A Load Balancing Algorithm For Long-Life Green Edge Systems

Roberto Beraldi, Gabriele Proietti Mattia
Department of Computer, Control and Management Engineering "Antonio Ruberti"
Sapienza University of Rome
Rome, Italy
ORCID: 0000-0002-9731-6321, 0000-0003-4551-7567

***Abstract*—We consider the case of a set of energy harvesting edge nodes, equipped with photovoltaic panels that implement some kind of monitoring service. To ensure that the service operates in an optimal way, nodes have sometimes offload some of their data to other nodes. We show that this kind of task offloading (migration) can improve service performance by avoiding temporary interruptions and prolonging the overall service lifetime. We present a centralized algorithm based on Linear Programming optimization problem solution and a distributed implementation.**

***Index Terms*—Green Edge Computing, Load Balancing, Lifespan, Decentralization**

I. Introduction

Edge computing promises to bring cloud-like services implementation closer to end users, reducing the need to transport data for processing, [1]. Accelerator-enhanced single-board computers (SBCs) can nowadays be used as edge devices running complex computations, e.g. inferencing models [2], and are progressively augmenting the application spectrum of this paradigm. In fact, an interesting application domain of edge computing is in rural areas, e.g. for environment or animal activity monitoring. For example, [3] used Nvidia Jetson Nano for early detection of beep's Varroa destructor mites within beehives. The mites cause varroosis (Varroosis apium), the world's most destructive honey bee disease, which inflicts substantially greater damage and incurs higher economic costs than all other known apicultural diseases, [3].

In such monitored sites, grid supply maybe not be easily accessible, and photovoltaic (PV) solar panels can be used instead. A vital role to enable the *smooth* running of edge computing systems with solar energy supply is to guarantee as much as possible enough available energies in all sites, [4]. Roughly speaking, if E_G is the total energy produced in one day by the PV panels, and E_C the energy consumed by the edge system during the day if $E_G \geq E_C$ then the system should never be interrupted, while if $E_G < E_C$ the lifespan of the system should be as long as possible and with minimum interruptions.

Intermittent operations can occur when the energy required now and here at a site is not available, see Figure 1. Short-term prediction can help in deciding whenever to postpone heavy computations (load time shift), however, solar power generation is heavily reliant on multiple meteorological factors, e.g. solar irradiance, cloud opacity, and air temperature and it is not always accurate moreover, postponing processing changes the characteristic of service. To achieve better service continuity it is worth viewing the local energy at all sites as a system-wide resource that is shared, which is achieved by offloading energy-consuming task processing from one node with low battery to a richer one [5].

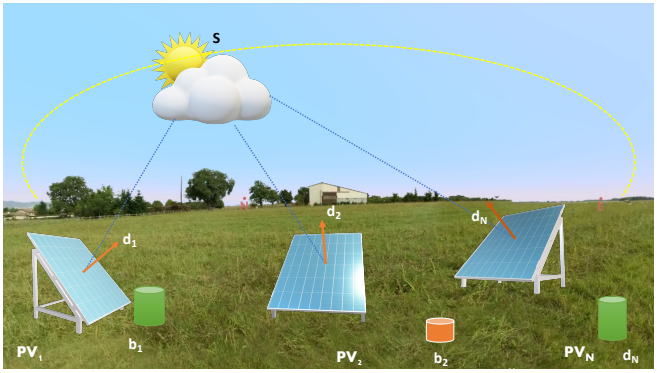

Fig. 1. PV panels with different orientations (surface normal's vector d_i) can generate different amounts of energy during the same day. In addition, a site may temporarily lack energy while others do not.

We here focus on a use-case scenario where a set of energy harvesting edge nodes, with their own camera implementing a common service of image detection, e.g. beehives monitoring, cooperate by delegating the inspection of an image acquired by its own camera, to another node in the set (task offloading) in case of the lack of enough local energy. The key concept in our proposal follows from the observation that the amount of solar radiation converted by two close PV panels into electrical energy at the same time may differ, e.g. due to their different orientations, surface sizes, or surface clean conditions but it is highly correlated. In addition, the power required may also differ as it depends on the current computation load. Overall,

This work was partially supported by project no. 202277 WMAE CUP B53D23012820006, "EdgeVision against Varroa (EV2): Edge computing in defence of bees" funded by the Italian's MUR PRIN2022 (ERC PE6) research program.

this may cause one node to temporarily run out of energy while the others have enough energy to process tasks. Shortly, this condition can be swapped. By offloading tasks to each other at proper times, the net result of the cooperation can be that neither of the two nodes is interrupted nor task computation postponed.

In this paper, we describe an energy-driven load-balancing algorithm that exploits the differences in energy accumulated and available to the nodes that reduce the event of service interruption at some sites.

The contribution of this paper is (i) a characterisation an off-grid green edge computing model and (ii) centralized numerical algorithm that implements energy balancing; (iii) a distributed algorithm implementation running on a simulator.

The paper is organized as follows: Section II reports the related works, Section III system model and problem formulation, Section IV the cenrtalized algorithm, Section V distributed implementation. Conclusions are given in Section VI.

II. Related Work

Several papers address the problem of energy efficiency in Edge and Fog Computing, via load balancing or proper resource allocation. [6] proposes an energy-aware load balancing and scheduling (ELBS) method based on fog computing. The work reports an energy consumption model of the workload on the fog node. [7] designs a novel Energy-aware Data Offloading (EaDO) technique to minimize the energy consumption and latency in the industrial environment. [8] studied a sustainable infrastructure in Fog-Cloud environment for processing delay-intensive and resource-intensive applications with an optimal task offloading strategy. The proposed offloading strategy optimizes two Quality-of-Service (QoS) parameters such as energy consumption and computational time. The model in these papers includes a cloud layer where the computation is eventually performed. [9] presents an energy-efficient Fog architecture considering the integration of renewable energy. Three resource allocation algorithms and three consolidation policies were studied. [10] instead proposes a Pareto-efficient algorithm that aims to simultaneously optimize both latency and energy efficiency in data stream processing for edge computing. Then, Lyu et al. in [11] propose an architecture that integrates the Cloud, the MEC (Mobile Edge Computing) layer and the IoT for implementing a selective offloading algorithm that is designed to minimize the energy consumption of devices. However, the approach does not consider the energy contribution that is harvested from solar panels. In [12] instead, the authors focus on the Internet-of-Vehicles (IoV) and propose an efficient scheduling framework to minimize the energy consumption of Green Roadside Units (RSUs) under latency constraints. Differently from our work, we hypothesize that each computing node has attached an accumulator, moreover, the approach that we follow is even decentralized. Wu et al. in [13] propose an algorithm called "GLOBE" which performs a joint geographical load balancing in MEC environments where energy-harvesting nodes are considered. The authors show, relying on Lyapunov optimization, that the approach achieves a close-to-optimal result compared to an offline algorithm that knows the full information about the system. A similar approach is studied in [14] which proposes a hierarchical task offloading that optimizes latency, energy consumption, and cloud fees. However, in these works, the decentralized approach is not considered. Similar approaches are then used in [15].

