




Article

VirtFogSim: A Parallel Toolbox for Dynamic Energy-Delay Performance Testing and Optimization of 5G Mobile-Fog-Cloud Virtualized Platforms

Michele Scarpiniti * , Enzo Baccarelli  and Alireza Momenzadeh 

Department of Information Engineering, Electronics and Telecommunications (DIET),
Sapienza University of Rome, via Eudossiana 18, 00184 Rome, Italy; enzo.baccarelli@uniroma1.it (E.B.);
alireza.momenzadeh@uniroma1.it (A.M.)

* Correspondence: michele.scarpiniti@uniroma1.it; Tel.: +39-06-44585869

Received: 14 February 2019; Accepted: 14 March 2019; Published: 19 March 2019



Abstract: It is expected that the pervasive deployment of multi-tier 5G-supported Mobile-Fog-Cloud technological computing platforms will constitute an effective means to support the real-time execution of future Internet applications by resource- and energy-limited mobile devices. Increasing interest in this emerging networking-computing technology demands the optimization and performance evaluation of several parts of the underlying infrastructures. However, field trials are challenging due to their operational costs, and in every case, the obtained results could be difficult to repeat and customize. These emerging Mobile-Fog-Cloud ecosystems still lack, indeed, customizable software tools for the performance simulation of their computing-networking building blocks. Motivated by these considerations, in this contribution, we present *VirtFogSim*. It is a MATLAB-supported software toolbox that allows the *dynamic joint optimization* and *tracking* of the energy and delay performance of Mobile-Fog-Cloud systems for the execution of applications described by general Directed Application Graphs (DAGs). In a nutshell, the main peculiar features of the proposed *VirtFogSim* toolbox are that: (i) it allows the *joint dynamic* energy-aware optimization of the placement of the application tasks and the allocation of the needed computing-networking resources under *hard constraints* on acceptable overall execution times; (ii) it allows the *repeatable* and *customizable* simulation of the resulting energy-delay performance of the overall system; (iii) it allows the *dynamic tracking* of the *performed resource allocation* under time-varying operational environments, as those typically featuring mobile applications; (iv) it is equipped with a user-friendly *Graphic User Interface* (GUI) that supports a number of graphic formats for data rendering; and (v) its MATLAB code is optimized for running atop *multi-core* parallel execution platforms. To check both the actual optimization and scalability capabilities of the *VirtFogSim* toolbox, a number of experimental setups featuring different use cases and operational environments are simulated, and their performances are compared.

Keywords: 5G-supported Mobile-Fog-Cloud platforms; energy and delay-constrained mobile applications; modeling and dynamic optimization; simulation and parallel execution

1. Introduction

Modern mobile devices are equipped with a number of both heterogeneous Network Interface Cards (NICs) and multimedia sensors that allow them to host emerging perception-related applications, such as face/gesture detection/classification, visual text translation, fusion of sensed data, and video image processing, just to cite a few. Typically, these applications are delay-sensitive and computation-intensive, while the computing and battery capacities of current mobile devices are still limited. In order to cope with these limitations, the so-called Mobile Cloud Computing (MCC)

paradigm proposes to offload computation-intensive tasks to the remote Cloud for the execution [1]. However, task offloading towards remote large-size cloud-based data centers undergoes large network latencies, mainly induced by the limited bandwidth still offered by multi-hop cellular Wide Area Networks (WANs) [1]. In principle, a more appealing approach could be, indeed, to allow mobile devices to exploit both the simultaneous utilization of the hosted NICs and the sub-millisecond network latencies featuring the forthcoming *Fifth-Generation* (5G) communication technology, in order to distribute task offloading suitably to *both* the remote Cloud and *nearby* small-sized virtualized data centers, generally referred to as *Fog nodes* [2].

The reason why the convergence of the three-pillar paradigm of Fog Computing (FC) [3], Cloud Computing (CC) [4] and 5G [5,6] is expected to improve both the energy and delay performance of task offloading of computing-intensive mobile applications relies on their complementary native features (see Table 1 for a synoptic comparison).

Table 1. The expected Fog-Cloud-5G synergic interplay for the support of future computing-intensive and delay-critical Mobile applications.

FOG		CLOUD		5G	
Distributed deployment and low-access latency	Fog nodes are expected to undergo a pervasive deployment. This reduces the access distance and, then, the resulting propagation delay	Centralized deployment and high-access latency	Cloud data centers typically serve large spatial areas and, then, incur high propagation delays	Real-time support for heterogeneous network technologies	A number of ultra-short-/short-/long-range UWB/WiFi/3G-4G Cellular network technologies are simultaneously supported and dynamically turned ON-OFF
Low-density resource virtualization	Clones of the served mobile devices may be allocated onto medium-/small-sized Fog servers in real time. In order to reduce the resulting bootstrapping delays, light low-density virtualization technologies are employed	High-density resource virtualization	Clones of the served devices are quasi-statically deployed by resorting to high-density virtualization of the computing resources of large servers	Dynamic provisioning, multiplexing, and isolation of the wireless bandwidth	Wireless bandwidth is dynamically provided on-demand to the requiring Mobile devices. Dynamic multiplexing is used, in order to provide inter-device isolation and per-device bandwidth guarantee.
Support for light delay-sensitive Mobile applications	Fog nodes may exploit low-latency WiFi links, in order to enable seamless light data/code offloading from Mobile devices	Support for computing-intense delay-tolerant applications	By exploiting their huge computing resources, Cloud data centers may support computing-intensive delay-tolerant applications offloaded by remote devices	Bandwidth pooling	The simultaneous utilization of multiple transmission technologies allows increasing the total available wireless bandwidth
Energy efficient	Resource-limited Mobile devices may save energy by exploiting nearby Fog nodes as computing clones	Energy hungry	Both network and server infrastructures hosted by Cloud data centers are resource-rich, but energy hungry	Ultra-low access latency	Sub-millisecond access delays are achieved by the synergic exploitation of multiple network technologies

Overall, it is expected that the forthcoming 5G paradigm will be capable of simultaneously exploiting the benefits of centralized CC infrastructures, distributed FC infrastructures, and heterogeneous radio technologies, in order to provide seamless resource-augmentation to resource-limited mobile devices. The pivotal idea is to take full advantage of the cooperative radio resource management and distributed computing capability at nearby Fog nodes, in order to: (i) shorten the communication latencies for the execution of delay-sensitive light tasks; and (ii) resort to centralized remote cloud data centers only for the execution of delay-tolerant workload-intensive tasks.

1.1. The Multi-Tiered Networked Simulated Environment

According to these considerations, Figure 1 reports the general architecture of the virtualized 5G-enabled Mobile-Fog-Cloud technological platform that is considered in this paper.

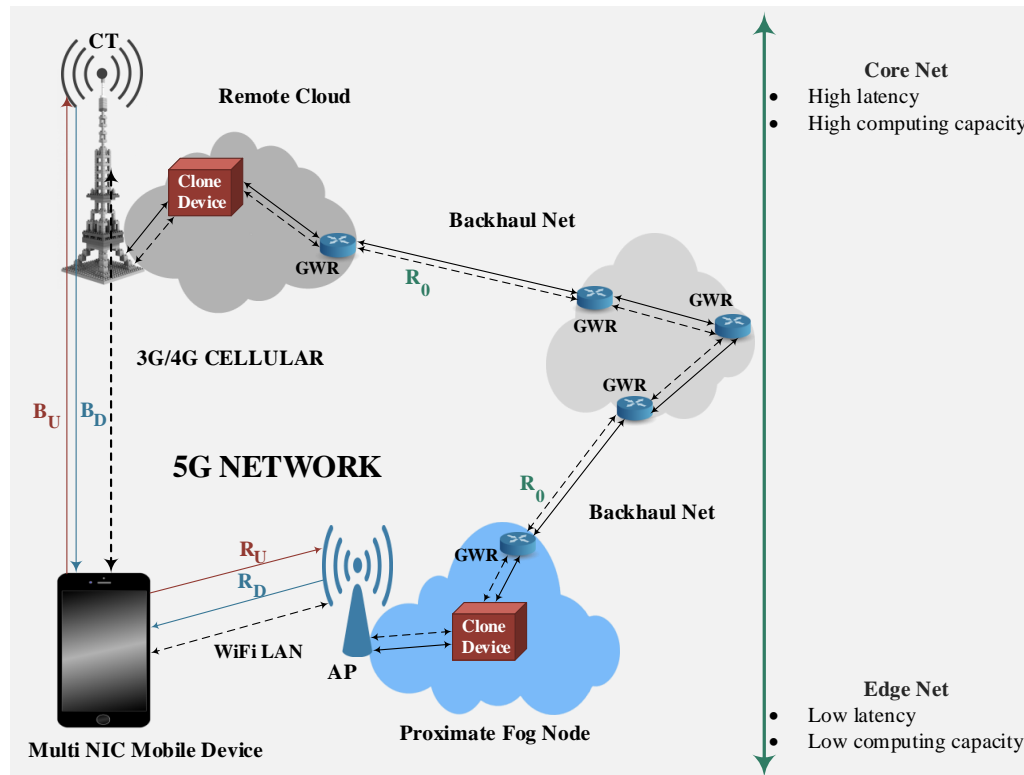


Figure 1. The considered 5G-supported Mobile-Fog-Cloud virtualized infrastructure. WAN: Wide Area Network; LAN: Local Area Network; CT: Cellular Tower; AP: Access Point; GWR: GateWay Router. Single (resp., double)-arrowed paths indicate one-way (resp., two-way) TCP/IP connections.

The platform is composed of a virtualized Mobile device, a proximate virtualized Fog node, and a remote virtualized Cloud node. These nodes inter-communicate through Transport-layer TCP/IP connections that are supported, in turn, by the underlying 5G networking infrastructure. Specifically, in Figure 1, we have that:

1. Mobile-Fog communication is supported by a two-way WiFi-based (possibly, mobile) single-hop TCP/IP connection. R_U (resp., R_D) is the steady-state throughput of the Mobile-to-Fog (resp., Fog-to-Mobile) upstream (resp., downstream) TCP/IP one-way connection. Their peak values depend, in turn, on the adopted WiFi technology;
2. Mobile-Cloud communication is supported by a two-way 3G/4G cellular (possibly, mobile and/or multi-hop) TCP/IP connection. B_U (resp., B_D) is the steady-state throughput of the Mobile-to-Cloud (resp., Cloud-to-Mobile) upstream (resp., downstream) TCP/IP one-way connection;
3. Cloud-Fog communication is supported by a two-way (possibly, wired and multi-hop) backhaul TCP/IP connection. R_0 is the corresponding steady-state throughput.

From a technological perspective, the key role of the 5G paradigm is to allow the computing nodes of Figure 1 to exploit simultaneously multiple heterogeneous radio technologies, in order to perform bandwidth pooling and/or trading network energy versus access latency (see the last column of Table 1). According to this consideration, the reference architecture of Figure 1 assumes that: (i) the mobile device is equipped with at least two Network Interface Cards (NICs), typically, a WiFi NIC and a 3G/4G Cellular NIC; (ii) the Fog node is equipped with a WiFi NIC and an Ethernet NIC; and (iii) the Cloud node is equipped with a Cellular NIC and an Ethernet NIC.

The mobile device must run a given application that is composed of multiple inter-connected tasks (i.e., subroutines or methods). Being energy limited and equipped with limited computing resources, the mobile device may decide to execute each task locally or offload it to the Fog node or the Cloud node. For this purpose, the Fog node (resp., the Cloud node) is virtualized, in order to host a Fog clone (resp., a Cloud clone) of the mobile device. These clones act as virtual processors and execute the offloaded tasks on behalf of the mobile device. At the Application layer, inter-task communications exploit the (aforementioned) Mobile-Fog, Mobile-Cloud, and Cloud-Fog 5G-supported TCP/IP connections.

In the considered reference scenario, the mobile clones at the Fog and Cloud nodes are implemented by resorting to the emerging CoNTainer (CNT)-based technology [7]. Its main peculiar feature is that it uses an Execution Engine, in order to *dynamically* carry out resource allocation (see Figure 2 of the *VirtFogSim* user guide for a description of the virtualized clone organization). As a consequence, the physical resources required by a container may be scaled up/down in *real time* by the corresponding Execution Engine. For this purpose, each server at the Mobile, Fog, and Cloud nodes hosts a number $nc \geq 1$ of containers. All the containers hosted by a same physical server share the pool of computing (e.g., CPU cycles) and networking (e.g., I/O bandwidth) physical resources made available by the physical CPU and NICs that equip the host server. The main task of the Execution Engine managing each container is to allocate dynamically the bandwidth and computing resources made available by the host server. In so doing, the containers at the Fog and Cloud nodes play the role of virtual clones for the associated mobile device and execute the tasks offloaded by the mobile device on behalf of it. For this purpose, each container acts as a Multi-core Virtual Processor, that comprises a number of (typically, homogeneous) Virtual Cores (VCs), whose processing frequencies are dynamically scaled up/down by the execution engine. The final goal of the Execution Engine is to allocate the pending application tasks over the set of available virtual cores according to the actually adopted task allocation strategy.

1.2. A Motivational Example

In order to gain a first intuitive insight about the potential energy-saving capability of the three-tiered networked computing platform of Figure 1, let us consider the toy example of Figure 2. It illustrates a “wise” strategy for allocating the tasks of an application DAG. The considered DAG is composed of eight tasks and nine (directed) edges. Each red-marked label (a, b, c) on the nodes reports the (possibly, profiled) energies needed for the execution of the corresponding task at the Mobile, Fog, and Cloud, respectively. In the considered example, Nodes A and H are the input and output tasks of the overall DAG, respectively. Hence, by design, they are forced to be executed on the Mobile device (hence, the energies needed for their executions on the Fog and Cloud nodes are infinite; see the corresponding red-marked labels of Figure 2). Furthermore, in Figure 2, each blue-marked (resp., black-marked) label (f, g) on the edges indicates the Mobile-to-Cloud and Cloud-to-Mobile (resp., Mobile-to-Fog and Fog-to-Mobile) energies needed for the inter-node transport of the data in the corresponding DAG edge. The computing energy \mathcal{E}_{CMP} and the networking energy \mathcal{E}_{NET} can be easily evaluated by summing the related elements in the red-marked, black marked, and blue-marked labels of Figure 2. These energies will be formally defined in Equations (5) and (6) of Section 3.2. The considered DAG being quite simple, it may be analyzed by hand. The analysis leads to the conclusion that the task allocation strategy that minimizes the total computing-plus-networking consumed energy $\mathcal{E}_{TOT} \triangleq \mathcal{E}_{CMP} + \mathcal{E}_{NET}$ allocates: (i) tasks $\{A, H\}$ to the Mobile device (see the green-colored tasks); (ii) tasks $\{C, G\}$ to the Fog (see the red-colored tasks); and (iii) tasks $\{B, D, E, F\}$ to the Cloud (see the blue-colored tasks). In so doing, the total energy consumed by the optimal task allocation strategy equates: $\mathcal{E}_{TOT} = 32.0$ (Joule), with the computing energy $\mathcal{E}_{CMP} = 20.5$ (Joule) and the networking energy $\mathcal{E}_{NET} = 11.5$ (Joule). By contrast, the more direct (but sub-optimal) strategies that execute all tasks at the Mobile, Fog, and Cloud would require: $\mathcal{E}_{TOT} = 57.0, 35.0,$ and 43.0 (Joule), respectively. This confirms that a “wise” task allocation strategy that exploits *all* the available Mobile-Fog-Cloud

computing nodes may lead to energy saving, even in the presence of *non-negligible* inter-node network energy consumption.

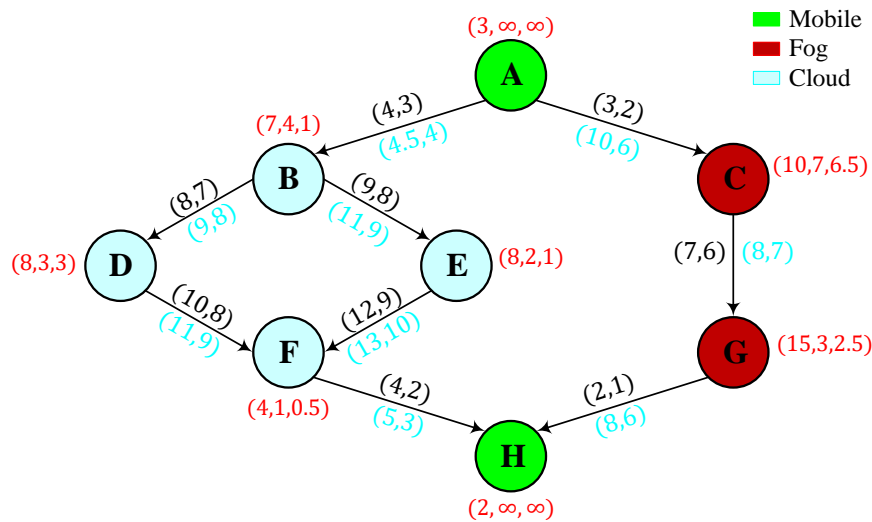


Figure 2. A toy example illustrating the energy-saving capability of multi-tiered Mobile-Fog-Cloud computing platforms that employ both WiFi and Cellular links, in order to connect the computing nodes.

1.3. Motivations, Main Contributions, and Organization of the Paper

Barring toy examples, it may be very challenging to plan online “wise” decisions about which tasks should be offloaded and towards which computing nodes, especially when the executions of the considered applications must be in *real time*, so that hard upper limits are present on the overall computing-plus-communication delays. Roughly speaking, the challenging issue stems from two main factors. First, since the combination of offloading decisions generally increases exponentially with the number of tasks of the considered DAG, exhaustive searches quickly become very time consuming, or even infeasible. Second, it is challenging to calculate analytically in closed-form the execution time of an application described by a DAG with a general (possibly, pseudo-random) topology, especially when the communication latencies must be also accounted for. In fact, the execution time generally depends on many factors like, for example, the parallelism of the Mobile-Cloud-Fog computing nodes, their (possibly, scalable) computing speeds, the number of virtual cores available at the Cloud and Fog nodes, as well as the available (and possibly scalable) inter-node network bandwidths, just to name a few.

Hence, to enable an energy-efficient real-time exploitation of the mobile technological platform of Figure 1, we need a flexible evaluation environment for the dynamic test of different task offloading and resource allocation strategies under programmable (e.g., settable by the user) models for the energy-delay profiles of the virtualized computing and network blocks composing the reference architecture of Figure 1. The hardware implementation of the 5G-supported Mobile-Fog-Cloud test-bed, although valuable, could be too cumbersome, and (moreover) it could not guarantee a repeatable and controllable performance environment.

Motivated by these considerations, in this paper, we present the general architecture and test the main functionalities of the *VirtFogSim* package. It is a new software toolbox that allows:

- simulation on parallel hardware machines;
- dynamic optimization;
- dynamic tracking;
- comparison;
- graphic rendering through an ad-hoc-designed Graphic Use Interface (GUI),

of the energy-delay performances of heuristic and meta-heuristic strategies for *joint* task offloading and *dynamic* resource allocation of application DAGs with *general* topologies over the three-tiered *networked virtualized* computing platform of Figure 1, under hard *real-time* constraints on the overall (i.e., computing-plus-communication) execution times. *VirtFogSim* allows the users to:

- test their desired application DAGs by customizing the simulation environment of Figure 1 through the setting of the 67 input parameters of the simulator package;
- track dynamically the energy-delay DAG performance against abrupt (possibly, unpredictable) changes of the simulated environment of Figure 1, like mobility-induced changes of the available up/down Cellular-WiFi bandwidths;
- optimize the obtained DAG performance against a number of metrics, like total consumed energy, network consumed energy, network bandwidth, computing frequency, and execution delays, just to name a few.

On the basis of an overview of the related work available in the open literature (see the next Section 2), we anticipate that the major peculiar features of the proposed *VirtFogSim* toolbox are the following ones:

- it allows the numerical evaluation of the *delay-constrained* minimization of the overall computing-plus-network energy consumed by the execution of the input DAG. The optimization is performed by task offloading and allocation of the per-core computing frequencies and up/down Cellular-WiFi bandwidths of the platform of Figure 1 in a *joint* and *adaptive* way;
- resource allocation is performed by explicitly accounting for the *container-based virtualized* nature of the reference platform of Figure 1. For this purpose, the *VirtFogSim* toolbox relies on a gradient-based primal-dual iterative procedure that implements a set of ad-hoc designed *adaptive* (e.g., time-varying) step-sizes: The goal is to speed up the convergence to the corresponding steady-states of the *per-core* computing frequencies and *per-connection* bandwidths to be dynamically allocated;
- it allows the user to test the energy-delay performances of six different task allocation strategies, namely the Genetic, Only-Mobile, Only-Fog, Only-Cloud, Only-Task Allocation, and Exhaustive-Search strategies (see Section 4), and to add new user-defined strategies;
- the code of the core engine of the simulator leverages the Parallel Toolbox of MATLAB, in order to exploit automatically the multi-core capability possibly retained by the execution environment of the simulator.

Furthermore, regarding the consideration of the formats of rendering data supported by the proposed *VirtFogSim* toolbox, we point out that it is equipped with a GUI that allows:

- the display of the *dynamic time behavior* of the performed resource allocation under the time-varying simulation environment set by the user and;
- the rendering of the data output by the simulator in tabular, bar-plot, and colored map graph formats (see Section 5).

Finally, a last contribution of the paper concerns the actual test of the numerical and scalability capabilities of the proposed *VirtFogSim* toolbox. In this regard, we point out that:

- the adaptive capability and the energy-delay performance of the (aforementioned) six task allocation strategies have been checked and compared under three benchmark DAGs. They refer to different use cases featuring some computing, multimedia, and scientific applications of practical interest and exhibit different symmetric/asymmetric/pseudo-random topologies of various sizes and in/out node degrees;
- the scaling capability of the simulator has been numerically profiled in terms of both simulation times and volumes of data that are exchanged among the running cores.

The rest of this contribution is organized as follows. After reviewing in Section 2 the related work on the current software tools for the simulation of Fog-Cloud computing platforms, in Section 3, we present the formal models describing the simulated technological platform of Figure 1. Section 4 is devoted to an in-depth description of the task offloading and dynamic resource allocation strategies supported by the current Version 4.0 of the *VirtFogSim* toolbox, as well as their implementation on multi-core parallel hardware execution platforms. Afterward, Section 5 describes the associated graphic formats for the display of the rendered data. Section 6 focuses on: (i) checking the numerical capability of *VirtFogSim*; and (ii) testing the scalability of the simulator in terms of both simulation time and memory consumption on multi-core parallel hardware execution platforms. Some hints for possible future developments and the availability of the *VirtFogSim* package are reported in the conclusive Sections 7 and 8. The reader may refer to Appendix A for the presentation of the dual-mode user interfaces of the *VirtFogSim* toolbox and to the final Appendix B for the full list of the (settable) input parameters of the *VirtFogSim* simulator and the corresponding meaning, roles, and measuring units.

2. Related Work

The development of simulation tools for Fog computing is still in its infancy. Roughly speaking, the major part of the current contributions constitute the follow up of some (quite recent) toolkits designed for the simulation of the (somewhat more consolidated) environment of Cloud computing and, then, account for a limited number of specific aspects of the Fog computing paradigm. Under this perspective, an overview of the current open literature leads to the conclusion that the development of simulation tools for Cloud-Fog computing platforms has proceeded along three main research lines.

The first (somewhat more traditional) research line leads to the development of toolboxes for the software simulation of networked large-scale Cloud data centers [8–10].

