PureEdgeSim: A Simulation Framework for Performance Evaluation of Cloud, Edge and Mist Computing Environments

Charafeddine Mechalikh, Hajer Taktak, and Faouzi Moussa

University of Tunis El Manar, Faculty of Sciences of Tunis,
LIPAH-LR11ES14, 2092Tunis, Tunisia
{charafeddine.mechalikh, taktakhajer, faouzimoussa}@gmail.com

Abstract. Edge and Mist Computing are two emerging paradigms that aim to reduce latency and the Cloud workload by bringing its applications close to the Internet of Things (IoT) devices. In such complex environments, simulation makes it possible to evaluate the adopted strategies before their deployment on a real distributed system. However, despite the research advancement in this area, simulation tools are lacking, especially in the case of Mist Computing [11], where heterogeneous and constrained devices cooperate and share their resources. Motivated by this, in this paper, we present *PureEdgeSim*, a simulation toolkit that enables the simulation of Cloud, Edge, and Mist Computing environments and the evaluation of the adopted resources management strategies, in terms of delays, energy consumption, resources utilization, and tasks success rate. To show its capabilities, we introduce a case study, in which we evaluate the different architectures, orchestration algorithms, and the impact of offloading criteria. The simulation results show the effectiveness of *PureEdgeSim* in modeling such complex and dynamic environments.

Keywords: Simulation, modeling, tasks orchestration, load balancing, Mist Computing, Edge Computing.

1. Introduction

With the emergence of IoT, connected devices are gradually invading our daily lives with increasingly broad fields of application: personal health equipment, smart buildings, smart grids, connected vehicles, etc. A recent study estimates that the number of connected devices will exceed 38.6 billion by 2025, with economic benefits in the health, energy, transportation, and construction sectors [1]. However, due to this growth, Cloud Computing has faced many challenges. Not only has it become unable to support the growing number of IoT devices and the data they continually generate, but it is also unable, due to its remote location, to meet their quality of service requirements such as low latency. To face this, a new paradigm is needed. The latter must provide computing, storage, and services like the conventional Cloud and meet the quality of service requirements of IoT applications such as low latency, high scalability, and mobility.

This need for a new computing paradigm gave birth to Edge and Mist Computing. While Edge Computing covers a wide range of applications such as Fog Computing, Mobile Edge Computing, and Cloudlets all of which extend the Cloud by providing

resources in the network layer of the IoT architecture [10, 16], Mist Computing allows resources to be harvested through the computation and communication capabilities offered in the perception layer [11]. As a result, most of the data generated by these devices can be processed locally, which reduces the latency, increases scalability, and minimizes energy consumption by saving the energy that would have been used to transfer data. However, in these complex and distributed environments, many issues need to be solved (e.g., load balancing, application placement, and resource discovery) and experimenting on a real distributed environment or testbeds [24] is not practical due to the cost and limited scalability.

The simulation makes it possible to evaluate the performance of the proposed approaches in a repeatable and controllable manner before their actual deployment in a real distributed system. Nevertheless, due to their heterogeneous, dynamic, and distributed nature, the simulation of Edge and Mist Computing environments is not such a simple task. Each IoT application (smart cities, connected vehicles, etc.) uses a heterogeneous mix of sensing and actuation devices. These devices, connected to telecommunication networks, can interact with one another or with computing infrastructures in order to compute their tasks. Simulating such environments will, therefore, require modeling the network, computation resources, the heterogeneity of devices, their behaviors, and the data they generate. Fig. 1 presents the aspects of modeling of Edge Computing environments. To model the virtualized resources (e.g., CPU, memory, storage), many existing solutions have extended and exploited Cloud Computing simulators such as *CloudSim* [3], which is a rich and highly extensible framework that enables the simulation of Cloud resources (virtual machines, hosts, data centers) and services. However, since transmission delays are directly proportional to the network workload, the use of fixed transmission delays as in these existing simulators is not practical, especially when evaluating the scalability of the system. On the other hand, the use of network simulators, such as *OMNET++* [20] and *NS-3* [7], allows efficient network modeling. However, users have to define all the other aspects of the simulation (Fig. 1) such as load generation, tasks orchestration, mobility model, and resources utilization models in order to assess the performance of their solutions, which takes a lot of time and effort.

Motivated by this, in this paper, we present *PureEdgeSim*, a simulation framework that enables the evaluation of resources management strategies and the performance evaluation of Cloud, Edge, and Mist Computing environments [11]. It covers all the modeling and simulation aspects of Edge Computing that are given in Fig. 1. *PureEdgeSim* offers a modular architecture where each of its modules deals with a specific part of the simulation. The *Network Module*, for example, is responsible for data transfer and bandwidth allocation. The *Location Manager* module deals with the geo-distribution of devices and their mobility. The *Data Centers Manager* module takes care of the generation of devices and their heterogeneity. Finally, the *Orchestrator* module, which is responsible for tasks offloading decisions. These modules also provide a default implementation and a set of adjustable parameters in order to ease experimentation and prototyping. As a result, researchers can quickly implement their solutions without wasting time on the specification of low-level details. To demonstrate its capabilities, a case study is introduced, in which we propose a simulation scenario that mimics a smart university campus. During this case study, we propose a multi-tier architecture that takes advantage of smart edge devices that have sufficient computing capacity. To support their heterogeneity and meet the QoS, we present a tasks orchestration algorithm that is based

on the Fuzzy Decision Tree. The simulation results show the effectiveness of *PureEdgeSim* in modeling such complex, heterogeneous, and dynamic environments. They also highlight the advantages of adopting Mist Computing and the effectiveness of the proposed algorithm that outperformed the competitor algorithms in every aspect of the comparison.

This paper is organized as follows: In section 2, the related work is presented. Section 3 describes *PureEdgeSim* architecture. In section 4, a use case scenario is proposed. The simulation results are assessed in section 5. Finally, section 6 concludes the paper and highlights future directions.

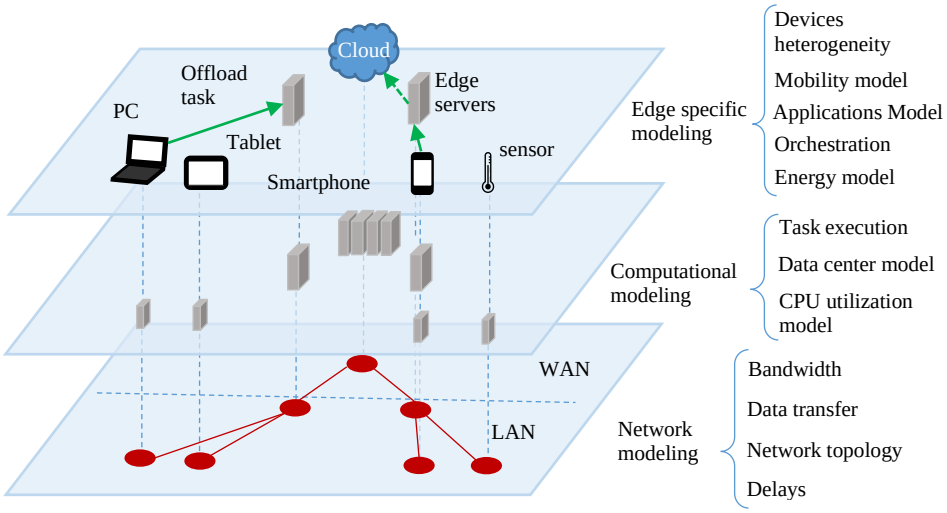

Fig. 1. The aspects of modeling Edge and Mist Computing environments

2. Related Work

In traditional Cloud Computing, devices at the edge of the network offload their tasks to the Cloud for processing them. This task offloading may be necessary for several reasons: some devices offload their tasks because of their low computing capabilities, devices with capacity-limited batteries must offload their tasks to extend their life, and so on.

