A cost efficient model for minimizing energy consumption and processing time for IoT tasks in a Mobile Edge Computing environment

João Gross and Cláudio Geyer

***Abstract*—With increase in the number of mobile devices connected to the Internet, data-intensive applications and increasingly restricted QoS requirements, together with the fact that these devices are powered by batteries with limited capacity, energy consumption and response latency for applications in the Internet of Things become an important research field. The use of the Cloud as the only alternative to downloading tasks raises doubts about its effectiveness in this scenario, given the high latency of communications. Mobile Edge Computing (MEC) is an architecture that introduces an intermediate computing layer between end devices and the Cloud. It provides mobile networks with Cloud Computing capabilities close to the end user, so that mobile device applications can offload their tasks, preserving their battery level and reducing response latency.**

In this scenario, a cost model with diversity of energy consumption variables and time elapsed for the MEC architecture is proposed, with the objective of allocating tasks in the best available cost option, aiming to reduce both the total energy consumed by the system and the total latency. The cost model is implemented using the TEMS (Time and Energy Minimization Scheduler) scheduling algorithm and validated with simulated experiments. The results show that it is possible to reduce the energy consumed in the system by up to 51.61% and the total time elapsed by up to 86.65% in the simulated cases with the parameters and characteristics defined in each test.

***Index Terms*—Mobile Edge Computing, Internet Of Things, Cost Model, Green Computing, Energy Consumption, Scheduling Algorithm.**

I. Introduction

BILLIONS of smart devices can now connect to the Internet in the form Internet of Things (IoT) due to advances in information technologies communication [1]. According to a Cisco report, these devices will generate 507,9 *ZB* of data in 2019 [2] and global revenue generated by big data technologies will skyrocket from $189 billion in 2019 to $274 billion in 2022 [3]. However, in the early days of computing, the situation was very different from today, when computers were still large in size and highly costly and tasks were performed on centralized machines. Since then, there have been several evolution cycles, each with characteristics that marked the evolution of its cycle in relation to the previous one.

In the first computing cycle, systems and applications were developed for a specific machine and its respective mode of operation, and processing was performed in *batch* for centralized processing [4]. The second cycle was marked by the advent of personal computers. The processing could be centralized or decentralized, in real time or in operation *batch*. However, personal computers were not connected to each other, so the computational capacity was limited to that available on the device itself.

The supercomputers allowed a centralized solution for applications with high computational demand, marking the beginning of the third computing cycle. Multiple machines forming *grids* and *clusters* were connected to each other in large collaborative groups providing a wide range of computational resources in a decentralized way.

In the early 2000s *Cloud Computing* became popular [5]. With easy access to computing and storage resources, intuitive interfaces and economically viable *pay-per-use* business models, *Cloud Computing* quickly gained ground among home users and businesses. The current computing cycle we live in, the fourth computing cycle, is marked by innovative technologies that have enabled thousands of devices to have Internet connectivity wherever they are. The Internet of Things, deals with this huge range of devices with connectivity that demand computational resources to perform their tasks. The increase in the number of smart devices connected to the Internet and the increasing generation of data on these devices create problems that did not exist before. The processing of this information on centralized servers makes the operation prohibitive, since the volume of data and the speed with which it is generated, makes it difficult for the user to have a quality experience, with low communication latencies, if the processing is exclusively remote, far from the location where data is created.

Problems related to latency and privacy arise when devices on the periphery of the network demand resources located in central units of the network, such as those offered by *Cloud Computing* services. In addition, there is considerable energy consumption, since modern Data Centers implement large-scale parallel processing, which requires large amounts of data to be transferred between the VMs of these equipments. As a consequence, the energy consumed by the Data Centers network can represent more than 20% of all energy consumed, while inter-VM communication can spend more than 33% of all processing time [6].

This scenario with intense data flows and thousands of IoT devices connected to the Internet demanded innovative architectures that allowed the processing of tasks close to the devices, in order to improve the *Quality of Service* (QoS).

Mobile Edge Computing (MEC) is a network architecture

João Gross and Cláudio Geyer were with the Parallel and Distributed Processing Group (GPPD), Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, 91509-900 Brazil e-mail: jlggross(at)inf.ufrgs.br, claudio.geyer(at)inf.ufrgs.br

that aims to reduce latency and provide better QoS [7] to end users, but focused on mobile networks such as 5G. Most applications today tend to offload task processing and data storage to remote servers, usually located away from the end user and the device. MEC allows provisioning of *Cloud COmputing* services close to mobile devices, bringing processing and storage closer to cellular base stations [8].

However, energy consumption in complex IoT networks, such as MEC environments, is an important problem as reported in previous works [11]. Most IoT sensors and actuators are small and have large energy limits. They generally operate on batteries or use intermittent sources of energy such as solar energy. Given the fact that IoT devices handle a lot of data, and require a lot of energy to process it, reducing energy consumption in networks with IoT devices is a goal worth exploring.

To solve this problem, a cost model was developed for the MEC environment composed of a set of variables from the network equipment and communication technologies used among its elements, so that the tasks generated in the system were allocated in a way that the energy consumption and total times for data transmissions and processing associated with each task could be minimized.

This paper is organized as follows: Section II presents some of the related previous work found in the literature. Section III presents the cost model for the system with three different allocation policies, (1) local processing in the IoT device, (2) local processing in the MEC server and (3) remote processing in the Cloud. Section IV introduces the heuristic algorithm designed to solve the system cost model. Section V details the implementation and show the results of the simulations using the TEMS scheduling algorithm to offload tasks in a MEC environment using the Cloud as a complementary task allocation option. Finally, Section VI concludes the paper.

II. Related Work

The development of cost models that aim to reduce energy consumption and response latency to applications in IoT systems is a topic discussed since the creation of the first Edge Computing architectures [9]. The use of the Cloud as the only option to offload tasks adds greater latencies to IoT applications, however it can be used as an alternative if the resources of the local network are exhausted [17]. There are few works that use the Cloud as an option to download tasks [11] [16]. Others, such as in [10], [12] and [14], use the Fog Computing architecture to offer local processing to IoT devices, but without using the Cloud. The use of the MEC architecture occurs in [11], [13], [15] and [16].