In this regard, CloudSim [8] is a broad simulation toolkit that allows modeling and simulation of applications on a remote Cloud platform according to the Infrastructure-as-a-Service provisioning model. It allows the user to setup a customized modeling of the major building blocks of conventional Cloud infrastructures, like Virtual Machines (VMs), resource provisioning policies, resource consolidation policies, and VM migration policies. As a consequence, at the present time, CloudSim seems to be the most popular simulator for Cloud computing scenarios. However, although the Fog and Cloud computing paradigms share some virtualization features that, in principle, enable the re-use of some Cloud solutions even for Fog computing (see Table 1), the Fog environment mainly targets the support of delay-sensitive (possibly mobile) applications. From this point of view, the CloudSim toolkit presents three main deficiencies. First, it does not allow the customized setup of network-related parameters, like the per-link wireless access bandwidths and the round-trip-times of Transport-layer TCP/IP connections. Second, it relies on VM-based virtualization and does not allow the modeling of emerging CNT-based virtualization. Third, the implemented resource allocation policies are of the static-type, and no support for dynamic resource tracking is provided.

GreenCloud [9] is a follow up of the NS2 network simulator. Its main focus is on the modeling and simulation of the energy profiles of some main computing and network components of the Cloud ecosystem. Being an extension of the NS2 toolkit, GreenCloud allows the customized setting and simulation of the full TCP/IP protocol stacks equipping the switches of the intra-cloud network. However, it does not account for the Mobile-to-Cloud wireless access links and does not allow the characterization of the submitted application as DAGs.

The main goal of iCanCloud [10] is the simulation of Cloud-supported large ecosystems with thousands of served devices through a suitable extension of the native OMNeT++ platform. Hence, scalability being the main concern of this toolkit, it does not support dynamic resource tracking and per-device performance optimization.

A second more recent research line attempts to address the (aforementioned) deficiencies of Cloud-oriented toolkits by explicitly accounting for the specific features of real Fog platforms for the support of IoT-based sensor applications [11–13].

Under this perspective, SimIOT [11] equips the (previously-developed) SimIC [14] simulator by including a bottom IoT layer, in order to allow the user to model the request of Cloud resources of a settable number of sensor/actuator devices. This is obtained by including a communication broker module atop the native SimIC platform, in order to gather the sensed data and redirect them to the SimIC Cloud agents. As a consequence, SimIOT explicitly lacks support of an intermediate Fog layer.

IOTSim [12] is implemented as an extension of (aforementioned) CloudSim. It is targeted to the simulation of Fog computing environments in which sensor-acquired big data streams have to be quickly processed. For this purpose, the IOTSim platform equips the native CloudSim architecture with a storage layer and, then, provides the software primitives for the simulation and storage-induced delays. However, the core processing engine supported by the current IOTSim release is MapReduce-compliant, so that it implements batch processing models for the support of delay-tolerant big data applications.

iFogSim [13] is another toolbox whose implementation is an extension of CloudSim. It aims to simulate IoT-based Fog platforms by providing: (i) suitable software Java-based primitives for modeling the energy-delay performances of sensors, actuators, Fog, and Cloud nodes; (ii) two service models, e.g., the Sense-Process-Actuate and the Stream processing services; and (iii) two heuristic task allocation policies, namely the All-Cloud and Edge-ward task placement policies. However, in the current release of iFogSim, we have that: (i) all the supported devices' profiling facilities and task allocation policies are inherently static; (ii) the network delays are assumed fixed; and (iii) support for the dynamic allocation and/or tracking of network bandwidths and computing frequencies is not provided.

Very recently, a third research line focused on the development of the software toolkit EdgeCloudSim [15], for the simulation and performance evaluation of general multi-tiered Cloud-Fog computing platforms. The main feature of EdgeCloudSim is the provisioning of a software environment for the setting and dynamic simulation of the profiles of WLAN/WAN networks, wireless network traffic, device mobility, Fog nodes, and Cloud nodes. For this purpose, EdgeCloudSim: (i) provides a customizable edge orchestration module for the performance evaluation of different task placement policies under different time scales; (ii) implements a customizable load generator that allows simulating the workload of the served mobile devices as streams of independent tasks with Poisson-distributed arrival times; and (iii) relies on the CloudSim support for the creation/management/migration/shut-down of VMs. Hence, like the proposed VirtFogSim, EdgeCloudSim aims at simulating the performance of multi-tiered networked Fog-Cloud platforms for the support of time-critical tasks. However, *unlike* VirtFogSim, EdgeCloudSim: (i) does not support the utilization of application DAGs and, then, does not allow the characterization of the inter-task dependency; (ii) being VM-oriented, does not support the characterization of container-based virtualization; (iii) does not provide software primitives for the tracking of the allocated network/computing resources; and (iv) being mainly focused on the delay-performance of the simulated platform, does not provide support for the modeling of the power/energy profiles of the Mobile, Fog, and Cloud nodes.

3. VirtFogSim: A View of the Simulated Formal Models

In this section, the basic definitions and formal assumptions about the constrained optimization problem tackled by the *VirtFogSim* engine are briefly explained. The reference framework is that already reported in Figure 1.

3.1. Profiling the Simulated Workflows

By definition, a task allocation vector:

$$\vec{x} \stackrel{\text{def}}{=} [x(1) \equiv 1, x(2), \dots, x(V-1), x(V) \equiv 1], \quad (1)$$

is a dimensionless $(1 \times V)$ ternary vector, whose components are defined as follows:

$$x(i) \stackrel{\text{def}}{=} \begin{cases} 0, & \text{if the } i^{\text{th}} \text{ task is executed at the Fog clone,} \\ 1, & \text{if the } i^{\text{th}} \text{ task is executed at the Mobile Device,} \\ 2, & \text{if the } i^{\text{th}} \text{ task is executed at the Cloud clone,} \end{cases} \quad i = 1, \dots, V.$$

By design, the first and last tasks are always executed at the Mobile device, and therefore $x(1) \equiv x(V) \equiv 1$ (see Equation (1)). Hence, the size of the set of the admissible task allocation vectors is 3^{V-2} .

By definition, a resource allocation vector:

$$\vec{RS} \stackrel{\text{def}}{=} [f_M, f_F, f_C, R_U, R_D, B_U, B_D] \quad (\text{bit/s}), \tag{2}$$

is a (1×7) non-negative row vector whose components are measured in bit/s. It reports the values assumed by (see Figure 1) in the order:

1. the processing frequency f_M at the Mobile device;
2. the processing frequency f_F at the Fog clone;
3. the processing frequency f_C at the Cloud clone;
4. the throughput R_U of the Mobile-to-Fog TCP/IP connection;
5. the throughput R_D of the Fog-to-Mobile TCP/IP connection;
6. the throughput B_U of the Mobile-to-Cloud TCP/IP connection;
7. the throughput B_D of the Cloud-to-Mobile TCP/IP connection.

3.2. The Considered Throughput-Constrained Joint Task and Dynamic Resource Allocation Problem

In this section, we give a glimpse of the constrained optimization problem whose solution is numerically evaluated by *VirtFogSim*. For this purpose, let $\mathcal{E}_M, \mathcal{E}_F,$ and \mathcal{E}_C (Joule) be the computing energies consumed by the Mobile device, Fog clone, and Cloud clone of Figure 1 in order to process the assigned tasks of the considered application DAG. Furthermore, let $\mathcal{E}_{WiFi}, \mathcal{E}_{CELL},$ and \mathcal{E}_{WD} (Joule) be the energies consumed by the Mobile-Fog, Mobile-Cloud, and Cloud-Fog two-way TCP/IP connections, in order to transport the inter-task data among the Mobile, Cloud, and Fog computing nodes. Finally, let:

$$\mathcal{E}_{TOT} \stackrel{\text{def}}{=} \vartheta_M \times \mathcal{E}_M + \vartheta_F \times \mathcal{E}_F + \vartheta_C \times \mathcal{E}_C + \mathcal{E}_{WiFi} + \mathcal{E}_{CELL} + \mathcal{E}_{WD} \quad (\text{Joule}), \tag{3}$$

be the resulting computing-plus-communication total consumed energy, with $\vartheta_M, \vartheta_F,$ and ϑ_C being binary $\{0, 1\}$ -valued constants.

Hence, the considered Joint Optimization Problem (JOP) is defined as follows:

$$\underset{\vec{RS}, \vec{x}}{\text{min}} \mathcal{E}_{TOT}, \tag{4a}$$

s.t.:

$$(1/T_{DAG}) \geq TH_0^{MIN}, \tag{4b}$$

$$0 \leq f_M \leq f_M^{MAX}, \tag{4c}$$

$$0 \leq f_F \leq f_F^{MAX}, \tag{4d}$$

$$0 \leq f_C \leq f_C^{MAX}, \tag{4e}$$

$$0 \leq R_U \leq R_U^{MAX}, \tag{4f}$$

$$0 \leq R_D \leq R_D^{MAX}, \tag{4g}$$

$$0 \leq B_U \leq B_U^{MAX}, \tag{4h}$$

$$0 \leq B_D \leq B_D^{MAX}, \tag{4i}$$

$$x(1) = x(V) = 1, \tag{4j}$$

$$x(i) \in \{0, 1, 2\}, \quad i = 2, \dots, (V - 1). \tag{4k}$$

In the above equations, we have that:

1. \vec{RS} is the resource allocation vector defined in Equation (2);
2. \vec{x} is the task allocation vector defined in Equation (1);
3. T_{DAG} (s) is the overall time needed to execute the assigned application DAG, in short, the DAG execution time. It generally depends on the optimization variables \vec{RS} and \vec{x} ;
4. TH_0^{MIN} (app/s) is the required minimum application throughput, i.e., the minimum number of application DAGs to be performed in the time interval of one second. It is a non-negative real number;
5. Equations (4c)–(4g) account for the maximum available resources;
6. Equations (4j)–(4k) account for the ternary nature of the considered task allocation vectors.

The considered JOP is solved in *VirtFogSim* by the *RAP_p* function described in Section 4.2. Before proceeding, four main remarks about the formulated JOP are in order.

First, it aims to *jointly* optimize the allocation of the tasks over the Mobile, Fog, and Cloud computing nodes of Figure 1 and the corresponding computing-plus-communication resources. Formally speaking, it is a mixed-integer non-convex optimization problem, that resists closed-form solution. In the sequel, we will denote as: \vec{RS}^* and \vec{x}^* the solution of the constrained JOP in Equations (4a)–(4k).

Second, for positive TH_0^{MIN} , the constraint in Equation (4b) enforces a *hard* (e.g., deterministic) QoS constraint on the minimum desired application throughput. As a consequence, too many larger values of TH_0^{MIN} may give rise to infeasible JOP.

Third, the actual values of the binary constants: ϑ_M , ϑ_F , and ϑ_C depend on the application service model adopted by the Service Provider. Specifically, ϑ_M and/or ϑ_F and/or ϑ_C are unit valued (resp., vanish) when the energies consumed by the Mobile, Fog, and Cloud computing nodes are for free (resp., are taxed).

Fourth, the summation of the first three terms in the objective function in (3) represents the consumed computing energy, i.e.,

$$\mathcal{E}_{CMP} \stackrel{\text{def}}{=} \vartheta_M \times \mathcal{E}_M + \vartheta_F \times \mathcal{E}_F + \vartheta_C \times \mathcal{E}_C \quad (\text{Joule}), \tag{5}$$

while the summation of the remaining three terms is the corresponding wasted network energy \mathcal{E}_{NET} , formally defined as follows:

$$\mathcal{E}_{NET} \stackrel{\text{def}}{=} \mathcal{E}_{WiFi} + \mathcal{E}_{CELL} + \mathcal{E}_{WD} \quad (\text{Joule}). \tag{6}$$

3.3. Simulated Computing and Networking Energy Profiles

In this section, we (briefly) present the analytical expressions of the computing and network energies involving the objective function in (4a), together with the corresponding analytical expression of the (constrained) T_{DAG} in (4b). The goal is to allow the *VirtFogSim* user to acquire basic insight about the roles played and the possible impacts of the input parameters in Table A2 of the simulator on the resulting solution (\vec{RS}^*, \vec{x}^*) of the underlying JOP in Equations (4a)–(4k). In this regard, four introductory remarks are in order.

First, since the Mobile device of Figure 1 may both upload to and download from the connected Fog and Cloud clones, the previously-defined network energies: \mathcal{E}_{WiFi} and \mathcal{E}_{CELL} split in the summation

of the corresponding up and down network energies plus the underlying idle energies, so that we can write:

$$\mathcal{E}_{WiFi} = \mathcal{E}_{WiFi}^{(IDLE)} + \mathcal{E}_{WiFi}^U + \mathcal{E}_{WiFi}^D \quad (\text{Joule}), \tag{7}$$

and:

$$\mathcal{E}_{CELL} = \mathcal{E}_{CELL}^{(IDLE)} + \mathcal{E}_{CELL}^U + \mathcal{E}_{CELL}^D \quad (\text{Joule}). \tag{8}$$

Second, in the considered framework of Figure 1, the Fog-Cloud (typically wired) backhaul is assumed sustained by a two-way TCP/IP transport connection that operates in the steady-state. Hence, according to the seminal analysis reported for example in [16], the corresponding transport throughput R_0 may be directly evaluated as in:

$$R_0 = \frac{1.22 \times MSS}{RTT_{WD} \times \sqrt{Prob_{LOSS}}} \quad (\text{bit/s}), \tag{9}$$

where (see Figure 1): (i) MSS (bit) is the maximum size of a TCP segment; (ii) RTT_{WD} (s) is the steady-state Round-Trip-Time of the two-way Fog-Cloud TCP/IP connection; and (iii) $Prob_{LOSS}$ is the associated steady-state segment loss probability.

Third, according to a number of both analytical and experimental models (see, for example, [17,18] and the references therein), the computing and network energy present in the objective function in Equations (4a)–(4k) is the summation of static energy and dynamic energy. Specifically, we note that:

1. the static energy accounts for the energy wasted by the device in the idle state (e.g., the device is turned ON, but it is not running). As a consequence, since the power consumed by the device in the idle state \mathcal{P}_{IDLE} (Watt) does not depend on the optimization variables, the static energy depends on the optimization variables $\vec{R}\vec{S}$ and \vec{x} only through the corresponding DAG execution time T_{DAG} (s);
2. the dynamic energy accounts for the energy wasted by the device when it is in the running state. As a consequence, since the resulting dynamic power depends on the operating computing frequency or communication bit rate, the dynamic energy of each device depends on the optimization variables $\vec{R}\vec{S}$ and \vec{x} through both the DAG execution time T_{DAG} (s) and the corresponding dynamic power \mathcal{P}_{DYN} (Watt).

Fourth, a number of both analytical and experimental studies recently appeared in the open literature [18–23], supporting the conclusion that the dynamic power $\mathcal{P}_{DYN}(y)$ (Watt) consumed by a computing (resp., network) device operating at the computing frequency (resp., bit-rate) y may be accurately profiled by a power-like relationship, e.g., $\mathcal{P}_{DYN} = K(y)^\gamma$ (Watt), where both the coefficient K and exponent γ depend on the specifically considered device and operating conditions. Hence, by definition, the corresponding dynamic energy $\mathcal{P}_{DYN}(y)$ (Joule) equates to:

$$\mathcal{E}_{DYN}(y) \stackrel{\text{def}}{=} \mathcal{P}_{DYN}(y) / y = K(y)^{\gamma-1} \quad (\text{Joule}). \tag{10}$$

Therefore, after indicating by:

1. $\delta(x)$ Kronecker’s delta (e.g., $\delta(x) = 1$ for $x = 1$ and $\delta(x) = 0$, otherwise); and by:
2. $u_{-1}(x)$ the unit-step function (e.g., $u_{-1}(x) = 1$ for $x > 0$ and $u_{-1}(x) = 0$, otherwise),

the above considerations lead to the following sum-power-like analytical models for the profiles of the (previously-introduced) computing energies:

$$\begin{aligned} \mathcal{E}_M = & \left(\frac{\mathcal{P}_{CPU-M}^{(IDLE)}}{nc_M} \right) \times T_{DAG} \times u_{-1} \left(\sum_{i=1}^V \delta(x(i) - 1) \right) \\ & + \left(\sum_{i=1}^V s(i) \delta(x(i) - 1) \right) K_M (1 - r_M) (f_M)^{\gamma_M-1}, \end{aligned} \tag{11a}$$

$$\begin{aligned} \mathcal{E}_F &= \left(\frac{\mathcal{P}_{CPU-C}^{(IDLE)}}{nc_C} \right) \times T_{DAG} \times u_{-1} \left(\sum_{i=1}^V \delta(x(i) - 2) \right) \\ &+ \left(\sum_{i=1}^V s(i) \delta(x(i) - 2) \right) K_C (1 - r_C) (f_C)^{\gamma_C - 1}, \end{aligned} \tag{11b}$$

$$\begin{aligned} \mathcal{E}_C &= \left(\frac{\mathcal{P}_{CPU-C}^{(IDLE)}}{nc_C} \right) \times T_{DAG} \times u_{-1} \left(\sum_{i=1}^V \delta(x(i) - 2) \right) \\ &+ \left(\sum_{i=1}^V s(i) \delta(x(i) - 2) \right) K_C (1 - r_C) (f_C)^{\gamma_C - 1}, \end{aligned} \tag{11c}$$

where all the device-depending parameters: $\mathcal{P}_{CPU}^{(IDLE)}$, nc , K , r , and γ involved in the above energy-profiling relationships are detailed in Table A2, together with their meaning/role and proper measuring units.

In order to introduce the energy profiles of the five up/down WiFi, up/down Cellular, and two-way backhaul connections referenced by Equations (7)–(9), let: $\mathcal{V}_{M \rightarrow F}$, $\mathcal{V}_{F \rightarrow M}$, $\mathcal{V}_{M \rightarrow C}$, $\mathcal{V}_{C \rightarrow M}$, and $\mathcal{V}_{F \leftrightarrow C}$ indicate the volumes (in bit) of data transported (see Figure 1): (i) from Mobile to Fog; (ii) from Fog to Mobile; (iii) from Mobile to Cloud; (iv) from Cloud to Mobile; and (v) exchanged between Fog and Cloud, respectively. Hence, directly from the basic definitions of the adjacency (binary-valued) $A \triangleq [a_{ij}]$ and (real-valued) $D_a \triangleq [d_{ij}]$ connection matrices, featuring the assigned DAG, as well as the task allocation vector \vec{x} in Equation (1), these volumes may be formally expressed as:

$$\mathcal{V}_{M \rightarrow F} = (1 + \overline{NF}_{WiFi}) \left(\sum_{i,j=1}^V a_{ij} \delta(x(i) - 1) \delta(x(j)) d_{ij} \right), \tag{12a}$$

$$\mathcal{V}_{F \rightarrow M} = (1 + \overline{NF}_{WiFi}) \left(\sum_{i,j=1}^V a_{ij} \delta(x(i)) \delta(x(j) - 1) d_{ij} \right), \tag{12b}$$

$$\mathcal{V}_{M \rightarrow C} = (1 + \overline{NF}_{CELL}) \left(\sum_{i,j=1}^V a_{ij} \delta(x(i) - 1) \delta(x(j) - 2) d_{ij} \right), \tag{12c}$$

$$\mathcal{V}_{C \rightarrow M} = (1 + \overline{NF}_{CELL}) \left(\sum_{i,j=1}^V a_{ij} \delta(x(i) - 2) \delta(x(j) - 1) d_{ij} \right), \tag{12d}$$

$$\mathcal{V}_{F \leftrightarrow C} = (1 + \overline{NF}_{WD}) \left(\sum_{i,j=1}^V a_{ij} (\delta(x(i) - 2) \delta(x(j)) + \delta(x(i)) \delta(x(j) - 2)) d_{ij} \right), \tag{12e}$$

where \overline{NF} is the average number of failures of the considered connection [22,23] (see also Table A2) and V is the number of tasks of the considered DAG. Hence, the idle, up, and down energies of the WiFi and Cellular network connections in Equations (7) and (8) may be profiled as follows:

$$\mathcal{E}_{WiFi}^{(IDLE)} = (\vartheta_M + \vartheta_F) \times \mathcal{P}_{WiFi-M}^{(IDLE)} \times T_{DAG} \times u_{-1} \left(\sum_{i=1}^V \delta(x(i)) \right), \tag{13a}$$

$$\mathcal{E}_{WiFi}^U = \mathcal{V}_{M \rightarrow F} \times \left(\vartheta_M \Omega_{WiFi}^{(TX)} (R_U)^{\xi_{WiFi}^{(TX)} - 1} + \vartheta_F \Omega_{WiFi}^{(RX)} (R_U)^{\xi_{WiFi}^{(RX)} - 1} \right), \tag{13b}$$

$$\mathcal{E}_{WiFi}^D = \mathcal{V}_{F \rightarrow M} \times \left(\vartheta_M \Omega_{WiFi}^{(RX)} (R_D)^{\xi_{WiFi}^{(RX)} - 1} + \vartheta_F \Omega_{WiFi}^{(TX)} (R_D)^{\xi_{WiFi}^{(TX)} - 1} \right), \tag{13c}$$

and:

$$\mathcal{E}_{CELL}^{(IDLE)} = (\vartheta_M + \vartheta_C) \times \mathcal{P}_{CELL-M}^{(IDLE)} \times T_{DAG} \times u_{-1} \left(\sum_{i=1}^V \delta(x(i) - 2) \right), \tag{14a}$$

$$\mathcal{E}_{CELL}^U = \mathcal{V}_{M \rightarrow C} \times \left(\vartheta_M \Omega_{CELL}^{(TX)} (B_U)^{\xi_{CELL}^{(TX)} - 1} + \vartheta_C \Omega_{CELL}^{(RX)} (B_U)^{\xi_{CELL}^{(RX)} - 1} \right), \tag{14b}$$

$$\mathcal{E}_{CELL}^D = \mathcal{V}_{C \rightarrow M} \times \left(\vartheta_M \Omega_{CELL}^{(RX)} (B_D)^{\xi_{CELL}^{(RX)} - 1} + \vartheta_C \Omega_{CELL}^{(TX)} (B_D)^{\xi_{CELL}^{(TX)} - 1} \right). \tag{14c}$$

Finally, the energy consumed by the backhaul connection reads as:

$$\mathcal{E}_{WD} = \left(\vartheta_C \mathcal{P}_{ETH-C}^{(IDLE)} + \vartheta_F \mathcal{P}_{ETH-F}^{(IDLE)} \right) \times T_{DAG} \times u_{-1} \left(\sum_{i,j=1}^V a_{ij} (\delta(x(i)) \delta(x(j) - 2) + \delta(x(i) - 2) \delta(x(j))) \right) + \mathcal{V}_{F \leftrightarrow C} \times \left(\frac{\mathcal{P}_{WD}}{R_0} \right), \quad (15)$$

where the corresponding wasted dynamic power \mathcal{P}_{WD} (Watt) is given by the product:

$$\mathcal{P}_{WD} = no_{HOP} \times \mathcal{P}_{HOP}, \quad (16)$$

of the number of hops no_{HOP} of the connection by the per-hop consumed power \mathcal{P}_{HOP} (Watt) (see Table A2).

