TÉCNICO LISBOA

Fog and Cloud Computing Optimization in Mobile IoT Environments

José Carlos Ribeiro Vieira

josecarlosvieira@tecnico.ulisboa.pt

Report for introduction to the Research in

Electrical and Computer Engineering

Supervisor(s): Prof. António Manuel Raminhos Cordeiro Grilo
Prof. João Coelho Garcia

January 2019

Contents

1 Introduction

World is growing at a fast pace and so is data. Agility and flexibility of big data applications are gradually taking the form of the Internet of Things (IoT), comprising *things* that have unique identities and are connected to the Internet, being accessible from anywhere in the world. Ubiquitous deployment of interconnected devices is estimated to reach 50 billion units by 2020 [1]. This exponential growth is broadly supported by the increasing number of mobile devices (e.g., smart phones, tablets), smart sensors (e.g., autonomous transportation, industrial controls, wearables), wireless sensors and actuators networks. The number of mobile devices are predicted to reach 11.6 billion by 2021, exceeding the world's projected population at that time (7.8 billion), where the subset of wearable ones are expected to be 929 million [2].

Managing the data generated by IoT sensors and actuators is one of the biggest challenges faced when deploying an IoT system. Although this kind of devices has evolved radically in the last years, battery life, computation and storage capacity remain limited. This means that they are not suitable for running heavy applications, being necessary, in this case, to resort to third parties.

Cloud Computing (CC) is a resource-rich environment that has been imperative in expanding the reach and capabilities of IoT devices. It enables clients to outsource the allocation and management of resources (hardware or software) that they rely upon to the cloud. In addition, to avoid over- or under-provisioning, Cloud Service Providers (CSPs) also afford dynamic resources for a scalable workload, applying a pay-as-you-go cost model. This way, cloud computing is the on-demand delivery of compute power, database storage, applications, and other IT resources through a cloud services platform via the Internet with pay-as-you-go pricing [3]. Besides, it also brings other advantages such as availability, flexibility, scalability, reliability, to mention a few.

Despite the benefits of cloud computing, there are two main problems, linked to IoT applications, which remain unresolved. The first and the most obvious, is the fact that cloud servers reside in remote data centers. Consequently, the end-to-end communication may be subject to long delays (characteristic of multi-hops transmissions over the Internet). Some applications, with ultra-low latency requirements, cannot support such delays. Augmented reality applications which use head-tracked systems, for example, require end-to-end latencies to be less than 16 ms [4]. Cloud-based virtual desktop applications require end-to-end latency below 60 ms if they are to match Quality of Service (QoS) of local execution [5]. Remotely rendered video conference, on the other hand, demand end-to-end latency below 150 ms [6]. On the other hand, the exponential growing number of IoT devices raises the second problem: as the number of connected devices increases, the bandwidth required to support them becomes too large for centralized processing (i.e., CC). To overcome these drawbacks, there are immediately two apparent solutions: (1) to increase the number of centralized cloud data centers, which will be too costly, and (2) to get more efficient with the data sent to the cloud.

The solution which has already been proposed is to bring the cloud closer to the end devices, where entities such as base-stations would host smaller sized clouds. This idea has brought the emergence of several computing paradigms such as cloudlets [7], fog computing [8], edge computing [9], and follow me cloud [10], to name a few. These are different solutions, often confused in the literature, which provide faster approaches and gain better situational awareness in a more timely manner. Regardless of their characteristics, they all share the same goal, implementing solution (2) (i.e., to get more efficient with the data sent to the cloud).

As it will be discussed later, fog computing, also known as fog networking or fogging, is the most comprehensive and natural paradigm to get more efficient with the data sent to the cloud. A simple definition of fog is "cloud closer to the ground", which gives an idea of its functioning. Fog is thus a decentralized computing infrastructure that aims to enable computing, storage, networking, and data management not only in the cloud, but also along the cloud-to-thing path as data traverses the network towards the cloud. Essentially, it extends cloud computing and services along the network itself, bringing them closer to where data is created and acted upon. Fog brought these services closer to the end devices due to its low hardware footprint and low power consumption. This way, both problems raised by the use of cloud computing, may be solved, or at least significantly mitigated. First, the path travelled by the data in the sense-process-actuate model is much shorter (ideally, just one hop to send data and another to receive the results), allowing latency to be much smaller compared to the traditional cloud computing. Second, through geographical distribution, there are significant amounts of data that are no longer travelling up to the cloud. Fog computing, prefers to process data, as much as possible, in the nodes closer to the edge of the network. In a simplistic manner, it only considers transferring the data further if there is not enough computational power to meet the demands.

Nevertheless, cloud is still more suitable than fog for massive data processing, when the latency constraints are not so tight. Therefore, even though fog computing has been proposed to grant support for IoT applications, it does not replace the needs of cloud-based services. In fact, fog and cloud complement each other. Together, they offer services even further optimized to IoT applications. It should be noted that, Internet connectivity is not essential for the fog-based services to work, which means that services can work independently and send necessary updates to the cloud whenever the connection is available [11].

Nonetheless, it is worth mentioning that similarly to cloud computing, fog can use the concept of virtualization to grant heterogeneity. IoT applications may span many Operating Systems (OSs) and application environments (e.g., Android, iOS, Linux, Windows), as well as diverse approaches to partitioning and offloading computation. There is churn in this space from new OS versions, patches to existing OS versions, new libraries, new language run-time systems, and so on. In order for fog to support all these variants, it can be introduced a level of abstraction that cleanly encapsulates this complexity in a Virtual Machine (VM) or a container. Moreover, it enables applications to coexist in a physical server (host) to share resources. Meanwhile, in fog computing, the use of virtualization techniques is also crucial to support migration of applications. The environment of fog computing is, by nature, unquestionably dynamic due to numerous factors such as: the mobility of nodes (i.e., clients and servers), changes in the network, number of applications, requests, nodes, etc. In this fashion, in order to guarantee the desired QoS, changes in the placement of applications may have to be performed through the migration of VMs or containers.

Summing up, in this context, virtualization is a vital technology at different levels namely: (1) isolation between untrusted user-level computations, (2) mechanisms for authentication and access control, (3) dynamic resource allocation for user-level computations, (4) ability to support a wide range of user-level computations, with minimal restrictions on their process structure, programming languages or OSs, (5) mobility, migration and tasks offloading mechanisms, (6) power efficiency, and (7) fault tolerance.

1.1 Motivation

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.

There is lack of support for mobile fog computing. Most of the existing literature assumes that the fog nodes are fixed, or only considers the mobility of IoT devices [11]. Less attention has been paid to mobile fog computing and how it can improve the QoS, cost, and energy consumption. For instance, a bus or a train could have computational power; as a fog node, it could provide offloading support to both end devices (inside and outside it) and other fog servers. The same could be applied to cars that are nowadays getting increasingly better in terms of computational power. Both would be extremely useful to enhance the resources and capabilities of fog computing. Specially in environments such as large urban areas, where traffic congestion is frequent or when those are parked (e.g., while an electric vehicle is charging). On top of that, it would reduce the implementation costs since it would no longer require such computational power in the fixed fog nodes. Finally, it would reduce the costs to the client both in terms of latency and energy consumption, since the fog nodes to which they are connected may be even closer.

Another limitation of fog computing is to take into account few parameters in the decision-making of migration. Most of the existing schemes that are proposed for fog systems, such as offloading, load balancing, or service provisioning, only consider few objectives (e.g., QoS, cost) and assume other objectives do not affect the problem [11]. Fog servers are less powerful than clouds due to the high deployment cost. If many requests are made to the same fog node at the same time, it will not have enough computational and storage capabilities to give a prompt response. So, it raises the question: *should a service currently running in one fog node be migrated to another one, and if yes, where?* While conceptually simple, it is challenging to make these decisions in an optimal manner. Offloading tasks to the closer available server seems to be the solution, however, to migrate either the VM or the container that was initially one-hop away from the IoT device to a multi-hop away server, will increase the network distance. Consequently, it raises the end-to-end latency and the bandwidth usage by the intermediate links. Besides, this decision still has to take into account the cost for both the client (e.g., migration time, computational delay) and the provider (e.g., computing and migration energy). Ignoring some of these variables can lead to wrong decisions that will both violate latency constraints of users' applications and damage or defeat the credibility of fog computing.

1.2 Objectives

This work intends to tackle two of the current limitations that are little or no treated in the literature. One is to provide mobility support in fog computing environments, not exclusively to the end devices but also to the fog nodes, and the other is to achieve multi-objective fog system design. These objectives shall be implemented in a toolkit allowing the simulation of resource management techniques in IoT and mobile fog computing environments. In order to achieve the aforementioned goals, this work involves studying the current mobility approaches that are publicly available, with respect to IoT or fog nodes. Also, analyze several optimization algorithms adopted in the field of data placement, to propose a novel architecture and to select the toolkit in which the proposed solution will be implemented and evaluated.

1.3 Outline

The remainder of the document is structured as follows. Section 2 presents a background section followed by the state-of-the-art and a review of relevant works in the context of the objectives described above. Section 3 describes our proposal to achieve those objectives. Section 4 defines the methodology of how the results obtained will be evaluated. Finally, Section 5 presents the scheduling of the future work and Section 6 concludes the document.

2 Related Work

The solution proposed in this document leverages knowledge obtained from studying several concepts and systems from the current state-of-the-art. In this section, an overview of those concepts and systems will be given, stating for each of them their advantages and disadvantages. This section is structured as follows. Section 2.1 presents different methods to push intelligence and computing power closer to the source of the data and why this work adopted fog computing for this purpose. Section 2.2 describes the generic fog computing architecture, its actors and the different orchestration approaches. Section 2.3 discusses several optimization algorithms regarding migration of virtual resources (e.g., VM). Finally, Section 2.4 shows several open-source simulators of fog-related computing paradigms along with their characteristics.

2.1 Related Computing Paradigms

In what concerns fog computing standardization, there is a lack of unanimity. As aforementioned, fog has been variously termed as cloudlets, edge computing, etc. Different research teams are proposing many independent definitions of fog (and fog-related computing paradigms). As there is a research gap in the definitions and standards for fog computing, this work follows the definitions that Ashkan Yousefpour et al. [11] present. This section presents some paradigms that were proposed in order to bring cloud closer to the end devices, and discusses their pros and cons. As a conclusion, we show why fog computing is the natural platform for IoT.

2.1.1 Mobile Computing

Mobile Computing (MC) is characterized by the processing being performed by mobile devices (e.g., laptops, tablets, or mobile phones). It is necessary to overcome the inherent limitations of environments where connectivity is sparse or intermittent and where there is low computing power. As this model only uses mobile devices to provide services to clients, there is no need for extra hardware. They already have built-in communication modules such as Bluetooth, WiFi, ZigBee, etc. As already mentioned, mobile devices have evolved in recent years. However, their resources are more restricted, compared to fog and cloud. This computing paradigm has the advantage of being characterized by a distributed architecture. The disadvantages of MC are mainly due to its hardware nature (i.e., low resources, balancing between autonomy and interdependence and the need for mobile clients to efficiently adapt to changing environments [12]. This restricts the applications where this paradigm is feasible. For instance, it is unsuitable for applications that require low-latency and which, at the same time, generate large amounts of data that needs to be stored or processed. Nonetheless, MC can use both fog and cloud computing to enhance its capacities and expand its scope of applications.

