Fog and Cloud Computing Optimization in Mobile IoT Environments

José C. Vieira, António M. Grilo, and João C. Garcia

Abstract—With the surge of ubiquitous demand on high-complexity and quality of mobile IoT services, new computing paradigms have emerged. Motivated by the long and unpredictable end-to-end communication latency experienced in cloud computing as well as the rapid growth of mobile traffic, fog computing emerges as the most comprehensive and natural paradigm to support real-time applications and to get more efficient with the data sent to the cloud. From the performed analysis, the lack of research in dynamic environments regarding both the client and fog nodes is noticeable. Moreover, most of the existing proposed schemes only consider few specific objectives for the system optimization as well as static computing resources. In the present work, a novel fog computing architecture has been designed and evaluated with the purpose of finding a solution to the aforementioned issues. A novel optimization problem formulation is also proposed in order to match the characteristics of the proposed architecture. A set of optimization algorithms were developed to solve the formulated problem. Moreover, the proposed architecture has been successfully implemented in a suitable developed simulation toolkit. The performance of the optimization algorithms was assessed in different static and mobile scenarios using the QoS offered to the clients and the system provider objectives as the main metrics. It is observed that the proposed architecture effectively helps to improve the offered QoS to its users in mobile and static environments while meeting the system provider objectives.

Index Terms—Fog Computing, IoT, Mobile Environments, Multi-Objective Optimization, iFogSim.

1 Introduction

Agility and flexibility of big data applications has been gradually taking the form of the Internet of Things (IoT). Ubiquitous deployment of interconnected devices is estimated to reach 50 billion units by 2020 [1]. This exponential growth is broadly supported by the expanding of the IoT across multiple domains, as it can improve efficiency in a wide variety of applications and markets, such as autonomous transportation, industrial controls, wearables, to name a few. Managing the data generated by IoT sensors and actuators is one of the biggest challenges faced when deploying IoT systems. Although this kind of device has evolved radically in the last years, due to battery life and processing capacity, these are not suitable for running heavy applications, being necessary, in this case, to resort to more powerful computing resources, likely owned by third parties. Cloud computing is a resource-rich environment that has been imperative in expanding the reach and capabilities of IoT devices by enabling clients to outsource the allocation and management of resources to the cloud.

Despite such potential, there are two main problems which remain unresolved. On the one hand, as cloud servers reside in remote data centers, the end-to-end communication may be subject to long and unpredictable delays, which some applications with stringent latency requirements cannot support. On the other hand, with the exponential growing number of IoT devices, the bandwidth required to support them becomes too large for centralized processing.

Motivated by these limitations, fog computing [2] emerges as the most comprehensive and natural paradigm to support real-time applications and to get more efficient with the data sent to the cloud. Fog is a decentralized computing infrastructure that aims to enable computing, storage, networking, and data management along the cloud-to-thing path as data traverses the network towards the cloud, bringing these services closer to where data is created and acted upon [3]. Even though fog computing has been proposed to grant support for IoT applications, it does not replace the needs of cloud-based services (e.g., for massive data processing). In fact, fog and cloud complement each other. Together, they offer services even further optimized to IoT applications.

This cooperation between fog and cloud services has a widely rage of use cases. For instance, drones, also known as Unmanned Aerial Vehicles (UAVs), can be employed in plenty of applications which are difficult or dangerous for humans (e.g., military, disaster or emergency missions). The mode of operation of drones has progressively changed from the traditional remote control by humans to a sophisticated autonomous control, able to operate without human intervention. However, as all processing required to analyze the collected data and make decisions are performed on-board, the flight autonomy decreases substantially. For instance, let us consider an extremely resource intensive application, say search for missing persons in disaster zones using image processing. Fog computing, being able to provide real-time responses, is capable of hosting and process the autonomous control module, and send its output results to the drone. With regard to the image processing module, as it has no such tight time constraints, it can be forwarded to the cloud, which will be responsible for hosting, and processing the collected data (e.g., video, images or heat sensor data) and give its output to the person in charge of the rescue mission. This way, the required processing operations on-board are tremendously reduced, resulting in an increase in terms of the overall duration of the flight. Note that, its up to the orchestrator, based on its objectives,

to decide whether to place the modules in the fog or in the cloud, provided that constraints (e.g., time boundaries) are fulfilled.

Despite the benefits that fog promises to offer, such as low latency, heterogeneity, scalability and mobility, the current model suffers from some limitations that must be overcome. On the one hand, most of the existing literature assumes that the fog nodes are fixed, or only considers the mobility of IoT devices. Less attention has been paid to mobile fog computing and how it can further enhance the QoS perceived by the users and the objectives of the system provider, such as the total energy consumption. On the other hand, most of the existing schemes that are proposed for fog systems, only consider few objectives (e.g., QoS, monetary cost) and assume other objectives do not affect the problem.

1.1 Summary of Contributions

To the best of our knowledge, we are the first to present a formal study for optimizing the resource allocation in the presence of mobile fog environments with mobile client and fog nodes. We jointly consider the costs of QoS, power consumption, processing, bandwidth, and migration in order to optimize the module placement and routing paths both for data dependencies between application modules and module migrations from a system provider perspective. In particular, we make the following five contributions.

We propose a novel fog computing architecture which has been designed and evaluated with the purpose of finding a solution to the aforementioned issues. Fog computing has a generic nature associated with the fact that it does not rely on any specific designed and purposely built technology. Therefore, the proposed architecture, in contrast to most previously proposed ones, is generic regarding all nodes, communications and applications. Additionally, bearing in mind dynamic environments, the proposed architecture also assumes the presence of mobile fog and client nodes. Migration of services between nodes, is supported and can be explored in order to further enhance the QoS perceived by the users in mobile environments.

We build a novel and comprehensive optimization problem formulation in order to match the characteristics of the proposed architecture. As the system may not always be able to ensure every single deadline due to the hardware limitations of fog nodes, from the system provider perspective we formulate two main independent and conflicting objectives. On the one hand, the objective is to grant QoS to as many applications as possible. On the other hand, the objective is to minimize its overall energy consumption, especially when it represents the main source of monetary costs. Meanwhile, bearing in mind the QoS required to support its clients it is also important to keep processor utilization as low as possible, as well as the bandwidth on links.

We develop a set of optimization algorithms to solve the formulated problem. Provided that constraints (e.g., delay bounds) are fulfilled, its up to the orchestrator, based on its predefined objectives, to decide which node should host and execute each user service. This decision-making and its impact onto the system performance were tested using several optimization algorithms including the CPLEX optimizer [4], brute force, random search and genetic algorithms.

We implement the proposed architecture in a suitable developed simulation toolkit. The developed simulation tool significantly extended the existing iFogSim simulator [5], making it more flexible and able to support more complex modelling, with emphasis on client and resource mobility.

We carry out extensive experiments to validate the performance of the optimization algorithms. The performance of the optimization algorithms was assessed in different static and mobile scenarios using the QoS offered to the clients and the system provider objectives as the main metrics. It is observed that the proposed architecture effectively helps to improve the offered QoS to its users in mobile and static environments while meeting the system provider objectives.

