT device. They are summarized in Fig. 6. Each plot in the figure represents the response times of an application, an each item in the x-axis corresponds to one gateway that has an IoT device (or user) that request the application. The results of our solution are labeled as *Partition* and the results of the ILP approach are labeled as *ILP*.

It is observed that the placement obtained with our proposal does not reduce the response time for all the applications, but it is shorter for 13 of the 20 applications.

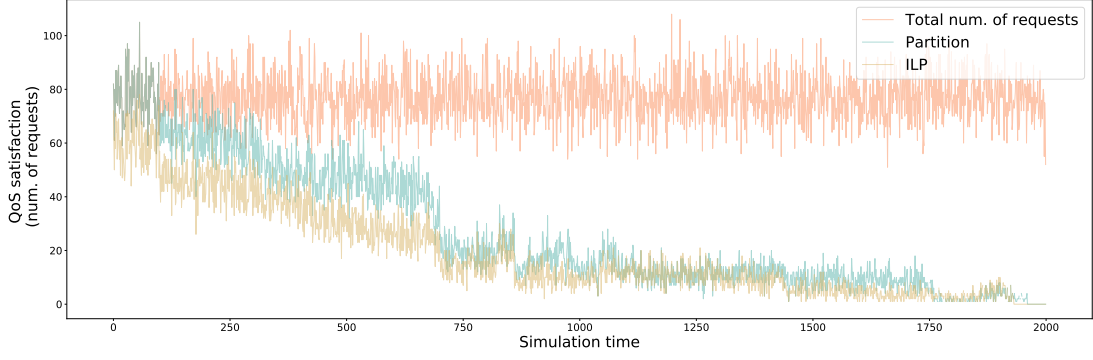


Fig. 4. Evolution of the QoS with regard to the fail of fog devices, in terms of the number of requests which satisfy application deadlines ( $|RT_{RQ_{US_a, APP_x}}^n| < DL_{APP_x}|$ ) compared with the total number of requests ( $|RQ_{US_a, APP_x}^n|$ ).

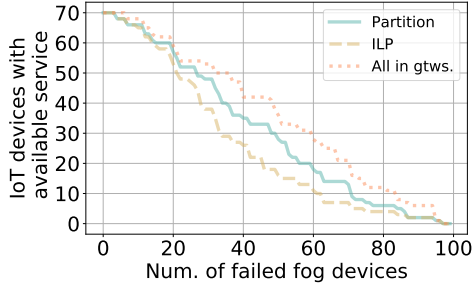


Fig. 5. Number of IoT devices that get services in regard with the number of failed fog devices ( $availability(APP_x)$ ).

Additionally, we can observed that in some applications an important damage of the response time is obtained. This is explained because both policies prioritize applications with shorter deadlines in front of the ones with longer deadlines. Nevertheless, there are less of these extreme cases, and with shorter times, when our policy is used: our policy only damages application 15 with a time of around 1000 ms, in front of four applications up to 400 s with the ILP policy (around 400000 ms for application 1, 300000 ms for application 8, 200000 ms for application 12, and 70000 for application 2).

In summary, our service placement policy shows a better behavior in terms of availability of the services that also results on a better QoS in the system. On the contrary, the response time of some applications results damaged but this behavior is also observed with the ILP policy, generating even worse response times.

## 5.2 Placement Results

This section is devoted to compare the placement of the services obtained from the execution of our algorithm with regard to the ILP one. This analysis is included to give a brief idea of how the services are spread across the fog devices.

Firstly, Fig. 7a shows that the placement of the services differs a lot between both placement policies. A mark in the plot of the figure indicates that a given service (y-axes) is placed in a given device (x-axes). Taking into account that the services of the same application have consecutive identifiers, it is also observed that in the case of our policy

(*Partition*), there are more cases of devices that allocate several services of the same application (consecutive marks in the same device).

Fig. 7b represent the resource usage of the fog devices. The y-axis represents the percentage of resources that are used by the services allocated in a given device and the x-axis are the devices ordered by these percentages in ascending order. By the analysis of the figure, we can observe that in the placement of our solution, there are almost the double of nodes that do not allocate any service (the resource usage is 0.0), and there is not any device that is fully used (resource usage of 1.0), with regard to the case of the ILP where almost 40 devices have a 100% usage of the resources. The first interpretation of these results is that the scale level of our solution is smaller than the ILP one, in fact, we calculated that our policy deployed 357 (and 1161 resource units) instances of the services and the ILP deployed 374 (and 1203 resource units), around 5% more services (3.6% more resources). Consequently, our solution is able to obtain better QoS and availability with a lower use of the fog resources (smaller number of instances). The second interpretation is that the services are more evenly distributed, since the workload of the devices is smaller, avoiding the saturation of the devices and keeping the system in a more flexible state in order to allocate new service instances.