3.4. Simulated Profiles of the per-DAG and per-Task Execution Times

The analytical expression assumed by T_{DAG} in Equation (4b) depends on the vectors \vec{RS} , \vec{x} , as well as the adopted inter-node Task Scheduling and intra-node Task Service disciplines. The current implementation of *VirtFogSim* assumes that the adopted Task Scheduling discipline over the computing nodes and the adopted Task Service discipline at each computing node are both of the *sequential* type. They are, indeed, the Task Scheduling and Task Service disciplines currently adopted by a number of middleware software tools for the support of mobile computing, such as MAUI, Volare, Cuckoo, and CloneCloud (see, for example, [24,25] and the references therein for an overview of state-of-the-art middleware solutions for the support of computing-intensive smartphone-based applications).

As direct consequence, the following sum expression holds for the analytical profile of T_{DAG} :

$$T_{DAG} = \sum_{i=1}^V \left(\delta(x(i) - 1) T_M(i) + \delta(x(i)) T_F(i) + \delta(x(i) - 2) T_C(i) \right), \quad (17)$$

where $T_M(i)$, $T_F(i)$, and $T_C(i)$ are the total execution times of the i^{th} task of the considered DAG when it is executed at the Mobile, Fog, and Cloud node, respectively.

In this regard, two main remarks are in order. First, each total execution time $T_N(i)$, $N \in \{M, F, C\}$, in Equation (17), is, by design, the summation of two terms. The first term is the computing time (e.g., the service time): $T_N^{SER}(i)$, $N \in \{M, F, C\}$, wasted for the processing of the i^{th} task at the computing node N , while the second term is the communication delay: $T_N^{NET}(i)$, $N \in \{M, F, C\}$, induced by the transport from the other two computing nodes of the input data needed for the processing of the i^{th} task at the computing node N . Second, when a set of tasks is processed by the same computing node, the resulting overall communication delay is, by design, zero [23,26,27].

Therefore, according to these two remarks, the following analytical expressions hold for the profiles of the per-node execution times in Equation (17):

$$T_M(i) = T_M^{SER}(i) + T_M^{NET}(i) = \left(\frac{\sum_{j=1}^V s(j) \delta(x(j) - 1)}{n_M f_M} \right) + \left[(1 + \overline{NF}_{WiFi}) \left(\frac{\sum_{j=1}^V a_{ji} d_{ji} \delta(x(j))}{R_D} \right) + (1 + \overline{NF}_{CELL}) \left(\frac{\sum_{j=1}^V a_{ji} d_{ji} \delta(x(j) - 2)}{B_D} \right) \right], \tag{18a}$$

$$T_F(i) = T_F^{SER}(i) + T_F^{NET}(i) = \left(\frac{\sum_{j=1}^V s(j) \delta(x(j))}{n_F f_F} \right) + \left[(1 + \overline{NF}_{WiFi}) \left(\frac{\sum_{j=1}^V a_{ji} d_{ji} \delta(x(j) - 1)}{R_U} \right) + \left(\frac{\sum_{j=1}^V a_{ji} d_{ji} \delta(x(j) - 2)}{R_0} \right) \right], \tag{18b}$$

$$T_C(i) = T_C^{SER}(i) + T_C^{NET}(i) = \left(\frac{\sum_{j=1}^V s(j) \delta(x(j) - 2)}{n_C f_C} \right) + \left[(1 + \overline{NF}_{CELL}) \left(\frac{\sum_{j=1}^V a_{ji} d_{ji} \delta(x(j) - 1)}{B_U} \right) + \left(\frac{\sum_{j=1}^V a_{ji} d_{ji} \delta(x(j))}{R_0} \right) \right], \tag{18c}$$

where $i = 1, \dots, V$ and $n_M, n_F,$ and n_C are the number of cores equipping the Mobile, Fog, and Cloud nodes, respectively (see Table A2).

3.5. Simulated Adaptive Resource Allocation Framework

In this subsection, we briefly present the general analytical framework implemented by the *RAP_p* and *FogTracker* functions of Sections 4.2 and 4.4, in order to perform the adaptive updating of the computing frequencies: $f_M, f_F, f_C,$ the wireless network bandwidths: $R_U, R_D, B_U, B_D,$ and the Lagrange multiplier λ associated with the throughput constraint in Equation (4b). The final goal is to allow the user to acquire insights about the role played by the input step-size a_{MAX} and related input step-size vector \vec{a}_{MAX-FT} on the adaptive capability of the resource allocation engine implemented by *VirtFogSim*.

For this purpose, after indicating by \mathcal{L} the Lagrangian function of the *JOP* of Equation (4), let: $y \in \{f_M, f_F, f_C, R_U, R_D, B_U, B_D, \lambda\}$ indicate a (scalar and non-negative) variable to be optimized, and let y_{MAX} be its allowed maximum value. Furthermore, after denoting by t an integer-valued iteration index, let $\nabla_y \mathcal{L}(t)$ be the derivative of the Lagrangian function \mathcal{L} with respect to the considered optimization variable y at iteration t . Hence, according to the so-called primal-dual iteration-based approach recently customized in [28] for broadband networked application scenarios, the adaptive updating of $y(t + 1)$ reads as follows:

$$y(t + 1) = \max\{0, \min\{y_{MAX}, y(t) - g_y(t) \nabla_y \mathcal{L}(t)\}\}, \tag{19}$$

where the outer $\max\{\cdot\}$ (resp., the inner $\min\{\cdot\}$) accounts for the non-negative (resp., maximum) value of y .

The peculiar feature of the updating relationship in (19) is that it resorts to a y -depending and time-varying gain function: $g_y(t),$ in order to guarantee *quick adaptation* of the optimization variable y in response to abrupt changes of the wireless environment of Figure 1. On the basis of the general results of [28] about the convergence to the steady-state of the iterations in (19), we planned to implement in the current version of *VirtFogSim* the following time-varying gain formula:

$$g_y(t + 1) = \max\{a_{MAX}, \min\{a_{MAX} \times y_{MAX}, \frac{1}{2} (y(t))^2\}\}. \tag{20}$$

An examination of the above formula unveils the role played by the (positive) step-size $a_{MAX},$ as well as its potential impact on the adaptive capability of the overall simulator. Specifically, on the basis

of (20), we expect that large (resp., small) values of a_{MAX} lead to quick (resp., slow) response to abrupt environmental changes, together with large (resp., small) possible oscillations in the steady-state. In this regard, we anticipate that the final goal of the *FogTracker* function of Section 4.4 is to guide the user towards the right trade-off among these two a_{MAX} -depending contrasting behaviors.

4. VirtFogSim: Supported Task Allocation Strategies and Their Parallel Execution

The engine of the *VirtFogSim* simulator is built of a main core of eight software routines that implement a number of strategies (e.g., optimization policies), in order to check the feasibility and, then, solve numerically the constrained optimization problem in (4). MATLAB is the native environment under which the optimization routines are developed and run.

In this regard, it must be remarked that the *VirtFogSim* package is capable of *automatically* turning ON and exploiting the multi-core capability retained by the supporting hardware platform, in order to run parallel programming and, then, speed up the execution of the implemented task allocation policies. This is done in a *fully-transparent* way, e.g., without any direct user involvement. For this purpose, some instructions for parallel programming on multi-core hardware platforms specifically supported by the *Parallel Toolbox* of MATLAB are utilized by the *VirtFogSim* package. However, *VirtFogSim* is capable of *automatically* switching to the *sequential* execution mode when the underlying hardware platform is single-core or the MATLAB *Parallel Toolbox* is not installed.

4.1. General Architecture of the Developed Simulation Platform

The main functionalities of *VirtFogSim* are implemented by the functions listed in Table 2. In the sequel of the paper, these functions are briefly described. A deep explanation about the usage of such functions can be found in the *VirtFogSim* User Guide, which can be downloaded along with the software package (see Section 8). The main functions in Table 2 use some auxiliary functionalities implemented in the set of routines listed in Table 3.

Table 2. Main functions implemented by *VirtFogSim*.

Function	Description
$[\tilde{y}, \tilde{\mathcal{E}}_{TOT}, \tilde{\mathcal{E}}_{NET}] = \text{RAP_p}(\vec{x}, \vec{z}, \vec{s}, Da, S_{RAP})$	It implements the Resource Allocation Problem described in Equation (4).
$[\mathcal{E}_{aux}, \mathcal{E}_{aux}^{NET}, \lambda_{aux}] = \text{FogTracker}(x_1, x_2, x_3)$	It tests the convergence rate to the steady-state and the steady-state stability of the primal-dual iterations performed by the <i>RAP_p</i> function when abrupt changes happen.
$[\vec{x}_{best}, \vec{RS}_{best}, \mathcal{E}_{best}, \mathcal{E}_{best}^{NET}, RB_{best}] = \text{GeneticTA_par}$	It runs the parallel Genetic Task Allocation strategy.
$[\vec{RS}_{OM}, \mathcal{E}_{OM}, \mathcal{E}_{OM}^{NET}] = \text{OM_S}$	It runs the Only Mobile strategy.
$[\vec{RS}_{OF}, \mathcal{E}_{OF}, \mathcal{E}_{OF}^{NET}] = \text{OF_S}$	It runs the Only Fog strategy.
$[\vec{RS}_{OC}, \mathcal{E}_{OC}, \mathcal{E}_{OC}^{NET}] = \text{OC_S}$	It runs the Only Cloud strategy.
$[\vec{x}_{OTAS}, \mathcal{E}_{OTAS}, \mathcal{E}_{OTAS}^{NET}] = \text{O_TAS_par}$	It runs the Only Task Allocation strategy.
$[\vec{x}_{ESS}, \vec{RS}_{ESS}, \mathcal{E}_{ESS}, \mathcal{E}_{ESS}^{NET}, RB_{ESS}] = \text{ES_S}$	It runs the Exhaustive Search strategy.

Regarding the description of the general software architecture of the simulator, *VirtFogSim* acts as the main program that:

1. allows the user to setup 67 input parameters that characterize the scenario to be simulated by the user (see Figure 1);
2. calls the *GeneticTA_par* function for parallel execution and returns the corresponding:
 - (a) $\vec{x}_{best} \in \{0, 1, 2\}^V$, i.e., ternary V -tuple best allocation vector;

- (b) $\vec{RS}_{best} \stackrel{\text{def}}{=} [f_M^*, f_F^*, f_C^*, R_U^*, R_D^*, B_U^*, B_D^*]$ (bit/s), i.e. the seven-tuple vector of the optimal resource allocation of the Mobile computing frequency, Fog computing frequency, Cloud computing frequency, Mobile-to-Fog transport throughput, Fog-to-Mobile transport throughput, Mobile-to-Cloud transport throughput, and Cloud-to-Mobile transport throughput (see Figure 1);
- (c) \mathcal{E}_{best} (Joule), i.e., the total computing-plus-network energy consumed by the infrastructure of Figure 1 under the returned \vec{x}_{best} task allocation vector. For this purpose, the *GeneticTA* function calls, in turn, the auxiliary functions *RAP*, *Crossover*, and *Mutation*;
3. optionally, calls the *OM_S* function and returns:
- (a) $\vec{RS}_{OM} \stackrel{\text{def}}{=} [f_M, f_F, f_C, R_U, R_D, B_U, B_D]$ (bit/s), i.e., the seven-tuple vector of the optimal resource allocation under the $(1 \times V)$ *All-Mobile* task allocation vector: $\vec{x}_{OM} \stackrel{\text{def}}{=} [1, 1, \dots, 1, 1]$;
- (b) \mathcal{E}_{OM} (Joule), i.e., the total computing-plus-network energy consumed by the infrastructure of Figure 1 under \vec{x}_{OM} . If the returned \mathcal{E}_{OM} is infinite, then the *All-Mobile* task allocation \vec{x}_{OM} is infeasible. The *OM_S* function calls, in turn, the *RAP* function;
4. optionally, calls the *OF_S* function and returns:
- (a) $\vec{RS}_{OF} \stackrel{\text{def}}{=} [f_M, f_F, f_C, R_U, R_D, B_U, B_D]$ (bit/s), i.e., the seven-tuple vector of the optimal resource allocation under the $(1 \times V)$ *All-Fog* task allocation vector: $\vec{x}_{OF} \stackrel{\text{def}}{=} [1, 0, \dots, 0, 1]$;
- (b) \mathcal{E}_{OF} (Joule), i.e., the total computing-plus-network energy consumed by the infrastructure of Figure 1 under \vec{x}_{OF} . If the returned \mathcal{E}_{OF} is infinite, then the *All-Fog* task allocation \vec{x}_{OF} is infeasible. The *OF_S* function calls, in turn, the *RAP* function;
5. optionally, calls the *OC_S* function and returns:
- (a) $\vec{RS}_{OC} \stackrel{\text{def}}{=} [f_M, f_F, f_C, R_U, R_D, B_U, B_D]$ (bit/s), i.e., the seven-tuple vector of the optimal resource allocation under the $(1 \times V)$ *All-Cloud* task allocation vector: $\vec{x}_{OC} \stackrel{\text{def}}{=} [1, 2, \dots, 2, 1]$;
- (b) \mathcal{E}_{OC} (Joule), i.e., the total computing-plus-network energy consumed by the infrastructure of Figure 1 under \vec{x}_{OC} . If the returned \mathcal{E}_{OC} is infinite, then, the *All-Cloud* task allocation \vec{x}_{OC} is infeasible. The *OC_S* function calls, in turn, the *RAP* function;
6. optionally, calls the *O_TAS_par* function for parallel execution and returns:
- (a) $\vec{x}_{OTAS} \in \{0, 1, 2\}^V$, i.e., the ternary V -tuple best allocation vector computed by *O_TAS* under the assumption that the optimization of the resource allocation is *not* performed;
- (b) \mathcal{E}_{OTAS} (Joule), i.e., the total computing-plus-network energy consumed by the infrastructure of Figure 1 under \vec{x}_{OTAS} . If the returned \mathcal{E}_{OTAS} is infinite, then the *O_TAS* task allocation \vec{x}_{OTAS} is infeasible. The *O_TAS* function calls, in turn, the *Crossover*, *Mutation*, and *evaluatestaticenergy_p* functions;
7. optionally, calls the *ES_S_par* function and returns:
- (a) $\vec{x}_{ESS} \in \{0, 1, 2\}^V$, i.e., the ternary V -tuple best task allocation vector computed by performing the exhaustive search over the full population of task allocation vectors of the size 3^{V-2} ;
- (b) $\vec{RS}_{ESS} \stackrel{\text{def}}{=} [f_M, f_F, f_C, R_U, R_D, B_U, B_D]$ (bit/s), i.e., the seven-tuple vector of the optimal resource allocation under \vec{x}_{ESS} ;
- (c) \mathcal{E}_{ESS} (Joule), i.e., the total computing-plus-network energy consumed by the infrastructure of Figure 1 under \vec{x}_{ESS} . If the returned \mathcal{E}_{ESS} is infinite, then the overall afforded optimization problem is infeasible. The *ES_S* function calls, in turn, the *RAP* function and requires that the task allocation vectors \vec{x}_{best} and \vec{x}_{OTAS} are already available;
8. optionally, calls the *FogTracker* function. It returns the time plots over the interval: $[1, iteration_number]$ of the:
- (a) total energy \mathcal{E}_{TOT} consumed by the overall Mobile-Fog-Cloud ecosystem;

- (b) the corresponding energy \mathcal{E}_{NET} consumed by the up/down WiFi, Cellular, and Backhaul connections of Figure 1; and,
- (c) the behavior of the λ multiplier associated with the hard throughput constraint of Equation (4b),

when abrupt changes in the pattern of the allocated tasks and/or maximum bandwidths of the WiFi/Cellular connections occur. The user may set the magnitude of these changes, in order to test various (possibly mobility induced) time-fluctuations of the simulated environment of Figure 1 (see Section 4.4 in the sequel for a full description of the *FogTracker* function and supported options).

The current version of the *VirtFogSim* simulator is equipped with a (rich) *Graphical User Interface* (GUI) that displays:

1. the numerical values of the up/down WiFi-Cellular-Backhaul bandwidths and Mobile-Fog-Cloud computing frequencies returned by the allocation policies *GeneticTA_par*, *O_TAS_par*, *ES_S_par*, *OM_S*, *OF_S*, and *OC_S* in the form of colored bar plots;
2. the (generally different) task allocation patterns performed by *GeneticTA_par*, *O_TAS_par*, *ES_S_par*, *OM_S*, *OF_S*, and *OC_S* in the form of suitably-colored labeled graphs of the underlying application DAG (see Appendix A.2 in the sequel for a description of the main screen-shoots returned by the GUI of *VirtFogSim*).

Furthermore, *VirtFogSim* allows the user to:

1. set 67 input parameters, in order to customize the desired computing and communication setup of the infrastructure of Figure 1;
2. select any subset of the (aforementioned) *GeneticTA_p*, *OM_S*, *OC_S*, *OF_S*, *O_TAS_par*, *ES_S_par*, and *FogTracker* functions, in order to test and compare various task and/or resource allocation strategies under the scenario dictated by the desired input parameters.

4.2. Supported Task Allocation Strategies and Adaptive Resource Allocation

In this subsection, we describe the supported task allocation strategies provided by the *VirtFogSim* and listed in Table 2.

We begin with the *parallel Only-Task Allocation Strategy* (*O_TAS_par*) function that acts as follows:

1. it performs task allocation by implementing (in a parallel way) a genetic algorithm. For this purpose, *O_TAS_par* calls the *Crossover* and *Mutation* functions over a randomly-generated population of $\{0, 1, 2\}$ -ternary task allocation vectors of size PS ;
2. it evaluates the energy of each tested ternary allocation vector by computing the corresponding energies: \mathcal{E}_M , \mathcal{E}_F , \mathcal{E}_C , \mathcal{E}_{WiFi} , \mathcal{E}_{WD} , and \mathcal{E}_{CELL} under the static (e.g., not optimized) maximal resource allocation vector: $\vec{RS}^{MAX} = [f_M^{MAX}, f_F^{MAX}, f_C^{MAX}, R_U^{MAX}, R_D^{MAX}, B_U^{MAX}, B_D^{MAX}, R_0]$ (bit/s). For this purpose, *O_TAS_par* no longer calls the *RAP_p* function, but calls the *evaluatestaticenergy_p* function.

The *O_TAS_par* outputs are:

1. the V -tuple $(0, 1, 2)$ -ternary minimum-energy task allocation vector: x_{OTAS} , which is computed by applying the genetic algorithm run by *O_TAS_par* under \vec{RS}^{MAX} ;
2. the corresponding total energy: \mathcal{E}_{OTAS} (Joule) and network energy: E_{OTAS}^{NET} (Joule) consumed by \vec{x}_{OTAS} under the (aforementioned) fixed maximal resource allocation vector \vec{RS}^{MAX} .

The function *O_TAS_par* assumes that all the global variables of parallel *VirtFogSim* are already setup. It also utilizes the following main set of local variables: *Popmatrix*, *Childlist*, *Mutationlist*, and *Candidatelist*, all of dimensions $PS \times (V + 1)$.

Algorithm 1 reports a pseudo-code of the O_TAS_par function. Due to the utilization of the parfor-cycle, the resulting asymptotic computational complexity of the O_TAS_par function scales up as: $\mathcal{O}((PS \times G_{MAX}) / n_{core})$, where:

1. PS is the population size of the checked task allocation vectors;
2. G_{MAX} is the number of iterations (that is, the number of generations) run by the genetic algorithm implemented by O_TAS_par ; and,
3. $n_{core} \geq 1$ is the number of available parallel cores that support the execution of *VirtFogSim*.

Algorithm 1 O_TAS_par function.

function: $[\vec{x}_{OTAS}, \mathcal{E}_{OTAS}, \mathcal{E}_{OTAS}^{NET}] = O_TAS_par(PS, CF, G_{MAX}, MN)$.

Output: $(1 \times V)$ ternary vector of the best task allocation $\vec{x}_{OTAS} \in \{0, 1, 2\}^V$ under the fixed maximal resource allocation vector; scalar total energy \mathcal{E}_{OTAS} (Joule) and network energy \mathcal{E}_{OTAS}^{NET} (Joule) consumed by \vec{x}_{OTAS} under the fixed maximal allocation vector.

▷ **Begin O_TAS_par function**
▷ **Initialization phase**

- 1: Randomly generate a list $\{\vec{x}_i \in \{0, 1, 2\}^V, i = 1, \dots, PS\}$ of task allocation vectors and store it into $[Popmatrix]$;
 - 2: Compute and store into the $(V + 1)^{th}$ column of $[Popmatrix]$ the energy consumed by each $\vec{x} \in [Popmatrix]$ by calling PS times the *evaluatestaticenergy_p* function;
 - 3: Sort the PS elements of $[Popmatrix]$ for increasing values of their energies;
 - 4: Copy the first row of the sorted $[Popmatrix]$ into \vec{x}_{OTAS} , \mathcal{E}_{OTAS} , and \mathcal{E}_{OTAS}^{NET} ;
 - 5: Set the number $Q = \lceil CF \times PS \rceil$ of the elements of $[Popmatrix]$ for crossover at each generation;
 - 6: **for** $j = 1 : G_{MAX}$ **do** ▷ **Iterative phase**
 - 7: Perform the pair-wise crossover of the first Q elements of $[Popmatrix]$ by calling $(Q/2)$ times the *Crossover* function and store the Q crossover elements in $[Childlist]$;
 - 8: Randomly mute MN positions of the last $(PS - Q)$ elements of $[PopMatrix]$ by calling $(PS - Q)$ times the *Mutation* function and store the mutated elements into $[Mutationlist]$;
 - 9: Compute and store the consumed energy of each element of $[Childlist]$ and $[Mutationlist]$ by calling PS times the function *evaluatestaticenergy_p*;
 - 10: Copy the first Q elements of $[Popmatrix]$ and the overall $[Childlist]$ and $[Mutationlist]$ into $[Candidatelist]$;
 - 11: Sort the $(PS + Q)$ elements of $[Candidatelist]$ for increasing values of their energies;
 - 12: Copy the first PS elements of $[Candidatelist]$ into $[Popmatrix]$;
 - 13: **if** the total energy of the first element of $[Popmatrix]$ is lower than \mathcal{E}_{OTAS} **then**
 - 14: Copy the first row of $[Popmatrix]$ into \vec{x}_{OTAS} , \mathcal{E}_{OTAS} and \mathcal{E}_{OTAS}^{NET} ;
 - 15: **end if**
 - 16: **end for**
 - 17: **return** $[\vec{x}_{OTAS}, \mathcal{E}_{OTAS}, \mathcal{E}_{OTAS}^{NET}]$. ▷ **End O_TAS_par function**
-

The RAP_p function implements primal-dual adaptive iterations on the cluster of available parallel workers that supports the execution of *VirtFogSim*. The goal is to perform the optimal resource allocation under the input task allocation vector \vec{x} . Its input formal variables are the vectors \vec{x} , \vec{z} , and \vec{s} and the matrices A , Da , and S_{RAP} . The corresponding output variables are the vector \vec{y} and the scalars $\tilde{\mathcal{E}}_{TOT}$ and $\tilde{\mathcal{E}}_{NET}$.