1.2 Paper Organization

The remainder of the document is structured as follows. Section 2 describes an overview of the proposed architecture and the main design choices of the developed system. Section 3 describes the proposed problem formulation in order to optimize the system performance. Section 4 provides a comparative evaluation of the performance of the developed algorithms relative to the considered application scenarios. This is done by resorting to the selected QoS metrics collected by means of multiple computer simulations. Section 5 presents the state-of-the-art and a review of relevant works in the context of the objectives described above. Finally, Section 6 concludes the document by summarizing the achieved results. Some future directions to improve the work are also given.

2 Architecture Proposal

The proposed architecture comprises entities, services, and a controller fully aware of the system performance and its characteristics. Based on the latter, the controller is capable of running the decision-making algorithm and, depending on the solution found, order nodes to perform the decided actions. It is also responsible to perform handovers whenever needed in order to ensure that mobile nodes are always connected to their closest fog or cloud node. Each node, according to the central control unit instructions, is then responsible for hosting modules and processing the respective data.

2.1 Entities

The physical infrastructure follows the typical architecture of fog computing depicted in Figure 1. Therefore, it is composed of entities such as fog, cloud and client nodes. As present in [3], fog computing relies in an infrastructure. Thus, there is always a static and interconnected network of fog and cloud nodes. Meanwhile, at the edge of the network there might exist some mobile fog nodes deployed inside buses, trains, etc. Similarly, clients may also be static or mobile. Regarding the communication links, static nodes might be connected to an arbitrary number of nodes using different communication technologies. Meanwhile, mobile nodes are always connected to their closest static fog node via cellular network. In this way, the physical infrastructure is modeled as a directed graph where vertices represent nodes and edges represent network links. Note that nodes

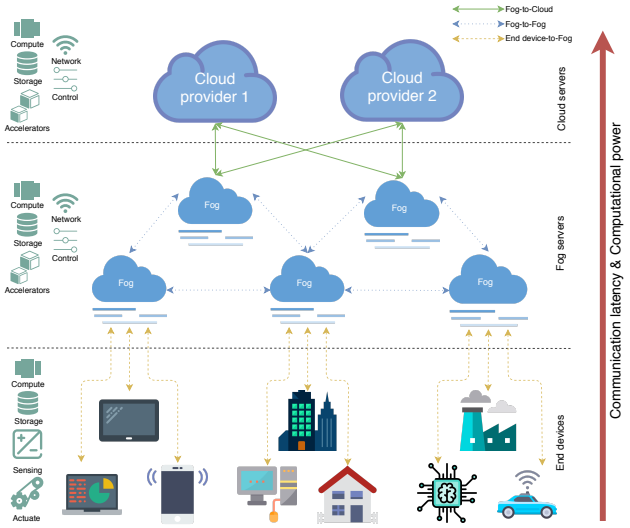

Figure 1. Typical architecture of fog computing.

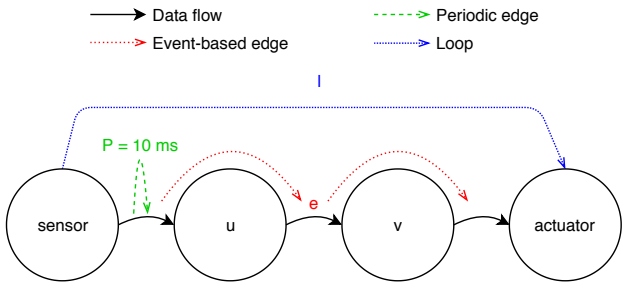

Figure 2. DAG of an application example.

are not directly connected to every other node, however there is always at least one network path between any pair of nodes.

2.2 Services

In this architecture, clients may deploy an arbitrary number of applications/services, each of which structured as a collection of modules using Distributed DataFlow (DDF) [6] programming model modeled as a Directed Acyclic Graph (DAG). In this way, applications can be further enhanced by deploying modules in multiple nodes, rather than running in a single node. As depicted in Figure 2, each DAG is composed of vertices representing modules and edges denoting data dependencies between modules. An application module represents a processing element of an application, which processes each incoming data chunk (i.e., tuple) and, following the model defined in the corresponding application edge, generates output tuples which are sent to next module in the DAG. It is defined with the type of nodes which can host it, as well as the maximum allowed migration time. An application edge denotes the data-dependency between a pair of application modules. It can be defined as periodic in which tuples are issued at regular intervals, or event-based in which tuples are issued per incoming tuple following a given probability or bursty model. Finally, an application loop defines a given path in the DAG which can be defined with the maximum allowed execution time.

Table 1
System variables.

Notation	Unit	Size	Description
N		1×1	Number of nodes
A		1×1	Number of applications
E		1×1	Number of network links
M		1×1	Number of modules
K		1×1	Number of dependencies
Z		1×1	Number of module pairs with dependencies
Q		1×1	Number of loop deadlines
f^{Fog}		$N \times 1$	Indicates whether each node is a fog or client node
f^{Mips}	$MIPS$	$N \times 1$	Processing capacity of each node
f^{Mem}	$Byte$	$N \times 1$	Memory capacity of each node
f^{Strg}	$Byte$	$N \times 1$	Storage capacity of each node
f^{bPw}	W	$N \times 1$	Busy power consumption of each node
f^{iPw}	W	$N \times 1$	Idle power consumption of each node
f^{Tx}	W	$N \times 1$	Cellular network transmitter power of each node
E^S		$E \times 1$	Source node for each network link
E^D		$E \times 1$	Destination node for each network link
E^L	s	$E \times 1$	Link latency for each network link
E^{Bw}	$Byte/s$	$E \times 1$	Link bandwidth for each network link
m^{Mips}	$MIPS$	$M \times 1$	Processing resources needed for each module
m^{Mem}	$Byte$	$M \times 1$	Memory resources needed for each module
m^{Strg}	$Byte$	$M \times 1$	Storage resources needed for each module
m^{MigD}	s	$M \times 1$	Migration deadline for each module
e^{Cpu}	MI	$K \times 1$	Tuple CPU size needed to be processed for each dependency
e^{Nw}	$Byte$	$K \times 1$	Tuple network size needed to be sent for each dependency
e^{Pe}	s	$K \times 1$	Periodicity of sending the tuple for each dependency
e^{Prob}		$K \times 1$	Probability of sending the tuple for each dependency
e^S		$K \times 1$	Source module for each dependency
e^D		$K \times 1$	Destination module for each dependency
l^S		$Z \times 1$	Source module for each pair of modules with dependencies
l^D		$Z \times 1$	Destination module for each pair of modules with dependencies
m^{Bw}	$Byte/s$	$M \times M$	Bandwidth needed between modules
m^{CPU}	MI	$M \times M$	CPU size of dependencies between modules
m^{NW}	$Byte$	$M \times M$	Network size of dependencies between modules
A^L		$Q \times M \times M$	Loop module list
A^D	s	$Q \times 1$	Loop deadline
A^A		$Q \times 1$	Loop application index
D		$N \times M$	Nodes where each module can be deployed
C		$N \times M$	Current module placement
α^p		1×1	Percentage of processing resources used for useful work
α^m		1×1	Percentage of memory resources used for useful work
α^s		1×1	Percentage of storage resources used for useful work
α^b		1×1	Percentage of bandwidth resources used for useful work
t^{Boot}	s	1×1	Constant VM setup time

Table 2
Problem variables.

