MDPI

Article

TEMS: An Algorithm to Minimize Energy Consumption and Elapsed Time for IoT Workloads in a Hybrid Architecture

Julio C.S. Anjos [1]*, João L.G. Gross [1], Kassiano J. Matteussi [1],Gabriel V. González [2], Valderi R.Q. Leithardt [3] and Claudio F.R. Geyer[1]

[1] Federal University of Rio Grande do Sul, Institute of Informatics, UFRGS/PPGC, Porto Alegre, RS, Brazil, 91501-970; {jcsanjos, jlggross, kjmatteussi,geyer}@inf.ufrgs.br
[2] Faculty of Science, Expert Systems and Applications Laboratory, University of Salamanca, 37008 Salamanca, Spain; gvg@usal.es
[3] Instituto Politécnico de Portalegre, VALORIZA Research Center, Portalegre, Portugal, 7300-555; valderi@ipportalegre.pt
* Correspondence: jcsanjos@inf.ufrgs.br; Tel.: +55 51 3308 6168 (Julio Anjos)

Abstract: Advances in communication technologies have made the interaction of small devices, such as smartphones, wearables, and sensors, scattered on the Internet, bringing a whole new set of complex applications with ever greater task processing needs. These Internet of Things (IoT) devices run on batteries with strict energy restrictions. They tend to offload task processing to remote servers, usually to Cloud Computing (CC) in datacenters geographically located away from the IoT device. In such a context, this work proposes a dynamic cost model to minimize energy consumption and task processing time for IoT scenarios in Mobile Edge Computing environments. Our approach allows for a detailed cost model, with an algorithm called TEMS that considers energy, time consumed during processing, the cost of data transmission, and energy in idle devices. The task scheduling chooses among Cloud or Mobile Edge Computing (MEC) server or local IoT devices to better execution time with lower cost. The simulated environment evaluation saved up to 51.6% energy consumption and improved task completion time up to 86.6%.

Keywords: Mobile Edge Computing; Internet Of Things; Cost Minimization Model; Energy Consumption; Scheduling Algorithm

Citation: Anjos, J.C.S.; Gross, J.L.G; Matteussi, K.J.; Leithardt, V.R.Q.; González , G.V; Geyer, C.F.R. TEMS: An Algorithm to Minimize Energy Consumption and Elapsed Time for IoT Workloads in a Hybrid Architecture. *DNA* **2021**, *1*, 1–16. https://doi.org/

Received:
Accepted:
Published:

1. Introduction

International Data Corporation (IDC) report appoints there will be 41.6 billion IoT devices up to 2025 with a potential for data generation up to 79.4 ZB [1]. In such a context, IoT applications have evolved the use of artificial intelligence, artificial vision, and object tracking, which require high computing power [2,3]. They usually rely on task processing offload and data storage to remote Cloud Computing (CC) Data Centers to boost processing time and reduce battery energy consumption [4]. Unfortunately, those remote servers are geographically located away from the end-user and IoT device, resulting in high latency due to delay and congestion over the communication channels [5–7]. Moreover, the use of a centralized control (provider-centric) cannot deliver proper connectivity or even support computation closer to the edge of the network, thus becoming inefficient for highly distributed scenarios.

Mobile Edge Computing (MEC) can represent an option to increase the performance of CC applications, as it denotes a network architecture designed to provide low latency with adequate Quality of Service (QoS) to end-users [8,9]. Besides, MEC relies on top of high-speed mobile networks such as 5G to allow fast and stable connectivity for mobile devices and users. Thus, CC services can be deployed close to mobile devices, in the MEC layer, bringing processing and storage closer to cellular base stations [10].

Nevertheless, energy consumption remains a clear issue to be overcome on mobile device networks, such as MEC environments [11]. Most IoT sensors and mobile devices run on batteries with limited energy capacity. Furthermore, IoT devices need to handle lots of data, which is also

energy-consuming. Thus, reducing energy consumption in networks with IoT devices is a goal worth exploring.

State-of-the-art presents a set of studies that use MEC to offload tasks to offer local processing for IoT devices. Some works [4,12–16] have measured the energy consumption for data transmission or even using Dynamic Voltage and Frequency Scaling (DVFS) techniques. In contrast, this proposal enables a most detailed cost model, including energy and time consumed during processing and the cost of data transmissions. CC is also considered an option for processing when local resources are depleted, making the network more reliable in stress scenarios [17].

This work explores the scheduling problems in Edge Computing environments, considering energetic consumption in a dynamic cost model to mitigate energy consumption in MEC environments. An algorithm called the **T**ime and **E**nergy **M**inimization **S**cheduler (TEMS) scheduling algorithm implements dynamic cost minimization policies correlating the system resource allocation with the more inexpensive cost for executing tasks. A simulator has also been developed for the MEC evaluation with IoT devices and associated CC resources. The TEMS algorithm gathers data about the environment and associated energy and time costs to decision-making about the task scheduling. MEC Simulator is available at https://github.com/jlggross/MEC-simulator.

The main contributions of this work are:

i) The methodology covers a considerable number of energy and time metrics for task processing and data transmissions, including the accounting of CPU cores idle energy consumption;
ii) The CPU processing time and energy consumption optimization using DVFS technique;
iii) The scheduling policies consider task processing in the IoT device itself, in a local MEC server, and in a remote Data Center from CC at the same time;

The remainder of this paper is organized as follows. Section 2 discusses previous related work found in the literature. Section 3 declares the problems in MEC environments. Section 4 introduces the dynamic cost minimization model for the system with three different allocation policies, local processing in the IoT device, local processing in the MEC server, and remote CC processing. Section 5 introduces the TEMS heuristic scheduling algorithm designed to solve the cost minimization model of the system. Section 6 details the implementation and shows the results of the experiments using the TEMS scheduling algorithm. Finally, Section 7 presents the conclusions.

2. Related Work

The energy consumption decrease and mitigating response latency to applications in IoT environments are a well-known issue but an open question since the first Edge Computing architectures [18]. However, the strategic use of CC as the only alternative to task processing adds higher latency to IoT applications [19].

Few proposals use CC as an option to task execution [11,20,21]. Other approaches use Fog Computing to allow local processing in IoT devices such as the works [13,22–24] or without applying the CC [25]. The MEC architecture is assessed in the studies of [4,11,12,21]. In contrast, TEMS is a three-layer architecture that combines MEC and CC added to local IoT computation. This approach also provides a cost model, with the energy and run time evaluations on the fly, including the data transmission costs.

As for the parameters used in third-party cost models, the energy consumption of task processing is used by all works mentioned. However, the energy consumption for data transmissions is shown in the works [4,12–14]. The processing time of tasks is evaluated in major of the works, except to [22,24] and the spent time on data transmissions is limited to studies of [4,11–13,21].

On the other hand, the energy consumption of the equipment in the idle state is measured exclusively in [22,24]. Models that include the battery level of the IoT devices are only [4,12,13]. Our proposed model uses the DVFS technique [15,16] to allow both the dynamic minimization of energy and execution time during task processing.