As for the variables used by the cost model, the energy consumption of task processing is unanimous, used by all works mentioned, however the energy consumption for data transmissions is restricted to [13], [14] and [15]. The processing time of tasks is also used by the majority, with the exception of [10] and [12], while the time spent on data transmissions is restricted to [11], [13], [14], [15] and [16]. Energy consumption when there is no processing, that is, with the equipment in idle state, is used only in [10] and [12].

The cost model developed in this research includes energy consumption and time spent for processing tasks, in addition to energy consumption and time spent on data transmissions, both for task data entry and task results. The waiting time in the queue of ready tasks is also considered. It corresponds to the time spent while waiting for the system scheduler to determine the processing location, used only in [11], [13] and [15]. The model also includes the battery level of the IoT devices, used only in [13], [14] and [15]. Finally, apart from processing in the IoT device or in the MEC server, the Cloud is included as an extra option for task offloading. The technique of DVFS (Dynamic Voltage and Frequency Scaling) is used by the model, allowing minimization of energy and time during the processing of tasks.

The following section details the proposed model with the three task allocation policies used, (1) local processing in the IoT device, (2) local processing in the MEC server and (3) remote processing in the Cloud.

III. System Cost Model

The system network has a finite set $D = \{1, 2, 3, ..., D\}$ of mobile IoT devices, $S = \{1, 2, 3, ..., S\}$ local servers and $W = \{1, 2, 3, ...W\}$ wireless communication channels. Each local device or MEC server can have zero or more tasks. The system has a total of $A = \{1, 2, 3, ..., A\}$ tasks. Each task A_i is represented by a tuple $A_i = (C_i, s_i, d_i, t_i)$, for $i \in A$.

For each task A_i, C_i represents the number of CPU cycles required for its complete execution, s_i and d_i represent, respectively, the code size and data entry and t_i represents the tasks deadline, that is, the maximum time to complete the task. If $t_i = 0$, then A_i is a normal task and don't have completion time requirements, whereas if $t_i = t'$ and $t_i > 0$, then A_i is a critical task and has to respect the deadline constrain.

Scheduling policies are represented by $p = \{0, 1, 2\}$. p_0 represents execution on the IoT device itself, p_1 represents execution on the local server and p_2 represents the execution on the remote *Cloud*.

The network has *k* wireless channels. The chosen wireless channel by the mobile device $j \in D$, for a task $i \in A$ is determined by $h_{i,p}$, in which $h_{i,p}$ is a *H* element. The set *H* has *n* records, with *n* being the number of system tasks. Thus, $H = \{h_{1,p}, h_{2,p}, ..., h_{n,p}\}$, in which each element $h_{i,p} \in \{0\} \cup \{1, 2, 3, ..., k\}$.

To execute task *i* on the IoT device, the selected policy is *0* ($p = 0$) and $h_{i,0} = 0$, representing that no communication channel was chosen. For task offloading to the local server $p = 1$ and for offloading to the remote Cloud $p = 2$. If $p = 1$ or $p = 2$, then a communication channel must be chosen. Given some task *i* allocated to the local server ($p = 1$) and with a communication channel chosen, e.g. channel number 4, then $h_{i,1} = 4$. The set of allocation decisions for all tasks with their respective communication channels selected is expressed by $H = \{h_{1,p}, h_{2,p}, ..., h_{n,p}\}$. Each task has only one record in the *H* set.

A. Local computing on the IoT device

Each $j \in D$ mobile device on the network has one or more CPUs. Thus, the CPUs available on network for the *j* device

are given by $PL_j = \{pl_{j,1}, pl_{j,2}, pl_{j,3}, ..., pl_{j,n}\}$. Every CPU $pl_{j,k} \in PL_j$ has a value of 0 or 1, 0 if it is vacant and 1 if it is busy. For vacant CPU, *idle* energy is counted and for occupied CPU dynamic energy ($E_{i,local,dynamic}$). Each CPU has an operating frequency ($f_{local,j,k}$), i.e. processing capacity, an effective commutative capacitance ($C_{local,j,k}$), depending on the chip architecture [16] and a voltage supply ($V_{local,j,k}$). In addition, each task *i* has a total number of CPU cycles (CT_i) for its execution.

So it is possible to calculate the total execution time of task $i \in A$, by the CPU $j \in PL$, as well as the dynamic energy consumed during the execution:

$$T_{i,local,dynamic} = \frac{CT_i}{f_{local,j,k}} \tag{1}$$

$$P_{i,local,dynamic} = C_{local,j,k} * V^2_{local,j,k} * f_{local,j,k} \tag{2}$$

$$E_{i,local,dynamic} = P_{i,local,dynamic} * T_{i,local,dynamic} \tag{3}$$

Equation 2 corresponds to the dynamic power a CPU consumes when processing some load which is $\propto CV^2f$ [18] [19]. If all CPUs on the device are busy, then its added to $T_{i,local,dynamic}$ a waiting cost defined as $T_{i,wait}$ which will be equal to the lowest $T_{i,local,dynamic}$ of the other tasks in execution, since as soon as one task ends, a CPU will be available. The scheduling algorithm is responsible for identifying which CPUs refer to which mobile devices. Thus, the equation of $T_{i,local,total}$ in case all CPUs of the device are occupied is given by:

$$T_{i,local,total} = \frac{CT_i}{f_{local,j,k}} + T_{i,wait} \tag{4}$$

Considering battery level and latency as model constraints, a D_j device must decide whether it is more appropriate to perform the task locally or remotely. As the battery level is a critical factor in the decision, the system will appreciate a policy that reduces energy consumption. If the task needs to be completed quickly, the tendency is to choose a policy that saves execution time. U_{localT} and $u_{localE} \in [0, 1]$ are used to represent the weight of these two limits, time and energy, respectively. The local cost of **one task *i*** can be expressed by:

$$\begin{aligned} Cost_{i,local,dynamic} = u_{localT} * T_{i,local,total} + \\ u_{localE} * E_{i,local,dynamic} \end{aligned} \tag{5}$$

In Equation 5 $0 \leq u_{localT} \leq 1$, $0 \leq u_{localE} \leq 1$ and $u_{localT} + u_{localE} = 1$, i.e. $u_{localT} = 1 - u_{localE}$. As mentioned in [15] the parameter u_{localE} works as a trade-off between execution time and energy consumption. The decision to download the task, giving preference to minimizing the execution time or energy consumption, is determined by the weighted parameter u_{localE}.

