A Dynamic Cost Model to Minimize Energy Consumption and Processing Time for IoT Tasks in a Mobile Edge Computing Environment

João Luiz Grave Gross, Kassiano José Matteussi,
Julio C. S. dos Anjos, and Cláudio Fernando Resin Geyer

Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, RS, Brazil, 91509–900
{jlggross,kjmatteussi,jcsanjos,geyer}@inf.ufrgs.br
https://www.inf.ufrgs.br/

Abstract. The rapid growth of IoT devices and applications with data-intensive processing has led to energy consumption and latency concerns. These applications tend to offload task processing to remote Data Centers in the Cloud, distant from end-users, increasing communication latency and energy costs. In such a context, this work proposes a dynamic cost model to minimize energy consumption and total elapsed time for IoT devices in Mobile Edge Computing environments. The solution presents a Time and Energy Minimization Scheduler (TEMS), validated through simulation, which resulted in a reduction in energy consumption by up to 51.61% and in task completion time by up to 86.65%.

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

1 Introduction

Billions of smart devices can now connect to the Internet in the form of *Internet of Things* (IoT) and applications have evolved, especially those used in artificial intelligence and computer vision, and require high computing power [6]. For these applications, IoT devices usually rely on task processing offload to remote *Cloud Computing* (CC) Data Centers, far away from the end-user, to boost processing time and reduce battery energy consumption. This results in higher latency, which became inefficient for high distributed scenarios [1].

Mobile Edge Computing (MEC) is a suitable alternative to CC, as it provides low latency and better QoS to end-users [3]. It relies on top of high-speed mobile networks such as 5G to allow fast and stable connectivity for mobile devices and users. But energy consumption remains an open issue [7], because most IoT devices run on batteries with limited energy capacity and need to handle lots of data, which is energy-consuming. Thus, reducing energy consumption in networks with IoT devices is a goal worth exploring.

This work tackles these issues by proposing a dynamic cost model to minimize energy consumption and task processing time for IoT devices in MEC environments. Also, we propose the TEMS scheduling algorithm that implements the

dynamic cost minimization model, which calculates the cost of different allocation options and chooses one that yields the lesser cost for the execution of tasks. Finally, the efficiency of TEMS is evaluated through simulation[1].

The main contributions of this work are i) we evaluate tasks with different profiles such as critical tasks (with a deadline) and non-critical tasks (without a deadline) in a variety of case scenarios; ii) Our methodology covers a considerable number of energy and time metrics for task processing and data transmissions, including the accounting of CPU cores idle energy; iii) The model uses a DVFS technique aiming to optimize CPU cores processing time and energy consumption; iv) Our model considers three possible processing sites, including processing in the IoT device itself, in a local MEC server and in a remote Data Center from CC; v) We develop and adapt experiments based on a well-defined simulation tool for scenarios with IoT devices, MEC servers, and CC Data Centers.

2 Dynamic Cost Minimization Model

In this section, we introduce a detailed view of our dynamic cost minimization model. Figure 1 presents the architecture design of the system decoupled in three layers, following a bottom-up approach:

- **IoT Layer (L1)**: Composed of IoT devices, which generate the application tasks. They run on batteries and have limited processing power.
- **MEC Layer (L2)**: Composed of MEC servers with a limited number of CPU cores and mid-range processing power. They are close to the end-users, providing small communication delays.
- **CC Layer (L3)**: Composed of Data Centers from CC. They are far located from the end-users, and geographically distributed, with high processing power and high network latency.

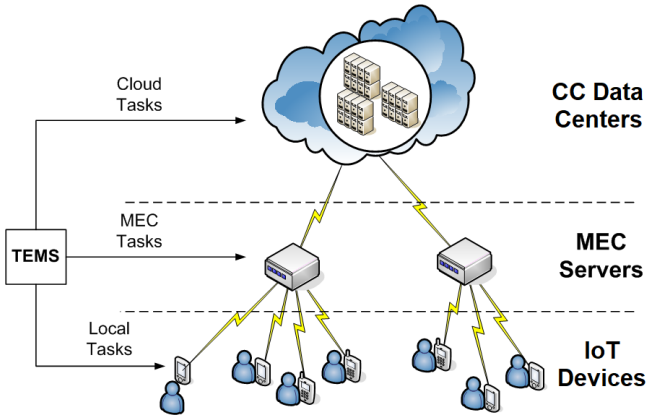

Fig. 1. System architecture and task allocation policies.