Edge and Mist Computing use the same offloading process. Tasks offloading allows edge nodes to work cooperatively in order to increase system throughput [18]. In [12], the authors proposed a mechanism that offloads tasks between mobile devices to balance their power consumption. This mechanism has extended network life by 400%. A related system has been developed in [25] called Serendipity. It allows mobile devices to remotely access the resources of other devices to run their applications, which resulted in minimizing the local power consumption while reducing the overall tasks completion time by 6.6 times. A. Mukherjee et al. [15] have introduced a framework that takes advantage of smart devices available at the network edge in order to perform data analytics in IoT. To do so, capacity-based partitioning was introduced, where data is partitioned according to the capacities of those devices. Although performance has

decreased when using those devices, the workload of the Cloud has also been reduced, which can solve the Cloud scalability challenge. In [22], an energy-sensitive tasks offloading algorithm has been proposed. It allows mobile devices to dynamically choose the Cloud or the Fog to offload their tasks according to their delay tolerance and power consumption. The results show that this approach outperforms Cloud-only and Fog-only strategies. In [24], the authors have proposed a platform that orchestrates tasks between IoT gateways, Fog servers, and the Cloud, depending on the availability of resources. In [14], V. Chamola et al. focused on reducing latency in Mobile Edge Computing, by searching for the best Fog node to execute tasks when the nearest node is overloaded. This algorithm has achieved a very low latency compared to the traditional scheme that only uses the closest Fog node. Always in Mobile Edge Computing, a Fuzzy Logic based orchestration algorithm was introduced in [28]. The results show the effectiveness of adopting Fuzzy Logic. However, fuzzy logic is not applicable to unknown systems that lack information, and setting exact fuzzy rules is a complicated task. Consequently, the results may not always be correct and are perceived on the basis of assumptions.

With the advancement of research in this field, simulation began raising much interest, leading to the development of many simulation frameworks such as *iFogSim*. *iFogSim* is a *CloudSim*-based simulation framework designed to simulate Fog Computing environments [4]. It allows several types of components (sensors, actuators, gateways, etc.) to be added and linked to form a topology. Nevertheless, this topology remains static, making it lacks mobility support, which is one of the main reasons for adopting Fog Computing. Also, simulating large-scale scenarios that involve hundreds (if not thousands) of devices will require adding and linking them one by one, which is inconvenient, involves a lot of effort, and time-consuming. Moreover, a fixed delay is assigned to each link, ignoring the effect of the network load on the transmission delays.

IoTSim [21] is another simulator that is based on *CloudSim*. It simulates batch-oriented IoT applications where data is sent in large amounts to a processing system, using a MapReduce large data processing model. *SimIoT* [9] is a toolkit that simulates the communication between IoT devices and the remote Cloud. It allows the experimentation of multi-user submission dynamically in the IoT environment. Nevertheless, it does not consider the heterogeneity of IoT devices, and their energy consumption is ignored. IoT will count more than 38.6 billion devices by 2025 [1]; all of them generate data continuously, making energy consumption a major concern. *EmuFog* [6] is an emulation framework for Fog Computing environments. It allows the simulation of Docker-based applications. Since *EmuFog* is based on *MaxiNet* [8], the events of each node (including CPU and memory utilization) are saved in a log. However, because it lacks a generic interface, it cannot deal with global metrics, such as response delay.

Finally, *EdgeCloudSim* [13] is another *CloudSim*-based simulator for Mobile Edge Computing that addresses some *iFogSim* limitations. It automatically generates the required number of edge devices, making it more scalable. It supports mobility to a certain extent, and the network model is more realistic. However, its mobility model is over-simplified. It also lacks modeling the power consumption of edge devices, their remaining energy, and their death on simulation runtime (i.e., when they run out of energy). It cannot execute tasks locally on these devices or offload them to other edge devices, limiting its use to Mobile Edge Computing scenarios.

Although Mist Computing has attracted a lot of interest [12, 15, 22, 25], simulation tools are still lacking. To the best of our knowledge, there is no simulator capable of modeling Mist Computing environments (Fig. 2), which involves processing data on edge

devices, modeling their heterogeneity, measuring their energy consumption and their resources utilization, etc. Motivated by this, we introduce *PureEdgeSim*, a simulation toolkit that is designed to simulate Cloud, Edge, and Mist Computing environments. Thus, enabling the simulation of a multitude of scenarios such as Mobile Devices Clouds [12, 25], Mobile Edge Computing [13, 14], and multi-level scenarios where different computing paradigms are used simultaneously (such as Foggy [24]).

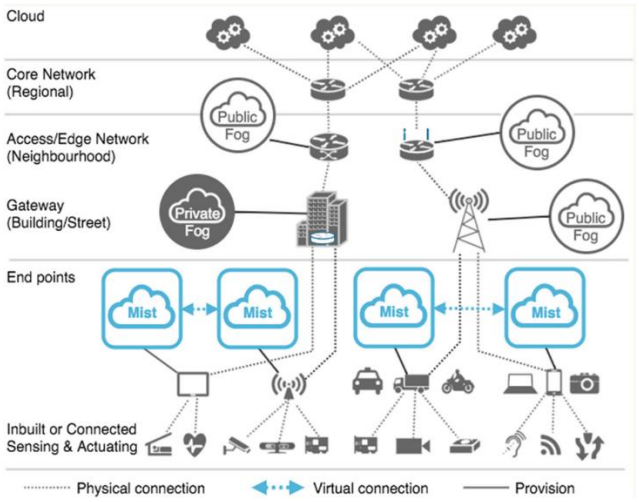

Fig. 2. The role of Mist Computing on the internet of things [26]

3. *PureEdgeSim* Architecture and Design

PureEdgeSim takes advantage of *CloudSim Plus* features [2], including the native support for the discrete events simulation, that is used during the communication between its components. It also leverages its rich and very extensible library that covers all the aspects of Cloud Computing from resources (i.e., data centers, hosts, etc.) to services (i.e., virtual machine allocation policies, CPU schedulers, etc.) enabling it to model computational tasks effectively (the middle layer of Fig. 1). Hence, only a few classes were added to model Edge and Mist Computing environments. Therefore, the *CloudSim Plus* layer is responsible for providing key components that are extended by *PureEdgeSim* (Fig. 3), and it is also behind the interactions of its components.

3.1. *PureEdgeSim* Modular Design

The simulation of Edge and Mist Computing environments allows evaluating the adopted resources management strategies before their actual deployment. However, the heterogeneity of possible scenarios complicates the task, especially when using a simulator such as *Omnet++* or *NS-3* where the user has to define all the aspects of the

simulation from the specification of resources, networking, energy, mobility, etc. which requires a lot of time and effort. To cope with it, *PureEdgeSim* follows a modular architecture that consists of seven modules, where each one of them deals with a specific Edge Computing modeling issue. To facilitate prototyping and experimentation, each module offers a default implementation with a ready to use set of adjustable parameters. These modules are:

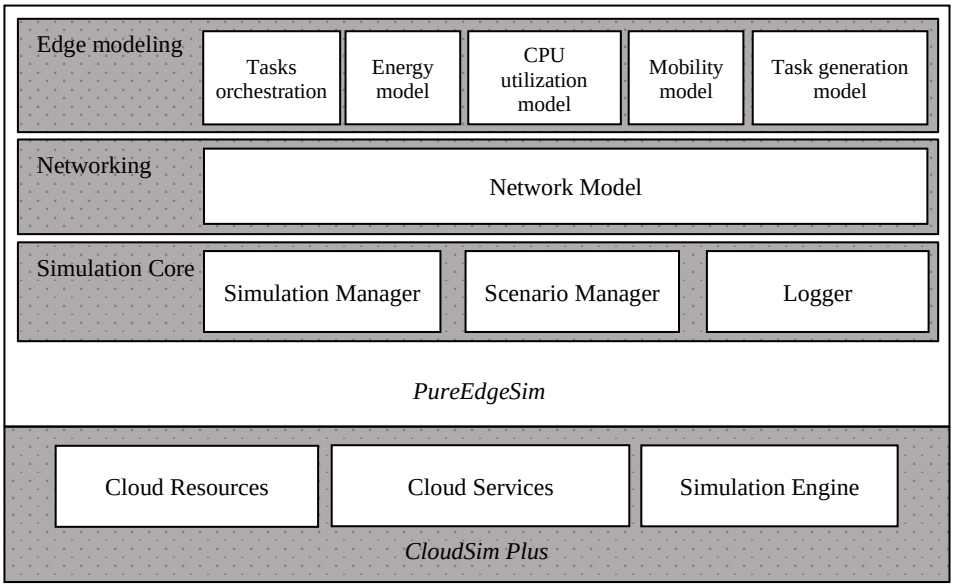

Fig. 3. *PureEdgeSim* layered architecture

Simulation Manager. This module is responsible for initiating and managing the simulation environment, scheduling events, and generating the output files. It contains three essential classes, the *Simulation Manager* class, which initializes the simulation environment, starts the simulation, and schedules its end. The second class is the *Simulation Logger*. It is responsible for generating the simulation output; it calculates the results, shows them at the end of every iteration, and saves them to a comma-separated value (CSV) format to easily exploit them later. Finally, the *Real-Time Display* class that displays real-time information such as the simulation map and other charts (network utilization, CPU utilization, and tasks failure rate). This helps to understand the course of the simulation better, and above all, to analyze the proposed solution in real-time, especially the mobility model.

Data Centers Manager Module. It consists of three classes (Fig. 4): (i) The *Edge Data Center* class that extends the *Data Center Simple* class of *CloudSim Plus* to model the heterogeneity of edge devices, (ii) the *Server Manager* that generates the required data centers and edge devices, their hosts, and their virtual machines according to the configuration files, and finally, (iii) the *Energy Model* class which is responsible for updating their energy consumption.

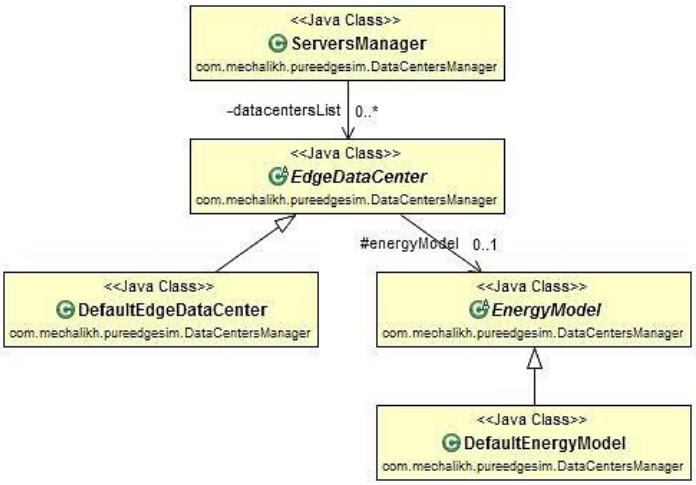

Fig. 4. The Data Centers Manager classes

Tasks Generator. *PureEdgeSim* supports the generally used applications models: the sense-process-actuate and the stream-processing models. In the first one, the data collected by sensors is sent to computing nodes for processing. The results of the processing are then sent back to the actuator to take the necessary actions. The second model involves a network of application modules that continuously process the data streams generated by sensors. The extracted information is stored for large-scale and long-term analysis [4].

In *PureEdgeSim*, the data emitted from sensors and edge devices are modeled as tasks. By default, the tasks generator assigns an application such as e-health, infotainment, and augmented reality (which can be defined in the applications XML file) to edge devices, where each application has its specific characteristics (i.e., data size, CPU utilization, latency-sensitivity, etc.). After that, it will generate the tasks of every device according to the assigned application type. This module consists of two main classes (Fig. 5): The first one is the *Task* class, which is inherited from the *Cloudlet Simple* class of *CloudSim Plus*, and the second is the *Task Generator*. The latter generates the tasks that will be offloaded during the simulation.

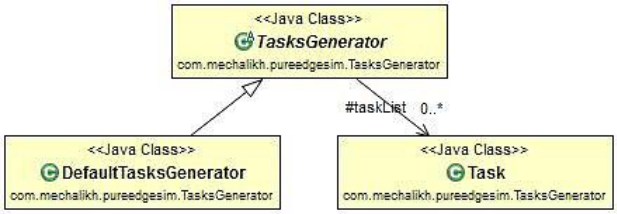

Fig. 5. The Tasks Generator classes

Location Manager. Geographical distribution and mobility are the main attributes of Edge and Mist Computing. Thanks to the geographical distribution of their computing

nodes [5], they can continue serving mobile devices during their mobility while providing the lowest latency. This has spawned a new generation of latency-sensitive applications such as connected vehicles. To support such scenarios, this module assigns an initial location to each device, and realistically manages their mobility. It contains two main classes (Fig. 6): The *Location* class, which represents the X and Y coordinate of the device, and the *Mobility* class that generates the next location for each mobile device.

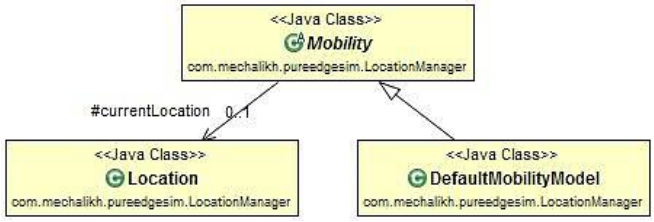

Fig. 6. The Mobility Manager classes

The Task Orchestration Module. The generation, capture, and analysis of data will be in volumes, variety, and orders of magnitude larger than before. Effective implementation of the infrastructure requires several key decisions: mainly how the data will be collected and how it will be processed. These decisions are influenced by two competing pressures: the use of the infrastructure and the Quality of Service required by the end-user [19]. Hence, the simulators must allow the implementation of custom resources management techniques in order to enable their wider applicability [17].

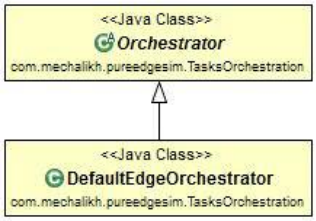

Fig. 7. The Tasks Orchestrator classes

PureEdgeSim allows this through the *Tasks Orchestration Module*. It consists mainly of the *Orchestrator* (Fig. 7), which represents the decision-maker. Depending on the used policy, it decides whether to offload the task or execute it locally and where to offload it. Users can quickly implement their orchestration policies (i.e., the tasks orchestration algorithm) by extending the *Orchestrator* class.

The Network Module. This module addresses the networking layer presented in Fig. 1. It primarily consists of the *Network Model* (Fig. 8). Unlike in *CloudSim Plus* (same for *CloudSim*), where the bandwidth allocated to each virtual machine remains static, this network model takes into account the network load at each instant of the simulation. When transferring data, at each instant of the transfer (i.e., from the beginning until the end), its allocated bandwidth will vary based on the network load at that moment. This network model also takes into consideration the bandwidth limit caused by WAN or

WLAN congestion. As a result, if multiple devices connect to the same WLAN access point, the bandwidth allocated to each device decreases. If this allocated bandwidth is below that of the WAN, the transfer speed of the data sent to (or received from) the Cloud will be limited by the low bandwidth of the WLAN and not by that of the WAN.