2.1.2 Mobile Cloud Computing

Cloud computing, as mentioned in Section 2.1.1, is a key element to validate the importance of MC. The interaction between them results in a new paradigm, called Mobile Cloud Computing (MCC). With MCC, applications can be partitioned at runtime and computationally intensive components can be offloaded from mobile devices to the cloud [13]. This way, unlike the resource-constrained MC, MCC has high availability of computing resources, scaling the type of applications where it is possible to use (e.g., augmented reality applications). Also, this characteristic increases the autonomy of mobile devices (i.e., battery lifetime) and enables a much broader range of mobile subscribers, rather than the previous laptops, tablets, or mobile phones. As MCC relies on cloud-based services, where its access is done through the network core by WAN connectivity, applications running on these platforms (i.e., cloud servers) require connection to the Internet all the time. On the one hand, both MCC and MC suffer from the intrinsic characteristics of mobility, such as frequent variations of network conditions (intensified under rapid mobility patterns), and, on the other hand, even if the mobile devices remain fixed, MCC suffers from the inherent disadvantage of using cloud-based services (i.e., communication latency), which makes it unsuitable for some delay-sensitive applications.

2.1.3 Mobile Ad hoc Cloud Computing

In order to overcome the inherent need of MCC for an infrastructure or a centralized cloud, Mobile Ad hoc Cloud Computing (MACC) was proposed [14]. MACC is a Mobile Ad hoc NETworks (MANET), consisting of a set of mobile ad hoc nodes that form a dynamic and temporary network enabled by routing and transport protocols. This computing paradigm allows to form local clouds which can be used for networking, storage and computation. MACC is imperative in use cases such as disaster recovery, car-to-car communication, factory floor automation, unmanned vehicular systems, etc. Although MACC do not rely on external cloud-based services, as MCC does, which mitigates the latency problem, it still has some limitations. Similarly to MC, power consumption constraints are a major concern. Moreover, the formed temporary local cloud may still be computationally weak, and as there is no infrastructure, mobile devices are also responsible for routing traffic among themselves.

2.1.4 Edge Computing

Edge Computing (EC), makes use of connected devices at the edge of the network to enhance its capabilities (i.e., management, storage, and processing power). It is located in the local IoT network, being ideally located one hop away from the IoT device and at most located a few hops away. Open Edge Computing defines EC as a computation paradigm that provides small data centers (edge nodes) in proximity to the users, enabling a dramatic improvement in customer experience through low latency interaction with compute and storage resources just one hop away from the user [15]. As the connected devices do not have to wait for a centralized platform to provide the requested service, nor are so limited in terms of resources as in the traditional MC, their service availability is relatively high. Also, the restrictions over the autonomy are not so tight once there are not only mobile devices. Nonetheless, EC has some drawbacks. Latency, in this context, is composed by three components: data transmission time, processing time and result receiving time. Even though the communication latency is negligible, processing time may be significant. This computing paradigm only uses edge devices, whose computation and storage capabilities may still be poor (e.g., routers, switches), compared to fog or cloud computing, so this processing latency may still be too high for some applications.

OpenFog Consortium states that fog computing is often erroneously called edge computing, but there are key differences between the two concepts [16]. Although they have similar concepts, edge computing tends to be limited to the edge devices (i.e., located in the IoT node network), excluding the cloud from its architecture. Whereas, fog computing is hierarchical and it is not limited to a local network, but instead it provides services anywhere from cloud to *things*. It is worth noting that the term edge used by the telecommunication's industry usually refers to 4G/5G base stations, Radio Access Networks (RANs), and Internet Service Provider (ISP) access/edge networks. Yet, the term edge that is recently used in the IoT landscape refers to the local network where sensors and IoT devices are located [11].

2.1.5 Cloudlet Computing

Cloudlet Computing (cC) is another mobile computing paradigm which aims to bring cloud closer to end devices through the use of cloudlets. It extends MCC by adding the cloudlet tier to its architecture. This way, as Y. Jararweh et al. [17] propose, cC is a 3-tier continuum: mobile-cloudlet-cloud. Cloudlet is, as the name suggests, a smaller sized cloud with lower computational capacity. It can be seen as a "data center in a box" [18], where mobile users can exploit their VM to rapidly instantiate customized-service software in a thin client fashion. This way, it is possible to offload computation from mobile devices to VM-based cloudlets (which are typically one hop away). Through those VMs, cloudlets are capable of providing resources to end devices in real-time over a WLAN network. The relatively low hardware footprint, results in moderate computing resources, but lower latency and energy consumption and higher bandwidth compared to cloud computing. Even though cloudlets are computationally powerful, they still need a connection to the cloud and its services for the following reasons [17]: (1) heavy non real-time jobs might be processed in the enterprise cloud while the real-time ones would be processed by the cloudlet, (2) accessing a file stored in the enterprise cloud, and (3) accessing some services that are not available inside the cloudlet. Despite the clear benefits of cC, fog computing offers a more generic alternative for not being limited solely in this 3-tier. This way, FC natively supports large amounts of traffic, and allows resources to be anywhere along the cloud-to-things continuum. As it will be shown later, cloudlets are great resources and, in this way, they can be combined with the fog computing paradigm.

Table 2.1: Features of fog computing related paradigms (adapted from [11]).

Feature	CC	MC	FC	EC	MCC	MACC	cC	mist
Heterogeneity support	✓		✓	✓	✓			✓
Infrastructure need	✓		✓	✓	✓		✓	✓
Geographically distributed			✓	✓			✓	✓
Location awareness		✓	✓	✓		✓	✓	✓
Ultra-low latency			✓	✓			✓	✓
Mobility support		✓	✓	✓	✓	✓	✓	✓
Real-time application support			✓	✓			✓	✓
Large-scale application support	✓		✓	✓				✓
Multiple IoT Applications	✓		✓				✓	✓
Virtualization support	✓		✓				✓	

2.1.6 Mist Computing

Mist computing emerges to push IoT analytics to the “extreme edge”. This computing paradigm is an even more dispersed version of fog. That means locating analytics tools not just in the core and edge, but also at the “extreme edge” [19]. Mist computing layer is composed by mist nodes that are perceived as lightweight fog nodes. They are more specialized and dedicated nodes with low computational resources (e.g., microcomputers, microcontrollers) that are even closer to the end devices than the fog nodes [20]. Therefore, mist computing can be seen as the first (non-mandatory) layer in the IoT-fog-cloud continuum. It extends compute, storage, and networking across the fog through the *things*. This decreases latency and increases subsystems’ autonomy. It can be implemented in order to enhance the services of predominance of wireless access and mobility support. The challenge with implementing mist computing systems lies in the complexity and interactions of the resulting network. These must be managed by the devices themselves as central management of such systems is not feasible.

2.1.7 Concluding Remarks

As was already mentioned, there are some other similar computing paradigms such as Follow Me Cloud (FMC), Follow Me edge-Cloud (FMeC) and Cloud of Things (CoT), to name a few. However, this state-of-the-art section had as first objective to investigate the most addressed concepts in the literature. The purpose was to understand their characteristics and to identify current limitations that must be tackled by novel solutions, in order to allow the deployment of delay-sensitive IoT systems in mobile environments. Table 2.1 compares the features of the paradigms described above.

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. As aforementioned, 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 (e.g., MACC) may be a better fit.

2.2 Fog Computing Architecture

Fog computing is a great resource to support IoT applications' requirements in mobile environments. Taking into account what has been mentioned in Section 1 and Section 2.1, it has the following fundamental characteristics which validate the statement mentioned above (refer to Table 2.1):

- **Heterogeneity support**. Supports collection and processing of data of different actors acquired through multiple types of network communication, wide diversity applications and services;
- **Geographical distribution**. Uses anything between the cloud and *things* to provide ubiquitous computing, allowing continuity of service in mobile environments;
- **Contextual location awareness, and low latency**. Provides low latency due to the proximity between the IoT devices and the fog nodes. Also, the contextual location allows them to be aware of the cost of communication latency with both other fog nodes and the end devices, allowing the distribution of applications across the network to be organized in a weighted manner;
- **Mobility support**. The exponential growth of mobile devices demands support for mobility techniques;
- **Real-time interactions**. Applications may involve real-time interactions rather than batch processing (e.g., as cloud does);
- **Scalability and agility of federated, fog-node clusters**. Fog is adaptive; may form clusters-of-nodes or cluster-of-clusters to support elastic compute, resource pooling, etc., supporting large-scale applications;
- **Multiple IoT applications**. Fog devices handle multiple IoT applications competing for their limited resources;
- **Virtualization support**. Introduces a software abstraction between the hardware and the OS and application running on the hardware;
- **Interoperability and federation**. Uses cooperation of different providers to support heavy applications such as real-time streaming. Moreover, it supports migration of applications to more suited fog servers depending on the current context;
- **Predominance of wireless access**. Most of the end devices only support wireless communication.

Nonetheless, as stated in Section 1.2, fog still has some limitations. In order to tackle those, its overall architecture must be understood. This includes knowing: what are the actors and how they interact, how IoT nodes connect to the fog servers, how clients outsource the allocation and management of resources that they rely upon to these servers, how migration is performed, etc.

2.2.1 Actors

Figure 2.1 shows the typical fog computing architecture. As stated before the presence of cloud servers is not imperative, however it is very important for numerous applications.

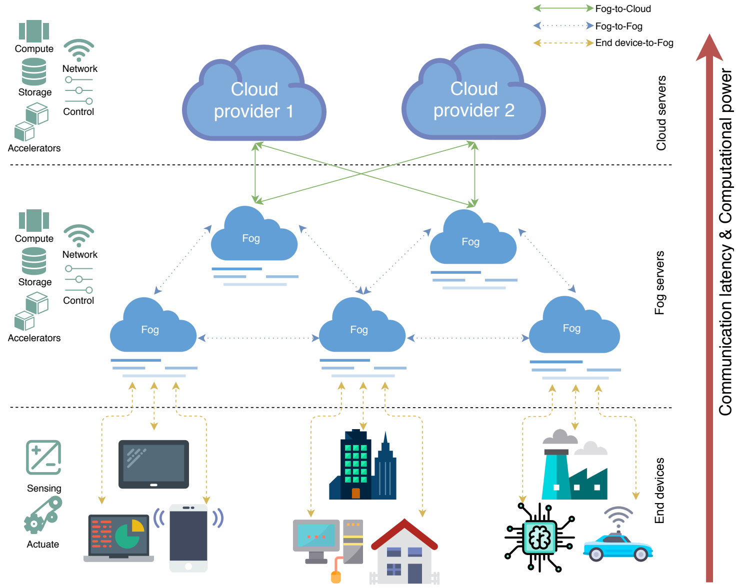

Figure 2.1: Typical architecture of fog computing.

Fog computing layer is composed by fog nodes/servers, which allow the deployment of distributed, latency-aware applications and services. Those nodes can be either physical (e.g., gateways, switches, routers, servers) or virtual (e.g., virtualized switches, virtual machines, cloudlets) components which provide computing resources to the connected end devices. When needed, they also provide network connectivity to centralized services (i.e., cloud). Moreover, fog nodes can operate in a centralized or decentralized manner or even be configured as stand-alone nodes.

Fog nodes are of most value in scenarios where data needs to be collected at the edge and where data from thousands or even millions of devices is analyzed and acted upon in micro and milliseconds [16]. In order to being able to support such large number of requests, especially those engaged in enhanced analytics, fog nodes may be equipped with additional hardware. Accelerators modules (refer to Fig. 2.1) can be implemented to provide supplementary computational throughput. For instance, hardware accelerators can be performed through Graphics Processing Units (GPUs); they are an optimal choice for applications that support parallelism or for stream processing. Also, fog nodes may be equipped with Field Programmable Gate Arrays (FPGAs) or even Digital Signal Processors (DSPs) for this propose.

