First Hop Mobile Offloading of DAG Computations

Vincenzo De Maio, Ivona Brandic
Vienna University of Technology
Institute of Software Technology and Interactive Systems
Favoritenstrasse 9-11/194, 1040 Vienna, Austria
Email: {vincenzo, ivona}@ec.tuwien.ac.at

***Abstract*—In recent years, Mobile Cloud Computing (MCC) has been proposed to increase battery lifetime of mobile devices. However, offloading on Cloud infrastructures may be infeasible for latency critical applications, due to the geographical distribution of Cloud data centers that increases offloading time. In this paper, we investigate the use of Mobile Edge Cloud Offloading (MECO), namely offloading to a heterogeneous computing infrastructure featuring both Cloud and Edge nodes, where Edge nodes are geographically closer to the mobile device. We evaluate improvements of MECO in comparison with MCC for objectives such as applications' runtime, mobile device battery lifetime and cost for the user. Afterwards, we propose the Edge Cloud Heuristic Offloading (ECHO) approach to find a trade-off solution between the aforementioned objectives, according to user's preferences. We evaluate our approach by simulating offloading of Directed Acyclic Graphs (DAGs) representing mobile applications through the use of Monte-Carlo simulations. The results show that (1) MECO can reduce application runtime by up to 70.7% and cost by up to 70.6% in comparison to MCC and (2) ECHO allows user to select a trade-off solution with at most 18% MAPE for runtime, 16% for cost and 0.5% for battery lifetime, according to user's preferences.**

***Index Terms*—Edge Computing; Mobile offloading; DAG scheduling; Heuristics; Monte-Carlo simulations.**

I. Introduction

Mobile devices are becoming more popular in the last years, thanks to their increasing hardware capabilities and faster processors. Consequently, mobile applications become more complex and provide an increasing amount of services. However, this complexity requires an increasing amount of processing power and energy. Since such devices have limited battery capacity, it is important to reduce energy consumption of mobile applications to increase devices' battery lifetime and consequently improve user experience.

In recent years, computation offloading has been proposed as a solution to this problem [1], [2]. Computation offloading pursues energy saving on mobile devices by offloading computation to a remote computing infrastructure. Results of the computation are then downloaded from this infrastructure to the mobile device. We refer to this technique as Mobile Cloud Computing (MCC) when offloading on a Cloud infrastructure.

Many works like [3], [4], [5], [6] discussed MCC. However, most of these works do not focus on latency critical applications. Since latency is strongly affected by geographical distance, a significant slowdown could be experienced when offloading these applications to the Cloud, due to the geographical distribution of Cloud data centers. This can prevent users offloading to Cloud and therefore affect provider's profit.

We discuss Mobile Edge Cloud Offloading (MECO) as a solution to this problem. In this approach, offloading latency is reduced by exploiting Edge nodes, geographically closer to the user. We propose a model for MECO, considering parameters such as application running time, battery lifetime and user cost. Based on this model, we define a simulation framework for MECO and use it to simulate offloading of Directed Acyclic Graphs (DAGs) representing mobile applications, showing improvements for the aforementioned objectives in comparison to MCC. Finally, we propose Edge Cloud Heuristic Offload (ECHO), an heuristic offloading approach to select a trade-off solution for these objectives, according to user's preferences.

Our evaluation uses different DAG models of mobile applications, used in works like [7], [8] to model computations. We perform our evaluation using Monte-Carlo simulations, as this method allows to accurately model the variability of the underlying Cloud/Edge infrastructure and the mobile devices.

Our results show that MECO can reduce running time of mobile applications in comparison to MCC by up to 70.7% and reduce cost by up to 70.6% with a 3% increase of battery lifetime. Also we show that ECHO can find a trade-off solution for different types of applications, with at most 18% MAPE (Mean Absolute Percentage Error) for runtime, 16% for cost and 0.5% for battery lifetime, according to user's preferences.

The paper is organized as follows: we provide a background for MECO in Section II. Then, we define our theoretical model in Section III and ECHO approach in Section IV. Simulation framework and experimental setup are defined in Section V. Results are discussed in Section VI. Finally, we discuss related work in Section VII and conclude the paper in Section VIII.

II. Background

A. Mobile applications

Mobile applications can be executed either on a mobile device or offloaded to a computational infrastructure. Applications are composed of different interdependent tasks, each one with different requirements. Dependencies and task requirements can affect task offloading: for example, task could not be offloaded because it needs specific devices, not available on the remote infrastructure (e.g. camera or GPS), or because task execution depends on another task's results. Both these issues are discussed in Section III-A.

B. Mobile offloading

With mobile offloading, a mobile application (or part of it) is offloaded from a mobile device to a remote computational infrastructure. In Figure 1, we summarize the offloading process. We assume that offloading is performed by a software component running on the mobile device called *offloading engine*. Offloading engine decides which tasks to offload and which tasks to execute on the mobile device, based on the application's and infrastructure's data collected during execution. Once offloading terminates, the infrastructure executes tasks and sends the results to the mobile device. This process repeats until the application terminates.

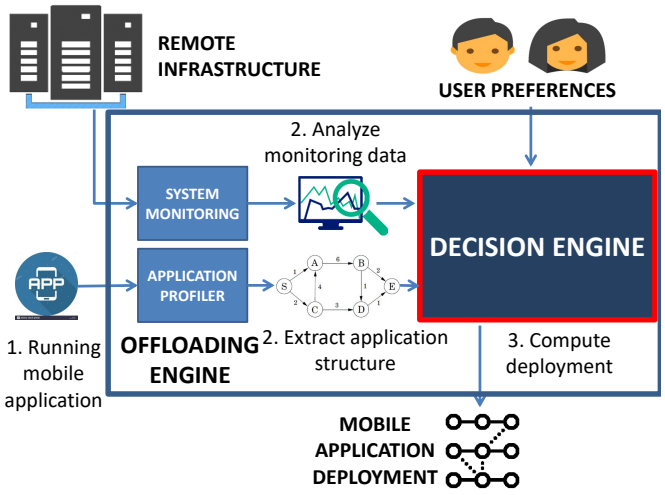


Fig. 1: Offloading model.

C. Offloading engine

We identify the main components of offloading engine in Figure 1: (1) *Application profiler* profiles the application, extracting DAG structure and identifying tasks' requirements and dependencies; (2) *System monitoring* is monitoring data about remote infrastructure, and (3) *Decision engine*, collects data from the other two and performs offloading decisions. In this work, we focus on the decision engine. We assume that decision engine has already collected the application DAG model annotated with requirements and offloading possibility for each task from the Application profiler, as well as the information about remote infrastructure from System monitoring. Application and infrastructure model are described respectively in Sections III-A and III-B. Concerning user preferences, we identify the following objectives: (1) *Application runtime*, the running time of the mobile application; (2) *Battery lifetime*, the lifetime of mobile device battery, and (3) *Cost*, the additional cost charged to user for offloading application to the remote infrastructure. Objectives are described in Section III-D.