Specifically, we have that:

1. \vec{x} is a $(1 \times V)$ -dimensional $\{0, 1, 2\}$ -ternary vector. It fixes the allocation to the Fog/Mobile/Cloud nodes of the V tasks that compose the considered application DAG. Specifically, $x(i) = 0, 1$, and 2 means that the i^{th} application task is executed at the Fog clone, Mobile device, and Cloud clone, respectively;

2. $z \triangleq [f_M^0, f_F^0, f_C^0, R_U^0, R_D^0, B_U^0, B_D^0, \lambda^0]$ is a (1×8) vector of real-valued non-negative scalars. It fixes the vector starting point of the primal-dual iterations to be performed. Its first seven components are measured in bit/s, while the starting Lagrange multiplier λ_0 is measured in Joule.
3. $\tilde{y} \triangleq [\tilde{f}_M, \tilde{f}_F, \tilde{f}_C, \tilde{R}_U, \tilde{R}_D, \tilde{B}_U, \tilde{B}_D]$ is a (1×7) vector of real-valued non-negative scalars. It returns the vector of the *optimal* resource allocation attained by the performed primal-dual iterations. Its seven components are measured in bit/s;
4. $\tilde{\mathcal{E}}_{TOT}$ is the total communication-plus-computing energy consumed by the computed optimal resource allocation \tilde{y} under the given task allocation x . $\tilde{\mathcal{E}}_{TOT}$ is measured in Joule.
5. $\tilde{\mathcal{E}}_{NET}$ is the overall network energy consumed by the computed optimal resource allocation \tilde{y} under the given task allocation x . $\tilde{\mathcal{E}}_{NET}$ is measured in Joule;
6. s , A and Da are respectively the global workload vector, adjacency matrix, and edge matrix of the underlying DAG;
7. S_{RAP} is the (1×54) -dimensional vector of all other global variables used by RAP_p for its execution.

Since RAP_p is executed by the calling functions in the body of parfor-cycles, the global variables: s , A , Da , and S_{RAP} must be passed to the RAP_p function as input parameters, in order to guarantee the synchronization of the parfor-cycles performed by multiple parallel workers.

The RAP_p function requires $(110 + 6 \times V)$ scalar local variables and two $(1 \times I_{MAX})$ vector local variables. The I_{MAX} parameter is stored by S_{RAP} and fixes the maximum number of allowed primal-dual iterations. The asymptotic implementation complexity of the RAP_p function scales as: $\mathcal{O}(8 \times I_{MAX})$.

The parallel *Exhaustive Search-Strategy* (ES_S_par) function returns:

1. the V -tuple $(0, 1, 2)$ -ternary \vec{x}_{ESS} . It is the globally best task allocation vector, which is generated by performing the exhaustive search over the full population of the $3^{(V-2)}$ ternary task allocation vectors. The first and last components of each ternary task allocation vector are unit valued, i.e., the first and last tasks of the application DAG are executed, by design, at the Mobile device;
2. the seven-tuple resource allocation vector: $\vec{RS}_{ESS} = [f_M^*, f_F^*, f_C^*, R_U^*, R_D^*, B_U^*, B_D^*]$ (bit/s), corresponding to the returned global task allocation vector \vec{x}_{ESS} ;
3. the total communication-plus-computing energy \mathcal{E}_{ESS} (Joule), network energy \mathcal{E}_{NET}^{ESS} (Joule), and bandwidth RB_{ESS} (bit/s) of the backhaul connection consumed under the returned searched global best task allocation vector \vec{x}_{ESS} .

Being obtained through an exhaustive search, \mathcal{E}_{ESS} is, by design, the global minimum total energy consumed under the considered application scenario of Figure 1.

In order to guarantee that the full search space of the the task allocation vectors is actually explored, the ES_S_par function calls the auxiliary function: *find_allocations*, which returns the: $(3^{(V-2)} \times V)$ -dimensional *POP* matrix of all possible $(0, 1, 2)$ -ary task allocation patterns. Afterward, ES_S_par calls the RAP_p function under each ternary task allocation vector returned by *find_allocations*, in order to evaluate the resulting best resource allocation vector, the corresponding total and network consumed energies, and the utilized backhaul bandwidth. After checking all the $3^{(V-2)}$ task allocations, ES_S_par sorts them for increasing values of their total consumed energies and, then, picks out the first element of the attained sorted list. This last comprises the globally best task allocation x_{ESS} , its associated resource allocation vector \vec{RS}_{ESS} , the corresponding consumed energies \mathcal{E}_{ESS} and \mathcal{E}_{NET}^{ESS} , and the utilized backhaul bandwidth RB_{ESS} .

A pseudo-code of the ES_E_par function is reported in Algorithm 2. The resulting asymptotic computational complexity scales up as: $\mathcal{O}\left(\left(3^{(V-2)} \times 8 \times G_{MAX}\right) / n_{core}\right)$.

Algorithm 2 *ES_S_par* function.

function: $[\vec{x}_{ESS}, \vec{RS}_{ESS}, \mathcal{E}_{ESS}^{NET}, RB_{ESS}] = ES_S_par$.

Output: $(1 \times V)$ ternary vector of the best task allocation over the full population \vec{x}_{ESS} ; (1×7) resource allocation vector: $\vec{RS}_{ESS} \triangleq [f_M^*, f_F^*, f_C^*, R_U^*, R_D^*, B_U^*, B_D^*]$ (bit/s) under \vec{x}_{ESS} ; total energy \mathcal{E}_{ESS} (Joule) and network energy \mathcal{E}_{ESS}^{NET} (Joule) and backhaul bandwidth RB_{ESS} (bit/s) consumed by \vec{x}_{ESS} .

▷ **Begin ES_S_par function**

- 1: Call the find-allocation function, in order to generate the full set list: $\{\vec{x}_i \in \{0, 1, 2\}^V, i = 1, \dots, 3^{V-2}\}$ of ternary V -tuple candidate task allocation vectors, with the first and last elements set to the unit;
 - 2: Store (by row) $\{\vec{x}_i\}$ into the first V columns of the $[POP]$ matrix;
 - 3: **for** $i = 1 : 3^{V-2}$ **do**
 - 4: Compute the resource allocation vector and the total consumed energy of each $\{\vec{x}_i\}$ by calling the RAP_p function, and store them (by row) into the last columns of the $[POP]$ matrix;
 - 5: **end for**
 - 6: Sort the 3^{V-2} elements of the $[POP]$ matrix for increasing values of their total energies;
 - 7: Store the first row of the sorted $[POP]$ matrix into $\vec{x}_{ESS}, \vec{RS}_{ESS}, \mathcal{E}_{ESS}, \mathcal{E}_{ESS}^{NET}$ and RB_{ESS} ;
 - 8: **return** $[\vec{x}_{ESS}, \vec{RS}_{ESS}, \mathcal{E}_{ESS}, \mathcal{E}_{ESS}^{NET}, RB_{ESS}]$.
▷ **End ES_S_par function**
-

The parallel Genetic Task Allocation (*GeneticTA_par*) function assumes that all the global variables of parallel *VirtFogSim* are already setup. The *GeneticTA_par* function jointly optimizes the task allocation and the resource allocation in an adaptive way. It returns:

1. the $(0, 1, 2)$ -ternary V -long vector \vec{x}_{best} of the best searched task allocation;
2. the corresponding (1×7) vector: $\vec{RS}_{best} = [f_M^*, f_F^*, f_C^*, R_U^*, R_D^*, B_U^*, B_D^*]$ (bit/s), of the optimal resource allocation computed by the RAP_p function under the best task allocation vector \vec{x}_{best} ;
3. the total and network energies: $\mathcal{E}_{best}, \mathcal{E}_{best}^{NET}$ (Joule) that are consumed under the returned \vec{RS}_{best} ;
4. the actual transmission rate: RB_{best} (bit/s) of the two-way Fog \leftrightarrow Cloud backhaul connection.

Task allocation is performed by *GeneticTA_par* by running a genetic algorithm. For this purpose, *GeneticTA_par* calls the *Crossover* and *Mutation* functions.

Resource allocation is performed by *GeneticTA_par* in an adaptive way. For this purpose, at each parallel iteration called by a parfor-cycle, *GeneticTA_par* calls the RAP_p function. Afterward, *GeneticTA_par* picks up and stores:

1. the best (that is, the minimum energy) task allocation vector \vec{x}_{best} ;
2. the corresponding resource allocation vector \vec{RS}_{best} ;
3. the total consumed energy \mathcal{E}_{best} and the corresponding network energy \mathcal{E}_{best}^{NET} computed up to the current generation.

PS is the size of each generation of task allocation vectors; G_{MAX} is the number of carried out generations; CF is the fraction of crossover elements of the current generation; and MN is the number of mutated locations on a per-task-allocation basis.

Let $n_{core} \geq 1$ be the number of parallel cores (e.g., parallel workers) managed by the *Parallel Toolbox* of MATLAB, in order to support the execution of the *VirtFogSim* package. Hence, the resulting asymptotic implementation complexity scales as in: $\mathcal{O}((PS \times G_{MAX} \times 8 \times I_{MAX}) / (n_{core}))$.

The *Only Mobile Strategy (OM_S)* function assumes that all the V tasks of the assigned application DAG are executed at the Mobile device. Then, it returns the (1×7) vector $\vec{RS}_{OM} = [f_M, f_F, f_C, R_U, R_D, B_U, B_D]$ (bit/s) of the corresponding resource allocation, the corresponding total computing-plus-communication consumed energy \mathcal{E}_{OM} (Joule), and the (vanishing) network energy: \mathcal{E}_{NET}^{OM} (Joule). It is expected that the returned f_M is not zero, while the returned $f_C, f_F, R_U, R_D, B_U, B_D$, and \mathcal{E}_{NET}^{OM} vanish. The asymptotic complexity of the *OM_S* function scales up as in: $\mathcal{O}(8 \times I_{MAX})$.

The *Only Fog Strategy* (OF_S) function assumes that the first and last tasks of the assigned application DAG are executed at the Mobile device, while all the remaining inner $(V - 2)$ tasks are executed at the Fog clone. Then, OF_S returns the (1×7) vector $\vec{RS}_{OF} = [f_M, f_F, f_C, R_U, R_D, B_U, B_D]$ (bit/s) of the corresponding resource allocation, the corresponding total computing-plus-networking consumed energy \mathcal{E}_{OF} , and the network energy \mathcal{E}_{NET}^{OF} (Joule). It is expected that the returned $f_M, f_F, R_U,$ and R_D are not zero, while the returned $f_C, B_U,$ and B_D vanish.

The *Only Cloud Strategy* (OC_S) function assumes that the first and last tasks of the assigned application DAG are executed at the Mobile device, while all the remaining inner $(V - 2)$ tasks are executed at the Cloud clone. Then, it returns the (1×7) vector $\vec{RS}_{OC} = [f_M, f_F, f_C, R_U, R_D, B_U, B_D]$ (bit/s) of the corresponding resource allocation, the corresponding total computing-plus-networking consumed energy \mathcal{E}_{OC} (Joule), and the network energy \mathcal{E}_{NET}^{OC} (Joule).

It is expected that the returned $f_M, f_C, B_U,$ and B_D are not zero, while the returned $f_F, R_U,$ and R_D vanish. The functions $OM_S, OF_S,$ and OC_S call a sequential version RAP of the RAP_p function, with the following initialization vector: $z = [0, 0, 0, 0, 0, 0, 1]$.

4.3. Implemented Auxiliary Functions

In this subsection, we provide the description of some auxiliary functions called from the previous allocation routines. These auxiliary functions are listed in Table 3.

Table 3. Auxiliary functions implemented by *VirtFogSim*.

Function	Description
$[Child_v^1, Child_v^2] = \text{Crossover}(Parent_v^1, Parent_v^2)$	It implements the Crossover operation.
$\vec{out}_v = \text{Mutation}(\vec{x}_v)$	It implements the Mutation operation.
$[\mathcal{E}_{aux}, \mathcal{E}_{NET}] = \text{evaluatestaticenergy_p}(\vec{x}_v, \vec{s}, A, Da, S_{RAP})$	It evaluates the static energy evaluation in O_TAS .
$TA = \text{find_allocations}(N, K)$	It returns all the K^N patterns of N numbers that can assume the first K integer values: $0, 1, 2, \dots, K - 1$.

In the *Crossover* function, $Parent_v^1$ and $Parent_v^2$ are two V -tuple parent vectors to crossover, while $Child_v^1$ and $Child_v^2$ are the resulting V -tuple vectors produced by the performed crossover operation. The crossover operation generates an integer index I over the set $\{2, 3, \dots, (V - 1)\}$. It is the pointer to the crossover point. Afterward, $Child_v^1$ and $Child_v^2$ are obtained by swapping the first I components of $Parent_v^1$ and $Parent_v^2$ with the last $(V - I)$ components of $Parent_v^2$ and $Parent_v^1$, respectively.

In the *Mutation* function, the global input parameter MN fixes the number of the components of the input vector \vec{x}_v to be mutated. \vec{x}_v is the V -tuple $(0, 1, 2)$ -ternary input vector to be mutated. \vec{out}_v is the V -tuple $(0, 1, 2)$ -ternary mutated output vector. It is guaranteed that the first and last components of \vec{out}_v are unit valued. The function *Mutation* generates MN random positions (with $1 \leq MN \leq (V - 2)$) and MN random $(0, 1, 2)$ -ternary mutation numbers. Then, it mutates \vec{x}_v at the generated MN positions by replacing the generated MN mutation numbers for the corresponding components of \vec{x}_v .

In the *evaluatestaticenergy_p* function for parallel execution, \vec{x}_v is the $(1 \times V)$ -long input $(0, 1, 2)$ -ternary task allocation vector to be checked. Since the function *evaluatestaticenergy_p* is called in the body of the parfor-cycle by the O_TAS_par function, all the needed global variables: $\vec{s}, A, Da,$ and S_{RAP} must be passed to *evaluatestaticenergy_p* as input parameters. This guarantees the synchronization of the used global variables under parallel execution. If \vec{x}_v is a feasible task allocation pattern, \mathcal{E}_{aux} and \mathcal{E}_{NET} are the total and network energies consumed by \vec{x}_v under the static maximal resource allocation vector defined as: $\vec{RS}^{MAX} = [f_M^{MAX}, f_F^{MAX}, f_C^{MAX}, R_U^{MAX}, R_D^{MAX}, B_U^{MAX}, B_D^{MAX}]$ (bit/s). If \vec{x}_v is an infeasible task allocation pattern, then $\mathcal{E}_{aux} = \mathcal{E}_{NET} = \infty$ is returned. The asymptotic complexity of the *evaluatestaticenergy_p* function is: $\mathcal{O}(7)$, where seven is the number of the computed resource variables.

Finally, the *find_allocations* function returns in the matrix *TA* all the possible: K^N patterns of N numbers that can assume the first K integer values: $0, 1, 2, \dots, K - 1$. This function is used by the *ES_S_par* strategy function to obtain all the possible allocations to be tested in the exhaustive search.

4.4. Dynamic Performance Tracking Function

The numerical outputs of the *FogTracker* function are three matrices that store the time behaviors of: (i) the consumed total energy \mathcal{E}_{TOT} (Joule); (ii) the corresponding consumed network energy \mathcal{E}_{NET} (Joule); and (iii) the *lambda* multiplier (Joule), for values of the iteration index going from one to the given *iteration_number*. The goal of *FogTracker* is to test the convergence rate to the steady-state and the steady-state stability of the primal-dual iterations performed by the *RAP_p* function when abrupt changes of the maximum allowed WiFi and Cellular up/down bandwidths and task allocation vectors simultaneously happen. For this purpose, at the time indexes:

1. $\text{round}((1/5) \times \text{iteration_number}) + 1$;
2. $\text{round}((2/5) \times \text{iteration_number}) + 1$;
3. $\text{round}((3/5) \times \text{iteration_number}) + 1$;
4. $\text{round}((4/5) \times \text{iteration_number}) + 1$,

the initial values of the global variables: $R_U^{MAX}, R_D^{MAX}, B_U^{MAX}, B_D^{MAX}$ undergo abrupt changes. These changes are obtained by multiplying them by the (non-negative) input factors: $\text{jump}_{WiFi}^1, \text{jump}_{CELL}^1, \text{jump}_{WiFi}^2$, and jump_{CELL}^2 . These jump coefficients are declared as global variables and set by the user. At the same time, the task allocation vector passes from \vec{x}_1 to \vec{x}_2 and, then, from \vec{x}_2 to \vec{x}_3 . Finally, it comes back to \vec{x}_1 .

The resulting time behaviors of the total energies, network energies, and lambda multiplier values over the time window: $[1, \text{iteration_number}]$ are plotted under the three user-specified values of a_{MAX} , which are stored by the (1×3) global input vector \vec{a}_{MAX-FT} . In so doing, the *FogTracker* function tests the convergence rate of the *RAP_p* function when the task allocation vector undergoes abrupt changes. The actual feasibility of the task allocation vectors: \vec{x}_1, \vec{x}_2 , and \vec{x}_3 is explicitly checked in the body of the *FogTracker* function. If at least one of these feasibility checks fails, the *FogTracker* function generates a suitable error message and, then, halts.

In detail, the *FogTracker* function performs the following nine steps:

1. it begins to run the *RAP_p* function over the time interval: $[1, \text{round}((1/5) \times \text{iteration_number})]$, under the original settings of: $R_U^{MAX}, R_D^{MAX}, B_U^{MAX}, B_D^{MAX}$ dictated by the main program *VirtFogSim* and the first input task allocation vector \vec{x}_1 ;
2. at the end of the iteration number: $\text{round}((1/5) \times \text{iteration_number})$, the WiFi maximum bandwidths: R_U^{MAX} and R_D^{MAX} are multiplied by the (non-negative) scaling factor: jump_{WiFi}^1 , while the maximum cellular bandwidths: B_U^{MAX} and B_D^{MAX} are multiplied by the corresponding (possibly, coincident) scaling factor: jump_{CELL}^1 . Furthermore, the task allocation vector is changed into \vec{x}_2 . In so doing, both the maximum available bandwidths and the task allocation vector undergo abrupt (typically, user mobility induced) variations;
3. the *RAP_p* function runs over the time-interval: $[\text{round}((1/5) \times \text{iteration_number}) + 1, \text{round}((2/5) \times \text{iteration_number})]$ under the setting of Step 2. Its initial vector is the last returned vector of the *RAP_p* at the previous iteration: $\text{round}((1/5) \times \text{iteration_number})$;
4. at the end of iteration number: $\text{round}((2/5) \times \text{iteration_number})$, all four maximum bandwidths: $R_U^{MAX}, R_D^{MAX}, B_U^{MAX}$, and B_D^{MAX} are restored to their original values. Furthermore, even the task allocation vector is set back to its original value \vec{x}_1 ;
5. the *RAP_p* function runs over the time interval: $[\text{round}((2/5) \times \text{iteration_number}) + 1, \text{round}((3/5) \times \text{iteration_number})]$ under the setting of Step 4. Its initial vector is the last returned vector of the *RAP_p* at the previous iteration number: $\text{round}((2/5) \times \text{iteration_number})$;

6. after the iteration number: $\text{round}((3/5) \times \text{iteration_number})$, the WiFi maximum up/down bandwidths: R_U^{MAX} , and R_D^{MAX} are multiplied by the non-negative scaling factor jump_{WiFi}^2 , while the cellular maximum up/down bandwidths: B_U^{MAX} and B_D^{MAX} are multiplied by the scaling factor: jump_{CELL}^2 . At the same time, even the task allocation vector is changed to the third value \vec{x}_3 . In so doing, both the maximum available bandwidths and the task allocation vector undergo abrupt (typically, user mobility-induced) variations;
7. the RAP_p function runs over the time-interval: $[\text{round}((3/5) \times \text{iteration_number}) + 1, \text{round}((4/5) \times \text{iteration_number})]$ under the setting of Step 6. Its initial vector is the last returned vector of the RAP_p at the previous iteration number: $\text{round}((3/5) \times \text{iteration_number})$;
8. after the iteration number: $\text{round}((4/5) \times \text{iteration_number})$, all four WiFi/CELL maximum up/down bandwidths are restored to their original values. Furthermore, even the task allocation vector is set back to the first value \vec{x}_1 ;
9. finally, the RAP_p function runs over the time-interval: $[\text{round}((4/5) \times \text{iteration_number}) + 1, \text{iteration_number}]$ under the setting of Step 8. Its initial vector is the last returned vector of the RAP_p at the previous iteration number: $\text{round}((3/5) \times \text{iteration_number})$.

Graphic plots of the time behaviors of the returned \mathcal{E}_{TOT} , \mathcal{E}_{NET} , and λ are displayed at the end of each *FogTracker* run (see Section 5 in the sequel). From the outset, it follows that the asymptotic complexity of the *FogTracker* function scales up as in: $\mathcal{O}(8 \times \text{iteration_number} \times 3)$.

Overall, Table 4 reports a synoptic view of the asymptotic computational complexities of the described *GeneticTA_par*, *OM_S*, *OF_S*, *OC_S*, *O_TAS_par*, *ES_S_par*, and *FogTracker* functions under their parallel implementations.

Table 4. A synoptic overview of the computational complexities of the main functions supported by the VirtFogSim package.

Function	Asymptotic Computational Complexity
<i>GeneticTA_par</i>	$\mathcal{O}((PS \times G_{MAX} \times 8 \times I_{MAX}) / (n_{core}))$
<i>OM_S</i>	$\mathcal{O}(8 \times I_{MAX})$
<i>OF_S</i>	$\mathcal{O}(8 \times I_{MAX})$
<i>OC_S</i>	$\mathcal{O}(8 \times I_{MAX})$
<i>O_TAS_par</i>	$\mathcal{O}((PS \times G_{MAX}) / n_{core})$
<i>ES_S</i>	$\mathcal{O}((3^{(V-2)} \times 8 \times G_{MAX}) / n_{core})$
<i>FogTracker</i>	$\mathcal{O}(8 \times \text{iteration_number} \times 3)$

5. VirtFogSim: Supported Formats of the Rendered Data

Under the current version of the simulator, both the *VirtFogSim* and *VirtFogSimGUI* interfaces (see Appendix A) support four main formats, in order to render the results output by the seven optimization algorithms of Table 4. The functions used to obtain these formats are listed in Table 5 and fully described in the *VirtFogSim* User Guide (see Section 8). These rendering formats are:

1. the Tabular format. It is enabled by the *print_solution* graphic function;
2. the Colored Bar Plot format. It is enabled by the *plot_solution* graphic function;
3. the Colored Time Plot format. It is enabled by the *plot_FogTracker* graphic function;
4. the Colored Labeled DAG Map format. It is enabled by the *plot_DAG* graphic function.

Table 5. Rendering functions for the supported formats implemented in *VirtFogSim*.

Function	Description
$\text{print_solution}(\vec{RS}, \vec{E}, RB, \text{strategy})$	It implements the tabular format.
$\text{plot_solution}(\vec{x}, \vec{RS}, \vec{E}, RB, \text{fignumber}, \text{strategy})$	It implements the colored bar plot format.
$\text{plot_DAG}(\vec{s}, A, Da, \vec{x}, \text{strategy}, \text{type})$	It implements the colored labeled DAG map.
$\text{plot_FogTracker}(\mathcal{E}_m, \mathcal{E}_m^{NET}, \lambda_m, \text{fignumber}, \text{interpolation})$	It implements the colored time plot format.