Notation	Unit	Size	Description
P		$N \times M$	Binary matrix representing the module placement
R		$Z \times E$	Binary matrix representing the tuple routing map
V		$M \times E$	Binary matrix representing the module migration routing map

3 Problem Formulation

The physical topology is composed of N interconnected nodes using E links/edges. Each node, n, is characterized by its processing, memory and storage capacities, f_n^{Mips}, f_n^{Mem}, and f_n^{Strg}, respectively. Also, by its idle, busy and mobile network transmission power consumptions, f_n^{iPw}, f_n^{bPw}, and f_n^{Tx} respectively. Additionally, f_n^{Fog} represents whether n is a client node or not ($f_n^{Fog} = 0$ means node n is a client, otherwise, $f_n^{Fog} = 1$). Still, each edge, e, is characterized by its latency, available bandwidth, source and destination nodes, E_e^L, and E_e^{Bw}, E_e^S, and E_e^D, respectively. Regarding the services, the system has A applications and M modules, which need to be hosted and processed. Each module, m, has its own resource requirements in terms of processing, memory and storage and a migration deadline, m_m^{Mips}, m_m^{Mem}, m_m^{Strg}, m_m^{MigD}, respectively. While the memory, storage and a migration deadline are known variables, the amount of processing capacity is computed through the analysis of the application edges. In total, in the system there

are K application edges. Each application edge, k, is characterized by its resource requirements in terms of processing and bandwidth, e_k^{Cpu} and e_k^{Nw}, respectively. Nonetheless, application edges can be periodic or event-based. In either case, these have a period and a probability of occurring which depends on the source and the path travelled inside the application DAG. Therefore, each application edge, k, is also defined by its periodicity, probability, source and destination modules, e_k^{Pe}, e_k^{Prob}, e_k^S, e_k^D, respectively. With the knowledge of the application edges, it is possible to compute the average required processing capacity of each module (Equation 1), the bandwidth required between modules (Equation 2) and, in the worst case scenario, the quantity of instructions in the queue waiting to be processed from a given module (Equation 3), and the size of data that can be in the queue to be transmitted between modules (Equation 4).

$$m_i^{Mips} = \sum_{k \in K} \frac{e_k^{Prob} e_k^{Cpu}}{e_k^{Pe}},\ e_k^D = i. \tag{1}$$

$$m_{i,j}^{Bw} = \sum_{k \in K} \frac{e_k^{Prob} e_k^{Nw}}{e_k^{Pe}},\ e_k^S = i,\ e_k^D = j. \tag{2}$$

$$m_{i,j}^{CPU} = \sum_{k \in K} e_k^{Cpu},\ e_k^S = i,\ e_k^D = j. \tag{3}$$

$$m_{i,j}^{NW} = \sum_{k \in K} e_k^{Nw},\ e_k^S = i,\ e_k^D = j. \tag{4}$$

Application edges with the same source and destination can be aggregated. This results into Z module pairs with dependencies, where each module pair with dependencies, z, is composed by its source and destination modules, l_z^S and l_z^D, respectively. Applications are also characterized by an arbitrary number of loops. The total number of application loops in the system is Q. Each loop, q, belonging to application A_q^A, is used to specify the process-control loop of interest to the user, which may (or not) have a maximum required execution deadline, A_q^D. These are characterized by their list of modules composing its path within the DAG, A_q^L. From the controller perspective, $D_{n,m}$ defines whether application module m can be placed in node n and $C_{n,m}$ represents whether the application module m, is currently instantiated in node n. Nonetheless, it also defines a percentage of resource capacity which is allocated to perform useful work regarding the processing, memory, storage and bandwidth as α^p, α^m, α^s, and α^b, respectively. Finally, it also defines the average setup time upon migrations as t^{Boot}.

For the sake of completeness, Tables 1 and 2 present the notation used for the problem formulation, the corresponding sizes, units and descriptions.

3.1 Objectives

As the system may not always be able ensure every defined deadline due to the hardware limitations of fog nodes, the main goal of the system provider is to maximize the number of applications which, in the worst case scenario, have their deadlines met, as well as to minimize its total power consumption and resources utilization. The latter comprises computing and networking resources both for application executions and overheads (e.g., migrations). For that purpose, its goal is to find the best module placement, P, routing path for each data dependency between pair of modules, R, and modules migration routing, V.

3.1.1 Quality of Service Cost

In order to maximize the number of applications with QoS assurances, the system provider, based on the state of its nodes and network links, estimates the worst case latency for each application loop and compares it to the defined deadline. As applications may be defined with multiple application loops, the demands of an application are considered to be fulfilled when the QoS demands of all its loops are ensured. This is formulated as the QoS cost, C_Q, as presented in Equation 5.

$$\begin{aligned} C_Q &= \sum_{a \in A} min\left(\sum_{q \in Q} e_q,\ 1\right), A_q^A = a, \\ e_q &= \begin{cases} 1, & if\ L_q^P + L_q^T > A_q^D; \\ 0, & otherwise. \end{cases}, \ \forall q \in [0, Q]. \end{aligned} \tag{5}$$

For each application loop, q, the total delay is characterized by its processing delay, L_q^P, and transmission delay, L_q^T. In order to compute these values, the A_q^L matrix is used, which defines each dependency between modules within the loop. For each generic dependency between module i and j, the processing latency, in the worst case scenario, corresponds to the sum of the processing requirements of all tuples processed by modules hosted in the same node as module j over the node allocated processing capacity, as presented in Equation 6.

$$L_q^P = \sum_{i,j \in M} A_{q,i,j}^L \sum_{n \in N} P_{n,j} \frac{\sum_{l,k \in M} m_{l,k}^{CPU} \times P_{n,k}}{\alpha^p f_n^{Mips}}. \tag{6}$$

Without loss of generality, for each generic dependency between module i and j, the transmission latency, in the worst case scenario, corresponds to the sum of the worst transmission latencies of each link comprised in the dependency path between these two modules. For each link travelled, the worst transmission latency corresponds to the sum of the network requirements of all tuples travelling through the same link over the allocated link transmission capacity, plus the link latency, as expressed in Equation 7.

$$\begin{aligned} L_q^T = \sum_{i,j \in M} A_{q,i,j}^L \sum_{e \in E} R_{z',e} \left(\frac{\sum_{z \in Z} R_{z,e} \times m_{l_z^S, l_z^D}^{NW}}{\alpha^b E_e^{Bw}} + E_e^L \right), \\ l_{z'}^S = i,\ l_{z'}^D = j. \end{aligned} \tag{7}$$

3.1.2 Power Cost

From the system provider perspective, as above mentioned, is also important to minimize the energy consumption once it represents the main source of monetary costs. The linear power consumption cost, C_{Pw}, presented in Equation 8, is related both to the processing cost, C_p, and to the transmission cost of tuples over the cellular network, C_B.

$$C_{Pw} = C_P(f^{bPw} - f^{iPw}) + C_B(f^{Tx}). \tag{8}$$

It represents the multiplication between the percentage of processor utilization and the power consumption when

using the full processor capacity. Similarly, the transmission power cost represents, for each link, the multiplication between the power consumption when using the full transmission capacity and the percentage of link utilization.