III. Model

A. Application model

Application modeling requires representing task requirements and interdependencies between the tasks. In [3], [9], mobile applications are defined as a graph where nodes are the tasks and edges represent the dependencies between them. Therefore, we define a mobile application as a DAG $\mathcal{A} = \{\mathcal{T}_a, \mathcal{L}_a\}$, where $\mathcal{T}_a$ defines the set of all the application's tasks and $\mathcal{L}_a \subseteq \mathcal{T}_a \times \mathcal{T}_a$ defines the dependencies between tasks. Tasks are defined as in Definition 1:

Definition 1. *A task* $t_i \in \mathcal{T}_a$ *is defined as* $t_i = (\texttt{MI}, \texttt{RAM}, \texttt{DATA}_{in}, \texttt{DATA}_{out}, \texttt{OFF})$ *where* MI *are the millions of instructions of the task,* RAM *the amount of memory required by the task,* $\texttt{DATA}_{in}$ *and* $\texttt{DATA}_{out}$ *respectively the size of task* t_i *input data and the size of* t_i *output and* $\texttt{OFF} \in \{0, 1\}$ *where* $\texttt{OFF} = 1$ *if* t_i *can be offloaded,* 0 *otherwise.*

By $\texttt{MI}(t_i)$, $\texttt{RAM}(t_i)$, $\texttt{DATA}_{in}(t_i)$, $\texttt{DATA}_{out}(t_i)$, $\texttt{OFF}(t_i)$ we denote, respectively, millions of instructions, required RAM, size of input and output data and offloading possibility of task t_i. We also define $\texttt{STORAGE}(t_i) = \texttt{DATA}_{in}(t_i) + \texttt{DATA}_{out}(t_i)$ as the amount of storage required by the task t_i. Links define precedence relationships between tasks, namely,

Definition 2. *A edge* $l_{i,j} \in \mathcal{L}_a$ *is a couple* $l_{ij} = (t_i, t_j)$*, where (1)* $t_i, t_j \in \mathcal{T}_a$*. and (2)* $\exists\, l_{ij} \implies t_i$ *is executed before* t_j*.*

For each l_{ij} we define $\texttt{latency}(l_{ij})$ as the maximum acceptable latency between tasks. Also, we define bandwidth requirements for l_{ij} as $\texttt{bw}(l_{ij})$. Finally, we define $\delta_{in}(t_i, \tau)$ and $\delta_{out}(t_i, \tau)$ respectively as the set of t_i incoming/outgoing edges at time instant τ.

B. Infrastructure model

The type of infrastructure that we define in this work is a Mobile Edge Computing (MEC) infrastructure.

Definition 3. *We define a MEC infrastructure as* $\mathcal{I} = \{\mathcal{N}_\mathcal{I}, \mathcal{L}_\mathcal{I}\}$*, where* $\mathcal{N}_\mathcal{I}$ *is the set of computational nodes and* $\mathcal{L}_\mathcal{I}$ *the network connections between the nodes.*

$\mathcal{N}_\mathcal{I}$, is defined as $\mathcal{N}_\mathcal{I} = \{\mathcal{C}_\mathcal{I}, \mathcal{E}_\mathcal{I}, md\}$, respectively the set of Cloud nodes, the set of Edge nodes and the mobile device md. We define the mobile device md as $md = (\texttt{CORES}, \texttt{MIPS}, \texttt{RAM}, \texttt{STORAGE}, \texttt{BATTERY}, \texttt{coords})$, where $\texttt{CORES}(md)$ is the number of cores of the device, $\texttt{MIPS}(md)$ represents the millions of instruction per second that each core of md can execute, $\texttt{RAM}(md)$ is the amount of RAM of the device, $\texttt{STORAGE}(md)$ is the amount of storage of the device, $\texttt{BATTERY}(md)$ the amount of energy of mobile device battery and $\texttt{coords}(md)$ the GPS coordinates of the mobile device. We define a Cloud node $c_i \in \mathcal{C}_\mathcal{I}$ as $c_i = (\texttt{CORES}, \texttt{MIPS}, \texttt{RAM}, \texttt{STORAGE})$, where $\texttt{CORES}(c_i)$ is the number of cores available on the node, $\texttt{MIPS}(c_i)$ is the million of instruction per second that c_i can execute, $\texttt{RAM}(c_i)$ is the amount of RAM of c_i, $\texttt{STORAGE}(c_i)$ is the amount of storage available on c_i. Edge computing moves computation to Edge nodes, geographically closer to the user, reducing latency of moving data to geographically distributed Cloud data centers across the Internet. Therefore, the main characteristics of Edge nodes are (1) a limited amount of resource compared to the cloud nodes, and (2) geographical proximity to the user, in this

case, the mobile device [10]. Therefore, we define an Edge node $e_i \in \mathcal{E}_\mathcal{I}$ as $e_i = (\texttt{CORES}, \texttt{MIPS}, \texttt{RAM}, \texttt{STORAGE}, \texttt{coords})$, where $\texttt{CORES}(e_i)$ is the number of cores available on the node, $\texttt{MIPS}(e_i)$ represents the millions of instruction per second that each core in e_i can execute, $\texttt{RAM}(e_i)$ is the amount of RAM available in e_i, $\texttt{STORAGE}(e_i)$ is the amount of storage available on the node and $coords(e_i)$ the GPS coordinates of the Edge node. This last parameter will be used to decide whether the Edge node is reachable by the mobile device. Regarding $\mathcal{L}_\mathcal{I}$, $\mathcal{L}_\mathcal{I} = \mathcal{N}_\mathcal{I} \times \mathcal{N}_\mathcal{I}$. For each link $l_{ij} \in \mathcal{L}_\mathcal{I}$ we define the latency between n_i and n_j $\texttt{latency}(n_i, n_j)$, where $\texttt{latency}(n_i, n_i) = 0$, and the bandwidth of the link $\texttt{bw}(n_i, n_j)$, where $\texttt{bw}(n_i, n_i) = \infty$.

C. Deployment

We define the deployment of tasks to the infrastructure's components and its relative constraints. Deployment of an application $\mathcal{A}$ over an infrastructure $\mathcal{I}$ is a mapping of the tasks in $\mathcal{T}_a$ to the nodes in $\mathcal{N}_\mathcal{I}$ and of the edges in $\mathcal{L}_a$ to the physical links in $\mathcal{L}_\mathcal{I}$. A task can be executed either on a mobile device or offloaded to a Cloud/Edge node. Since dependencies between tasks impose an order on their execution, deployment of $\mathcal{A}$ over $\mathcal{I}$ is performed in different time steps. In each time step we execute only tasks $t_i \in \mathcal{T}_a$ such that $\delta_{in}(t_i, \tau) = \emptyset$. Such tasks are defined as *ready*. We define as $\mathcal{T}_a(\tau)$ the set of tasks in $\mathcal{T}_a$ that are ready at time step τ. We define the *partial* deployment of tasks $\mathcal{T}_a(\tau)$ as a set $\mathcal{D}_\tau$ of pairs (t_i, n_j), with $\mathcal{D}_\tau \subseteq \mathcal{T}_a(\tau) \times \mathcal{N}_\mathcal{I}$, such that

$$(t_i, n_j) \in \mathcal{D}_\tau \iff t_i \text{ is allocated to node } n_j. \quad (1)$$

More tasks can be assigned to a computational node, as soon as node capacity constraints are respected. We define $\mathcal{D}_\tau(n_i)$ as the set of tasks mapped to node n_i at instant τ, namely

$$t_i \in \mathcal{D}_\tau(n_j) \iff \exists j : (t_i, n_j) \in \mathcal{D}_\tau. \quad (2)$$

We define a *valid* partial deployment for application $\mathcal{A}$ on infrastructure $\mathcal{I}$, $\mathcal{D}_\tau(\mathcal{A}, \mathcal{I})$. A partial deployment is valid only if (1) all the tasks are deployed only once on the infrastructure and (2) all deployments of tasks on nodes satisfy the capacity constraints of the nodes, namely:

Definition 4. *A deployment $\mathcal{D}_\tau(\mathcal{A}, \mathcal{I})$ of application $\mathcal{A}$ on infrastructure $\mathcal{I}$ is* valid $\iff$