Specifically, the *print_solution* function prints on the MATLAB prompt the result obtained by running the tested strategies under the selected DAG and given input parameters (see Table A2). Its input parameters are: (i) the vector \vec{RS} collecting the seven allocated resources; (ii) the vector \vec{E} collecting the total computing-plus-networking consumed energy; (iii) the bandwidth RB of the Fog↔Cloud two-way backhaul connection; and (iv) a string *strategy* representing the name of the tested strategy. Then, the function prints the following results in a numerical form:

1. the Computing/Communication Ratio (*CCR*);
2. the required minimum DAG execution throughput (TH_0^{MIN});
3. the per-core computing frequency at the Mobile device (f_M);
4. the per-core computing frequency at the Fog clone (f_F);
5. the per-core computing frequency at the Cloud clone (f_C);
6. the transport bit rate of the WiFi-based TCP/IP Mobile-to-Fog connection (R_U);
7. the transport bit rate of the WiFi-based TCP/IP Fog-to-Mobile connection (R_D);
8. the transport bit rate of the 3G/4G Cellular TCP/IP Mobile-to-Cloud connection (B_U);
9. the transport bit rate of the 3G/4G Cellular TCP/IP Cloud-to-Mobile connection (B_D);
10. the transport rate of the TCP/IP two-way Fog↔Cloud backhaul connection (R_B);
11. the total computing-plus-networking consumed energy (\mathcal{E}_{TOT});
12. the consumed computing energy (\mathcal{E}_{COM});
13. the consumed networking energy (\mathcal{E}_{NET});
14. the percent networking-to-total ratio of the consumed energies: $\%(\mathcal{E}_{NET}/\mathcal{E}_{TOT})$.

If the input vector \vec{RS} is empty, only a reduced set of information is printed.

The *plot_solution* function displays the: (i) utilized computing/bandwidth resources; (ii) task allocation pattern; and (iii) consumed energies, returned by the carried out *VirtFogSim* run in terms of suitable three-colored bar plots. In addition to the input used by the previous function, the other parameters are: (i) the (0, 1, 2)-ary allocation vector \vec{x} to be displayed; and (ii) the number of the *fignumber* to be displayed. The function renders a figure that is composed of three horizontal subplots. These subplots display:

1. the returned task allocation in the form of a three-color bar plot;
2. the allocated computing frequencies and network bandwidths in the form of a plot with eight bars;
3. the total, computing and networking consumed energies in the form of a plot with three bars.

If the passed task allocation vector \vec{x} and/or resource allocation vector \vec{RS} are empty, then only one or two subplots are rendered. An illustrative screen-shoot of the bar-plots rendered by the execution of the *GeneticTA_par* function is shown in Figure 3.

The *plot_DAG* function returns a graphic representation of the application DAG passed as the input. Its input parameters are: (i) the vector \vec{s} containing the nodes' weights of the DAG; (ii) the adjacency matrix A of the DAG; (iii) the matrix Da of the weights of the DAG edges; (iv) the vector \vec{x} of task allocation to be visualized; (v) a string of characters with the name of the policy that will be shown; and (vi) an optional three-valued parameter *type*, which allows selecting the desired DAG visualization format as follows:

1. all DAG nodes with the same color and unlabeled;

2. all DAG nodes with the same color and labeled by increasing identification numbers;
3. each DAG node is numbered and colored on the basis of the allocation actually stored by the input vector \vec{x} .

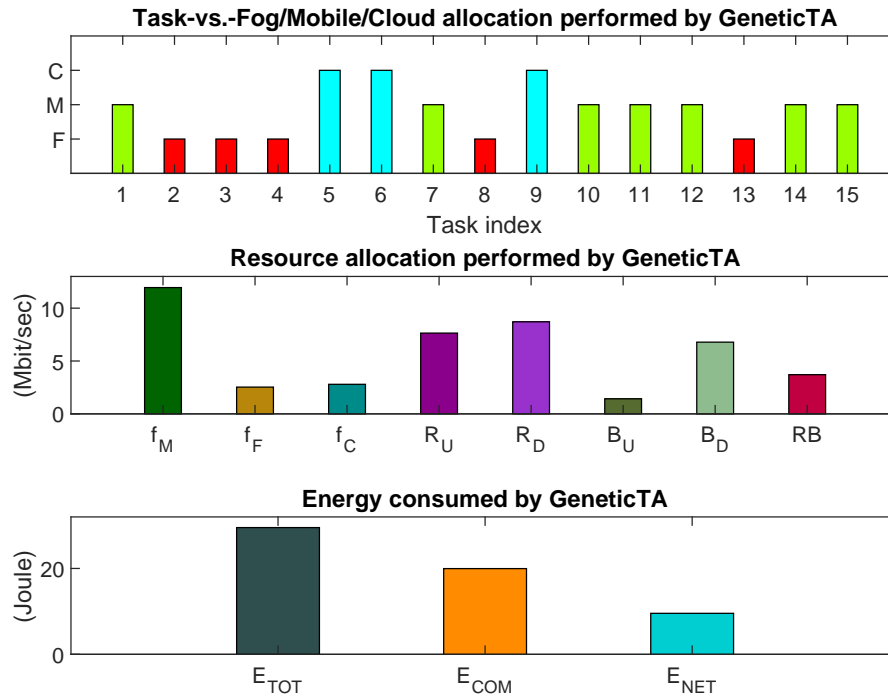


Figure 3. An illustrative screen-shoot of the bar-plots rendered by the execution of the *GeneticTA_par* function. (top) reported task allocation; (middle) reported resource allocation; (bottom) consumed total and network energies. Analogous bar-plot graphs are reported at the end of the executions of the other task allocation strategies supported by *VirtFogSim*.

The *plot_DAG* function needs the *Bioinformatics Toolbox* to be installed in the MATLAB environment. By default, the graph returned by *plot_DAG* retains the following features:

1. all the workload values labeling the DAG nodes and edges are expressed in kilo-bit;
2. red-colored tasks are understood to be allocated to the Fog node;
3. green-colored tasks are understood to be allocated to the Mobile node; and,
4. blue-colored tasks are understood to be allocated to the Cloud node.

An illustrative screen-shoot of the colored task map DAG rendered by the execution of the *GeneticTA_par* function is shown in Figure 4.

Finally, the *plot_fogTracker* function provides the graphic capabilities needed for a proper plot of the time-traces of the total energies, networking energies, and lambda multipliers generated by the *FogTracker* function under the three values of the step-size that are stored by the vector \vec{a}_{MAX-FT} (see Table A2). Its input parameters are: (i) the matrix \mathcal{E}_m of the total consumed energies returned by *FogTracker*; (ii) the matrix \mathcal{E}_m^{NET} of the consumed networking energies returned by *FogTracker*; (iii) the matrix λ_m of the values of the lambda multipliers returned by *FogTracker*; and (iv) an optional parameter *interpolation* allowing the plot of an interpolated version of the figures if set to one. The function renders a figure composed of three horizontal sub-plots that trace:

1. the total consumed computing-plus-networking energies under the three values of the step-size stored by \vec{a}_{MAX-FT} ;
2. the corresponding consumed networking energies;
3. the related values of the lambda multiplier.

Some illustrative screen-shoot of the dynamic plots rendered by the *FogTracker* function are shown in Figures 7–9 of Section 6.2.

DAG allocation performed by Genetic TA -- Black labels in kilobit

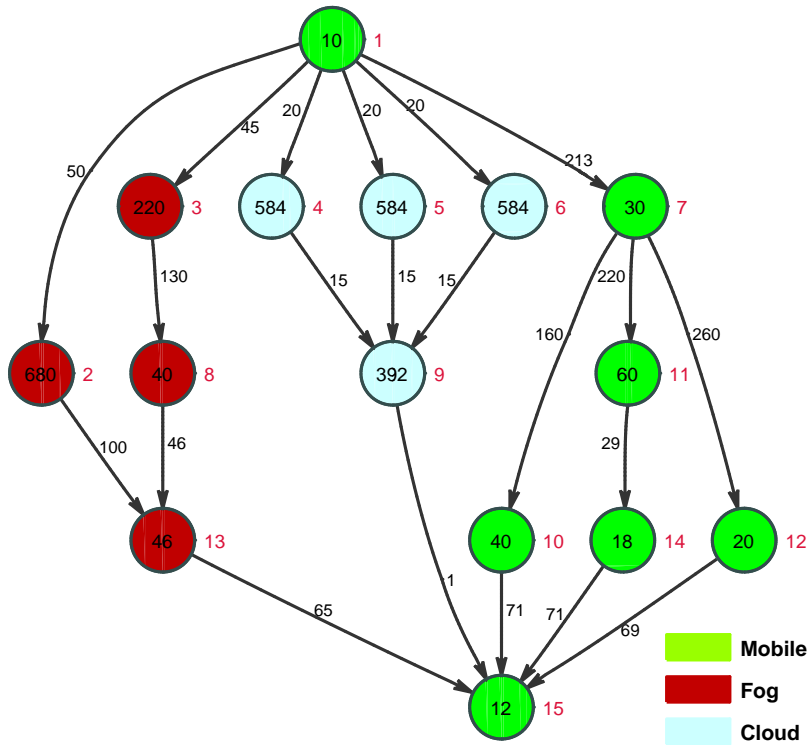


Figure 4. An illustrative screen-shoot of the colored task map DAG rendered by the execution of the *GeneticTA_par* function. Analogous bar-plot graphs are reported at the end of the executions of the other task allocation strategies supported by *VirtFogSim*.

Pre-Loaded Application DAGs

In the current version of the *VirtFogSimGUI* interface (see Figure A1 in Appendix A) is available an archive that stores fourteen test DAGs, together with the related sets of (suitably-tuned) input parameters. These DAGs are ready-for-use, e.g., they may be retrieved by the user and, then, run under both the (previously-described) interfaces of the simulator.

The archived DAGs were retrieved from the current literature [25–27,29–33] and feature a number of heterogeneous real-world applications of practical interest. Their topologies cover a large spectrum (e.g., tree, fork, parallel, mesh, and hybrid topologies, just to name a few), and their number of nodes ranges from $V = 9$ to $V = 45$.

6. VirtFogSim in Action: Testing Its Numerical Capabilities

The aim of this section is to provide insights about the actual capability of the developed *VirtFogSim* package by: (i) numerically testing and comparing the energy-delay-tracking performance of its natively-supported optimization tools of Section 4.4 (see also Table 4) under three use cases of practical interest; and (ii) checking the performance of the underlying MATLAB code by numerically profiling its simulation times and volumes of inter-core exchanged data over a spectrum of multi-core execution environments.

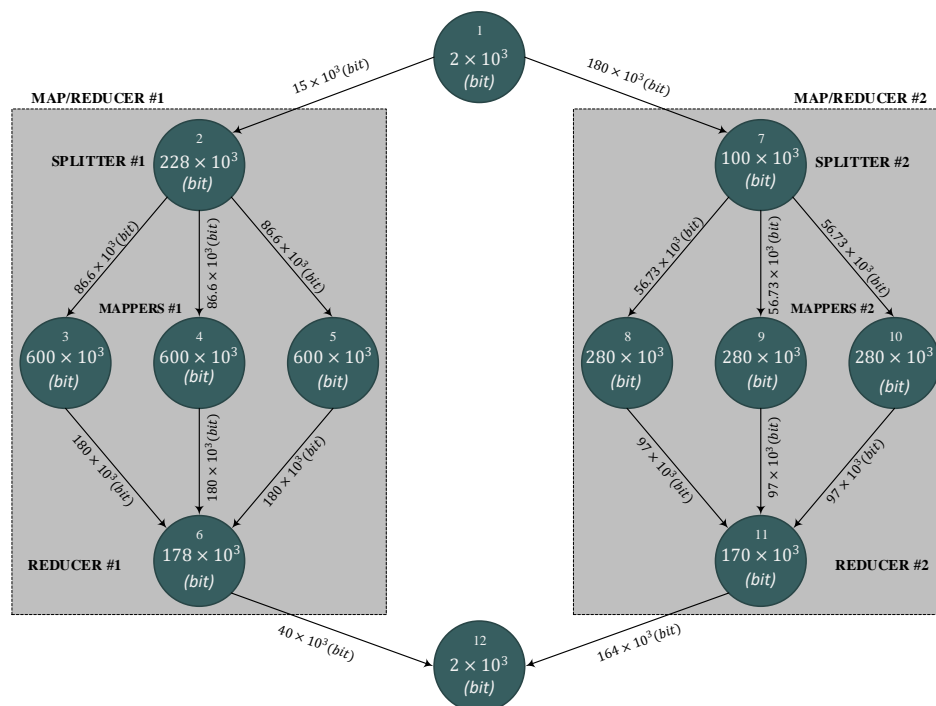
All the carried out simulations have been done by exploiting the capabilities of a hardware execution platform equipped with: (i) an Intel 10-core i9-7900X processor; (ii) 32 GB of RAM DDR 4; (iii) an SSD with 512 GB plus an HDD with 2 TB; (iv) a GPU ZOTAC GeForce GTX 1070. The release R2018a of MATLAB provided the underlying software execution platform. It is equipped with the MATLAB Parallel Toolbox, in order to exploit the multi-core capability (possibly) offered by the host hardware platform.

We anticipate that, unless otherwise stated, all the simulations have been carried out under the parameter setting reported in the last column of the final Table A2 in the Appendix.

6.1. Use Cases and Related DAGs

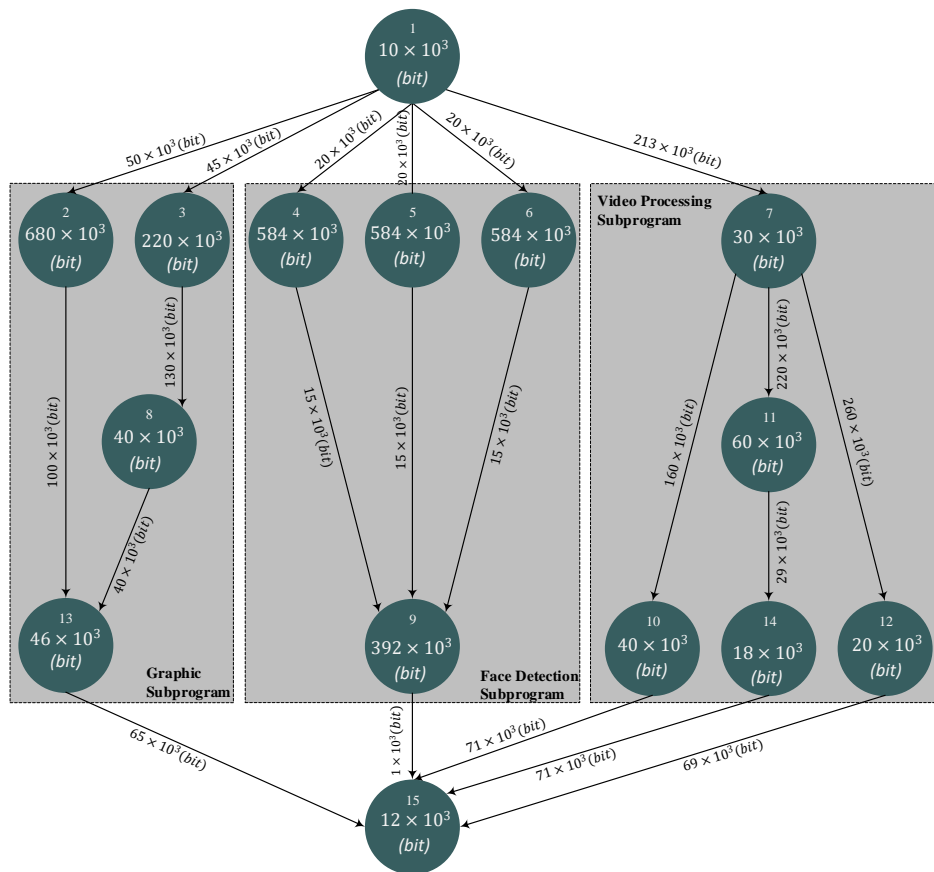
Three real-world use cases have been selected, in order to test the performance of the proposed *VirtFogSim* package under different application environments featured by (very) heterogeneous workflows and DAG topologies. Figures 5 and 6 report the graphs of the considered test DAGs.

DAG1 in Figure 5a describes the workflow of a (small-sized) parallel MAP/REDUCER computing application [34]. This DAG is composed of the parallel combination of two MAP/REDUCER nodes (e.g., MAP/REDUCER#1 on the left and MAP/REDUCER#2 on the right of Figure 5a, respectively). They share the same input and output nodes (e.g., Nodes 1 and 12 in Figure 5a, respectively), which are forced to be executed by the mobile device. Nodes 2 and 7 act as load balancers, while Nodes 3, 4, 5 and 8, 9, 10 act as Mappers. Finally, Nodes 6 and 11 perform reduce operations. The resulting DAG topology of Figure 5a possesses the following features: (i) it is symmetric; (ii) all six input-output paths are equal-length; and (iii) MAP/REDUCER#1 (resp., MAP/REDUCER#2) is more computing-intensive (resp., more communication-intensive) than MAP/REDUCER#2 (resp., MAP/REDUCER#1).



(a) Graph of the considered DAG1.

Figure 5. Cont.



(b) Graph of the considered DAG2.

Figure 5. Graphs of the tested DAGS. (a) DAG1 and (b) DAG2. All the numeric labels are in Kbit. The total workload (e.g., summation of the node labels) and inter-node traffic (e.g., summation of the edge weights) of the test DAGs are the same. They equate to 3.32 (Mbit) and 1.66 (Mbit), respectively.

DAG2 in Figure 5b details the workflow of a video navigation application [33], which involves the parallel execution of three sub-programs, namely a graphic sub-program (left section of Figure 5b), a subprogram for face detection (middle section of Figure 5b), and a video-processing subprogram (right section of Figure 5b). All these sub-programs share the same input and output nodes (e.g., Nodes 1 and 15 in Figure 5b), which implement data-rendering functionalities and, then, are forced to be executed at the Mobile device. DAG2 is medium-sized (e.g., it is composed of 15 tasks and 21 edges), and its topology possesses the following features: (i) it is asymmetric; (ii) it is composed of the parallel combination of three heterogeneous sub-DAGs, which exhibit fork, parallel, and tree-shaped sub-topologies, respectively; and (iii) the face detection and video processing subprograms are computing- and communication-intensive, respectively, while the graphic subprogram is of a mixed type.

Lastly, DAG3 in Figure 6 describes the workflow for the simulation of Newton’s equations describing pseudo-chaotic molecular dynamics [29]. It is a large-sized DAG (e.g., it is composed of 41 tasks and 70 edges), and its topology exhibits the following main features: (i) it is very irregular and mimics a *random* graph, which combines a number of heterogeneous sub-graphs with chain, parallel, fork, tree, and mesh-type sub-topologies; (ii) it is composed of a large number of *pseudo-generated* edges, which join *randomly*-selected task pairs. As a consequence, in Figure 6, there are a number of inter-crossing edges, as well as a number of variable-length paths that go from the input task to the output one. (iii) Task workloads and edge weights have been randomly generated.

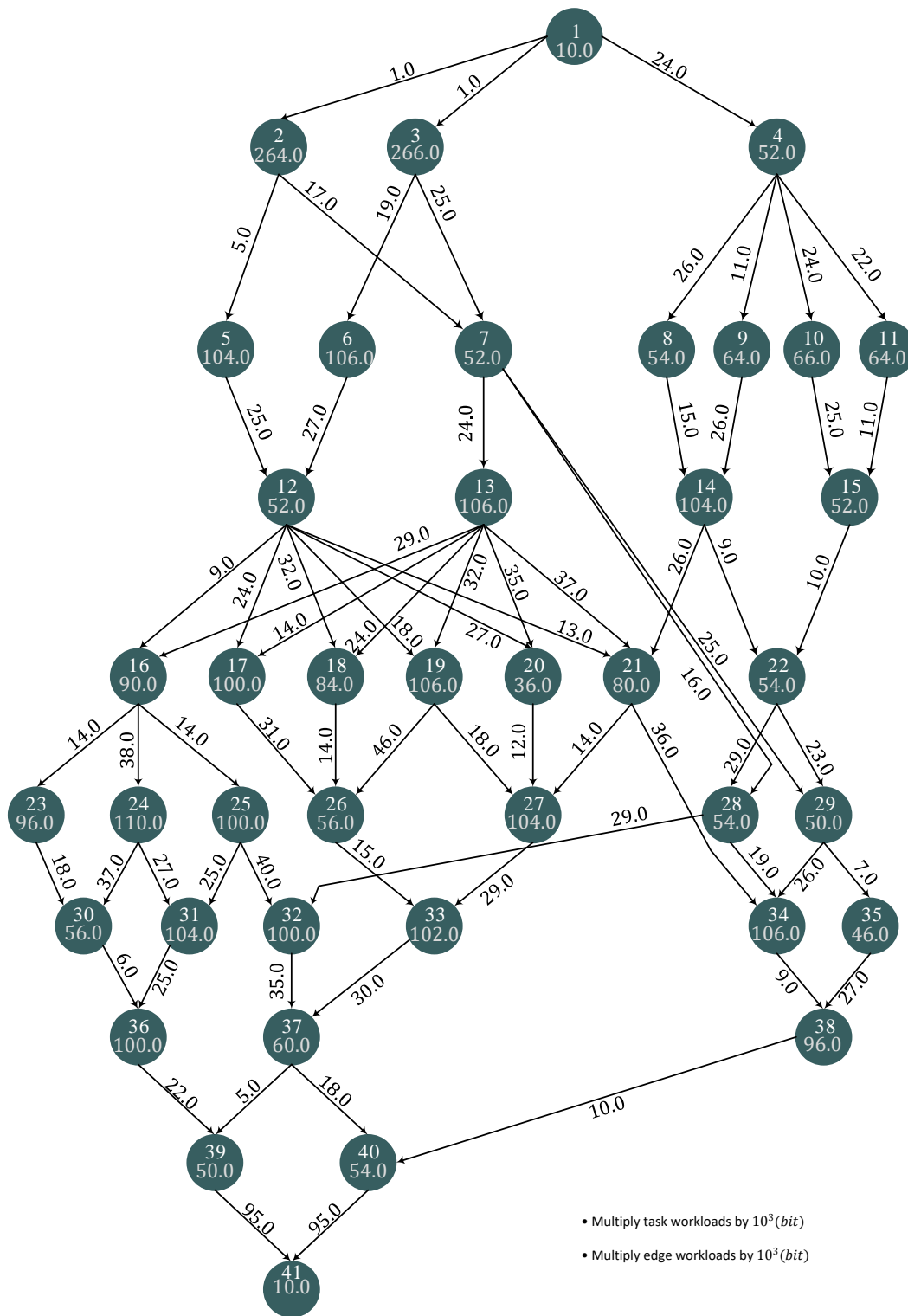


Figure 6. Graphs of the tested DAG3. All the numeric labels are in Kbit. The total workload (e.g., summation of the node labels) and inter-node traffic (e.g., summation of the edge weights) of the test DAGs are the same. They equate to 3.32 (Mbit) and 1.66 (Mbit), respectively.

Finally, we point out that the summations of task workloads and edge weights of all considered DAGs of Figures 5 and 6 are the same and equate to 3.32 (Mbit) and 1.66 (Mbit), respectively. In so doing, we expect that the results of the performance comparisons carried out in the next sub-sections are fair.

6.2. Comparative Tracking Performance under Intermittent WiFi Connectivity

The goal of this section is to test the convergence speed to the steady-state and the steady-state stability of the primal-dual iterations implemented by the RAP_p function of Section 4.4 when abrupt changes of the available WiFi up/down bandwidths and task allocation patterns simultaneously happen. For this purpose, we run the *FogTracker* function of Section 4.4 at $T_{DAG} = 0.3$ under the (previously-described) three test DAGs. The obtained dynamic behaviors of the total consumed energies \mathcal{E}_{TOT} , network energies \mathcal{E}_{NET} , and λ multipliers are reported in Figure 7a–c (resp., Figures 8a–c and 9a–c) for various test values of the speed-up factor a_{MAX} of Section 3.5 under DAG1 (resp., DAG2 and DAG3).

