Maximizing Mobiles Energy Saving Through Tasks Optimal Offloading Placement in two-tier Cloud

Houssemeddine Mazouzi, Nadjib Achir, Khaled Boussetta

Version of Record: https://www.sciencedirect.com/science/article/pii/S0140366419301720
Manuscript_92291733a105576377abeb810c269e53

Maximizing Mobiles Energy Saving Through Tasks Optimal Offloading Placement in two-tier Cloud: a Theoretical and an Experimental Study

Houssemeddine MAZOUZI[a], Khaled BOUSSETTA[a,b], Nadjib ACHIR[a]

[a] *Laboratory L2TI, Institut Sup Galilée, Université Paris 13, Sorbonne Paris Cité 99 Avenue J-B Clement, 93430 Villetaneuse, France*
[b] *Laboratory Agora, CITI, Inria/INSA Lyon, 56 Boulevard Niels Bohr, 69100 Villeurbanne, France*

Abstract

In this paper, we focus on tasks offloading over two tiered mobile edge computing environment. We consider several users with energy constrained tasks that can be offloaded over edge clouds (cloudlets) or on a remote cloud with differentiated system and network resources capacities. We investigate offloading policy that decides which tasks should be offloaded and determine the offloading location on the cloudlets or on the cloud. The objective is to minimize the total energy consumed by the users. We formulate this problem as a Non-Linear Binary Integer Programming. Since the centralized optimal solution is NP-hard, we propose a distributed linear relaxation heuristic based on Lagrangian decomposition approach. To solve the subproblems, we also propose a greedy heuristic that computes the best cloudlet selection and bandwidth allocation following tasks' energy consumption. We

*Corresponding author
Email addresses: mazouzi.houssemeddine@univ-paris13.fr (Houssemeddine MAZOUZI), khaled.boussetta@univ-paris13.fr (Khaled BOUSSETTA), nadjib.achir@univ-paris13.fr (Nadjib ACHIR)

compared our proposal against existing approaches under different system parameters (CPU resources), variable number of users and for six applications, each having specific traffic pattern, resource demands and time constraints. Numerical results show that our proposal outperforms existing approaches. In addition to the theoretical approach, we evaluate our offloading policy using real experiments. In this case, we setup a real testbed composed of client terminal, offloading server located either at the edge or at a remote Cloud. We also implemented our proposal as an offloading middleware on both the client and the offloading server. Using this testbed, we were able to evaluate our offloading decision policy for multi-users context with three real Android OS applications, with different traffic patterns and resource demands. We also discuss the performance of our proposal for each application and we analyze the multi-users effect.

Keywords:

computation offloading; mobile cloud computing; mobile edge computing; cloudlet; Lagrangian decomposition; Offloading Middleware.

---

1. Introduction

In recent years, mobile devices have undergone a major transformation, moving from small devices with limited capabilities to important everyday accessories with important capabilities. This recent advances in hardware and software mobile technology have led to an exponential growth in mobile application markets. Unfortunately, even as mobile applications become more and more intensive, the computing power of mobile devices remains limited compared to what we can find in data-centers and cloud. Furthermore,

because the limited weight and size and therefore the life of the battery, a powerful approach to improving the performance of mobile applications and reducing the shortage of mobile device resources is required. One possible approach is to enable mobile devices to offload some of their intensive workloads to remote high-performance virtual machines. Unfortunately, even though clouds have rich computing and storage resources, they are generally geographically far away from users. In this case, this approach may suffer from significant and fluctuating delays on the Internet. This finding is particularly problematic for some mobile applications, such as augmented reality or cloud gaming, which requires a reduced response time.

To reduce this long access delay, an emerging tendency is to push the cloud to the network edge, mainly located within existing wireless Access Points (APs), ADSL box or Base Stations (BSs). This proximity gives users the opportunity to offload their tasks to this Edge cloud or cloudlets. This new paradigm is known as "**Mobile Edge Computing (MEC)**". A cloudlet can be seen as a small data center, and because of this geographical proximity between users and cloudlets, the access delay on the task offloading can be greatly reduced, compared to remote clouds, and thus significantly improving user experiences. In this paper, we focus on multi-user MEC, where users offload their applications to edge servers or cloudlets. In this case, both the wireless bandwidth and the computing resources must be shared among the users.

The work presented in this paper is an extension of our previous work [1]. In [1], we present a computation offloading policy for multi-users multi-cloudlets environment, named Efficient cloudlet Selection Offloading policy

(ECESO). Our objective is to determine which users should offload their tasks and to which cloudlet in order to minimize the overall energy consumed by the users. Basically, ECESO assigns each user to the best cloudlet in order to reduce the energy consumption of all users, according to the available network and system resources. We formulate this problem as a Binary Integer Programming (BIP) and we propose a distributed linear relaxation heuristic based on Lagrangian decomposition approach. Our policy consists of two decision levels:

- The local offloading decision level that concerns the users associated with the same access point (AP), in order to solve the offloading sub-problem of this AP.
- The global offloading decision level ensures that the offloading solution given by the local offloading decision level complies with the cloudlet resource constraints.

In addition to this theoretical approach, we added in this paper a complete experimental approach in order to evaluate our proposal. Our objective was to evaluate our proposed policy with real Android mobile applications, we also designed and implemented a computation offloading middleware for Android-based terminals. This middleware was integrated on both the mobile terminal and the offloading server. Basically, we integrated our offloading policy to the mobile terminal via a client offloading middleware in order to decide if the task should be executed locally or offloaded to the edge/remote cloud. In the remote virtual machine, we also integrated a server offloading middleware to ensure that the offloaded tasks are correctly executed. Using

this testbed we were able to run multi-users offloading experiments in local edge and in the cloud, with the aim to compare the observed performances to the placement decision made by our proposal ECESO.

The rest of paper is organized as follows: Section 2 presents related work, and section 3 describes the modeled system. Problem formulation and solving are detailed in Section 4. Theoretical and experimental evaluation of our offloading policy is discussed in section 5. Finally, a conclusion is drawn in Section 6.

2. Related Work

Several works were proposed to explore computation offloading in order to improve the performance of the mobile devices. Some work focused on the wireless bandwidth allocation in order to take offloading decision, such as Meng-Hsi Chen et al. [2], Xu Chen et al. [3], Songtao Guo et al. [4, 5], and Keke Gai et al. [6]. The work presented by Meng-Hsi et al. is one of the first works supporting multi-user computation offloading in mobile cloud computing. It decides which task must be performed in the remote cloud and which task must be performed locally. Then, it allocates the wireless bandwidth to each offloaded task in order to reduce the energy consumption of the mobile device. Xu Chen et al. policy was designed to a single cloudlet mobile-edge environment. Each user tries to offload its tasks, accordingly with the available wireless bandwidth to reduce the energy consumption. Another offloading approach for multi-users was presented by Songtao Guo et al. The offloading policy decides which tasks should be offloaded and allocates the wireless bandwidth to each offloaded task. Then, it allocates

the local processor frequency. Lastly, Keke Gai et al. propose a scheduler to assign the tasks between the local mobile device and the remote cloud in order to save energy consumption. In a multi-cloudlet scenario, the computational capacity of each cloudlet is limited, unlike these heuristics, the computation offloading policy must select the best cloudlet to each user with the purpose to achieve high performance.

More recently, many works focus on cloudlets placement heuristics in the MEC environment. The main goal is to find how many cloudlets are needed and where place them, such as Mike Jia et al. [7, 8], Hong Yao et al. [9], and Longjie Ma et al. [10]. Mike Jia et al. heuristic tries to find the best cloudlets placement in a large network, then select a cloudlet to perform the computation tasks of each AP. The K-median clustering based on user density is used to place the cloudlets. Then each AP is statically assigned to a cloudlet. Similarly, Hong Yao et al. was designed to support heterogeneous cloudlets environment. Finally, Longjie Ma et al. was introduced to find the minimal number of cloudlets that must be placed to improve the user experience quality. However, the density of mobile-users are dynamic and changes over time. So, static assignment of the APs to cloudlets may decrease the performance of the computation offloading. To confirm this assumption, our heuristic ECESO consider dynamic cloudlet selection and the wireless bandwidth allocation with the aim of minimizing energy consumption of mobile devices.

Anwesha Mukherjee et al. [11, 12] and Mike Jia et al. [13] focus on the dynamic cloudlet selection in order to reduce the offloading cost. Anwesha Mukherjee et al. designed a multi-level offloading policy to optimize the

energy consumption. The users offload to the nearest cloudlet in the first step. According to the resource availability in this cloudlet, it can perform the tasks or offload the task to another cloudlet and so on. Mike Jia et al. introduced a heuristic to balance the load between the cloudlet. Its main goal is to migrate some tasks from overloaded cloudlets to underloaded cloudlets to reduce the execution time. These works propose dynamic cloudlet selection heuristics, but they do not consider the wireless bandwidth in a multi-user environment. In our previous work [14], we introduce D2M-ECOP a new offloading policy for multi-cloudlet MEC environment. D2M-ECOP focuses on selecting the best cloudlet dynamically to perform the tasks of each user. This proposition achieves high perform in muli-cloudlet MEC environment. However, it does not consider the offloading to the remote cloud in case of overloaded of the cloudlet, which can affect the performance of the computation offloading in such scenario.

All the offloading policies presented above focus on reducing the offloading cost and try to offload the tasks to a predetermined offloading server, a remote cloud or cloudlet. Consequently, the performance of computation offloading can be decreased due to the dynamic density of users in such environment. Therefore, designing a new offloading policy is mandatory. The new policy must consider many offloading servers for whom a user can offload its tasks, and compute optimal task placement by considering two-tier MEC environment.

Other works of the literature have proposed computation offloading platforms for mobile application. Eduardo Cuervo et al. [15] implement Maui plateform for Windows Phone terminals. This platform uses interface and

annotations programming paradigm [16] to indicate which task to offload. Byung-Gon Chun et al. [17] have proposed offloading platform named Clonecloud, which uses Android Virtual Machine to offload tasks on the remote cloud. Clonecloud uses threads-based offloading strategy to offload tasks in order to reduce the completion time of the application. Similarly, Jose Benedetto et al. [18] design MobiCOP, which is an Android computation offloading platform based on Google cloud computing platform. Unfortunately, all the platforms described above have been designed to make the offloading possible but not to choose where to offload. In addition, they do not consider offloading to a multi-cloudlet MEC environment. In this paper, we developed a new computation offloading platform dedicated to multi-cloudlet MEC environment. Using this plateform, we set up a real experimentation in order to validate our offloading policy.

3. System description and modeling

This section presents the MEC system modeling. It describes the computation offloading tasks model, and the network communication model. Then, it details the offloading cost considered in our offloading problem. Table 1 presents all the variables used to modeling our multi-user multi-cloudlet computation offloading problems.

Table 1: Notations and definitions used in the problem modeling

Symbol	Notations and Definitions
K	the number of cloudlets available in the network.
M	the number of APs in the network.
N_m	the number of users associated to the AP m.
$f_{m,n}$	the local computing capacity of the n^{th} user of the m^{th} AP.
F_k	the computing capacity of the cloudlet k.
c_k	the computing resource allocation on cloudlet k.
$dw_{m,n}$	the amount of data to download by the n^{th} user of the m^{th} AP from the MEC.
$up_{m,n}$	the amount of data uploaded to the MEC from the n^{th} user of the m^{th} AP.
$B^{up}_{m,n}$	the allocated upload data rate for the n^{th} user of the m^{th} AP.
$B^{dw}_{m,n}$	the allocated download data rate for the n^{th} user of the m^{th} AP.
$P^{tx}_{m,n}$	power consumption when the Wi-Fi interface is transforming data.
$P^{rx}_{m,n}$	power consumption when the Wi-Fi interface is receiving data.
$P^{Idle}_{m,n}$	power consumption when the Wi-Fi interface is in Idel state
$t_{m,n}$	the maximum tolerated delay according the QoS of the task of the n^{th} user of the m^{th} AP.
$T^{t}_{m,n,k}$	the communication time when n^{th} user of the m^{th} AP offload to cloudlet k.
$T^{l}_{m,n}$	the local processing time for n^{th} user of the m^{th} AP.
$T^{e}_{m,n,k}$	the remote processing time for n^{th} user of the m^{th} AP in cloudlet k.
$\mathcal{Z}^{l}_{m,n}$	the local energy consumption for n^{th} user of the m^{th} AP.
$\mathcal{Z}^{e}_{m,n,k}$	the remote energy consumption for n^{th} user of the m^{th} AP in cloudlet k.
$\gamma_{m,n}$	the computational resource required by the task of the n^{th} user of the m^{th} AP.
λ_m	the Lagrangian multiplier of the subproblem m.
$x^{k}_{m,n}$	the offloading decision variable for the task of n^{th} user of the m^{th} AP in the cloudlet k.
$y_{m,n}$	the category to which belong the tasks (static or dynamic offloading decision task).