Finally, Fig. 7c shows the relationship between the service placement and the hop distance between the allocated service and the IoT device that requests it. A point in the scatter plot indicates how many IoT devices has a given distance with a service of the application they request. For example, in the case of our policy, there are around 100 services that are allocated in the gateways where the IoT devices are connected (a hop distance of 0.0). On the contrary, the ILP policy allocates more than 160 services in the gateways, the point (0,160) in the plot. We observe that the services are distributed more evenly and placed further from the gateways (higher distances) for the case our policy. Consequently, the ILP is able to place the services closer to the IoT devices. Despite this, our policy shows a better general behavior also in terms of application response time.



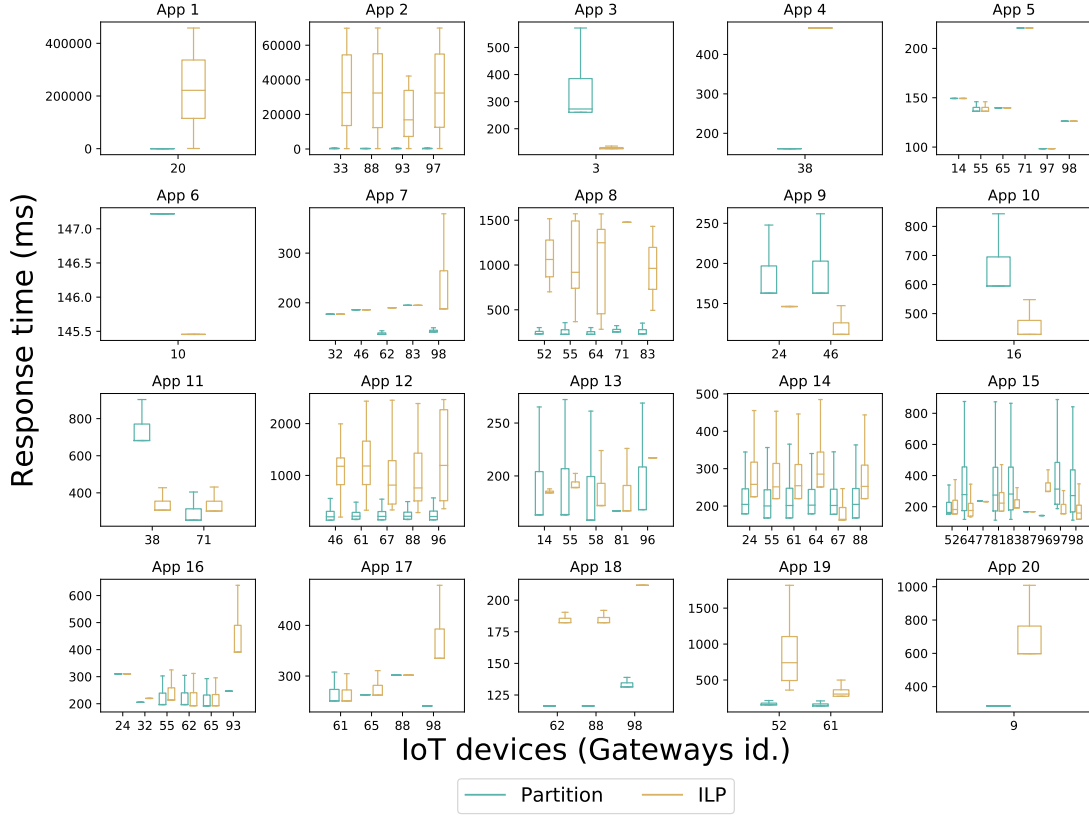


Fig. 6. User perceived response times of the applications for each user (or IoT device) in the system ( $RT_{RQ_{US_a,APP_x}^n}$ ).