The CPU chosen to perform the task has the the value 1 assigned to it, that is, $pl_{j,k} = 1$. When the execution ends, the corresponding variable is assigned to 0, signaling that the CPU is vacant.

Both the number of CPU cycles to perform the task and the capacitance of the CPU are immutable variables. However, the frequency and voltage of the CPUs can be adjusted to minimize local cost.

B. Local computing on the MEC server

Local servers, as in IoT devices, are expected to have multiple CPUs. Thus, the CPUs available on a local server S_j are given by $PS_j = \{ps_{j,1}, ps_{j,2}, ps_{j,3}, ..., ps_{j,n}\}$. Each CPU $ps_{j,k}$ has an operating frequency ($f_{server,j,k}$), an energy capacitance coefficient of the chip architecture ($k_{server,j,k}$) and a supply voltage ($V_{server,j,k}$).

Communications that take place between the device and the local server use wireless communication channels and they can cause mutual interference between the each other. The data transfer rate for offloading the task *i* can be calculated according to *Shannon's formula*:

$$r_i(h) = W * \log_2\left(1 + \frac{p_m * g_{j,m}}{N + I_i}\right) \tag{6}$$

$$I_i = \sum_{n \in A | \{i\} : h_{n,p} = h_{i,p}} p_{m'} * g_{j',m'} \tag{7}$$

For Equation 6, W is the channel bandwidth in *Hz*, $g_{j,m}$ represents the channel gain between the mobile device *m* and a server local, represented by *j*. The variable p_m represents the transmission power of the mobile device *m* when offloading the task *i* to the local server. *N* represents the power of the thermal noise of the wireless channel allocated to $h_{i,p}$ and I_i represents the power of mutual interference between the users of the same channel.

Unlike processing on the device, processing on the local server requires data (d_i) and code (s_i) from the application to be transmitted to the server and the generated results (d'_i) must be transmitted back to the origin. Thus, the time required for a $j \in D$ mobile device to transmit data and download results from the server is given by:

$$T_{i,mec-up}(h) = \frac{s_i + d_i}{r_i(h)} \tag{8}$$

$$T_{i,mec-down}(h) = \frac{d'_i}{r_i(h)} \tag{9}$$

The total time required to complete the task execution on the local server considers $T_{i,mec-up}(h)$ and $T_{i,mec-up}(h)$ and the task execution time $T_{i,mec,dynamic}$. It also considers $T_{i,wait}$, when the task waits for resources to be available in the server. The $T_{i,wait}$ is equal to the fastest task to finish if all CPU are in use and zero if a CPU is already available. The total time and execution time are given by:

$$T_{i,mec,dynamic} = \frac{CT_i}{f_{mec,j,k}} \tag{10}$$

$$\begin{aligned} T_{i,mec,total} = T_{i,mec-up}(h) + T_{i,mec,dynamic} + \\ T_{i,mec-down}(h) + T_{i,wait} \end{aligned} \tag{11}$$

The energy consumed for the transmission of data from the IoT device to the local server and from the local server to the IoT device, can be expressed as:

$$E_{i,mec-up}(h) = p_{wireless}(s_i, d_i) * T_{i,mec-up}(h) \quad (12)$$

$$E_{i,mec-down}(h) = p_{wireless}(d'_i) * T_{i,mec-down}(h) \quad (13)$$

In Equations 12 and 13 $p_{wireless}$ is the transmission power for the wireless technology chosen in the system. Finally, the dynamic power and energy consumption of the *i* task processing on the CPU $ps_{j,k} \in PS_j$ and the total dynamic consumption are given by:

$$P_{i,mec,dynamic} = C_{mec,j,k} * V^2_{mec,j,k} * f_{mec,j,k} \quad (14)$$

$$E_{i,mec,dynamic} = P_{i,mec,dynamic} * T_{i,mec,dynamic} \quad (15)$$

$$\begin{aligned} E_{i,mec,total} = {} & E_{i,mec-up}(h) + \\ & E_{i,mec,dynamic} + \\ & E_{i,mec-down}(h) \end{aligned} \quad (16)$$

The cost equation for the local server is expressed as follows:

$$\begin{aligned} Cost_{i,mec,dynamic} = {} & u_{mecT} * T_{i,mec,total} + \\ & u_{mecE} * E_{i,mec,total} \end{aligned} \quad (17)$$

The coefficients $u_{mecT} + u_{mecE} \in [0, 1]$, where $0 \leq u_{mecT} \leq 1$, $0 \leq u_{mecE} \leq 1$ and $u_{mecT} + u_{mecE} = 1$. These coefficients can be scaled to prioritize one or the other cost, depending on the needs of the system applications or the purpose of the network. If the priority is to reduce energy consumption, the u_{mecE} coefficient should have greater weight, as it represents a greater value in the final cost, there are better conditions to be minimized. The same applies to latency, because if the system's applications have very short deadlines, u_{mecT} may have a greater weight than u_{mecE}, being more likely to be minimized.