[1] MEC Simulator available at https://github.com/jlggross/MEC-simulator.

For the system we assume set D to be the mobile IoT devices, S the local MEC servers, W the wireless communication channels and A the tasks. Each task A_i is represented by a tuple $A_i = (C_i, s_i, d_i, t_i)$, where $i \in A$.

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 source code and data entry sizes. t_i represents the deadline of the task, which is the maximum time to complete its execution. Tasks can be normal or critical if the deadline is positive.

Also, for every task scheduled to MEC or CC, a wireless channel is associated, which is represented by set H. If a task i is associated with a channel w, then $h_i = w$, otherwise $h_i = 0$ for computation in the IoT device, where h_i is an element of H. Every $h_i \in W \cup 0$.

2.1 Local Computing in the IoT Device

Each mobile device $j \in D$ has one or more CPU cores, which is given by $PL_j = \{pl_{j,1}, pl_{j,2}, ..., pl_{j,n}\}$. We compute *idle* energy for vacant CPU cores, and dynamic energy ($E_{i,local}$) for occupied ones. Each core has an operating frequency ($f_{local,j,k}$), an effective commutative capacitance ($C_{local,j,k}$) [9], and a voltage supply ($V_{local,j,k}$). Each task i has a total number of CPU cycles (CT_i) for its execution.

The total execution time [9] (Equation 1), dynamic power (Equation 2), and dynamic energy consume [5] (Equation 3) can be calculated as:

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

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

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

The lesser cost for the system, that provides a better relationship between battery energy consumption and latency, can be expressed by:

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

In Equation 4 u_{localT} and $u_{localE} \in [0, 1]$, and $u_{localT} + u_{localE} = 1$. As mentioned in [8] these coefficients are used to represent the weight of time and energy, and work as a trade-off to prioritize the minimization of one of the costs.

2.2 Local Computing in the MEC Server

Every local MEC servers has multiple CPU cores, which is given by $PS_j = \{ps_{j,1}, ps_{j,2}, ..., ps_{j,n}\}$. A $ps_{j,k}$ core has an operating frequency ($f_{mec,j,k}$), an effective commutative capacitance ($C_{mec,j,k}$), and a supply voltage ($V_{mec,j,k}$). Communications between IoT devices and local servers that use the same wireless channel cause mutual interference between each other (I_i). In this case, the data transfer rate ($r_i(h_i)$) attenuates according to *Shannon's formula* [9]:

$$r_i(h_i) = W * \log_2 \left(1 + \frac{p_m * g_{j,m}}{N + I_i}\right) \tag{5}$$

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

For Equation 5, W is the channel bandwidth, $g_{j,m}$ is the channel gain between a mobile device m and a local server j. The variable p_m is the transmission power of m when offloading task i to j. N is the power of the thermal noise of the wireless channel, and h_n the wireless channel for task n.

The time required to send data and source code, and download results from an IoT device to a local server are shown below. Also, the total time accounts for these two times plus the task execution time $T_{i,mec}$, calculated the same way as for the IoT devices.

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

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

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

The energy consumed by the data transmissions is calculated by the transmission power ($p_{wireless}$) times the elapsed time ($T_{i,mec-up}(h_i)$ or $T_{i,mec-down}(h_i)$), resulting in $E_{i,mec-up}(h_i)$ and $E_{i,mec-down}(h_i)$, respectively. Finally, the dynamic energy consumed ($E_{i,mec}$) is calculated as $P_{i,mec} * T_{i,mec}$, and the total dynamic energy consumption is given by:

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

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

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

2.3 Remote Computing in the Cloud

CC is assumed to have unlimited resources, which is why cores are not distinguished, and *idle* energy not computed. The CC Data Center formulas are very similar from the MEC server formulas, but with some more components such as time spent and energy consumed for transmissions between MEC and CC layers.

$$\begin{aligned} T_{i,cloud,total} = {} & T_{i,mec-up}(h_i) + T_{i,cloud-up} + T_{i,cloud} + \\ & T_{i,cloud-down} + T_{i,mec-down}(h_i) \end{aligned} \tag{12}$$

$$\begin{aligned} E_{i,cloud,total} = {} & E_{i,mec-up}(h_i) + E_{i,cloud-up} + E_{i,cloud} + \\ & E_{i,cloud-down} + E_{i,mec-down}(h_i) \end{aligned} \tag{13}$$

Finally, the cost to run a single task i in CC is given by:

$$Cost_{i,cloud} = u_{cloudT} * T_{i,cloud,total} + u_{cloudE} * E_{i,cloud,total} \tag{14}$$