All these execution time and energy consumption variables are consolidated in our model. A monitor for battery levels of IoT devices allows rational energy consumption. Two task policy types, critical - to deal with deadline constraints- and regular -to deal with common executions- are employed in the scheduling algorithm.

3. Problem Statement

A single cost model needs to evaluate deploying all used variables to data transmissions such as time and energy consumption. Also, must choose the local to task execution among MEC or Fog, or Cloud environments. This is a multiple-objective optimization problem. Therefore, find a minimal cost to all these system variables is an NP-Hard problem.

The solution must consider the energy consumption, such as the computation variables, energy to send tasks, energy consumption to data download, energy for the task processing, energy cost with CPU idle, and battery level. Also, it requires estimating the execution time to variables, like time spent on task transmission, wait time in queues, task runtime, and time spent on downloads. Simultaneously, the system must choose one among three distinct environments to produce the best performance considering energy optimize.

Thus, the computation to solve these issues is hard to model. Deciding between three different environments is another complex task due to needing to produce good data and task distributions. We propose a dynamic solution in real-time for each task using an Integer Linear Programming (ILP) optimization to achieve this challenge.

4. Model to minimize cost dynamically

This section introduces the model to minimize cost dynamically.

4.1. Architecture and Task Processing Flow

Figure 1 exhibits the architectural scheme on three decoupled layers under a bottom-up view:

- **IoT Layer (L1)**: IoT device layer generates application tasks. These devices have a limited processing capability and operate with batteries;
- **MEC Layer (L2)**: MEC server layer has a restricted number of CPUs and less processing capability than the CC environment. MEC servers are closer to the IoT devices, producing smaller communication delays;
- **CC Layer (L3)**: CC Data Centers compose this layer. These servers are high processing capability geographically distributed and far located from the IoT devices. They also add high network latency due to data transmission with more communication hops, if compared to other layers.

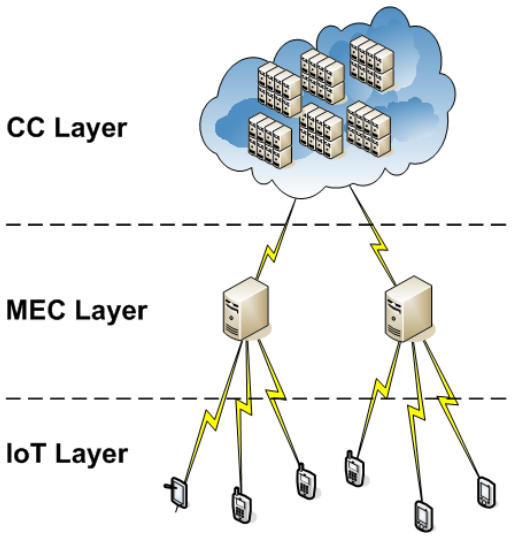


Figure 1. System architecture for the TEMS algorithm.

Table 1 summarizes the notation used throughout this proposal.

Table 1: Notation adopted for the model description.

Symbol	Description
D	A set of mobile IoT devices
j	An individual mobile device.
S	The number of MEC servers.
sc	The source code.
d	The input data.
r	Data transfer rate.
I	Mutual interference rate.
d'_i	The generated results forward back to the origin.
t	The deadline associated with the task.
W	The number of wireless channels.
A	The task set that will be executed.
i	An individual task.
H	The communication channel associated with the task.
h_i	The wireless channel associated to task i.
W	The bandwidth associated with the channel.
PL	A CPU core set.
k	An individual core.
$pl_{j,n}$	A core n for a mobile device j.
E	Energy consumption in idle time.
C	Effective commutative capacitance.
C_c	A CPU cycle for a CPU core.
Cc_T	Total clock cycles.
V_{local}	Voltage in the IoT processor device.
P	Power consumed.
T	Total execution time.
$T_{i,mec}$	Total execution time in the MEC server.
f	Processor frequency.
S_j	Local MEC server.
PS	A processor in the MEC server.
f_{mec}	Frequency of one core in a MEC server processor.
C_{mec}	Effective commutative capacitance in a processor in the MEC server.
V_{mec}	Voltage in the MEC processor device.
$g_{(S_l,j)}$	Channel gain between the local MEC server and the mobile device.
$P_{i,mec}$	The Consumed power in MEC server.
$E_{i,mec}$	The dynamic energy consumed in MEC server.
$Cost_{i,mec}$	The total cost in MEC server.

The model associates the cost in terms of energy consumed and elapsed time for the allocation policy of each layer, taking into account task processing and data transmissions costs. The DVFS technique is used to calculate processing costs, proving the best pair of CPU core voltage and CPU core operating frequency that reduces total cost. Finally, the TEMS scheduler seeks the best cost among all three allocation policies and selects the lowest one. The scheduler decides between MEC and CC layers to offload a task. Otherwise, the processing takes place on the device itself.

The rest of this section covers an extension of the cost model previous work shown in [26]. First, it is introduced assumptions about the network and the architecture components. After that, the cost models for local computing in the IoT device, local computing in the MEC server, and remote computing in the Cloud are shown. Finally, the individual costs are combined into a final equation that represents the total cost.

4.2. Network Model

The network is composed of mobile IoT devices, MEC servers, and a Cloud provider. The wireless links determine the communication channels between IoT devices and MEC servers, as in Figure 1. Each task represents a tuple $A_i = (C_i, sc_i, d_i, t_i)$ composed of CPU cycles needed to conclude an execution (Cc), the source code (sc), the input data (d), and the deadline (t) associated with the task. The deadline associated represents if a task is normal ($t = 0$) or it is critical ($t > 0$).

Also, each scheduled task has a wireless channel ($H = \{h_1, h_2, ..., h_a\}$) from the IoT device to MEC and from the MEC to CC with its correspondent bandwidth (W). If the task runs in the local device, it does not have an associated channel.

4.3. Local Computing in the IoT Device

Each mobile device has a respective number of CPU cores ($PL_j = \{pl_{j,1}, pl_{j,2}, ..., pl_{j,n}\}$). The energy consumption in *idle* time is computed in Equation 1 based on a total number of CPU cycles (Cc_i), operating frequency ($f_{local,j,k}$), voltage supply ($V_{local,j,k}$) and in the effective commutative capacitance ($C_{local,j,k}$) of each core, which dependents of the chip architecture [21]. Equation 1 computes the local dynamic energy consumed by the IoT device.

The total execution time of task is calculated based on total cycles C_{cT_i} [27] where $i \in A$ to a CPU core $j \in PL$ in Equation 2. The dynamic power consumed during the execution is $\propto CV^2 f$ [28] in Equation 3.

$$E_{i,local} = P_{i,local} * T_{i,local} \tag{1}$$

Where,

$$T_{i,local} = \frac{C_{cT_i}}{f_{local,j,k}} \tag{2}$$

$$P_{i,local} = C_{local,j,k} * V^2_{local,j,k} * f_{local,j,k} \tag{3}$$

Considering battery level and latency as model constraints, a device D_j must decide whether it is more appropriate to process 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. The local cost of one task i is expressed in Equation 4.

$$Cost_{i,local} = u_{localT} * T_{i,local,total} + u_{localE} * E_{i,local} \tag{4}$$

The coefficients $u_{localT} \in [0,1]$ and $u_{localE} \in [0,1]$ are weightings, where $u_{localT} + u_{localE} = 1$. These variables represent a trade-off between execution time and energy consumption and minimize one of the costs, according to Wang *et al.* [4].