3.1. MEC Environment

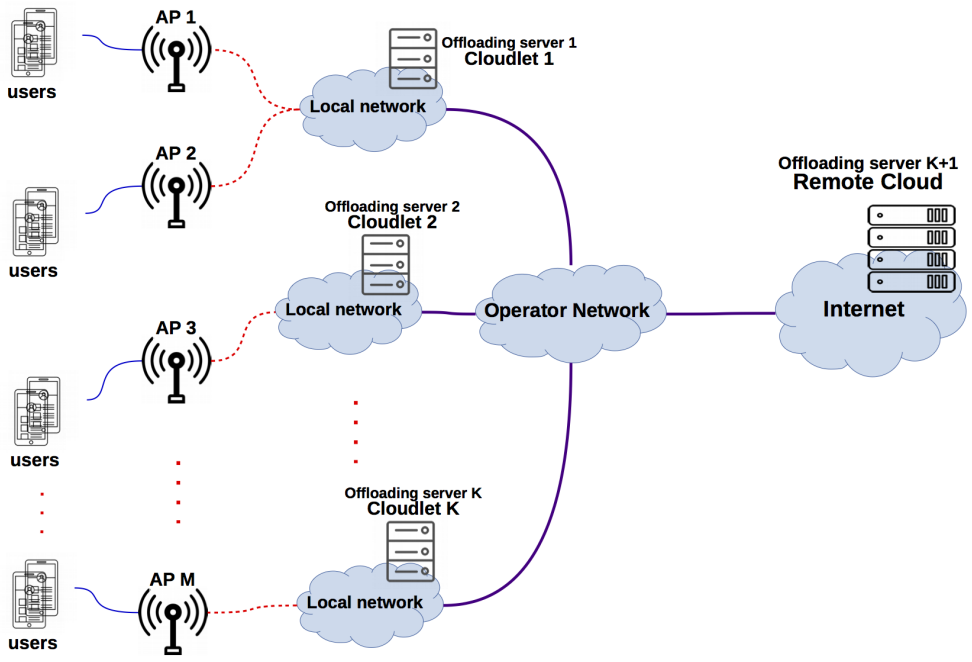


Figure 1: MEC environment

We consider the MEC environment illustrated in Figure 1. The infrastructure is composed of M APs, K deployed cloudlets and one remote cloud. In the remainder, we will refer to the cloud as the $(K+1)$th offloading server. Similarly to [8], we assume that the number of cloudlets is less than the number of access points ($K < M$).

Let $\mathcal{M}$ denote the set of APs. The remote cloud and the cloudlets constitute a set, denoted $\mathcal{K}$, of offloading servers. All these servers offer computation resources to perform offloaded tasks. In each server $k \in \mathcal{K}$, the resources are characterized by a fixed capacity, denoted F_k, of computational resource

units. A computational resource unit is expressed in Ghz and is defined as the number of cycles per second allocated to perform a task. We denote c_k the number of cycles per second allocated by servers k to perform any given offloaded task. Similarly to [4, 3], we consider that c_k, $\forall k \in \mathcal{K}$, is fixed and does not change during the computation. We also assume that the number of computation resources units, F_k, is limited in cloudlets while it is infinite in the cloud. Formally, $\forall k \in \{1, 2, ..., K\}$, $F_k \ll F_{K+1} = \infty$.

3.2. Tasks Requirements

Let consider that the system is observed at a given time. Extension to continuous observation time is possible by discretizing the time into contiguous observation intervals. For each observation time, we assume that each AP $m \in \mathcal{M}$ is associated to a set of users, denoted $\mathcal{N}_m$ of size N_m.

Let (m, n) denote the n^{th} user associated with the m^{th} AP. At observation time, we assume that each user (m, n), $\forall m, n \in (\mathcal{M}, \mathcal{N}_m)$, have exactly one task, denoted $r_{m,n}$. This task is characterized by the number of CPU cycles needed for its computation, denoted in the following as $\gamma_{m,n}$. The purpose of this work is to decide if a task should be performed locally, on the user's terminal or remotely in one of the $\mathcal{K}$ servers. To this end, and similarly to [19], we distinguish two offloading decision tasks:

1. The *static offloading decision task*
2. The *dynamic offloading decision task*

For the first category, the tasks are always offloaded. The offloading decision is taken when the application is designed. In this case, the tasks must be performed remotely on the cloudlets or cloud either because they

require some specific hardware/software environment (e.g. GPU, operating systems) which are not available on the user's terminal or in order to fulfill some application's constraints (e.g. security issues). Accordingly, for static offloading decision task, our objective is to select in which server $k \in \mathcal{K}$ the task must be offloaded.

For the second category, the offloading decision is taken at runtime. These tasks can be executed either locally or offloaded to one of the $k \in \mathcal{K}$ offloading servers. In this case, our objective is to decide if a *dynamic offloading decision task* has to be executed locally (on the user's device) or if it should be offloaded, and if so, to which offloading server $k \in \mathcal{K}$.

Finally, in order to differentiate between the two categories, we associate to each task $r_{m,n}$ a binary variable $y_{m,n}$, equal to 0 if $r_{m,n}$ is a *dynamic offloading decision task* and to 1 if $r_{m,n}$ is a *static offloading decision task*.

3.3. Local Processing time

In the case where a task, $r_{m,n}$, is performed locally, the execution time of the task only depends on the computing capabilities of the user's terminal. Indeed, the transmission time is equal to zero. In the following, we assume that the user's device can allocate, at a given observation time, a computation capacity, denoted as $f_{m,n}$, to perform the task locally. We also assume that this computing capacity remains unchanged throughout the execution of the task. This capacity is expressed as the number of cycles per second. We can therefore deduce the local processing time of the task $r_{m,n}$ as follows:

$$T^l_{m,n} = \frac{\gamma_{m,n}}{f_{m,n}} \tag{1}$$

3.4. Remote Processing time

In the case where the task $r_{m,n}$ is offloaded on server k then the following steps are required:

1. In order to perform task $r_{m,n}$ remotely on server k a given amount of data must be uploaded from the user's terminal to server k, denoted hereinafter as $up_{m,n}$ and expressed in bits. The time required to transmit this data depends on the bandwidth available for the user's (m, n) terminal at the AP m and also to the delay from the AP m to the server k on the backhaul network. Lets denote by $B^{up}_{m,n}$ the upload bandwidth allocated to the user m, n in the AP m to transmit the data required for the task $r_{m,n}$. We will detail in section 3.5 how to compute this bandwidth when a given number of users associated to a same AP must offload their tasks. Accordingly, we can then derive the transmission time to upload the data of an offloaded task $r_{m,n}$ from user (m, n) terminal to its AP m as follows:

$$T^{up}_{m,n} = \frac{up_{m,n}}{B^{up}_{m,n}} \tag{2}$$

   Similarly, let $B^{bh}_{m,k}$ denotes the end to end backhaul bandwidth between the AP m and the offloading server k. We can express the transmission time of the data associated with an offloaded task $r_{m,n}$ from AP m to server k as follow:

$$T^{bh,k}_{m,n} = \frac{up_{m,n}}{B^{bh}_{m,k}} \tag{3}$$

2. This uploaded data is then used by the offloaded server k to perform

the task. The remote computing time of task $r_{m,n}$ can be expressed as the ratio of the CPU cycles required for the task's computing ($\gamma_{m,n}$) to the number of cycles per second allocated at server k (c_k):

$$T_{m,n}^{e,k} = \frac{\gamma_{m,n}}{c_k} \tag{4}$$

3. Finally, when the task's computation is finished, the server k returns back the results to the user's terminal. We denotes by $dw_{m,n}$ the amount or results data returned back to the user (m, n). This quantity is also expressed in bits. We can thus derive the transmission time of task $r_{m,n}$ from server k to AP m as follow:

$$T_{m,n}^{k,bh} = \frac{dw_{m,n}}{B_{m,k}^{bh}} \tag{5}$$

In the same way, We can also express the transmission time to download the remote computation result of the offloaded task $r_{m,n}$ from AP m to user (m, n) terminal as follows:

$$T_{m,n}^{dw} = \frac{dw_{m,n}}{B_{m,n}^{dw}} \tag{6}$$

Here, $B_{m,n}^{dw}$ denotes the allocated bandwidth to transmit the offloaded task results from the AP m to user (m, n).

Finally, we can then compute the completion of task $r_{m,n}$ when the latter is offloaded on server k as follows:

$$T_{m,n}^{k} = T_{m,n}^{up} + T_{m,n}^{bh,k} + T_{m,n}^{e,k} + T_{m,n}^{k,bh} + T_{m,n}^{dw} \tag{7}$$

3.5. Shared wireless access bandwidth

As introduced in the last section, the available bandwidth in upload and download for each user in an AP depends on the number of concurrent transmissions, which means in our case the number of offloaded tasks. In the following we denote by π_m the number of tasks that are offloaded through AP $m \in \mathcal{M}$ to a server $k \in \mathcal{K}$.

In this paper, we consider that access points use 802.11n technology. However, this work can be easily extended to cellular technologies by considering existing models from the literature. As we will emphasize in section 20, the key point here is the fact that the available bandwidth is inversely proportional to the number of offloaded tasks, which makes our optimization problem non-linear.

In order to estimate the bandwidth at each AP, we adopt the Bianchi analytical model of WiFi channels [20, 21] where the characteristics of DCF mechanisms in 802.11n are considered. The parameters are summarized in Table 2.

Table 2: The characteristics of DCF mechanisms in IEEE 802.11n

Parameter	Value
MAC header	272 bits
PHY header	128 bits
ACK	112 bits + PHY header
W(wireless bandwidth)	150 mbps
CW (Contention Window)	15
R (Maximum backoff counter)	7
ϱ (slot time)	9 μs
ϕ (propagation time)	1 μs
SIFS, DIFS	10 μs, 28 μs

Using the Bianchi model [20, 21], the bandwidth of an AP where π_m users are competing for a transmission can be expressed by the following formula:

$$B(\pi_m) = \frac{E(\text{Playload information transmitted in a slot time})}{E(\text{length of a slot time})} \tag{8}$$

$$= \frac{ps.E}{(1-pb).\varrho + ps.\overline{T}_s + (pb-ps).\overline{T}_c} \tag{9}$$

Where

- ps denote the probability that a successful transmission occurs in a slot time.

$$ps = \pi_m \tau (1-\tau)^{\pi_m - 1}$$

Here, τ denotes the stationary probability that one mobile device transmits a packet in a slot time. It can be expressed as:

$$\tau = \frac{1}{1 + \frac{1-p}{2(1-p^{R+1})}[\sum_{j=0}^{R} p^j.(2^j CW - 1) - (1-p^{R+1})]} \tag{10}$$

Where p is the probability of a collision during the transmission of a packet. We can express it as:

$$p = 1 - (1-\tau)^{\pi_m - 1} \tag{11}$$

Fixed point method can be used to solve equations 10 and 11 to determine the value of τ.

- E is the average time to transmit the packet payload information of size d. It can be computed as follows:

$$E = \frac{d}{W} * \frac{CW}{CW - 1} \tag{12}$$

- pb denotes the probability that the channel is busy. It can be computed as follows:

$$pb = 1 - (1 - \tau)^{\pi_m} \tag{13}$$

- $\overline{T}_s$ is the average time the channel is sensed busy because of a successful transmission. Let T_{MPDU}, T_{ACK}, SIFS and DIFS denote the time to transmit the MPDU (including MAC header, PHY header), the time to transmit an ACK, the SIFS time and the DIFS time, respectively.

$$\overline{T}_s = (T_{MPDU} + SIFS + T_{ACK} + DIFS) * \frac{CW}{CW - 1} + \phi$$

Here, T_{MPDU}, T_{ACK} and DIFS denote the time to transmit the MPDU (including MAC header, PHY header), the time to transmit an ACK, and the DIFS time, respectively.

- $\overline{T}_c$ is the average time the channel is sensed busy by each station during a collision.

$$\overline{T}_c = T_{MPDU} + SIFS + T_{ACK} + DIFS + \phi$$

Using equation 9 we can compute the allocated bandwidth to transmit the input data of offloaded task $r_{m,n}$ from user (m, n) to its AP m as follows:

$$B^{up}_{m,n} = \frac{B(\pi_m)}{\pi_m} \tag{14}$$

In the same way, we can obtain the allocated bandwidth to transmit the offloaded task results from the AP m to user (m, n) as follows:

$$B^{dw}_{m,n} = \frac{B(\pi_m)}{\pi_m} \tag{15}$$

3.6. Completion time constraint

Let $T_{m,n}$ denotes the total processing time of task $r_{m,n}$. From the above system description, one can see that $T_{m,n}$ depends on the processing time and eventually, in case of offloading, upload and download transmission times. Precisely, if a task $r_{m,n}$ is performed locally, then $T_{m,n} = T^l_{m,n}$. Otherwise, if a task $r_{m,n}$ is offloaded on server k the $T_{m,n} = T^k_{m,n}$.

Hence, to integrate applications' Quality of Services requirements, we associate to each offloadable task a time constraint threshold. Precisely, we define a maximum completion time threshold, denoted $t_{m,n}$ to any task $r_{m,n}$, $\forall(m, n) \in (\mathcal{M}, \mathcal{N}_m)$.

The completion time is a hard constraint for any task, $r_{m,n}$, $\forall(m, n) \in (\mathcal{M}, \mathcal{N}_m)$. In other words, the optimal offloading policy must satisfy the following constraint:

$$T_{m,n} < t_{m,n} \tag{16}$$

As stated before, the purpose of our offloading policy is to determine

which tasks should be offloaded and to which server in order to satisfy the completion time constraint. In addition to this constraint, our purpose is to determine the optimal offloading policy which minimizes the overall energy consumed by the user's terminals. In the following sections we will detail energy consumption models for both, local and remote tasks processing.

3.7. Energy consumption: local processing

Following the model proposed in [22] of power consumption due to tasks processing on portable devices, we can then derive the total amount of energy consumed to process task $r_{m,n}$ locally as follows:

$$\mathcal{Z}^{l}_{m,n} = \kappa * (f_{m,n})^3 * T^{l}_{m,n} = \kappa * (f_{m,n})^2 * \gamma_{m,n} \tag{17}$$

where κ is the effective switched capacitance, which depends on the chip architecture, and is used to adjust the processor frequency. Similarly to [22], we set $\kappa = 10^{-9}$.