Other approaches focused on energy-aware task scheduling can be seen in [16], [17] and [18].

Symbol	Meaning
	Model
V	Set of n nodes
$\mathbf{X}^{n\times n}$	Flow migration matrix where each x_{ij} describes the rate of tasks migrated from node i to j
e_{w_i}	Energy required by node i to process an image
e_{s_i}	Energy required by node i to send an image
e_{r_i}	Energy required by node i to receive an image
$p_i(\mathbf{X})$	Power required by node i given the migration matrix $\mathbf{X}$
p_{m_i}	Power required by node i to serve the whole local traffic λ_i
p_{min_i}	Minimum power required by node i
p_{max_i}	Maximum power required by node i
r_i	Power required by node i to receive and serve remote traffic
s_i	Power saved by node i when sending local traffic
b_i	Battery charge available at node i
$\mathbf{b}$	Battery vector $(b_1, b_2, \dots b_n)$
$ls_i(b_i, \mathbf{X})$	Lifespan of node i, given matrix $\mathbf{X}$ and battery b_i
$ls_T(\mathbf{b}, \mathbf{X})$	Lifespan of the system given battery vector $\mathbf{b}$ and matrix $\mathbf{X}$
	Adaptive Heuristic
τ	Round period
ϵ	Tolerance threshold for balancing condition
α	Step-size for increments and decrements of migration ratios

TABLE I
LIST OF MAIN SYMBOLS USED

III. System Model

We consider a set V of N nodes. Each node i receives a stream of tasks (images) at rate λ_i frames per second (fps) directly from a camera connected to the device (called hereafter local traffic or *user traffic*) and offload (delegates without processing) a flow of $0 \leq x_{ij} \leq \lambda_i, x_{ii} = 0$ images to node i, which is the entry i, j of the migration matrix $\mathbf{X}$. By definition $x_{ii} = 0$

Energy requirements: We assume that the time required to process a task is τ_{w_i}, while the time for sending and receiving a task $\tau_{s_i} = \tau_{r_i} < \tau_{w_i}$. A node absorbs p_{I_i} watts when it is idle and at most p_{M_i} watts when working. When CPU is fully utilized it absorbs an additional power p_w and when sending/receiving a power $p_s < p_w$. Hence:

$$e_{w_i} = p_w \tau_{w_i}$$

$$e_{s_i} = p_{s_i} \tau_s$$

where clearly $e_{w_i} > \{e_{s_i}, e_{r_i}\}$. The rate of task execution of a node i is:

$$\lambda_i' = \lambda_i - \sum_j x_{ij} + \sum_j x_{ji}$$

Clearly, as $\sum_i \lambda_i' = \sum_i \lambda_i$ all local tasks are eventually processed by some node. The overall power requirement of node i is:

$$p_i(\mathbf{X}) = p_{I_i} + e_{w_i}\lambda_i' + e_r \sum_j x_{ji} + e_s \sum_j x_{ij}$$

which for convenience is written as

$$p_i(\mathbf{X}) = p_{m_i} + r_i \sum_j x_{ij} + s_i \sum_j x_{ji}$$

where

$$p_{m_i} = p_{I_i} + \lambda_i e_{w_i} \qquad r_i = e_{w_i} + e_{r_i} \qquad s_i = -e_{w_i} + e_{s_i}$$

The term $r_i > 0$ takes into account the additional energy needed to receive and process remote tasks, while $s_i < 0$ is the energy saved because tasks are sent to other nodes rather than being processed locally.

Assume now that the battery charge of node i is b_i joules, the lifespan of a node is defined as:

$$ls_i(b_i, \mathbf{X}) = \frac{b_i}{p_i(\mathbf{X})} \tag{1}$$

The lifespan has the property that if the battery is increased by a factor α then the lifespan increases by the same factor. In addition, the admissible lifespan interval of node i is the interval

$$I_i = [t_m, t_M] \tag{2}$$

where $t_M = \frac{b_i}{p_{min_i}}$, is the longest lifespan of the node assuming the node is used only for offloadingand $t_m = \frac{b_i}{p_{max_i}}$, is shortest duration computed assuming the maximum power absorbed, see Figure 2.

The 'battery-is-empty' event frequency is:

$$f_i(b, \mathbf{X}) = ls_i^{-1}(b_i, \mathbf{X}) = p'_{m_i} + r'_i \sum_j x_{ji} + s'_i \sum_j x_{ij}$$

where the normalized coefficients are:

$$p'_{m_i} = \frac{p_{m_i}}{b_i} \quad r'_i = \frac{r_i}{b_i} \quad s'_i = \frac{s_i}{b_i}$$

In general, the set of edge nodes is not homogeneous in terms power requirements.

Definition III.1 (Power compatibility)**.** *A set V of nodes is power compatible if $max_{i \in V}\ \{p_{min_i}\} < min_{i \in V}\ \{p_{max_i}\}$.*

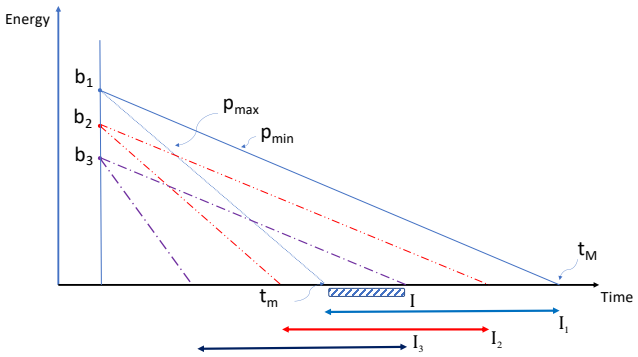

Fig. 2. An example of lifespan intervals for three nodes, given their initial battery and maximum and minimum power absorbed (details of node 1 are given). Any value in the interval I can be matched by a proper migration matrix $\mathbf{X}$.

A. Problem formulation

To shape the approach that will allow us to find an optimal migration matrix $\mathbf{X}^*$, we first need to define some term and then tie the notion of optimality to the final performance goal. A matrix $\mathbf{X}$ is compatible with the user traffic vector $(\lambda_1, \lambda_2, \ldots, \lambda_N)$ if $\sum_j x_{ij} = \lambda_i, x_{ij} \geq 0$. Two matrices $\mathbf{X}$ and $\mathbf{X}$' are traffic-equivalent if any node processes the same traffic, i.e. $x_{ii} = x'_{ii}$.