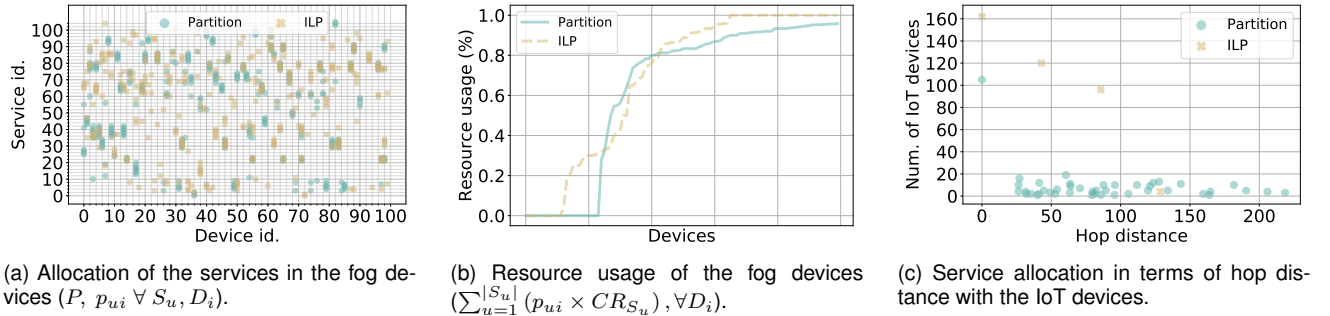


Fig. 7. Comparison of the services placement between our partition-based algorithm and the ILP optimizer.

## 6 CONCLUSION

We have proposed an algorithm for service placement in fog devices based on the partition of the fog devices (into communities) and the services of the applications (into transitive closures) for the optimization of the QoS of the system and the service availability for the users (or IoT devices).

Two simulation scenarios have been executed, one including fails in the fog devices and another one without fails, to measure the response time of the applications, the service availability and the number of request that were served satisfying the application deadlines. The service placement obtained with our policy resulted in a higher QoS and service availability, with regard to the placement

of an ILP-based algorithm. In the case of the user perceived response time, our policy obtained better times for 13 of the total 20 applications. Both policies showed a high degradation of service for some applications, but in the case of the ILP, this degradation happened in more applications and resulting in longer response times.

As future works, the use of complex networks and graph theory for the optimization of other parameters of the systems, such as service cost, network usage, migration cost, and service provider cost could be studied. By the own nature of the proposed policy, the optimization of these other metrics probably would need to be combined with other type of heuristics to obtain suitable results, and consequently, further research is necessary.



## ACKNOWLEDGMENTS

This research was supported by the Spanish Government (Agencia Estatal de Investigación) and the European Commission (Fondo Europeo de Desarrollo Regional) through grant number TIN2017-88547-P (MINECO/AEI/FEDER, UE).