2.4 System Dynamic Cost Minimization Equation

For every task i the minimum cost is chosen between all three allocation options, one from each layer. The total system cost is equal the sum of all task costs and idle energy costs.

$$Cost_i = min(Cost_{i,local}, Cost_{i,mec}, Cost_{i,cloud}) \tag{15}$$

$$Cost_{system} = \sum_{i=1}^{A} Cost_i + E_{local,idle} + E_{mec,idle} \tag{16}$$

3 Time and Energy Minimization Scheduler (TEMS)

The TEMS heuristic scheduling algorithm executes the dynamic cost minimization model with reduced computational cost. It has complexity $O(n^2)$.

Algorithm 1: Time and Energy Minimization Scheduler (TEMS)

Result: Task mapping to the processing nodes
execute Step 1 - Collection of system information and initialization
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*;

In Algorithm 1 are presented the steps of TEMS. Step 1 defines the sets of the IoT devices, MEC servers, communication channels, and battery levels of the IoT devices. A *Lower Safety Limit* (LSL) is set for the battery capacity, which may not be reached, preventing the device to run out of energy. The algorithm collects the number of CPU nodes available in each IoT device and MEC server, their operating frequencies, and supply voltages.

In step 2, TEMS classifies the tasks between critical and non-critical. Critical tasks are ordered by deadline and scheduled to the CPU with the minimum total elapsed time. Then normal tasks are scheduled to the CPU with lesser total cost.

In step 3, tasks are monitored for their completion status, and when completed, the CPU core resources are released and made available for other allocations in step 2. The battery level check is performed for the IoT devices. Task cancellation may occur if the elapsed time is higher then the deadline or if the IoT device runs out of battery.

Finally, in step 4, the battery level from each IoT device is collected, as well as newly created tasks. Execution continues as long as tasks are being created.

4 Evaluation

This section explains the simulation details and the different experimental scenarios used. The simulated environment was comprised by IoT devices type

Arduino Mega 2560, with five operating frequencies and corresponding supply voltages. 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. The CPU cores have three operating frequencies and corresponding supply voltages. For CC it was chosen 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. The network throughput was configured to achieve up to 1 Gbps speed and latencies to 5ms, for both 5G and fiber optics communications [2]. Moreover, two vehicular applications were defined [4], 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).

a. Number of MEC servers: This experiment used Application 1 in two different scenarios. Both with 500 tasks distributed to 100 IoT devices, but one with only a single MEC server (plot 1) and the second with two MEC servers (plot 2). Figure 2 shows the results for the execution of Application 1.

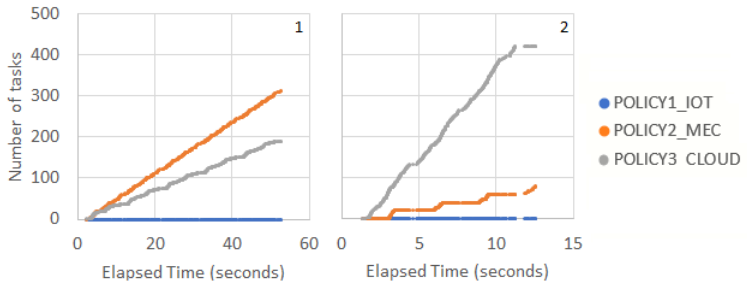

Fig. 2. Task allocation for Application 1.

The energy and time coefficients were set, respectively, as 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 1 to plot 2, the increase in the number of MEC servers made fewer tasks be to allocated in the CC layer, reducing total energy consumed. Compared to another scenario with the same characteristics, but without MEC servers, the reduction in energy consumption for case 1 was 42.51%, while for case 2 it was 44.71%. Thus, the use of MEC servers helps the system to lower the total energy consumed.

b. Impact on 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.

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. Now, energy had a high-value

Table 1. Costs for Application 2, varying the cost coefficients for energy and time.