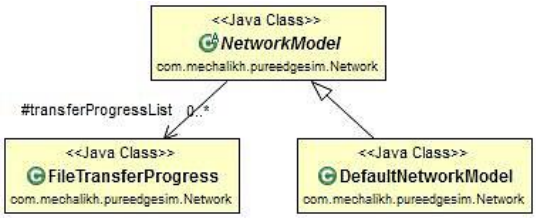

Fig. 8. The Network Model classes

The Scenario Manager Module. Each use case requires a heterogeneous combination of devices. The heterogeneity involves the devices mobility, energy source, computing capacity, the heterogeneity of applications, and their requirements (e.g., latency). Therefore, the simulation framework must be able to support the diversity of devices and their different Quality of Service requirements [17, 19]. Besides, Edge Computing simulators need to be easily extended with new types of devices and applications without modifying their internals [17]. The *Scenario Manager* module guarantees this by loading the simulation parameters and the user scenario from the input files. It contains two principal classes, the *Simulation Parameters* class, which acts as a placeholder for the different parameters, and the *Files Parser* that loads the user scenario and settings from specific configuration files representing the input method.

3.2. The Simulation Duration and Realism

As mentioned previously, *PureEdgeSim* relies on *CloudSim Plus*, one of the commonly used Cloud Computing simulators that provides a reliable code base for modeling computational tasks. To fulfill the remaining simulation requirements (Fig. 1), *PureEdgeSim* offers the most realistic network, energy, and mobility models as compared to existing solutions. However, since it is a discrete event simulator, the simulation time complexity will depend on the number of events generated at runtime, as seen in Fig. 9, where there is a clear correlation between the number of events and the duration of the simulation. To reduce simulation time, *PureEdgeSim* offers a quick and full control of the simulation environment through its set of parameters, where users can trade-off between simulation realism and duration. Hence, the realism will depend on the user settings, especially the update intervals (Table 3). The shorter these intervals, the more accurate and realistic the simulation will be, but also, the longer it will take (Fig. 9).

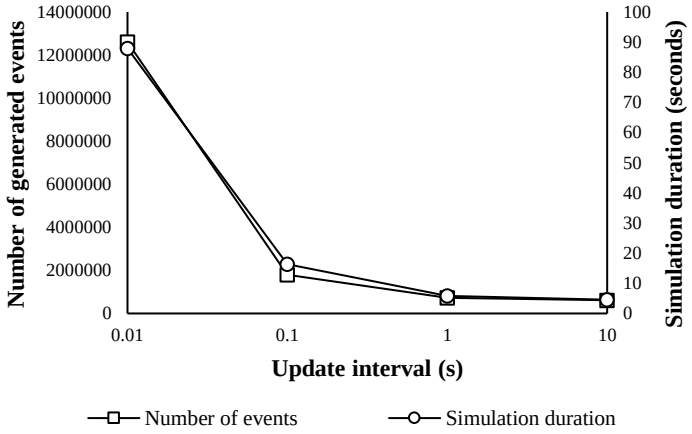

Fig. 9. The impact of update interval on the number of generated events and the simulation duration

3.3. Ease of Use and Extensibility

Simulators can be used to compare the performance of different configurations in order to determine the factors that affect performance the most, e.g., the network settings, the number of entities used in the simulation, the amount of resources, etc. Treating all these variables programmatically is a challenge. To reduce time and effort, each module provides a default implementation (e.g., the *Default Edge Data Center* class in the *Data Centers Manager* (Fig. 4), the *Default Mobility Model* (Fig. 6), etc.). These ready-to-use models also offer a fully customizable environment through a multitude of parameters and configuration files, allowing users to customize the components behavior without changing the original code.

Extensibility is another essential feature of *PureEdgeSim*. Even though *PureEdgeSim* uses its pre-built models by default, users can always create and integrate their custom models when building their simulation scenarios if any of these default implementations do not meet their needs, without having to modify the *PureEdgeSim* code base.

As a result, the simulation scenario can be quickly built by following these simple steps:

a) The implementation of custom models, if needed.
b) The definition of Cloud and Edge resources and the application characteristics: This can be done by editing the following configuration files that are located under the *settings/* folder:

- The Cloud data centers file: In this file, the user defines the Cloud data centers, their power consumption rate, describes their hosts (CPU, storage, ram), and the virtual machines of each of these hosts.
- The Edge data centers (i.e., Cloudlets, servers) file: Like the Cloud data centers file, this file defines the edge data centers, their specifications, their locations, and their energy consumption rates.

```
<?xml version="1.0"?>
<edge_devices>
    </device>
                    <mobility>false</mobility>
                    <speed>0</speed>
                    <battery>true</battery>
                    <percentage>20</percentage>
                    <batteryCapacity>56.2</ batteryCapacity >
                    <idleConsumption>1.7</idleConsumption>
                    <maxConsumption>23.6</maxConsumption>
                    <generateTasks >false</generateTasks >
                    <hosts>
                            <host>
                                    <core>8</core>
                                    <mips>110000</mips>
                                    <ram>8192</ram>
                                    <storage>1048576</storage>
                                    <VMs>
                                            <VM>
                                                    <core>8</core>
                                                    <mips>110000</mips>
                                                    <ram>8192</ram>
                                                    <storage>1048576</st
                                            </VM>
                                    </VMs>
                            </host>
                    </hosts>
```

Fig. 10. The edge devices XML file

- The edge devices file: Instead of defining the devices one by one (which takes a considerable amount of time and effort due to the large number of devices), in *PureEdgeSim*, the user will only define the types of devices and the percentage of each one of them. Then, according to this percentage and the total number of devices (which is set in the simulation parameters file), the *Data Centers Manager* will generate the devices of each type. To support the heterogeneity of devices, *PureEdgeSim* offers endless possibilities, varying from simple sensors (i.e., without any computing capacity) to smartphones, laptops, as well as sophisticated servers that run hundreds of virtual machines. It can be done by specifying the mobility of the device (i.e., whether the devices of this type are mobile), the power source (i.e., if they are battery-powered), the capacity of the battery in watt-hour if it is battery-powered, the power consumption rates in watt-Hour, and the computing capacity in MIPS (Million Instructions Per Second). Fig. 10 gives an example of a device type (see the laptop type in Table 1).
- The applications file: This file defines the set of IoT applications that are needed by the *Tasks Generator* (see Table 2). Each application is defined by: the usage percentage that represents the proportion of devices running that application, the generation rate which is the number of tasks generated per minute, its delay tolerance in seconds, the task length which refers to the number of its instructions in MI (Million Instructions) and determines its

execution time, the size in Kbytes of the offloading request, the container image, and the returned results.

c) Setting the simulation parameters: To ease the implementation of simulation scenarios, each of the previous modules provides a set of adjustable parameters (Table 3), that can be found in the *simulation_parameters.properties* file.

To demonstrate the capabilities of *PureEdgeSim*, especially when it comes to the simulation of Edge Computing environments and its support for the heterogeneity of both edge devices and scenarios, in the next section, we propose a case study.

4. Application and Case Study

As proof of concept, in this section, we propose a simulation scenario. We will focus on aspects that can only be modeled by *PureEdgeSim*, such as the support for Mist Computing scenarios, network utilization, mobility of devices, and their energy consumption. First, we will introduce a tasks orchestration platform. We also propose a tasks orchestration algorithm that is based on Fuzzy Decision Tree. Finally, a simulation scenario is given by which we evaluate their performance.