Intuitively, a matrix $\mathbf{X}$ is more efficient than another matrix $\mathbf{X}$', if they are traffic-equivalent and if the total amount of transmissions using $\mathbf{X}$ is lower than the number of transmissions of $\mathbf{X}$'. For this reason we use the notation $|\mathbf{X}|$ to mean the sum of all the not diagonal elements of the matrix $|\mathbf{X}| = \sum_{i \neq j} x_{ij}$. We denote with $\mathcal{S}$ the set of migration matrices.

As the service deteriorates even when one single node goes down, the lifespan of the system associated with a matrix $\mathbf{X}$ is defined as:

$$ls_T(\mathbf{X}) = min\{ls_i(\mathbf{X})\}$$

The sustainable rates, x_{ij} are defined as those that do not make any node in the system to require more than p_{M_i} watts. In fact, any violation of the requirement will move a node to a congested state, e.g. the node enqueues and eventually drops tasks. We denote with $\mathcal{S}_A \subseteq \mathcal{S}$ the set of matrices that do not violate this requirement.

Definition III.2 (Optimal Solution)**.** *Given a set V of N nodes, a flow migration matrix $\mathbf{X}^*$ is optimal if $ls_T(\mathbf{X}^*) \geq ls_T(\mathbf{X})$ for all $\mathbf{X} \in \mathcal{S}_A$*

Our problem is to find an optimal assignment of all flow variables which minimizes the maximum lifespan of the system. To make the problem solution simple we use the property that under no power limitations, the function ls_T is maximum when all nodes have the same lifespan.

1) Constrains: We now formulate the constraint in matrix form.

Power absorption: A node cannot absorb more than a maximum nominal value, p_{M_i}

$$\forall i \in V \quad p_{m_i} + r_i \sum_j x_{ji} + s_i \sum_j x_{ij} \leq p_{M_i}$$

In matrix notation this constrain can be expressed:

$$C\mathbf{x} \leq \mathbf{p}_w$$

where is the cost offloading matrix:

$$[C]_{ij} = \begin{cases} s_i, & \text{if } j/N = i \wedge j/N \neq j \mod N \\ r_i, & \text{if } j \mod N = i, j/N \wedge j \mod N \\ 0 & \text{otherwise} \end{cases}$$

$\mathbf{x}$ is a column vector $\mathbf{x}^T = (x_{11}, x_{12}, \ldots, x_{1N}, x_{21} \ldots x_{NN})$, C is a $N \times N^2$ and $\mathbf{p}_w = \mathbf{p}_M - \mathbf{p}_m$, i.e. the difference among the maximum and minimum power vectors:

Uniformity. Nodes must have the same frequency:

$$i \in V \quad p'_{m_i} + r'_i \sum_j x_{ji} + s'_i \sum_j x_{ij} = p'_{m_{i+1}} + r'_{i+1} \sum_j x_{ji+1} + s'_{i+1} \sum_j x_{i+1j} \quad (3)$$

which can be expressed as:

$$D(\mathbf{p}'_m + C'\mathbf{x}) = \mathbf{0} \quad (4)$$

where D is a $N \times N$ difference matrix:

$$[D]_{ij} = \begin{cases} 1, & \text{if } i = j \\ -1, & \text{if } j = (i+1) \mod N \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

and C' is the $N \times N^2$ normalized cost matrix:

$$[C']_{ij} = \begin{cases} s'_i, & \text{if } j/N = i \wedge j/N \neq j \mod N \\ r'_i, & \text{if } j \mod N = i \wedge j/N \neq j \mod N \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

Single Hop Forwarding: a node can only offload a fraction of its local traffic λ_i to other nodes, i.e. nodes cannot act as intermediaries. This constraint is introduced for the performance reason of reducing transmission delay.

$$\forall i \in V \quad \sum_j x_{ij} \leq \lambda_i \;\; x_{ii} = 0 \;\; x_{ij} \geq 0 \quad (7)$$

By defining the $N \times N^2$ forwarding matrix as

$$[F]_{ij} = \begin{cases} 1, & j/N = i, i \leq N \vee i = N+1 \\ & j/N = (j \mod N) \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

the forward constrain becomes:

$$F\mathbf{x} \leq \mathbf{\Lambda}, \quad \mathbf{x} \geq \mathbf{0}, x_{ii} = 0 \quad (9)$$

where

$$\Lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_N \end{bmatrix} \quad (10)$$

Topology: In addition to single hop forwarding, task offloading may occur only if there is a communication link between two nodes. Let E be the set of communication network channels. Then $x_{ij} \geq 0$ only if $(i,j) \in E$ otherwise it must be zero.

B. Problem formulation

We now can formulate the following *Migration Problem (MP)*,

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{1}^T\mathbf{x} \\ \text{subject to:} \quad & C\mathbf{x} \leq \mathbf{p}_w \\ & F\mathbf{x} \leq \mathbf{\Lambda} \\ & D(C'\mathbf{x} + \mathbf{p}'_m) = 0 \\ & \mathbf{x} \geq \mathbf{0} \\ & x_{ij} = 0 \forall (i,j) \notin E \end{aligned}$$

For example, for $N = 3$ the main matrixes have the following elements:

$$C = \begin{bmatrix} 0 & s_1 & s_1 & r_1 & 0 & 0 & r_1 & 0 & 0 \\ 0 & r_2 & 0 & s_2 & 0 & s_2 & 0 & r_2 & 0 \\ 0 & 0 & r_3 & 0 & 0 & r_3 & s_3 & s_3 & 0 \end{bmatrix}$$

$$F = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{bmatrix}$$

The problem is feasible if the solution set is not empty, i.e. there is a matrix that satisfies all the constrains.

Theorem III.1. *Let $\mathbf{b} = (b_1, b_2, \ldots b_N)$ be a battery vector and $I = I_1 \cap I_2 \ldots \cap I_N$ the intersection of all the admissible associated lifespan intervals defined by Equation 2. If $I \neq \emptyset$ then the problem MP is feasible.*

Proof. Let $t^* \in I$. The coefficients of the migration matrix are constrained only by $2N$ equations, N due to the power that has to be absorbed where the i-th equation is:

$$b_i/t^* = p_{m_i} + r_i \sum_j x_{ij} + s_i \sum_j x_{ji}$$

and N due to the loads:

$$\sum_{j=1}^{N} x_{ij} = \lambda_i$$

which is an underdetermined linear system of $2N$ equations and N^2 unknowns. □

Corollary III.1. *If a system is power compatible then MP is feasible for any constant battery vector,* $b_i = K$.