It is worth noting that, once fog nodes can be anything with computational and storage power in the cloud-to-things continuum, the links formed in these architectures (i.e., End device-to-Fog, Fog-to-Fog and Fog-to-Cloud) can be of any type. For instance, end devices can be connected to fog servers by wireless access technologies (e.g., WLAN, WiFi, 3G, 4G, ZigBee, Bluetooth) or wired connection. Moreover, fog nodes can be interconnected by wired or wireless communication technologies, and they can be linked into the cloud by core network using fiber transmission with low-latency.

In this architecture, the connected sensors located at the edge, generate data that can adopt two models. First, in a sense-process-actuate model, the information collected is transmitted as data streams, which is acted upon by applications running on fog devices and the resultant commands are sent to actuators. In this model, the raw data collected often does not need to be transferred to the cloud; data can be processed, filtered, or aggregated in fog nodes, producing reduced data sets. The result can then be either stored inside fog nodes or actuated upon through the actuators. Second, in a stream-processing model, sensors also send data streams, where the information mined (from the incoming streams) is stored in data centers for large-scale and long-term analytics. In this case, big data needs to be stored and does not have significant latency constraints. Being fog servers less powerful than the cloud ones, cloud is far more suited for this kind of operations. Yet, fog servers can still shrink data, doing some intermediate processing as in the previous model. This meets the aforementioned statement - although cloud is not always essential for the functioning of fog, in some applications it is beneficial or even crucial.

The applications deployed by the connected end users into the fog nodes can be treated either as a whole or as a Distributed Dataflow (DDF) programming model, in which the applications are moduled as a collection of modules. N. Giang et al. [21] propose a DDF for IoT applications which use computing infrastructures across the fog and the cloud, allowing the application flow to be deployed on multiple physical devices rather than one. This can be particular useful to deploy less restricted modules in terms of latency to the upper fog layers (ideally to the cloud), leaving the fog nodes in the lower layers less overloaded, being able to respond faster to modules within tighter latency bounds. As already mentioned, fog utilizes virtualization mechanisms due to the numerous advantages offered. Hence, hosting an application involves creating a set of VMs or execution containers (e.g., Docker) and assign them to a set of physical or virtual components along the cloud-to-things continuum.

2.2.2 Orchestration

When an end device needs to offload some work to a third party, it needs somehow to know where to outsource the allocation and management of resources. To do so, this architecture also needs a discovery service which concerns in finding the best available fog server, given certain capabilities and requirements. In this context, J. Gedeon et al. [22], propose a brokering mechanism in which available surrogates (i.e., fog nodes) advertise themselves to the broker. When it receives client requests, considering a number of attributes such as network information, hardware capabilities, and distance, it finds the best available surrogate for the client.

Finally, fog also needs an orchestration layer to monitor the current context in order to be able to take management decisions with regard to applications and data placement. In this context, C. Guerrero et al. [23] state that most of the literature considers the existence of a central broker or orchestrator gathering all system information (i.e., monitoring fog devices, clients, cloud and services), leading to poor scalability and high orchestration algorithm complexity when the number of elements is high (e.g., in smart cities). To overcome this bottleneck, O. Skarlat et al. [24, 25] consider the concept of fog colonies. Each colony has an orchestrator and an arbitrary number of fog cells (a software component running on fog nodes). These fog cells are responsible for receiving tasks from IoT devices, and depending on the available resources, decide whether to execute it in the current fog node or to transfer it to the orchestrator. This top hierarchic element (inside a colony) will then migrate this task to the cloud or to another fog colony, if the current one is not able to handle it. The authors also state that these orchestrators could be replaced applying a decentralized approach. However, it would lead to extensive coordination and voting between the involved fog cells. On the other hand, F. Bonomi et al. [26] propose a distributed orchestration. This is performed implementing a Foglet software agent in each fog node. It has small footprint yet capable of monitoring machine's health and state. This information is then pushed to the distributed and persistent storage for global processing. The distributed database is also responsible for storing business policies defined by the fog administrators. The distributed policy is then embedded in every Foglet. Specifically, when a Foglet receives a request, it will gather policies from the policy repository and information relative to the currently active service instances from the services directory. With these informations, it tries to find an instance that satisfies the defined constraints. If such was found it, forwards the request, otherwise a new instance needs to be created.

Upon the decision to migrate some application/module, service continuity is an important parameter once downtime may degrade the perceived QoS by the end user. To perform this operation, the exploited technologies are the VMs and containers. For instance, VM synthesis reduces the image size by splitting it into multiple layers, transferring only the application-specific layer which includes both the static binary program and the runtime memory data [27]. However, this can still involve hundreds of megabytes. Also, live migration was introduced to reduce the downtime from the traditional non-live migration. The latter, suspends the VM, transfers all the content from one physical machine to another and resumes it only after the process, continuing from the same state as before suspension. In order to perform live migration there are two different approaches. On the one hand, the pre-copy memory migration firstly transfers VM's memory state. Meanwhile, the VM keeps running. If a page gets modified (dirtied), it will be re-sent. This keeps going until either a small, Writable Working Set (WWS) has been identified, or a given number of iterations is reached. Then, the VM is suspended and sent along with the remaining dirtied pages to the target machine. On the other hand, post-copy creates and sends a snapshot of the VM state from the source physical machine to the destination, being launched at its completion. Meanwhile, during the transfer, the VM is still running in the source machine. Upon copy completion, the memory state that was kept changing is copied on demand (by page-faults) from the source to the destination machine, reducing the downtime services [28]. Too many page-faults may degrade the performance of applications running inside the VM, thus pre-paging can also be used. It ensures that the next pages to be sent to the destination machine are pages in the vicinity of the last fault. Moreover, memory access patterns may also be implemented to further enhance this mechanism.

Despite these advances, this process can be impractical in mobile fog environments due to its large size. Container represents a lighter virtualization technique. It allows developers to package up an application with all the parts needed (e.g., libraries and other dependencies). These applications share the OS of the physical machine and some libraries and/or binaries, allowing container's size to be much smaller. This eases both migration and hosting applications (i.e., more applications in a single machine and does not require restarting the OS upon migration) [29].

Nonetheless, there are two different approaches that can be exploited and implemented into the migration decisions. On the one hand, the reactive approach, as the name suggests, only performs migration when it is needed. When users and fog nodes become out of range, migration is performed to ensure QoS in mobile environments. However, I. Farris et al. [30] argue that downtime is not the only degrading factor of service continuity, but also the overall migration which impacts Quality of Experience (QoE) of users. On the other hand, the proactive approach deploys replicas of the user service in neighboring fog nodes (e.g., using mobility patterns). Also, periodically the state in these nodes is updated. The main goal is to reduce the migration time and improve QoE. However, this approach brings new costs at different levels such as computation, memory, and networking.

Fog servers can provide reduced latencies and help in avoiding/reducing traffic congestion in the network core. However, this comes at a price: more complex and sophisticated resource management mechanisms are needed. This raises new challenges to be overcome such as dynamically deciding when, and where (device/fog/cloud) to carry out processing of requests to meet their QoS requirements. Furthermore, in mobile environments such mechanisms must incorporate mobility (i.e., location) of data sources, sinks and fog servers in the resource management and allocation process policies to promote and take advantage of proximity between fog and end devices.

2.3 Migration Optimization in Mobile Fog Environments

When an IoT device needs to offload some heavy application to a third party, ideally it will be connected to the nearest server, securing a hop away fog server to ensure the shortest network delay. However, as their physical distance increases either by device or server movement, their network distance (i.e., the number of hops) will also increase. Hence, both latency and bandwidth usage by the intermediate links will increase, resulting in poor connectivity. This away, in such dynamic environments the decision-making of where to offload the work is a major concern. Moreover, even if both clients and servers are static, the end-to-end latency may increase due to unexpected crowds of mobile clients seeking to connect or making requests to the same fog server simultaneously, which may lead to QoS violations.

In Section 2.2 it was discussed that applications can be offloaded as a whole, or as a set of modules which may have different latency constraints. Regardless of their type, whenever it is justified the system needs to be readjusted. This is performed through the exchange of VMs or containers (containing the applications or modules) between fog nodes. For this reason, it is necessary to answer the following questions: *When is this exchange justified? And what is the best placement for those applications and/or modules?* As stated in Section 1.2, this work intends to implement multi-objective management decision-making in a novel architecture. Hence, this state-of-the-art section intends to study some proposed mechanisms in the literature.

2.3.1 QoS-Aware

The first objective that fog computing has to guarantee is QoS. When users outsource some delay constrained task or application, they expect fog to be adaptive enough so that they can move while their time boundaries are met. Without this objective fog computing is useless once it appears, in part, to help cloud computing to overcome this limitation.

In this context, the work performed by T. Rodrigues [31] et al. is focused on increasing the QoS offered to its users by lowering both transmission and processing delays. Their goal is achieved by finding the best placement of each mobile user's VM. They assume that each user is connected to the cloudlet which offers the best Received Signal Strength (RSS), that, in turn, is also responsible for hosting its VM. Therefore, their goal is to compute the optimal transmitting power for each cloudlet which will control the RSS and, consequently, change user connections. In order to optimize the formulated problem, is applied a Particle Swarm Optimization (PSO) model. Their architecture assumes the presence of a central unit which is responsible for collecting the physical locations of users and cloudlets and then to execute the model. It is worth noting that when any user changes its connection (i.e., connects to another cloudlet), its VM is also migrated, however this delay is not considered. Also, their work considers that each VM task arrival rate follows a Poisson process with the same rate for all users and does not specify what is the algorithm optimal frequency of execution.

The study performed by X. Sun et al. [32] presents, a case scenario where the end devices are mobile. To perform this work they use a cloudlet network architecture to bring the computing resources from the centralized cloud to the edge. They present the PRofIt Maximization Avatar pLacement (PRIMAL) strategy. PRIMAL maximizes the trade-off between the migration gain (i.e., the end-to-end delay reduction) and the migration cost (i.e., the migration overheads incured in the avatar, which compromise its performance), by migrating the avatars (a software clone located in a cloudlet) to their optimal locations, using pre-copy live migration. To solve the formulated problem, they use the Mixed-Integer Quadratic Programming tool in the CPLEX solver to find the heuristic solution of PRIMAL. It is worth noting that the considered gain only considers the end-to-end delay reduction between the user's base station and user's avatar. Both gain and cost do not consider the servers state nor network state.

2.3.2 Bandwidth-Aware

Minimization of network utilization is one of the main objectives of fog computing. In fact, fog appears to overcome this inherent limitation of cloud computing. Thus, aside from ensuring QoS, it is also important to reduce bandwidth usage. This utilization of network is essentially due to three factors: the transmission of virtualized resources (VMs or containers) which contain the applications/modules, transmission of data between the end device and the deployed application into the fog nodes, and control messages exchanged between fog nodes. If the applications are deployed using DDF programming model a fourth factor rises, the data transmission between modules. In this section, the reviewed literature propose models to mitigate bandwidth usage providing long-term QoS, reducing the number of migrations.

B. Ottenwälder et al. [33] consider an environment with mobile devices and fixed fog nodes, where users offload real-time applications such as Complex Event Processing (CEP). CEP is a paradigm where changes in sensor measurements are modeled as events, while the application is modeled as set of event-driven operators. They state that each migration comes with a cost, consequence of the local state that also needs to be migrated along with the operators. Thus, frequent migration would significantly decrease the system performance. To overcome this limitation, they propose a placement and migration method for fog providers to support operator migrations in Mobile Complex Event Processing (MCEP) systems. Their method plans the migration ahead of time through knowledge of the MCEP system and predicted mobility patterns towards ensuring application-defined end-to-end latency restrictions and reducing the network utilization. These predicted mobility patterns were captured using three different methods: uncertain locations from the *dead reckoning* approach (linear), certain locations that could stem from a *navigation* system (navi), and *learned* transitions between leaf broker (learned). This method allows a minimization of migration costs by selecting migration targets that ensure a low expected network utilization for a sufficiently long time.