3.8. Energy consumption: Remote Processing

The total amount of energy consumed by the user's device to perform the task remotely is equal to the energy used when the device 1) turns the radio in the transmission mode to send the data to the remote server, 2) turn the radio in idle mode to wait the task completion and finally 3) turn the radio in the reception mode in order to receive the result data from the remote server. The consumed energy can thus be expressed as follows:

$$\mathcal{Z}^{k}_{m,n} \;=\; P^{tx}_{m,n} * T^{up}_{m,n} + P^{idle}_{m,n} * (T^{bh,k}_{m,n} + T^{e,k}_{m,n} + T^{k,bh}_{m,n}) + P^{rx}_{m,n} * T^{dw}_{m,n} \quad (18)$$

where $P^{tx}_{m,n}$ is the power consumption when the radio interface is in transmission mode, $P^{rx}_{m,n}$ is the power consumption when the radio interface is in the reception mode and $P^{idle}_{m,n}$ is the power consumption when the radio interface in idle mode. As is commonly assumed [22], we suppose that

$$P^{idle}_{m,n} \leq P^{rx}_{m,n} \leq P^{tx}_{m,n} \quad (19)$$

4. Problem Formulation and Solving

As introduced earlier, the objective of this paper is to propose an efficient offloading policy that decides which tasks should be offloaded and to which offloading server (cloudlets or cloud), while minimizing the total energy consumed by the mobiles. Given our system description and according to the QoS and offloading servers' resources capabilities constraints, our problem can be formulated as follows:

$$
\begin{aligned}
&\textbf{Minimize} \sum_{m}^{M} \sum_{n}^{N_m} \mathcal{Z}_{m,n} \\
&\textbf{Subject to:} \\
&C1 : \sum_{k=1}^{K+1} x_{m,n}^{k} \leq 1, \forall m \in \mathcal{M}, (m,n) \in \mathcal{N}_m \\
&C2 : y_{m,n} - \sum_{k=1}^{K+1} x_{m,n}^{k} \leq 0, \forall m \in \mathcal{M}, (m,n) \in \mathcal{N}_m \\
&C3 : T_{m,n} \leq t_{m,n}, \forall m \in \mathcal{M}, (m,n) \in \mathcal{N}_m \\
&C4 : \sum_{m}^{M} (\sum_{n}^{N_m} x_{m,n}^{k} * c_k) \leq F_k, \forall k \in \mathcal{K} \\
&C5 : x_{m,n}^{k} \in \{0,1\}, \forall m \in \mathcal{M}, (m,n) \in \mathcal{N}_m, k \in \mathcal{K}
\end{aligned}
\tag{20}
$$

As indicated above, our objective is to minimize the total amount of energy consumed by the mobiles. Here $x_{m,n}^{k}$ is the offloading decision of the task of the user (m,n) to the offloading server k, which means that $x_{m,n}^{k} = 1$ if the user (m,n) offloads its task to the offloading server k, and 0 otherwise. $\mathcal{Z}_{m,n}$ is the amount of energy consumed by the task of the user n on the AP m, and can be computed as following:

$$
\mathcal{Z}_{m,n} = (1 - \sum_{k=1}^{K+1} x_{m,n}^{k}) * \mathcal{Z}_{m,n}^{l} + \sum_{k=1}^{K+1} x_{m,n}^{k} * \mathcal{Z}_{m,n}^{k}
$$

Constraint $C1$ ensures that each task is assigned at most to one offloading server. Constraint $C2$ guarantee that any static offloading decision task must

be assigned to exactly one offloading server, and a dynamic offloading decision task may be assigned to at most one offloading servers. The next constraint $C3$ ensures that the QoS required by the task, in terms of completion time, must be less than a given threshold. The processing time of task $r_{m,n}$ can be expressed as following:

$$T_{m,n} = (1 - \sum_{k=1}^{K+1} x_{m,n}^{k}) * T_{m,n}^{l} + \sum_{k=1}^{K+1} x_{m,n}^{k} * T_{m,n}^{k}$$

The next constraint $C4$ shows that it is not possible to exceed the offloading capacity of the offloading server. Finally, constraint $C5$ ensures that the decision variable, $x_{m,n}^{k}$, is a binary variable.

Theorem 1. The problem defined by equations 20 is a Non-Linear Binary Integer Problem (NLBIP) with an exponential function and constraints. It is an NP-hard problem.

Proof. Let us consider a special case where the same number of users are associated with each AP and all tasks are static offloading decision. So, all the tasks must be offloaded to the cloudlets and the bandwidth allocated to each user is known in advance. Thus, the special case is Linear Binary Integer Problem (LBIP). In fact, this special case can be easily reduced to the General Assignment Problem (GAP) with assignment restrictions, which is NP-hard as shown in [23]. Since the special case is NP-hard, the problem 20 is also NP-hard. □

One possible approach to resolve the above problem is to use some decomposition techniques such as Lagrangian relaxation. Thus, we introduce

the Lagrangian multipliers $\lambda = [\lambda_k, k \in \mathcal{K}]^T$ on the constraint $C5$, since it is considered as a complicating constraint [24]. Here, λ_k denotes the price of all the tasks performed by the k^{th} offloading server. The Lagrangian function is given by:

$$L(X, \lambda) = \sum_{m}^{M} \sum_{n}^{N_m} (\mathcal{Z}_{m,n} + \sum_{k}^{K+1} \lambda_k x_{m,n}^k * c_k) - \sum_{k}^{K+1} \lambda_k F_k$$

In this case, the Lagrange dual problem for the primal problem (20) is then given by:

$$\max_{\lambda} \min_{X} L(X, \lambda)$$

We can see that the Lagrange dual problem is separable into two levels. The first level is the inner minimizing and consists of M subproblems for the M APs. The second level is the outer maximization and represents the master problem that consider the global variable and constraint $C4$.

4.1. Local Computation Offloading Decision Heuristic

As introduced in the last section, we decompose the Lagrange Dual problem into M subproblems. Each subproblem concerns one AP and aims to offload the task which belongs to the users associated to that AP. This subproblem can be formulated as following:

$$\textbf{Minimize} \sum_{n}^{N_m} (\mathcal{Z}_{m,n} + \sum_{k}^{K+1} \lambda_k \cdot x_{m,n}^k \cdot c_k)$$

Subject to:

constraints C1 - C3 and C5 (21)

From the last formulation, we can observe that we need to compute the bandwidth allocated to each user. Unfortunately, according to equation 8 this bandwidth depends on the number of users that offload their tasks (π_m). To overcome this dependency problem, we use a branching heuristic. Basically, it consists of finding the optimal value of π_m, that gives the minimum offloading cost of the subproblem 21. We can easily derive a lower bound for π_m, since the minimum number of tasks that should be offloaded by the users are the tasks that are considered as *static offloading decision tasks*. Similarly, we can also derive an upper bound for π_m, which corresponds to the total number of tasks (N_m) belonging to the users of the AP m. Consequently, we have to add one additional constraint $C6$ to the subproblem formulation (21), as following:

$$C6 : \sum_{n}^{N_m} \sum_{k}^{K+1} x_{m,n}^{k} = \pi_m \tag{22}$$

To solve our subproblem, we propose a distributed greedy heuristic to select which user should offload its task and to which offloading server (cloudlet). Our proposed heuristic is illustrated in the algorithm 1. Basically, we start our algorithm by finding the best offloading server for all static offloading tasks, that minimize the Lagrangian cost $\mathcal{Z}_{m,n}^{k} + \lambda_k \cdot c_k$ under the constraints $C1 - C3$ and $C5 - C6$. There after, since the wireless bandwidth at the AP maybe not enough to offload all the remaining dynamic offloading decision tasks, we propose to define an order to determine which task must be offloaded at first. To do this, we compute for each dynamic offloading decision

task an offloading priority defined as following:

$$a_{m,n} = \mathcal{Z}^{l}_{m,n} - \underset{k \in \mathcal{K}}{min}(\mathcal{Z}^{k}_{m,n}) \tag{23}$$

This offloading priority depicts the potential gain, in terms of energy saving, between a local execution or an offloading of the task. The idea here is that when $a_{m,n}$ increases, the offloading of the task is preferred since more energy is saved at the mobile. Finally, once the number of the offloading task is equal to the current offloading capacity (π_m), the remaining tasks are assigned to being performed locally by the users.

Algorithm 1 *ECESO offloading heuristic*

Output: output the offloading decisions $\mathcal{X}$ and cost $\mathcal{Z}$;

1: **for** each value of π_m **do**
2: allocate bandwidth using equation 14 and 15 ;
3: offload each static offloading decision task to the offloading server k that minimizes $\mathcal{Z}^{k}_{m,n} + \lambda_k c_k$ under constraints $C1 - C3$ and $C5 - C6$.
4: compute $a_{m,n}$ for every dynamic offloading decision task ;
5: Sort dynamic offloading decision tasks in decreasing order of a_{n_m} ;
6: offload each dynamic offloading decision task to the offloading server k that minimizes $\mathcal{Z}^{k}_{m,n} + \lambda_k c_k$ under constraints $C1 - C3$ and $C5 - C6$. Otherwise, the task must be performed locally.
7: update the best offloading cost $\mathcal{Z}$ and decisions $\mathcal{X}$;
8: **end for**

4.2. Global Offloading Decision and Lagrangian Adjustment Heuristics

The outer level of the Lagrangian dual problem refers to the global offloading decision problem. It ensures a feasible offloading solution of the primal problem. As known, the optimal solution of the Lagrange dual requires an exhaustive search of all solutions' space and Lagrange multiplier

values which is a difficult task in general. Consequently, we need to adopt an alternative approach. In this work, we use the Subgradient-based heuristic proposed in [24]. This heuristic has three steps, first solve the subproblems, using our proposed heuristics, for the current value of λ. Then, we check if the obtained solution is feasible or not. If so, we use a Lagrangian Adjustment Heuristic (LAH) to get a feasible solution using local searches. The idea of LAH is to check if every offloading server respect the constraint $C4$. When an offloading server does not respect this constraint, LAH tries to migrate some tasks offloaded from this offloading server to other offloading servers that respects all constraints. Finally, we update the Lagrange multipliers as following:

$$\lambda_k(t+1) = \lambda_k(t) + \theta(t) * (\sum_{m}^{M}(\sum_{n}^{N_m} x_{m,n}^{k} * c_k) - F_k) \tag{24}$$

where $\theta(t)$ is the update step. In this work, we use Held and Karp formula [24] to update this step as following:

$$\theta(t) = \eta(t) * \frac{\mathcal{Z}^* - \mathcal{Z}(t)}{\sum_{k=1}^{K+1}(\sum_{m}^{M}\sum_{n}^{N_m} x_{m,n}^{k} * c_k - F_k)^2} \tag{25}$$

where $\eta(t)$ is a decreasing adaptation parameter $0 < \eta(0) \leq 2$, $\mathcal{Z}^*$ is the best obtained solution of the problem 20 and $\mathcal{Z}(t)$ refers to the current solution

of the Lagrangian Dual. We have:

$$\eta(t+1) = \begin{cases} \vartheta * \eta(t) & \text{if } \mathcal{Z}(t) \text{ did not increase} \\ \\ \eta(t) & \text{otherwise} \end{cases} \tag{26}$$

As suggested in [24], we set the values of $\vartheta = 0.9$ and $\eta(0) = 2$. The master problem repeats these steps until the stop conditions, which are the maximum number of iterations It_{max} and the maximum error ε ($\theta(t) < \varepsilon$).

5. Performance Evaluation

5.1. Numerical Assessment

5.1.1. System Parameters

In this section, we evaluate the performance of our offloading policy by evaluating several performance metrics, i.e. the average number of offloaded tasks and the energy saving from the offloading under different settings. The MEC environment consists of a metropolitan area, which is composed of 20 APs and four cloudlets already deployed among the network. In addition, two network topologies are considered. The ring topology, in which the cloudlets are equidistantly deployed in the AP, i.e: cloudlet 1 is collocated with the AP 1, cloudlets 2 with the AP 6, cloudlet 3 with the AP 11 and cloudlet 4 with the AP 16. The second topology is generated by GT-ITM [25] tool, where the cloudlets are randomly deployed. In order to get a better understanding of the offloading policies performances, we consider real mobile applications. Table 3 illustrates the characteristics of the used applications,

where γ indicates the computing resources required to perform the applications, up represents the data that must be transmitted to the remote server, dw the data that should be received from the remote server and t the maximum tolerated delay according to the QoS required by the application. The first three applications are static offloading decision tasks, and the remaining applications are dynamic offloading decision tasks [19].

Table 3: The characteristic of the real-world applications used for our tests.

Application	$\gamma_{(m,n)}$ (Giga CPU cycles)	$up_{(m,n)}$ (Kilobyte)	$dw_{(m,n)}$ (Byte)	$t_{(m,n)}$ (Second)
	static offloading decision tasks			
FACE	12.3	62	60	5
SPEECH	15	243	50	5.1
OBJECT	44.6	73	50	13
	dynamic offloading decision tasks			
Linpack	50	10240	120	62.5
CPUBENCH	3.36	80	80	4.21
PI BENCH	130	10240	200	163

In addition, we consider two cloudlets configurations, in which we use real-world setting, used by the public cloud provider such as: Amazon AWS and Microsoft azure [3, 4, 10], to simulate the behavior of ECESO real-world scenarios. In the first configuration, each cloudlet has a computing capacity of 1000 Giga cycles/s, and allocates 10 Giga cycles to perform every offloaded task ($F_k = 1000$ and $c_k = 10$). In the second configuration, we consider heterogeneous cloudlets, where cloudlets 1 and 2 have a computing capacity of 500 Giga cycles/s and cloudlets 3 and 4 have a computing capacity of 1500 Giga cycles/s. Both upload and download bandwidths of each AP are

set to 150 mbps and the bandwidth allocated to each user is estimated using the parameter settings used in Bianchi model [21]. Regarding the backhaul network, we use the parameters presented in [26]. Moreover, as in [8], we assume a homogeneous users distribution in the network. As in [22], we also consider that the power consumed in transmission mode is equal to the power consumed in the reception mode and is equal to ten times the power consumed in idle mode. We set $P_{m,n}^{idle}$ to 100 $mWatts$. The local computing capability of each user was randomly chosen from $F_{m,n} \in [0.8, 1, 1.2]$ Giga cycles/s. Finally, we consider that each user randomly chooses an application from those described in the table 3.