1) $\bigcup_{n_j \in \mathcal{N}_\mathcal{I}} \mathcal{D}(n_j) = \mathcal{T}_a(\tau)$;
2) $(t_i, n_j) \in \mathcal{D}_\tau(\mathcal{A}, \mathcal{I}), \texttt{OFF}(t_i) = 0 \implies n_j = m_d$;
3) $(t_i, n_j) \in \mathcal{D}_\tau(\mathcal{A}, \mathcal{I}) \iff$
   a) $\sum_{t_i \in \mathcal{D}_\tau(n_j)} \texttt{CORES}(t_i) \leq \texttt{CORES}(n_i)$;
   b) $\sum_{t_i \in \mathcal{D}_\tau(n_j)} \texttt{RAM}(t_i) \leq \texttt{RAM}(n_i)$;
   c) $\sum_{t_i \in \mathcal{D}_\tau(n_j)} \texttt{STORAGE}(t_i) \leq \texttt{STORAGE}(n_i)$;
   d) $\bigcup_{t_i \in \mathcal{D}_\tau(n_j)} \delta_{in}(t_i, \tau) \subseteq \delta_{in}(n_i, \tau)$;
   e) $\bigcup_{t_i \in \mathcal{D}_\tau(n_j)} \delta_{out}(t_i, \tau) \subseteq \delta_{out}(n_i, \tau)$;
   f) $\forall (n_i, n_j) \in \delta_{in}(t_i, \tau)$ $\texttt{latency}(n_i, n_j) \geq \texttt{latency}(n_i, n_j)$, $(n_i, n_j) \in \delta_{in}(n_i, \tau)$;
   g) $\forall (n_i, n_j) \in \delta_{out}(t_i, \tau)$ $\texttt{latency}(n_i, n_j) \geq \texttt{latency}(n_i, n_j)$ $(n_i, n_j) \in \delta_{out}(n_i, \tau)$.

We also define a function ϕ that returns the computational node where a task t_i is executed. Formally,

$$\phi(t_i) = n_j : \exists (t_i, n_j) \in \mathcal{D}(\mathcal{A}, \mathcal{I}). \quad (3)$$

Once execution of a task t_i terminates at a given instant $\tau(t_i)$, it is removed from the DAG and a partial deployment of the tasks in the set $\mathcal{T}_a(\tau(t_i))$ is performed. $\mathcal{T}_a(\tau(t_i))$ is obtained removing from $\mathcal{L}_a$ all the edges in $\delta_{out}(t_i, \tau)$ and adding to $\mathcal{T}_a(\tau(t_i))$ all the tasks $t_i \in T_a$ such that $\delta_{in}(t_i, \tau) = \emptyset$. Deployment is complete when $\mathcal{T}_a = \emptyset$.

We define a complete deployment $\mathcal{D}(\mathcal{A}, \mathcal{I})$ as the union of all partial deployments, namely

$$\mathcal{D}(\mathcal{A}, \mathcal{I}) = \bigcup_{\tau = [0, \tau_{end}]} \mathcal{D}_\tau(\mathcal{A}, \mathcal{I}), \quad (4)$$

where τ_{end} is the time when $\mathcal{T}_a = \emptyset$.

D. Problem definition

Our goal is to find a valid deployment $\mathcal{D}(\mathcal{A}, \mathcal{I})$ that optimizes for different objectives. According to objectives selected in Section II-C, we define offloading in Equation 5.

$$\begin{cases} \min RT(\mathcal{D}(\mathcal{A}, \mathcal{I})) \\ \max BL(\mathcal{D}(\mathcal{A}, \mathcal{I})) \\ \min UC(\mathcal{D}(\mathcal{A}, \mathcal{I})) \\ \text{With } \mathcal{D}(\mathcal{A}, \mathcal{I}) s.t. \text{Definition 4} \end{cases} \quad (5)$$

Functions $RT(\mathcal{D}(\mathcal{A}, \mathcal{I}))$, $BL(\mathcal{D}(\mathcal{A}, \mathcal{I}))$ and $UC(\mathcal{D}(\mathcal{A}, \mathcal{I}))$ are defined in Equations 6, 11 and 17, respectively

1) Running time: The first goal is to minimize the application runtime, that we define as

$$RT(\mathcal{D}(\mathcal{A}, \mathcal{I})) = \tau_{end}, \quad (6)$$

where τ_{end} is the time when $\mathcal{T}_a = \emptyset$. For each task t_i, $\tau(t_i)$ depends on the running time of t_i on a node n_i, namely $RT_{local}(t_i, n_i)$, and the time for offloading task t_i to node n_i, $OT(t_i, n_i)$. We define $RT_{local}(t_i, n_i)$ as follows:

$$RT_{local}(t_i, n_i) = \frac{\texttt{MI}(t_i)}{\texttt{MIPS}(n_i)} \quad (7)$$

Offloading time OT_{up} depends on data transferred to $\phi(t_i)$ the bandwidth available between mobile device md and $\phi(t_i)$. To offload t_i, we need to transfer the binary and the input data $\texttt{DATA}_{in}(t_i)$. We assume that the binary size is proportional to the number of instructions. Therefore, we define OT_{up} as

$$OT_{up}(t_i) = \frac{\texttt{MI}(t_i) \cdot \texttt{instr\_size} + \texttt{DATA}_{in}(t_i)}{bw(md, \phi(t_i))}. \quad (8)$$

We assume $\texttt{instr\_size} = 5 \times 10^{-9}$ (https://www.strchr.com/x86_machine_code_statistics). Conversely, OT_{down} is the time to download results from the target node $\phi(t_i)$:

$$OT_{down}(t_i) = \frac{\texttt{DATA}_{out}(t_i)}{bw(md, n_j)}, \quad (9)$$

finally, we define the running time of a task t_i as

$$RT(t_i) = \tau(t_i) + OT_{up}(t_i) + RT_{local}(t_i, \phi(t_i)) + OT_{down}(t_i), \quad (10)$$

where $\tau(t_i)$ is the instant at which the task t_i is deployed. If task is executed on the mobile device, $RT(t_i) = RT_{local}(m_i, n_i)$, as $bw(md, md) = \infty$ (See Section III-B).

2) Battery lifetime: Battery lifetime is defined as the percentage of energy budget left on device md at the end of the execution, namely

$$BL(\mathcal{D}(\mathcal{A},\mathcal{I})) = \frac{\texttt{BATTERY}(md) - E_d(md, \mathcal{D}(\mathcal{A},\mathcal{I}))}{\texttt{BATTERY}(md)}. \quad (11)$$

Energy is defined as the integral of the instantaneous power over time. Therefore, we define energy consumption on the mobile device md for deployment $\mathcal{D}(\mathcal{A},\mathcal{I})$.

$$E_d(md, \mathcal{D}(\mathcal{A},\mathcal{I})) = \int_0^{RT(\mathcal{D}(\mathcal{A},\mathcal{I}))} \sum_{t_i \in \mathcal{D}(md,\tau)} P_p(t_i,\tau) + \\ + \sum_{t_i:\phi(t_i)\neq md} P_{\texttt{off}}(t_i, \phi(t_i), \tau)\ d\tau. \quad (12)$$

Where $P_p(md,\tau)$ is the instantaneous power draw for processing on mobile device md at the instant t, $P_{\texttt{off}}(t_i,\tau)$ is the instantaneous power draw for offloading task.

a) Mobile device energy consumption model: For energy consumption of mobile device, we employ the CPU model designed by [11], described by Equation 13:

$$P_p(md,\tau) = \sum_{i=0}^{i<\texttt{CORES}(md)} \beta_{\texttt{freq}}(i,\tau) \cdot \mathcal{U}_{cpu}(md,\tau) + \beta_{\texttt{base}}, \quad (13)$$

where $\beta_{freq}(i,\tau)$ is a constant dependent on the frequency of core i at the instant τ, β_{base} is a hardware dependent constant and $\mathcal{U}_{cpu}(md,\tau)$ is the CPU utilization of device md at the time τ, as defined by Equation 14.