4.4. Local Computing in the MEC Server

A local MEC server can have several CPU cores. Thus, the CPU cores 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 core $ps_{j,k}$ has an operating frequency ($f_{mec,j,k}$), an effective commutative capacitance ($C_{mec,j,k}$), and a supply voltage ($V_{mec,j,k}$).

IoT devices and MEC servers cause mutual interference between each other (I_i) because they use the same wireless channel. Thus, the data transfer rate ($r(h_i)$) to offload task i to the channel (h_i) attenuates according to *Shannon's formula* [21]. The data transfer rate is determined in Equation 5 and the mutual interference between wireless channels is computed in Equation 6.

$$r(h_i) = W * \log_2\left(1 + \frac{p_j * g_{(S_l,j)}}{N + I_i}\right) \tag{5}$$

$$I_i = \sum_{n \in A | \{i\}: h_n = h_i} p_{j'} * g_{(S'_l, j')} \tag{6}$$

For Equation 5, the variable p_j is the transmission power of a mobile device j during offloading task i to the local server, and N is the power of the thermal noise of the wireless channel.

In the local server, data and source code need to be sent to the application processing, and the generated results must be sent back to the origin. Thus, the time required for an IoT device to sent data (Equation 7) and after to do the result download (Equation 8) from the local server can be computed as.

$$T_{i,mec-up}(h_i) = \frac{sc_i + d_i}{r_i(h_i)} \tag{7}$$

$$T_{i,mec-down}(h_i) = \frac{d'_i}{r_i(h_i)} \tag{8}$$

The total time required to complete the task execution in the local server considers the send (Equation 7), the download (Equation 8), and the task execution time in the MEC server calculated like in Equation 2. The total time for a MEC server is given as in Equation 9.

$$\begin{aligned} T_{i,mec,total} = T_{i,mec-up}(h_i) + T_{i,mec} + \\ T_{i,mec-down}(h_i) \end{aligned} \tag{9}$$

The energy spent for the data communications from the IoT device to the local MEC server and vice versa is calculated by (Equation 1), which can be either the time to sent (*mec-up*) or download (*mec-down*) data. Furthermore, the dynamic energy consumed by the MEC server is calculated in the same fashion as that in the IoT device (Equation 1). Equation 10 gives the total dynamic energy consumption.

$$\begin{aligned} E_{i,mec,total} = E_{i,mec-up}(h_i) + E_{i,mec} + \\ E_{i,mec-down}(h_i) \end{aligned} \tag{10}$$

Moreover, the cost computation for the local server is expressed in Equation 11.

$$\begin{aligned} Cost_{i,mec} = u_{mecT} * T_{i,mec,total} + \\ u_{mecE} * E_{i,mec,total} \end{aligned} \tag{11}$$

4.5. Remote Computing in the Cloud

The CPU cores in CC are not distinguished because they are a single shared resource comparable to a CPU processor. It is not really a device. The CC equations are analogous to those of the local MEC server. Data transference between MEC and CC is composed of both the elapsed time and consumed energy to produce a total cost. The elapsed time is expressed by (Equation 12) and (Equation 13), while consumed energy is expressed in Equations 14 and 15.

$$T_{i,cloud-up} = \frac{s_i + d_i}{r} \tag{12}$$

$$T_{i,cloud-down} = \frac{d'_i}{r} \tag{13}$$

$$E_{i,cloud-up} = p_{wireless} * T_{i,cloud-up} \tag{14}$$

$$E_{i,cloud-down} = p_{wireless} * T_{i,cloud-down} \tag{15}$$

Note that in Equations 12 and 13, r is not dependent on h_i, because transmissions between MEC and CC are done on fiber optic cables, and there is no mutual interference effect attenuating the data transmission rate. CC processing time (Equation 16) and dynamic energy consumed (Equation 17) are calculated the same way for MEC servers. The total elapsed time and total energy consumption for CC are as follows.

$$T_{i,cloud,total} = T_{i,mec-up}(h_i) + T_{i,cloud-up} + T_{i,cloud} + T_{i,cloud-down} + T_{i,mec-down}(h_i) \quad (16)$$

$$E_{i,cloud,total} = E_{i,mec-up}(h_i) + E_{i,cloud-up} + E_{i,cloud} + E_{i,cloud-down} + E_{i,mec-down}(h_i) \quad (17)$$

Finally, the cost to run a single task i in the Cloud is given in Equation 18 as.

$$Cost_{i,cloud} = u_{cloudT} * T_{i,cloud,total} + u_{cloudE} * E_{i,cloud,total} \quad (18)$$

The *idle* energy cost of CC is not considered, since the CPU offer is virtually infinite. Therefore it does not make sense to account for this cost, which would cause the system to have equally infinite cost.

For every task i the minimum cost is chosen between all three allocation policies, one from each layer, as in Equation 19.

$$Cost_i = min(Cost_{i,local}, Cost_{i,mec}, Cost_{i,cloud}) \quad (19)$$

The total system cost, represented by Equation 20, is equal the sum of all task costs plus the idle energy of CPU cores from IoT devices and from MEC servers.

$$Cost_{system} = \sum_{i=1}^{A} Cost_i + E_{local,idle} + E_{mec,idle} \quad (20)$$

4.6. Model Constraints for IoT Device Battery

A healthy battery level is essential to the proper operation of IoT devices. If the battery level B_j of an IoT device j is below a *Lower Safety Limit* (LSL), task allocation on the device is disabled to keep the device alive with the remaining battery. If B_j reaches zero, all tasks generated by device j are canceled. Therefore, to prevent this from happening, the cost equations are subject to the following constraints: $B_j > E_{i,local}$, $B_j > E_{i,mec-up}(h)$. These constraints are considered in the scheduling algorithm.

5. The TEMS Algorithm

The Time and Energy Minimization Scheduler (TEMS) heuristic scheduling algorithm was developed in order to execute the dynamic cost minimization model with reduced computational cost.

Algorithm 1 exhibits four steps of TEMS. Step 1 is the step detection that forms a data set of IoT devices, MEC servers, and configuration of communication channels. The battery levels of the IoT devices are collected, and the LSL is established. The algorithm regards the number of cores into CPU available in each IoT device and MEC server, the operating frequency, and operating voltages. This process can also occur in CC Data Centers, but the number of CPUs is expected to be unlimited.

Algorithm 1: TEMS

```
Result: Task mapping to the processing nodes
execute Step 1 - Task of information collection and system setup
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
```

Algorithm 2 details step 2 from TEMS, which is the task allocation decision-making process of the scheduler. Here, firstly tasks are classified between critical and regular.