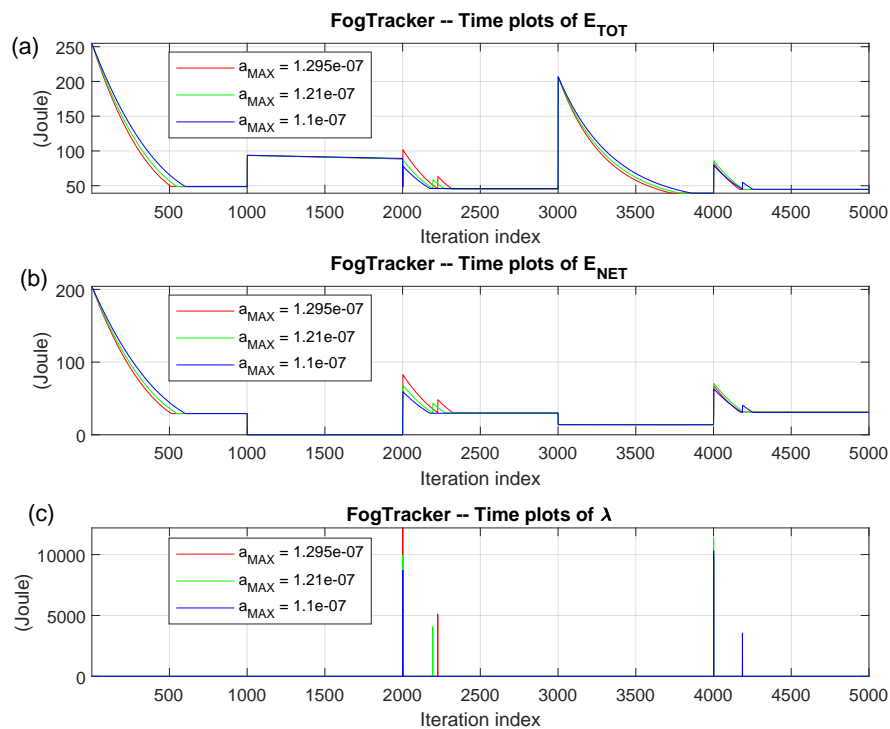


Figure 7. Tracking performance under DAG1 for three values of the speed-factor a_{MAX} . (a) time behaviors of the total energy \mathcal{E}_{TOT} ; (b) time behaviors of the corresponding network energy \mathcal{E}_{NET} ; and (c) time behaviors of the λ multiplier.

All the reported time behaviors refer to a common application scenario under which both task allocation patterns and up/down WiFi bandwidths undergo simultaneous abrupt changes at the iteration indexes $t = 1, 1000, 2000, 3000,$ and 4000 (see the corresponding step-like jumps of the plots of Figures 7–9). These changes are typically triggered by device mobility, which may give rise to an *intermittent* availability of the WiFi connectivity. More in detail, we have that: (i) at $t = 1$, the up/down cellular (resp., WiFi) bandwidths are turned ON (resp., turned OFF), and all tasks are allocated to the Cloud node; (ii) at $t = 1000$, the up/down WiFi bandwidths are turned ON, and all tasks are allocated to the Mobile node; (iii) at $t = 2000$, the WiFi bandwidths are still turned OFF, and all tasks are re-allocated to the Cloud node; (iv) at $t = 3000$, the up/down WiFi bandwidths are turned ON once time, and all tasks are allocated to the Fog node; and, finally, (v) at $t = 4000$, the up/down WiFi bandwidths are definitively turned OFF, and all tasks are re-migrated to the Cloud node. After each change of the setup environment, the RAP_p function runs, in order to re-allocate both the per-clone computing frequencies at the Mobile-Fog-Cloud nodes and the corresponding up/down Cellular-WiFi bandwidths properly.

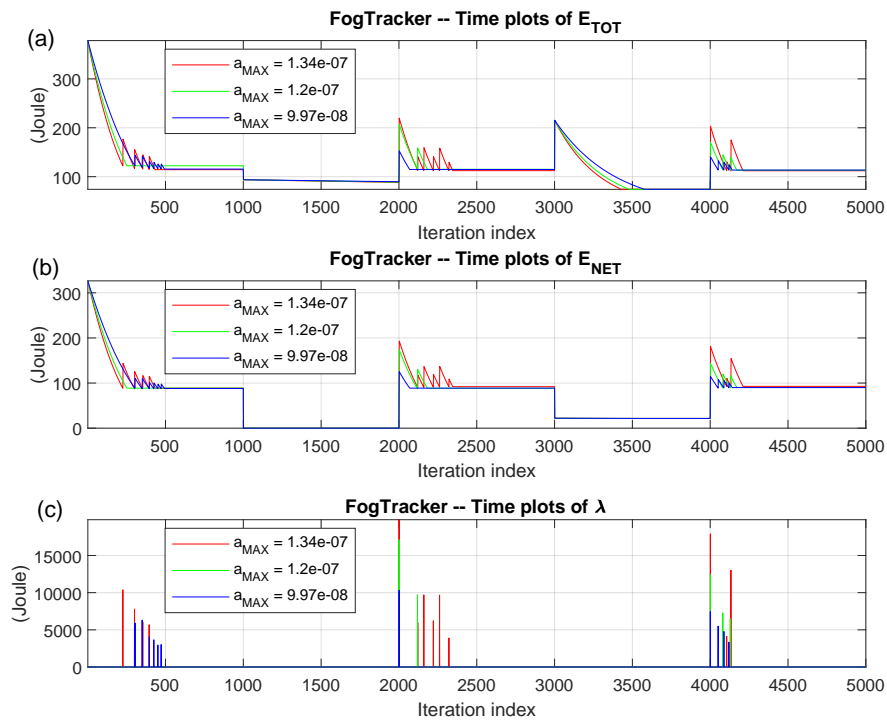


Figure 8. Tracking performance under DAG2 for three values of the speed-factor a_{MAX} . (a) time behaviors of the total energy \mathcal{E}_{TOT} ; (b) time behaviors of the corresponding network energy \mathcal{E}_{NET} ; and (c) time behaviors of the λ multiplier.

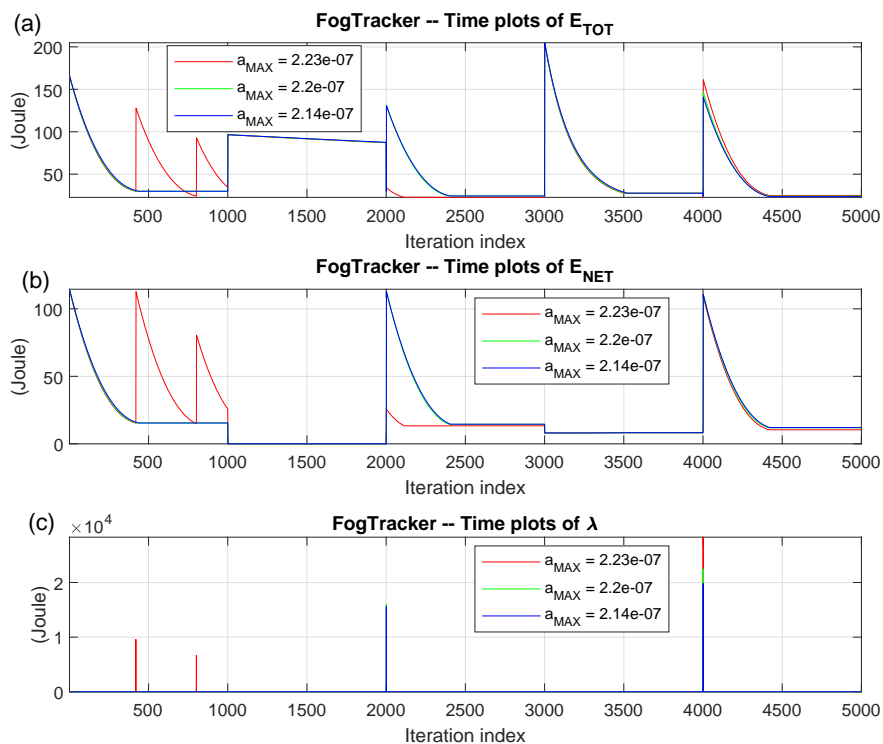


Figure 9. Tracking performance under DAG3 for three values of the speed-factor a_{MAX} . (a) time behaviors of the total energy \mathcal{E}_{TOT} ; (b) time behaviors of the corresponding network energy \mathcal{E}_{NET} ; and (c) time behaviors of the λ multiplier.

An examination of the time-plots reported in Figures 7–9 leads to three main insights. First, even in the presence of the (aforementioned) abrupt changes of the operating setups, the corresponding

λ multipliers remain almost surely vanishing (see Figures 7c, 8c, and 9c), and this supports the conclusion that all the performed resource allocations computed by the RAP_p function are, indeed, *feasible* (e.g., they meet the constraint in Equation (4) on the allowed maximum DAG execution time). Second, we have numerically ascertained that the abrupt step-like jumps experienced by the plots of \mathcal{E}_{NET} in Figures 7b, 8b, and 9b at the changing instants are the combined effects of *both* the ON-OFF availability of the WiFi connectivity and the associated re-allocation of the Cellular up/down bandwidths. Third, a comparative examination of the red-green-blue colored plots of Figures 7a, 8a, and 9a confirms that bigger values of the speed-up factor a_{MAX} of Section 3.5 speed-up the convergence to the corresponding steady-states, but also tend to introduce larger oscillation phenomena.

Overall, two final lessons are learned by the carried out tracking analysis. First, at least under the considered application scenarios, values of a_{MAX} ranging over the (quite broad) interval: 9.5×10^{-8} – 2.25×10^{-7} exhibit good tradeoffs among the contrasting requirements of quick reaction to abrupt (possibly mobility-induced) changes of the operative scenarios and stable behavior in the steady-state. Second, values of the primal-dual iterations I_{MAX} limited up to 500–700 suffice for attaining stable resource allocations, even in the presence of abrupt changes of the operation environment.

6.3. Comparative Task and Resource Allocation Performance

In this section, we compare the task placement, resource allocation, and energy performance of the (previously-described) *GeneticTA_par* strategy under the considered test DAGs. This is done for values of the allowed execution times T_{DAG} of 0.3, 0.6, 0.9, 1.5, and 3.0 (s). The final goal is to acquire insights about the effects of the DAG topologies, their sizes, and corresponding distributions of the task workloads and edge weights on the allocation patterns returned by *GeneticTA_par*.

The numerical results obtained by running the *VirtFogSim* toolbox are reported in Tables 6–8 under DAG1, DAG2, and DAG3, respectively.

A view of these results leads to the following four main sets of remarks.

Consumed total energies and allocated resources: A comparative examination of the profiled values reported in the 13th column of Tables 6–8 points out that, in all simulated cases, the total consumed energy \mathcal{E}_{TOT} remains limited up to 32 Joule. Since all the test DAGs share the same sum values of the task workloads and edge weights, this supports the conclusion that, at the first order, these factors play a major role in dictating the energy efficiency of the performed task and resource allocations. However, a more detailed examination of these results also unveils the following two trends: (i) at fixed DAG, the consumed energies tend to decrease for increasing values of T_{DAG} ; and (ii) at fixed T_{DAG} , the energy consumption tends to decrease by passing from the (more symmetric and regular) DAG1 to the (more random and irregular) DAG3 (see the DAG graphs of Figures 5 and 6). Roughly speaking, the first trend reflects the fact that larger values of T_{DAG} make the underlying application environment more delay-tolerant. As a consequence, the RAP_p function lowers the steady-state computing frequencies and/or the network bandwidths (see the numerical values reported by the corresponding columns of Tables 6–8). This reduces, in turn, the dynamic (e.g., resource-depending) components of the total consumed energies. The second trend confirms the fact that, by design, the optimization capability of genetic-based strategies generally increases with the size and/or the pseudo-random irregular nature of the topology of the underlying DAGs.

Table 6. Task, resource allocation, and energy consumption returned by *GeneticTA_par* under DAG1. Resources are measured in Mbit/s and energies in Joule.

T_{DAG}	Mobile Tasks	Fog Tasks	Cloud Tasks	f_M	f_F	f_C	R_U	R_D	B_U	B_D	BR	\mathcal{E}_{TOT}	\mathcal{E}_{NET}
0.3	{1, 12}	{3, 7 – 9, 11}	{2, 4 – 6, 10}	11.81	3.00	2.50	7.42	8.25	3.96	1.16	0.0	31.81	16.72
0.6	{1, 12}	{3, 7 – 9, 11}	{2, 4 – 6, 10}	11.75	2.38	2.47	6.97	7.61	2.31	0.95	0.0	28.76	13.70
0.9	{1, 3, 12}	{–}	{2, 4 – 11}	11.60	0.00	2.00	0.00	0.00	0.75	0.76	0.0	27.59	11.38
1.5	{1, 3, 12}	{2, 4 – 11}	{–}	11.50	1.82	0.00	6.75	7.15	0.00	0.00	0.0	26.11	9.48
3.0	{1, 2, 3, 12}	{4 – 11}	{–}	11.35	1.79	0.00	6.72	7.10	0.00	0.00	0.0	25.75	9.45

Table 7. Task, resource allocation, and energy consumption returned by *GeneticTA_par* under DAG2. Resources are measured in Mbit/s and energies in Joule.

T_{DAG}	Mobile Tasks	Fog Tasks	Cloud Tasks	f_M	f_F	f_C	R_U	R_D	B_U	B_D	BR	\mathcal{E}_{TOT}	\mathcal{E}_{NET}
0.3	{1, 7, 10 – 12, 14, 15}	{2, 3, 8, 13}	{4 – 6, 9}	11.90	2.47	2.48	7.50	8.51	0.95	6.64	0.0	27.09	8.45
0.6	{1, 7, 10 – 12, 14, 15}	{2, 3, 8, 13}	{4 – 6, 9}	11.80	2.35	2.37	7.35	8.39	0.93	6.52	0.0	26.78	8.35
0.9	{1, 7, 10 – 12, 14, 15}	{2 – 6, 8, 9, 13}	{–}	11.68	1.83	0.00	6.85	8.21	0.00	0.00	0.0	23.63	7.33
1.5	{1, 7, 10 – 12, 14, 15}	{2 – 6, 8, 9, 13}	{–}	11.58	1.73	0.00	6.65	8.11	0.00	0.00	0.0	22.81	7.30
3.0	{1 – 3, 7, 8, 10 – 12, 14, 15}	{4 – 6, 9}	{–}	10.98	1.64	0.00	6.55	8.01	0.00	0.00	0.0	22.50	7.22

Table 8. Task, resource allocation, and energy consumption returned by *GeneticTA_par* under DAG3. Resources are measured in Mbit/s and energies in Joule.

T_{DAG}	Mobile Tasks	Fog Tasks	Cloud Tasks	f_M	f_F	f_C	R_U	R_D	B_U	B_D	BR	\mathcal{E}_{TOT}	\mathcal{E}_{NET}
0.3	{1, 41}	{39, 40}	{2 – 38}	11.92	4.37	2.74	0.00	6.27	0.95	0.00	3.70	18.37	6.16
0.6	{1, 41}	{39, 40}	{2 – 38}	11.90	4.35	2.48	0.00	6.26	0.94	0.00	3.70	17.85	6.02
0.9	{1, 41}	{38 – 40}	{2 – 37}	11.77	4.25	2.47	0.00	6.25	0.95	0.00	3.70	17.60	6.01
1.5	{1, 41}	{35 – 40}	{2 – 34}	11.67	5.01	2.45	0.00	6.23	0.92	0.00	3.70	17.17	5.81
3.0	{1, 41}	{34 – 40}	{2 – 33}	11.50	4.05	2.37	0.00	6.09	0.90	0.00	3.70	17.17	5.78

Consumed network energies: The impact of the DAG topology and distribution of the edge weights on the resulting consumed network energy \mathcal{E}_{NET} seems to be, indeed, more relevant. In fact, a comparative examination of the results reported in the last column of Tables 6–8 points out that: (i) under DAG1, the percent ratios of the network energies to the corresponding total ones quickly decrease for increasing values of T_{DAG} and pass from 52% at $T_{DAG} = 0.3$ s to: 36.5% at $T_{DAG} = 3.0$ s; and (ii) under DAG2 and DAG3, the corresponding network-to-total energy ratios are not so sensitive to the values of T_{DAG} . They remain clipped, indeed, around: 31–32%, and around: 33.5–33.7% under DAG2 and DAG3, respectively. This supports the conclusions that: (i) DAG1 features a communication-intensive application, especially at low values of T_{DAG} ; while (ii) DAG2 and DAG3 describe more computing-intensive applications.

Task allocation patterns: Columns 2, 3, and 4 of Tables 6–8 report the Identification Numbers (IDs) of the tasks allocated by *GeneticTA_par* to the Mobile, Fog, and Cloud nodes under DAG1, DAG2, and DAG3, respectively. Although the reported allocation patterns may strongly depend on the specifically considered DAGs, nevertheless, three main quite general trends seem to stand out. First, at low values of T_{DAG} , medium-sized and communication-intensive tasks are typically allocated at the Fog node, while large-sized and communication-light tasks are allotted at the Cloud node. The Mobile device only executes small-sized, but communication-intensive tasks. Second, more and more tasks are shifted from the Cloud node to the Fog node and/or to the Mobile device for decreasing values of T_{DAG} .

Utilization of the Fog-Cloud backhaul connection: A peculiar feature of the three-tiered technological platform of Figure 1 is the presence of a (possibly multi-hop and/or wired) two-way backhaul connection, which interconnects the Fog and Cloud nodes. Hence, it may be of interest to attain insights about its utilization for the support of the (previously-mentioned) task allocation patterns performed by *GeneticTA_par*. Intuitively, we expect that the backhaul connection is used when there are large-sized tasks to be allocated at the Cloud and the volumes of data output by the execution of these tasks are large. Therefore, since the Fog-to-Mobile WiFi connection is more energy efficient than the corresponding Cloud-to-Mobile cellular one, it may be energy-saving to transport the processed data from the Cloud to the Fog over the backhaul connection and, then, use the Fog as a *relay* node, in order to forward these data to the Mobile device by exploiting the WiFi downlink of Figure 1. This is, indeed, the strategy dictated by *GeneticTA_par*, in order to execute DAG3. In fact, an examination of the corresponding Table 8 points out that the winner strategy returned by *GeneticTA_par* utilizes: (i) the cellular bandwidth B_U for the upload of the data to be processed by the Cloud; (ii) the backhaul bandwidth BR for forwarding the processed data to the Fog node; and (iii) the WiFi bandwidth R_D for the final download of the processed data to the Mobile device.

The final conclusion is that, in the considered operating scenario, the utilization of (single-hop) Cloud-to-Mobile and/or Mobile-to-Fog links is *less* energy efficient than the exploitation of the (multi-hop) Cloud-Fog-Mobile path.

6.4. The Performance Impact of Different Task and Resource Allocation Strategies

In this section, we compare the energy performances of the (previously-described) task and resource allocation strategies supported by the *VirtFogSim* toolbox under the considered test DAGs. The three-fold final goal is to acquire some insight about: (i) the energy reduction stemming from the dynamic optimization of the computing-networking resources versus the corresponding case of *static* resource usage; (ii) the performance gap between the genetic-based and the Exhaustive Search-based allocation strategies; and (iii) the energy-saving capability offered by the Mobile-Fog-Cloud *three-tier* computing platform of Figure 1 versus the corresponding Mobile only, Mobile-Cloud, and Mobile-Fog ones.

The profiled results are reported in Figures 10–12 under DAG1, DAG2, and DAG3, respectively. Their examination gives rise to the following three main sets of remarks.

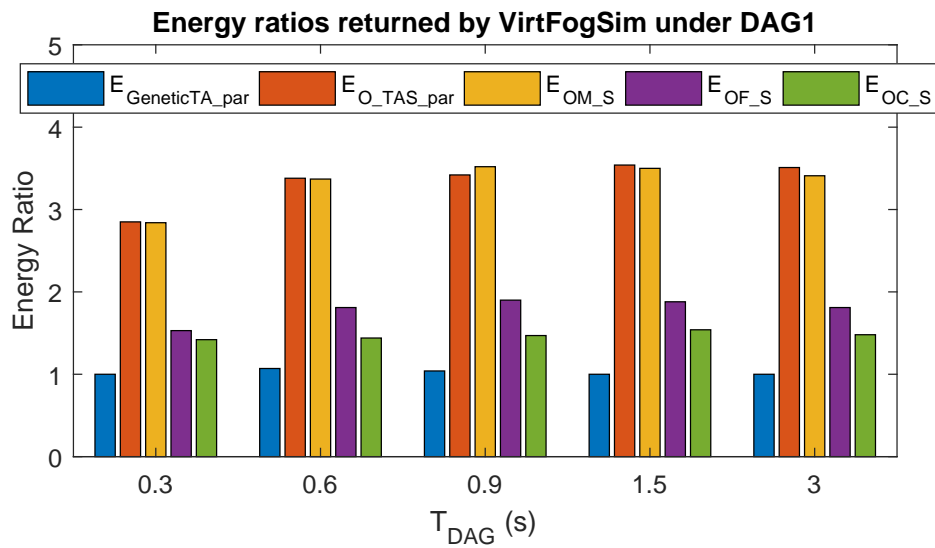


Figure 10. Energy ratios returned by VirtFogSim under DAG1 for $T_{DAG} = 0.3$ s, 0.6 s, 0.9 s, 1.2 s, and 1.5 s. All the reported ratio values are normalized with respect to the corresponding total energies consumed by ES_S_par .

Dynamic-vs.-static resource allocation: A number of benchmark (even quite recent) contributions [26,30,31,33] afford the problem of the resource augmentation of mobile devices by developing various heuristic/meta-heuristic/optimal solutions for energy-efficient task offloading. However, they neglect considering, indeed, the companion problem of dynamic scaling of the computing and/or network resources. Hence, a key (still pending) question is: how much energy may be actually saved by jointly performing task and dynamic resource allocation? In this regard, we point out that both the $GeneticTA_par$ and O_TAS_par strategies implemented by the $VirtFogSim$ toolbox rely on the same genetic-based meta-heuristic for performing task placement. However, $GeneticTA_par$ also performs dynamic resource allocation by invoking the auxiliary RAP_p function, while O_TAS_par does not scale up/down the involved computing frequencies and wireless bandwidths of Figure 1 and clips them at their corresponding maximum values (see Section 4.2). Hence, a direct comparison of the energies consumed by $GeneticTA_par$ and O_TAS_par allows us to profile how much energy may be saved by performing dynamic resource allocation.

In this regard, an examination of the numerical results reported in the first and second bars of Figures 10–12 leads to three main insights. First, the energy ratio: $\mathcal{E}_{O_TAS_par} / \mathcal{E}_{GeneticTA_par}$ ranges over the intervals: 2.85–3.50, 3.46–4.10, and: 4.50–4.56 under DAG1, DAG2, and DAG3, respectively. Second, at fixed DAG, the energy savings stemming from performing dynamic resource allocation reaches their maxima at values of T_{DAG} of the order of 0.6–0.9 s, while tending to somewhat decrease at smaller and higher execution delays. Third, the average energy-saving stemming from dynamic optimization is more relevant under DAG3.

Overall, the performed analysis unveils that the dynamic optimization of the allocated computing-networking resources plays, indeed, a *pivotal* role in reducing the energy consumption of the overall technological platform of Figure 1.