Case	E_{Coeff}	T_{Coeff}	Cost	E_{Total}	T_{Total}	Policy
$C1$	1/5	4/5	0.01859	0.14550	0.03336	MEC
$C2$	2/5	3/5	0.02597	0.14276	0.03469	MEC
$C3$	3/5	2/5	0.03318	0.14276	0.03469	MEC
$C4$	4/5	1/5	0.03544	0.07040	0.25000	IoT

coefficient, which made the scheduler choose the policy that provided the lowest energy cost, reducing total cost. In case 4 energy consumption reduced up to 51.61% compared to the other cases. To reduce task completion time, coefficients from cases 1, 2 and 3 are better, with a reduction of up to 86.65% compared to case 4.

c. Impact of input data size: As the size of data entry increases from 3.6MB to 3.6GB, the calculated costs progresses 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 scales, allocation policies that require data transmissions become very costly, and allocation on the device itself becomes increasingly advantageous. We observed that with more tasks with less data per task it is possible to reduce energy and time by up to 29% compared to a scenario with fewer tasks and lots of data per task. With this approach less time is spent waiting for data transmission to end, maximizing CPU usage in the MEC servers.

d. IoT device battery consumption: Battery should stay in healthy levels to avoid reaching the LSL threshold, avoiding the device to be unavailable for processing. Also, adequate task processing workloads may help save battery, extending the operation time of the IoT device.

e. Deadline for critical tasks: Very low deadlines made tasks to be canceled because any given policy could execute the task in a feasible time. Therefore, the deadline must be properly configured for at least one of the allocation policies to have enough time to process and conclude the task correctly.

f. Impact on using the DVFS technique: With DVFS activated, the total energy consumption decreased by 13.74%, while the total time increased by 28.32% in comparison with DVFS off. This demonstrates the effectiveness of the proposed model, and the scheduling algorithm in minimizing total energy consumption. Although the whole time may have been longer in the approach with DVFS, it is no problem because the tasks were completed within the time limit imposed by the deadline.

5 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, energy and time reduction are mostly needed. The TEMS scheduler could choose the best allocation options in the system, reducing both

energy consumption and elapsed time. Experiments show that the adequate adjustment of the cost coefficients were essential for the final cost perceived by the scheduling algorithm. Adequate coefficients allowed the energy in the system to be reduced by up to 51.61% or the total times to be reduced by up to 86.65%, ending critical tasks before deadline. The system has become more sustainable and the user experience has not been affected.

The use of MEC servers helped extend the battery life of the IoT devices and made task execution more agile. Also, using the DVFS technique brought interesting results, helping to reduce the overall energy consumption. Major contributions are the TEMS algorithm, the addition of data transmissions to the model, accounting for idle costs, calculating transmission rate interference, use of the DVFS technique, and the interaction with the CC layer to provide resources whenever the local network becomes saturated.

As future works, we can relate the improvement of the system cost model, with the insertion of new variables and new experiments to explore applications in new scenarios such as industry, healthcare, aviation, mining, among others.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. PROPESQ-UFRGS and by projects: "GREEN-CLOUD: Computação em Cloud com Computação Sustentável" (#162551-0000 488-9) and SmartSent (#172551-0001 195-3) from FAPERGS, PNPD Capes, and CNPq Brazil, program PRONEX 122014.

References

1. Aijaz, A.: Towards 5g-enabled tactile internet: Radio resource allocation for haptic communications. In: 2016 IEEE Wireless Communications and Networking Conference Workshops. pp. 145–150 (2016)
2. Gupta, A., Jha, R.K.: A survey of 5g network: Architecture and emerging technologies. IEEE Access pp. 1206–1232 (2015)
3. Haouari, F., Faraj, R.: Fog computing potentials, applications, and challenges. In: 2018 International Conference on Computer and Applications. pp. 399–406 (2018)
4. Jansson, J.: Collision Avoidance Theory with Application to Automotive Collision Mitigation. Ph.D. thesis (2005)
5. Liu, Y., Yang, H., Dick, R.P.: Thermal vs energy optimization for dvfs-enabled processors in embedded systems. In: 8th International Symposium on Quality Electronic Design. pp. 204–209 (2007)
6. Ravindranath, L., Bahl, P.: Glimpse: Continuous, real-time object recognition on mobile devices. In: Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems. p. 155–168 (2015)
7. Sarangi, S.R., Goel, S., Singh, B.: Energy efficient scheduling in iot networks. In: Proceedings of the 33rd Annual ACM Symposium on Applied Computing. p. 733–740 (2018)
8. Wang, C., Dong, C., Qin, J.: Energy-efficient offloading policy for resource allocation in distributed mobile edge computing. In: 2018 IEEE Symposium on Computers and Communications. pp. 366–372 (2018)
9. Yu, H., Wang, Q., Guo, S.: Energy-efficient task offloading and resource scheduling for mobile edge computing. In: 2018 IEEE International Conference on Networking, Architecture and Storage. pp. 1–4 (2018)