Algorithm 2: Step 2 – Task allocation

```
1  foreach task A_i from list of critical tasks do
2      foreach free CPU core pl_{j,k} ∈ PL_j do
3          policy 1: it calculates IoT device execution time
4      foreach free CPU core ps_{j,k} ∈ PS_j do
5          policy 2: it calculates MEC server execution time and transmission times
6      policy 3: it calculates CC execution time and transmission times
7      it is evaluate IoT battery level and offload task to the CPU core with minimal total time
8  foreach task A_i from list of regular tasks do
9      foreach free CPU core pl_{j,k} ∈ PL_j do
10         policy 1: it calculates energy consumption, execution time and cost for the IoT device
11     foreach free CPU core ps_{j,k} ∈ PS_j do
12         policy 2: it calculates energy consumption for dynamic processing and data transmission
13         policy 2: it calculates execution time and transmission times
14         policy 2: it calculates MEC server cost
15     policy 3: it calculates energy consumption for dynamic processing and data transmission
16     policy 3: it calculates execution time and transmission times
17     policy 3: it calculates CC total cost
18     it is evaluate IoT battery level and offload task to the CPU core with minimal cost
```

The time and energy consumption for task processing on the different CPU cores of the network is calculated, as well as the time and energy consumption of the data transmissions for MEC servers and CC Data Centers. Critical tasks are the first to be scheduled due to the sensitivity of the deadline. They are ordered by the deadline and allocated by the lowest total elapsed time. Then regular tasks are ordered by creation time and allocated by the minimum total cost. The battery level of IoT devices is continuously evaluated in lines 7 and 18 to check if the constraints are respected.

Step 3 monitors the task completion status. When one task was completed, the CPU core resources are released to turn available for new allocations in step 2. However, tasks that use CC resources do not need to release them since CC is supposed to have unlimited resources, absorbing any number of tasks. Task cancellation may occur if the elapsed time is higher than the deadline or if the IoT device runs out of battery.

Finally, in step 4, the battery level from each IoT device is collected, and after it creates new tasks again. Execution continues as long as tasks are being created.

5.1. Algorithm Complexity

The TEMS algorithm complexity analysis considers the four steps in Algorithm 1. The task of information collection and system setup occur a single time in the system setup. This step identifies "n" mobile devices added to the network, and it has "m" processor cores. There are a total of "n" local MEC servers with the "m" core processors and a number of "n" wireless network channels to "n" tasks with a tuple of four variables each. The algorithm must choose among "n" possible options with three variables each and nine coefficients to the cost equation. Thus for step 1, the complexity is as in Equation 21.

$$\begin{aligned} nm + nm + n + 4n + 3n + 9 = \\ 2nm + 8n + 9 = O(nm) \end{aligned} \tag{21}$$

However, as the smartphone nowadays has a limit of eight cores. Also, simple IoT devices, for instance, Arduino Mega 2560, have a single core. On the other hand, MEC servers can be composed of up to five Raspberry Pi IV with four cores. Thus, the processor cores number is less than the amount of then IoT devices, *i.e.*, $m << n$, and if m is a mensurable and a finite number, then it is reasonable to think that $m \approx k$ and in this scenario $O(nm) = O(kn) \approx O(n)$. Therefore, in step 1 the complexity is $O(n)$.

In step 2, the task allocation has a sort function with $O(nlog(n)$ complexity in the worst case and $O(n)$ in the best case. A seek is executed two times among n tasks into n local devices

and MEC servers to achieve the lower execution time and energy consumption. This task has $O(n^2)$ complexity. Also, the cloud allocation tasks have $O(1)$ complexity. TEMS algorithm was developed with Python programming, and the Python sorting executes three times. Thus, the complexity for step 2 is described as in Equation 22.

$$\begin{aligned} 3\,O(nlog(n)) + 2\,[O(n^2) + O(1)] = \\ 2\,O(n^2) + 3\,O(nlog(n)) + 2\,O(1) = O(n^2) \end{aligned} \tag{22}$$

Thus, the step 2 has a complexity $O(n^2)$.

Step 3 has n interactions of simple tasks, so $O(n)$, and step 4 seeks the battery level in n IoT devices, *i.e.*, $O(n)$.

Hence, considering all TEMS algorithm steps and exchanging these steps by respective individual complexity as in Equation 23.

$$O(TEMS) = O(n) + O(n^2) + O(n) + O(n) = O(n^2) \tag{23}$$

Therefore, the TEMS Algorithm complexity is $O(n^2)$.

6. Evaluation

This section explains the simulation details and the different experimental scenarios used.

6.1. Simulated Hardware and Software Stack

The simulated environment was designed with low, mid-range and high processing power devices for IoT, MEC and CC layers, respectively. For IoT devices we chose Arduino Mega 2560, with five operating frequencies and corresponding supply voltages for DVFS. The MEC servers were simulated on top of 5 Raspberry Pi 4 Model B boards, each board with a Quad-core Cortex-A72 1.5GHZ (ARM v8) 64-bit, summing a total of 20 CPU cores per server. These CPU cores have three operating frequencies and corresponding supply voltages. Table 2 specifies the voltage-frequency pairs, and the capacitance of the underlying hardware architecture of IoT and MEC devices. These values combined are used to calculate the power consumed by a device according to the selected values.

Table 2: Devices variables for power calculation.

Hardware	**Voltage-Frequency Pairs**	**Capacitance**
IoT device	(5V-16MHz), (4V-8MHz), (2,7V-4MHz), (2.3V-2MHz), (1.8V-1MHz)	2.2 nanoFarad
MEC Server	(1.2V-1,500MHz), (1V-1,000MHz), (0.825V-750MHz), (0.8V-600MHz)	1.8 nanoFarad

For CC we chose Data Centers with Intel Xeon Cascade Lake processors of 2.8 GHz per CPU core, reaching up to 3.9 GHz with *Turbo Boost* on[1]. Here, there are no voltage or capacitance variables. Instead, the resulting power is used, 13.85 Watts and 24.28 Watts, for configuration with and without Turbo Boost, respectively.

The network throughput was configured to achieve up to 1 Gbps speed and latencies to 5ms, for both 5G and fiber optics communications [29] [30]. Moreover, two vehicular applications were defined [31], one with high processing workload and high task creation time (Application 1), and another with low processing workload and low task creation time (Application 2). Application 2 creates more tasks than Application 1 in the same time interval, but each task has lower processing requirements. Table 3 shows the characteristics of each application.

[1] Technical information can be found in the datasheets of the electronic components.