$$\mathcal{U}_{cpu}(n,\tau) = \frac{\sum_{t_i \in \mathcal{D}(n)} \texttt{MI}(t_i,\tau)}{\texttt{MIPS}(n)} \quad (14)$$

b) Energy consumption for offloading: According to [12], energy consumption of network transfer has a linear relationship with the time required to perform the transfer of the task t_i on the node n_j. Such relationship depends on the type of connection between the nodes, the bandwidth available on the link and if the node is an Edge or a Cloud node. We employ Equation 15 for energy consumption of offloading:

$$P_{off}(t_i, n_j, \tau) = \epsilon_{conn}(n_j) \cdot \mathcal{U}_{net}(t_i, n_j, \tau) + \texttt{K}_{conn}(n_i), \quad (15)$$

where $\epsilon_{conn}(n_i)$ models the relationship between network utilization and instantaneous power draw and $\mathcal{U}_{net}(t_i, n_j, \tau)$ is the utilization of network at time instant τ, namely

$$\mathcal{U}_{net}(t_i, n_j, \tau) = \frac{\texttt{DATA}_{net}(t_i,\tau)}{bw(md, n_j)}, \quad (16)$$

where $\texttt{DATA}_{net}(t_i,\tau)$ is the amount of data offloaded at time instant τ. Time for transmission between md and n_j is calculated as $\frac{\texttt{DATA}_{in,out}}{bw(md,n_j)} + \texttt{latency}(md, n_j)$. $\texttt{K}_{conn}(n_i)$ represents a hardware related constant.

3) User cost model: The cost for user depends on where tasks are executed, the amount of resources used and the running time. User cost for a deployment $\mathcal{D}(\mathcal{A},\mathcal{I})$ is defined as the sum of the costs each task execution, as in

$$UC(\mathcal{D}(\mathcal{A},\mathcal{I})) = \sum_{t_i \in \mathcal{A}} uc(t_i). \quad (17)$$

At the time we write, finding a pricing strategy for Edge computing is seen as an open research challenge by [13] and typical pricing strategies used for Cloud are not applicable to the Edge contexts [14]. Therefore, we define our cost model for Edge and Cloud as follows: if a task t_i is offloaded on Cloud, the user pays the price for Cloud resources during the running time of t_i. If the task is offloaded on Edge, the user will pay the price they would pay on the Cloud, plus a additional quantity, defined by a function p_e, for executing task on Edge. Such increment is because execution on Edge will reduce the latency, increasing the value perceived by the user. Also, providers can ask an higher price for execution on Edge, due to the additional cost for deploying nodes in proximity of the user. Therefore, this additional p_e should maximize both provider's revenue and user satisfaction. We define then the cost for executing task t_i as

$$uc(t_i) = \begin{cases} 0,\ \phi(t_i) = md \\ p(\phi(t_i)) \cdot RT(t_i),\ \phi(t_i) \in \mathcal{C}_\mathcal{I} \\ p(\phi(t_i)) \cdot RT(t_i) + p_e(\phi(t_i), \eta),\ \phi(t_i) \in \mathcal{E}_\mathcal{I} \end{cases} \quad (18)$$

Where p_e depends on a η parameter that models if user prefers a lower latency or a cheaper price. We define the $p_e(v_i, \eta)$ function in Equation 19.

$$p_e(t_i, \eta) = \frac{T_f(t_i)}{\eta} - \sqrt{\frac{\eta \cdot p(\mathcal{M}(t_i)) + T_f}{\eta^2 \cdot \min_{n_j \in \mathcal{E}_\mathcal{I}} RT(t_i, n_j)}}, \quad (19)$$

where $T_f = \sum_{n_i \in \mathcal{C}_\mathcal{I}} \frac{\texttt{latency}(md,n_i)}{|\mathcal{C}_\mathcal{I}|} + \frac{1}{\texttt{CORES}(n_i)} - \sum_{n_k \in \mathcal{E}_\mathcal{I}} \frac{\texttt{latency}(md,n_k)}{|\mathcal{E}_\mathcal{I}|}$ and η is a value between 0.01 and 1, where a value closer to 0.01 means that user prefers to have lower latency, while a value closer to 1 indicates that user prefers price over latency. In [15] it is shown that this function maximizes both providers' revenue and users' satisfaction, making it a possible pricing model for Edge.

IV. ECHO APPROACH

Finding a deployment for application $\mathcal{A}$ on infrastructure $\mathcal{I}$ is an optimization problem with three objectives, as defined in Equation 5. Solutions to this type of problems can be found with multiobjective metaheuristics, such as MOPSO [16] and NSGA-II [17], or list scheduling approaches like [18]. However, here we propose ECHO (Edge Cloud Heuristic Offloading), a heuristic based offloading approach similar to works like [19]. This choice is due to the shorter computation time in comparison to aforementioned approaches, that makes it more suitable for offloading of latency sensitive applications. Deployment process is summarized in Section III-C. The main activity of this process is selecting a target for the execution

Algorithm 1 ECHO Approach

```
1: function ECHO(t, α, β, γ)
2:     compNodes ← compatibleNodes(t)
3:     R̂T ← ∞, ÛC ← ∞, B̂T ← −∞
4:     for n ∈ compNodes do
5:         if RT(t, n) < R̂T then
6:             R̂T ← RT(t, n)
7:         end if
8:         if UC(t, n) < ÛC then
9:             ÛC ← uc(t, n)
10:        end if
11:        if BL(t, n) > B̂L then
12:            B̂L ← BL(t, n)
13:        end if
14:    end for
15:    mS ← ∞
16:    for n ∈ compNodes do
17:        if score(n, s, R̂T, ÛC, B̂L, α, β, γ) < mS then
18:            mS ← score(n, s, R̂T, ÛC, B̂L, α, β, γ)
19:            target ← n
20:        end if
21:    end for
22: return target
23: end function
```

of a task. The selected target should be capable of optimizing for all the objectives defined in Section III-D. The idea behind ECHO is to find a target n minimizing

$$score(n, s, \hat{RT}, \hat{UC}, \hat{BL}, \alpha, \beta, \gamma) = \alpha(RT(t,n) - \hat{RT}(t)) + \beta(UC(t,n) - \hat{UC}(t)) + \gamma(\hat{BL}(t) - BL(t,n)), \quad (20)$$

where $\hat{RT}(t)$, $\hat{UC}(t)$ and $\hat{BT}(t)$ represent the local optimum value for each objective, namely the minimum runtime, the minimum user cost and the maximum battery lifetime. α, β and γ are user-defined parameters defining how much the user values that specific objective. The heuristic is summarized in Algorithm 1. In lines from 4 to 14, we iterate over the compatible nodes (the nodes that are capable of hosting the task t) to find $\hat{RT}(t)$, $\hat{UC}(t)$ and $\hat{BL}(t)$. Then, we select among the compatible nodes the one with the lowest score (lines 16-21). Score is calculated according to Equation 20.

V. Experimental setup

A. Simulation framework

The evaluation of different scenarios is performed through simulations, because of the unavailability of real-world Edge infrastructure to perform our experiments. At the time we write, several simulation frameworks for Edge/Fog computing has been proposed, such as iFogSim [20] and EdgeCloudSim [21]. However, they either do not support cyber-foraging or do not allow us to specify our energy and cost model. Also, Monte-Carlo simulations are more suited for the variability of the target environment, according to [22], and have several applications on DAG scheduling, such as [7], [23]. For this reason, we decided to extend FogTorchPI [24] to perform our simulations. FogTorchPI is a Monte-Carlo simulation framework for selection of the best deployment for an application on a Fog infrastructure, according to parameters such as application requirements and QoS provided by infrastructure. To perform our simulation, we developed an extended version of FogTorchPI based on the model defined in Section III. We (1) added support for mobile devices, (2) added an energy consumption model for mobile devices, (3) included our cost model, (4) added support for DAG scheduling and (5) included our offloading model. The extended version of the framework is available online (https://bitbucket.org/vindem/fogtorchpi-extended). The input of our Monte-Carlo simulation is the infrastructure setup, consisting of the hardware characteristics of the computational nodes, including mobile device, network configuration and QoS, as well as the mobile application that has to be executed. The simulation is run 1000000 times to ensure a confidence interval of 95%. Since different deployments can be generated during these runs, due to the high variability of the simulated environment, we store all the obtained deployments in a histogram, storing for each deployment the frequency at which it occurs. We consider only the deployment with the highest frequency in the histogram for comparisons of different algorithms.