C. Remote computing on the Cloud

The equations for latency and energy consumption in the centralized Cloud are very similar to those of the local server. The equation for the total latency calculation does not have the component $T_{i,wait}$, since it is assumed that the Cloud has unlimited resources, without the need to wait for a free CPU. In addition, there is the time spent for cellular data transmission between the IoT device and the MEC server and between the MEC server and the Cloud, that is, two data transmissions for *upload* and two for *download*. The energy consumption equation considers the energy spent on data transmissions, as follows:

$$\begin{aligned} T_{i,cloud,total} = {} & T_{i,mec-up}(h) + T_{i,mec-cloud-up} + \\ & T_{i,mec-cloud-up} + T_{i,cloud,dinamico} + \\ & T_{i,mec-down}(h) \end{aligned} \quad (18)$$

$$E_{i,mec-cloud-up} = p_{wired}(s_i, d_i) * T_{i,mec-cloud-up} \quad (19)$$

$$E_{i,mec-cloud-down} = p_{wired}(r_i) * T_{i,mec-cloud-down} \quad (20)$$

$$E_{i,cloud,dinamico} = p_{cloud,dinamico} * T_{i,cloud,dinamico} \quad (21)$$

$$\begin{aligned} E_{i,cloud,total} = {} & E_{i,servidor-up}(h) + E_{i,mec-cloud-up} + \\ & E_{i,cloud,dinamico} + E_{i,mec-cloud-down} + \\ & E_{i,mec-down}(h) \end{aligned} \quad (22)$$

The Equations 19 and 20 represent the energy cost for data transmission over a wired network, via optical fiber, and Equation 21 represents the dynamic consumption of the Cloud. There is no use of the DVFS technique by the system scheduler in the Cloud, as this feature is restricted to the service provider. Finally, the cost in *idle* of the Cloud is not considered, since the CPU offer is virtually infinite, therefore it does not make sense to account for this cost.

$$\begin{aligned} Cost_{i,cloud,dynamic} = {} & u_{cloudT} * T_{i,cloud,total} + \\ & u_{cloudE} * E_{i,cloud,total} \end{aligned} \quad (23)$$

The coefficients $u_{cloudT} + u_{cloudE} \in [0, 1]$, where $0 \leq u_{cloudT} \leq 1$, $0 \leq u_{cloudE} \leq 1$ and $u_{cloudT} + u_{cloudE} = 1$. To reduce latency, the best choice is that the latency coefficient is much higher than the energy consumption coefficient, that is, $u_{cloudT} \gg u_{cloudE}$. In this way, the cost minimization will occur mainly on latency, since the energy consumption costs for the system, when there is communication with Cloud, would be minimal. The correct choice of computation coefficients also prevents the scheduling algorithm from prioritizing the allocation of tasks to the Cloud.

D. Idle cost for IoT devices and MEC servers

The idle energy multiplies the specific power of each CPU when in idle by the time it is in this condition, given by t'. The calculation of power consumption for free CPUs is given by:

$$E_{local,idle} = \sum_{j=1}^{m} \sum_{i=1}^{n} P_{idle,pl_{j,i}} * t', \text{ if } pl_{j,i} = 0 \quad (24)$$

$$E_{mec,idle} = \sum_{j=1}^{m} \sum_{i=1}^{n} P_{idle,ps_{j,i}} * t', \text{ if } ps_{j,i} = 0 \quad (25)$$

E. System cost equation

The system cost equation is developed as an Integer Linear Problem (ILP). For each of the system's scheduling policies there is a specific cost equation. The cost of each task *i* is given by the following minimizing equation:

$$\begin{aligned} Cost_{i,dynamic} = min(&\alpha * Custo_{i,local,dynamic}, \\ &\beta * Custo_{i,mec,dynamic}, \\ &\gamma * Custo_{i,cloud,dynamic}) \end{aligned} \tag{26}$$

In the Equation 26 $\alpha, \beta, \gamma \in [0, 1]$ and $\alpha + \beta + \gamma = 1$. The three coefficients can be scaled to prioritize computation in one or another scheduling policy. The choice of the lowest cost value among the three calculated defines which scheduling policy will be chosen for that task, whether it is running locally on the device, locally on the local server, or remotely on the Cloud. The system cost equation considers the minimized costs for the execution of each task, that is, dynamic cost. In addition, it considers energy costs in idle for mobile devices and local servers, as follows:

$$\begin{aligned} Cost_{system} = &\sum_{i=1}^{A} Cost_{i,dynamic} + \\ &\alpha * E_{local,idle} + \\ &\beta * E_{mec,idle} \end{aligned} \tag{27}$$

F. Constrains on battery level for IoT devices

A healthy battery level is essential for the proper functioning of IoT devices. If the battery level is below a *Lower Safety Limit* (LSL), task allocation on the device is disabled, to preserve the functioning of the equipment with the remaining battery. Therefore, all cost equations for allocation in the IoT device are subject to the following constrains:

- $B_{i,local} > E_{i,local}$
- $B_{i,local} > E_{i,server-up}(h)$
- $B_{i,local} > LIS$

$B_{i,local}$ represents the battery level of the mobile device generating for task *i*. With $B_{i,local}$ below the LSL the device has the allocation policy 0 disabled. If $B_{i,local}$ reaches zero, all tasks generated by the device are canceled.

IV. Time and Energy Minimization Scheduler (TEMS)

The TEMS heuristic scheduling algorithm was developed in order to solve the system cost model. It has complexity $O(n^2)$ and provides processing with reduced computational cost and faster solutions. It executes the system cost model, seeking to minimize both energy consumption and task processing time. The general structure of the algorithm is shown in the image of the Algorithm 1.

In step 1, the data sets of the IoT devices, MEC servers and communication channels are defined. The battery level of the IoT devices is collected and the LSL is set. The algorithm collects the number of CPUs available in each IoT device and in the MEC servers, their operating frequencies and supply voltages. Finally, the set of choices for the communication channels associated to each task (set H) are initialized and all CPUs are defined as available, with no processing load.