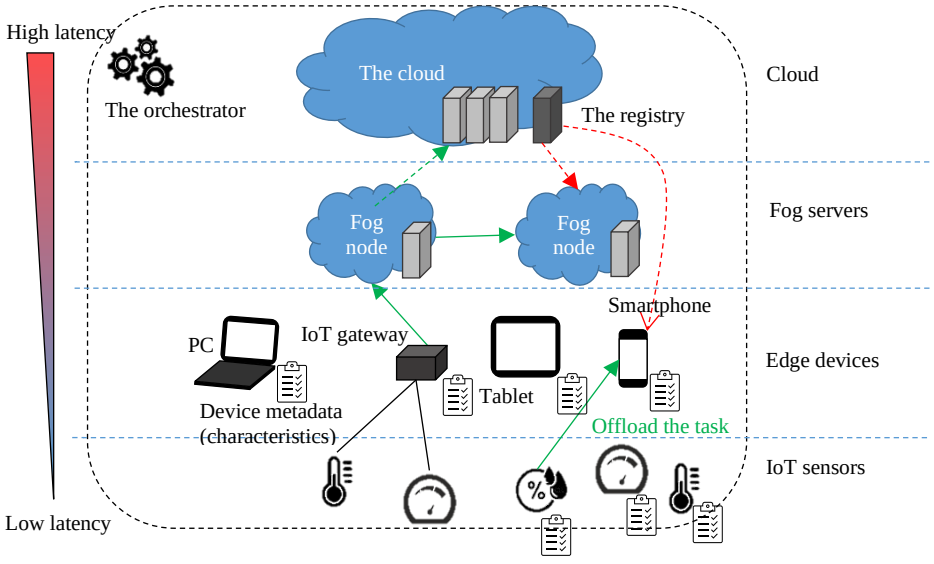

Fig. 11. The proposed architecture

4.1. The Tasks Orchestration Platform

We introduce a multi-tier architecture (Fig. 11). It consists of: The IoT sensors layer, that represents the source of data, the smart edge devices layer, the Fog layer, and the Cloud one. By relying on close devices equipped with enough computing capacity (PCs, smartphones, tablets, smart TVs, etc.), it enables QoS compliance and better scalability.

While these devices may offer low computing capabilities compared to the Fog or the Cloud, their potential lies in their ever-growing number, which is expected to exceed 10 billion by 2025 (excluding IoT sensors) [1]. Additionally, they can deliver the lowest latency and support mobility to some extent, given their massive geographical distribution and location (i.e., a hope away from each other). Hence, latency-sensitive applications can be placed in these devices, while computationally intensive ones can be placed on the Cloud or Fog servers based on their delay tolerance.

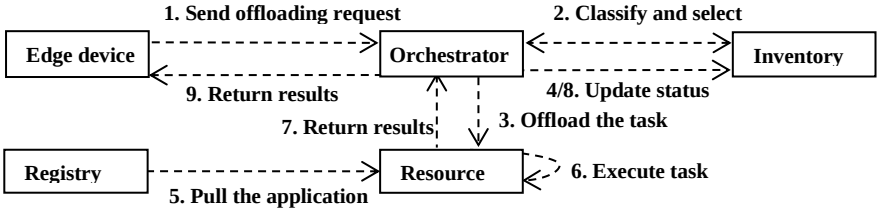

Fig. 12. The task offloading flow and the role of the orchestrator

It consists of the following entities (Fig. 12):

a) IoT sensors, which are resource-limited devices. They must offload their tasks elsewhere for processing. For this purpose, a task offloading request will be sent to the orchestrator. The request provides information about the status of the device and its requirements, e. g. application ID (i.e., container ID), data to be processed, task latency sensitivity, that are necessary to find the best offloading destination.
b) The application: A piece of software hosted on a container (e.g., Node.js script).
c) The registry: The repository from where the required application will be pulled.
d) The resource: The potential destination of the task, where it will be executed.
e) The orchestrator (i.e., the decision-maker): decides where the task will be executed, using a specific orchestration algorithm. This algorithm will also be hosted on a container, to facilitate its download, execution, and update.
f) The inventory (i.e., the list of available resources): When a device joins the network, it communicates its meta-data (Fig. 11), including its resources and its remaining energy, to the orchestrator. The orchestrator will add this device to its inventory. Then, when receiving a task offloading request, it will classify the resources (its inventory) to find the one that best suits this task.

To minimize energy consumption and delays, a decentralized orchestration strategy will be used, in which, the orchestrators will be selected using a cluster head selection.

4.2. The Orchestration Algorithm

The proposed architecture can guarantee a high quality of service. However, managing these heterogeneous resources is not an easy task. Several factors, such as the remaining power, resource utilization, and network condition, should be taken into account. These dynamic factors can vary unexpectedly. Traditional multi-constraint optimization cannot be applied due to insufficient information about the nature of the tasks and arrival times, which necessitates an online solution that can adapt to this ever-changing environment.

To guarantee this, we present a tasks orchestration algorithm that is based on the fuzzy decision tree. Fuzzy decision trees combine the advantages of fuzzy logic and decision tree, among which are: it can handle uncertainties without requiring a complex mathematical model and support multi-criteria decision processes, its computing complexity is low, which is essential for an online decision algorithm, and it requires less preparation effort. This algorithm is based on Yuan et al. [29] fuzzy decision tree approach, and consists of two stages (Algorithm 1):

Algorithm 1. The proposed algorithm

```
1.  procedure findDestination(request, inventory)
2.  type ← fuzzyDecisionTree1.classify (request)
3.  if (type = 'Cloud')
4.     offload(offloadingRequest, Cloud)
5.  else if (type = 'Fog)
6.     offload(offloadingRequest, Fog)
7.  else
8.        maxTruthLevel ← −1
9.        destination ← null
10.       finestClass ← low
11.       for each resource ∈ inventory do
12.          (class, truthLevel) ← fuzzyDecisionTree2.classify(resource)
13.          if (class > finestClass) //get the optimal destination
14.             finestClass ← class
15.             maxTruthLevel ← truthLevel
16.             selectedDevice ← resource
17.          else if (class = finestClass)
18.             newTruthLevel ← truthLevel
19.             if (maxTruthLevel = −1 or newTruthLevel > maxTruthLevel)
20.                maxTruthLevel ← newTruthLevel
21.                selectedDevice ← resource
22.             end_if
23.          end_if
24.       end_for
25.       offload(request, selectedDevice)
26.    end_if
27. end
```

a) The first stage: During this stage, the tasks are classified into Cloud, Fog, or Mist tasks based on the following criteria: (i) the task latency-sensitivity: the reason for adopting Edge and Mist computing; if a task is tolerant to delay it will be sent to the Cloud, otherwise to the Fog or edge devices depending on Fog utilization and the device mobility. (ii) Fog resources utilization: Fog servers are not supposed to be as powerful as the Cloud; for this reason, they may be overloaded. This may lead to high delays, causing the failure of many tasks. In this case, the Cloud can be a good alternative if the WAN bandwidth is high enough. (iii) Device mobility: if the device offloading the task is mobile, edge devices should be avoided. (iv) WAN bandwidth: if it is below a certain threshold, the Cloud should be avoided.

b) The second stage: If the Cloud or the Fog has been chosen in the first stage, the task will be directly offloaded. However, if the choice has been made on edge devices, the algorithm will classify them during the second stage to find the most suitable one. This classification will be based on the following criteria: (i) The utilization of device resources. (ii) Energy source: battery-powered devices should be skipped when possible. (iii) The mobility of both devices: The failure of a task due to mobility happens when a device (the one offloading the task or the one executing it)

relocate before finishing it. Thus, offloading the task to a mobile device may cause its failure. This risk of failure becomes even higher when both devices are mobile.