B. Computational nodes

Our first goal is to compare Mobile Cloud Computing (MCC) and Mobile Edge Cloud Offloading (MECO). For the MECO scenario, we set a infrastructure with 1 Cloud node and 1 Edge node. For the MCC scenario, we set instead a infrastructure with 2 Cloud nodes, to compare scenarios with the same amount of computational nodes. We assume that these amount of nodes is a realistic estimation of the nodes that are reachable by a mobile device and that is providing enough resources for execution of selected applications. We assume that CPU, RAM and storage specifications of Cloud nodes, as well as mobile device specifications, do not change during each different run of the simulation. This is because in real world scenarios, hardware configuration of computational nodes is rarely changing during one single application execution. We assume that Edge nodes have less capabilities than Cloud nodes in terms of cores, MIPS, RAM and storage [10]. The hardware specifications and hardware resources cost for each node are shown in Table I.

Node	Cores	RAM	Storage	MIPS per core	Core cost	RAM cost	Storage cost
Cloud-*	64	128	1000	15	0.03	0.02	0.01
Edge-*	16	8	250	15	0.03	0.02	0.01
Mobile	2	8	16	4	0	0	0

TABLE I: Hardware configuration.

Concerning mobile device, we need to consider also energy consumption for the calculation of battery lifetime. We use the energy model defined by Equation 11, with the energy consumption coefficients specified by [12]. Coefficients are summarized in Table II.

C. Network infrastructure

Connections between mobile devices and Cloud/Edge infrastructure is often unreliable, due to several environmental factors and user mobility. Therefore, we need to model this

Coefficient	Value
$\beta_{\texttt{freq}}$	6.9320
$\beta_{\texttt{base}}$	$625.25e-6$
ϵ_{3g}	$0.025e-6$
$\mathcal{K}_{3g}$	$3.5e-6$
ϵ_{wifi}	$0.007e-6$
K_{wifi}	$5.9e-6$

TABLE II: Energy coefficients for Equations 13 and 15.

unreliability in our scenario to perform an accurate simulation. We model the QoS provided by each link $l_i \in \mathcal{L}_{\mathcal{I}}$ as a random variable $QoS(l_i) = (latency(l_i), bw(l_i))$. We assume that each node is reachable by the mobile device using two different types of connections: 3G and WiFi. The availability of the two connections is also determined according to a random variable. If both are available during the execution, the faster between the two is selected. For the QoS, we use the probability distribution of FogTorchPI [24] which is summarized in Table III.

Connection	Availability	QoS profile		Probability
		Latency (ms)	Bandwidth (Mbps)	
3G	0.75	54	7.2	0.9957
		∞	0	0.043
WiFi	0.25	15	32	0.9
		15	4	0.09
		∞	0	0.01

TABLE III: Network availability distribution.

$latency = \infty$ and $bandwidth = 0$ mean that connection is not available. For Cloud offloading, also Internet transmission delay has to be considered, that is estimated by [25] to be between 100 and 300 milliseconds. We model it as a Gaussian random variable with $\mu = 200$ and $\sigma = 33.5$.

D. Mobile applications setup

Rather than typical DAG used for DAG scheduling simulations, we select DAGs of mobile applications, described in works like [3], [4], [26], [5]. This choice is because these test cases are more suited for the scenario that we are investigating than typical scientific workflows used for DAG scheduling simulations. The DAG structure used for this paper comes from the description of each application in the aforementioned works. We select four applications: (1) *Navigator*, that models the behavior of a GPS navigation software; (2) *Facerecognizer*, that models an image processing application that recognizes a face in a picture; (3) *Antivirus*, that models the behavior of a antivirus software and (4) *Chess*, that models a chess game between the user and the phone AI. We select the first two because they are representative of typical mobile applications, the Antivirus app to model the behavior of a data intensive application and Chess to model the behavior of a computational intensive application. For each DAG, the darker nodes are the ones that are not offloadable. We describe each DAG in the following sections.

1) Navigator: The application is described in [8], while its structure is summarized in Figure 2 and its requirements are outlined in Table IV. According to our analysis, the parameter that is mostly affecting application offload is the size of the map, over which the app computes the path to reach destination. Therefore, we vary the map size over $\texttt{map} = 25, 50, 100, 500$mb, that we observed to be realistic map sizes handled by mobile navigation applications. These values are set as $\frac{1}{\lambda}$ parameter of the exponential distribution used to generate different map size for each run. Map size is used as output data for MAPS task and as input/output for PATH_CALC and TRAFFIC.

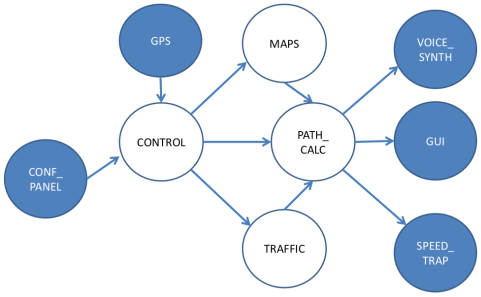


Fig. 2: DAG of Navigator app.

Task	CPUs	MI	RAM (GB)	Storage (GB)	Input data (mb)	Output data (mb)
CONF _PANEL	1	$\frac{1}{\lambda} = 1$	1	1	5	5
GPS	1	$\frac{1}{\lambda} = 1$	3	5	5	5
CON TROL	2	$\frac{1}{\lambda} = 2$	3	1	5	5
MAPS	2	$\frac{1}{\lambda} = 3$	5	5	$\frac{1}{\lambda} = \texttt{map}$	$\frac{1}{\lambda} = \texttt{map}$
PATH _CALC	1	$\frac{1}{\lambda} = 5$	2	5	$\frac{1}{\lambda} = \texttt{map}$	$\frac{1}{\lambda} = \texttt{map}$
TRAF FIC	1	$\frac{1}{\lambda} = 5$	1	5	$\frac{1}{\lambda} = \texttt{map}$	$\frac{1}{\lambda} = \texttt{map}$
VOICE _SYNTH	1	$\frac{1}{\lambda} = 2$	1	5	5	20
GUI	1	$\frac{1}{\lambda} = 2$	1	5	1	1
SPEED _TRAP	1	$\frac{1}{\lambda} = 2$	1	5	10	10

TABLE IV: Navigator app requirements.

2) Facerecognizer: Facerecognizer app is described in [3]. From the description, we extract the DAG of Figure 3. According to our study, the parameters affecting the most the offloading process is the size of the image that application has to process. The image size is used as input size of FIND_MATCH and DETECT_FACE. For these values, we perform different experiments choosing different values for $\frac{1}{\lambda}$, namely $\texttt{image} = \{10, 100, 500, 1000\}$kb. All parameters are summarized in Table V.

3) Antivirus: Antivirus app is described in [4]. This application scans the content of the local file system against a library of 1000 virus signatures. For this application, the most impacting parameters are the size of the virus signatures library and the size of the file to be scanned. For the input

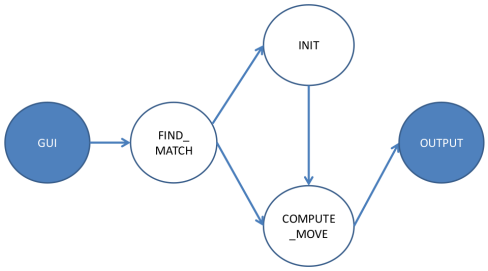


Fig. 3: DAG of Facerecognizer app.