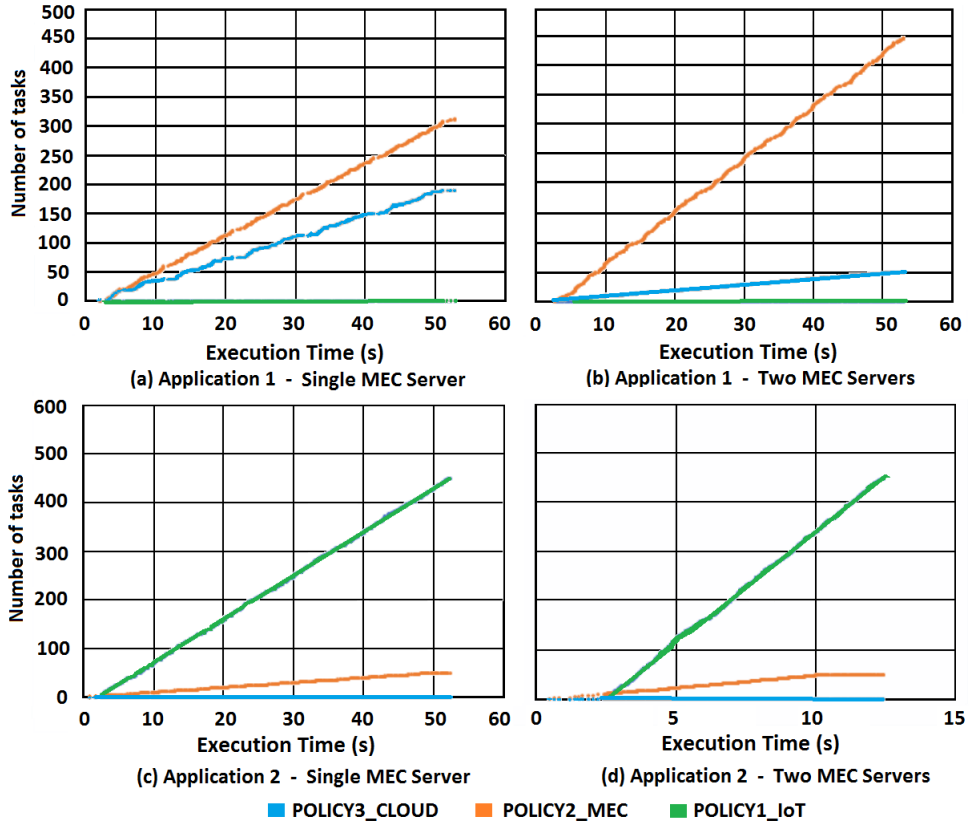


Figure 2. Task allocation for Application 1 and Application 2.

Table 3: Characteristics of chosen applications.

Characteristics	Application 1	Application 2
Task generation rate (s)	10	0.1
Input Data (MB)	36.3	4
Result data size (bytes)	1,250	625
Computational workload (Millions of CPU cycles)	2,000	20
Critical Tasks (% from total tasks)	10	50
Deadline for critical tasks (milliseconds)	500	100

6.2. Experiments and Results

The tested scenarios used different configurations for parameters such as the computational workload of each task, coefficients for energy consumption, and elapsed time. Also, it evaluated the size of data entry and results, task generation rate, deadline of critical tasks, level of batteries for IoT devices, and use of DVFS. The main goal is to see the TEMS algorithm respond to task allocation and energy and time reduction. The results are discussed below.

6.2.1. Use of MEC servers

This experiment evaluates how TEMS behaves when varying the number of MEC servers in the system. Application 1 is used and the workload is configured according to description in Table 3. The tested scenario has 500 tasks distributed to 100 IoT devices in two different cases, one with a single MEC server, in Figure 2.(a) and (c) and another with two MEC servers, in Figure 2.(b) and (d). Figure 2 depicts the results for the execution of Application 1 and Application 2 in both cases with $10 \cdot 10^6$ CPU cycles

The energy and time coefficients were set, respectively, to $4/5$ and $1/5$, that is, a high weight was given to the energy consumed so that it could be minimized. In Figure 2 from plot 2.a to plot 2.b and from plot 2.c to plot 2.d there is an increase in the number of MEC servers,

from one to two, which made fewer tasks be to allocated in the CC layer. This positively impacts the total energy consumed because tasks running in the MEC layer demand less energy when the workload is higher. However, when the workload is composed of little tasks with a high transfer rate, the scheduler tends to maintain all tasks nearest from devices due to deadline restrictions.

Table 4 shows the relationship between the number of MEC Servers related to energy consumption. When comparing cases A and B with a third case C with no MEC servers, the reduction in energy consumption for case A was 42.51%, while for case B 44.71%. In case C, tasks are just offloaded to the Cloud, adding too much energy consumption to the system. Thus, the use of MEC servers helps the system to lower the total energy consumed.

Table 4: The MEC Server benefit related to the energy consumption.

Variables	**One MEC Case A**	**Two MEC Case B**	**Without MEC Case C**
$E_{CPU}(J)$	2,752.26	1,725.18	5,074.35
$E_{Trans}(s)$	835.60	1,725.17	1,166.21
$E_{mec}(J)$	3,587.86	3,450.35	6,240.56
$T_{CORE}(s)$	956.21	1,225.64	347.07
$T_{Trans}(s)$	199.75	159.69	290.31
$T_{TOTAL}(s)$	1,155.96	1,385.33	637.38
$Trans = T_{mec-up} + T_{mec-down}$			

With Application 2, that has lower workload compared to Application 1, the allocation profile changed. Most allocations took place on the device itself, regardless of the number of MEC servers. The cause to this phenomenon is due to the low processing workload of Application 2. The hardware of 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. Thus, for a small processing workload, IoT devices are the first allocation option.

6.2.2. IoT device battery consumption

Figure ?? shows an experiment executed for Application 2 with 10,000 tasks, 100 IoT devices and 2 MEC servers. Initially, Figure 3(a) shows the tasks are allocated according to their type. Regular tasks run on the IoT device itself due to the lower cost among all allocation policies, while critical tasks run on the server, as the total time is reduced compared to the IoT device, even though the energy cost is higher. Thus, tasks are distributed for local processing in the IoT device and in the MEC server.

Figure 3(b) represents the battery level of one IoT device of the system. At around 15 microseconds, the battery level reaches the LSL, which corresponds to 10% of the maximum battery capacity. From this moment forward, TEMS no longer allows tasks to run on the IoT devices, causing a sudden increase in the number of allocations to the MEC server for the newly created tasks.

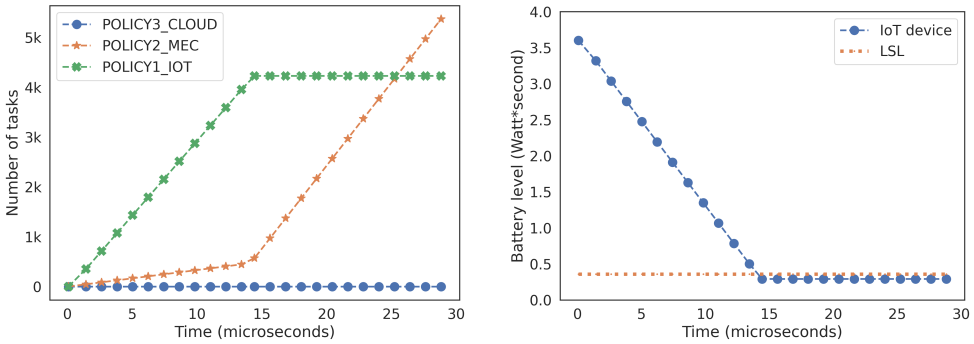
Our analysis indicates that low battery levels quickly reach LSL and make IoT devices unavailable for processing. High computational workloads also negatively affect the battery level. Therefore, a battery with a healthy energy level and adequate task processing workloads, allows the allocation to be performed on the IoT device, without making it unavailable due to lack of battery, contributing to total cost reduction.