## REFERENCES

- [1] O. Consortium *et al.*, "Openfog reference architecture for fog computing," Tech. Rep., February, Tech. Rep., 2017.
- [2] S. Filiposka, A. Mishev, and K. Gilly, "Community-based allocation and migration strategies for fog computing," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, April 2018, pp. 1–6.
- [3] Z. Wen, R. Yang, P. Garraghan, T. Lin, J. Xu, and M. Rovatsos, "Fog orchestration for internet of things services," *IEEE Internet Computing*, vol. 21, no. 2, pp. 16–24, Mar 2017.
- [4] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized iot service placement in the fog," *Service Oriented Computing and Applications*, Oct 2017. [Online]. Available: <https://doi.org/10.1007/s11761-017-0219-8>
- [5] 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.
- [6] 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*, Jun 2018. [Online]. Available: <https://doi.org/10.1007/s12652-018-0914-0>
- [7] L. Ni, J. Zhang, C. Jiang, C. Yan, and K. Yu, "Resource allocation strategy in fog computing based on priced timed petri nets," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1216–1228, Oct 2017.
- [8] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, vol. 91, no. Supplement C, pp. 205 – 228, 2015, special Issue: Performance 2015. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0166531615000619>
- [9] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, "Cost efficient resource management in fog computing supported medical cyber-physical system," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 1, pp. 108–119, Jan 2017.
- [10] K. Velasquez, D. P. 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. [Online]. Available: <https://doi.org/10.1007/s12243-016-0524-9>
- [11] Z. Huang, K.-J. Lin, S.-Y. Yu, and J. Y. jen Hsu, "Co-locating services in iot systems to minimize the communication energy cost," *Journal of Innovation in Digital Ecosystems*, vol. 1, no. 1, pp. 47 – 57, 2014. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S2352664515000061>
- [12] L. Yang, J. Cao, G. Liang, and X. Han, "Cost aware service placement and load dispatching in mobile cloud systems," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1440–1452, May 2016.
- [13] V. B. C. Souza, W. Ramírez, X. Masip-Bruin, E. Marín-Tordera, G. Ren, and G. Tashakor, "Handling service allocation in combined fog-cloud scenarios," in *2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–5.
- [14] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3702–3712, Dec 2016.
- [15] S. Filiposka, A. Mishev, and C. Juiz, "Community-based vm placement framework," *The Journal of Supercomputing*, vol. 71, no. 12, pp. 4504–4528, Dec 2015. [Online]. Available: <https://doi.org/10.1007/s11227-015-1546-1>
- [16] C. Guerrero, I. Lera, and C. Juiz, "On the influence of fog colonies partitioning in fog application makespan," in *2019 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, August 2018.
- [17] I. Lera, C. Guerrero, and C. Juiz, "Comparing centrality indices for network usage optimization of data placement policies in fog devices," in *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, April 2018, pp. 115–122.
- [18] Y. Elkhatib, B. Porter, H. B. Ribeiro, M. F. Zhani, J. Qadir, and E. Rivière, "On using micro-clouds to deliver the fog," *IEEE Internet Computing*, vol. 21, no. 2, pp. 8–15, Mar 2017.
- [19] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, *Fog Computing: A Platform for Internet of Things and Analytics*. Cham: Springer International Publishing, 2014, pp. 169–186.
- [20] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue, "On reducing iot service delay via fog offloading," *IEEE Internet of Things Journal*, vol. PP, no. 99, pp. 1–1, 2018.
- [21] M. Vogler, J. M. Schleicher, C. Inzinger, and S. Dustdar, "A scalable framework for provisioning large-scale iot deployments," *ACM Trans. Internet Technol.*, vol. 16, no. 2, pp. 11:1–11:20, Mar. 2016. [Online]. Available: <http://doi.acm.org/10.1145/2850416>
- [22] A. Krylovskiy, M. Jahn, and E. Patti, "Designing a smart city internet of things platform with microservice architecture," in *2015 3rd International Conference on Future Internet of Things and Cloud*, Aug 2015, pp. 25–30.
- [23] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder, "Incremental deployment and migration of geo-distributed situation awareness applications in the fog," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, ser. DEBS '16. New York, NY, USA: ACM, 2016, pp. 258–269. [Online]. Available: <http://doi.acm.org/10.1145/2933267.2933317>
- [24] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables devops: Migration to a cloud-native architecture," *IEEE Software*, vol. 33, no. 3, pp. 42–52, May 2016.
- [25] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E*, vol. 69, no. 2, p. 026113, Feb. 2004. [Online]. Available: <http://link.aps.org/doi/10.1103/PhysRevE.69.026113>
- [26] S. Fortunato, V. Latora, and M. Marchiori, "Method to find community structures based on information centrality," *Phys. Rev. E*, vol. 70, p. 056104, Nov 2004. [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRevE.70.056104>
- [27] A. Alahmadi, A. Alnowiser, M. M. Zhu, D. Che, and P. Ghodous, "Enhanced first-fit decreasing algorithm for energy-aware job scheduling in cloud," in *2014 International Conference on Computational Science and Computational Intelligence*, vol. 2, March 2014, pp. 69–74.
- [28] H. S. Warren Jr, "A modification of warshall's algorithm for the transitive closure of binary relations," *Communications of the ACM*, vol. 18, no. 4, pp. 218–220, 1975.
- [29] I. Lera and C. Guerrero, "Yafs, yet another fog simulator," <https://github.com/acscuib/YAFS>, accessed: 2018-02-03.



**Isaac Lera** received his Ph.D. degree in Computer Engineering at the Balearic Islands University in 2012. He is an assistant professor of Computer Architecture and Technology at the Computer Science Department of the University of the Balearic Islands. His research lines are semantic web, open data, system performance, educational innovation and human mobility. He has authored in several journals and international conferences.



**Carlos Guerrero** received his Ph.D. degree in Computer Engineering at the Balearic Islands University in 2012. He is an assistant professor of Computer Architecture and Technology at the Computer Science Department of the University of the Balearic Islands. His research interests include web performance, resource management, web engineering, and cloud computing. He has authored around 40 papers in international conferences and journals.



**Carlos Juiz** received his Ph.D. degree in Computer Engineering at the Balearic Islands University in 2001. He is an associate professor of Computer Architecture and Technology at the Computer Science Department of the University of the Balearic Islands. His research interests include performance engineering, cloud computing and IT governance. He has authored around 150 papers in different international conferences and journals.