4.3. The Simulation Scenario

To demonstrate the capabilities of *PureEdgeSim*, we introduce a simulation scenario that imitates a smart university campus. A smart campus consists of integrating information and communication technologies by deploying sensors in several locations to get useful information that will be exploited to manage and optimize resources, increase energy efficiency, and improve education [23]. In this scenario, students who own mobile devices (e.g., smartphones) relocate after a random amount of time. To do this, the *Default Mobility Model* will be used. The area also involves other devices (Table 1): simple sensors (e.g., wearables) and non-mobile devices (e.g., laptops and gateways).

Table 1. The types of Edge devices

Edge devices types	Laptop	Smartphone	IoT Gateway	Sensor
Mobility	No	Yes	No	No
Speed (meters per second)	0	1.4	0	0
Battery-powered	Yes	Yes	No	No
Generate tasks	No	Yes	No	Yes
Percentage of devices (%)	20	30	10	40
Battery-capacity (Wh)	56.2	18.75	-	-
Idle energy consumption rate (Wh)	1.7	0.078	1.6	0.036
Max energy consumption rate (Wh)	23.6	3.3	5.1	-
CPU (GIPS)	70	25	16	-
CPU Cores	8	8	4	-
Ram (Gbyte)	8	4	2	-
Storage (Gbyte)	1024	128	32	-

The last type refers to simple sensors that do not have sufficient computing capacity (only generate data/tasks). The energy consumption rates were measured from the following devices under different workloads: a laptop running Windows 10 (Intel® processor Core™ i7-8550U), a smartphone running Android 10 (HiSilicon Kirin 710), and a Raspberry Pi 3 Model B+ running Raspbian. On the other hand, the CPU values are obtained by running a Dhrystone benchmark [27].

To extend their lives, battery-powered devices should, if possible, offload their tasks, while devices with computationally intensive tasks should offload them to minimize execution time. The *Proposed* multi-tier architecture will be evaluated against:

1. The *Cloud-Only* architecture where all the tasks are offloaded to the Cloud.
2. The widely adopted *Fog-and-Cloud* architecture: in this case, the tasks can be offloaded either to the Cloud or Fog servers.

The Cloud will be represented by one data center with a total of 4000 GIPS, distributed over 16 virtual machines, while the Fog will have a similar data center but with lower computing capacity (3200 GIPS), which should be enough for this small simulation area.

To model the different possibilities, four types of applications with different characteristics are used (Table 2): (i) A health application: The data generated by wearables will not exceed a few kilobytes and will not use significant processing power

[28]. (ii) An augmented reality application [30]: The data sent is an image, usually about 1 Mbyte in size. (iii) Other heavy computing tasks requiring additional computing power, e.g., machine learning, may also be included [30]. (iv) An infotainment application [31].

Table 2. The types of applications

Applications types	**Health**	**Augmented reality**	**Computation-intensive**	**Infotainment**
Usage percentage (%)	20	30	20	30
Generation rate (tasks per minute)	20	20	2	4
Latency sensitivity	Yes	Yes	No	No
Task length in Giga Instructions (GI)	1500	5	50	10
Request size (Kbytes)	20	1500	3000	50
Results size (Kbytes)	20	50	200	50

When offloading the task, the orchestrator will choose the offloading destination using one of the following orchestration algorithms:

1. *Proposed*: The proposed Fuzzy Decision Tree based algorithm.
2. *ECOOA:* The energy-oriented tasks orchestration algorithm [22], which is the closest in terms of criteria.
3. *Fuzzy Logic*: The Fuzzy Logic based tasks orchestration algorithm [28].

The simulation parameters for this scenario are resumed in Table 3.

Table 3. The simulation parameters

Parameter	Value
Simulation duration	30 (min)
Update interval	0.01 (s)
Min number of Edge devices	100
Max number of Edge devices	500
Edge devices counter step size	100
Edge devices range	10 (meters)
Simulation area size	200 x 200 (meters)
Network update interval	0.1 (s)
WLAN bandwidth	300 (Mbits/s)
WAN bandwidth	20 (Mbits/s)
WAN propagation delay	0.2 (s)
Orchestrators deployment	Decentralized.
Orchestration algorithm	Proposed, ECOOA, Fuzzy Logic.
Architectures	Cloud-Only, Fog-and-Cloud, Proposed.

5. Simulation Results and Discussion

To demonstrate the effectiveness of *PureEdgeSim* in modeling Cloud, Edge, and Mist computing environments, in this section, the proposed platform will be evaluated against existing solutions. Although *PureEdgeSim* can generate charts automatically, the figures presented in this section have been created from the output CSV file using Microsoft Excel. This file offers more than 60 different ready-to-use metrics, from which the user can also derive others as well. For instance, the network delay in Fig. 13 is obtained by dividing the total network utilization by the number of sent tasks. Similarly, the service

time is the sum of network delay and execution time. During this evaluation, we will focus on meaningful metrics that determine the Quality of Service and reflects the scalability of the proposed solution, such as the tasks failure rate, energy consumption, delays, network usage, and CPU utilization.

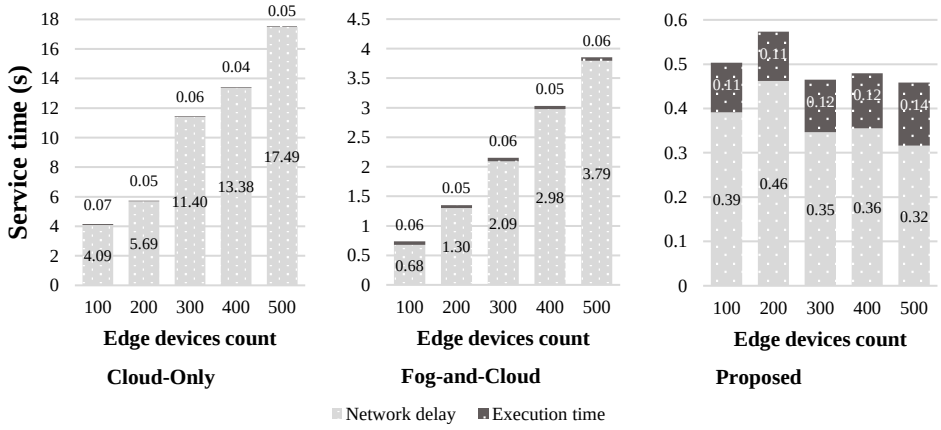

Fig. 13. Average tasks service time

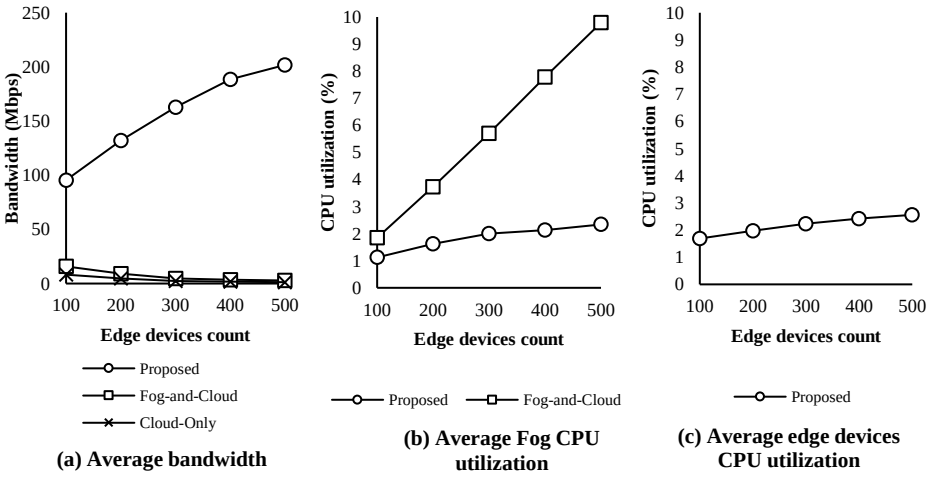

Fig. 14. A comparison between the different architecture

5.1. Evaluating the Architectures

The average service time, which consists of execution time and network delay, is given in Fig. 13. The latter is considered an essential factor that has a direct effect on the Quality of Service. When using the *Cloud-Only* architecture in which all the tasks are offloaded

to the remote Cloud, the service time has been very long, although the Cloud provides the highest computing capacity, due to the high use of the backhaul network. By using the Fog along with the Cloud, the delay has been reduced remarkably. However, it increases as the number of devices grows. On the other hand, the tasks completion delay stayed almost stable when using the proposed architecture; despite their low computing capacity, the use of edge devices has managed to decrease the average delay regardless of the number of devices to an average of 0.5 seconds.