3.1.3 Processing Cost

Bearing in mind the QoS required to support its clients, it is also important to keep processors utilization as low as possible. This raises the processing cost, C_P, presented in Equation 9. In this cost, $P_n \times m^{Mips}$ represents the total amount of processing capacity required for all modules placed in node n. Similarly to Equation 7, the processing capacity of node n is considered to be only a part of its full capacity, f_n^{Mips}, by multiplying it by α^p.

$$C_P = \sum_{n \in N} f_n^{Fog} \frac{P_n \times m^{Mips}}{\alpha^p f_n^{Mips}}. \tag{9}$$

Moreover, as clients advertise to the controller the amount of resources that they are willing to use, the system provider is not motivated to minimize its utilization. In fact, it is willing to use the client nodes as much as possible once it represents no power consumption costs for itself. Therefore, the binary system variable f_n^{Fog} is introduced. Note that this affects both processing and power costs.

3.1.4 Bandwidth Cost

The bandwidth cost, C_B, presented in Equation 10, is similar to the processing cost.

$$C_B = \sum_{z \in Z} m^{Bw}_{l_z^S, l_z^D} \sum_{e \in E} f_i^{Fog} \frac{R_{z,e}}{\alpha^b E_e^{Bw}},\ i = E_e^S. \tag{10}$$

In this cost, $\sum_{z \in Z} R_{z,e} \times m^B_{l_z^S, l_z^D}$ gives the amount of bandwidth resources required on link/edge e to support every dependency between pair of modules while using a given matrix R. Therefore, similarly to the processing cost, the objective is to minimize the result of required bandwidth resources over the quantity of available ones. Note that, similarly to Equation 9, the binary system variable f_i^{Fog} is introduced in order to set to zero the cost of transmitting tuples from client nodes. This is motivated by the fact that client nodes can only process their own modules and are only connected to a static fog or cloud node. Therefore, a client node never acts as a bridge of transmission between two nodes. In this way, the system provider does not have to minimize its bandwidth usage nor the energy consumption inherent to its use.

3.1.5 Migration Cost

Migrations are an additional overhead to the system, which are included within the operational maintenance. Although some bandwidth is reserved to perform migrations ($1 - \alpha^b$), these still affect the applications of other users due to the fact that network links only transmit a tuple at a time. Therefore, the goal is to use the links which have higher available bandwidth in order to minimize its impact. In this way, the migration cost, C_M, presented in Equation 11 follows the same principle of processing and bandwidth costs. Note that the migration, as already mentioned, is performed using full VM migration. Therefore the migration size accounts with the total VM size (i.e., $s_m = m_m^{Strg} + m_m^{Mem}$).

$$C_M = \sum_{m \in M} s_m \sum_{e \in E} f_i^{Fog} \frac{V_{m,e}}{(1 - \alpha^b) E_e^{Bw}},\ i = E_e^S. \tag{11}$$

3.2 Constraints

Although the aim is to optimize the system provider objectives, there are some constraints which need to be respected. This section presents the formulation and description of each constraint.

3.2.1 Problem Variables Type

The constraints presented in Equations 12-14 define that P, R, and V are binary matrices, respectively.

$$P_{i,j} \in \{0, 1\},\ \forall i \in [0, N],\ \forall j \in [0, M]. \tag{12}$$

$$R_{z,e} \in \{0, 1\},\ \forall z \in [0, Z],\ \forall e \in [0, E]. \tag{13}$$

$$V_{m,e} \in \{0, 1\},\ \forall m \in [0, M],\ \forall e \in [0, E]. \tag{14}$$

3.2.2 Resource Utilization

Equations 15-18 restrict the resource usage for the processor, memory, storage, an bandwidth, respectively. The constraints defined in Equations 19-20 ensure that no divisions by 0 are performed.

$$P \times m^{Mips} \leq \alpha^p \times f^{Mips}. \tag{15}$$

$$P \times m^{Mem} \leq \alpha^m \times f^{Mem}. \tag{16}$$

$$P \times m^{Strg} \leq \alpha^s \times \leq f^{Strg}. \tag{17}$$

$$\sum_{z \in Z} m^{Bw}_{l_z^S, l_z^D} \times R_{z,e} \leq \alpha^b \times E_e^{Bw},\ \forall e \in [0, E]. \tag{18}$$

$$f_n^{Mips} > 0,\ \forall n \in [0, N]. \tag{19}$$

$$E_e^{Bw} > 0,\ \forall e \in [0, E]. \tag{20}$$

3.2.3 Possible Deployment

Equation 21 ensures that each application module is placed in one and only one node. Equation 22 ensures that each application module is placed in a valid node.

$$\sum_{n \in N} P_{n,m} = 1,\ \forall m \in [0, M]. \tag{21}$$

$$P \leq D. \tag{22}$$

3.2.4 Routing Paths

Equations 23-24 express the data dependencies and migration routing paths, respectively. These routing paths are computed similarly to the how weighed shortest path problems are computed [7]. However, in this case, weights appear due to the influence of the defined objectives.

$$\begin{aligned} &\sum_{i \in E} R_{z,i} - \sum_{j \in E} R_{z,j} = P_{n,l_z^S} - P_{n,l_z^D}\ , \\ &\forall z \in [0, Z],\ \forall n \in [0, N],\ E_i^S = n,\ E_j^D = n. \end{aligned} \tag{23}$$

$$\begin{aligned} &\sum_{i \in E} V_{m,i} - \sum_{j \in E} V_{m,j} = C_{n,m} - P_{n,m}\ , \\ &\forall m \in [0, M],\ \forall n \in [0, N],\ E_i^S = n,\ E_j^D = n. \end{aligned} \tag{24}$$

3.2.5 Migration Deadline

Equation 25 expresses that no migration should exceed the defined deadline.

$$L_m^M \leq m_m^{MigD},\ \forall m \in [0, M]. \tag{25}$$

In order to compute the migration time, it is considered only the bandwidth reserved for this kind of operations, as it is presented in equation 26. Additionally, VMs have a given setup time, t^{boot}, which is also considered in this formulation. In this case, the binary variable b is 1 if and only if the node currently hosting the VM, m, is not the same as the one given by the placement matrix P.

$$\begin{aligned} L_m^M =\ & b \times t^{Boot} + \sum_{e \in E} V_{m,e} \left(\frac{m_m^{Strg} + m_m^{Mem}}{(1-\alpha^b) \times E_e^{Bw}} + E_e^L \right), \\ & b = C_{n,m} - P_{n,m},\ C_{n,m} = 1. \end{aligned} \tag{26}$$

3.3 Final Problem Formulation

As some of the formulated objectives are independent and conflicting objectives (e.g., QoS and power consumption), the problem is defined as a multi-objective problem presented in Equation 27.

$$\min_{P,R,V} \quad F = [\, C_Q, C_{Pw}, C_P, C_B, C_M \,]^{\,T} \tag{27a}$$

$$s.\,t. \quad (12) - (26) \tag{27b}$$

As the defined costs have different units and order of magnitude, the proposed problem formulation as well as the developed algorithms used to solve it, are priority-based. The latter includes the CPLEX optimizer, Brute Force Algorithm (BFA), Random search Algorithm (RA) and Genetic Algorithm (GA), which, due to space constraints, have their implementation details abstracted from this document.