Algorithm 1: Time and Energy Minimization Scheduler (TEMS)

```
Result: Task mapping to the processing nodes
execute Step 1 - Collection of system information and inicialization
repeat
    execute Step 2 - Task allocation
    execute Step 3 - Task conclusion monitor
    execute Step 4 - New tasks and device battery level monitor
until user decides to keep running;
```

Steps 2, 3 and 4 are shown in more details in the following subsections.

A. Step 2 - Task allocation

First, tasks are classified between critical tasks and normal tasks. Critical tasks are ordered by deadline, while normal tasks are stored in an unsorted list. The time and energy consumption for task processing on the different CPUs of the network are calculated, as well as the time and energy consumption of the data transmissions, for MEC servers and for the Cloud. The allocation of critical tasks occurs on the CPU that offers the lowest total time and normal tasks are allocated on the CPU that offers the lowest total cost.

Critical tasks are the first to be allocated due to the sensitivity of the deadline and also because more normal than critical tasks are expected from the system. Therefore, the allocation of critical tasks should not significantly impact the efficiency of the system.

B. Step 3 - Task conclusion monitor

In Step 3 tasks are monitored for their status, whether running or completed, and when completed, the CPU resources are released and made available for other allocations in Step 2. If the task is being executed in the Cloud, there is no need to release resources once the task is completed, since the premise used for the Cloud is that it has unlimited resources, and therefore can absorb any number of tasks to perform without having to free up other resources.

The battery level test is performed only for mobile IoT devices, since the local servers are not mobile and have external power that allows them to operate uninterrupted. Task cancellation is not a desirable result, but it is an important metric that must be monitored to identify the efficiency level of the proposed algorithm. However, task cancellation will occur naturally if the task's workload is very large and the deadline small, without any allocation policy allowing the task to be completed in a timely manner, however fast the CPU may be.

C. Step 4 - New tasks and device battery level monitor

Finally, in Step 4, the battery level from each IoT device is collected, as well as new newly created tasks. Steps 2, 3 and 4

Algorithm 2: Step 2 – Task allocation

order critical tasks by deadline
foreach *task* A_i *from ordered list* **do**
  foreach *CPU* $pl_{j,k} \in PL_j$ **do**
    if *(*$B_{j,local} > LIS$*) and (*$B_{j,local} > E_{estimada}$*) and (*$pl_{j,k} = 0$*)* **then**
      calculate IoT device execution time
    end
  end
  foreach *CPU* $ps_{j,k} \in PS_j$ **do**
    if $ps_{j,k} = 0$ **then**
      calculate MEC server execution time and transmission times
    end
  end
  calculate Cloud execution time + transmission times
  offload task to the CPU with minimal total time
end
foreach *task* A_i *from list of normal tasks* **do**
  foreach *CPU* $pl_{j,k} \in PL_j$ **do**
    if *(*$B_{j,local} > LIS$*) and (*$B_{j,local} > E_{estimada}$*) and (*$pl_{j,k} = 0$*)* **then**
      calculate energy consumption, execution time and cost for the IoT device
    end
  end
  foreach *CPU* $ps_{j,k} \in PS_j$ **do**
    if $ps_{j,k} = 0$ **then**
      calculate energy consumption for dynamic processing and data transmission
      calculate execution time and transmission times
      calculate MEC server cost
    end
  end
  calculate energy consumption for dynamic processing and data transmission, execution time and transmission times and final cost for the Cloud
  offload task to the CPU with minimal cost
end

continue in continuous and parallel execution as long as there are tasks running, or until the network administrator decides to end the processes. In a real system these steps would continue to run indefinitely.

V. Implementation and Evaluation

The TEMS algorithm was implemented in Eclipse IDE version 2019-12 (4.14.0) and written in the Java programming language version 1.8.0_241. The simulations were run in a machine with CPU Intel Quad-Core i7-7700HQ 2,80 GHz and 16 GB of RAM, running OS Windows 10 Home 64-bit.

The simulated network considered as IoT devices Arduino Mega 2560 boards, with operating frequencies of 16 MHz, 8 MHz, 4 MHz, 2 MHz and 1 MHz and corresponding supply voltages of 5 V, 4 V, 2.7 V, 2.3 V and 1.8 V. The Arduino's microcontroler was set with capacitance of 2.2 nF (nanofarads) [20]. For the MEC layer servers composed of 5 Raspberry Pi 4 Model B boards were chosen. Each Raspberry Pi 4 Model B is equipped with a Quad-core Cortex-A72 1.5GHZ (ARM v8) 64-bit, summing a total of 20 CPUs per server. Each CPU has operating frequencies of 1500 MHz, 1000 MHz, 750 MHz and 600 MHz and corresponding supply voltages of 1.2 V, 1 V, 0.825 V and 0.8 V [21]. The capacitance of the CPU is set to 1.8 nF. For the Cloud it was chosen Data Centers with Intel Xeon Cascade Lake processors of 2.8 GHz per CPU, reaching up to 3.9 GHz with Turbo Boost on.

For data transmissions, it was established that both 5G and fiber optic communications could reach speeds of up to 1Gbps. For both, latency was set at 5ms [22] [23].

Two vehicular applications were defined [24] for the experiments as shown in Table I.

TABLE I
Characteristics of the chosen applications.

Characteristics	Application 1	Application 2
Task generation rate (seconds)	10	0,1
Data entry (mega *bytes* - MB)	36,3	4
Results (bytes)	1250	625
Computational load (millions of CPU cycles)	2000	20
Critical Tasks (%)	10	50
Deadline for critical tasks (ms)	500	100

The scenarios tested in the simulations varied the parameters set to the applications, such as computational load of each task, coefficients for energy consumption and time, size of data entry and results, task generation rate, critical task deadline, level of batteries for IoT devices and use of DVFS to see how the TEMS algorithm would respond to task allocation, energy consumption and time spent and task completion status. Some of the results are discussed in the subsections that follow.