Task	Cores	MI	RAM (GB)	Storage (GB)	Input data (kB)	Output data (kB)
GUI	1	$\frac{1}{\lambda} = 2$	1	1	$\frac{1}{\lambda} = \texttt{image}$	$\frac{1}{\lambda} = \texttt{image}$
FIND _MATCH	4	$\frac{1}{\lambda} = 4$	1	1	$\frac{1}{\lambda} = \texttt{image}$	$\frac{1}{\lambda} = \texttt{image}$
INIT	4	$\frac{1}{\lambda} = 4$	1	1	$\frac{1}{\lambda} = \texttt{image}$	$\frac{1}{\lambda} = \texttt{image}$
DETECT _FACE	4	$\frac{1}{\lambda} = 8$	1	1	$\frac{1}{\lambda} = \texttt{image}$	$\frac{1}{\lambda} = \texttt{image}$
OUTPUT	1	$\frac{1}{\lambda} = 4$	1	1	$\frac{1}{\lambda} = \texttt{image}$	$\frac{1}{\lambda} = \texttt{image}$

TABLE V: Facerecognizer app requirements.

size of the SCAN_FILE component, we perform different experiments using as $\frac{1}{\lambda}$ the values $\texttt{file} = \{10, 100, 1000\}$ and 1000 as $\frac{1}{\lambda}$ for the output size of LOAD_LIBRARY, as done in [4]. The other parameters are summarized by Table VI.

Task	Cores	MI	RAM (GB)	Storage (GB)	Input data (kb)	Output data (kb)
GUI	1	$\frac{1}{\lambda} = 4$	1	1	5	5
LOAD _LIBRARY	1	$\frac{1}{\lambda} = 2$	1	1	5	1000
SCAN _FILE	2	$\frac{1}{\lambda} = 2$	2	1	$\frac{1}{\lambda} = \texttt{file}$	5
COMPARE	1	$\frac{1}{\lambda} = 2$	1	1	$\frac{1}{\lambda} = \texttt{file} + 1000$	1
OUTPUT	1	$\frac{1}{\lambda} = 2$	1	1	1	5

TABLE VI: Antivirus app requirements.

4) Chess: Chess app is described in [3]. In this application, what affects the most the offloading decisions is the COMPUTE_MOVE task, that represents the computation of next AI move. This task consists in evaluating all the possible moves and can be a very computationally intensive task. Therefore, we set the $\frac{1}{\lambda}$ for the cores required by the task to and vary the $\frac{1}{\lambda}$ for the MI of the task between $\texttt{chess} = \{5, 10, 20, 40\}$ to simulate the varying computational requirements. Task requirements are summarized in Table VII.

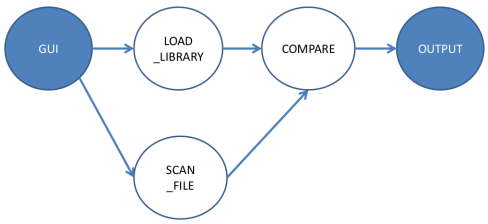


Fig. 4: DAG of Antivirus app.

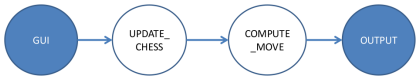


Fig. 5: DAG of Chess app.

Task	Cores	MI	RAM (GB)	Storage (GB)	Input data (kB)	Output data (kB)
GUI	1	$\frac{1}{\lambda} = 4$	1	1	5	5
UPDATE _CHESS	1	$\frac{1}{\lambda} = 2$	1	1	5	5
COMPUTE _MOVE	4	$\frac{1}{\lambda} = \texttt{chess}$	2	1	5	5
OUTPUT	1	$\frac{1}{\lambda} = 2$	1	1	5	5

TABLE VII: Chess app requirements.

VI. Results

We describe the simulation results for mobile applications offloading. The goal is to show how MECO improves performance in comparison to MCC. To this end, we employ three heuristics: (1) MinMin, defined in [19] that for each time step, offloads ready tasks in the order that guarantees the shortest running time; (2) MinCost, that for each time step offloads ready tasks to the node that guarantees the minimum user cost (that means, it will select the mobile device, if it has enough resources available), and (3) MaxBattery, that for each time step offloads ready tasks to the node that guarantees the maximum battery lifetime. The results that are shown in this section are averaged over 1000000 runs, that ensures us enough statistic significance. We use the metrics defined in III-D and the parameters identified in V.

A. MECO vs MCC

We perform two sets of experiments: in the first one, we vary the main parameter of each application (e.g. map size for Navigator, image size for Facerecognizer, file size for Antivirus, MI for compute move in Chess) and evaluate the difference between MECO and MCC for runtime, cost and battery lifetime. In the second set of experiments, we fix the application's main parameter and perform multiple executions of the applications' offloadable tasks. For the first set of experiments, we observe a difference between 1-2% for all objectives in all applications. This is because in this case, offloading is performed only once per offloadable task. Therefore, the effects of MECO are observed only once, with a limited effect on applications. However, by increasing the number of runs of the offloadable part, we observe a decrease of the running time when using also Edge nodes by up to 39% for Navigator, 70.7% Facerecognizer and around 30% for Antivirus. In Figure 6 we show only the results for Facerecognizer application, for brevity. Results for runtime, cost and battery lifetime are obtained using, respectively, MinMin, MinCost and MaxBatt. For Chess application, however, we do not observe any improvement with MECO: this is because of the low amount of data transmissions happening on this application, that reduces the effects of lower latency on runtime. Conversely, the high improvement in terms of runtime for

Facerecognizer is justified by the high amount of data transfers during the application's execution. Regarding cost and battery lifetime, all applications express trends similar to Facerecognizer results. In Figure 6b we observe that cost depends on η parameter and the running time. For $\eta = 0.01$, for example, cost of MECO dramatically increases, as this setting generates higher prices for Edge nodes. For other values, we see that for a lower number of runs (25-50) MCC is the most economic alternative, due to the additional price for using Edge nodes. However, for higher number of runs, MECO becomes the cost-effective alternative, as the dramatic reduction of runtime has a positive effect also on the cost for offloading the application. We observe a similar trend for other applications, with MCC being the cost-effective alternative until 50 runs for $\eta = 0.1$ and a slight decrease for $\eta = 1.0$. When increasing the number of runs, MECO becomes the cost effective alternative also for $\eta = 0.1$. Concerning battery lifetime, we observe an increase around 3% when using Edge nodes. This is because energy consumption network transmission is significantly lower than the energy consumption of computation, therefore reducing the effects of lower latency on energy consumption. In conclusion, MECO is recommended for long running applications with many offloadable tasks, regarding runtime and battery lifetime. Regarding cost, it depends on the η and the number of runs, as for some values of η and for less than 50 runs, the runtime improvement does not payoff the additional cost of MECO.

B. ECHO

In this section we evaluate the performance of ECHO versus three other heuristics: MinMin, MinCost and MaxBatt. We use these heuristics to evaluate how the solution selected by ECHO is close to the value of these other heuristics. We can see that, for $\eta = 0.01$, the trade-off identified by ECHO is either close to the minimum runtime or to the minimum cost. This is because the higher price values obtained setting $\eta = 0.01$ has a strong effect on the score of each solution: as we can see from Figures 7a and 8a, the lower difference in terms of cost cause the heuristic to find a trade-off with a cost closer to the minimum, causing however a runtime two times higher. Afterwards, when the difference in terms of runtime increases, the heuristic selects a runtime closer to the minimum, causing however a dramatic increase in terms of cost. This can be fixed by choosing different values for the weight coefficients. However, for higher values of η, our heuristics is able to find a solution that almost overlaps with all the other optimal values, due to the lower difference in prices generated by this setting of η. In Table VIII we summarize our results for all applications, calculating the MAPE of our solution between the optimal value for each objective. We can see that results are strongly related to the values of weights α, β and γ: in Table VIIIa, where all objectives have the same importance, we see that trade-off solution gets a lower MAPE for runtime, with a negative effect on cost, due to the bigger range over which runtime values are distributed; However, by giving an higher importance to cost (an higher β value), we see that MAPE for cost objective is lower. It is also of note that by setting $\alpha = 1$ and $\beta = \gamma = 0$ we obtain the MinMin results, while for $\beta = 1$ and $\alpha = \gamma = 0$ the MinCost and for $\gamma = 1$ and $\alpha = \beta = 0$ the MaxBatt results.