Genetic-vs.-Exhaustive Search performance comparison: The focus of this subsection is on the tradeoffs among energy performance and execution complexity that are attained by the (meta-heuristic) $GeneticTA_par$ and the (optimal) exhaustive E_ESS_par strategies. In this regard, we point out that all the simulations of the $GeneticTA_par$ function are carried out at population size $PS = 120$ and per-population number of generations $G_{MAX} = 100$. In so doing, the resulting computational complexity of the run $GeneticTA_par$ code is of the order of (see Table 4) $\mathcal{O}(120 \times 100)$ under all test DAGs. At the same time, since the implementation complexity of the E_ESS_par function scales up exponentially as $\mathcal{O}(3^{(V-2)})$ with the size V of the considered DAG, the execution of E_ESS_par

under DAG3 is out of discussion, while the resulting execution complexities are of the order of $\mathcal{O}(3^{10})$ and $\mathcal{O}(3^{13})$ under DAG1 and DAG2, respectively. As a consequence, the (profiled) execution times of E_{ESS_par} are about five- and 133-times larger than the corresponding ones of $GeneticTA_par$.

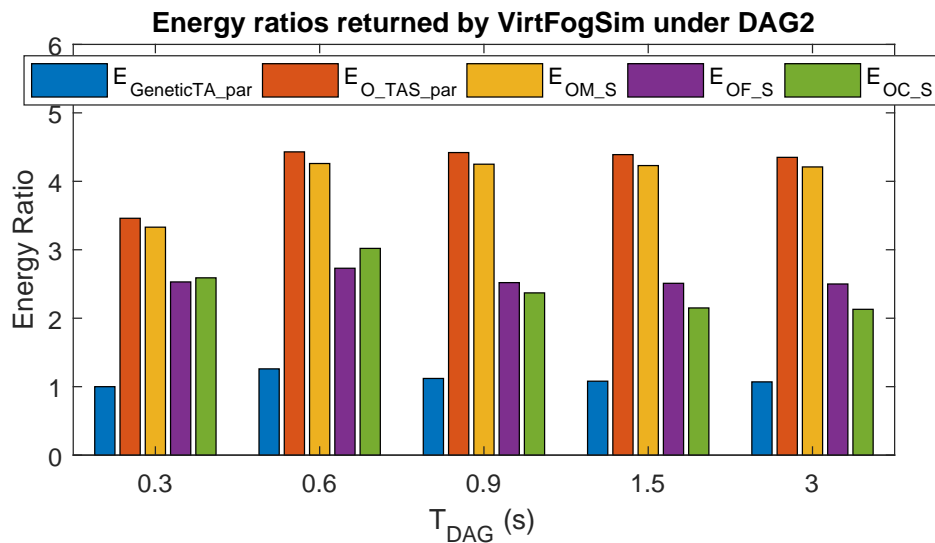


Figure 11. Energy ratios returned by VirtFogSim under DAG2 for $T_{DAG} = 0.3$ s, 0.6 s, 0.9 s, 1.2 s, and 1.5 s. All the reported ratio values are normalized with respect to the corresponding total energies consumed by ES_S_par .

Regarding the analysis of the first bars of Figures 10 and 11, the following three main insights may be drawn. First, under the (small-sized) DAG1, $GeneticTA_par$ and E_{ESS_par} return the same optimal task and resource allocation patterns at $T_{DAG} = 0.3, 1.5, 3.0$ s. Furthermore, the energy ratio: $E_{GeneticTA_par} / E_{E_{ESS_par}}$ is limited up to 1.07 (e.g., 7%) in the remaining two cases. Second, under the (medium-sized) DAG2, $GeneticTA_par$ and E_{ESS_par} share the same energy consumption at $T_{DAG} = 0.3$, while the corresponding energy ratio: $E_{GeneticTA_par} / E_{E_{ESS_par}}$ is no larger than 1.26 in the other cases.

Overall, the carried out analysis supports the conclusion that the tested implementation of $GeneticTA_par$ is capable of attaining, indeed, good performance-vs.-complexity tradeoffs.

Three tiered-vs.-single and two-tiered execution platforms: A potential drawback of multi-tiered distributed computing platforms is that the number of involved communication links tends to grow with the number of inter-connected tiers, and this may lead, in turn, to an increment of the network component of the overall consumed energy. Hence, the goal of this subsection is to give insight into the following (rather basic) question: What is the net tradeoff among the reduction of the computing energy arising from the utilization of multi-tiered computing nodes and the corresponding increment of the network energy needed for their inter-connection? In order to address this question, the *VirtFogSim* toolbox makes available the OM_S , OF_S , and OC_S strategies. By design, they utilize only the Mobile device, the two-tiered Fog-Mobile platform, and the two-tier Cloud-Mobile platform for the execution of the application DAGs. Furthermore, all these strategies perform dynamic scaling of the utilized computing frequencies and wireless network bandwidths (see Section 4.2).

Hence, since $GeneticTA_par$ exploits, by design, all the Mobile, Fog, and Cloud nodes of Figure 1 for task placement, a comparative analysis of the energy ratios in the last three bars of Figures 10–12 provides a direct response to the above question by providing the following three main insights.

First, the energy ratio: $E_{OM_S} / E_{GeneticTA_par}$ takes values over the intervals: 2.84–3.84, 3.3–3.9, and 4.8–5.1, under DAG1, DAG2, and DAG3, respectively. The corresponding intervals of the energy ratios: $E_{OF_S} / E_{GeneticTA_par}$ and $E_{OC_S} / E_{GeneticTA_par}$ are: 1.53–1.88, 2.0–2.73, 1.23–1.32, and 1.35–1.52, 1.99–2.59, 1.35–1.63, respectively. Hence, in the carried out simulations, the minimum (e.g., worst-case)

energy-savings guaranteed by the three-tier Mobile-Fog-Cloud platform over the Mobile, Mobile-Fog, and Mobile-Cloud ones are: 184%, 23% and 35%, respectively.

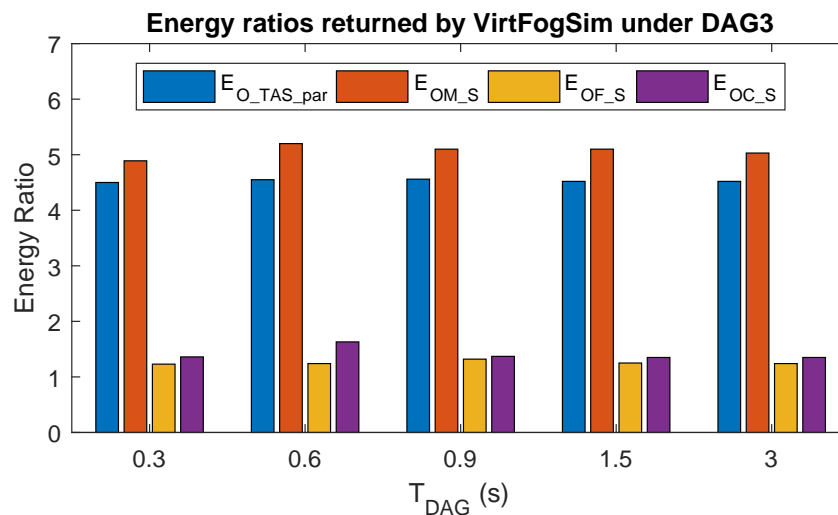


Figure 12. Energy ratios returned by VirtFogSim under DAG3 for $T_{DAG} = 0.3$ s, 0.6 s, 0.9 s, 1.2 s, and 1.5 s. All the reported ratio values are normalized with respect to the corresponding total energies consumed by *GeneticTA_par*.

Second, at fixed T_{DAG} , the average energy savings offered by the Mobile-Fog-Cloud platform over the considered benchmark ones tend to be somewhat more substantial under DAG2. Roughly speaking, this is due to the fact that DAG2 is the parallel combination of three sub-DAGs, whose energy-saving executions “naturally” lead to the simultaneous utilization of all the available Mobile, Fog, and Cloud computing nodes (see Figure 5b).

Third, regarding the comparison of the relative energy performances of the *OM_S*, *OF_S*, and *OC_S* strategies, we may conclude that: (i) the *OM_S* strategy is the most energy-consuming under all the tested cases; and (ii) in general, the *OC_S* (resp., *OF_S*) strategy is more energy saving than the *OF_S* (resp., *OC_S*) one under the more regular (resp., more random) DAG1 and DAG2 (resp. DAG3).

As a concluding remark, we point out that the reported numerical results give practical evidence of the support offered by the proposed *VirtFogSim* toolbox in the analysis and optimization of the multiple performance aspects of the multi-tiered technological platform of Figure 1.

6.5. Scalability of the Simulation Time of the VirtFogSim Toolbox

The two-fold goal of this section is to test the scalability of the execution times of the *VirtFogSim* toolbox versus both the number of cores available for its execution and the computational complexity of the run task allocation strategies. For this purpose, DAG1 at $T_{DAG} = 0.3$ has been considered, and then, both the execution times and the total input/output volumes (in MByte) of data transferred to/from the available pool of working cores have been numerically profiled through the *tictoc* and *ticByte–tocByte* commands available in the *Parallel Toolbox* of MATLAB. The attained results are reported by the bar-plots of Figures 13 and 14. Each reported result has been obtained by averaging over 10 independent runs of the (complexity-tunable) *GeneticTA_par* task allocation strategy.

An examination of the execution times of Figure 13 points out that the speed-up factors attained at $n_{core} = 2, 4, 6, 8,$ and 10 over the benchmark case of $n_{core} = 1$ are around 1.91, 3.51, 4.98, 5.94, and 6.60, respectively. Hence, these speed-factors scale in a (quasi-) *linear* way for values of n_{core} limited up to 3–4, while the scaling becomes more and more *sub-linear* for larger values of n_{core} . The reason behind this scaling behavior of the execution times is provided by a comparative examination of the corresponding total input/output data traffic generated by the carried out parallel executions. In this regard, the traffic bars of Figure 13 unveil that both the generated volumes of data (which is the most)

and the corresponding growing rate increase for increasing values of n_{core} . The resulting net effect is that the experienced execution times become more and more dominated by the corresponding inter-core communication times, so that the corresponding relative decrements in the execution times tend to decrease for large values of n_{core} .

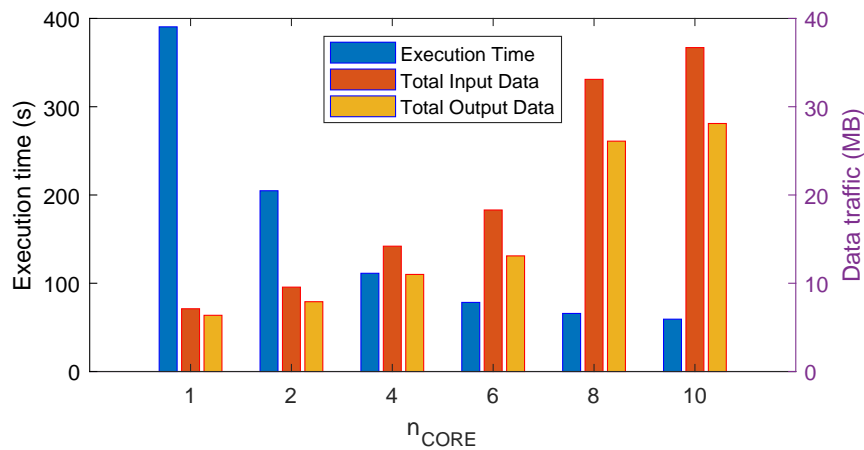


Figure 13. Bar-plots profiling the simulation times and corresponding input/output data traffics generated by parallel executions of *GeneticTA_par* for increasing values of the number n_{core} of the employed cores under DAG1 at $PS = 300$. The case: $n_{core} = 1$ corresponds to the sequential execution of *GeneticTA_par*.

This conclusion is further corroborated by the bar plots of Figure 14. They report the profiled execution times and generated inter-core traffic volumes obtained by running *GeneticTA_par* for increasing values of the population size PS at fixed $n_{core} = 10$. In fact, since the computational complexity of *GeneticTA_par* scales in a linear way with PS (see Table 4), it could be expected that the same scaling behavior would be also exhibited by the corresponding profiled execution times. However, an examination of the execution-time bars of Figure 14 unveils that the slow-down factors attained at $PS = 100, 150, 200, 250,$ and 300 against the benchmark case of $PS = 50$ do not scale, indeed, linearly, and are around 1.76, 2.38, 3.16, 3.40, and 4.53, respectively. Even in this case, the reason is the nonlinear growing behaviors of the corresponding input/output volumes of inter-core data (see the bars of data traffic in Figure 14).

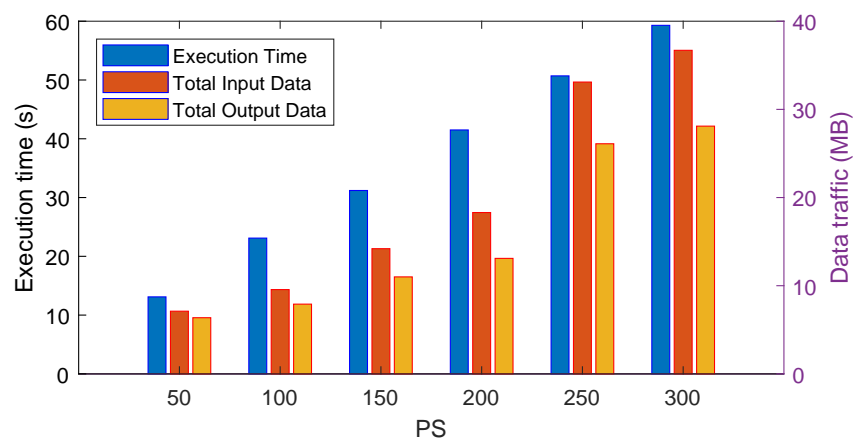


Figure 14. Bar-plots profiling the simulation times and corresponding input/output data traffics generated by parallel executions of *GeneticTA_par* for increasing values of the population size PS . The case of DAG1 at $n_{core} = 10$.

In this regard, we point out that similar trends are also exhibited by the (profiled) execution times of the other parallel functions: *ES_S_par* and *O_TAS_par* implemented by the *VirtFogSim* package.

To conclude this section, we underline that the RAM used by *VirtFogSim* is essentially that allocated by the MATLAB environment. Additional memory space is used to store all the variables involved by the chosen strategies. This additional memory space depends mainly on the DAG size V and the population size PS , but anyway, it is limited up to the interval 50–150 MB for the tested cases.

7. Conclusions and Future Developments

The actual development of delay-sensitive 5G-supported mobile applications demands for the dynamic profiling and energy optimization of emerging multi-tiered Fog-Cloud virtualized ecosystems. Since field-trials and deployment of test-beds are expensive and could not guarantee repeatable results, the development of customizable simulation toolboxes is welcome. *VirtFogSim* is compliant with this expectation by developing a new software environment that accounts for the main system parameters featuring the computing and network aspects of Mobile-Fog-Cloud technological platforms. The core engine of the *VirtFogSim* toolbox allows the optimization, simulation, and tracking of a number of heuristic/meta-heuristic/exhaustive search-based policies for the energy-saving dynamic allocation of tasks and computing-networking resources needed for the delay-constrained execution of mobile applications described by general DAGs. The GUI equipping the *VirtFogSim* package allows a user-friendly rendering of the simulated data under a number of easy-to-understand graphic formats.

The current version of the *VirtFogSim* package being the first open-access release, it is amenable to further extensions along three main directions.

According to the underlying 5G paradigm, additional customizable primitives for the profiling and performance simulation of massive MIMO wireless access technologies could provide a first valuable extension [35–37]. Furthermore, the current version of the simulator relies on purely reactive formal methods for the dynamic optimization of the needed computing-plus-networking resources. Including pro-active optimization tools that are capable of exploiting mobility-triggered resource forecasting could be second extension of potential interest [38]. Finally, software functionalities for the support and performance evaluation of (possibly, mobility and/or failure-triggered) energy and delay-efficient task migration strategies could represent a third extension of practical interest.

8. Availability of the VirtFogSim Package

The complete software package of the *VirtFogSim* simulator and the corresponding User Guide may be downloaded for free on the GitHub repository site at: <https://github.com/mscarpiniti/VirtFogSim>. In addition, it can be downloaded from the authors' web pages, specifically at: <http://enzobaccarelli.site.uniroma1.it>, by accessing the section: Downloadable packages.

Author Contributions: The authors contributed equally to this work.

Funding: This work has been supported by the project: “GAUChO—A Green Adaptive Fog Computing and networking Architectures” funded by the MIUR Progetti di Ricerca di Rilevante Interesse Nazionale (PRIN) Bando 2015, Grant 2015YPXH4W_004, and by the projects: “Vehicular Fog energy-efficient QoS mining and dissemination of multimedia Big Data streams (V-Fog and V-Fog2)” and “SoFT: Fog of Social IoT”, funded by Sapienza University of Rome Bando 2016, 2017, and 2018.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following main abbreviations are used in this manuscript:

CC	Cloud Computing
CNT	Container
ES	Exhaustive Search
DAG	Direct Acyclic Graph
FC	Fog Computing

GUI	Graphical User Interface
LAN	Local Area Network
NIC	Network Interface Card
OC	Only Cloud
OF	Only Fog
OM	Only Mobile
OTA	Only Task Allocation
VM	Virtual Machine
WAN	Wide Area Network

Appendix A. VirtFogSim: Supported Dual-Mode User Interfaces

The current version of the simulator supports two user interfaces, *VirtFogSim* and *VirtFogSim Graphic User Interface (VirtFogSimGUI)*, respectively. As detailed in the following two sub-sections, both interfaces make available the same set of basic optimization routines of Table 4, and then, they provide the same set of numerical results. However,

1. the *VirtFogSim* interface is oriented to a scientific use of the simulator. Its utilization requires some basics about the MATLAB environment. Hence, it may be appealing for skilled research users, who desire to work in an interactive way and are mainly interested in checking and optimizing the performance of their own customized DAGs under (possibly multiple) customized simulation setups;
2. the *VirtFogSimGUI* interface provides a rich set of self-explanatory, ready-to-use native facilities that allow less (or even un-) skilled users to directly run the simulator under a number of pre-loaded (but, in any case, customizable) application scenarios. Hence, since its utilization does not require any specific skill regarding the MATLAB environment, it allows the user to interact with the simulator as a “black-box”. For this purpose: (i) the obtained numerical results are rendered in the form of graphics, colored maps, and plots, in order to make their interpretation and mining more intuitive; and (ii) an easy-to-consult on-line version of this User Guide is also enclosed.

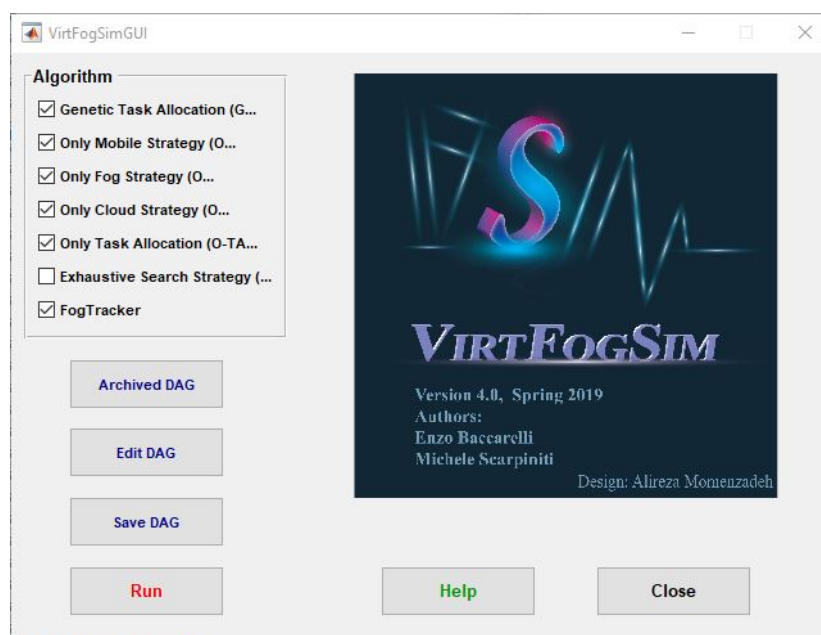


Figure A1. A screen-shoot of the GUI interface.

Appendix A.1. The VirtFogSim Interface

This interface is activated by entering the command: `VirtFogSim` in the command line of a running MATLAB session. The script acts as the main program of the *VirtFogSim* package. This script allows the user to select any subset of the natively-supported optimization algorithms by setting some binary flags to one (for running it) or to zero (for not running it) and to choose which DAG should be analyzed by writing the name of the related configuration script. If the user desires to change the configuration, the script defining the DAG can be opened in the editor window of MATLAB.

By programming in the MATLAB language, this allows the user to:

1. edit the desired DAG by setting the corresponding workload vector \vec{s} , adjacency matrix A , and edge weight matrix D ;
2. define the desired simulation setup by editing the input parameters listed in Table A2;
3. select any subset of the supported optimization functions of Table 4;
4. resort to the on-line *help* of MATLAB, in order to display information about any specific function supported by the *VirtFogSim* package; and,
5. write and call user-defined customized functions that are not natively supported by the current version of the simulator.

Depending on the selected functions (see Table 4), after running *VirtFogSim*, the attained numerical results are displayed in tabular form, and/or as colored time-plots, and/or as colored bar plots, and/or as colored DAG maps (see Section 5 for additional details about the graphic rendering capabilities and related options supported by the simulator).

Appendix A.2. The VirtFogSimGUI Interface

The GUI interface is open by entering the command: `VirtFogSimGUI` in the command line of a running MATLAB session. The screen-shoot of the displayed graphic window is reported in Figure A1.

This interface is supported by the MATLAB code: *VirtFogSim_Run*, which acts as the main script of the overall simulator.

An examination of Figure A1 points out that the GUI interface of the simulator supports seven pre-built functions (namely, *Help*, *Algorithm*, *Archived DAG*, *Edit DAG*, *Save DAG*, *Run*, and *Close*), which may be activated by the user by clicking on the corresponding buttons. Table A1 lists these native GUI functions and points out their meaning and associated actions.

Table A1. A synoptic overview of the functionalities offered to the user by the GUI interface.

Available GUI Functions	Associated Actions
<i>Help</i>	Allows access to the User Guide of the simulator by opening a dedicated PDF file.
<i>Algorithm</i>	Allows selecting any subset of the natively-supported optimization algorithms by clicking the corresponding labels.
<i>Archived DAG</i>	Allows retrieving an already archived DAG with the corresponding simulation setup, in order to run it.
<i>Edit DAG</i>	Allows editing a new DAG and/or a new simulation setup by compiling the list of input parameters of Table A2.
<i>Save DAG</i>	Allows saving the last edit of DAG and assigning it an identification label.
<i>Run</i>	Allows running the selected optimization algorithm under the retrieved/edit DAG and associated simulation setup.
<i>Close</i>	Shuts down the current working session of the <i>VirtFogSim</i> simulator and closes all the figure windows.