Also in this context, W. Zhang et al. [34] state that previous studies have proposed a static distance-based MDP for optimizing migration decisions. However, these models fail to consider dynamic network and server states in migration decisions, assuming that all the important variables are known. Moreover, they also point out another unaddressed problem which lies in the recalculation time interval of the method. Since running MDP is a heavy computing task, a short recalculation interval introduces a considerable overhead to the server. On the other hand, a long recalculation interval may translate into lazy migration, resulting in periods of transgression of QoS guarantees. In order to overcome these issues, the authors propose SEGUE. This model achieves optimal migration decisions by providing a long-term optimal QoS to mobile users in the presence of link quality and server load variation. Additionally, SEGUE adopts a QoS aware scheme to activate the MDP model. In other words, it only activates the MDP model when QoS violation is predicted. Thus, it avoids unnecessary migration costs and bypasses any possible QoS violations while keeping a reasonable low overhead in the servers. The problem formulation is then formulated as a cost-reward between the predicted long term QoS improvement and the service downtime.

2.3.3 Energy-Aware

In order to achieve the QoS objective, the placement of applications and their modules has often to be moved between different entities that compose the things-fog-cloud architecture which evolves energetic costs (both in terms of processing and communication). For instance, it is needed to exchange control messages, communicate between modules placed at different nodes, change the module placement, etc. Thus, energy-aware must be an important factor to be taken into account in the decision making algorithm of *when* and *where* to offload work to another entity in order to minimize fog infrastructure providers' cost.

In this context, R. Deng et al. [35] focused on investigating system power consumption and network delay trade-off in cloud-fog services. They formulate a workload allocation problem, which suggests the optimal workload allocations between fog and cloud toward the minimal power consumption with the constrained service delay. This was performed through the modeled power consumption and delay functions of each part of the fog-cloud computing system. It is worth noting that power consumption only considers energy consumption of work computation, disregarding communication costs. The problem is then tackled using an approximate approach through decomposition, and formulation of three subproblems, being solved through existing optimization techniques. This work also does not considers dynamic environments. All variables are static including the position of fog nodes and end devices. Also there is no cooperation between fog nodes and the communication delay between a fog node and a cloud server is only characterized by its latency, ignoring the bandwidth. Similarly to the majority of the presented works, the decision-making is performed in a centralized manner.

Y. Xiao et al. [36] investigate two performance metrics for fog computing networks: the QoS of mobile users and the power efficiency of fog nodes. In their scheme, fog nodes can process or offload to other fog nodes part of the workload that was initially sent to the cloud. Fog nodes decide whether to offload the workload to neighbors or locally process it, under a given power constraint. A distributed optimization algorithm based on Alternating Direction Method of Multipliers (ADMM) via variable splitting is proposed. This allows to achieve the optimal workload allocation solution that maximizes QoS of users under the given power efficiency. In this work, power efficiency of each fog node is measured by the amount of consumed energy to offload each unit of workload from the cloud. Note that their work do not use the concept of VMs nor DDF programming model, and the considered environment is static (i.e., nodes and clients are static), avoiding the inherent migration problems.

2.3.4 Cost-Aware

As aforementioned, besides guarantying QoS to its users, fog service providers also need to maximize their profit. Hence, it is important to develop an accurate cost model in order to accept and implement fog computing. Besides, similarly to what cloud does, fog has to implement a pay-as-you-go cost model in order to provide services on-demand to its users, without under- or over-provisioning, and charging a fair price. To this end, the cost model needs to apply a communication model, an energy model, and a resource utilization model.

In this context, L. Gu et al. [37] state the importance of fog computing in medical cyber-physical systems as the number of users grows. They state that different infrastructure service providers may apply different charging policies. Therefore, in this paper, the authors aim to minimize the overall resource management cost while satisfying the QoS requirements. They formulate the cost minimization problem in a form of Mixed-Integer NonLinear Programming (MINLP) with joint consideration of communication BS association, subcarrier allocation, computation BS association, VM deployment and task distribution. To tackle the high computational complexity of solving this problem, they linearize it into a Mixed-Integer Linear Programming (MILP) problem. This way they are able to solve the optimal programming model using solvers such as CPLEX and Gurobi. However, it is still time-consuming due to the existence of many integer variables. To this end, they further propose an LP-based two-phase heuristic algorithm. It is worth noting that this work explores placement of VMs in fog computing, whereas it does not tackle this problem in mobile environments, disregarding both users and servers mobility, consequently not addressing the inherent migration problems. Moreover, the proposed method to verify if the QoS requirements are met do not consider the servers state nor network state.

O. Skarlat et al. [25] start by describing a conceptual framework for resource provisioning and service placement in fog. They consider the concept of fog colonies (refer to Section 2.2.2) using a cooperative execution of IoT applications (DDF programming model). Based on this concept, their work, formalizes an optimization problem that aims to adhere to the deadlines on deployment and execution time of applications and to maximize the utilization of existing resources in fog, rather than in cloud, leading to lower execution cost. To solve this placement problem, they apply different approaches, namely the exact optimization method and its approximation through a greedy first fit heuristic and a Genetic Algorithm (GA). They also compare the results, in the fog simulation toolkit iFogSim, to a classical approach that neglects fog resources and runs all services in a centralized cloud. The goal of the evaluation is to identify the best approach to solve the proposed optimization problem in terms of resulting QoS, QoS violations, and cost. The latter is composed only by the execution costs in cloud infrastructures, neglecting execution costs in fog nodes. Their work does not provide mobility mechanisms, not addressing the inherent migration problems. Moreover, all the communications within each fog colony need to be performed through the respective fog orchestration control, introducing new non-negligible communication latencies. Also, the proposed method to verify if the QoS requirements are met do not consider the servers state nor network state.

The work performed by T. Bahreini et al. [38] also addresses multi-tier placement (DDF programming model). The authors formulate the Multi-Component Application Placement Problem (MCAPP). Their objective is to find a mapping between components and servers, such that the total placement cost is minimized. This cost is composed of four types of costs at each time slot: (1) the cost of running one component in a specific server, (2) cost of relocating one component from one server to another, (3) communication cost between one component and the user, and (4) communication between components. With the objective to minimize the overall cost incurred when running the application, they formulate the offine version of the problem as a Mixed Integer Linear Program (MILP) and then developed a heuristic algorithm for solving the online version of the problem. The algorithm is based on an iterative matching process followed by a locals search phase in which the solution quality is improved. This way they use simple algorithmic techniques, avoiding complex approaches such as those based on MDPs. They state that the proposed algorithm has low complexity and adds a negligible overhead to the execution of the applications. Although this work considers the location of servers in the estimation of costs (2) through (4), it does not consider an environment with mobile fog nodes. Also, it only considers the presence of only one user with only one application. Moreover, in each time slot, each server is used by at most one component, not properly taking advantage of fog computing.

A different approach was taken by D. Ye et al. [39]. They leverage the characteristics of buses and propose a scalable fog computing paradigm with servicing offloading in bus networks. Knowing that buses have fixed mobility trajectories and strong periodicity, they consider a fog computing paradigm with service offloading in bus networks which is composed by roadside cloudlets and bus fog servers.

The roadside cloudlet consists of three components: dedicated local servers, location-based service (LBS) providers, access points (APs). The dedicated local servers virtualize physical resources and act as a potential cloud computing site. LBS providers offer the real time location of each bus in bus networks. APs act as gateways for users and bus fog servers within the communication coverage to access the roadside cloudlet. As cloudlets have limited computational and storage resources, they may become overloaded. The bus fog server is a virtualized computing system on bus, which is similar to a light-weight cloudlet server. Hence, those buses not only provide fog computing services for the users on bus, but also are motivated to accomplish the computation tasks offloaded by roadside cloudlets. This allocation strategy is accomplished using GA, where the objective is to minimize the cost that roadside cloudlets spend to offload their computation tasks. Note that there is only considered mobility of fog nodes (i.e., users are static). Also, their problem assumes the applications are deployed as a whole, not addressing the DDF programming model advantages and difficulties in fog computing. Nonetheless, the proposed method to verify if the QoS requirements are met do not consider the servers state nor network state.

2.3.5 Multi-Objective

In some cases rather than only one objective, it might be indispensable to improve the system performance from several perspectives/objectives. However, those can be independent and conflicting objectives. Therefore, unlike the previous sections, the current one aims to present works that were intended to study multi-objective migration optimization algorithms.

Y. Nan et al. [40] aim to provide an energy-efficient data offloading mechanism to ensure minimization of long-term system cost (measured by the money spending on energy consumption) and yet guarantee that users do not perceive a poor QoS. Their work assumes that fog nodes have two sources of energy. The primary source is the solar or green energy which has no monetary cost, however it is finite (in each time slot, the volume of electricity converted from solar energy is a stochastic value depending on the weather conditions). As a backup energy supply, fog nodes have also access to the non-free grid or brown power supply. In addiction, they also assume the presence of cloud data centers which, in this case, have only access to the grid power supply. Their work describes an online adaptive algorithm, Lyapunov Optimization on Time and Energy Cost (LOTEC). LOTEC is a quantified near optimal solution and is able to make control decision on application offloading by adjusting the two-way trade-off between average response time and average cost. This decision-making distributes the incoming applications to the corresponding tiers without a priori knowledge of users and system status. Note that, in this work, there are no VM support nor DDF programming model. Also, there is no fog cooperation and both users and fog nodes are static, not addressing the inherent problems of migration.

The work performed by L. Yang et al. [41] aim to minimize both the average latency of all the users' request loads and the overall costs of service providers. The latter is composed by minimization of both resource usage on cloudlets and service placement transitions. The authors state that this three-way trade-off is a difficult problem. Moreover, the request load could vary significantly and frequently in both spatial and temporal domain due to the mobility of users. Such dynamic request load implies a periodic update of decisions, keeping in mind both the current performance and the expected future workload (using user's mobility pattern and services access pattern to predict the distribution of user's future requests). In order to solve this three way trade-off, the authors first formulate the snapshot problem, named Basic Service Placement Problem (BSPP), which aims to optimize the access latency with the capacity constraints of cloudlets. As it is hard to solve, they design a competitive heuristic to BSPP which outperforms a set of benchmark algorithms. Their work further extends BSPP in order to minimize the above three-way trade-off. To do so, they normalize the costs and apply a weighted sum, allowing the formulation of a single objective problem. It is worth noting that this work does not consider DDF programming model, energy and bandwidth models, routing problems nor mobility of fog nodes.