Application	Runtime	Cost	Battery
Navigator	15.97	17.3	0.48
Facerecognizer	18.06	42.5	0.18
Antivirus	10.37	68.1	0.07
Chess	2.48	99.3	0.01

(a) $\alpha = \beta = \gamma = 1.0$.

Application	Runtime	Cost	Battery
Navigator	52.4	4.56	0.48
Facerecognizer	68.75	5.53	0.18
Antivirus	34.29	16.59	0.07
Chess	2.5	3.65	0.01

(b) $\alpha = 0.1, \beta = \gamma = 1.0$.

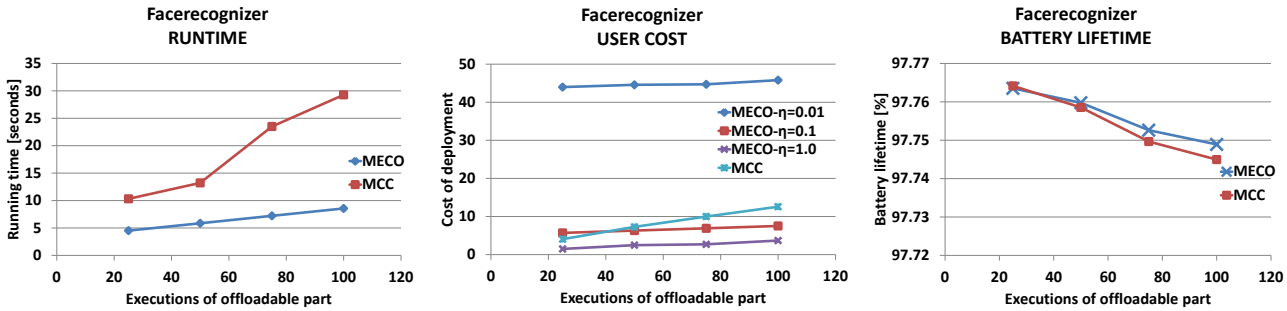
TABLE VIII: MAPE between results of ECHO and other heuristics, calculated for $\eta = \{0.01, 0.02, 0.05, 0.1, 1.0\}$

VII. Related Work

Several MCC frameworks have been proposed in [3], [4], [5], [26], [27], [6], [28]. In [8] an offloading approach based on model checking is presented. In [29] a programming model for mobile applications considering MCC is introduced. Paper [30] proposes container-based virtualization for MCC. The possibility to employ Edge nodes to support mobile application has been discussed by [31]. In [32] authors focus on reducing mobile application execution delay using MECO, while [33] identifies a trade-off between energy consumption and execution delay. However, both these works target fully offloadable applications. More similar to our work, in [9] a policy for partial offloading of DAG-structured mobile applications is proposed. However, they consider only battery lifetime. Works like [34], [35] aim at finding a trade-off solution for power and delay. A discussion of different offloading techniques for the Edge context can be found in [36]. In addition to these works, we also consider costs for user. Cost models for the Edge have been discussed in [15], [14]. DAG scheduling heuristics are described in [19], [37], while Monte-Carlo simulations for DAG scheduling are discussed in [7], [22].

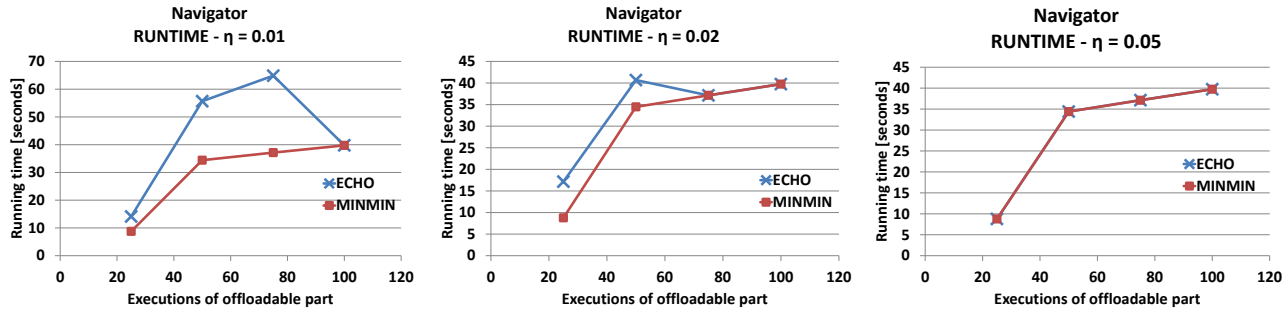
VIII. Conclusion and future Work

This work's main contribution can be summarized as follows: first, we describe MECO as a possibility to enhance capabilities of mobile devices, considering as objectives running time, user cost and battery lifetime. Then, we design a simulation framework and use it to compare MECO and MCC. Comparison is performed by simulating offloading of DAG models representing real-world mobile applications and showing that MECO can reduce mobile application runtime by up to 70.7% and cost by up to 70.6% with a 3% increase of battery lifetime. Finally, we propose ECHO, a heuristic approach for offloading that allows users to find a trade-off solution between three objectives. As future work, we plan to investigate different offloading techniques, based on multi-objective metaheuristics such as PSO and NSGA-II. Also, we plan to validate the simulator and extend our model by considering (1) multiple mobile devices, (2) different Edge/Cloud providers and (3) workloads of mobile applications, based on the LiveLab [38] traces. Finally, we would like to consider different objectives, such as provider profits and infrastructure's energy consumption.



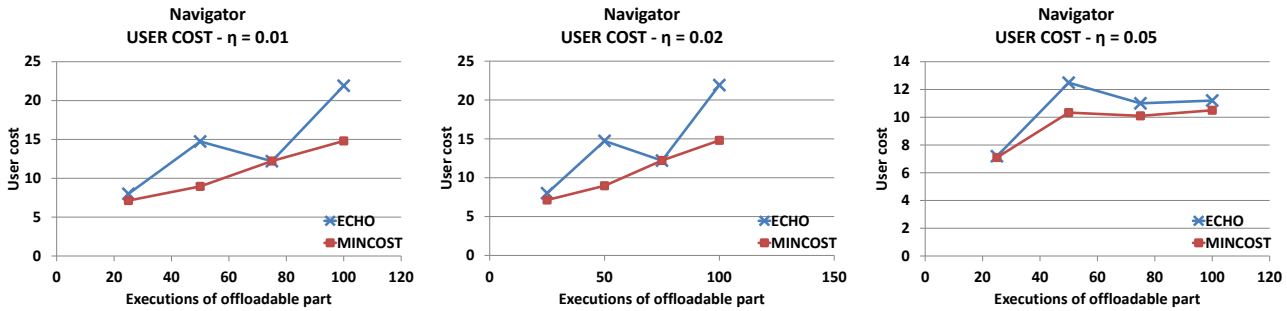
(a) Runtime. (b) User cost. (c) Battery lifetime.

Fig. 6: Results for Facerecognizer application with increasing executions of offloadable parts.