In order to evaluate the performances of our proposal, we propose to compare it with the following offloading policies:

- **DOTA** [10, 8]: In **DOTA**, each AP is associated with the nearest cloudlet. In this case, all users connected to this AP offload their tasks to the same cloudlet. When a cloudlet is overloaded, the tasks are migrated to the remote cloud.

- **CBL** [13, 7]: Using **CBL** we also associate each AP to the nearest cloudlet. Thus, all users connected to that AP have to offload their tasks into that same cloudlet. However, unlike **DOTA**, when the cloudlet is overloaded, the tasks are migrated to another cloudlet.

- **FCO** [2, 3]: In this policy, all users offload their tasks to the cloud.

In order to compare these offloading policies, we also define comparison metrics depicting the gain of a given offloading policy $\mathcal{P}$. This gain represents

the benefit of the offloading policy $\mathcal{P}$ compared to case where the task is offloaded to the cloud (i.e. **FCO** policy). We formulate the gain as following:

$$\text{gain of } \mathcal{P} = 100 * \frac{(\text{cost of FCO} - \text{cost of } \mathcal{P})}{\text{cost of FCO}}$$

5.1.2. Results Analysis

In Figure 2, we plot the gain of our policy (ECESO) compared to the gain of DOTA and CBL, when considering a network topology following the configuration 1 with homogeneous cloudlet characteristics. As expected, we can observe that the gain decreases when the number of users increases. This is manly due to the fact that the backhaul cost increases when the number of offloaded task increases. We can also observe that the performances of ECESO, DOTA and CBL are almost equivalent, except when the number of users exceeds 300, where we notice that our approach is slightly better. This is due to the fact that ECESO tries to maximize the bandwidth allocated to each user and offload in priority the tasks with the greatest impact on the cost. Consequently, less tasks are offloading compared to the other offloading policies.

In Figure 3, we compare the performances of ECESO, BCL and DOTA when heterogeneous cloudlets are deployed, for both ring and GT-ITM topologies. As we can see, when few number of users are considered (< 100), the performances of the three policies are equivalent. However, when the number of users increases ECESO outperforms both BCL and DOTA, for both configurations of network topologies. This is due to the fact that when we increase the number of users both cloudlets 1 and 2 cannot support all the offloaded tasks. In this case, DOTA policy start to migrate the overloaded tasks to the

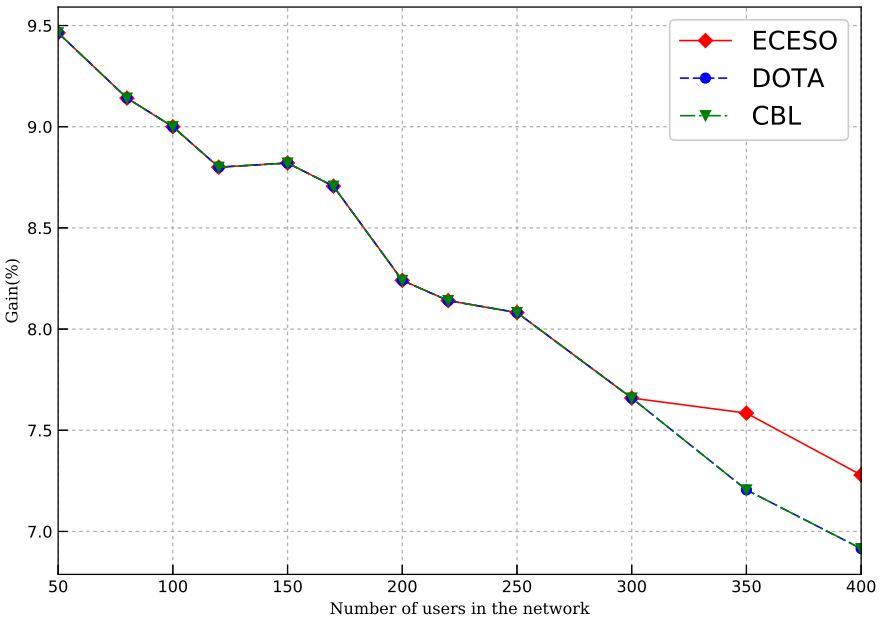


Figure 2: Offloading gain over homogeneous cloudlets configuration

494 remote cloud, which adds more offloading costs. On the other hand, CBL
495 tries to migrate the overloaded tasks from cloudlets 1 and 2 to cloudlets 3
496 and 4 and also some additional offloading cost but less than DOTA. How-
497 ever, using ECESO, for each task, we can select the best cloudlet at the
498 offloading decisions step, which reduce the additional offloading cost due to
499 the migration of tasks introduced in DOTA and CBL. Finally, we notice that
500 the effect of the network topology on the performances of ECESO is more
501 important than the DOTA and CBL. This is due to the fact that ECESO
502 uses the topology to select the cloudlets. However, DOTA and CBL assign
503 statically the AP to the cloudlet.

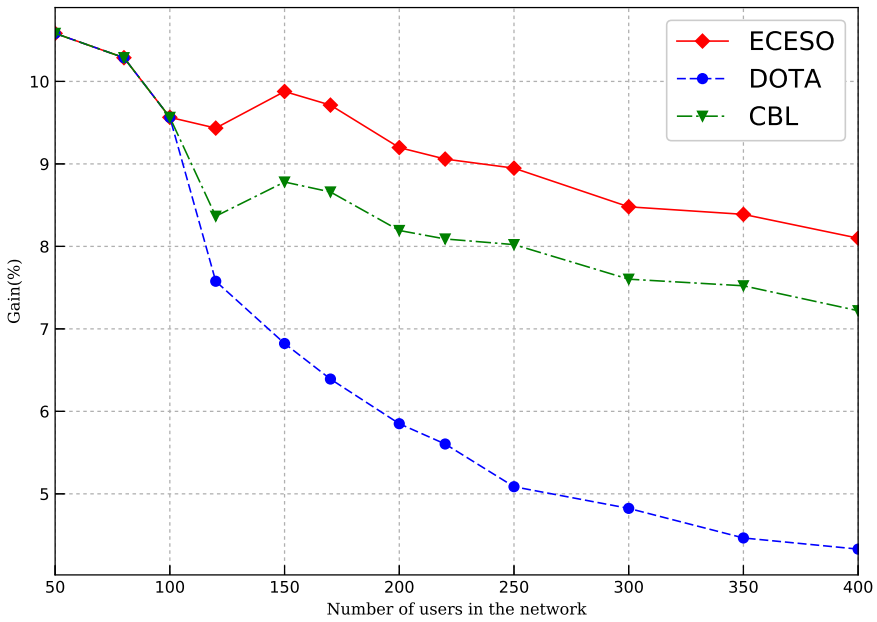
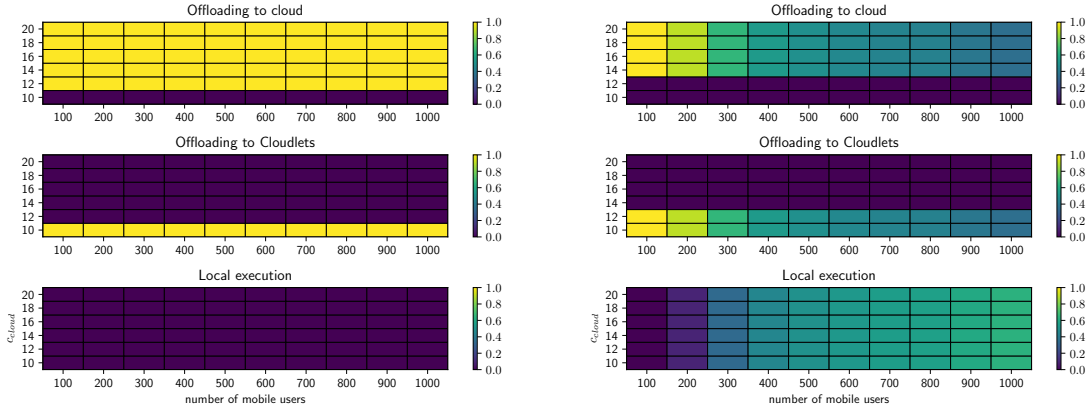
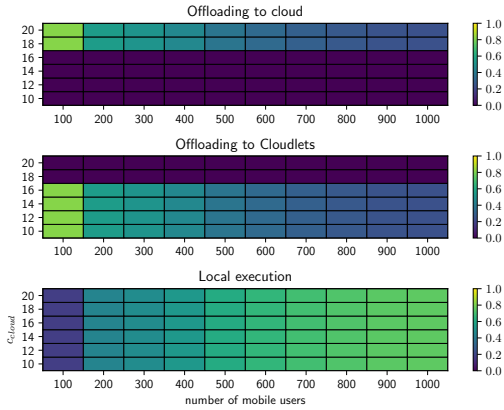


Figure 3: Offloading gain over heterogeneous cloudlets.

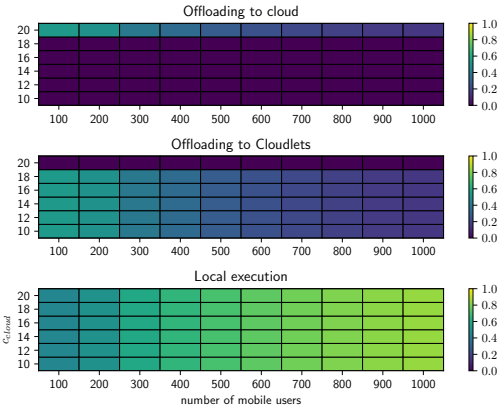


(a) static offload apps

(b) CPUBENCH



(c) PIBENCH



(d) Linpack

Figure 4: Comparison of offloaded tasks to cloudlets and the remote cloud under different users and cloud computing capacity (c_{cloud} in Giga cycles/s).

Finally, in Figure 4 we investigate how each application behaves according to the amount of resources allocated to tasks on the cloudlet and on the cloud. Precisely, we fix the computing capacity allocated to each user in the cloudlets to 10 Giga cycles/s and we vary this capacity between 10 to 20 Giga cycles/s in the cloud [3, 4, 10]. As we can see, not all the applications have the same behavior according to the amount of resources allocated on the cloud. Basically, all static offloading decision tasks (FACE, SPEECH, and OBJECT) are always offloaded to the cloudlets or the cloud have the same computation computing 10 Giga cycles/s. However, where the remote cloud has greater computing capacity than cloudlets ($c_{cloud} > c_{cloudlets}$) all the tasks are offloaded to the remote cloud. Figure 4(b), 4(c) and 4(d) illustrate the performances of the ECESO policy for *dynamic offloading decision tasks*, CPUBENCH, PIBENCH and Linpack, respectively. The ratio of the offloading tasks decreases when the number of users in the network increases, for example, for CPUBENCH applications 100% of tasks are offloaded where the number of users is not greater than 200, but only 30% are offloaded where 1000 users are in the network. This decreasing of the ratio of offloaded tasks is due to the wireless bandwidth in the AP. We also note that the ratio of the offloaded tasks depends on the application characteristics when the applications require a lot of computing resources and transmit a huge amount of data (Linpack and PIBENCH) the ratio of offloaded tasks is lower. As a result, choose the placement of the tasks, remote cloud or cloudlet, depends on many factors, in the figures we noticed that the computing resource allocation in the remote cloud, the computing resource allocation in the cloudlets and the characteristics of the application affect directly the placement of the

offloaded tasks.

5.2. Experimental Assessment

5.2.1. Testbed

In addition to the numerical results presented in the previous section, we evaluated our proposal using real experiments. In this case, we setup a testbed composed of several hardware components, as illustrated in figure 5. The first component consists of the client device acting as a mobile terminal, the second component is the offloading server located either at the edge Cloud or the remote Cloud and finally the last component is a network topology to connect the client device to the offloading server.

For the client device, even if our testbed can be used with real Android mobile terminal, we decided to consider an Android client using a Raspberry Pi 3 device. The main idea is to have a controlled experimental environment, since we need to measure the energy consumed by the device and also to control the bandwidth for the client. We used a Raspberry Pi 3 Model B (RPi3) powered by a Quad Core Broadcom BCM2837 64bit ARMv8 processor at 1.2 GHz and 1GB LPDDR2 of RAM at 900 MHz. On this client device, we installed Android 8 "Orio" as operating system and also our client offloading middleware. Finally, we developed and implemented three dynamic offloading decision tasks applications: Linpack, CPUBENCH and PI BENCH. Each of these bench applications was implemented as a fragment within a common Java Android application. The size of the `APK` source file is 5427.2 Kilobytes.