L. Wang et al. [42] address the social VR applications to study the problem of placing VMs deployed in fog environments such that, the total cost in the overall cost in the fog system is minimized. Although motivated by VR applications, the authors state that this problem is fundamental for any applications that require interactions between either mobile user and the respective VM or user and VMs of other users. The placement problem is to decide where to place the service entity of each user among the cloudlets in order to achieve economic operations of cloudlets as well as QoS. This problem is non-trivial due to the following challenges: (1) cloudlets are heterogeneous in terms of activation and running costs, (2) VMs need to exchange metadata frequently with the associated users and other VMs (of other users), and (3) due to the fact that cloudlets are not intentionally designed to simultaneously accommodate many VMs, especially for VR applications where specific hardware such as GPU may be involved, resource contention needs to be controlled. The authors model the aforementioned challenges with four types of cost: activation cost, the placement cost, the proximity cost, and the collocation cost. These costs are then formulated as a single objective problem by using a weighted sum. They formulate the problem as a combinatorial optimization, which is NP-hard. To solve the problem, they propose ITerative Expansion Moves (ITEM) algorithm, a novel algorithm based on iteratively solving a series of minimum graph cuts. The algorithm is flexible and is applicable in both offline/static (i.e., no movement) and online/dynamic (i.e., with users mobility) scenarios. It is worth noting that the concept of DDF is not applied in this work nor considers fog nodes mobility.

Motivated by the trade-off between local execution power consumption and the offloading delay, the work performed by L. Liu et al. [43] has the objective of minimize energy consumption, delay, and payment cost (E&D&P) for mobile devices in fog computing environments, using queuing theory. Specifically, three types of queues are applied, namely: mobile devices are considered as a M/M/1 queue, fog node as a M/M/c queue with a defined maximum request rate, and cloud as a M/M/∞ queue. Both wireless transmission and computing capabilities are explicitly and jointly considered when modeling this three-way trade-off. They formulate the optimization problem by finding the optimal offloading probability and transmit power. Using the scalarization method, they were able to transform the multi-objective into a single-objective optimization problem. In order to solve that single-objective problem they proposed an Interior Point Method (IPM)-based algorithm which can reduce the accumulated error and improve the calculation accuracy during the iteration process effectively. Note that, in the considered system, there is no movement and there exists only one fog node, which does not fulfill the ubiquity characteristic of fog computing.

L. Wang [44] et al. address two categories of costs, namely: static, which includes the operation cost and the service quality cost, and dynamic, comprising the reconfiguration cost and the migration cost. While the former is independently incurred inside each time slot, the latter is only charged for decision transitions across consecutive time slots. Operation cost refers to the incurred cost in terms of resources utilization (i.e., Central Processing Unit (CPU) and memory) or energy in each cloudlet. Service quality cost, which aims to capture the user perceived QoS, is proportional to the network delay between the user and its workload which may be distributed over several cloudlets. Reconfiguration cost regards to the increase of workload across time slots in each cloudlet. Finally, the migration cost includes both bandwidth cost on the network and the migration delay (both moving out of and into each cloudlet). The single objective problem formulation takes into account all these costs using a weighted sum. They propose Mobility-agnostic Online Edge Resource Allocation (MOERA) based on the "regularization" technique, which decomposes the problem into subproblems and solve them using convex programming. This algorithm receives as input the user's workload and location and decides how resources should be allocated, such that the workload demands from every user is fulfilled while the overall cost system is minimized. Note that their work do not consider the DDF programming model, avoiding its advantages and difficulties in fog computing, and all fog nodes are static.

2.3.6 Concluding Remarks

The presented literature addresses different objectives regarding optimization in fog environments. For instance in Section 2.3.1, the conferred works perform a single objective optimization in order to minimize the QoS offered to the end users. The works in Section 2.3.2, Section 2.3.3 and Section 2.3.4, also perform a single objective optimization, however, besides taking into consideration the QoS, they also consider bandwidth usage, energy and cost, respectively. In Section 2.3.5 were presented works that combine, in a multiple objective optimization, some of the above mentioned objectives.

For the sake of analysis between the works described above, Table 2.2 presents a comparison of the features supported, and Table 2.3 compares the problem formulation, from whose perspective the problem is being optmized, as well as the algorithm(s) implemented in order to solve the problem formulation.

Although their approaches contribute to the improvement of fog computing, they do not account all the aspects that this work aims to cover. Regarding Table 2.2 it is noticeable that there is lack of support for using DDF programming model. As already discussed, in Section 2.2.1, it can bring advantages to fog computing. Also, some works do not support the use of VM, considering, in this case, only the workload that needs to be processed. However, as previously mentioned, in Section 2.2, one of the main goals of fog computing is to support running multiple IoT applications at the same time. To do so, it is mandatory to use some kind of virtualization technique as discussed in Section 1. It also is clear that the analyzed works fail to consider a fully dynamic environment. 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 the environments in which the fog is used are not static, it is crucial 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.

With respect to the optimization problem, Table 2.3, 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 form 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 wants to provide a service to its users, however is owns any physical infrastructure. Finally the mobile devices perspective is similar to the service provider in the sense that needs to request resources from other 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, we consider that 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 always be a constrain. For instance, if we consider a hard real-time application (e.g., autonomous car controller), in the worst case scenario each deadline have to be guaranteed to be fulfilled. In this regard, it is mandatory to take into account the state of servers and links, however, as aforementioned, most of the works which define QoS constrains do not take these parameters into consideration. Note that the works performed in [35] and [40] were able to compute the QoS by taking into account these parameters because there work considers no fog cooperation (i.e., the path is always client-fog-cloud). Similarly the work performed in [33] was also capable to do it, however, as there were fog cooperation, the key element was to know each data and migration routing. Finally, the work [34] also considers these parameters, however, in this case, those were captured with their refined hybrid push/probe technique which introduces some overhead to the system. 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 2, 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. As aforementioned, in Section 2.2.1, nodes can be anything with computational and storage resources and the communications can also be of any type. This way, those constrains should always be accomplished in order to ensure the solution found do no exceed the available resources.

As above mentioned, there were identified some flaws in the reviewed literature. Our aim is to propose a novel architecture which allows to overcome them. To do so, this architecture should be flexible enough to allow supporting different applications, each with an arbitrary number of modules with different demands encapsulated in VMs (i.e., allow the use of VMs and DDF programming model). These applications may be deployed by different users located in with arbitrary locations. Servers in our architecture should also be able to support multiple VMs at the same time (if there exists enough available

Table 2.2: Features comparison of the above described works.

Ref.	VM support	Multiple VM server support	DDF	Multiple fog nodes	Fog co-operation	Cloud server	Users mobility	Fog servers mobility	Location aware	Migration
[31]	✓	✓		✓	✓				✓	✓
[32]	✓	✓		✓	✓		✓		✓	✓
[33]	✓	✓	✓	✓	✓	✓	✓		✓	✓
[34]	✓	✓		✓	✓		✓		✓	✓
[35]				✓		✓				
[36]				✓	✓	✓				
[37]	✓	✓		✓	✓				✓	
[25]	✓	✓	✓	✓	✓	✓				
[38]	✓		✓	✓	✓	✓	✓		✓	✓
[39]	✓	✓		✓	✓			✓	✓	
[40]				✓		✓				
[41]	✓	✓		✓	✓	✓	✓		✓	✓
[42]	✓	✓		✓	✓		✓		✓	✓
[43]						✓				
[44]	✓	✓		✓	✓		✓		✓	✓

Table 2.3: Problem comparison of the above described works.

		Objectives				Variables			Constrains						
Ref.	Optimization perspective	QoS cost	Bandwidth cost	Power cost	Operational cost	Placement	Data routing	Migration routing	Resources	Bandwidth	Power	QoS deadline	Migration deadline	Optimization manner	Algorithm
[31]	Fog provider	✓				✓								Centralized	PSO
[32]	Fog provider	✓				✓			✓					Centralized	MIQP
[33]	System provider		✓			✓	✓	✓	✓			✓		Distributed	Heuristics
[34]	Fog provider	✓				✓								Centralized	MDP
[35]	System provider			✓		✓			✓	✓		✓		Centralized	GBD, Hungarian, IPM
[36]	Fog provider	✓				✓			✓		✓			Distributed	ADMM-based
[37]	Service provider				✓	✓			✓			✓		Centralized	MILP, LP-based
[25]	Fog provider				✓	✓			✓			✓		Distributed	LP, GA
[38]	Service provider				✓	✓								Centralized	Heuristic, MILP
[39]	Fixed fog provider				✓	✓			✓			✓		Centralized	GA
[40]	System provider	✓			✓	✓			✓	✓		✓		Centralized	Lyapunov-based
[41]	Service provider	✓			✓	✓			✓					Centralized	Heuristics
[42]	Service provider	✓			✓	✓								Centralized	Edmonds–Karp
[43]	Mobile devices	✓		✓	✓	✓			✓	✓				Centralized	IPM-based
[44]	Service provider	✓			✓	✓			✓					Centralized	Regularization-based

resources), and be able to communicate between them (if there exist some connection), allowing the fog cooperation feature. Moreover, as discussed in Section 1.1, this work aims to implement fog computing in a completely mobile environment, thus it is also objective to support mobility of both users and fog nodes, as well as location awareness. Meanwhile, as there is movement, connections will be changed as time passes by (e.g., handovers may occur and bandwidth of mobile connections may change), therefore some rearrangements in the placement of VMs, through migration, may be necessary in order to ensure all deadlines are met. Nonetheless, as mentioned in Section 1, the presence of cloud servers are a key element to support fog computing, thus it is also objective to include them. With the above mentioned, this novel architecture covers all the features presented in Table 2.2.

Finally, regarding the placement of VMs, our architecture aims to assume the perspective of a system operator (i.e., owns both fog and cloud servers). In this perspective, the main goals are to minimize the energy consumption of the nodes (by considering processing and communication operations) while keeping the percentage of processor and link resources usage as low as possible. This is important because if a new client enters in the system and asks to deploy a new application and in the surrounding nodes there is no available resources, the system either needs to migrate some VMs in order to support the new client or it denies the access to the user. In either case the operator will have negative effects. This is also applied to the case where a deadline from a current running application is no longer ensured. In this case, the operator needs to migrate some VMs, however if the surrounding nodes have no more available resources, migrations might be longer or the number of migrations might be higher. These objectives must be minimized while ensuring the resources of nodes and links are not exceeded and the QoS, both during the application execution and migration, is met. As discussed above, in order to compute the QoS in the worst case scenario, both server and network state should be considered. This way, the problem variables are the placement, data and migration routing.

2.4 Toolkits

As stated in Section 1.2, the proposed solution, which will be described later in Section 3, will be implemented in a carefully selected toolkit. In order to perform this selection, a survey was made on the currently available simulators. Table 2.4 compares fog and related computing paradigm simulators via comparison of their characteristics.

- **Programming language**. This is important to evaluate the simplicity, level of abstraction offered, maintainability, extensibility, its popularity, etc. As can be observed, almost all are Java-based, being that all opt for object-oriented programming;
- **Documentation**. Unlike the availability, documentation is not always available or sometimes is scarce. In those cases, it is an impediment to the extensibility and maintenance of the corresponding simulators. This parameter includes official documentation, tutorials, community, wiki, etc;
- **Graphical support**. Provide a Graphical User Interface (GUI) may be helpful. Instead of defining

Table 2.4: Comparison of fog and related computing paradigms simulators ('*' - extends CloudSim, '**' - extends iFogSim, '✓✗' - limited).