A. Varying the tasks computational load

The applications of the Table I were executed in different scenarios. The chosen scenarios considered a fixed number of 500 tasks, and the four cases presented for each application in Figure 1 used, respectively, (a) 100 IoT devices and 1 MEC server, (b) 100 IoT devices and 2 MEC servers, (c) 500 IoT devices and 1 MEC server and (d) 500 IoT devices and 2 MEC servers.

For these experiments the energy and time coefficients were set, respectively, at $4/5$ and $1/5$, that is, a high weight was given to the energy consumed, so that it could be minimized. In Figures 1-1.(a) and 2.(a) the difference is in the number of MEC servers. With the increase in the number of MEC servers fewer tasks need to be allocated in the Cloud. This positively impacts the total energy consumed. Compared to a scenario with 500 tasks and 100 IoT devices, but without the use of MEC servers, the reduction in energy consumption for case 1 was 42.51%, while for case 2 it reached 44.71%. Thus, the use of MEC servers helps the system to lower the total energy consumed.

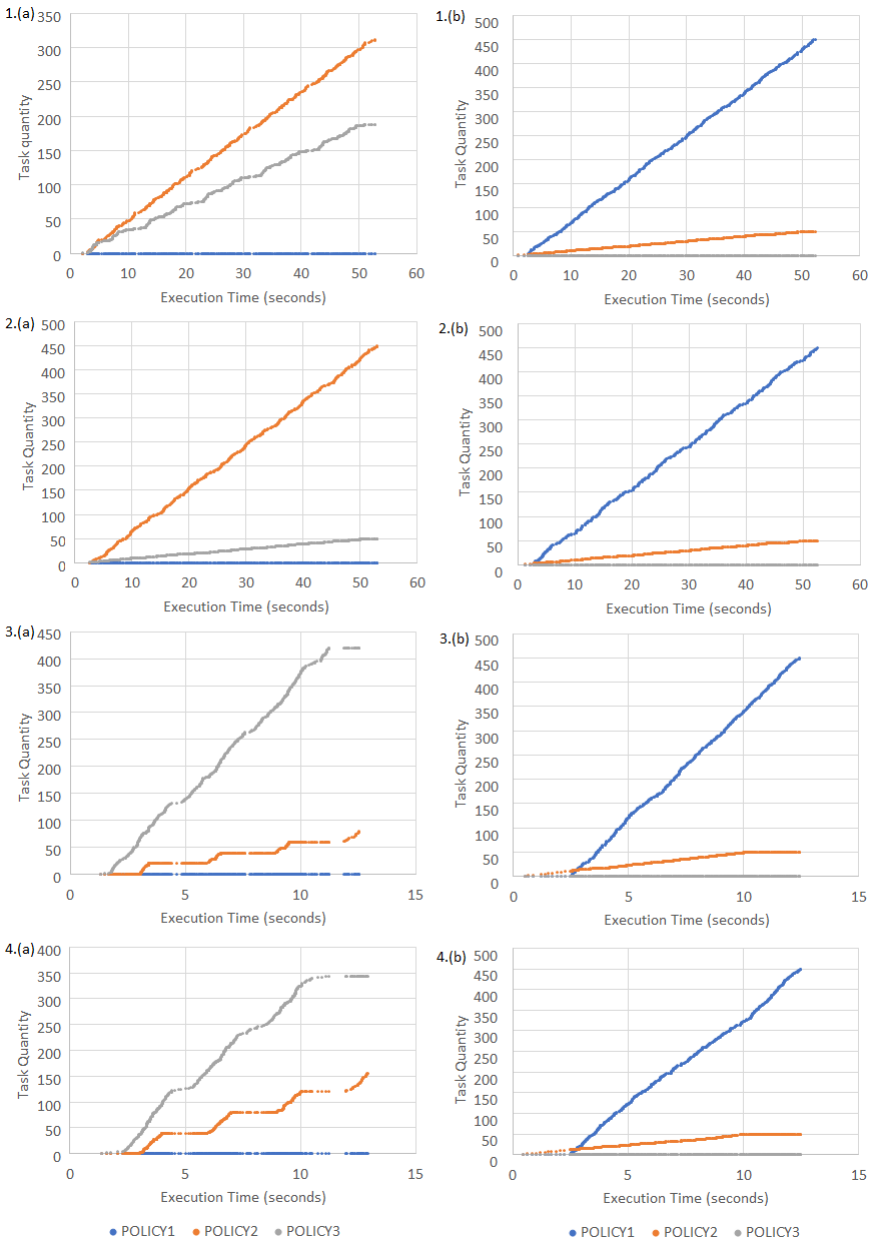

Fig. 1. Processing load for Application 1 defined to $2.000 * 10^6$ CPU cycles ((a) images) and for Application $20 * 10^6$ CPU cycles ((b) images), both varying the quantity of IoT devices and MEC servers.

Figures 1-3.(a) and 4.(a) have 500 tasks. Due to the accelerated generation of tasks, local servers are saturated and most allocations occur in the Cloud. For cases 1.(a), 2.(a), 3.(a) and 4.(a) no allocation is made on the IoT devices. This is because the cost of this allocation policy for the application 1 workload is very high. This way, the allocation policy on the IoT device is discarded, giving priority to MEC servers, and in the absence of these, the Cloud. Thus, the task processing load should not be too high, in order to allow IoT devices to have lower costs, and be chosen as an alternative to task allocation.

All figures (b) correspond to use cases of Application 2. It is noticeable that most allocations occur on the device itself, regardless of the number of IoT devices or MEC servers. The cause of this phenomenon is due to the low processing load of Application 2. The hardware of the IoT devices presents higher energy consumption per CPU cycle, however, it does not require data transmissions, which add energy cost and elapsed time to the system. Therefore, for a low processing load, IoT devices are the first allocation option.