4 Experimental Results

The system performance evaluation was performed in different static and mobile scenarios using the different implemented decision-making algorithms, while adopting the QoS offered to the clients and the system provider objectives as the main metrics. These evaluations are performed onto the developed fog simulation tool which has extended the existing iFogSim simulator in order to implement the proposed architecture. Although its expected behavior had been successfully verified, due to space constraints, its implementation and validation are abstracted from this document. In this way, there are only presented the tests performed regarding the comparison of the different proposed algorithms, the impact of the relative tolerances on the mitigation of the objective priority impositions, and also the impact, limitations and benefits of the migrations in mobile environments.

4.1 Algorithms Comparison

In order to compare the performance of the different proposed algorithms, a static environment with location awareness was implemented. In this environment, there are 6 fog nodes and 4 similar mobile clients, resulting in a total of $N = 10$ nodes. These are interconnected through $E = 18$ links. Each client aims to deploy the Intelligent Surveillance System [5] application, resulting in a total of $M = 24$ modules, and $Z = 20$ dependencies. Therefore, the sizes of the problem variables for the current scenario are given as follows: $P \rightarrow 10 \times 24$ and $R \rightarrow 20 \times 18$. In order to introduce more diversity to the system, one of the fog nodes is characterized by null power consumption and another is mobile. In this test, the BFA is not able to run due to the prohibitively large number of candidate solutions. Hence, it is only compared the performances of the CPLEX optimizer, genetic, and random search algorithms. For that purpose, the simulation area is defined with a square side size of $2.000m$. All nodes are then placed in a random physical position within the simulation area. Then, the controller is responsible to connect each mobile node to its closest fog node. Therefore, in each simulation, the controller needs to optimize the system performance under different characteristics. Note that, although being mobile nodes, these remain in their initial physical positions. This test is performed 10 times for each algorithm, always using the following priority set: *QoS* = 5, *Power* = 4, *Processing* = 3, *Bandwidth* = 2, *Migration* = 1. Note that the higher the value the greater the priority. The results are presented in Table 3. Note that *Config. 2* allows a longer execution time of the corresponding algorithm than the *Config. 1*. As it is possible to verify, the CPLEX optimizer, while finding the optimal solution, was always able to accomplish the defined loop deadlines. Although only two performances are presented (due to space constraints), it can be verified that the solutions found by the GA can be further enhanced by tuning its parameters. As its new solutions are generated based on its previously achieved knowledge through successive generations, increasing the number of individuals in each population allows it to achieve more diversity of solutions, which, in turn, results in achieving better solutions. Moreover, increase its convergence stop condition will also mitigate its premature convergence allowing to achieve better results. In this tests, the random search algorithm has achieved the worst performances. Its parameters for the *Config. 2* were selected in order to achieve similar execution times compared to the *GA-Config. 2*. However, as it can be verified, its improvements compared to the *RA-Config. 1* are negligible. This behaviour was expected due to the fact that, contrary to the GA, it does not use the previously found solutions in the calculation of the new ones. From Table 3, it can be verified that the average simulation results mach the average algorithm results. For instance, regarding the CPLEX values, as $\alpha^p = 0.95$,

Table 3
Performance comparison of the different proposed optimization algorithms.

	CPLEX	*GA-Config.1*	*GA-Config.2*	*RA-Config.1*	*RA-Config.2*
Algorithm Estimates					
QoS value	0	1, 2	0, 4	2, 4	2, 1
Power value	0, 49882	5, 83741	11, 1925	9, 05386	10, 55996
Processing value	0, 23222	0, 79353	0, 80216	0, 87309	0, 927475
Bandwidth value	0, 1515	0, 28923	0, 17707	1, 08821	1, 493319
Simulation Results					
Total energy $[J]$	16662, 067	16712, 73	16763, 59	16742, 69	16757, 32
Total CPU occ. $[\sum \%]$	21, 91679	74, 670418	75, 69641	81, 93536	87, 07744
Total NW occ. $[\sum \%]$	12, 11133	23, 15415	14, 15426	36, 91411	8, 94607

and $\alpha^b = 0.85$, $0,23222 \times 100 \times \alpha^p \simeq 21,91679\%$, and $0,1515 \times 100 \times \alpha^b \simeq 12,11133\%$. Note that this behavior is reflected in every parameter of every algorithm except for the network usage in the RA. This is occurs due to the fact that the random routing path generation does not verify if there are repeated hops. Therefore, some solutions may oversize the required bandwidth. While in the GA this is hidden due to its gathered knowledge when defining new solutions, in the RA, for relatively large scenarios, it will probably find oversized solutions regarding the bandwidth objective.

4.2 Study of the Impact of the Relative Tolerances

It is important to provide flexibility to the system in order to mitigate the impact of the imposed objective priorities. To this end, it is used the scenario defined in Section 4.1. The results are presented in the Table 4. Note that each test is also executed 10 times, using the GA with the *Config. 2* parameters and the priority set defined in Section 4.1. Also, note that *Tol. set 0* implements 0% relative cost tolerance in each objective. The other tolerance sets are described below (when omitted, objectives are defined with 0% relative tolerance). As it can be verified, when using the

Table 4
Impact comparison of different tolerance sets using the GA with the *Config. 2*.