Proof. The interval I is not empty. □

IV. A CENTRALIZED CONTROL ALGORITHM FOR CHARGE EQUALIZATION.

Our first implementation relies on a centralized algorithm based on the numerical solution of the MP problem, called centralized planner algorithm, which is carried out using the numerical solver HiGHS of the scipy Python extension. The following pseudo-code formalizes the idea. It is assumed that the planner is aware of the state of the system of edge nodes, e.g. battery charge and incoming traffic. The planner computes a new migration matrix based on a currently available battery vector $\mathbf{b}$ provided that it is higher than a threshold activation value, and applies the matrix for a period of time τ called a *migration round*. If batteries are not recharged, then at the end of the round $\mathbf{b} = \mathbf{0}$. If batteries are recharged, a new matrix is computed and a new round starts. Rounds may have different lengths. To solve the MP problem the current battery vector $\mathbf{b}$ is first used; if the problem is unfeasible the migration matrix related to a constant battery vector is computed and applied (which is always feasible).

Algorithm 1 Centralized Planner Algorithm

```
while b > b_T do
    if MP(b) is feasible then
        (X, τ) ← Solve MP(b)
    else
        Δb(t) ← min{b(t)}1
        (X, τ) ← Solve MP(Δb)
    end if
    migrate using X for time interval τ
end while
```

A. Solar power production data set

The data used in our experiments were collected using a plant of 20 solar panels at the latitude of 41°.7', with an elevation of 45° degrees and orientation of North-West 330°; each panel is 2×1 m, and a declared efficiency of 20.38% in Standard Test Condition. Data were sampled every 5 min. In the experiments, at sec, granularity data is obtained by linear interpolation among two adjacent sample points, and the power is scaled to simulate a panel with a smaller surface. We selected some representative behaviors labeled as day-1,day-2, and day-3. The shape is scaled of 0.33%, e.g. the 6kW peak production is mapped to 20W, e.g. produced by a 45×35 cm panel.

The following table summarizes the parameters used in our experiments.

We now report some result considering three representative days.

B. Day 1: Partially cloudy day winter time

In the first case, we consider the profile reported in Figure 3. The power was produced on March 19, 2023, and shows a characteristic pattern where the power falls suddenly due to the presence of clouds in the sky. This scenario shows how cooperation can avoid interrupting the service. Two nodes receive traffic at rates $\lambda_1 = 7$ and $\lambda_2 = 8$ and we assume that due to different solar panel orientations and the panel's surface condition, the efficiency of the panels is $\xi_1 = 0.95$ and $\xi_1 = 0.75$. Figure 4 shows the available energy in the accumulators when nodes do not cooperate. Node 2 has a service interruption after a couple of hours, and stops working before the other.

Parameter	Sym.	Value
Idle Power	P_I	2 W
Maximum Power	P_M	5 W
CPU power consumption	p_w	2.7 W
RTX power consumption	p_s	0.3 W
Image processing time	τ_w	8.3 ms
Receiving time	τ_r	10 ms
Sending time	τ_s	10 ms

TABLE II
PROFILE OF EDGE NODES

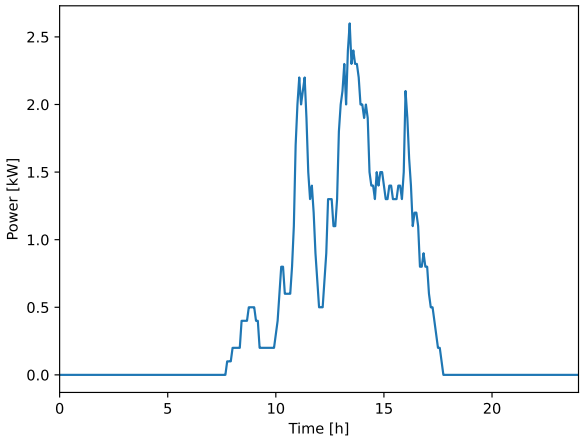

Fig. 3. The daily power production on March 19, 2023 (day-1).

Figure 5 shows the accumulator profile when migration is used. A migration run starts when the energy accumulated in both batteries is at least 3000 Joules (called the cooperation activation threshold) and ends after the system lifespan is computed solving the LP problem with the current battery profile. Let t be the time when this event occurs. The LP problem is instantiated with the battery vector and the migration matrix $\mathbf{X}$ is applied until $t + ls$. At this time, both accumulators should be empty, but since they are recharged this is no true. Figure 5 shows 8 migration rounds.

C. Day 2: Partially cloudy day spring time

The second trace we considered is reported in Figure 6, which also refers to a partially cloudy day, but in June. The daylight is longer and the power is higher. We study now N=2 nodes and the consequence of accumulators having a finite

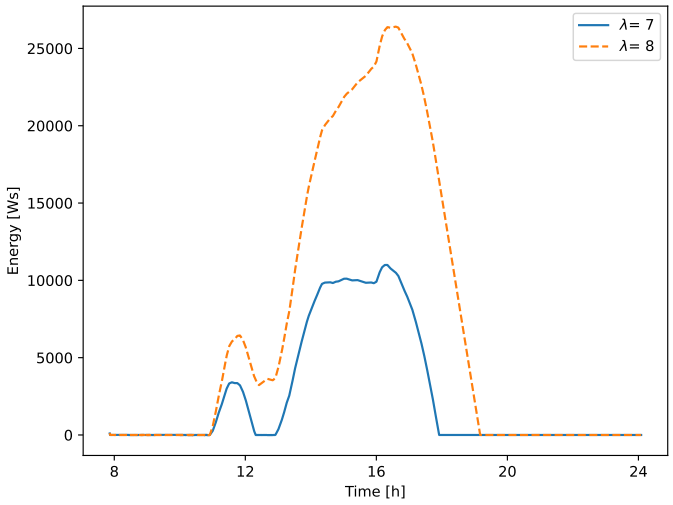

Fig. 4. Battery energy profile without cooperation.