B. *Varying coefficients for energy consumption and time*

This experiment used Application 2 with $2*10^6$ CPU cycles, instead of $20 * 10^6$. Each case has 500 tasks, 100 IoT devices and one MEC server. The coefficients used for energy were $1/5$, $2/5$, $3/5$, $4/5$ and the time coefficients were respectively $4/5$, $3/5$, $2/5$ and $1/5$.

TABLE II
COSTS FOR APPLICATION 2 WITH A LOAD OF $2 * 10^6$ CPU CYCLES, VARYING THE COST COEFFICIENTS FOR ENERGY AND TIME. E_{Total} IN *W*s* AND T_{Total} IN SECONDS.

$C_{E1.0}$	$C_{E2.0}$	$C_{E3.0}$	$C_{E4.0}$	E_{Total}	T_{Total}	f (*MHz*)	T (*V*)	Pol.
0,01859	0,02607	0,03355	0,04102	0,1455	0,0334	1.500	1,200	2
0,01867	0,02599	0,03332	0,04065	0,1439	0,0340	1.000	1,000	2
0,01877	**0,02597**	**0,03318**	0,04038	0,1428	0,0347	750	0,825	2
0,01894	0,02609	0,03324	0,04039	0,1426	0,0354	600	0,800	2
0,03506	0,04855	0,06203	0,07552	0,2670	0,0647	2.800	-	3
0,03518	0,04855	0,06203	0,07552	0,2696	0,0645	3.900	-	3
0,04067	0,03967	0,03867	0,03767	0,1100	0,1250	16	5,000	1
0,07136	0,05939	0,04741	**0,03544**	0,0704	0,2500	8	4,000	1
0,13547	0,10428	0,07308	0,04189	0,0321	0,5000	4	2,700	1
0,26822	0,20310	0,13799	0,07287	0,0233	1,0000	2	2,300	1
0,53428	0,40190	0,26952	0,13713	0,0143	2,0000	1	1,800	1

Table II lists the costs perceived by the system task scheduler for Application 2 with a processing load of $2 * 10^6$ CPU cycles. In bold, the lowest costs for each pair of energy and time coefficients are highlighted. For Cases 2 and 3, there was no change in energy consumption or in total time, since the lowest cost was found in the same frequency range, voltage and allocation policy. In Case 1 the lowest cost, and the corresponding allocation, occurred for the MEC server, with DVFS configured at 1500 MHz and 1.2 V. As for Case 4, with energy coefficient at $4/5$ and the time coefficient at $1/5$ the allocation took place on the device itself, with DVFS configured at 8 MHz and 4 V.

With focus on reducing energy consumption, choosing $4/5$ and $1/5$ coefficients for energy and time, respectively, offers a reduction of up to 51.61% in the energy consumption of normal tasks compared the other cases. If the focus is on reducing task completion time, energy coefficients of $1/5$, $2/5$ and $3/5$, with their respective time coefficients, one can achieve a reduction of up to 86.65% in the completion of normal tasks in the face of the most costly case.

C. Varying the size of the application data entry

This simulation used Application 1, with a workload configured at $200 * 10^6$ CPU cycles, energy cost coefficient at $4/5$, 500 tasks, 100 IoT devices and 2 MEC servers. Entries of 4 different sizes, 3.6 MB, 36 MB, 362 MB and 3.6 GB were used.

As the size of the data entry increases, the costs calculated for each case progress in policies for MEC servers and the Cloud. The cost of policy for IoT devices allocation remains the same, as no data transmissions are carried out. When data entry scales, allocation policies that require data transmissions become very costly and allocation on the device itself becomes increasingly advantageous. Thus, it is important to scale applications so that data transfers across the network are not too large per task, avoiding high data transmission costs.

Two other cases were designed, one with 5000 tasks and 362 MB per task and another with 500 tasks and 3.6 GB per task. The system's energy consumption and total elapsed time for the 500 task case was 59,160.92 W*s and 14,515.21 s. For the case with 5000 tasks, the costs were 45,011.49 W*s and 10,305.94 s, that is, a decrease of 23.92% and 29%, respectively. Therefore, as explained above, in an environment with continuous task generation with low with low deadline and the need for low latencies, tasks should not be very data intensive, as data transmissions add additional energy and time expenses.

D. Varying task generation rates, critical task deadline, battery level and DVFS activation

The task generation rate had a big impact on task allocation, because with a very fast generation rate, the CPU resources available on the IoT devices and the MEC server were quickly occupied, leaving only task allocation to the Cloud CPUs. Therefore, very low task generation rates flood the network with tasks, depleting the resources of the local network, leaving only the Cloud as an alternative to allocation, which increases the energy costs of the system.

Very low deadlines presented problems for the simulated applications, since the available allocation policies had difficulty in completing the critical tasks in an adequate time. This caused several tasks to be cancelled.

Some tests were performed varying the battery level of the IoT devices. It has been observed that very low battery levels quickly reach LSL and make IoT devices unavailable for processing. Very high computational loads also disadvantage batteries. Therefore, a battery with a healthy energy level and adequate processing loads, allows the allocation to be carried out on the IoT device, without making it unavailable due to lack of battery.

In addition, simulations were prepared for scenarios with and without the use of the DVFS technique. In the analyzed cases, with activated DVFS, the total energy consumption decreased by 13.736%, while the total time increased by 28.32%. This demonstrates the effectiveness of the system model, as well as the scaling algorithm, in minimizing total energy consumption. The total time may have been longer in the approach with DVFS, but it is not problematic because the application tasks were completed within the time limit imposed by the deadline.

VI. Conclusion

In an environment with accelerated generation of large volumes of data and mobile devices connected to the Internet with restricted Qos requirements and battery limitations, MEC is an alternative that provides better latency and reduced energy consumption, compared to using the Cloud. Task allocation must be adequate in this environment, seeking to minimizing total processing times and energy consumption, also reducing its environmental impacts with less CO_2 emissions.