From the user perspective, a typical *VirtFogGUI*-enabled working session should proceed according to the following ordered list of steps:

1. enter *VirtFogGUI* at the MATLAB prompt, in order to open the GUI interface;
2. click the *Help* button, in order to access the PDF version of the User Guide. This step is optional;
3. select any desired subset of the supported optimization procedures by clicking the corresponding labels in the *Algorithm* window. This step is mandatory;
4. retrieve an already stored DAG in *.mat format and the associated list of input parameters by accessing the *Archived DAG*. Alternatively, build a new DAG and set the desired input parameters of Table A by clicking the *Edit DAG* button. This step is mandatory;
5. store in the *Archived DAG* database the last edit of DAG in *.mat format by clicking the *Save DAG* button. An identification label for the *Save DAG* is required. This step is optional;
6. click the *Run* button, in order to start the execution of the selected algorithms under the (retrieved or edit) selected DAG. At the end of the execution, the attained results are returned in tabular form, and/or as colored time-plots, and/or as colored bar plots, and/or as colored DAG maps (see Section 5 for additional details about the graphic rendering capabilities and related options supported by the simulator);
7. after finishing the current working session of the simulator, click the *Close* button, in order to shut down the *VirtFogSimGUI* interface.

Appendix B. Full List of the Input Parameters of VirtFogSim

The following Table A2 reports the full list of the (settable) input parameters of the current Version 4.0 of the simulator, together with their meaning/role, measuring units, and default values used for the simulation of Section 6.

Table A2. Input parameters of the *VirtFogSim* simulator and their simulated settings of Section 6.

Parameter	Meaning/Role	Measuring Units	Simulated Settings
V	Number of tasks of the application DAG	Dimensionless	$9 \leq V \leq 45$
\vec{s}	V -tuple row vector of the task workloads	(bit)	Computing- and communication-intensive DAG
A	$(V \times V)$ binary (i.e., $\{0, 1\}$) DAG adjacency matrix	Dimensionless	
Da	$(V \times V)$ weight matrix of the (directed) edges of the DAG	(bit)	Computing- and communication-intensive DAG
n_M	Number of the (virtual) computing cores equipping the Mobile device	Dimensionless	$n_M = 1$
n_F	Number of the (virtual) computing cores equipping the device clone at the Fog node	Dimensionless	$n_F = 4$
n_C	Number of (virtual) computing cores equipping the device clone at the Cloud node	Dimensionless	$n_C = 12$
NF_{WiFi}	Average per-connection number of failures of the one-way Mobile→Fog and Fog→Mobile WiFi TCP/IP connections	Dimensionless	$NF_{WiFi} = 1.1$
NF_{CELL}	Average per-connection number of failures of the cellular one-way Mobile→Cloud and Cloud→Mobile TCP/IP connections	Dimensionless	$NF_{CELL} = 0.1$

Table A2. Cont.

Parameter	Meaning/Role	Measuring Units	Simulated Settings
N_{FWD}	Average per-connection number of failures of the two-way Cloud↔Fog TCP/IP backhaul connection	Dimensionless	$N_{FWD} = 0.01$
f_M^{MAX}	Per-core maximum computing frequency at the Mobile device	(bit/s)	$f_M^{MAX} = 12 \times 10^6$
f_F^{MAX}	Per-core maximum computing frequency at the Fog clone	(bit/s)	$f_F^{MAX} = 12 \times 10^6$
f_C^{MAX}	Per-core maximum computing frequency at the Cloud clone	(bit/s)	$f_C^{MAX} = 12 \times 10^6$
R_U^{MAX}	Maximum bit rate of the WiFi-based one-way Mobile→Fog TCP/IP connection	(bit/s)	$R_U^{MAX} = 8.0 \times 10^6$
R_D^{MAX}	Maximum bit rate of the WiFi-based one-way Fog→Mobile TCP/IP connection	(bit/s)	$R_D^{MAX} = 9.0 \times 10^6$
B_U^{MAX}	Maximum bit rate of the cellular one-way Mobile→Cloud TCP/IP connection	(bit/s)	$B_U^{MAX} = 6.5 \times 10^6$
B_D^{MAX}	Maximum bit rate of the cellular one-way Cloud→Mobile TCP/IP connection	(bit/s)	$B_D^{MAX} = 7.0 \times 10^6$
TH_0^{MIN}	Scalar non-negative real parameter. It is the minimum required application throughput	(DAG/s)	$0.333 \leq TH_0^{MIN} \leq 3.33$
θ_M	Binary $\{0, 1\}$ -parameter. It depends on the utilized application service model at the Mobile device	Dimensionless	$\theta_M = 0$ (resp., $\theta_M = 1$) if computing-plus-networking energy consumed by the Mobile device is (resp., is not) for free
θ_F	Binary $\{0, 1\}$ -parameter. It depends on the utilized application service model at the Fog node	Dimensionless	$\theta_F = 0$ (resp., $\theta_M = 1$) if the computing-plus-networking energy consumed by the Fog clone is (resp., is not) for free
θ_C	Binary $\{0, 1\}$ -parameter. It depends on the utilized application service model at the Cloud node	Dimensionless	$\theta_C = 0$ (resp., $\theta_C = 1$) if computing-plus-networking energy consumed by the Cloud clone is (resp., is not) for free
$P_{CPU-M}^{(IDLE)}$	Power consumed in the idle state by the physical CPU at the Mobile device	(Watt)	$P_{CPU-M}^{(IDLE)} = 1.2$
$P_{CPU-F}^{(IDLE)}$	Power consumed by a single physical server in the idle state at the Fog node	(Watt)	$P_{CPU-F}^{(IDLE)} = 220$
$P_{CPU-C}^{(IDLE)}$	Power consumed by a single physical server in the idle state at the Cloud node	(Watt)	$P_{CPU-C}^{(IDLE)} = 440$
nc_M	Number of containers simultaneously running on the Mobile CPU	Dimensionless	$nc_M = 1$

Table A2. Cont.

Parameter	Meaning/Role	Measuring Units	Simulated Settings
nc_F	Number of containers simultaneously running on a single physical server at the Fog node	Dimensionless	$nc_F = 16$
nc_C	Number of containers simultaneously running on a single physical server at the Cloud node	Dimensionless	$nc_C = 24$
γ_M	Positive exponent of the dynamic power consumption of the CPU at the Mobile device	Dimensionless	$\gamma_M = 3.2$
γ_F	Positive exponent of the dynamic power consumption of a single physical server at the Fog node	Dimensionless	$\gamma_F = 3.1$
γ_C	Positive exponent of the dynamic power consumption of a single physical server at the Cloud node	Dimensionless	$\gamma_C = 3.0$
k_M	Positive parameter. It profiles the dynamic power consumption of the CPU at the Mobile device	(Watt)/(bit/s) $^{\gamma_M}$	$k_M = 7.5 \times 10^{-21}$
k_F	Positive parameter. It profiles the dynamic power consumption of a single physical server at the Fog node	(Watt)/(bit/s) $^{\gamma_F}$	$k_F = 9.78 \times 10^{-20}$
k_C	Positive scalar parameter. It profiles the dynamic power consumption of a single physical server at the Cloud node	(Watt)/(bit/s) $^{\gamma_C}$	$k_C = 1.14 \times 10^{-19}$
r_M	Scalar fraction of the overall computing power shared by the computing cores at the Mobile device	Dimensionless	$r_M = 0$
r_F	Scalar fraction of the overall computing power shared by the computing cores of a single physical server at the Fog node	Dimensionless	$r_F = 0.2$
r_C	Scalar fraction of the overall computing power shared by the computing cores of a single physical server at the Cloud node	Dimensionless	$r_C = 0.1$
$P_{ETH}^{(IDLE)}$	Power consumed in the idle state by each physical Ethernet NIC at the Fog and Cloud nodes	(Watt)	$P_{ETH}^{(IDLE)} = 10^{-4}$
$P_{WiFi-M}^{(IDLE)}$	Power consumed in the idle state by each physical WiFi NIC at the Mobile device and Fog node	(Watt)	$P_{WiFi-M}^{(IDLE)} = 1.3$
$P_{CELL-M}^{(IDLE)}$	Power consumed in the idle state by each physical cellular NIC at the Mobile device and Cloud node	(Watt)	$P_{CELL-M}^{(IDLE)} = 0.82$
$\zeta_{WiFi}^{(TX)}$	Positive scalar exponent of the dynamic power consumption of the WiFi NIC in the transmit mode	Dimensionless	$\zeta_{WiFi}^{(TX)} = 2.40$
$\zeta_{WiFi}^{(RX)}$	Positive scalar exponent of the dynamic power consumption of the WiFi NIC in the receive mode	Dimensionless	$\zeta_{WiFi}^{(RX)} = 2.20$

Table A2. Cont.

Parameter	Meaning/Role	Measuring Units	Simulated Settings
$\zeta_{CELL}^{(TX)}$	Positive scalar exponent of the dynamic power consumption of the cellular NIC in the transmit mode	Dimensionless	$\zeta_{CELL}^{(TX)} = 2.45$
$\zeta_{CELL}^{(RX)}$	Positive exponent of the dynamic power consumption of the cellular NIC in the receive mode	Dimensionless	$\zeta_{CELL}^{(RX)} = 2.34$
η	Positive exponent of the Round Trip Times (RTTs) of the TCP/IP WiFi and Cellular connections	Dimensionless	$\eta = 0.6$
RTT_{WiFi}	Average RTT of the TCP/IP WiFi up/down connections	(s)	$RTT_{WiFi} = 1.0 \times 10^{-2}$
RTT_{CELL}	Average RTT of the TCP/IP cellular up/down connections	(s)	$RTT_{CELL} = 1.0 \times 10^{-1}$
RTT_{WD}	Average RTT of the (possibly, multi-hop) two-way backhaul	(s)	$RTT_{WD} = 1.0$
MSS	Maximum size of a TCP segment	(bit)	Typically $MSS = 1.2 \times 10^4$
$Prob_{LOSS}$	Loss probability of the Cloud↔Fog TCP/IP connection	Dimensionless	$Prob_{LOSS} = 1.56 \times 10^{-5}$
$\lambda_{WiFi}^{(TX)}$	Power profile of the WiFi NIC in the transmit mode	$\frac{(\text{Watt})}{(\text{bit/s})^{\zeta_{WiFi}^{(TX)}} \times (\text{s})^\eta}$	$\lambda_{WiFi}^{(TX)} = 5.0 \times 10^{-14}$
$\lambda_{WiFi}^{(RX)}$	Power profile of the WiFi NIC in the receive mode	$\frac{(\text{Watt})}{(\text{bit/s})^{\zeta_{WiFi}^{(RX)}} \times (\text{s})^\eta}$	$\lambda_{WiFi}^{(RX)} = 1.4 \times 10^{-14}$
$\lambda_{CELL}^{(TX)}$	Power profile of the cellular NIC in the transmit mode	$\frac{(\text{Watt})}{(\text{bit/s})^{\zeta_{CELL}^{(TX)}} \times (\text{s})^\eta}$	$\lambda_{CELL}^{(TX)} = 2.31 \times 10^{-13}$
$\lambda_{CELL}^{(RX)}$	Power profile of the cellular NIC in the receive mode	$\frac{(\text{Watt})}{(\text{bit/s})^{\zeta_{CELL}^{(RX)}} \times (\text{s})^\eta}$	$\lambda_{CELL}^{(RX)} = 8.1 \times 10^{-15}$
I_{MAX}	Maximum number of primal-dual iterations performed by the RAP_p function	Dimensionless	$500 \leq I_{MAX} \leq 700$
a_{MAX}	Speed-factor of the iterations performed by the RAP_p function	Dimensionless	$10^{-7} \leq a_{MAX} \leq 8 \times 10^{-7}$
no_{HOP}	Number of hops of the two-way Cloud↔Fog backhaul connection	Dimensionless	$no_{HOP} = 4$
P_{HOP}	Per-hop average power consumed the two-way backhaul connection	(Watt)	$P_{HOP} = 5.7 \times 10^{-1}$
PS	Population size of the simulated genetic functions. It must be even and no less than 4	Dimensionless	$120 \leq PS \leq 400$
CF	Fraction of the population size that undergoes genetic crossover	Dimensionless	$CF = 0.5$
G_{MAX}	Positive scalar integer parameter. It is the number of generations run by the genetic functions	Dimensionless	$20 \leq G_{MAX} \leq 100$
MN	Number of elements of each task allocation vector that undergo genetic mutation	Dimensionless	$MN = \text{round}((V - 2) / 2)$
\vec{a}_{MAX-FT}	3-tuple vector of positive speed-factors of the gradient-based iterations performed by <i>FogTracker</i>	Dimensionless	Each element of the vector \vec{a}_{MAX-FT} is in the range: $[7.0 \times 10^{-8}, 9.0 \times 10^{-7}]$

Table A2. Cont.

Parameter	Meaning/Role	Measuring Units	Simulated Settings
$jump_{WiFi}^1$	Non-negative parameter. It is the first multiply scaling applied by <i>FogTracker</i> to R_U^{MAX} and R_D^{MAX}	Dimensionless	$jump_{WiFi}^1 = 0.0$
$jump_{CELL}^1$	Non-negative parameter. It is the first multiply scaling applied by <i>FogTracker</i> to B_U^{MAX} and B_D^{MAX}	Dimensionless	$jump_{CELL}^1 = 1.0$
$jump_{WiFi}^2$	Non-negative parameter. It is the second multiply scaling applied by <i>FogTracker</i> to R_U^{MAX} and R_D^{MAX}	Dimensionless	$jump_{WiFi}^2 = 1.0$
$jump_{CELL}^2$	Non-negative parameter. It specifies the second multiply scaling applied by <i>FogTracker</i> to B_U^{MAX} and B_D^{MAX}	Dimensionless	$jump_{CELL}^2 = 0.0$
<i>iteration_number</i>	Number of iterations performed by <i>FogTracker</i> (must be in multiples of 5)	Dimensionless	<i>iteration_number</i> = 5×10^3

References

1. Khan, A.U.R.; Othman, M.; Madani, S.A.; Khan, S.U. A survey of mobile cloud computing application models. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 393–413. [\[CrossRef\]](#)
2. Gupta, A.; Jha, R.K. A survey of 5G network: Architecture and emerging technologies. *IEEE Access* **2015**, *3*, 1206–1232. [\[CrossRef\]](#)
3. Baccarelli, E.; Vinueza Naranjo, P.G.; Scarpiniti, M.; Shojafar, M.; Abawajy, J.H. Fog of Everything: Energy-efficient Networked Computing Architectures, Research Challenges, and a Case Study. *IEEE Access* **2017**, *5*, 9882–9910. [\[CrossRef\]](#)
4. Checko, A.; Christiansen, H.L.; Yan, Y.; Scolari, L.; Kardaras, G.; Berger, M.S.; Dittmann, L. Cloud RAN for mobile networks: A technology overview. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 405–426. [\[CrossRef\]](#)
5. Baccarelli, E.; Biagi, M.; Bruno, R.; Conti, M.; Gregori, E. Broadband Wireless Access Networks: A Roadmap on Emerging Trends and Standards. In *Broadband Services: Business Models and Technologies for Community Networks*; Wiley Online Library: Hoboken, NJ, USA, 2005; Chapter 14; pp. 215–240. [\[CrossRef\]](#)
6. Baccarelli, E.; Scarpiniti, M.; Momenzadeh, A. Fog-Supported Delay-Constrained Energy-Saving Live Migration of VMs Over MultiPath TCP/IP 5G Connections. *IEEE Access* **2018**, *6*, 42327–42354. [\[CrossRef\]](#)
7. Pahl, C.; Brogi, A.; Soldani, J.; Jamshidi, P. Cloud container technologies: A state-of-the-art review. *IEEE Trans. Cloud Comput.* **2017**. [\[CrossRef\]](#)
8. Calheiros, R.N.; Ranjan, R.; Beloglazov, A.; De Rose, C.A.; Buyya, R. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exp.* **2011**, *41*, 23–50. [\[CrossRef\]](#)
9. Kliazovich, D.; Bouvry, P.; Audzevich, Y.; Khan, S.U. GreenCloud: A packet-level simulator of energy-aware cloud computing data centers. In Proceedings of the 2010 IEEE Global Telecommunications Conference (GLOBECOM 2010), Miami, FL, USA, 6–10 December 2010. [\[CrossRef\]](#)
10. Núñez, A.; Vázquez-Poletti, J.L.; Camineiro, A.C.; Castañé, G.G.; Carretero, J.; Llorente, I.M. iCanCloud: A flexible and scalable cloud infrastructure simulator. *J. Grid Comput.* **2012**, *10*, 185–209. [\[CrossRef\]](#)
11. Sotiriadis, S.; Bessis, N.; Asimakopoulos, E.; Mustafee, N. Towards simulating the Internet of Things. In Proceedings of the 28th International Conference on Advanced Information Networking and Application Workshops, Victoria, BC, Canada, 13–16 May 2014. [\[CrossRef\]](#)
12. Zeng, X.; Garg, S.K.; Strazdnis, P.; Jayaraman, P.; Georgakopoulos, D.; Ranjan, R. IOTSim: A simulator for analysing IoT applications. *J. Syst. Archit.* **2017**, *72*, 93–107. [\[CrossRef\]](#)
13. Gupta, H.; Dastjerdi, A.V.; Ghosh, S.K.; Buyya, R. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Softw. Pract. Exp.* **2017**, *47*, 1275–1296. [\[CrossRef\]](#)

14. Sotiriadis, S.; Bessis, N.; Antonopoulos, N.; Anjum, A. SimIC: Designing a new inter-cloud simulation platform for integrating large-scale resource management. In Proceedings of the 27th IEEE International Conference on Advanced Information Networking and Applications (ANIA 2013), Barcelona, Spain, 25–28 March 2013. [[CrossRef](#)]
15. Sonmez, C.; Ozgovde, A.; Ersoy, C. EdgeCloudSim: An environment for performance evaluation of Edge Computing systems. *Trans. Emerg. Telecommun. Technol.* **2018**, *29*, e3493. [[CrossRef](#)]
16. Padhye, J.; Firoiu, V.; Towsley, D.; Kurose, J. Modeling TCP throughput: A simple model and its empirical validation. In Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Vancouver, BC, Canada, 31 August–4 September 1998; ACM: New York, NY, USA, 1998; pp. 303–314. [[CrossRef](#)]
17. Vallina-Rodriguez, N.; Crowcroft, J. Energy management techniques in modern mobile handsets. *IEEE Commun. Surv. Tutor.* **2013**, *15*, 179–198. [[CrossRef](#)]
18. Altamimi, M.; Abdrabou, A.; Naik, K.; Nayak, A. Energy cost models of smartphones for task offloading to the cloud. *IEEE Trans. Emerg. Top. Comput.* **2015**, *3*, 384–398. [[CrossRef](#)]
19. Kwak, J.; Choi, O.; Chong, S.; Mohapatra, P. Processor-network speed scaling for energy: Delay tradeoff in smartphone applications. *IEEE/ACM Trans. Netw.* **2016**, *24*, 1647–1660. [[CrossRef](#)]
20. Zhang, L.; Tiwana, B.; Qian, Z.; Wang, Z.; Dick, R.P.; Mao, Z.M.; Yang, L. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In Proceedings of the 2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Scottsdale, AZ, USA, 24–29 October 2010; pp. 105–114.
21. Huang, J.; Qian, F.; Gerber, A.; Mao, Z.M.; Sen, S.; Spatscheck, O. A close examination of performance and power characteristics of 4G LTE networks. In Proceedings of the 10-th International Conference on Mobile Systems, Applications, and Services, Lake District, UK, 25–29 June 2010; ACM: New York, NY, USA, 2012; pp. 225–238. [[CrossRef](#)]
22. Xiao, Y.; Cui, Y.; Savolainen, P.; Siekkinen, M.; Wang, A.; Yang, L.; Ylä-Jääski, A.; Tarkoma, S. Modeling energy consumption of data transmission over Wi-Fi. *IEEE Trans. Mob. Comput.* **2014**, *13*, 1760–1773. [[CrossRef](#)]
23. Lim, Y.s.; Chen, Y.C.; Nahum, E.M.; Towsley, D.; Gibbens, R.J. Improving energy efficiency of MPTCP for mobile devices. *arXiv* **2014**, arXiv:1406.4463.
24. Mukherjee, A.; De, D.; Roy, D.G. A power and latency aware cloudlet selection strategy for multi-cloudlet environment. *IEEE Trans. Cloud Comput.* **2016**, *7*, 141–154. [[CrossRef](#)]
25. Yang, S.; Kwon, D.; Yi, H.; Cho, Y.; Kwon, Y.; Paek, Y. Techniques to minimize state transfer costs for dynamic execution offloading in mobile cloud computing. *IEEE Trans. Mob. Comput.* **2014**, *13*, 2648–2660. [[CrossRef](#)]
26. Yang, L.; Cao, J.; Yuan, Y.; Li, T.; Han, A.; Chan, A. A framework for partitioning and execution of data stream applications in mobile cloud computing. *ACM SIGMETRICS Perform. Eval. Rev.* **2013**, *40*, 23–32. [[CrossRef](#)]
27. De Maio, V.; Brandic, I. First hop mobile offloading of dag computations. In Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2018), Washington, DC, USA, 1–4 May 2018; pp. 83–92. [[CrossRef](#)]
28. Peng, Q.; Walid, A.; Hwang, J.; Low, S.H. Multipath TCP: Analysis, design, and implementation. *IEEE/ACM Trans. Netw.* **2016**, *24*, 596–609. [[CrossRef](#)]
29. Topcuoglu, H.; Hariri, S.; Wu, M.Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* **2002**, *13*, 260–274. [[CrossRef](#)]
30. Cuervo, E.; Balasubramanian, A.; Cho, D.K.; Wolman, A.; Saroiu, S.; Chandra, R.; Bahl, P. MAUI: Making smartphones last longer with code offload. In Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, San Francisco, CA, USA, 5–18 June 2010; ACM: New York, NY, USA, 2010; pp. 49–62. [[CrossRef](#)]
31. Ra, M.R.; Sheth, A.; Mummert, L.; Pillai, P.; Wetherall, D.; Govindan, R. Odessa: Enabling interactive perception applications on mobile devices. In Proceedings of the 9-th International Conference on Mobile Systems, Applications, and Services, Bethesda, MD, USA, 28 June–1 July 2011; ACM: New York, NY, USA, 2011; pp. 43–56. [[CrossRef](#)]
32. Juve, G.; Chervenak, A.; Deelman, E.; Bharathi, S.; Mehta, G.; Vahi, K. Characterizing and profiling scientific workflows. *Future Gener. Comput. Syst.* **2013**, *29*, 682–692. [[CrossRef](#)]

33. Mahmoodi, S.E.; Uma, R.; Subbalakshmi, K. Optimal joint scheduling and cloud offloading for mobile applications. *IEEE Trans. Cloud Comput.* **2016**. [[CrossRef](#)]
34. Sarkar, A.; Gosh, A.; Nath, A. MapReduce: A comprehensive study on applications, scope and challenges. *Int. J. Adv. Res. Comput. Sci. Manag. Stud.* **2015**, *3*, 256–272.
35. Baccarelli, E.; Biagi, M. Performance and optimized design of space-time codes for MIMO wireless systems with imperfect channel estimates. *IEEE Trans. Signal Process.* **2004**, *52*, 2911–2923. [[CrossRef](#)]
36. Baccarelli, E.; Biagi, M.; Pelizzoni, C. On the information throughput and optimized power allocation for MIMO wireless systems with imperfect channel estimation. *IEEE Trans. Signal Process.* **2005**, *53*, 2335–2347. [[CrossRef](#)]
37. Baccarelli, E.; Biagi, M. Power-allocation policy and optimized design of multiple-antenna systems with imperfect channel estimation. *IEEE Trans. Veh. Technol.* **2004**, *53*, 136–145. [[CrossRef](#)]
38. Baccarelli, E.; Cusani, R. Recursive Kalman-type optimal estimation and detection of hidden Markov chains. *Signal Process.* **1996**, *51*, 55–64. [[CrossRef](#)]



© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).