The second component of our testbed is the offloading server. As shown in the figure 5, the offloading server can be located at the edge Cloud or at the remote Cloud. For the edge Cloud, we used a desktop PC powered by an

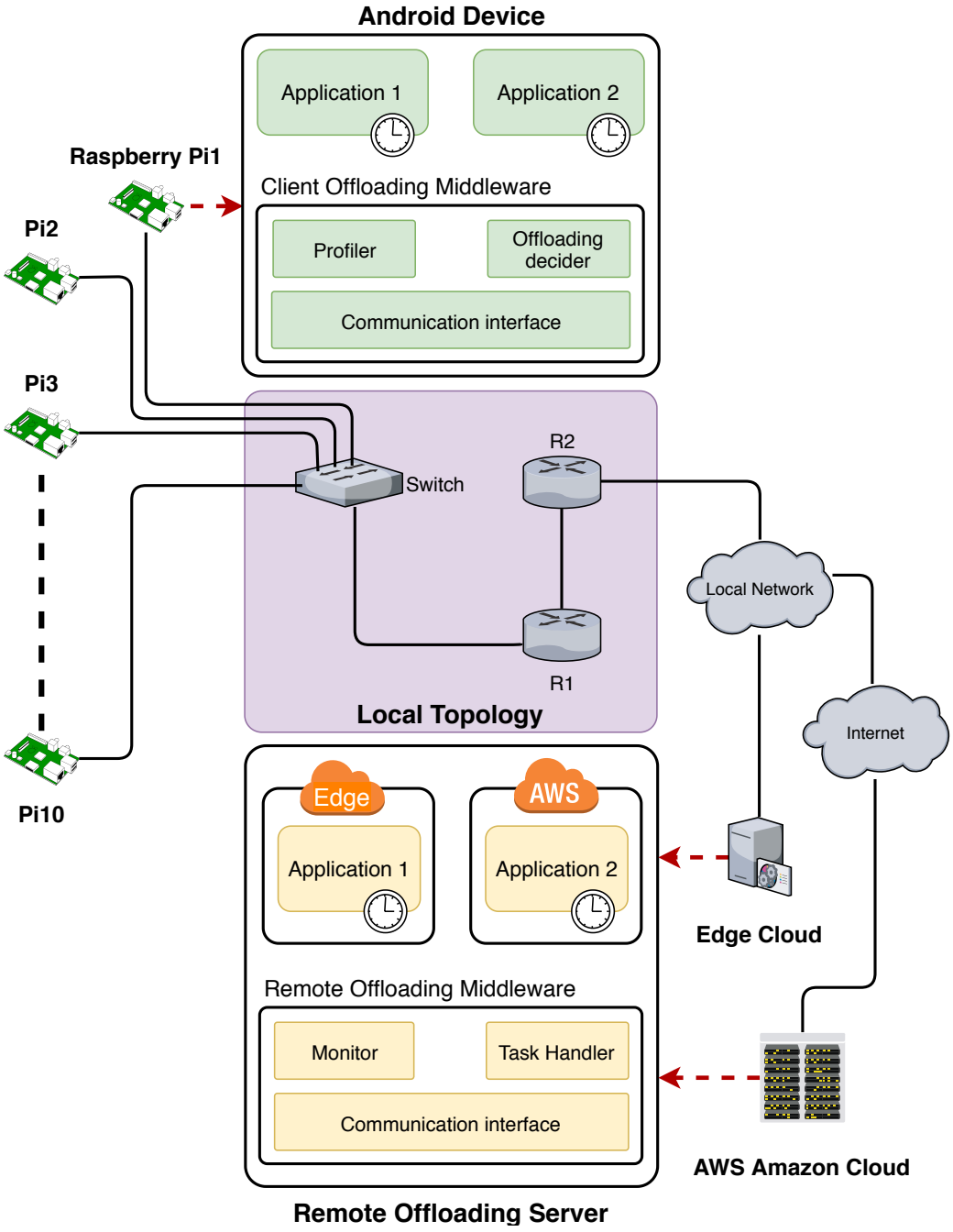


Figure 5: An Overview Architect of Computation Offloading Platform

`Intel I7-6700 8 cores CPU` at `3.40 GHz` and `16 GB DDR3` of RAM with an `Ubuntu 18.04.2 LTS` as operating system. In order to run native Android applications, we also installed, using VirtualBox, a virtual machine with Android-x86 distribution as operating system. Android-x86 [27] is an open source project to port Android on x86 platform. Finally, we also implemented and installed on this distribution our server offloading middleware. For the remote Cloud, we deploy our offloading server on AWS Amazon Cloud using the `t2.medium` instance. This instance is powered by high-frequency Intel Xeon processors with 2 vCPU and a memory of 4 GB. As for the edge Cloud, we also installed our server offloading middleware on this virtual machine.

To connect the Android client device to the offloading server, we have set up a network topology composed of two Cisco routers. The first router is connected to the client via a switch (LAN) and the second router allows our testbed to be connected to the edge Cloud through a local network and to the remote Cloud via an Internet connection. To connect the two routers, we use a serial link allowing us to control the speed at which the data, in bits per second (bps), is sent between the two routers. The main objective is to control the bandwidth between the offloading client and the offloading server. In our experiments, we used the bandwidth values offered by the serial link which are: $1.2Kbps$, $4.8Kbps$, $9.6Kbps$, $38.4Kbps$, $72Kbps$, $125Kbps$, $500Kbps$, $5.3Mbps$ and $8Mbps$.

We also implemented and deployed our offloading middleware. This middleware implementation is based on client-server architecture. On both the client and the server, our middleware is integrated to Android operating system as a new service which must be executed in the background by the An-

droid OS. Inspired by the offloading platforms from literature [15, 18, 17, 28], our offloading middleware uses component-based design pattern [29]. For instance, the client offloading middleware has three main functions: the *profiler function*, the *offloading decider function* and the *communication function*. The objective of the profiler is to collect the information related to the application (i.e. task), such as the hardware usage and the network bandwidth. This information is stored in a local database in order to be used by a second function, which is the offloading decider function. The main objective of the offloading decider function is to decide whether the task should be executed locally or offloaded to the edge or to the remote Cloud. We implemented our offloading policy within this function. Finally, the last function is the communication function, which is in charge of handling the communications between the client offloading middleware and the remote offloading middleware. Basically, in case of offloading, this function sends both the `APK` source code and the input data corresponding to the offloaded task to the offloading server.

In addition to the client offloading middleware, we also developed a remote offloading middleware. The main objective of this middleware is to execute the task offloaded from the client, and also to return the results obtained at the end of the execution. This remote offloading middleware is also composed of three main functions: *task handler function*, *monitor function* and *communication function*. The main objective of the task handler function is to load and execute the `APK` source code of each received task. The task handler is also in charge of maintaining an isolated execution environment between the received tasks by executing each task in a separated

thread. At the end of the task execution the task handler gathers the results and, using the communication function, sends it back to the client offloading middleware. Finally, monitor function saves information (i.e. logs) related to the task execution.

5.2.2. Discussion

The first purpose of the experiments was to characterize the benchmark applications: Linpack, CPU-Bench and PI-Bench. Each of these applications was parameterized in order to study the system under three CPU resource requirement situations: 1) *Light*, 2) *Medium* and 3) *Full*. Precisely, Linpack is a software that performs matrix calculations. To generate *light* processing requirements, we parameterized Linpack with 17 square matrix sizes, ranging from 1 to 17. *Medium* and *full* configurations were generated by multiplying the above matrix sizes by a factor of 40 and 70, respectively. CPU-Bench is a software that generates a random floating point and a random integer of a predefined size n. To generate light, medium and full configurations, we set $n = 10^6$, 10^7 and 10^8, respectively. Finally, three CPU resource requirement situations for Pi-Bench are obtained by computing *pi* with an approximation of 10^3, 10^4 and 10^5 decimal places. All the shown results are average values obtained after executing 10 runs with the same experiments setting.

Using our testbed, we measured the number of CPU cycles ($\gamma_{m,n}$), the quantity of uploaded data ($up_{m,n}$) and the quantity of downloaded data ($dw_{m,n}$) of the three bench applications, under *Light*, *Medium* and *Full* configurations. The results are shown in Table 4. One can observe that all the measured parameters are different from those reported from the literature [19] in table 3, except downloaded data for Linpack. New versions

and the possibility to execute these applications by setting some parameters might explain such a gap. Notice also that due to our implementation of these three bench applications as fragments within the same Java Android application, the quantity of the downloaded data is always equal to the size of the `APK` source file (5427.2 bytes). Among the three bench applications, Linpack is the one that generates the lowest number of CPU cycles. Under *Full* configuration, it is also the one that leads to the highest number of CPU cycles.

Table 4: The characteristic of the bench apps

Application	$\gamma_{m,n}$ (Giga CPU cycles)	$up_{m,n}$ (Kilobyte)	$dw_{m,n}$ (Byte)
Linpack			
Light	0.2033	5427,2	120
Medium	547.406	5427,2	120
Full	2909.11	5427,2	120
PIBENCH			
Light	2.203	5427,2	56
Medium	45.539	5427,2	56
Full	1310.882	5427,2	56
CPUBENCH			
Light	8.696	5427,2	30
Medium	293.613	5427,2	30
Full	745.435	5427,2	30

Figure 6 shows the energy consumption measured in our testbed for Linpack *easy*, when it is executed locally on the mobile terminal, on the edge with 1 or 2 vCPU and on the clouds located at Paris or at the Ohio, both allocating 2 vCPU. The x axis represents the bandwidth of the serial link

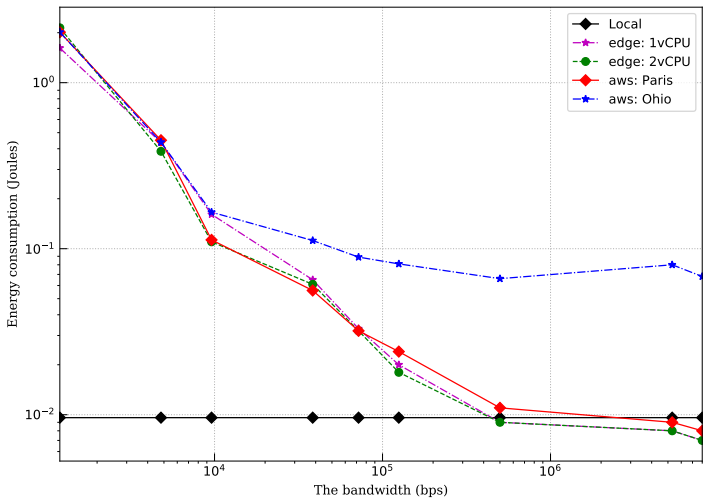


Figure 6: Linpack *easy*: Energy Consumption

that we have varied from 1.2 Kbps to 8 Mbps. Notice that logarithmic scale is used for x and y axis. As one can see, when the offered bandwidth is restricted to less than $500Kbps$, the lowest energy consumption is obtained for local execution. Above $500kbps$, offloading Linpack *easy* on the edge with 2vCPU offers better energy performances. Starting from $5.3Mbps$ offloading on the cloud located at Paris outperforms the local execution. These experimental results prove that, even for applications that require light processing resources, offloading could be energetically beneficial when the available bandwidth to the remote server is sufficiently high.

Another situation fostering the remote execution is observed when the task's processing needs is important. This is clearly illustrated for Linpack *medium* in figure 7. We can notice a certain hierarchy in the performances: first the closest edge, then the remote cloud, and lastly the local execution.

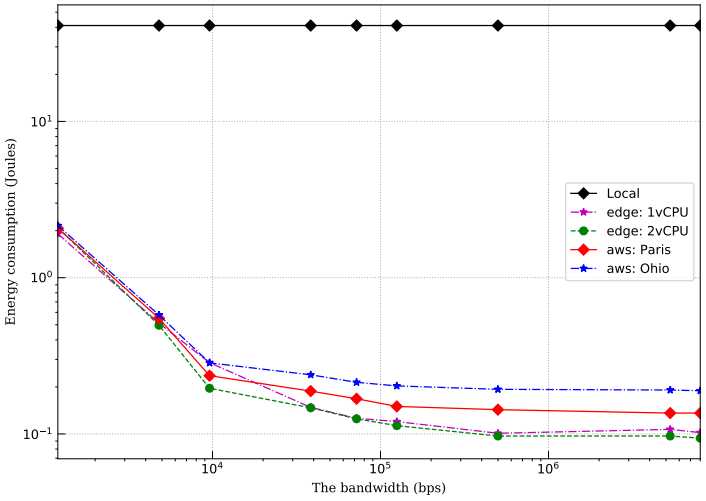


Figure 7: Linpack *medium*: Energy Consumption

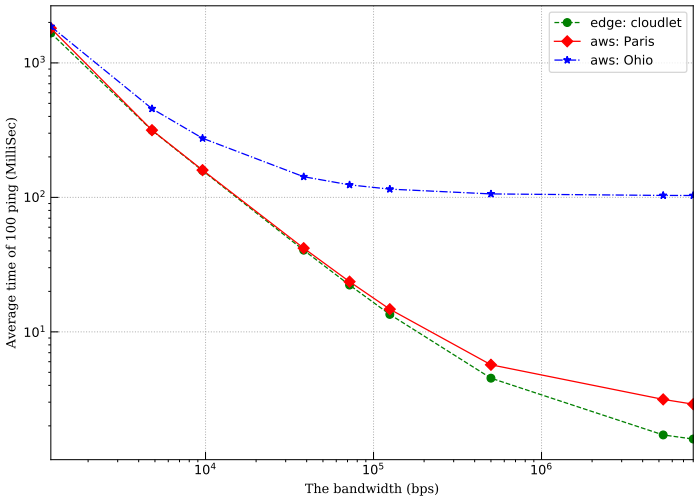


Figure 8: Round Trip Time between the terminal and different locations

A noticeable discrepancy can be observed in figure 7 between the clouds at Paris and at Ohio. We believe that this is related to differentiated network latency from our lab to these two clouds. The average Round Trip Time (RTT) measured from the terminal toward different remote servers is drawn in figure 8. We can see that the RTT for the edge and the cloud in Paris are very close to each other, while the RTT to the cloud in Ohio increases significantly for larger bandwidth values.