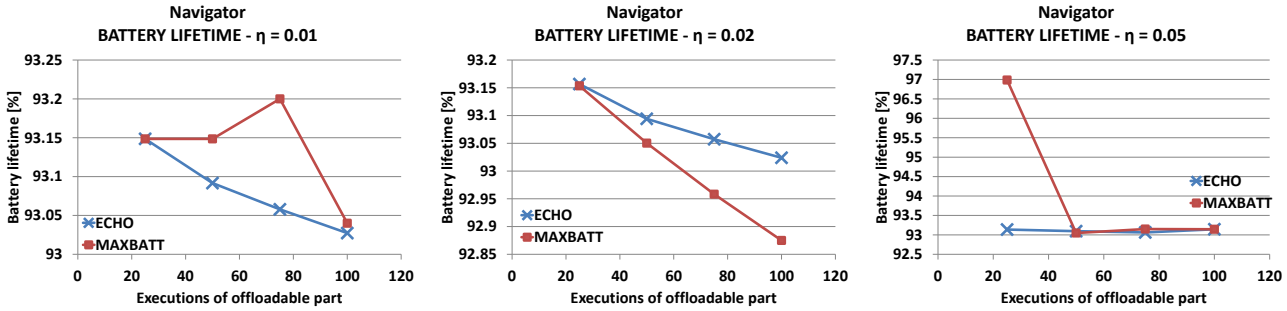
(a) $\eta = 0.01$. (b) $\eta = 0.02$. (c) $\eta = 0.5$.

Fig. 7: Application Navigator, runtime for different η.



(a) $\eta = 0.01$. (b) $\eta = 0.02$. (c) $\eta = 0.05$.

Fig. 8: Application Navigator, cost for different η.



(a) $\eta = 0.01$. (b) $\eta = 0.02$. (c) $\eta = 0.05$.

Fig. 9: Application Navigator, battery lifetime for different η.

ACKNOWLEDGMENT

The work described in this paper has been funded through the Haley project (Holistic Energy Efficient Hybrid Clouds) as part of the TU Vienna Distinguished Young Scientist Award 2011 and Rucon project (Runtime Control in Multi Clouds), FWF Y 904 START-Programm 2015.

REFERENCES

[1] K. Kumar and Y. H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[2] N. Palmer, R. Kemp, T. Kielmann, and H. Bal, "Ibis for mobility: Solving challenges of mobile computing using grid techniques," in *Proceedings of the 10th Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '09. ACM, 2009, pp. 17:1–17:6.

[3] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10. ACM, 2010, pp. 49–62.

[4] B. Chun, S. Ihm, P. Maniatis, and M. Naik, "Clonecloud: Boosting mobile device applications through cloud clone execution," *CoRR*, vol. abs/1009.3088, 2010.

[5] M. A. Khan, H. Debnath, N. R. Paiker, N. Gehani, X. Ding, R. Curtmola, C. Borcea, undefined, undefined, undefined, and undefined, "Moitree: A middleware for cloud-assisted mobile distributed apps," *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, vol. 00, pp. 21–30, 2016.

[6] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, *Energy Efficient Information Monitoring Applications on Smartphones through Communication Offloading*. Springer Berlin Heidelberg, 2012, pp. 60–79.

[7] W. Zheng and R. Sakellariou, "A monte-carlo approach for full-ahead stochastic dag scheduling," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, 2012, pp. 99–112.

[8] L. Aceto, A. Morichetta, and F. Tiezzi, "Decision support for mobile cloud computing applications via model checking," in *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, 2015, pp. 199–204.

[9] M. Deng, H. Tian, and B. Fan, "Fine-granularity based application offloading policy in cloud-enhanced small cell networks," in *2016 IEEE International Conference on Communications Workshops (ICC)*, 2016, pp. 638–643.

[10] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. ACM, 2012, pp. 13–16.

[11] F. A. Ali, P. Simoens, T. Verbelen, P. Demeester, and B. Dhoedt, "Mobile device power models for energy efficient dynamic offloading at runtime," *Journal of Systems and Software*, vol. 113, pp. 173 – 187, 2016.

[12] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 2009, pp. 280–293.

[13] A. Ahmed and E. Ahmed, "A survey on mobile edge computing," in *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, Jan 2016, pp. 1–8.

[14] M. Al-Roomi, S. Al-Ebrahim, S. Buqrais, and I. Ahmad, "Cloud Computing Pricing Models: A Survey," *International Journal of Grid and Distributed Computing*, vol. 6, no. 5, pp. 93–106, 2013.

[15] T. Zhao, S. Zhou, X. Guo, Y. Zhao, and Z. Niu, "Pricing policy and computational resource provisioning for delay-aware mobile edge computing," *2016 IEEE/CIC International Conference on Communications in China, ICCC 2016*, 2016.

[16] J. Hao, Z. Jin-hua, and C. liang jun, "Multi-objective particle swarm optimization algorithm based on enhanced ϵ-dominance," in *2006 IEEE International Conference on Engineering of Intelligent Systems*, pp. 1–5.

[17] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr 2002.

[18] H. M. Fard, R. Prodan, and T. Fahringer, "Multi-objective list scheduling of workflow applications in distributed computing infrastructures," *Journal of Parallel and Distributed Computing*, vol. 74, no. 3, pp. 2152 – 2165, 2014.

[19] B. Li, Y. Pei, H. Wu, and B. Shen, "Heuristics to allocate high-performance cloudlets for computation offloading in mobile ad hoc clouds," *J. Supercomput.*, vol. 71, no. 8, pp. 3009–3036, Aug. 2015.

[20] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments," *CoRR*, vol. abs/1606.02007, 2016.

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

[22] W. Nowak, "Introduction to stochastic search and optimization. estimation, simulation, and control," *IEEE Transactions on Neural Networks*, vol. 18, no. 3, pp. 964–965, May 2007.

[23] "A monte carlo algorithm for real time task scheduling on multi-core processors with software controlled dynamic voltage scaling," *Applied Mathematical Modelling*, vol. 38, no. 7, pp. 1929 – 1947, 2014.

[24] A. Brogi, S. Forti, and A. Ibrahim, "How to best deploy your fog applications, probably," in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, 2017, pp. 105–114.

[25] M. DeVirgilio, W. D. Pan, L. L. Joiner, and D. Wu, "Internet delay statistics: Measuring internet feel using a dichotomous hurst parameter," in *2013 Proceedings of IEEE Southeastcon*, 2013, pp. 1–6.

[26] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 Proceedings IEEE INFOCOM*, 2012, pp. 945–953.

[27] J. Flinn, SoYoung Park, and M. Satyanarayanan, "Balancing performance, energy, and quality in pervasive computing," in *Proceedings 22nd International Conference on Distributed Computing Systems*. IEEE Comput. Soc, 2012, pp. 217–226.

[28] M. H. Jiang, O. W. Visser, I. S. W. B. Prasetya, and A. Iosup, "Mirror: A computation-offloading framework for sophisticated mobile games," in *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2017.

[29] F. Lordan and R. M. Badia, "Compss-mobile: Parallel programming for mobile cloud computing," *Journal of Grid Computing*, vol. 15, no. 3, pp. 357–378, Sep 2017.

[30] S. Wu, C. Niu, J. Rao, H. Jin, and X. Dai, "Container-based cloud platform for mobile computation offloading," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 123–132.

[31] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 26–35, 2017.

[32] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," *CoRR*, vol. abs/1604.07525, 2016.

[33] S. Sardellitti, G. Scutari, and S. Barbarossa, "Distributed joint optimization of radio and computational resources for mobile cloud computing," in *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, 2014, pp. 211–216.

[34] O. Muoz, A. P. Iserte, J. Vidal, and M. Molina, "Energy-latency trade-off for multiuser wireless computation offloading," in *2014 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, 2014, pp. 29–33.

[35] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Power-delay tradeoff in multi-user mobile-edge computing systems," *CoRR*, vol. abs/1609.06027, 2016.

[36] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[37] L. F. Bittencourt, R. Sakellariou, and E. R. M. Madeira, "Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2010, pp. 27–34.

[38] C. Shepard, A. Rahmati, C. Tossell, L. Zhong, and P. Kortum, "Livelab: Measuring wireless networks and smartphone users in the field," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 3, pp. 15–20, Jan. 2011.