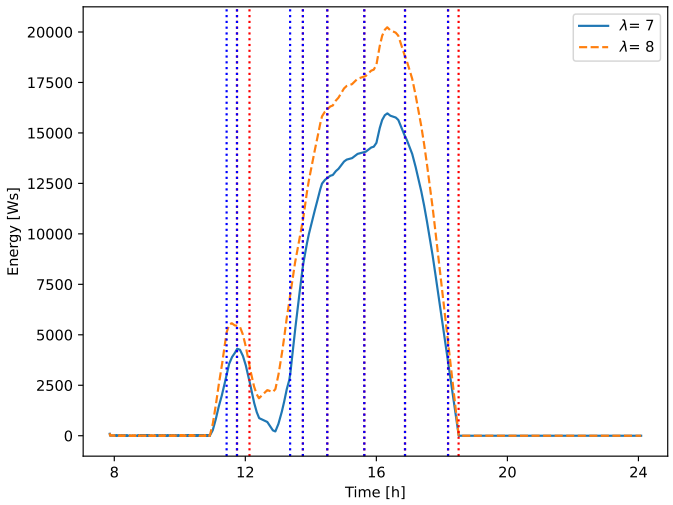

Fig. 5. Battery energy profile with cooperation. Vertical lines indicate the boundary of a new migration round, activation threshold is 3000 Joules.

capacity of $120kJ$ by simulating two consecutive days with the same profile. Figure 7 shows the battery charge without migration. Since the energy produced is higher, no service interruption is registered. Also, nodes have a longer lifespan. However, the first node stops at 4 am on the second day, almost three hours before the first one. It is worth noting that the node 2 accumulator is fully charged so that the energy produced is lost. We call this event green energy loss. In this specific case, energy loss amount to 7045 Joules, which is a source of inefficiency.

Figure 7 shows the profiles when the two nodes cooperate. In this case, the lifespan of node 1 is prolonged by approx one hour and a half at the expense of node 2. Both nodes stop working at the same time.

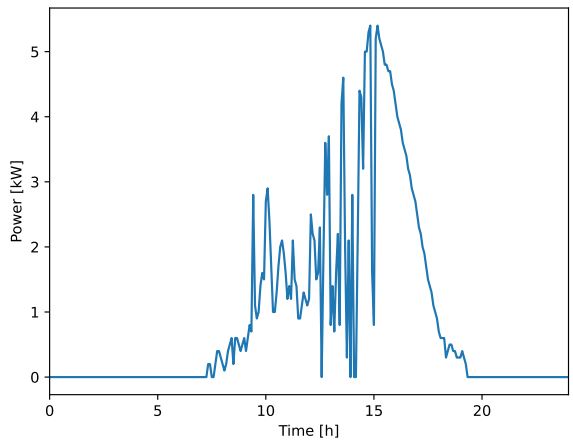

Fig. 6. The daily power production on June 13 2023, (day-2)

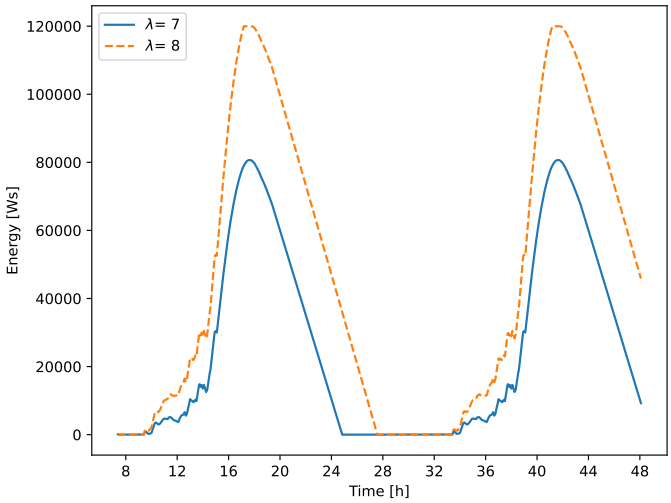

Fig. 7. Accumulated charge for day-2 profile, no cooperation.

D. Day 3: Sunny day spring time

We now consider the profile of Figure 9, related to a sunny day, N=4 nodes with accumulator capacity of $160kJ$ ($\approx$ $44.5Wh$). The panel efficiency was set to efficiency factor $\xi = 0.93, 0.87, 0.85$, and $\xi = 0.8$ and the load to $\lambda_1 = 7, \lambda_2 = 8, \lambda_3 = 9, \lambda_4 = 10$.

Figure 10 shows the available energy in the accumulators over two days whitout cooperation. Even if the total amount of energy produced in a day is enough to supply the nodes, it is not stored and consumed in an optimal way, making the service running on node 3 and node 4 stop.

Figure 11 shows the same profile when nodes cooperate, i.e. node node migrates towards the other. The are two cooperation rounds. The first round ($\approx$ 11:00 am - 1 pm) makes the battery become full rougly at the same time (at 1 pm). The second one starts after midnight and ends in the morning. Thanks to

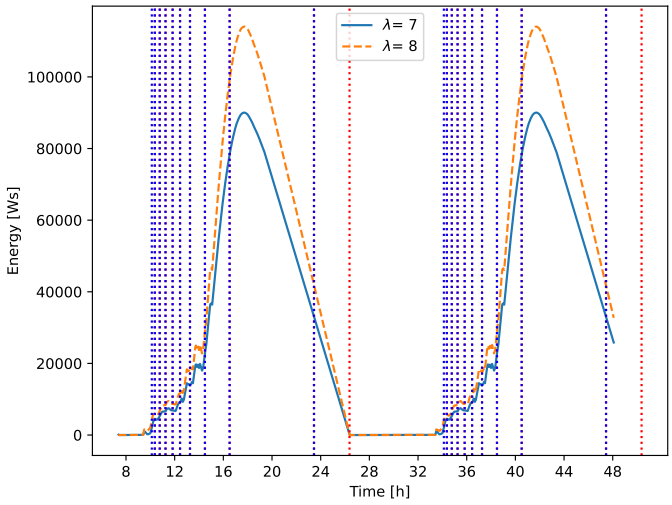

Fig. 8. Accumulated charge for two days using day-2 profile, with cooperation.

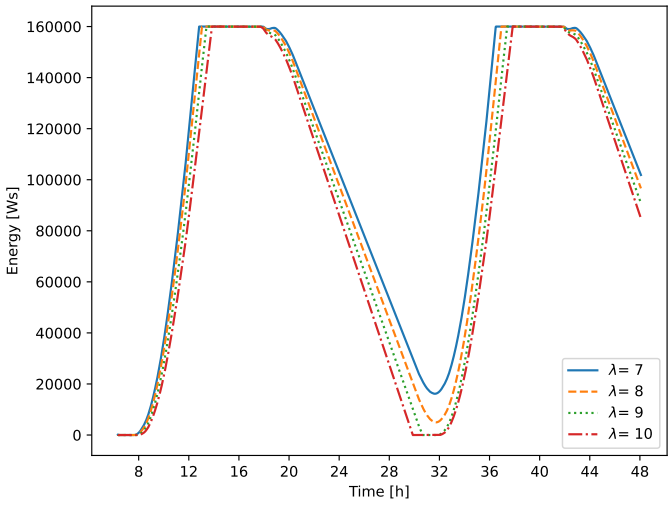

Fig. 10. Accumulated energy by 4 nearby nodes with different power produced and required, no cooperation.