Fig. 14 shows the average CPU usage of the Fog and edge devices, as well as the allocated bandwidths using the different architectures. Since all tasks are transferred to the Cloud when using the *Cloud-Only* architecture, the backhaul network becomes overloaded, resulting in the lowest allocated bandwidth, as shown in Fig. 14 (a). This low bandwidth justifies the high service time depicted in Fig. 13. The use of Fog servers has managed to double the average allocated bandwidth for each task. However, the Fog CPU utilization rose dramatically when the number of devices grows (Fig. 14(b)). This increase will force it to offload their tasks surplus to the Cloud, thus, raising the network utilization again, as seen in Fig. 14(a).

Thanks to its horizontal scalability, Mist Computing benefits from the rapid growth of devices. The growth of devices, in this case, means the availability of more resources. Thus, the workload remains stable regardless of the number of devices, which is confirmed by Fig. 14(c), where the average CPU usage of edge devices has remained stable at around 2% when the proposed multi-tier architecture was used. As edge resources grow, there is less need to offload tasks to the Fog or the remote Cloud, resulting in low Fog CPU usage, and an increase in allocated bandwidth.

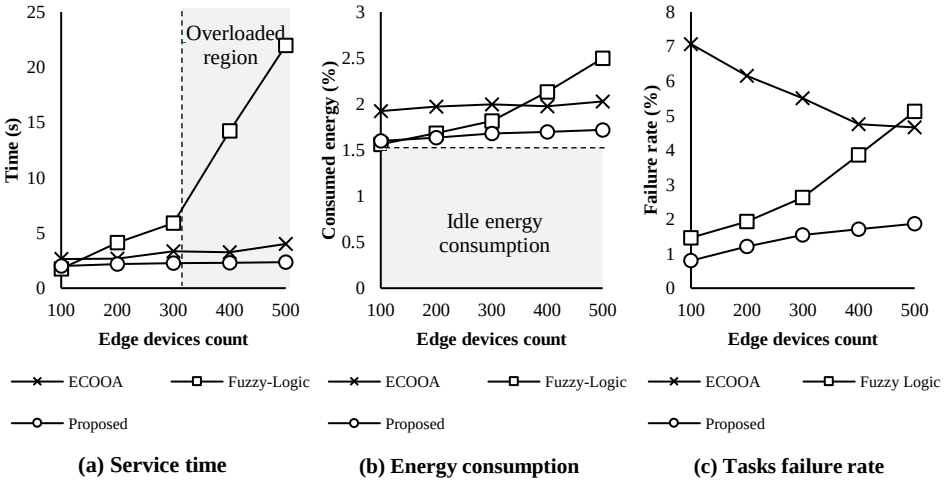

Fig. 15. A comparison between the different algorithms in terms of service time, energy consumption, and failure rate

5.2. Evaluating the Orchestration Algorithms

Fig. 15 compares the different algorithms in terms of service time, the power consumption of battery-powered devices, and tasks failure rate, which are the main performance criteria. The *Fuzzy-Logic* algorithm offloads delay-tolerant tasks to the Cloud when network bandwidth is sufficient, while the other ones are offloaded to the Fog or edge devices as long as their utilization is low. When the number of devices grows (the overloaded area that is highlighted in gray in Fig. 15(a)), the Cloud workload increases. To compensate, this algorithm will offload the excess to the Fog and edge devices, increasing their utilization and consuming their power. When their CPU utilization exceeds a certain threshold, the algorithm will switch to the Cloud regardless of bandwidth, resulting in long service time. This high delay has caused the failure of many tasks, which justifies the correlation between all these three charts. The *ECOOA* has performed better since it aims to minimize delays and energy consumption. However, because it does not distinguish between tasks, a significant portion of latency-sensitive tasks were offloaded to the Cloud. As a result, higher tasks failure rate.

Besides avoiding mobile devices, the proposed algorithm avoids battery-powered devices as much as possible. As a result, the energy consumption of those devices has been reduced by 79.9% as compared to competitor algorithms (if we exclude the idle energy consumption highlighted in gray in Fig. 15(b)). Because it avoids those devices, a considerable number of tasks are offloaded to the Fog (as long as its utilization is low), reducing the service time by 50.8%, and the failure rate by 60% compared to the closest competitor algorithm.

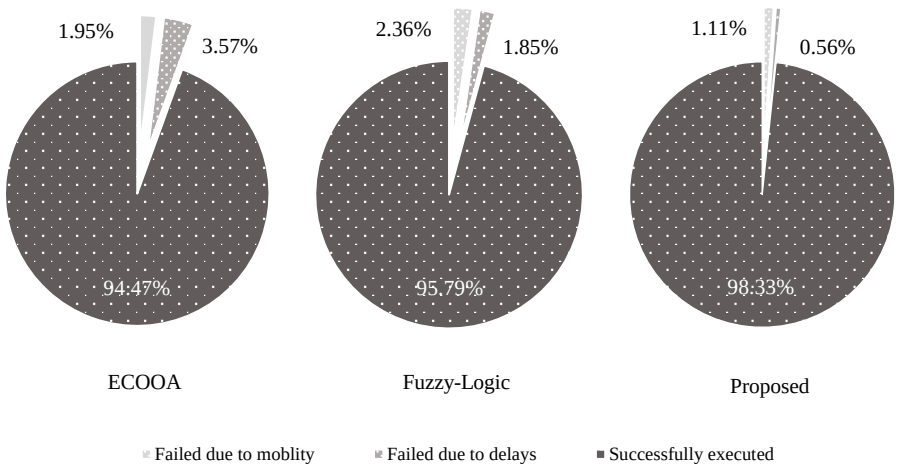

Fig. 16. The tasks failure reasons (400 devices)

The rates of the tasks that are failed due to mobility and high delays are depicted in Fig. 16. As opposed to competitor algorithms, the proposed one considers the heterogeneity of devices and the requirements of their applications. As a result, if a task is not latency-sensitive, it will be offloaded to the Cloud if available. Otherwise, it will

be offloaded to the Fog or another edge device. However, because mobile devices are avoided as much as possible, more tasks have been offloaded to the Fog. Hence, reducing failure due to mobility by 43.2%, and since it has decreased the service time, as seen in Fig. 15 (a), the failure due to high delays has also been reduced by 69.8% compared to competitor algorithms.

6. Conclusion and Future Work

Edge and Mist Computing are two emerging computing paradigms that bring Cloud applications close to IoT devices. As a result, decreasing the latency and leading to a more scalable network. In such complex systems, the simulation makes it possible to evaluate the adopted strategy and to analyze its performance before its deployment. Motivated by this, in this paper, we introduced *PureEdgeSim*, a simulation toolkit designed to simulate the Cloud, Edge, and Mist Computing environments and to evaluate their performances. To demonstrate its effectiveness, a case study was introduced. We focused on the aspects that can only be simulated using *PureEdgeSim*, such as the support for the heterogeneity of devices, support for mobility, the realistic network model, etc. which reflects the effectiveness of *PureEdgeSim* and its extensibility.

