Lifespan and energy-oriented load balancing algorithms across sets of nodes in Green Edge Computing

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

***Abstract*—Green-powered edge computing architectures allow bringing computation in areas that are not reached by the power grids. More often, in applications for Precision Agriculture and Smart Cities, we could have a set of nodes that are coupled with an accumulator which is, during the day, re-charged by the energy harvested by small solar panels. With the latest advances in technology, the edge node is generally assimilated to be a low-power Single Board Computer (SBC), and it is able to carry out even relatively demanding tasks. For example, it can run deep learning models to images or video sequences captured in loco by cameras. However, due to the differences in terms of power consumption and weather conditions, each node experiences a different lifespan, some nodes may even shut down prematurely, causing the interruption of the portion of the deployed service. In this paper, we propose three decentralized algorithms that solve the problem by making the nodes cooperatively balance the traffic in order to level and maximize their lifespan. By comparing the approaches in two different experiments by using a cluster of Raspberry Pi 4 we show that our solutions allow to increase the lifespan of the service of 10% on average wrt the case in which no algorithm is applied.**

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

I. Introduction

Single Board Computers (SBCs) that are generally small in size and have non-negligible computational power, are often suitable to be put in strategic positions, for example, in a Smart City [1] thus concretizing the Edge Computing paradigm. Moving the computation to the Edge enables the possibility of drastically reducing processing latency since it avoids the necessity of sending the data to process in the cloud. The computational power available in the Edge is limited but its peculiar characteristic is that it is efficient in terms of the number of operations that (TOPS) can be carried out per Watt [2]). Moreover, the energy profile is usually restrained and they require an amount of power that can be easily harvested by modern solar panels (Figure 1). In recent years, renewable energies have become a prominent research topic, and they particularly fit the context of Edge Computing [3], thus enabling Green Edge Computing.

In our work, we consider sets of Edge nodes in which one or more services can be deployed. We assume that the power grid is not accessible, this can be the case of rural areas which need to be actively monitored, or even in specific zones of Smart Cities in which the cost of bringing the power grid can be relevant. The solution that we envision is that by using solar energy, nodes can exploit energy harvesting (EH) by themselves.

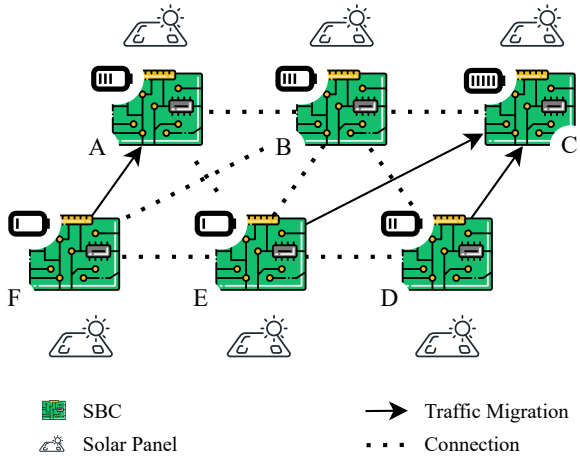


Fig. 1. Scheme of a set of Edge nodes which are coupled with a solar panel and an accumulator (Energy Harvesting technique). The core idea for balancing the energy is offloading a portion of the tasks which a node has to perform to neighbors which have more energy in the batteries, in this case nodes F, E and D.

As anticipated, Edge nodes are characterized by two essential traits: (i) they require low operation power, usually ranging from about 10W (Raspberry Pi 4[1]) to 60W (nVidia Jetson AGX Orin 64GB[2]) and (ii) they have non-neglibible computing power which can also be used to run deep learning inference [4], [5]. We indeed imagine, as a use case, that each node is coupled with a camera and then it has to perform some deep learning task on the captured images or videos.

[1]https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
[2]https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/

The node can also be equipped with special processors (like TPUs or NPUs) which can carry out ML tasks with very high energy efficiency.

We suppose no central entity in our environment, so each node can only interact with its neighbors and the interaction takes the form of offloading thus enabling cooperation. Indeed, each node is requested to perform a certain number of tasks per second (at rate λ) which can for example supposed to be frame processing, and if needed, it can offload a portion of this rate to neighbors. Since each node is characterized by a specific amount of residual battery energy, which determines its lifespan, in order to increase the availability of the service, nodes need to balance their energy consumption. It can be easily shown that what dominates the energy consumption is the rate λ, therefore the task offloading causes a decrease of energy consumption in the node which forwards the tasks and an increase in the one which receives them. Figure 1 shows a simplified environment characterized by a set of Edge nodes that are inter-connected and, for example, Nodes F, E and D forward their traffic to a given neighbor. Then, we also need to consider that Edge nodes are heterogeneous, they have different rates of task execution[3] (which we call μ) and different energy consumptions, therefore there is the need to design an algorithm which finds the correct amount of tasks that each node needs to offload to others to maximize the lifespan of the service in the long run and possibly avoiding a node's shutdown due to low battery. The contributions of this paper can be summarized as follows:

1) design of three decentralized algorithms which balance the energy consumption of the nodes by relying on cooperative tasks offloading;
2) benchmark of the algorithms in a cluster of 11 Raspberry Pi 4 SBCs by using the FaaS as task model and the P2PFaaS [6] framework for the implementation;
3) definition of a set of performance metrics targeting the behavior of the algorithms with respect to service availability, lifespan and lifespan variance;
4) comparison of the algorithm both in a standalone scenario and in a solar panel assisted scenario [7] which solar energy traces from real panels

The rest of the paper is organized as follows. In Section II we compare our work to other similar works in literature which address energy and load balancing in Edge Computing, in Section III we provide the model of the system and the performance evaluation metrics, then in Section IV we illustrate the proposed algorithms and in Section V we provide the performance comparison of the algorithms in the experimental setting. Finally, we draw the conclusions in Section VI. Table I shows all the symbols used in the paper.

II. Related Work

This paper aims to design distributed and decentralized algorithms which exploit load balancing for adjusting the energy consumption of nodes in Fog and Edge Computing with the final objective of extending the lifespan of the nodes and, as a consequence, of the service itself. We present in this section a series of similar works that target energy-aware load balancing.

We can define a first set of works that target low-level packet routing and load balancing by taking into consideration the energy aspect. Adil et al., for example, in [8] focus on Wireless Sensor Networks (WSNs) which are networks characterized by nodes that are usually powered by batteries and therefore they must efficiently communicate over the network to maximize the lifespan. The authors indeed propose an energy-efficient load balancing scheme that is based on Energy Gauge Nodes (EGNs) which advertise the residual energy of the nodes and this information is used for performing a uniform load balancing of packets. A similar approach is followed by Sampayo et al. [9] which explores the usage of wake-up radio which has ultra-low power consumption, and Ali et al. [10] which exploits clusters energy balanced nodes. Differently from our work, the authors of these works are not specifically targeting Edge and Fog Computing, in which the nodes are not only sensors but can actively execute the whole or part of the computation, moreover, we also consider the green energy sources.

The second set of works instead more specifically regards Green Edge Computing [11]. In particular, Lyu et al. in [12] 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 is tested only in simulation and it is not considering the energy contribution that is harvested from the solar panels. In [13] 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 decentralized. Wu et al. in [14] 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 [15] which proposes a hierarchical task offloading that optimizes latency, energy consumption and cloud fees. However, in these works, which only provide models and simulation, the decentralized approach is not considered, which is the core of our work. Similar approaches are then used in [16].

Other approaches focused on energy-aware task scheduling can be seen in [17], [18] and [19].

III. Model & Metrics

We suppose to have a set of $\mathcal{N}$ edge nodes. Each node i in the set receives a constant rate of tasks per second to be executed which is called λ_i, these tasks can be associated

[3]The number of tasks that a node can execute in a given second

Symbol	Meaning
$\mathcal{N}$	Set of nodes
$x_i(t)$	Net load (in req/s) of node i at time t
λ_i	Traffic to node i (in reqs/)
μ_i	Service rate of node i (in reqs/s)
$\rho_i(t)$	Utilization rate of node i at time t (defined as λ_i/μ_i)
$m_{ij}(t)$	Percentage (over λ_i) of tasks forwarded from node i to node j at time t
P_{IDLE}	Idle power absorption of a node i (in W)
P_{WORK}	Power absorption for unit of load $\rho_i(t)$ for node i (in W)
B_i	Initial battery energy of node i (in Wh)
$l_i(t)$	Residual lifespan at time t for node i
$p_i^l(t)$	Power absorption of node i depending on the load (in W)
$p_i^t(t)$	Total power absorption of node i
$b_i^p(t)$	Residual battery energy at time t for node i
$s_i(t)$	Power harvested by the solar panel attached to node i at time t (in W)
$e_i(t)$	Energy reference value at time t for node i which can be $b_i^p(t)$ or $l_i(t)$ depending on the experiment
$e_i^a(t)$	Average energy reference value at time t between node i and its neighbors at time t
τ	round time in round-based algorithms