	Tol. set 0	*Tol. set 1*	*Tol. set 2*	*Tol. set 3*	*Tol. set 4*
Algorithm Estimates					
QoS value	$0,4$	$0,3$	$2,1$	$2,2$	$3,1$
Power value	$11,1925$	$4,01428$	$1,13023$	$1,89349$	$2,09963$
Processing value	$0,80216$	$0,43944$	$0,51758$	$0,31808$	$0,14907$
Bandwidth value	$0,17707$	0.06783	0.65662	0.47124	0.43221
Simulation Results					
Total energy [J]	$16763,59$	$16695,44$	$16668,06$	$16675,29$	$16677,19$
Total CPU occ. [$\sum\%$]	$75,69641$	$41,40978$	48.6356	$29,98647$	$14,04985$
Total NW occ. [$\sum\%$]	$14,15426$	$4,50257$	$52,46875$	$37,60341$	$34,51329$

Tol. set 1, its impact on the QoS is negligible. However, its impact on the power objective is significant. Since a QoS objective tolerance of 33.3% is applied, it is able to focus more on the power consumption, and, consequently achieve better results regarding this metric. Regarding the *Tol. set 2*, the impact in the QoS objective is more noticeable. In this case, the GA prefers to give QoS assurance to only 50% of the total number of applications aiming to achieve a better power consumption performance. With respect to the *Tol. set 3*, the 50% QoS tolerance is maintained and a 33.3% tolerance is added to the power objective. While doing so, the impact on the QoS objective is negligible and the power consumption increases for the sake of reducing the processing objective value. Finally, regarding the *Tol. set 4*, where the QoS tolerance is maintained at 50% and the tolerance for the power objective is increased to 50%, the increase both in the QoS and the power values in order to reduce the processing value is noticeable. The impact on the QoS is verified due to the fact that the average rate of new better solutions is reduced, resulting in less optimal solutions regarding this cost. Nonetheless, as expected, it was able to further reduce the processing cost. From Table 3, it can be verified that the average simulation results mach the average algorithm results. Note that this test could be implemented using any of the proposed algorithms, any possible combination of priorities, and any possible combination of relative tolerances. However, due to space constrains, this test is restricted to the above mentioned sets. Nonetheless, it shows that it is possible to reduce the impact of the defined objective priorities, allowing more flexibility to the system.

4.3 Study of the Impact of the Migrations

In this section it is studied the impact, limitations and benefits of the use of migrations while considering mobile environments. The considered deterministic mobile environment is composed of three static and interconnected fog nodes, a mobile fog node and two mobile clients with periodic and fixed trajectories. These mobile nodes are defined with the same velocities, and periodically change their connection, through handovers, between different static nodes. In this scenario, both users aim to deploy the Intelligent Surveillance System in which the application loop *L1* is defined with a deadline of $15ms$, and *L2* with a deadline of $10ms$. In this case, the system is tested using the CPLEX optimizer, in order to optimize the defined scenario in a deterministic manner. The simulation time is set to $300s$. Its results are presented in Table 5. *Set 1* implements the never migrate method, and *Set 2*, *Set 3*, and *Set 4* obey to the priority set defined in Section 4.1, where the VM/container sizes are $5MB$, $50MB$, and $500MB$, respectively. Regarding the values referring to the velocity of $5km/h$, it is possible to verify that, when implementing the *Set 1*, the *Client 1* has almost all loops corresponding to its second loop violated. This only occurs in *Client 1* due to the fact that is the only client node to suffer an handover (due to the initial physical positions). Meanwhile, when implementing the *Set 2* due to the small VM size, its application in terms of the number of accomplished loops is negligibly affected. On the contrary, its number of violated loops is reduced (in this case is zero). Nonetheless, it is possible to verify that the *Client 2*, due to its module placements, suffers some loop violations during the migration of VMs of the *Client 1*. This behaviour increases when increasing the size of the VMs. In the same way, the number of tuples lost also increases. Note that in *Set 4*, there are 4 migrations instead of 3 because, in each machine, $\alpha^s \times f^{Mem}$ only allows to support 3 modules of $500Mb$. Looking through the values referring to the velocity of $50km/h$, this behaviour is more evident. For instance, the number of loops not accomplished while using the *Set 1*, increases substantially for both clients due to the occurrence of handovers (which connect the clients to more network distant nodes compared to the ones hosting its modules). Moreover, when using the *Set 1* through *Set 4*, the number of violated loops is mitigated. However, the number of completed loops is increasingly reduced due to the number of tuples lost during migrations. In this case, as expected, the number of migrations is reduced as the VM sizes increase. This is explained due to the fact that, as migrations take longer, when handovers occur, it is more probable that the migration was already being executed. Therefore, even though the controller decides that the module currently being migrated should be migrated to a different destination, the number of migrations is only incremented once. Finally, looking through the values referring to the velocity of

Table 5
Migration impact under different velocities and VM sizes.

	Velocity = $5km/h$				Velocity = $50km/h$				Velocity = $120km/h$			
	Set 1	*Set 2*	*Set 3*	*Set 4*	*Set 1*	*Set 2*	*Set 3*	*Set 4*	*Set 1*	*Set 2*	*Set 3*	*Set 4*
# Completed *C1 L1*	59979	57893	57159	49975	59935	45141	40584	3072	59911	25454	16775	227
# Violated *C1 L1*	0	0	0	0	28956	57	601	0	25512	0	0	99
# Completed *C1 L2*	2998	2893	2857	2697	2996	2256	2065	573	2991	1342	812	400
# Violated *C1 L2*	2619	0	0	0	1567	0	0	0	1642	1271	29	0
# Completed *C2 L1*	59999	59999	59999	55999	59931	45064	39855	7774	59881	27109	16381	0
# Violated *C2 L1*	0	0	0	0	28934	0	0	0	25510	0	0	0
# Completed *C2 L2*	2999	2999	2999	2999	2996	2246	1986	586	2992	1342	943	0
# Violated *C2 L2*	0	2	29	99	1597	2	29	99	1542	2	0	0
Total CPU occ. $[\sum \%]$	10.11	14.3574	14.279	12.379	10.102	9.0347	8.3027	1.7994	10.095	5.7343	3.81364	0.055
Total NW occ. $[\sum \%]$	8.4628	15.7501	18.1545	37.0627	34.3802	16.7107	57.8769	347.29	34.6151	23.29	109.685	426.02
Total Energy $[J]$	424757	424655	424655	424669	424756	424698	424693	424661	424756	424662	424657	424650
# Packet success	366074	359465	357384	327168	365808	280091	254447	53806	365429	165199	109666	3778
# Packet drop	0	2103	2837	14021	60	29795	39561	109154	212	67436	86843	116441
# Handover	1	1	1	1	14	14	14	14	35	35	35	35
# Migration	0	3	3	4	0	44	40	33	0	107	92	20

$120km/h$, it is possible to verify that this behaviour is even more noticeable. When implementing *Set 4*, the number of completed loops regarding the *Client 1* is extremely reduced, and *Client 2* not even completes a single loop (due to high nodes mobility). Even so, it is clear to see that migrations can be used in order to benefit both the clients and the system provider objectives. In this case, the defined deadline for migrations was set to a sufficient high value so that the controller would always migrate the VMs in order to reduce the QoS objective value. However, as the deadlines are defined by the user itself, the number of tuples lost is not relevant. Moreover, as two fog nodes were defined with a null power consumption (e.g., when deployed inside a bus or connected to a green power source), the use of migrations has allowed to reduce the total system power consumption, provided that the QoS cost is already minimized.