The results from the experiments made it possible to observe the behavior of the system regarding the allocation of tasks by policy, the number of completed and canceled tasks, the modification of the costs perceived by the system scheduler and their impact on energy consumption and the times required for the completion of tasks. Among the experiments carried out, it was concluded that the adequate adjustment of the cost coefficients for energy consumption and elapsed time were essential for the final cost perceived by the scheduling algorithm to minimize both the energy consumed and the processing time of the tasks. Adequate coefficients allowed the system's energy to be reduced by up to 51.61% or the total times to be reduced by up to 86.65%, ending tasks before deadline. The system has become more sustainable and the user experience has not been affected. The use of MEC servers allowed to extend the battery life of the IoT devices and helped to make more agile execution of tasks with a high processing load profile.

The effectiveness of the cost model for the MEC system was proven in the various simulations carried out. The TEMS algorithm allowed the reduction of energy consumption and total times, according to the weighting of the coefficients for energy and time, the adjustment of the parameters used in the applications and the CPU resources offered on the network, with their respective operating frequency and supply voltage characteristics using the DVFS technique.

References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, *Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications*, IEEE Communications Surveys Tutorials, 2015.

[2] N. Hassan, S. Gillani, E. Ahmed, I. Yaqoob and M. Imran, *The Role of Edge Computing in Internet of Things*, IEEE Communications Magazine, 2018.

[3] Rob Marvin, *The Big Data Market Is Set to Skyrocket by 2022*, Available on: https://www.pcmag.com/news/368958/the-big-data-market-is-set-to-skyrocket-by-2022, Access: january 02, 2020.

[4] United States. Department of the Army, *Army ADP Review*, Department of Defense, Department of the Army, 1977.

[5] M. Heck, J. Edinger, D. Schaefer and C. Becker, *IoT Applications in Fog and Edge Computing: Where Are We and Where Are We Going?*, 27th International Conference on Computer Communication and Networks (ICCCN), 2018.

[6] E. Baccarelli, N. Cordeschi, A. Mei, M. Panella, M. Shojafar and J. Stefa, *Energy-efficient dynamic traffic offloading and reconfiguration of networked data centers for big data stream mobile computing: review, challenges, and a case study*, IEEE Network, 2016.

[7] F. Haouari, R. Faraj and J. M. AlJa'am, *Fog Computing Potentials, Applications, and Challenges*, International Conference on Computer and Applications (ICCA), 2018.

[8] Y. Yu, *Mobile edge computing towards 5G: Vision, recent progress, and open challenges*, China Communications, 2016.

[9] M. Satyanarayanan, P. Bahl, R. Caceres and N. Davies, *The Case for VM-Based Cloudlets in Mobile Computing*, IEEE Pervasive Computing, 2009.

[10] J. Wan, B. Chen, S. Wang, M. Xia, D. Li and C. Liu, *Fog Computing for Energy-Aware Load Balancing and Scheduling in Smart Factory*, IEEE Transactions on Industrial Informatics, 2018.

[11] S. R. Sarangi, S. Goel and B. Singh, *Energy Efficient Scheduling in IoT Networks*, Proceedings of the 33rd Annual ACM Symposium on Applied Computing, 2018.

[12] H. Wu and C. Lee, *Energy Efficient Scheduling for Heterogeneous Fog Computing Architectures*, IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), 2018.

[13] G. Zhang, W. Zhang, Y. Cao, D. Li and L. Wang, *Energy-Delay Tradeoff for Dynamic Offloading in Mobile-Edge Computing System With Energy Harvesting Devices*, IEEE Transactions on Industrial Informatics, 2018.

[14] H. Gedawy, K. Habak, K. A. Harras and M. Hamdi, *Awakening the Cloud Within: Energy-Aware Task Scheduling on Edge IoT Devices*, IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), 2018.

[15] C. Wang, C. Dong, J. Qin, X. Yang and W. Wen, *Energy-efficient Offloading Policy for Resource Allocation in Distributed Mobile Edge Computing*, IEEE Symposium on Computers and Communications (ISCC), 2018.

[16] H. Yu, Q. Wang and S. Guo, *Energy-Efficient Task Offloading and Resource Scheduling for Mobile Edge Computing*, IEEE International Conference on Networking, Architecture and Storage (NAS), 2018.

[17] Z. Zhan, X. Liu, Y. Gong, J. Zhang, H. Chung and Y. Li, *Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches*, ACM Computing Surveys, 2015.

[18] Intel, *Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor (White Paper)*, Intel Corporation, 2004. Available on: https://web.archive.org/web/20150812030010/http://download.intel.com/design/network/papers/30117401.pdf, Access: january 10, 2020.

[19] Y. Liu, H. Yang, R.P. Dick, H. Wang and L. Shang, *Thermal vs Energy Optimization for DVFS-Enabled Processors in Embedded Systems*, 8th International Symposium on Quality Electronic Design, 2007.

[20] Atmel, *ATmega/V-2560 Datasheet, 2014*,

[21] Raspberry Pi Foundation, *Raspberry Pi 4 Model B - FAQ*, Available on: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/, Access: october 08, 2020.

[22] A. Gupta and R. K. Jha, *A Survey of 5G Network: Architecture and Emerging Technologies*, IEEE Access, 2015.

[23] A. Brogi, S. Forti and A. Ibrahim, *Deploying Fog Applications: How Much Does It Cost, By the Way?*, CLOSER, 2018.

[24] J. Jansson, *Collision Avoidance Theory with Application to Automotive Collision Mitigation*, 2015.

João Luiz Grave Gross João is a master's degree student with the Federal University of Rio Grande do Sul in Brazil, whose research areas include Internet of Things, Edge Computing and Energy Consumption.

Claudio Resin Geyer PhD Claudio Resin Geyer is a Full Professor in the Federal University of Rio Grande do Sul in Brazil, whose main research areas include Parallel and Distributed systems, Big Data, Ubicomp, Internet of Things, Smart Cities, Cloud, Fog and Edge Computing.