To conclude, Fog servers will only delay the scalability problem rather than resolving it permanently. They suffer from that same issue as the conventional Cloud, which requires continuously scaling them up to accommodate the rapid growth of connected devices. On the other hand, Mist Computing represents a cost-effective alternative that makes the growth in the number of connected devices in its favor, thanks to its horizontal scalability. With the appropriate pricing model, the latter could easily help to overcome the Cloud limitations. However, due to the heterogeneity of IoT devices, conventional optimization techniques are not sufficient. Other aspects, such as the mobility of the devices, their residual energy, and the latency sensitivity of its application, must be taken into account. By considering them, the proposed orchestration algorithm, which is based on the Fuzzy Decision Tree, outperformed the state-of-the-art solutions in all aspects of the comparison. It has reduced the tasks failure rate by 60%, the energy consumption by 79.9%, and service time by 50.8%, thanks to its set of criteria.

As future work, we are planning to add a pricing model to this simulator and the support for virtual machines migration as well. By introducing this simulator, we hope that this modest work encourages the adoption of Mist Computing in the Internet of Things and enables the development of novel resource management strategies.

Software Availability

The *PureEdgeSim* simulator and the examples are available for download at: https://github.com/CharafeddineMechalikh/PureEdgeSim

References

1. Strategy Analytics (2019). Strategy Analytics: Internet of Things Now Numbers 22 Billion Devices But Where Is The Revenue?.
2. Silva Filho, M. C., Oliveira, R. L., Monteiro, C. C., Inácio, P. R., & Freire, M. M. (2017). 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.
3. Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R. (2011). CloudSim: a toolkit for modeling and simulation of Cloud Computing environments and evaluation of resource provisioning algorithms. Software: Practice and experience, 41(1), 23-50.
4. Gupta, H., Vahid Dastjerdi, A., Ghosh, S. K., & Buyya, R. (2017). 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, 47(9), 1275-1296.
5. Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012, August). 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.
6. Mayer, R., Graser, L., Gupta, H., Saurez, E., & Ramachandran, U. (2017). Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures. In 2017 IEEE Fog World Congress (FWC) (pp. 1-6). IEEE.
7. Riley, G. F., & Henderson, T. R. (2010). The ns-3 network simulator. In Modeling and tools for network simulation (pp. 15-34). Springer, Berlin, Heidelberg.
8. Wette, P., Dräxler, M., Schwabe, A., Wallaschek, F., Zahraee, M. H., & Karl, H. (2014). Maxinet: Distributed emulation of software-defined networks. In 2014 IFIP Networking Conference (pp. 1-9). IEEE.
9. Sotiriadis, S., Bessis, N., Asimakopoulou, E., Mustafee, N., Towards simulating the internet of things. 2014 28th International Conference on Advanced Information Networking and Applications Workshops; 2014; Victoria, Canada.
10. Marín-Tordera, E., Masip-Bruin, X., García-Almiñana, J., Jukan, A., Ren, G. J., & Zhu, J. (2017). Do we all really know what a Fog node is? Current trends towards an open definition. Computer Communications, 109, 117-130.
11. Dogo, E. M., Salami, A. F., Aigbavboa, C. O., & Nkonyana, T. (2019). Taking cloud computing to the extreme edge: A review of mist computing for smart cities and industry 4.0 in Africa. In Edge Computing (pp. 107-132). Springer, Cham.
12. Mtibaa, A., Fahim, A., Harras, K. A., & Ammar, M. H. (2013). Towards resource sharing in mobile device Clouds: Power balancing across mobile devices. In ACM SIGCOMM Computer Communication Review (Vol. 43, No. 4, pp. 51-56). ACM.
13. Sonmez, C., Ozgovde, A., & Ersoy, C. (2018). Edgecloudsim: An environment for performance evaluation of edge computing systems. Transactions on Emerging Telecommunications Technologies, 29(11), e3493.
14. Chamola, V., et al. (2017). Latency aware mobile task assignment and load balancing for edge Cloudlets. In Pervasive Computing and Communications Workshops (PerCom Workshops), 2017 IEEE International Conference on (pp. 587-592). IEEE.
15. Mukherjee, A., et al. (2014). Angels for distributed analytics in iot. In Internet of Things (WF-IoT), 2014 IEEE World Forum on (pp. 565-570). IEEE.
16. Ai, Y., Peng, M., & Zhang, K. (2018). Edge Computing technologies for Internet of Things: a primer. Digital Communications and Networks, 4(2), 77-86.
17. Kecskemeti, G., Casale, G., Jha, D. N., Lyon, J., & Ranjan, R. (2017). Modelling and simulation challenges in internet of things. IEEE Cloud Computing, 4(1), 62-69.
18. Aazam, M., & Huh, E. N. (2015). Fog Computing micro datacenter based dynamic resource estimation and pricing model for IoT. In Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference on (pp. 687-694). IEEE.

19. Svorobej, S., Takako Endo, P., Bendechache, M., Filelis-Papadopoulos, C., Giannoutakis, K. M., Gravvanis, G. A., ... & Lynn, T. (2019). Simulating Fog and Edge Computing scenarios: An overview and research challenges. Future Internet, 11(3), 55.
20. Varga A, Hornig R. An overview of the OMNeT++ simulation environment. In: Simutools '08 Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshop; 2008; Marseille, France.
21. Zeng, X., Garg, S. K., Strazdins, P., Jayaraman, P. P., Georgakopoulos, D., & Ranjan, R. (2017). IOTSim: A simulator for analysing IoT applications. Journal of Systems Architecture, 72, 93-107.
22. Zhao, X., Zhao, L., & Liang, K. (2016). An Energy Consumption Oriented Offloading Algorithm for Fog Computing. In International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness (pp. 293-301). Springer, Cham
23. Fernández-Caramés, T. M., & Fraga-Lamas, P. (2019). Towards Next Generation Teaching, Learning, and Context-Aware Applications for Higher Education: A Review on Blockchain, IoT, Fog and Edge Computing Enabled Smart Campuses and Universities. Applied Sciences, 9(21), 4479.
24. Santoro, D., Zozin, D., Pizzolli, D., De Pellegrini, F., & Cretti, S. (2017). Foggy: a platform for workload orchestration in a Fog Computing environment. In Cloud Computing Technology and Science (CloudCom), 2017 IEEE International Conference on (pp. 231-234). IEEE.
25. Shi, C., Lakafosis, V., Ammar, M. H., & Zegura, E. W. (2012). Serendipity: enabling remote computing among intermittently connected mobile devices. In Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing (pp. 145-154). ACM.
26. Liyanage, M., Chang, C., & Srirama, S. N. (2018). Adaptive mobile Web server framework for Mist computing in the Internet of Things. International Journal of Pervasive Computing and Communications.
27. Weicker, R. P. (1984). Dhrystone: a synthetic systems programming benchmark. Communications of the ACM, 27(10), 1013-1030.
28. Sonmez, C., Ozgovde, A., & Ersoy, C. (2019). Fuzzy workload orchestration for edge computing. IEEE Transactions on Network and Service Management, 16(2), 769-782.
29. Yuan, Y., & Shaw, M. J. (1995). Induction of fuzzy decision trees. Fuzzy Sets and systems, 69(2), 125-139.
30. Dong, Z. Y., Zhang, Y., Yip, C., Swift, S., & Beswick, K. (2020). Smart campus: definition, framework, technologies, and services. IET Smart Cities, 2(1), 43-54.
31. Guo, J., Song, B., He, Y., Yu, F. R., & Sookhak, M. (2017). A survey on compressed sensing in vehicular infotainment systems. IEEE Communications Surveys & Tutorials, 19(4), 2662-2680.