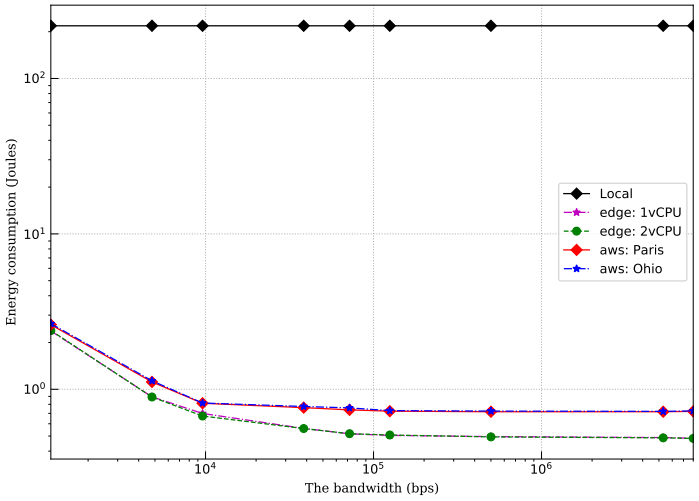


Figure 9: Linpack *full*: Energy Consumption

As shown in Figure 9, the observations made for Linpack *medium* are reinforced for the *full* variant. Because Linpack *full* is a CPU-intensive application, it is obvious that its completion time is mainly dominated by the computation duration. The transmission delay is proportionally negligible. This explains why the effect of differentiated RTT between Paris and Ohio is hardly observable in figure 9. However, the performance gap between the

edge and the clouds is very apparent and almost insensitive to the bandwidth, when the latter reaches $72Kbps$. We can deduce that the local edge offers higher CPU capacity than those provided at the clouds, despite the fact that both have allocated 2vCPU resources. This discrepancy illustrates the impact of the heterogeneity of the hardware and the software technologies, which are used in a multi-tier mobile edge computing infrastructure.

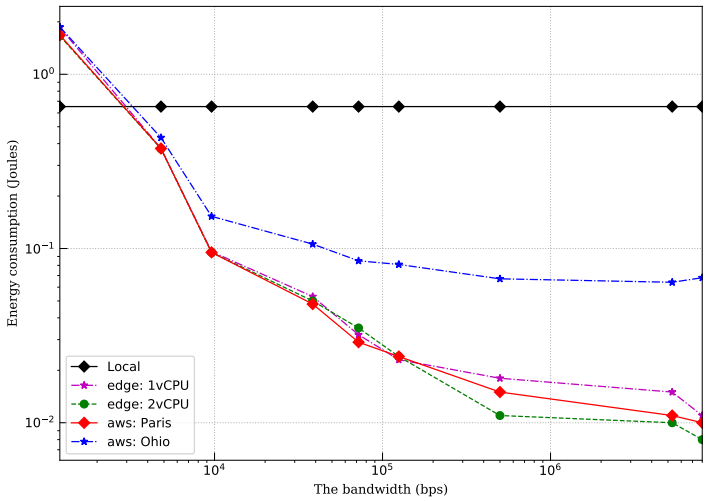


Figure 10: CPUBench *easy*: Energy Consumption

Energy consumption for CPU-Bench *easy*, *medium* and *full* are shown in figures 10, 11 and 12, respectively. The general observations regarding Linpack holds for CPUBench. However, for CPU-Bench *easy* one can see in figure 10 that when the bandwidth is restricted to less than $125Kbps$ the cloud in Paris and the local edge have almost the same performances, with a very slight advantage for the cloud at Paris. This can be explained by the fact that transmission duration over networks that exhibit important

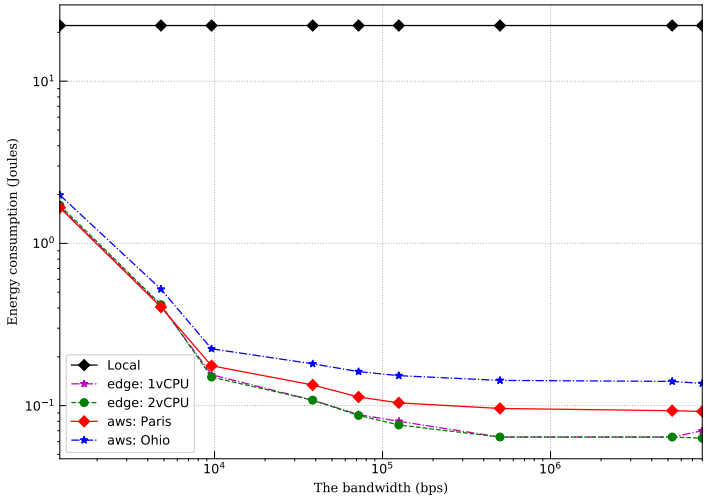


Figure 11: CPUBench *medium*: Energy Consumption

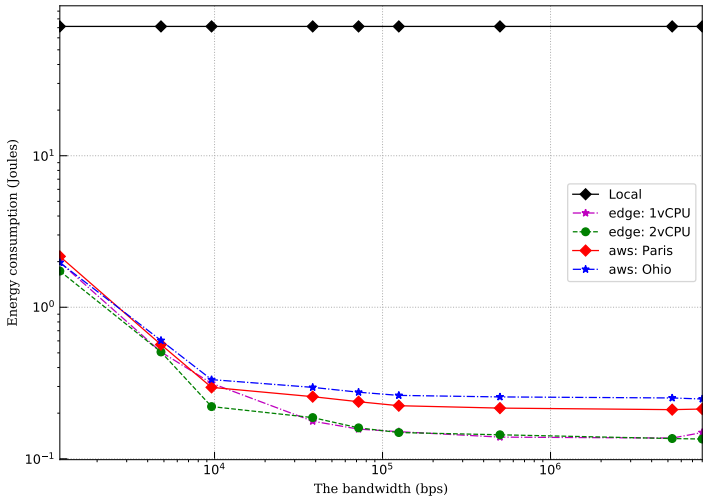


Figure 12: CPUBench *full*: Energy Consumption

delays represent a significant proportion part with respect to the completion time of an offloaded light task. As shown in figure 8 when the bandwidth is restricted to less than $125Kbps$ the Round Trip Time from the terminal toward the local edge is elevated and quite similar to the RTT to the cloud at Paris.

Figures 13, 14 and 15 are relative to Pi-Bench. Compared to the previous bench applications one can notice for *easy* and *medium* cases that the edge with 2vCPU outperforms significantly the edge with 1vCPU. We believe that this is due to the multi-threaded nature of this application. The computation time is lower in 2vCPU compared to 1vCPU thanks to the ability of the former edge to process in parallel the computation of this multi-threaded application. The advantage of offloading such multithreaded application in a multi-CPU sever is clearly shown in figure 15 for the CPU-intensive case. The clouds in Paris and Ohio, where 2vCPU resources are allocated, outperform the 1vCPU edge.

Completion time for PiBench *easy*, *medium* and *full* are shown in Figures 16, 17 and 18, respectively. One can remark that this completion time curves have the same shape and follow the same hierarchy than those observed in energy consumption figures 14 and 15. Actually, in all our experiments we noticed that the consumed energy on the terminal is quasi-stationary during the execution time of an offloaded task. Consequently, the consumed energy is proportional to the completion time of an offloaded task. This observation holds for Linpack and CPU-Bench, as well[1]. The corollary to this finding is that in case of offloading, the remote location that minimizes

[1]To save space, we choose not to show the completion time for Linpack and CPU-Bench

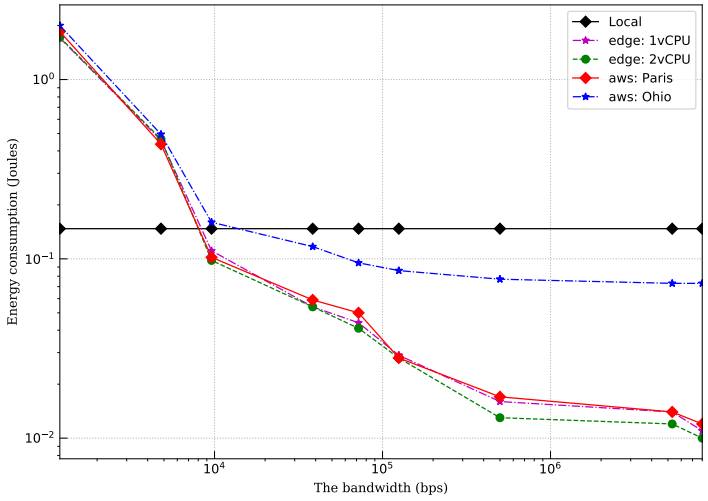


Figure 13: PiBench *easy*: Energy Consumption

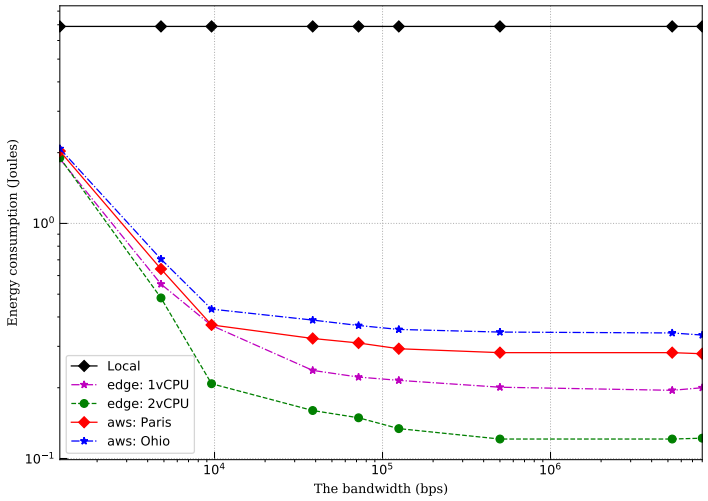


Figure 14: PiBench *medium*: Energy Consumption

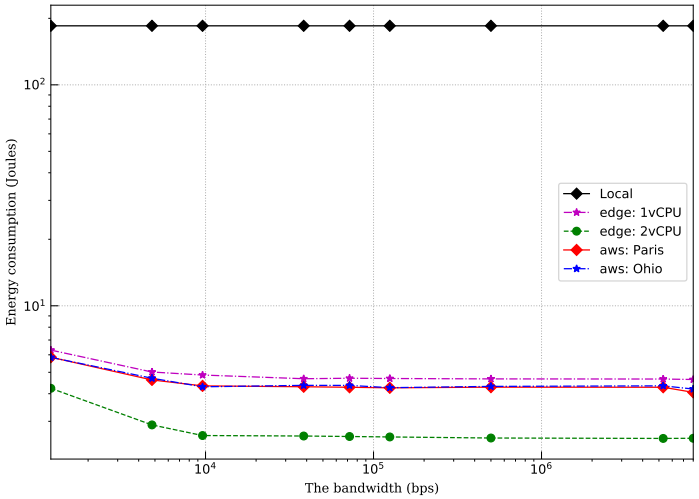


Figure 15: PiBench *full*: Energy Consumption

the completion time is also the one that optimize the consumed energy. This statement has been validated in all the experiments that we have run, for the three bench applications, under different CPU and network configurations.

When a task is executed locally, the measured energy is also quasi-stationary during the processing time. However, as illustrated in figures 6, 7, 9, 10, 11, 12 13, 14 and 15 the difference of consumed energy between local and remote execution is of several order of magnitude. Yet, in almost all the experiments that we have run, we noticed that the decision among local and offloading that minimize the completion time is also the one that optimize the consumed energy. The unique observed exception case to this rule is Pi-Bench *easy* when the bandwidth is restricted to $9.6Kbps$. Comparing figure 16 to 13, we can remark that when the bandwidth is restricted to $9.6Kbps$ the local execution minimize the completion time, while energy

consumption is minimized when Pi-Bench *easy* is offloaded on the 2vCPU edge.

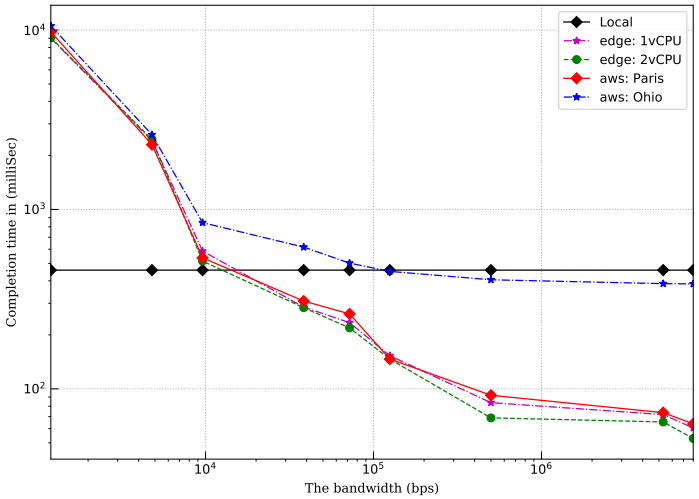


Figure 16: PiBench *easy*: completion time

From figures 6, 7, 9, 10, 11, 12 13, 14 and 15 we can easily identify the locations that optimize the consumed energy for different applications and network configurations. The right column in table 5, shows the location that minimizes the consumed energy that we derived from experimental results for the three bench applications under different processing loads and network configurations. The left column indicates the optimal locations obtained by our proposed framework ECESO. To fit with the applications that we evaluated in our testbed, we applied the measured parameters indicated in table 4 to ECESO. We can see that in the majority of the cases the theoretical solution match with the placement derived from the experimentation results. The few mismatches are indicated in red. For Linpack, the ECESO placement