6.2.3. Variation of input data size

This experiment evaluates how the costs of each allocation policy change according to the data size for tasks from Application 1. We built a simulation with 500 tasks, 100 IoT devices, two MEC servers and energy cost coefficient configured to $4/5$. Four different input sizes were used, 3.6 MB, 36 MB, 362 MB, and 3.6 GB.

As stated in Figure 4, as data entry size increases, the calculated costs progress in the MEC and CC allocation policies. The cost to execute in the IoT devices remains the same, as no data transmissions are carried out. When data entry size scales, allocation policies that require

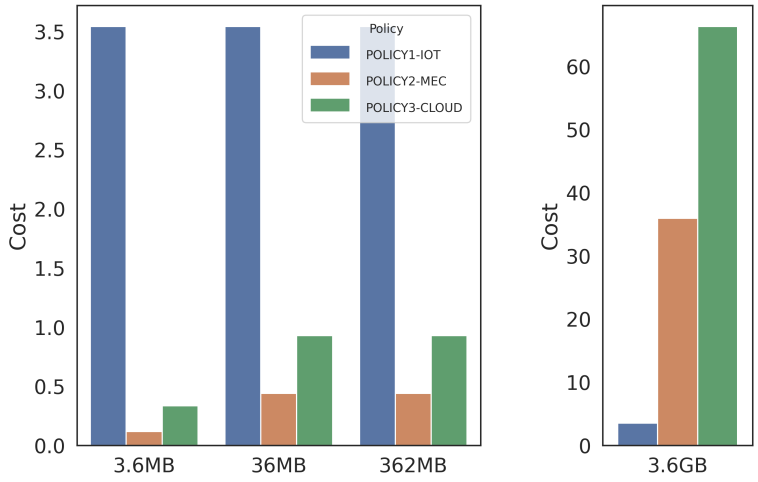
Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 10 March 2021 doi:10.20944/preprints202103.0285.v1



(a) Task allocation in the system. (b) Battery consumption of IoT devices.

Figure 3. Task allocation behavior for Application 2 when IoT device battery level reaches the LSL.

data transmissions become very costly, and allocation on the device itself turns increasingly advantageous. This increase in cost for MEC and CC policies is quite evident in Figure 4(b), for inputs of 3.6 GB, even the cost scale had to be adjusted to represent the values better. Therefore, it is crucial to design applications so that data transfers over the network are not too large per task, avoiding high data transmission costs.



(a) Cases A, B and C. (b) Case D.

Figure 4. Policy costs for input data size variation on Application 1.

We also designed two other cases, one with 5,000 tasks and 362 MB per task and another with 500 tasks and 3.6 GB per task, with roughly 1.8 TB in total each. The system energy consumption and the total elapsed time for the 500 tasks case were 59,160.92 Joules (J) and 14,515.21 seconds. For the experiment with 5,000 tasks, the costs were 45,011.49 J and 10,305.94 s, that is, a decrease of 23.92% and 29%, respectively. Therefore, tasks should preferably not be super data-intensive, if dependent on MEC or CC, as data transmissions add additional energy and time expenses.

6.2.4. Use of different energy and time coefficients

This experiment used Application 2 in four different scenarios. Each case with 500 tasks, 100 IoT devices, and one MEC server. The energy coefficients were set to $1/5$, $2/5$, $3/5$, $4/5$ and the time coefficients to $4/5$, $3/5$, $2/5$ and $1/5$.

Table 5 lists the minimum costs perceived by the system task scheduler for each case. The lowest calculated cost was the same for cases 2 and 3, with MEC as an offloading option. Case 1

had the lowest calculated cost among all coefficient pairs. In these three cases the time coefficient had high values, and MEC was chosen because task execution got the lowest processing times. For case 4, the allocation took place on the IoT device itself, with DVFS configured at 8 MHz and 4 V. Now, energy has a high-value coefficient, which made the scheduler choose the policy that provided the lowest energy cost, reducing total cost.

Table 5: Costs for Application 2, varying the cost coefficients for energy and time.

Case	u_E	u_T	Cost 10^{-3}	E_{Total} mJ	T_{Total} ms	Freq. MHz	Voltage V	Policy
C1	1/5	4/5	18.59	145.50	33.36	1,500	1.20	MEC
C2	2/5	3/5	25.97	142.76	34.69	750	0.83	MEC
C3	3/5	2/5	33.18	142.76	34.69	750	0.83	MEC
C4	4/5	1/5	35.44	70.40	250.00	8	4.00	IoT

To reduce energy consumption, the best option is to use $4/5$ as an energy coefficient. With this configuration, the minimization of energy consumption is prioritized, saving up to 51.6% compared to the other cases. Alternatively, to reduce task completion time, coefficients from cases 1, 2 and 3 are better, with a reduction of up to 86.6% compared to case 4. For cases 1 to 3, a considerable reduction in total elapsed-time was perceived because running the task in the IoT device for the evaluated application made the time component spike.

6.2.5. Impact of task generation rate variation

In this experiment we chose Application 2 to execute on four scenarios, with task generation rates of 0.05, 0.1, 0.2, and 0.3 seconds. All scenarios were configured with 500 tasks, 100 IoT devices and 1 MEC server. Figure 5 shows that small task generation rates flood the network with tasks, rapidly consuming all local resources. As an effect, TEMS allocates the majority of tasks to the CC layer, even though time and energy costs are higher, because no local CPU core is available. Although, when task generation rate increases, tasks enter the network in more sparse time periods, and local resources can absorb much of the processing demands. In this scenario, less offloading is done to the CC layer, reducing total system costs.

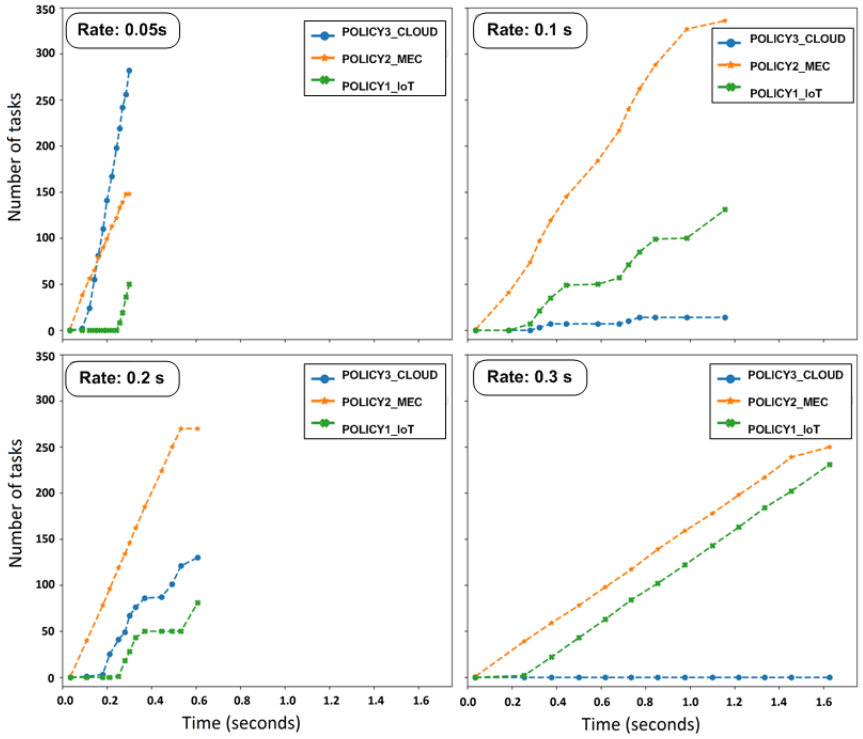


Figure 5. Task allocation for different task generation rates.