Simulation Platform	Programming language	Documentation	Graphical support	Energy-aware	Cost-aware	Virtual machine support	Application models	Communication model	Migration support	Mobility/ Location-aware	Fog/Edge support	Last commit	Web page	Paper
CloudSim	Java	✓		✓	✓	✓	✓	✓✗	✓			2016	[45]	[46]
CloudNetSim++	C++		✓	✓	✓	✓	✓	✓	✓			2015	[47]	[48]
GreenCloud	C++/ Otcl	✓		✓		✓	✓	✓	✓			2016	[49]	[50]
iCanCloud	C++	✓	✓	✓	✓	✓	✓	✓				2015	[51]	[52]
CloudSched	Java		✓	✓		✓						2015	[53]	[54]
CloudAnalyst*	Java		✓		✓	✓	✓	✓✗	✓	✓		2009	[45]	[55]
DynamicCloudSim*	Java			✓	✓	✓	✓	✓✗	✓			2017	[56]	[57]
CloudReports*	Java		✓	✓	✓	✓	✓	✓✗	✓			2012	[58]	[59]
RealCloudSim*	Java	✓✗	✓	✓	✓	✓	✓	✓	✓			2013	[60]	
DCSim	Java	✓✗		✓		✓	✓	✓✗	✓			2014	[61]	[62]
CloudSim Plus*	Java	✓		✓	✓	✓	✓	✓	✓			2018	[63]	[64]
CloudSim Plus Automation*	Java	✓		✓	✓	✓	✓	✓	✓			2018	[65]	
DISSECT-CF	Java	✓		✓		✓		✓✗	✓			2018	[66]	[67]
WorkflowSim*	Java	✓		✓	✓	✓	✓	✓✗	✓			2015	[68]	[69]
Cloud2Sim*	Java			✓	✓	✓	✓	✓✗	✓			2016	[70]	[71]
CloudSimDisk*	Java			✓	✓	✓	✓	✓✗	✓			2015	[72]	[73]
iFogSim*	Java	✓	✓	✓	✓	✓	✓	✓✗			✓	2016	[74]	[75]
MyiFogSim**	Java		✓	✓	✓	✓	✓	✓✗	✓	✓	✓	2017	[76]	[77]
iFogSimWithData Placement**	Java		✓	✓	✓	✓	✓	✓✗			✓	2018	[78]	[79]
EdgeCloudSim	Java	✓			✓	✓	✓	✓		✓	✓	2018	[80]	[81]
YAFS	Pyth- on	✓		✓✗			✓✗	✓✗			✓	2018	[82]	
FogTorch	Java					✓	✓	✓✗			✓	2016	[83]	[84]

the entire architecture programmatically, researchers can define it in a user-friendly environment;

- **Energy-aware**. As aforementioned, energy is one of the multi-objectives that this work intends to cover. When implementing the migration optimization algorithm, the more realistic the energy model, the more realistic the algorithm will be. Although CloudSim provides energy-conscious resource management techniques/policies (supports modeling and simulation of different power consumption models and power management techniques), GreenCloud is a more fine-grained simulator to this end. Its energy models are implemented for every data center element. Moreover, due to the advantage in the simulation resolution, energy models can operate at the packet level as well. This allows updating the levels of energy consumption whenever a new packet leaves or arrives from the link, or whenever a new task execution is started or completed at the server [50];
- **Cost-aware**. Similarly, cost-aware is also an important parameter. It is related to the execution of tasks, bandwidth usage and the energy spent during the migrations. Thus, a trade-off between QoS and cost has to be defined. Moreover, migration results in an increase usage of computing resources that are performing non-useful work (overhead). Therefore, a cost model referring to

the quantification in monetary terms of the usage of infrastructure service providers' resources is important, once it allows to apply a pay-as-you go model;

- **Application models**. This is an important feature in terms of QoS because it allows specifying the computational requirements for the application and a specific completion deadline;
- **Communication model**. CloudSim can model network components, such as switches, but lacks fine-grained communication models of links and Network Interface Cards (NIC) causing VM migration and packet simulation to be network-unaware [48]. CloudNetSim++, on the other hand, supports a simulation model of real physical network characteristics such as network congestion, packet drops, bit error, and packet error rates. Moreover, GreenCloud allows communications based on TCP/IP protocol. It allows capturing the dynamics of widely used communication protocols such as IP, TCP, UDP, etc. Whenever a message needs to be transmitted between two simulated elements, it is fragmented into a number of packets bounded in size by network Maximum Transmission Unit (MTU). Then, while routed in the data center network, these packets become a subject to link errors or congestion-related losses in network switches [50];
- **Migration support**. This policy allows applying data placement techiniques (i.e., application and workload migration) to benefit high QoS;
- **Mobility/Location-aware**. As already explained, mobility/location-aware is quite an essential feature in fog computing. It allows maintaining (as much as possible) the end-to-end latency as both users and servers move. There are few simulators that support this feature. For instance, in cloud environments, CloudAnalyst [55] is a tool whose goal is to support the evaluation of social network applications, according to the geographic distribution of users and data centers. However, in fog environments, to the best of our knowledge, there is no support for mobility of fog nodes. The only that provides mobility/location-aware that is currently available, is MyiFogSim. It is an extension of iFogSim to support users' mobility through migration of VMs between cloudlets [77].

3 Description of the Project

Based on the reviewed works, the solution proposed in this section is to develop and implement a novel architecture which will provide mobility support to fog nodes as well as to end devices in a simulation toolkit. Besides, it will also optimize the decision-making of migration by implementing multi-objective decisions into the algorithm, namely: QoS, cost, energy and bandwidth.

3.1 Simulation Toolkit

From Table 2.4, it can be clearly observed that most of the existing simulation toolkits are CloudSim-based. More recent works in fog computing have begun to implement their investigation works in iFogSim, which is also based on CloudSim. Moreover, recently some other simulators have been proposed as extensions to iFogSim. Since such attention has been given to this “family” of simulators, which count with a larger community, information and documentation compared to other isolated simulators, our work will be based on iFogSim.

On the one hand, iFogSim already has really fortunate characteristics. For instance, it has built-in energy (based on CPU utilization), cost (depending on memory, storage, bandwidth and CPU utilization), application (by defining deadlines for modules and applications) and communication models (by defining delays and bandwidth). Also, it already supports virtualization techniques (using VMs). Moreover, this simulator supports DDF programming model, where different modules may be deployed in different machines, creating dependencies between them. In other words, an application module at a given machine is responsible for processing all data generated from modules hosted at machines below in the hierarchy. On the other hand, this simulator has some minor negative points compared with other simulators (refer to Table 2.4), however those are not critical. For example, the built-in communication model is unrealistic by disregarding low-level network issues such as link errors or congestion-related losses. However, these can be treated as high-level attributes such as latency or bandwidth of connections. Moreover, as berofe mentioned, GreenCloud is a more fine-grained simulator in what respects to energy consumption. Still, in iFogSim, energy models are able to vary according to the CPU usage. Furthermore, as network usage is already measured in both directions (downlink and uplink), it is possible to take into account those values in the calculation of energy consumption (as we intend to do).

Despite the above-mentioned issues, iFogSim still has some major drawbacks such as not providing communication between fog nodes at the same level; rather it only provides parent-child communication. This feature is of most importance as discussed before (e.g., to implement an architecture based on fog colonies). Moreover, it does not allow mobility of fog nodes nor end devices and its application placement is static.

3.2 Data Placement Optimization

Based on iFogSim, in order to achieve the desired objectives, firstly is necessary to implement "horizontal" communication between fog nodes.

At the beginning of each simulation, iFogSim executes a placement strategy which is responsible to distribute the miscellany of application modules among the available fog nodes. These nodes are deployed in a hierarchical manner where each fog node has a parent (except for the cloud). Thus, a path is composed by the intermediate fog nodes, the links between them, the cloud and the gateway (fog node which is connected to the sensors and/or actuators). This strategy, favors the placement of modules closer to the end devices. As already mentioned, modules have some dependencies (i.e., the data from one module may be the input of other module). Specifically, this strategy goes for each fog node in a given path search for what are the modules that can be placed (i.e., all their predecessors/dependencies were already placed in south fog nodes). Then, for each one of these modules, it will verify if the current node is able to host it, based on the available CPU. If not, the module is transferred vertically in the hierarchy until some machine is able to host it. This is poorly efficient in terms of load balancing, QoS guarantees and search complexity. It is worth noting that this strategy is performed before the actual simulation begin. Therefore it does not take into account the real migration problems.

Our aim is to search not only vertically but also along all neighborhoods. In a first stage, as iFogSim does not considers geographical positioning of fog servers, we will assume that fog nodes at the same level, are in the close vicinity. Once this is achieved, in order to provide scalability, we aim to divide the search problem into small systems (e.g., as fog colonies does). To do so, we aim so implement graph partitioning functionality that iFogSimWithDataPlacement simulator has already implemented. Specifically this feature is implemented using three components. First, the graph modeling engine creates an undirected graph where vertices represent fog nodes and edges model physical links. Then, the graph weighting engine is responsible for allocate weights to both vertices (based on the number of data items produced in each fog node) and edges (based on the number of data flows passing through these links). Finally, the graph partition engine is responsible to apply a k-way partitioning method to divide it into k sub-graphs using Metis [85], applying the typical criteria: balancing the vertex weights between sub-graphs and minimizing the sum of cut edges weights. We also intend to compare this approach with different criteria such as the end-to-end latency, physical distance or network distance (i.e., number of hops).

At this point, we are now able to implement and test some algorithms proposed in the reviewed literature (refer to Section 2.3) and described later. This would take into account the available parameters in iFogSim to minimize latencies and some other objectives such as energy, cost, bandwidth and, afterwards, jointly consider them all. To perform this implementation, iFogSim already has some built-in valuable attributes, as shown below:

- costPerCPU
- costPerMem
- costPerStorage
- costPerBw
- uplinkLatency
- uplinkBandwidth
- downlinkBandwidth
- idlePower
- busyPower

As such, it will be possible to model the minimization problems and solve them with the algorithms presented in Section 3.5. The objective is to evaluate their performance in terms of both actual optimization and computational complexity. Note that at this stage we are facing a static environment where the optimization is performed before the actual simulation.

3.3 Mobility Support

After the above implementation, the aim is to provide mobility support. For this purpose MyiFogSim, which is also an extension of iFogSim, already provide some important features. It provides mobility support to end devices through migration of virtual machines between fog nodes. On the one hand, their approach introduces the migration policy which is responsible to answer to *when* the VM should be migrated using the user movement (i.e., position, speed, and direction). On the other hand, upon the decision to migrate, they have introduced the migration strategy which defines *where* and *how* the VM should it be migrated. While the former defines when the migration should happen in order to guarantee QoS in the process, the latter regards to the type of migration (i.e., non-live container/VM migration or VM live migration using post-copy) and simple strategies to define the fog node destination such as shortest distance or lowest latency. However, as some review works have shown (e.g., [32, 34]), these greedy strategies are not perfect at all, and other parameters have to be taken into account. Hence, this work intends to implement some work already performed, but extend it to incorporate new features and exploit aspects that were not yet explored. Besides, this work also aims to provide fog servers mobility.

3.4 Architecture

Based on the above mentioned our implementation consists in applying the architecture presented in Figure 3.1. Grey classes are from iFogSim and CloudSim simulators, yellow classes are from MyiFogSim simulator and the green ones are from iFogSimWithDataPlacement. Note that those classes are the ones presented in the corresponding papers, being only the most important ones in their implementations. Our implementation will be focused in the white classes. It is worth mentioning that both MyiFogSim and iFogSimWithDataPlacement, as shown in Table 2.4, do not have documentation and some errors were already found.