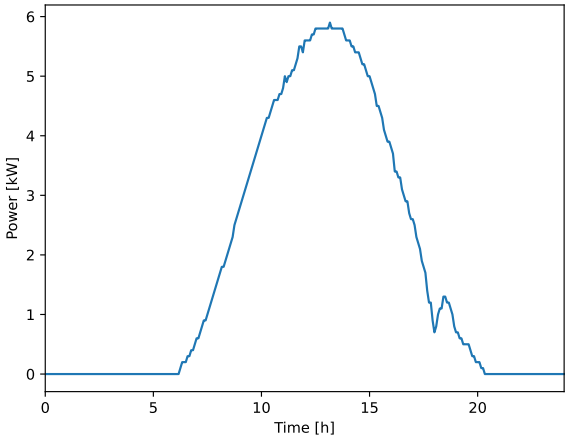

Fig. 9. The daily power production on June 16, 2023 (day-3)

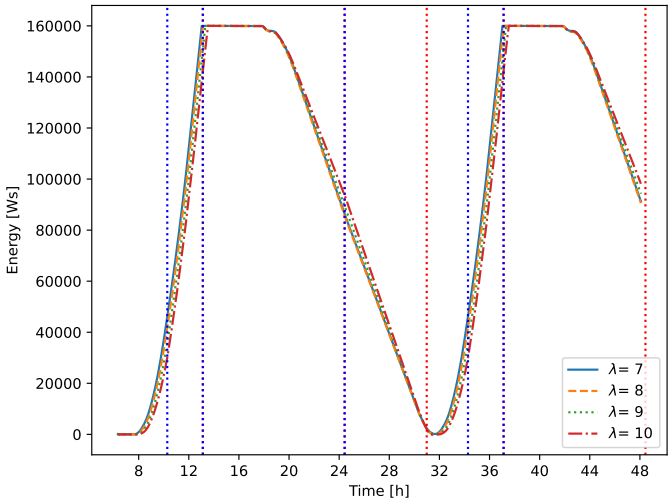

Fig. 11. Accumulated energy by 4 nearby nodes with different power produced and required, with cooperation.

cooperation now no nodes stop working.

V. Adaptive Heuristic

The main limitation of the optimization method, even if applied continuously as described in Algorithm 1, is that it requires the real-time parameters of each node involved in the system. In a hypothetical real implementation, these nodes must have a reference entity, which can be another node or even the cloud, which gathers all the data from the nodes, runs the algorithm, it finds the solution $\mathbf{X}$, and then it returns the new migration ratios configuration to the nodes. This approach does not scale with the number of nodes, which in Edge environments can also be significant. To solve this problem, we propose, in this Section, a fully decentralized heuristic that enables each node to find a solution to the problem only knowing the real-time parameters of its neighbors. The idea behind the algorithm is to make each node progressively update its migration ratios toward the neighbors by following a round pattern of period τ. The update is done by a small step size α. In this way, the solution is continuously adapted and can even react to unpredictable changes in system parameters (i.e. a node can go down or saturate). We now need to define the rule for which a migration ratio has to be increased or decreased of the step size α. The key idea is that if we have a set of nodes with different lifespans, to maximize the lifespan of the system, which is the objective of the optimization problem (Definition III.2), the nodes which have a low lifespan have to migrate part of their tasks to the nodes which have a higher lifespan. Intuitively, this will make the nodes with

a lower lifespan increase it, while the others will decrease it until they match. More precisely, given a round r, for any given node i if we compute the average lifespan among a node and its neighbors, called $l_i^{r,a}$, and lifespans of all nodes are all different, we necessarily have some node neighbor node $j \in V$ for which its lifespan at round r is $l_i^r > l_i^{r,a}$ and some node $k \neq j$ for which its lifespan is $l_i^r < l_i^{r,a}$. Since our algorithm has to be resilient to variation in traffic, we impose a tolerance zone, and we give the following

Definition V.1 (Lifespan-Balancing)**.** *Given a node $i \in V$, the average lifespan among it and its neighbors as $l_i^{r,a}$ and ϵ the tolerance coefficient, we define is as lifespan-balanced if*

$$l_i^{r,a} \cdot (1 - \epsilon) \leq l_i^r \leq l_i^{r,a} \cdot (1 + \epsilon) \tag{11}$$

The purpose of the algorithm is to balance the lifespan of the nodes by progressively increasing or decreasing the migration ratios, keeping in mind that nodes that are below the average lifespan always need to migrate tasks only and only to nodes that are above the average lifespan. Indeed, it is not reasonable to forward traffic to nodes below the average lifespan since this would progressively make low-lifespan nodes not increase their lifespan, thus not approaching the solution to the problem.

Algorithm 2 shows our proposed method. We suppose that every τ seconds, each node in the system calls the function UPDATERATIOSROUND, which updates the migration ratios of the node. Migration ratios are always used for forwarding decisions since they can be interpreted as migration probabilities. Suppose that node i reaches the round period. Migration ratios are updated by following these steps:

1) first of all, we compute the average lifespan between the current node i and its neighbors. In real environments, the node i asks the neighbors for their lifespan. The lifespan estimation is computed using Equation 1 with the battery residual energy value at the beginning of the last round and the average energy consumption of the node during the last round. We also compute the bounds of the tolerance zone.
2) the algorithm then checks if the migration ratios must be decreased because the node i could be migrating too much traffic, and it is now out of balance since its lifespan fell above the higher bound of the tolerance zone. To return in balance, the node must decrease its lifespan, and this can be node by decreasing by a step size a the positive migration ratios toward its neighbors;
3) at this point, we check if the current node's lifespan is above the tolerance zone's lower bound. If this is true, then we consider the node as lifespan-balanced, and no further action is taken;
4) if the current node's lifespan is instead below the tolerance zone's lower bound, then it means then decrease the migration ratios towards the nodes whose lifespan fell as well below the tolerance zone's lower bound if the current node i is migrating tasks to them. This is because, as already described, we need to avoid forwarding tasks to nodes that are in the same condition;
5) now we can proceed to increase the lifespan of α towards the neighbors whose lifespan is above the tolerance zone's higher bound only if that neighbor is rejecting fewer tasks than the current node i or if, even better, it is not rejecting tasks. We include this condition to avoid the trivial behavior of forwarding tasks only to decrease the lifespan without concerns about whether they are executed. When increasing migration ratios, we also check if their sum is not exceeding one.