5 Related Work

A wide range of computing paradigms have been proposed in order to bring cloud closer to the end devices, such as mobile computing [8], mobile cloud computing [9], mobile ad hoc cloud computing [10], edge computing [11], cloudlet computing [12], mist computing [13], to name a few. These computing paradigms present different pros and cons, having been proposed to cover different use cases. Even so, fog computing is suited for many use cases, including data-driven computing and low-latency applications, being the most versatile and comprehensive one. Furthermore, fog is flexible enough to interact and take advantage of other paradigms such as edge, cloud, cloudlet and mist computing. Nonetheless, it may not be suitable for a few extreme use cases, such as disaster recovery or sparse network topologies where ad hoc computing may be a better fit.

In the context of fog and related computing paradigms, whenever it is justified the system needs to be readjusted. This is performed through the exchange of VMs between fog nodes. For this reason, it is required to answer the following questions: *When is this exchange justified? And what is the best placement for those applications and/or modules?*

The presented literature addresses different objectives regarding optimization in fog environments. For instance, [14], and [15] perform a single objective optimization in order to minimize the QoS offered to the end users. The works [16] and [17] also perform a single objective optimization regarding the bandwidth usage. R. Deng et al. [18], and Y. Xiao et al. [19] focused on lowering the power consumption. [20], [21], [22], [23] focused on monetary cost minimization. The studies performed in [24], [25], [26], [27], and [28] combine, in a multiple objective optimization, some of the above mentioned objectives. For the sake of analysis between the works described above, Table 6 presents a comparison of the features supported, and Table 7 compares the problem formulation, from whose perspective the problem is being optimized, as well as the algorithm(s) implemented in order to solve the problem formulation.

Table 6
Features comparison of the above described works.

Ref.	VM support	Multiple VM server support	DDF	Multiple fog nodes	Fog coopera-tion	Cloud server	Users mobility	Fog servers mobility	Location aware	Migration
[14]	✓	✓		✓	✓				✓	✓
[15]	✓	✓		✓	✓		✓		✓	✓
[16]	✓	✓	✓	✓	✓	✓	✓		✓	✓
[17]	✓	✓		✓	✓		✓		✓	✓
[18]				✓		✓				
[19]				✓	✓	✓				
[20]	✓	✓		✓	✓				✓	
[21]	✓	✓	✓	✓	✓	✓				
[22]	✓		✓	✓	✓	✓	✓		✓	✓
[23]	✓	✓		✓	✓			✓	✓	
[24]				✓		✓				
[25]	✓	✓		✓	✓	✓	✓		✓	✓
[26]	✓	✓		✓	✓		✓		✓	✓
[27]						✓				
[28]	✓	✓		✓	✓		✓		✓	✓

Regarding Table 6 it is noticeable that there is lack of support for using DDF programming model. Also, some works do not support the use of VM, considering, in this case, only the workload that needs to be processed. However, in order to support running multiple IoT applications at the same time it is mandatory to use some kind of virtualization technique. It is also clear that even though some take into account mobility either from the mobile devices or fog nodes, none of them acknowledges both simultaneously. Migration is also a challenge little explored. However, due to the fact that fog is used in dynamic environments, it is crucial to support migration in order to rearrange the placement of applications or modules whenever needed. Finally, in order to improve the system flexibility, the presence of multiple fog nodes and the cooperation between them is also essential, however, this is not considered in some cases.

Table 7
Problem comparison of the above described works.

Ref.	Optimization perspective	Objectives				Variables			Constraints					Optimization manner	Algorithm
		QoS cost	Bandwidth cost	Power cost	Operational cost	Placement	Data routing	Migration routing	Resources	Bandwidth	Power	QoS deadline	Migration deadline		
[14]	Fog provider	✓				✓								Centralized	PSO
[15]	Fog provider	✓				✓			✓					Centralized	MIQP
[16]	System provider		✓			✓	✓	✓	✓			✓		Distributed	Heuristics
[17]	Fog provider	✓				✓								Centralized	MDP
[18]	System provider			✓		✓			✓	✓		✓		Centralized	GBD, Hungarian, IPM
[19]	Fog provider	✓				✓			✓		✓			Distributed	ADMM-based
[20]	Service provider				✓	✓			✓			✓		Centralized	MILP, LP-based
[21]	Fog provider				✓	✓			✓			✓		Distributed	LP, GA
[22]	Service provider				✓	✓								Centralized	Heuristic, MILP
[23]	Fixed fog provider				✓	✓			✓			✓		Centralized	GA
[24]	System provider	✓			✓	✓			✓	✓		✓		Centralized	Lyapunov-based
[25]	Service provider	✓			✓	✓			✓					Centralized	Heuristics
[26]	Service provider	✓			✓	✓								Centralized	Edmonds–Karp
[27]	Mobile devices	✓		✓	✓	✓			✓	✓				Centralized	IPM-based
[28]	Service provider	✓			✓	✓			✓					Centralized	Regularization-based

With respect to the optimization problem, Table 7, there are several approaches. These works aim to optimize their formulated problem from different perspectives. The fog provider perspective owns a fog infrastructure which may request resources from a cloud provider. The system provider assumes the ownership of both fog and cloud infrastructures. The fixed fog provider (special case of fog provider) perspective assumes only the possession of the fixed fog infrastructure. The service provider or broker, as the name suggests, aims to provide a service to its users, however it does not owns any physical infrastructure. Finally, the mobile devices perspective is similar to the service provider in the sense that it needs to request resources from fog and/or cloud providers, however, in this case, the objectives refer to the mobile devices (e.g., minimize the energy consumption of mobile devices).

Independently from the optimization perspective, there are some flaws in the reviewed literature. First, in order to support real-time IoT applications, its demands in terms of response deadline should be a constraint. For instance, considering a hard real-time application (e.g., autonomous car controller), it is absolutely imperative that responses occur within the required deadline. In this regard, it is mandatory to take into account the servers and network states. However, most of the works which define QoS constraints do not take these parameters into consideration. From a similar perspective, users may also demand a maximum time to migrate its service due to service degradation during this period. As shown, in Table 7, none of the reviewed works has addressed this issue. Finally, some works do not consider the amount of resources each node can provide (i.e., CPU, memory and storage) and/or do not consider the amount of bandwidth available in the links. Nodes can be anything with computational and storage resources and the communications can also be of any type. This way, these constraints should be accomplished in order to ensure that the solution found does not exceed the available resources in each link and node.

6 Conclusion and Future Work

In this work, a novel fog computing system architecture and optimization problem formulation have been proposed and developed in a computer simulation tool. The proposed architecture allows the mobility of IoT and fog nodes while seeking to preserve the user QoS through migration mechanisms. The proposed problem formulation allows to optimize the system performance from a system provider perspective. The proposed framework was tested through computer simulation under different decision-making algorithms. The developed simulation tool significantly extended the existing iFogSim simulator, making it more flexible and able to support more complex modelling, with emphasis on client and resource mobility.