As it can be seen, our work will implement four main classes, namely: MobileFogDevice, GA, PSO and MDP. The MobileFogDevice class will extend FogDevice. Therefore, it will inherit all its attributes and methods. Besides, it will have an object of MobileDevice from MyiFogSim, containing attributes such as the direction, speed and geographical position. The remaining classes will focus on implementing some optimization algorithms reviewed in Section 2.3. Note that, at this moment is difficult to define *when* the optimization algorithm recalculation will be performed. This is because, first we need to evaluate their behavior in terms of computational complexity and execution time. However, based on the reviewed literature, there are three options. The first, proposed by MyiFogSim, is to recalculate only when the

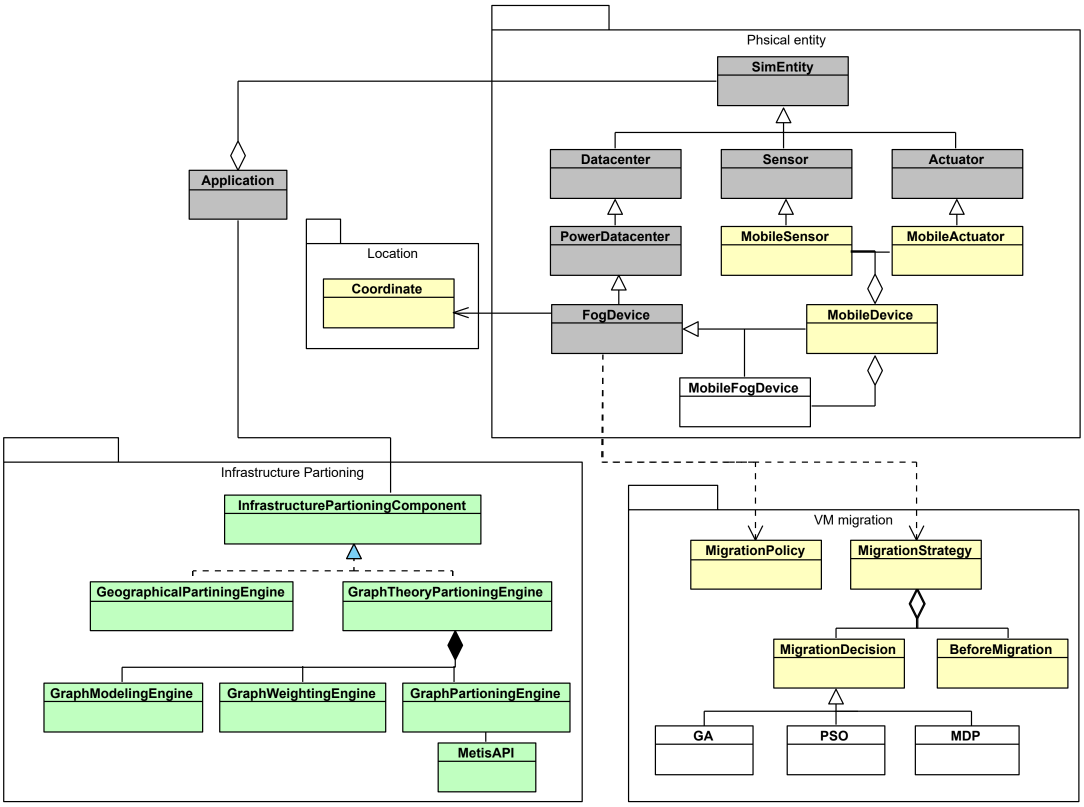

Figure 3.1: UML Class diagram of the main added functionalities.

user is predicted to go out of range from a base station. Based on its position, direction and speed it is possible to predict an handover. When this is verified, the algorithm is executed to compute the best placement to its VM (compute the cloudlet destination). Another approach is to recalculate periodically, but the optimal time interval is never specified (it is only said that it depends on the system dynamics). Finally, the third approach, is to only recalculate when a QoS violation is predicted. This would evolve several parameters such as server and network states as well as mobility and request patterns (e.g., as W. Zhang et al. [34] and B. Ottenwälder et al. [33] did).

3.5 Optimization Algoritms

As already mentioned, we intend to implement and test some proposed algorithms in the reviewed literature. From the panoply of solvers, we intend to study the behavior of the three presented below. Later, depending on the available time, more algorithms may be tested.

3.5.1 Genetic Algorithm

GA is an adaptive heuristic search algorithm. It is based on the idea of natural selection and genetics. Specifically, it is an iterative random search which is able to provide high-quality solutions for optimization problems and search problems.

The algorithm has an arbitrary number of individuals composing the population. During the process, each iteration is characterized by a generation of individuals (of size population). Each individual represents a point in the search space and a possible solution. Yet, for each iteration, it is simulated the process of natural selection or "survival of the fittest". Each individual has a chromosome with genes. By evaluating them, it is possible to define its fitness score. Using this score, there are three operations in the creation of the next generation, namely: the selection operator which selects the individuals that will be part of the next generation, the crossover operator which represents mating between two individuals, resulting in a new individual with a chromosome composed by random genes from the two parents, and, finally, the mutation operator which will randomly introduce genes in the individuals resulting from the crossover operator to maintain the diversity in population to avoid the premature convergence [86].

3.5.2 Particle Swarm Optimization

PSO is inspired by flocking behavior of birds and the schooling behavior of fish. PSO resembles the approach that bees take to find the right flower to collect honey from, or swarms of birds and how they behave collaboratively, as a super-organism. In this case, unlike GA, there is no creation or deletion of individuals. Instead, they move on a landscape where their fitness is measured over time.

In this algorithm, particles are initialized with random solutions. Each particle will store the best fitness for itself and for the swarm as a whole. Each particle will identify the global best by comparing its best finding with all best findings of all particles. Then, they will compute the vector which will conduct them in the direction of the global best. The movement is characterized by the time step and velocity. Note that an initialization with spread of particles too local will possibly lead to a local minimum finding, while a too spread out, will possible not converge [87].

3.5.3 Markov Decision Process

Unlike the previous ones, MDP belongs to the Reinforcement Learning (RL) family of Machine Learning (ML) algorithms. RL refers to goal-oriented algorithms where a software agent is supposed to determine the ideal behavior within a specific context in order to maximize its performance (i.e., some notion of cumulative reward). So that the agent learn its behavior, a reward feedback is required (i.e., reinforcement signal). When this process is repeated, it is known as a Markov Decision Process.

In this model, there are a set of possible states where the agent can be in, a set of models, a set of possible actions, a real valued reward function and a policy. A model or transition model defines the probability to go to state s', while being at state s and taking action a. The real valued reward function or, simply, reward indicates the benefit to take action a while being at state s and ending up in state s'. The policy, which represents the goal, is responsible to give the optimal action to take in every state. Finally, in order to solve MPDs it is needed to resort to dynamic programming (a method which divides the whole problem into smaller and simpler sub-problems), more specifically the Bellman equation [88, 89].

4 Evaluation Methodology

The evaluation of the proposed architecture will be performed both in qualitatively and quantitatively manner. The former will be assessing by evaluating if the objectives defined in Section 1.2 were effectively accomplished. Meanwhile, the latter will be responsible for the evaluation of performance metrics in terms of computational overheads in the orchestrator(s), bandwidth minimization, energy consumption reduction, cost shrinkage to the infrastructure service providers, and the QoS offered to the end users. After the implementation of the extensions that are in sight to be implemented in the simulator (refer to Section 3), it will be possible to evaluate the system by simulating a similar controlled environment as depicted in Figure 4.1. The connections were already discussed in Section 2.2, specifically in Figure 2.1.

The qualitative evaluation will be performed through the following appreciations:

- Verify if the simulator is correctly able to support horizontal communication;
- Check if the implemented optimization algorithms actually converge and give a proper solution;
- Test mobile environments similar to what is depicted in Figure 4.1;
- Verify if the optimization algorithms still converge in mobile environments.

Regarding quantitative evaluation, it will be given special attention to the following metrics:

- Analise both complexity and execution time of each optimization algorithm;
- Measure the QoS offered to applications and if its time boundaries are met with each algorithm;
- Compare those values with the ones offered by the current version of iFogSim;
- Measure values of bandwidth usage, energy consumption, and cost for infrastructure service providers achieved in both static and dynamic environments with each algorithm;
- Compare the obtained results for the static environment with the values achieved by iFogSim.

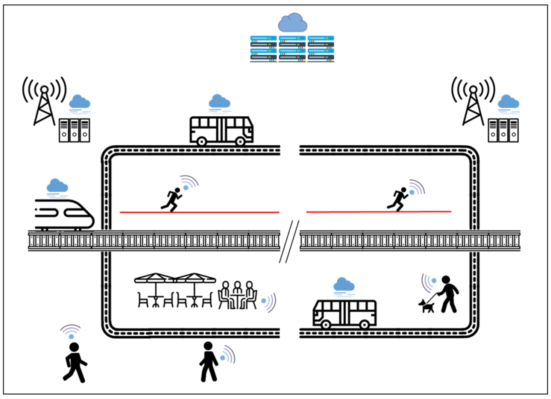

Figure 4.1: Example of setup to evaluate the system and its performance.

5 Schedule of Future Work

Future work is scheduled as follows:

- February, 1st - February, 21th: Choice of applications and specifications of test target scenarios;
- February, 22th - March, 21th: Communication between fog nodes at the same level;
- March, 22th - May, 30th: Static Optimization;
- May, 23th - June, 5th: Preliminary results;
- June, 6th - July, 31th: Mobility implementation;
- August, 1st - September, 25th: Optimization with mobility;
- September, 18th - October, 8th: Final results;
- March, 1st - October, 10th: Write the dissertation about the work performed.

The Gantt diagram corresponding to the thesis schedule is shown in Figure 5.1.

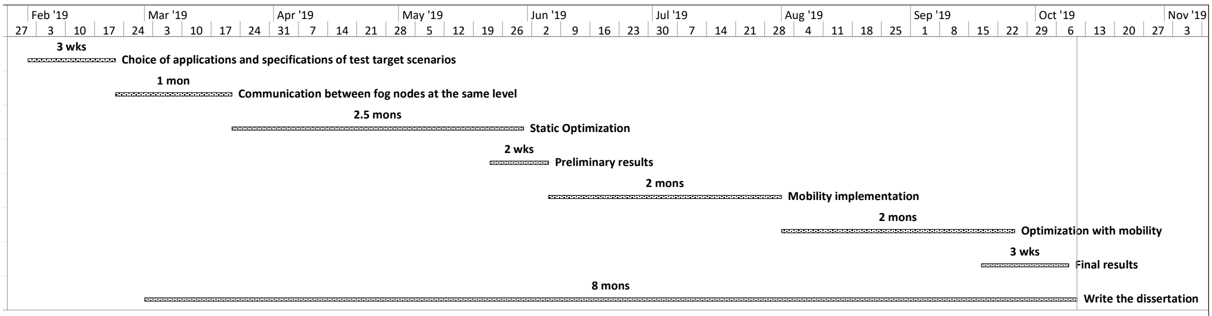

Figure 5.1: Gantt diagram for the thesis schedule.

6 Conclusion

With the evolution of technology, new challenges arise. Regarding IoT devices, the main faced difficulties are lack of processing, memory and battery capabilities. These are even more pronounced when applications are computationally demanding. Unfortunately, cloud computing is not able to solve them. One can deploy applications to a remote cloud. However, the experienced latency will be too costly due to multi-hop communication. Therefore, the processing time reduction will not pay the price of latency for applications with defined time boundary constraints. With fog computing, it becomes possible to fill this gap. It brings cloud closer to the *things*, reducing the end-to-end latency to a negligible value. Still, fog is a new computing paradigm and has some challenges to be resolved. This work intends to tackle, firstly, the lack of mobility support for mobile fog nodes and mobile users (as discussed, there are no studies including both in the literature). Secondly, since server and network state are not static (the experienced end-to-end latency may vary) with regard to migration support, this work aims to provide multi-objective decision-making optimization. Finally, in order to allow future researchers to further improve and validate fog computing, it is also objective to implement the architecture defined in Section 3 in an open-source simulation toolkit.