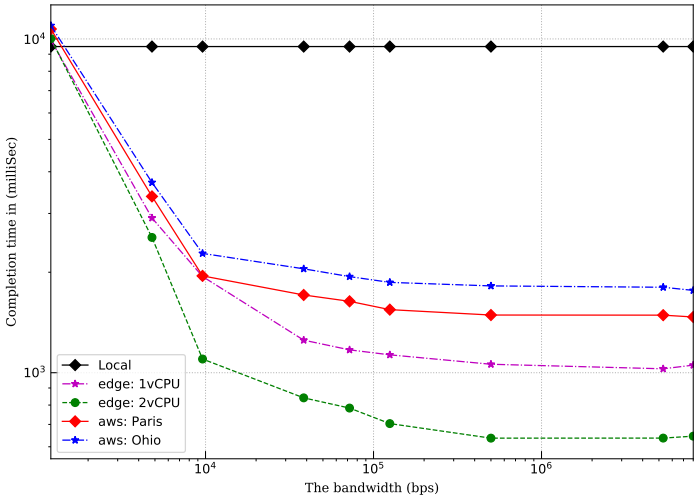


Figure 17: PiBench *medium*: completion time

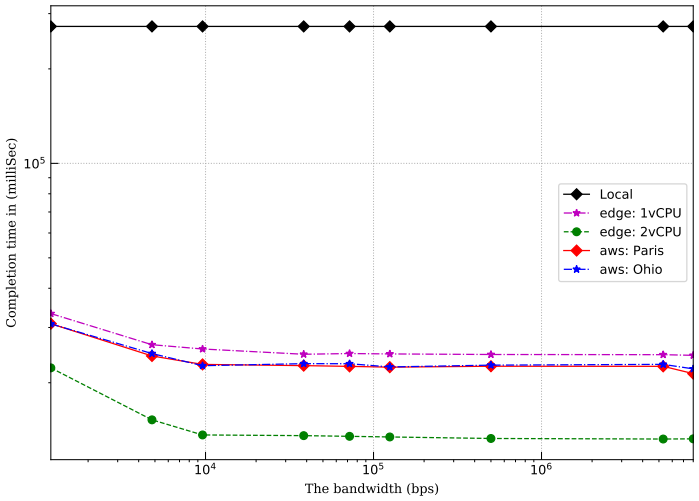


Figure 18: PiBench *full*: completion time

solution fits with the best placement observed *via* experiments in 22 cases among 27 tested configurations. Thus compared to experiments, our proposal have determined the optimal task execution location in 81.4% of the cases. To quantify the under-performance of ECESO in terms of consumed energy, we calculate the relative difference of the experimentally consumed energy among: 1) the case where task's computational location is computed by our proposal ECESO (left column in table 5) 2) the case where optimal placement solution is derived from experiments observations (right column in table 5). Averaging over the 27 possible cases, the under-performance of ECESO in terms of consumed energy with respect to an optimal placement derived from experimental results, is limited to 1.962%.

The near-optimality of ECESO, is also confirmed for Pi-Bench and CPU-Bench. For Pi-Bench, the optimal placement is achieved by ECESO in 85.18% of the studied cases, with an average under-performance of 2.57% of consumed energy. For CPU-Bench optimal placement is obtained by ECESO in 70.37% of the cases with an average under-performance of 3.76% of consumed energy.

Finally, in the last set of experiments, we assessed the effect of multi-users context on offloading performances. We incrementally varied the number of users. However, due to the constraints on the available infrastructure nodes, at the time when we ran our experiments, we fixed the allocated resources both at the cloud and the edge to 1vCPU and we limited the number of mobile terminals to ten. Yet, we believe that the general tendencies, which we will comment hereafter, still hold for larger scale.

To save space, we also choose to discuss for the multi-users context case

Table 5: Offloading decisions for each application: ECESO *vs* Experiments

Bandwidth	ECESO			Experimentation		
	Easy	Medium	Full	Easy	Medium	Full
			Linpack			
1 200	*Local*	*2vCPU*	*2vCPU*	*Local*	*1vCPU*	*1vCPU*
4 800	*Local*	*2vCPU*	*2vCPU*	*Local*	*2vCPU*	*2vCPU*
9 600	*Local*	*2vCPU*	*2vCPU*	*Local*	*2vCPU*	*2vCPU*
38 400	*Local*	*2vCPU*	*2vCPU*	*Local*	*2vCPU*	*2vCPU*
72 000	*Local*	*2vCPU*	*2vCPU*	*Local*	*2vCPU*	*1vCPU*
125 000	*Local*	*2vCPU*	*2vCPU*	*Local*	*2vCPU*	*1vCPU*
500 000	*Local*	*2vCPU*	*2vCPU*	*Local*	*2vCPU*	*2vCPU*
5 300 000	*Local*	*2vCPU*	*2vCPU*	*Local*	*2vCPU*	*2vCPU*
8 000 000	*Local*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*
			PiBench			
1 200	*Local*	*2vCPU*	*2vCPU*	*Local*	*1vCPU*	*2vCPU*
4 800	*2vCPU*	*2vCPU*	*2vCPU*	*Local*	*2vCPU*	*2vCPU*
9 600	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*
38 400	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*
72 000	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*
125 000	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*
500 000	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*
5 300 000	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*
8 000 000	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*
			CPUBench			
1 200	*2vCPU*	*2vCPU*	*2vCPU*	*Local*	*Paris*	*2vCPU*
4 800	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*Paris*	*2vCPU*
9 600	*2vCPU*	*2vCPU*	*2vCPU*	*Paris*	*2vCPU*	*2vCPU*
38 400	*2vCPU*	*2vCPU*	*2vCPU*	*Paris*	*1vCPU*	*1vCPU*
72 000	*2vCPU*	*2vCPU*	*2vCPU*	*Paris*	*2vCPU*	*2vCPU*
125 000	*2vCPU*	*2vCPU*	*2vCPU*	*1vCPU*	*2vCPU*	*2vCPU*
500 000	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*1vCPU*	*1vCPU*
5 300 000	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*
8 000 000	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*	*2vCPU*

only the results of CPUBench. Our choice is motivated by the fact that following the one-user experiments results (see table 5)), CPUBench is the only application in our benchmark for which we observed three offloading decisions: local, edge and cloud.

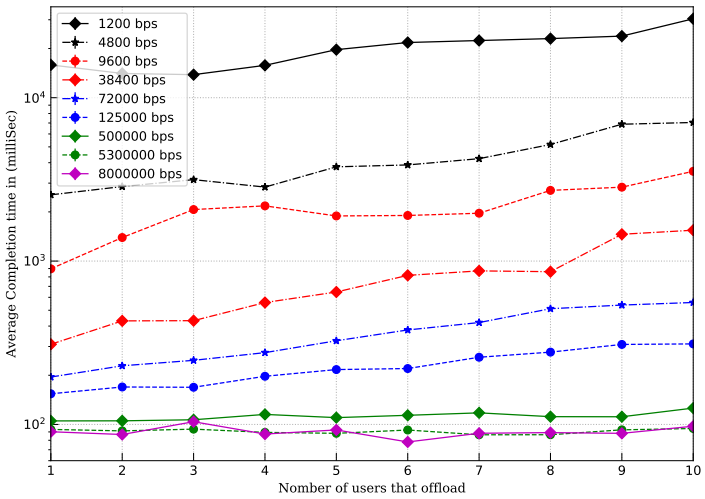


Figure 19: Multi-users completion time of CPUBench *easy* in the cloud

The completion time of CPU-Bench when it is offloaded on the cloud is represented in figures 19, 20, and 21, for *easy*, *medium* and *full* mode, respectively. Each curve in those figures is associated to an access bandwidth, between 1.2 kbps to 8Mbps. The x-axis represents the number of offloading users, while the y-axis represents in logarithmic scale the completion time in milliseconds. One can see that the completion time increases significantly with the number of users, especially for bandwidth capacities less than 500kbps. The contention among several users on the access network increases the transmission delay, especially when the bandwidth capacity is

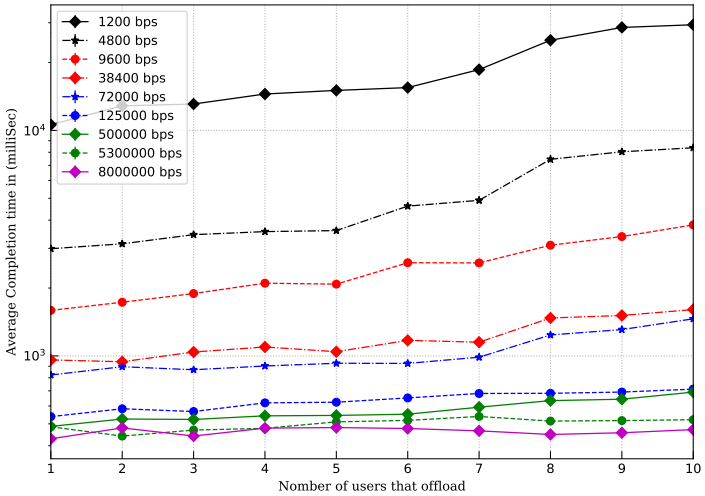


Figure 20: Multi-users completion time of CPUBench *medium* in the cloud

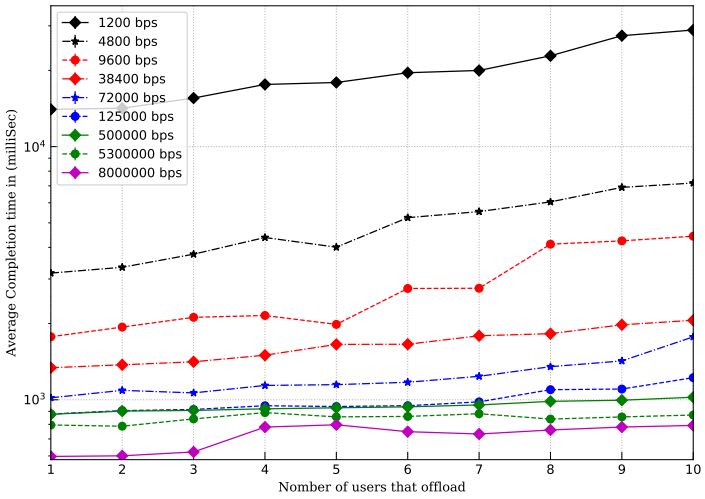


Figure 21: Multi-users completition time of CPUBench *full* in the cloud

quite low. This effect is observed even for CPU-Bench in *full*, despite the fact the processing delay *versus* transmission delay is quite important for this mode. Compared to offloading on the edge, which is shown in figures 22 23 and 24, one can see that the completion time in the edge increases slightly with the number of users, but the slope is much lower than for the cloud.

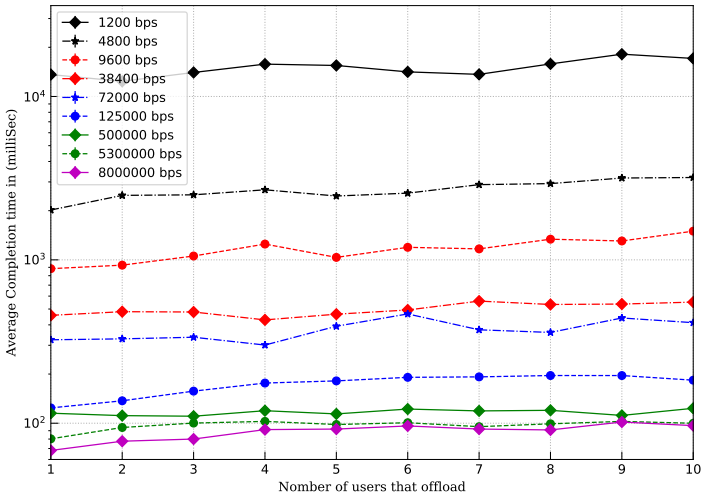


Figure 22: Multi-users completion time of CPU-Bench *easy* in 1vCPU edge

Completion time of 10 concurrent CPU-Bench users with local, edge and cloud execution locations is shown in figures 25, 26, and 27, for *easy*, *medium* and *full* mode, respectively. The x-axis represents in logarithmic scale the access bandwidth, which varies between 1.2 Kbps to 8Mbps, while the y-axis represents in logarithmic scale the completion time in milliseconds.

As shown in figure 25, local execution is the best placement for CPU-Bench in *easy* mode when bandwidth capacity is extremely constrained: less than 9,6Kbps. Indeed, in such condition, the transmission delay is very high

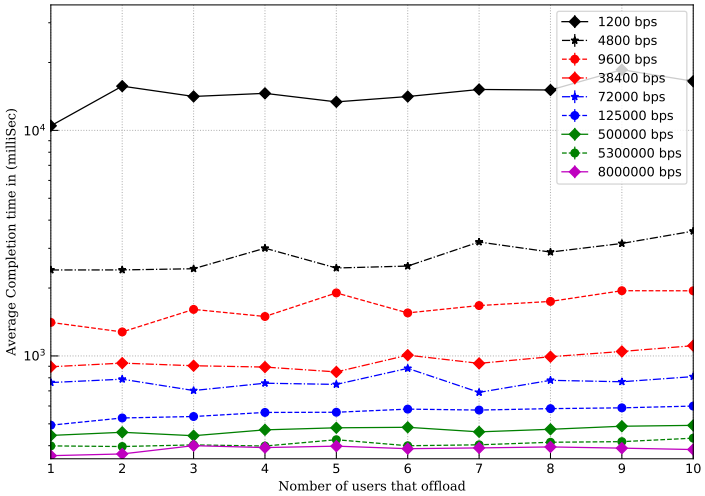


Figure 23: Multi-users completion time of CPU-Bench *medium* in 1vCPU edge

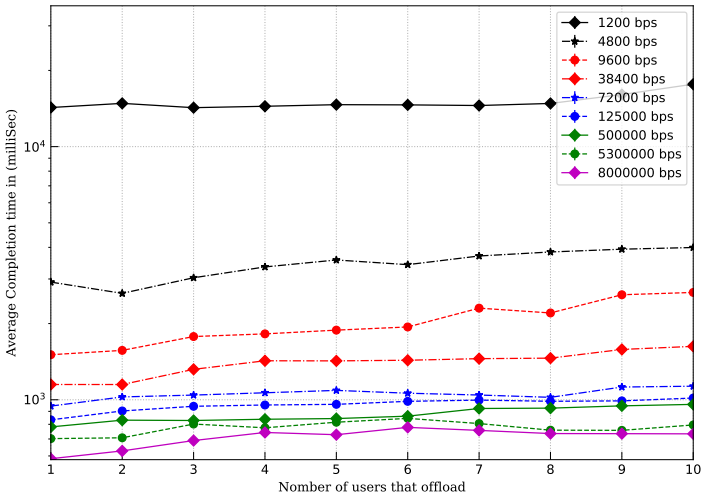


Figure 24: Multi-users completion time of CPU-Bench *full* in 1vCPU edge

and is proportionally much important than the processing delay, even when the application is executed in the edge or the cloud. Faster execution in the edge or in the cloud can not compensate the large transmission time when the access bandwidth is extremely low and shared among many (10) users. For such case, it is not worth it to offload and it is better to execute the task locally on the mobile terminal. For bandwidth capacity larger than 10Kbps, the experiments results show that the best decision is to offload on the edge. This result is interesting because it indicates that even when 10 users are contending on a quite limited bandwidth capacity of about 10Kbps, it still worth it to offload a computational intensive task such as CPU-Bench to the edge. Figures 25, 26, and 27 show that in almost all the cases the best offloading location is the edge. The only exception is for *easy* mode with a bandwidth access set to 5.3Mbps. As shown in figure 8, for higher bandwidth capacity (larger than few Mbps) the transmission delay to the edge is almost similar to the cloud. The difference is less than 10 milliseconds. In these cases, the completion time is almost the same on the edge and on the cloud, especially when the computational resources required by a task are low, which is the case for CPU-Bench in *easy* mode.