The algorithm returns the updated migration ratios which are then set and used during the next round.

Algorithm 2 Lifespan-levelling Decentralized Adaptive Heuristic

```
Ensure:
  neighbors ← list of current node's neighbours
  ratios ← array of migration ratios one for each neighbor
  τ ← round period
Require:
  self ← the current node which is running the algorithm
  lifespans ← array of neighbors' lifespans
  ε ← the tolerance coefficient
  α ← the step size

  function UPDATERATIOSROUND(ratios, neighsIPs, states, ε, α)
    [1. Compute the average lifespan value and the tolerance zone limits]
    selfLifespan ← GetLifespanLastRound(self)
    avgLifespan ← (sum(lifespans) + selfLifespan) / (len(lifespans) +1)
    avgHigh ← avgLifespan · (1 + ε)
    avgLow ← avgLifespan · (1 − ε)

    [2. Check if the current lifespan is above the average and ratios must be reduced]
    if selfLifespan > avgHigh then
      for n ∈ neighbors do
        [2a. Reduce of step α every positive ratio]
        if ratio[n] > 0 and ratios[n] - α ≥ 0 then
          ratios[n] ← ratios[n] - α
        end if
      end for
    end if
    [3. Check current state is above or equal to the low limit of the tolerance zone, in that case, the node is balanced]
    if selfLifespan ≥ avgLow then
      return ratios
    else
      for n ∈ neighbors do
        [4. Reduce the ratio to nodes that are below the tolerance zone]
        if lifespan[n] ≤ avgLow and ratios[n] - α ≥ 0 then
          ratios[n] ← ratios[n] - α
        end if
        [5. Increase the ratio to nodes that are above the tolerance zone only if the lifespan is above the tolerance zone and the drop rate is less than the current]
        selfDropRate ← GetDropRateLastRound(self)
        nDropRate ← GetDropRateLastRound(n)
        if states[n] > avgHigh and sum(ratios + α) ≤ 1 and (nDropRate = 0 or nDropRate < selfDropRate) then
          ratios[n] ← ratios[n] + α
        end if
      end for
    end if
    return ratios
  end function
```

A. Simulations

The performances of Algorithm 2 have been measured in simulation. In particular, we set up a Python discrete-event simulator by using the Simpy[1] library. The simulator has been published as open source[2].

In the simulator, we attached a service (Q_s) and a transmission queue (Q_t) to each node. Tasks arrivals are distributed according to a Poissonian distribution with average λ_i. This means that the inter-arrival times are drawn from an exponential distribution with average $\frac{1}{\lambda_i}$. Service times instead are drawn from a Gaussian distribution with average $\frac{1}{\mu_i}$ with $\sigma = 1.3 \times 10^{-3}$.

Figure 12 show the result of an experiment with four nodes and the same parameters of the optimization algorithm with results presented in Figure 10 without the cooperation. The chart is used as a baseline, and we can observe how the residual battery is misaligned as well as the power consumption of the nodes, which is responsible for the different discharge slopes in the battery percentage chart. Moreover, the non-cooperation experiment also shows how nodes have different drop rates over time, and this depends on the different arrival rates. In particular, Node 3, the node with the highest load λ, reaches the 5% of dropped jobs when nodes are all up and running[3].

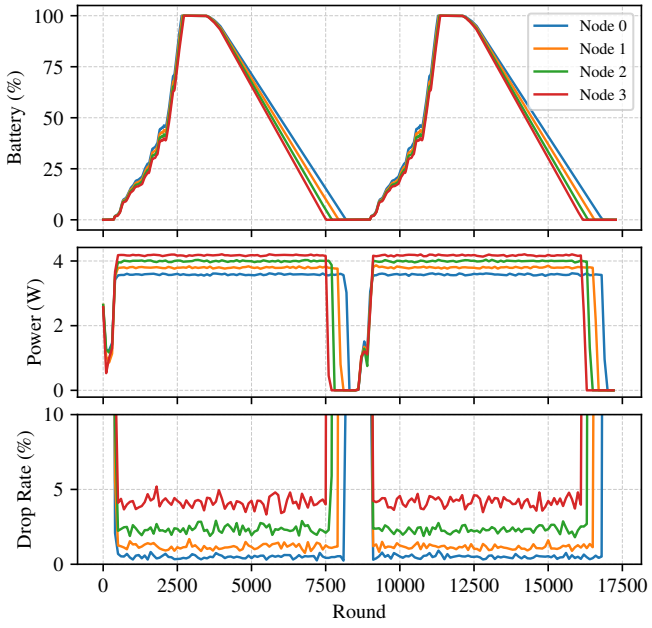

Fig. 12. Traces of the residual battery capacity, the instant power consumption, and the drop rate of the experiment with 4 nodes with $\lambda = 7, 8, 9, 10$, using the day-2 solar trace (Figure 6) with efficiencies $\xi = .93, .87, .85, .80$ respectively and no cooperation. Task arrivals follow a Poisson distribution.

Within the same context, we used Algorithm 1 and Figure 13 shows the result of the experiment. What we can observe is that the battery traces and the power consumption are aligned, and this is achieved thanks to the design of the dynamic of the migration ratios. As a side effect, this also increases the minimum lifespan of the node, which is equal to the maximum since they are aligned. We can notice how Node 3, in both the two cycles of the solar traces, goes down to the lack of energy later with respect to the case in which cooperation is not employed. Moreover, at the same time, even the drop rate is leveled and reduced by the algorithm. In particular, it reaches the 1% for all the nodes in the system.

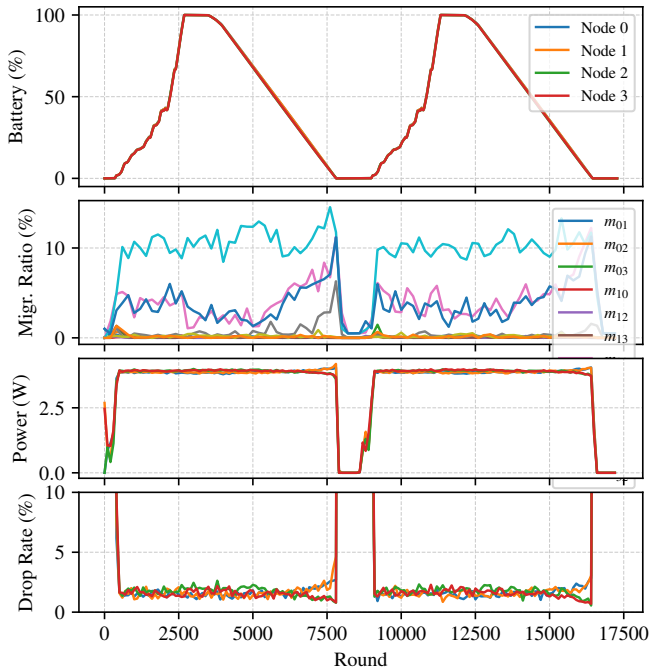

Fig. 13. Traces of the residual battery capacity, migration ratios, the instant power consumption, and the drop rate of the experiment with 4 nodes with $\lambda = 7, 8, 9, 10$, using the day-2 solar trace (Figure 6) with efficiencies $\xi = .93, .87, .85, .80$ respectively and updating the migration ratios by using the proposed Algorithm 2. Task arrivals follow a Poisson distribution.