Fog computing has still more than one definition is the literature. It is consensual that is structured as a network of interconnected nodes with processing capabilities, using a wide range of communication technologies which stand between client endpoints and the cloud. These nodes intend to offer their computing power at the service of delay sensitive client applications, in order to provide responses faster than the cloud. Fog computing has a generic nature associated with the fact that it does not rely on any specific designed and purposely built technology. Therefore, the proposed architecture, in contrast to most previously proposed ones, is generic regarding all nodes, communications and applications. Additionally, bearing in mind dynamic environments, the proposed architecture also assumes the presence of mobile fog and client nodes. Migration of services between nodes, is supported and can be explored in order to further enhance the QoS perceived by the users in mobile environments. Provided that constraints (e.g., delay bounds) are fulfilled, its up to the orchestrator, based on its predefined objectives, to decide which node should host and execute each user service. This decision-making and its impact onto the system performance were tested using several optimization algorithms including the CPLEX optimizer, BFA, RA and GA. The performance evaluation of the proposed system is achieved through simulation of different scenarios in the developed fog computing simulator, which further extends the iFogSim simulator capabilities. The performance impact of each component of the system was assessed through some relevant QoS metrics regarding both the client and the system provider objectives. The performance of scenarios using location awareness has been assessed without the influence of mobility. On the one hand, it is shown that, in contrast to the RA, the GA is able to be further enhanced by tuning its configuration parameters due to its gathered knowledge through successive generations. On the other hand, it was demonstrated that even though the problem is being optimized in a priority-based fashion, the usage of relative objective tolerances can be implemented in the interest of providing more flexibility to the system. Finally, the system performance is assessed in a simple and deterministic mobile environment. The latter has allowed to verify the achievable benefits of implementing mobility-triggered migrations of the VMs. These benefits concern both the perceived QoS by the users and the system provider objectives. Nonetheless, it was also verified that this mobility management mechanism has its limitations, which are most notorious when the VM sizes increase and in the presence of highly dynamic environments. In such environments, partition into independently managed fog colonies is a promising technique that should be explored

in extensions of this work.

There are some additional points that can be explored to continue and possibly upgrade what was accomplished in the present work. First, this work only considers the presence of 4G/5G cellular communications with full coverage. This can be further extended by implementing other mobile communication technologies (e.g., Wi-Fi). The implementation of *dead zones* can also be explored. Also, when migrations or handovers occur, some tuples may be lost. A more sophisticated system could implement efficient rerouting protocols in order to mitigate the impact onto the users when these occur. Moreover, the migrations impact can be even further mitigated. First, by implementing live migration mechanisms, which reduces the total down time service. Then, by effectively allocating bandwidth resources for migration purposes in the simulator, the impact of migrations in the QoS of other users could be mitigated. A mobility-aware proactive-based approach optimization, should further reduce the impact of migrations. Furthermore, the implemented architecture considers that the problem size remains constant. In more realistic environments, clients could dynamically enter or leave the system. Moreover, when the desired QoS is not ensured by the optimization algorithm, non-satisfiable clients could be removed from the system in order to improve the service of satisfiable clients. In this way, the QoS cost could be implemented as a constraint instead of an optimization objective. Finally, the implemented architecture still has some limitations due to the high complexity problem formulation. Instead of following a centralized approach, it could be explored the concept of fog colonies. Each colony could be delimited by a graph partitioning technique, number of hops, physical distance, etc. In this way, each colony orchestrator could implement the proposed problem formation. Then, the global optimization could be implemented using a distributed approach or even using a hierarchy of orchestrators in order to simultaneously increase scalability and resource efficiency.

REFERENCES

[1] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," *CISCO white paper*, vol. 1, no. 2011, pp. 1–11, 2011.

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

[3] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *arXiv preprint arXiv:1808.05283*, 2018.

[4] "Cplex optimizer," https://www.ibm.com/analytics/cplex-optimizer, (Accessed on 10/25/2019).

[5] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

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

[7] L. Taccari, "Integer programming formulations for the elementary shortest path problem," *European Journal of Operational Research*, vol. 252, no. 1, pp. 122–130, 2016.

[8] M. Satyanarayanan, "Fundamental challenges in mobile computing," in *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*. Acm, 1996, pp. 1–7.

[9] M. Shiraz, A. Gani, R. H. Khokhar, and R. Buyya, "A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1294–1313, 2013.

[10] J.-P. Hubaux, T. Gross, J.-Y. Le Boudec, and M. Vetterli, "Toward self-organized mobile ad hoc networks: the terminodes project," *IEEE Communications Magazine*, vol. 39, no. 1, pp. 118–124, 2001.

[11] "Open edge computing," http://openedgecomputing.org/about.html, (Accessed on 10/24/2018).

[12] Y. Jararweh, L. Tawalbeh, F. Ababneh, and F. Dosari, "Resource efficient mobile computing using cloudlet infrastructure," in *Mobile Ad-hoc and Sensor Networks (MSN), 2013 IEEE Ninth International Conference on*. IEEE, 2013, pp. 373–377.

[13] "Cisco pushes iot analytics to the extreme edge with mist computing - rethink," https://rethinkresearch.biz/articles/cisco-pushes-iot-analytics-extreme-edge-mist-computing-2/, (Accessed on 10/25/2018).

[14] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "A pso model with vm migration and transmission power control for low service delay in the multiple cloudlets ecc scenario," in *Communications (ICC), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1–6.

[15] X. Sun and N. Ansari, "Primal: Profit maximization avatar placement for mobile edge computing," in *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[16] B. Ottenwälder, B. Koldehofe, K. Rothermel, and U. Ramachandran, "Migcep: operator migration for mobility driven distributed complex event processing," in *Proceedings of the 7th ACM international conference on Distributed event-based systems*. ACM, 2013, pp. 183–194.

[17] W. Zhang, Y. Hu, Y. Zhang, and D. Raychaudhuri, "Segue: Quality of service aware edge cloud service migration," in *Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on*. IEEE, 2016, pp. 344–351.

[18] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1171–1181, 2016.

[19] Y. Xiao and M. Krunz, "Qoe and power efficiency tradeoff for fog computing networks with fog node cooperation," in *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*. IEEE, 2017, pp. 1–9.

[20] 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, 2017.

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

[22] T. Bahreini and D. Grosu, "Efficient placement of multi-component applications in edge computing systems," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 5.

[23] D. Ye, M. Wu, S. Tang, and R. Yu, "Scalable fog computing with service offloading in bus networks," in *Cyber Security and Cloud Computing (CSCloud), 2016 IEEE 3rd International Conference on*. IEEE, 2016, pp. 247–251.

[24] Y. Nan, W. Li, W. Bao, F. C. Delicato, P. F. Pires, Y. Dou, and A. Y. Zomaya, "Adaptive energy-aware computation offloading for cloud of things systems," *IEEE Access*, vol. 5, pp. 23 947–23 957, 2017.

[25] 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, 2016.

[26] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser, "Service entity placement for social virtual reality applications in edge computing," in *Proc. INFOCOM*, 2018.

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

[28] L. Wang, L. Jiao, J. Li, J. Gedeon, and M. Mühlhäuser, "Moera: Mobility-agnostic online resource allocation for edge computing," *IEEE Transactions on Mobile Computing*, 2018.