Task generation rates should be designed with a time interval that favors usage of local resources, which may help reduce total costs depending on the application and the costs of each allocation policy.

By configuring deadlines with very restrictive time limits, the experiments showed that critical tasks were canceled because the scheduler could not find an allocation policy to achieve task completion. To avoid this behavior, deadlines must be appropriately configured so that at least one allocation policy has sufficient time to process and complete the task correctly.

6.2.6. Using the DVFS technique

With DVFS enabled, total energy consumption decreased by 13.74%, while total time increased by 28.32% compared to DVFS off. This demonstrates the effectiveness of the proposed model and the scheduling algorithm in minimizing the total energy consumption. Although the whole time may have been longer in the approach with DVFS, it is not a problem because tasks were completed within the time limit imposed by the deadline.

Challenges about the time complexity of the DVFS technique have already been discussed by Chen *et al.* [32]. Our model addresses this problem by limiting the possible voltage-frequency pairs used to calculate dynamic power for task execution, allowing results to be obtained in feasible execution time.

7. Conclusion

Energy and time reduction are mostly needed for environments where large volumes of data and mobile devices are connected to the Internet with restricted QoS requirements and battery limitations. The TEMS algorithm chooses the most suitable allocation options in the system, reducing energy waste and elapsed time. The experiments indicated that the cost coefficient regulations are essential for the final cost perceived by the scheduling algorithm. Adequate coefficients allowed a decrease of energy consumption up to 51.6% and an execution time reduction up to 86.6%, ending critical tasks inside the deadline. Thus, the system becomes more sustainable, and the user experience is kept stable.

The use of MEC servers helps increase the battery life of the IoT devices and enables agile task execution. Moreover, using the DVFS technique caused exciting results, supporting the energy consumption decrease. This work allowed contributions such as the TEMS algorithm and combining data transmission to the cost model. The model also considers idle costs, data transmission rate interference, using the DVFS technique, and the interaction with the CC layer to provide computational resources whenever the local network becomes overloaded.

As future works, we can indicate the progression of the system cost model to more fine grain, with the insertion of new variables and new environments to explore applications in different scenarios such as industry, healthcare, aviation, and mining.

Acknowledgments: "SmartSent" (#17/2551-0001 195-3), CAPES (Finance Code 001), PNPD program, CNPq, PROPESQ-UFRGS-Brasil and FAPERGS Project "GREEN-CLOUD - Computação em Cloud com Computação Sustentável" (#16/2551-0000 488-9). Proyeto Uso de algoritmos y protocolos de comunicación en dispositivos con énfasis en la privacidad de los datos and Laboratório de Telecomunicações de Portugal IT—Branch Universidade da Beira Interior, Covilhã.

Funding: This work has partially supported by PROPESQ-UFRGS-Brasil and by "Fundação para a Ciência e a Tecnologia" under Projects UIDB/04111/2020 and FORESTER PCIF/SSI/0102/2017. And Junta De Castilla y León—Consejería De Economía Y Empleo: System for simulation and training in advanced techniques for the occupational risk prevention through the design of hybrid-reality environments with ref. J118.

Author Contributions: Conceptualization, J.C.S.A, J.L.G.G and K.J.M; methodology, J.C.S.A, J.L.G.G and C.F.R.G; simulator, J.L.G.G; validation, J.C.S.A and J.L.G.G; writing–original draft preparation, J.C.S.A, K.J.M, J.L.G.G and V.R.Q.L; writing–review and editing, G.V.G and J.C.S.A

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

Abbreviations

The following abbreviations are used in this manuscript:

IDC	International Data Corporation
CC	Cloud Computing
MEC	Mobile Edge Computing
QoS	Quality of Service
DVFS	Dynamic Voltage and Frequency Scaling
TEMS	Time and Energy Minimization Scheduler
ILP	Integer Linear Programming
CPU	Central Processing Unit
IoT	Internet of Things
LSL	Lower Safety Limit
ARM	Advanced RISC Machines