VI. Conclusions and Future Work

We have studied the case of a set of energy harvesting edge nodes, equipped with photovoltaic panels that implement some kind of monitoring service. To ensure that the service operates in the optimal way, nodes offload their data to other energy richear nodes. We show that this kind of task offloading (migration) can improve service performance by avoiding temporary interruption and prolonging the overall service lifetime. We presented a centralized algorithm based on Linear Programminng optimization problem solution and a distributed implementation.

VII. Appendix

We provide here a simple quantification of the impact of orientation on the amount of energy generated by a panel. We consider two panels with the same azimuth angle π and

[1]https://pypi.org/project/simpy/

[2]https://gitlab.com/gabrielepmattia/simulator-2023-cloudcom

[3]When nodes go down instead, the tasks are marked as rejected, for this reason, the drop rate goes to 100%.

different elevation α. Let $\mathbf{S}$(t) be the position of the sun in the sky at time t, and $\mathbf{P}_i$ the direction of a tilted surface representing solar panel i. The total power generated by the surface has several components, among which the direct irradiance (W/m^2) $ir_d(t)$ - usually the most significant one, is considered here:

$$ir_d(t) = \mathbf{S}(t) \cdot \mathbf{P}$$

We calculated the total energy due to this component numerically, using the *pvlib* library, which provides the altitude, azimuth coordinates of the sun given the latitude and longitude of a point in the Earth:

$$e = \int_{t_r}^{t_s} ir_d(t)dt$$

where t_s and t_s are the sunrise and sunset times. For example, by changing the elevation from $\alpha = \pi/6$ to $\alpha = \pi/7$, considering the path of the sun at latitude 42 degrees July 27, e changes from $\approx 5821J/m^2$ to $\approx 5053J/m^2$.

REFERENCES

[1] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. S. Goren, and C. Mahmoudi, "fog computing conceptual model," NIST, Tech. Rep., 2018.

[2] H. Feng, G. Mu, S. Zhong, P. Zhang, and T. Yuan, "Benchmark analysis of yolo performance on edge intelligence devices," *Cryptography*, vol. 6, no. 2, 2022.

[3] A. Wachowicz, J. Pytlik, B. Małysiak-Mrozek, K. Tokarz, and D. Mrozek, "Edge computing in iot–enabled honeybee monitoring for the detection of varroa destructor," vol. 32, no. 3, 2022. [Online]. Available: https://doi.org/10.34768/amcs-2022-0026

[4] X. Chang, W. Li, J. Ma, T. Yang, and A. Y. Zomaya, "Interpretable machine learning in sustainable edge computing: A case study of short-term photovoltaic power output prediction," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 8981–8985.

[5] R. Beraldi and G. Proietti Mattia, "On off-grid green solar panel supplied edge computing," in *2022 IEEE 19th International Conference on Mobile Ad Hoc and Smart Systems (MASS) (IEEE MASS 2022)*, Denver, USA, Oct. 2022.

[6] 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*, vol. 14, no. 10, pp. 4548–4556, 2018.

[7] A. Hazra, M. Adhikari, T. Amgoth, and S. N. Srirama, "Fog computing for energy-efficient data offloading of iot applications in industrial sensor networks," *IEEE Sensors Journal*, vol. 22, no. 9, pp. 8663–8671, 2022.

[8] M. Adhikari and H. Gianey, "Energy efficient offloading strategy in fog-cloud environment for iot applications," *Internet of Things*, vol. 6, p. 100053, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660519300265

[9] A. Gougeon, B. Camus, and A.-C. Orgerie, "Optimizing green energy consumption of fog computing architectures," in *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2020, pp. 75–82.

[10] T. Loukopoulos, N. Tziritas, M. Koziri, G. Stamoulis, and S. Khan, "A pareto-efficient algorithm for data stream processing at network edges," vol. 2018-December, 2018, Conference paper, p. 159 – 162, cited by: 5.

[11] X. Lyu, H. Tian, L. Jiang, A. Vinel, S. Maharjan, S. Gjessing, and Y. Zhang, "Selective offloading in mobile edge computing for the green internet of things," *IEEE Network*, vol. 32, no. 1, pp. 54–60, 2018.

[12] Z. Ning, J. Huang, X. Wang, J. J. P. C. Rodrigues, and L. Guo, "Mobile edge computing-enabled internet of vehicles: Toward energy-efficient scheduling," *IEEE Network*, vol. 33, no. 5, pp. 198–205, 2019.

[13] H. Wu, L. Chen, C. Shen, W. Wen, and J. Xu, "Online geographical load balancing for energy-harvesting mobile edge computing," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[14] H. Ma, P. Huang, Z. Zhou, X. Zhang, and X. Chen, "Greenedge: Joint green energy scheduling and dynamic task offloading in multi-tier edge computing systems," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 4, pp. 4322–4335, 2022.

[15] H. Jiang, X. Dai, Z. Xiao, and A. Iyengar, "Joint task offloading and resource allocation for energy-constrained mobile edge computing," *IEEE Transactions on Mobile Computing*, 2022.

[16] A. Mohammadzadeh, M. Akbari Zarkesh, P. Haji Shahmohamd, J. Akhavan, and A. Chhabra, "Energy-aware workflow scheduling in fog computing using a hybrid chaotic algorithm," *The Journal of Supercomputing*, pp. 1–36, 2023.

[17] S. D. Vispute and P. Vashisht, "Energy-efficient task scheduling in fog computing based on particle swarm optimization," *SN Computer Science*, vol. 4, no. 4, p. 391, 2023.

[18] S. Zhu, K. Ota, and M. Dong, "Green ai for iiot: Energy efficient intelligent edge computing for industrial internet of things," *IEEE Transactions on Green Communications and Networking*, vol. 6, pp. 79–88, 2022.