Bibliography

[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] "Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021 white paper - cisco." https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html. (Accessed on 10/25/2018).

[3] "What is cloud computing? - amazon web services." https://aws.amazon.com/what-is-cloud-computing/. (Accessed on 11/01/2018).

[4] S. R. Ellis, K. Mania, B. D. Adelstein, and M. I. Hill, "Generalizeability of latency detection in a variety of virtual environments," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 48, pp. 2632–2636, SAGE Publications Sage CA: Los Angeles, CA, 2004.

[5] B. Taylor, Y. Abe, A. Dey, M. Satyanarayanan, D. Siewiorek, and A. Smailagic, "Virtual machines for remote computing: Measuring the user experience," *Carnegie Mellon University*, 2015.

[6] T. Szigeti and C. Hattingh, *End-to-end qos network design*. Cisco press, 2005.

[7] M. Satyanarayanan, "Cloudlets: at the leading edge of cloud-mobile convergence," in *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*, pp. 1–2, ACM, 2013.

[8] 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*, pp. 13–16, ACM, 2012.

[9] S. Davy, J. Famaey, J. Serrat-Fernandez, J. L. Gorricho, A. Miron, M. Dramitinos, P. M. Neves, S. Latré, and E. Goshen, "Challenges to support edge-as-a-service," *IEEE Communications Magazine*, vol. 52, no. 1, pp. 132–139, 2014.

[10] T. Taleb and A. Ksentini, "Follow me cloud: interworking federated clouds and distributed mobile networks," *IEEE Network*, vol. 27, no. 5, pp. 12–19, 2013.

[11] 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.

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

[13] 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.

[14] 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.

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

[16] O. C. A. W. Group *et al.*, "Openfog reference architecture for fog computing," *OPFRA001*, vol. 20817, p. 162, 2017.

[17] 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*, pp. 373–377, IEEE, 2013.

[18] M. Satyanarayanan, V. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, 2009.

[19] "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).

[20] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. S. Goren, and C. Mahmoudi, "Fog computing conceptual model," tech. rep., 2018.

[21] 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*, pp. 155–162, IEEE, 2015.

[22] J. Gedeon, C. Meurisch, D. Bhat, M. Stein, L. Wang, and M. Mühlhäuser, "Router-based brokering for surrogate discovery in edge computing," in *Distributed Computing Systems Workshops (ICDCSW), 2017 IEEE 37th International Conference on*, pp. 145–150, IEEE, 2017.

[23] C. Guerrero, I. Lera, and C. Juiz, "On the influence of fog colonies partitioning in fog application makespan," in *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 377–384, IEEE, 2018.

[24] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource provisioning for iot services in the fog," in *2016 IEEE 9th Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 32–39, IEEE, 2016.

[25] 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.

[26] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, “Fog computing: A platform for internet of things and analytics,” in *Big data and internet of things: A roadmap for smart environments*, pp. 169–186, Springer, 2014.

[27] L. Ma, S. Yi, and Q. Li, “Efficient service handoff across edge servers via docker container migration,” in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, p. 11, ACM, 2017.

[28] M. R. Hines, U. Deshpande, and K. Gopalan, “Post-copy live migration of virtual machines,” *ACM SIGOPS operating systems review*, vol. 43, no. 3, pp. 14–26, 2009.

[29] 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*, pp. 258–269, ACM, 2016.

[30] I. Farris, T. Taleb, M. Bagaa, and H. Flick, “Optimizing service replication for mobile delay-sensitive applications in 5g edge network,” in *Communications (ICC), 2017 IEEE International Conference on*, pp. 1–6, IEEE, 2017.

[31] 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*, pp. 1–6, IEEE, 2017.

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

[33] 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*, pp. 183–194, ACM, 2013.

[34] 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*, pp. 344–351, IEEE, 2016.

[35] 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.

[36] 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*, pp. 1–9, IEEE, 2017.

[37] 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.

[38] 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*, p. 5, ACM, 2017.

[39] 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*, pp. 247–251, IEEE, 2016.

[40] 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. 23947–23957, 2017.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] “The clouds lab: Flagship projects - gridbus and cloudbus.” `http://www.cloudbus.org/cloudsim/`. (Accessed on 11/15/2018).

[46] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, “Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[47] “cloudnetsim.seecs.edu.pk.” `http://cloudnetsim.seecs.edu.pk/`. (Accessed on 11/15/2018).

[48] A. W. Malik, K. Bilal, S. Malik, Z. Anwar, K. Aziz, D. Kliazovich, N. Ghani, S. U. Khan, and R. Buyya, “Cloudnetsim++: a gui based framework for modeling and simulation of data centers in omnet++,” *IEEE Transactions on Services Computing*, no. 4, pp. 506–519, 2017.

[49] “Greencloud - the green cloud simulator.” `https://greencloud.gforge.uni.lu/`. (Accessed on 11/15/2018).

[50] D. Kliazovich, P. Bouvry, and S. U. Khan, “Greencloud: a packet-level simulator of energy-aware cloud computing data centers,” *The Journal of Supercomputing*, vol. 62, no. 3, pp. 1263–1283, 2012.

[51] “Web site.” `https://www.arcos.inf.uc3m.es/old/icancloud/Home.html`. (Accessed on 11/15/2018).

[52] A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente, “icancloud: A flexible and scalable cloud infrastructure simulator,” *Journal of Grid Computing*, vol. 10, no. 1, pp. 185–209, 2012.

[53] “Cloudsched download — sourceforge.net.” `https://sourceforge.net/projects/cloudsched/`. (Accessed on 11/15/2018).

[54] W. Tian, Y. Zhao, M. Xu, Y. Zhong, and X. Sun, “A toolkit for modeling and simulation of real-time virtual machine allocation in a cloud data center,” *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 1, pp. 153–161, 2015.

[55] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, “Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications,” in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pp. 446–452, IEEE, 2010.

[56] “marcbux/dynamiccloudsim: Automatically exported from code.google.com/p/dynamiccloudsim.” `https://github.com/marcbux/dynamiccloudsim`. (Accessed on 11/15/2018).

[57] M. Bux and U. Leser, “Dynamiccloudsim: Simulating heterogeneity in computational clouds,” *Future Generation Computer Systems*, vol. 46, pp. 85–99, 2015.

[58] “thiagotts/cloudreports: An extensible simulation tool for energy-aware cloud computing environments.” `https://github.com/thiagotts/CloudReports`. (Accessed on 11/15/2018).

[59] T. T. Sá, R. N. Calheiros, and D. G. Gomes, “Cloudreports: an extensible simulation tool for energy-aware cloud computing environments,” in *cloud computing*, pp. 127–142, Springer, 2014.

[60] “Realcloudsim download — sourceforge.net.” `https://sourceforge.net/projects/realcloudsim/`. (Accessed on 11/15/2018).

[61] “digs-uwo/dcsim: Dcsim: A data centre simulation tool for evaluating dynamic virtualized resource management.” `https://github.com/digs-uwo/dcsim`. (Accessed on 11/15/2018).

[62] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya, “Dcsim: A data centre simulation tool for evaluating dynamic virtualized resource management,” in *Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm)*, pp. 385–392, IEEE, 2012.

[63] “Cloudsim plus — a modern, full-featured, highly extensible and easier-to-use java 8 framework for modeling and simulation of cloud computing infrastructures and services.” `http://cloudsimplus.org/`. (Accessed on 11/15/2018).

[64] M. C. Silva Filho, R. L. Oliveira, C. C. Monteiro, P. R. Inácio, and M. M. Freire, “Cloudsim plus: a cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness,” in *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*, pp. 400–406, IEEE, 2017.

[65] "manoelcampos/cloudsim-plus-automation: Human readable scenario specification for automated creation of simulations on cloudsim plus and cloudsim." `https://github.com/manoelcampos/cloudsim-plus-automation`. (Accessed on 11/15/2018).

[66] "kecskemeti/dissect-cf: Discrete event based energy consumption simulator for clouds and federations." `https://github.com/kecskemeti/dissect-cf`. (Accessed on 11/15/2018).

[67] G. Kecskemeti, "Dissect-cf: a simulator to foster energy-aware scheduling in infrastructure clouds," *Simulation Modelling Practice and Theory*, vol. 58, pp. 188–218, 2015.

[68] "Workflowsim/workflowsim-1.0: Wiki pages." `https://github.com/WorkflowSim/WorkflowSim-1.0`. (Accessed on 11/15/2018).

[69] W. Chen and E. Deelman, "Workflowsim: A toolkit for simulating scientific workflows in distributed environments," in *E-science (e-science), 2012 IEEE 8th International Conference on*, pp. 1–8, IEEE, 2012.

[70] "Cloud2sim download — sourceforge.net." `https://sourceforge.net/projects/cloud2sim/`. (Accessed on 11/15/2018).

[71] P. Kathiravelu and L. Veiga, "An adaptive distributed simulator for cloud and mapreduce algorithms and architectures," in *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, pp. 79–88, IEEE, 2014.

[72] "Udacity2048/cloudsimdisk: A new storage modeling for cloudsim simulator.." `https://github.com/Udacity2048/CloudSimDisk`. (Accessed on 11/15/2018).

[73] B. Louis, K. Mitra, S. Saguna, and C. Åhlund, "Cloudsimdisk: Energy-aware storage simulation in cloudsim," in *Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th International Conference on*, pp. 11–15, IEEE, 2015.

[74] "Cloudslab/ifogsim: The ifogsim toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments." `https://github.com/Cloudslab/iFogSim`. (Accessed on 11/15/2018).

[75] 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.

[76] "marciocomp/myifogsim: Myifogsim is a simulator for fog computing concerns. it extends the ifogsim simulator to support virtual machine migration policies for mobile users." `https://github.com/marciocomp/myifogsim`. (Accessed on 11/15/2018).

[77] M. M. Lopes, W. A. Higashino, M. A. Capretz, and L. F. Bittencourt, "Myifogsim: A simulator for virtual machine migration in fog computing," in *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*, pp. 47–52, ACM, 2017.

[78] "medislam/ifogsimwithdataplacement." https://github.com/medislam/iFogSimWithDataPlacement. (Accessed on 11/15/2018).

[79] M. I. Naas, J. Boukhobza, P. R. Parvedy, and L. Lemarchand, "An extension to ifogsim to enable the design of data placement strategies," in *Fog and Edge Computing (ICFEC), 2018 IEEE 2nd International Conference on*, pp. 1–8, IEEE, 2018.

[80] "Cagataysonmez/edgecloudsim: Edgecloudsim: An environment for performance evaluation of edge computing systems." https://github.com/CagataySonmez/EdgeCloudSim. (Accessed on 11/15/2018).

[81] C. Sonmez, A. Ozgovde, and C. Ersoy, "Edgecloudsim: An environment for performance evaluation of edge computing systems," in *Fog and Mobile Edge Computing (FMEC), 2017 Second International Conference on*, pp. 39–44, IEEE, 2017.

[82] "yafs pypi." https://pypi.org/project/yafs/. (Accessed on 11/15/2018).

[83] "di-unipi-socc/fogtorch." https://github.com/di-unipi-socc/FogTorch. (Accessed on 11/15/2018).

[84] 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, 2017.

[85] "Metis - serial graph partitioning and fill-reducing matrix ordering — karypis lab." http://glaros.dtc.umn.edu/gkhome/metis/metis/overview. (Accessed on 12/20/2018).

[86] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[87] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of machine learning*, pp. 760–766, Springer, 2011.

[88] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, pp. 679–684, 1957.

[89] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas, *Dynamic programming and optimal control*, vol. 1. Athena scientific Belmont, MA, 2005.