References

1. Reinsel, D.; Gantz, J.; Rydning, J. *The Digitalization of The World: From Edge to Core*, us44413318 ed.; Vol. 1, *IDC White Paper*, Seagate Inc: Framingham, Massachusetts, 2018; pp. 1–28.
2. Chen, T.Y.H.; Ravindranath, L.; Deng, S.; Bahl, P.; Balakrishnan, H. Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices. Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems; Association for Computing Machinery: New York, NY, USA, 2015; SenSys '15, pp. 155–168. doi:10.1145/2809695.2809711.
3. Matteussi, K.J.; Zanchetta, B.F.; Bertoncello, G.; Dos Santos, J.D.D.; Dos Anjos, J.C.S.; Geyer, C.F.R. Analysis and Performance Evaluation of Deep Learning on Big Data. 2019 IEEE Symposium on Computers and Communications (ISCC), 2019, pp. 1–6. doi:10.1109/ISCC47284.2019.8969762.
4. Wang, C.; Dong, C.; Qin, J.; Yang, X.; Wen, W. Energy-efficient Offloading Policy for Resource Allocation in Distributed Mobile Edge Computing. 2018 IEEE Symposium on Computers and Communications (ISCC), 2018, pp. 00366–00372. doi:10.1109/ISCC.2018.8538612.
5. Matteussi, K.J.; Geyer, C.F.R.; Xavier, M.G.; Rose, C.A.F.D. Understanding and Minimizing Disk Contention Effects for Data-Intensive Processing in Virtualized Systems. 2018 International Conference on High Performance Computing Simulation (HPCS), 2018, pp. 901–908. doi:10.1109/HPCS.2018.00144.
6. Aijaz, A. Towards 5G-enabled Tactile Internet: Radio resource allocation for haptic communications. 2016 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), 2016, pp. 145–150.
7. Sales Mendes, A.; Jiménez-Bravo, D.M.; Navarro-Cáceres, M.; Reis Quietinho Leithardt, V.; Villarrubia González, G. Multi-Agent Approach Using LoRaWAN Devices: An Airport Case Study. *Electronics* **2020**, *9*. doi:10.3390/electronics9091430.
8. Haouari, F.; Faraj, R.; AlJa'am, J.M. Fog Computing Potentials, Applications, and Challenges. 2018 International Conference on Computer and Applications (ICCA), 2018, pp. 399–406. doi:10.1109/COMAPP.2018.8460182.
9. Silva, L.A.; Leithardt, V.R.Q.; Rolim, C.O.; González, G.V.; Geyer, C.F.R.; Silva, J.S. PRISER: Managing Notification in Multiples Devices with Data Privacy Support. *Sensors* **2019**, *19*. doi:10.3390/s19143098.
10. Yu, Y. Mobile edge computing towards 5G: Vision, recent progress, and open challenges. *China Communications* **2016**, *13*, 89–99. doi:10.1109/CC.2016.7833463.
11. Sarangi, S.R.; Goel, S.; Singh, B. Energy Efficient Scheduling in IoT Networks. Proceedings of the 33rd Annual ACM Symposium on Applied Computing; Association for Computing Machinery: New York, NY, USA, 2018; SAC '18, pp. 733–740. doi:10.1145/3167132.3167213.
12. Zhang, G.; Zhang, W.; Cao, Y.; Li, D.; Wang, L. Energy-Delay Tradeoff for Dynamic Offloading in Mobile-Edge Computing System With Energy Harvesting Devices. *IEEE Transactions on Industrial Informatics* **2018**, *14*, 4642–4655. doi:10.1109/TII.2018.2843365.
13. Gedawy, H.; Habak, K.; Harras, K.A.; Hamdi, M. Awakening the Cloud Within: Energy-Aware Task Scheduling on Edge IoT Devices. 2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), 2018, pp. 191–196. doi:10.1109/PERCOMW.2018.8480266.
14. Skarlat, O.; Schulte, S.; Borkowski, M.; Leitner, P. Resource Provisioning for IoT Services in the Fog. 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), 2016, pp. 32–39.
15. Chen, Y.L.; Chang, M.F.; Yu, C.W.; Chen, X.Z.; Liang, W.Y. Learning-Directed Dynamic Voltage and Frequency Scaling Scheme with Adjustable Performance for Single-Core and Multi-Core Embedded and Mobile Systems. *Sensors* **2018**, *18*, 3068. doi:10.3390/s18093068.
16. Jin, X.; Goto, S. Hilbert Transform-Based Workload Prediction and Dynamic Frequency Scaling for Power-Efficient Video Encoding. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **2012**, *31*, 649–661. doi:10.1109/TCAD.2011.2180383.
17. Anjos, J.C.S.; Matteussi, K.J.; De Souza, P.R.R.; Grabher, G.J.A.; Borges, G.A.; Barbosa, J.L.V.; González, G.V.; Leithardt, V.R.Q.; Geyer, C.F.R. Data Processing Model to Perform Big Data Analytics in Hybrid Infrastructures. *IEEE Access* **2020**, pp. 1–1. doi:10.1109/ACCESS.2020.3023344.
18. Satyanarayanan, M.; Bahl, P.; Caceres, R.; Davies, N. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing* **2009**, *8*, 14–23. doi:10.1109/MPRV.2009.82.
19. Alkhalaileh, M.; Calheiros, R.N.; Nguyen, Q.V.; Javadi, B. Data-intensive application scheduling on Mobile Edge Cloud Computing. *Journal of Network and Computer Applications* **2020**, *167*, 102735. doi:j.jnca.2020.102735.
20. Bui, N.H.; Pham, C.; Nguyen, K.K.; Cheriet, M. Energy efficient scheduling for networked IoT device software update. 2019 15th International Conference on Network and Service Management (CNSM). IEEE, 2019, pp. 1–5.

21. Yu, H.; Wang, Q.; Guo, S. Energy-Efficient Task Offloading and Resource Scheduling for Mobile Edge Computing. 2018 IEEE International Conference on Networking, Architecture and Storage (NAS), 2018, pp. 1–4. doi:10.1109/NAS.2018.8515731.
22. Wan, J.; Chen, B.; Wang, S.; Xia, M.; Li, D.; Liu, C. Fog Computing for Energy-Aware Load Balancing and Scheduling in Smart Factory. *IEEE Transactions on Industrial Informatics* **2018**, *14*, 4548–4556. doi:10.1109/TII.2018.2818932.
23. Galache, J.A.; Yonezawa, T.; Gurgen, L.; Pavia, D.; Grella, M.; Maeomichi, H. ClouT: Leveraging Cloud Computing Techniques for Improving Management of Massive IoT Data. 2014 IEEE 7th International Conference on Service-Oriented Computing and Applications, 2014, pp. 324–327.
24. Wu, H.; Lee, C. Energy Efficient Scheduling for Heterogeneous Fog Computing Architectures. 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), 2018, Vol. 01, pp. 555–560. doi:10.1109/COMPSAC.2018.00085.
25. Anjos, J.C.S.; Matteussi, K.J.; De Souza, P.R.R.; da Silva Veith, A.; Fedak, G.; Barbosa, J.L.V.; Geyer, C.R. Enabling Strategies for Big Data Analytics in Hybrid Infrastructures. 2018 International Conference on High Performance Computing Simulation (HPCS), 2018, pp. 869–876. doi:10.1109/HPCS.2018.00140.
26. Gross, J.L.G.; Matteussi, K.J.; dos Anjos, J.C.S.; Geyer, C.F.R. A Dynamic Cost Model to Minimize Energy Consumption and Processing Time for IoT Tasks in a Mobile Edge Computing Environment. Service-Oriented Computing; Kafeza, E.; Benatallah, B.; Martinelli, F.; Hacid, H.; Bouguettaya, A.; Motahari, H., Eds.; Springer International Publishing: Cham, 2020; Vol. 12571, p. 101–109. doi:10.1007/978-3-030-65310-1_8.
27. Tanenbaum, A.S.; Austin, T. *Structured Computer Organization*, 6th ed.; Prentice Hall, 2012.
28. Liu, Y.; Yang, H.; Dick, R.P.; Wang, H.; Shang, L. Thermal vs Energy Optimization for DVFS-Enabled Processors in Embedded Systems. 8th International Symposium on Quality Electronic Design (ISQED'07), 2007, pp. 204–209. doi:10.1109/ISQED.2007.158.
29. Gupta, A.; Jha, R.K. A Survey of 5G Network: Architecture and Emerging Technologies. *IEEE Access* **2015**, *3*, 1206–1232. doi:10.1109/ACCESS.2015.2461602.
30. Brogi., A.; Forti., S.; Ibrahim., A. Deploying Fog Applications: How Much Does It Cost, By the Way? Proceedings of the 8th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,. INSTICC, SciTePress, 2018, pp. 68–77. doi:10.5220/0006676100680077.
31. Jansson, J. Collision Avoidance Theory with Application to Automotive Collision Mitigation. PhD thesis, Department of Electrical Engineering Linköping University, SE–581 83 Linköping, Sweden, 2005.
32. Chen, Y.L.; Chang, M.F.; Yu, C.W.; Chen, X.Z.; Liang, W.Y. Learning-Directed Dynamic Voltage and Frequency Scaling Scheme with Adjustable Performance for Single-Core and Multi-Core Embedded and Mobile Systems. *Sensors* **2018**, *18*, 3068. doi:10.3390/s18093068.