In table 6, we compare the best offloading decision of CPU-Bench with 10 users, which we observed through experiments (figures 25, 26, and 27) to the decision obtained by our proposed algorithm ECESO. Each line is associated to a given bandwidth capacity at the access network. Thus combining with the three modes of CPU Bench, we have in total 27 cases. Table 6 show that the decision of ECESO matches with the best placement observed through experiments in 66.7% of the cases. The mismatches are due to the conser-

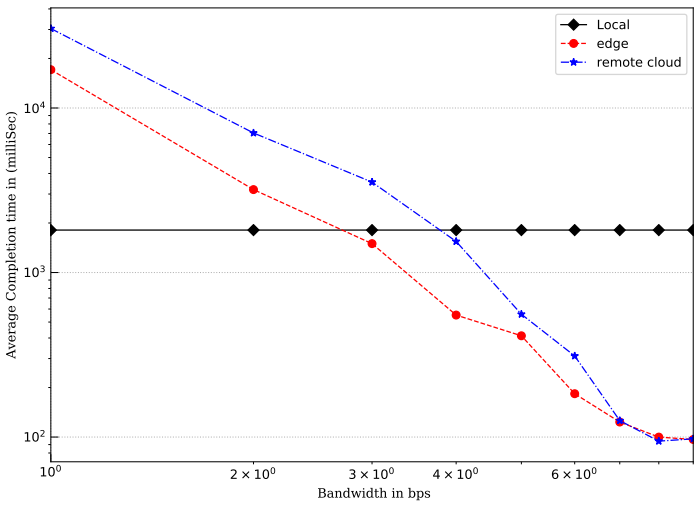


Figure 25: 10 users completion time of CPU-Bench *easy*

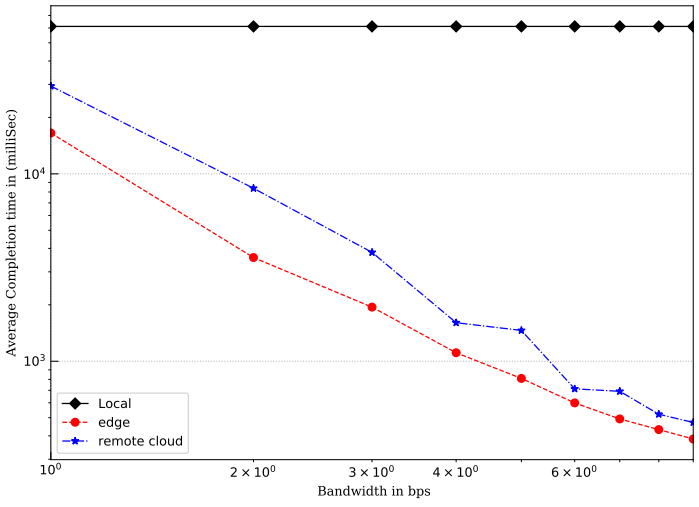


Figure 26: 10 users completion time of CPU-Bench *medium*

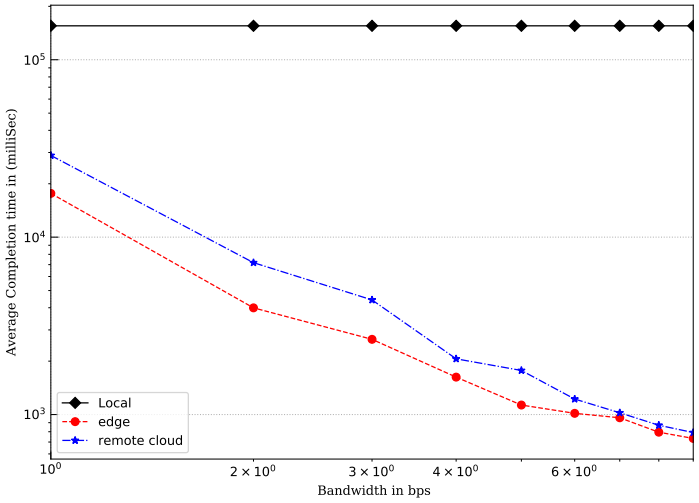


Figure 27: 10 users completion time of CPU-Bench *full*

vative nature of our algorithm. For example, for extremely low bandwidth capacity (low than 3.2 Kbps) ECESO recommends to execute the task locally rather than offload it to the edge. Similarly, for *easy* mode with 5.3 Mbps bandwidth access case that we discussed above, our algorithm suggests to offload the task on the edge, while experiments have shown that offloading on the cloud can also minimize the completion time. The conservative nature of ECESO is mainly inherent to our modelling, which induces some assumptions that appears to be conservative compared to real system behaviors, as observed in experiments.

6. Conclusion

In this paper, we propose a new computation offloading policy in two-tier MEC environment. The proposed offloading policy decides which users

Table 6: Offloading decision for CPUBench with 10 users: ECESO *vs* Experiments

Bandwidth	ECESO			Experimentation		
	Easy	Medium	Full	Easy	Medium	Full
1 200	*Local*	*Local*	*Local*	*Local*	*1vCPU*	*1vCPU*
4 800	*Local*	*Local*	*Local*	*Local*	*1vCPU*	*1vCPU*
9 600	*Local*	*Local*	*Local*	*1vCPU*	*1vCPU*	*1vCPU*
38 400	*Local*	*Local*	*Local*	*1vCPU*	*1vCPU*	*1vCPU*
72 000	*1vCPU*	*1vCPU*	*1vCPU*	*1vCPU*	*1vCPU*	*1vCPU*
125 000	*1vCPU*	*1vCPU*	*1vCPU*	*1vCPU*	*1vCPU*	*1vCPU*
500 000	*1vCPU*	*1vCPU*	*1vCPU*	*1vCPU*	*1vCPU*	*1vCPU*
5 300 000	*1vCPU*	*1vCPU*	*1vCPU*	*Paris*	*1vCPU*	*1vCPU*
8 000 000	*1vCPU*	*1vCPU*	*1vCPU*	*1vCPU*	*1vCPU*	*1vCPU*

should offload, to which offloading server and allocated the wireless bandwidth accordingly. First, we formulate the problem as a Non-Linear Binary Integer Program (NLBIP). Then, we propose an efficient heuristic to solve the problem using Lagrangian decomposition approach. The proposed heuristic uses a branching algorithm to maximize the bandwidth allocation and minimize the energy consumption. The numerical results show performance improvements in terms of the energy consumption compared to existing offloading policies under different scenarios. Finally, we implemented our offloading policy on a real testbed. Using this testbed, we were able to evaluate our offloading decision policy for three real Android OS applications, with different traffic patterns, resource demands and multi-users context.

References

[1] H. Mazouzi, N. Achir, K. Boussetta, Maximizing mobiles energy saving through tasks optimal offloading placement in two-tier cloud, in: Proceedings of the 21st ACM International Conference on Modeling,

Analysis and Simulation of Wireless and Mobile Systems, MSWIM '18, ACM, New York, NY, USA, 2018, pp. 137–145.

[2] M.-H. Chen, B. Liang, M. Dong, Joint offloading decision and resource allocation for multi-user multi-task mobile cloud, in: 2016 IEEE International Conference on Communications (ICC), IEEE, 2016, pp. 1–6.

[3] X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, IEEE/ACM Transactions on Networking 24 (5) (2016) 2795–2808.

[4] S. Guo, B. Xiao, Y. Yang, Y. Yang, Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing, in: 35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016, Vol. 2016-July, IEEE, 2016, pp. 1–9.

[5] S. Guo, J. Liu, Y. Yang, B. Xiao, Z. Li, Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing, IEEE Transactions on Mobile Computing 18 (2) (2019) 319–333.

[6] K. Gai, M. Qiu, H. Zhao, Energy-aware task assignment for mobile cyber-enabled applications in heterogeneous cloud computing, Journal of Parallel and Distributed Computing 111 (2018) 126–135.

[7] M. Jia, J. Cao, W. Liang, Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks, IEEE Transactions on Cloud Computing PP (99).

[8] Z. Xu, W. Liang, W. Xu, M. Jia, S. Guo, Efficient algorithms for capacitated cloudlet placements, IEEE Transactions on Parallel and Distributed Systems 27 (10) (2016) 2866–2880.

[9] H. Yao, C. Bai, M. Xiong, D. Zeng, Z. Fu, Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing, Concurrency and Computation: Practice and Experience 29 (16).

[10] L. Ma, J. Wu, L. Chen, Dota: Delay bounded optimal cloudlet deployment and user association in wmans, in: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, IEEE Press, 2017, pp. 196–203.

[11] A. Mukherjee, D. De, D. G. Roy, A power and latency aware cloudlet selection strategy for multi-cloudlet environment, IEEE Transactions on Cloud Computing PP (99) (2016) 1–14.

[12] D. G. Roy, D. De, A. Mukherjee, R. Buyya, Application-aware cloudlet selection for computation offloading in multi-cloudlet environment, The Journal of Supercomputing (2016) 1–19.

[13] M. Jia, W. Liang, Z. Xu, M. Huang, Cloudlet load balancing in wireless metropolitan area networks, in: Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on, Vol. 2016-July, 2016, pp. 1–9.

[14] H. Mazouzi, N. Achir, K. Boussetta, Dm2-ecop: An efficient computation offloading policy for multi-user multi-cloudlet mobile edge com-

puting environment, ACM Trans. Internet Technol. 19 (2) (2019) 24:1–24:24.

[15] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, Maui: making smartphones last longer with code offload, in: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys 2010), San Francisco, California, USA, June 15-18, 2010, ACM, pp. 49–62.

[16] R. C. Martin, Agile software development: principles, patterns, and practices, Prentice Hall, 2002.

[17] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, Clonecloud: elastic execution between mobile device and cloud, in: European Conference on Computer Systems, Proceedings of the Sixth European conference on Computer systems, EuroSys 2011, Salzburg, Austria, April 10-13, 2011, ACM, pp. 301–314.

[18] J. I. Benedetto, G. Valenzuela, P. Sanabria, A. Neyem, J. Navon, C. Poellabauer, Mobicop: A scalable and reliable mobile code offloading solution, Wireless Communications and Mobile Computing 2018.

[19] Y. Gao, W. Hu, K. Ha, B. Amos, P. Pillai, M. Satyanarayanan, Are cloudlets necessary?, School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-15-139.

[20] I. Tinnirello, G. Bianchi, Y. Xiao, Refinements on ieee 802.11 distributed coordination function modeling approaches, IEEE Transactions on Vehicular Technology 59 (3) (2010) 1055–1067.

[21] G. Bianchi, Performance analysis of the ieee 802.11 distributed coordination function, IEEE Journal on selected areas in communications 18 (3) (2000) 535–547.

[22] A. Carroll, G. Heiser, et al., An analysis of power consumption in a smartphone., in: USENIX annual technical conference, Vol. 14, Boston, MA, 2010, pp. 21–21.

[23] S. O. Krumke, C. Thielen, The generalized assignment problem with minimum quantities, European Journal of Operational Research 228 (1) (2013) 46–55.

[24] J. Tang, C. Yan, X. Wang, C. Zeng, Using lagrangian relaxation decomposition with heuristic to integrate the decisions of cell formation and parts scheduling considering intercell moves, IEEE Transactions on Automation Science and Engineering 11 (4) (2014) 1110–1121.

[25] M. Thomas, E. W. Zegura, Generation and analysis of random graphs to model internetworks, Tech. rep., Georgia Institute of Technology (1994).

[26] J. Oueis, E. Calvanese-Strinati, A. De Domenico, S. Barbarossa, On the impact of backhaul network on distributed cloud computing, in: Wireless Communications and Networking Conference Workshops (WCNCW), 2014 IEEE, IEEE, 2014, pp. 12–17.

[27] OSBoxes, Android x86.
URL https://www.osboxes.org/android-x86/

[28] L. López, F. J. Nieto, T.-H. Velivassaki, S. Kosta, C.-H. Hong, R. Montella, I. Mavroidis, C. Fernández, Heterogeneous secure multi-level re-

mote acceleration service for low-power integrated systems and devices, Procedia Computer Science 97 (2016) 118–121.

[29] F. Foukalas, Y. Ntarladimas, A. Glentis, Z. Boufidis, Protocol reconfiguration using component-based design, in: IFIP International Conference on Distributed Applications and Interoperable Systems, Springer, 2005, pp. 148–156.