TABLE I
LIST OF SYMBOLS USED

with the processing of the data generated at the node at the same rate, in our use case a task is an image frame processing. The same nodes can execute tasks at rate μ_i which is again supposed to be constant over time. From experimental measures, we assessed that SBCs consume an approximately fixed amount of power when they are idle, and we call this contribution P_{IDLE}, then there is another contribution that is variable since it depends on the load at time t which we call $p_l(t)$. We can define the total power consumption of the node i at time t as:

$$p_i^T(t) = P_{\text{IDLE}} + P_{\text{WORK}}^l(t) \tag{1}$$

In general, nodes' variable energy consumption derives from many micro-operations that beyond the CPU time can regard data transmission over Ethernet (or WiFi), data read or written in RAM or even access to persistent storage. Without loss of generality, we can assume that all of these operations are proportional to the number of tasks that the nodes have to execute because these tasks may involve all the above-mentioned operations. With this assumption the contribution to the power consumption $p_l(t)$ is determined by the effective rate of tasks that the node i is executing at time t which can be seen as:

$$x_i(t) = \lambda_i - \sum_{j \in \mathcal{N}} \lambda_i m_{ij}(t) + \sum_{j \in \mathcal{N}} \lambda_j m_{ji}(t) \tag{2}$$

where we call $m_{ij}(t)$ the percentage of tasks generated by node i which are forwarded to j at time t in a cooperative manner. Since we consider small sets of edge nodes we suppose these nodes to be arranged in a fully connected topology and therefore the net number of tasks rate that a node executes is given by the tasks that it generates from which we subtract the rate of task offloaded and the rate of tasks received from the neighbors. If we call $\rho_i(t)$ the node's load at time t

$$\rho_i(t) = \frac{x_i(t)}{\mu_i} \tag{3}$$

and we define P_{WORK} as the amount of power required by node i for a unit of load. Then, we can rewrite Equation 1 as:

$$p_i^T(t) = P_{\text{IDLE}} + P_{\text{WORK}}\rho_i(t) \tag{4}$$

The value P_{WORK} in Watts (W) can be easily extracted by running benchmarks on real devices. We also assume that to each node i is attached a battery of initial capacity B_i^0 which is assumed to match the full capacity B_i. Moreover, the battery can be recharged with the power harvested by a solar panel which is described by the function $s_i(t)$. We define the law which regulates the residual battery capacity over time t as:

$$b_i(t) = B_i^0 - \int_0^t p_i^T(t) + \int_0^t s_i(t) \tag{5}$$

We proceed to find the solution of the problem, which are the $m_{ij}(t)$ functions, by proposing three different algorithms in Section IV whose purpose is the one of making nodes that have less energy availability to forward to nodes which have more. Further development of the model is left as future work since in this work we focus on the effective solution applied to a cluster of real SBCs.

A. Performance Metrics

The proposed algorithms have been evaluated according to the following metrics:

- σ: the average variance between the battery capacities over the entire experiment. The metric allows us to measure how the algorithm is able to keep the batteries' charge aligned over time. We divide the experiment into n equal slots of time and we measure the average battery capacity in every segment. In formulas, for a given node i and the j-th segment of time which starts from t_j^s to t_j^e we define the average battery of a node as:

$$b_{a,i,j} = \frac{1}{t_j^e - t_j^s} \int_{t_j^s}^{t_j^e} b_i(t) \tag{6}$$

Then in a given slot j we compute the variance of the average batteries capacities as

$$\sigma_{i,j} = \frac{1}{j} \sum_{k=0}^{j} (b_{a,i,k} - \frac{1}{N} \sum_{z=0}^{|\mathcal{N}|} b_{a,z,j})^2 \tag{7}$$

and finally, we derive the metrics which best describe the behavour of the variance during the experiment:

$$\sigma = \frac{1}{N} \sum_{i=0}^{n} \sigma_{i,j} \tag{8}$$

- d_f: the time in which the first node fully discharges and dies. Since the service becomes compromised when some nodes become unusable, this value indicates how long the service remains at full potential. The metric is described as, given T_d the set of t_i^* which is the time in which node i dies (for which $b_i(t_i^*) = 0$)

$$d_f = \min T_d \tag{9}$$

- d_g: the time gap between the first node that reaches zero battery capacity value which declares its death and the last one which reaches the same state. The metric is describes as, given T_d the set of t_i^* which is the time in which node i dies (for which $b_i(t_i^*) = 0$):

$$d_g = \max T_d - \min T_d \tag{10}$$

- r: the average percentage of tasks that are not served by the nodes. Since nodes have finite queues when they saturate or the battery runs out they start to drop requests, given r_i as the percentage of tasks rejected by a node i, the metric in question is the average among all the nodes;
- $e_i^m(t)$: defines the difference between the current energy reference value of node i and the minimum value among all the nodes, in other words:

$$e_i^m(t) = e_i(t) - \min_j e_j(t) \tag{11}$$

The metric is mainly used in the charts for visualizing the behavior of the balancing algorithm over time. The energy reference value $e_i(t)$ is described in the next section.

IV. Proposed Algorithms

We now present three distributed and decentralized load-balancing algorithms whose purpose is to balance the energy consumption of the nodes to extend the lifespan of the service. In each algorithm, we assume that a node receives a task to be executed and a scheduling decision must be taken. The decision describes whether to execute it locally or to forward it to a neighbor node.

The purpose of the algorithms follows the general idea that envisions a node to forward part of its tasks to nodes which is in a "better state". Regarding the state we both consider the residual battery level (in percentage) which, at time t, can be expressed as:

$$b_i^p(t) = \frac{b_i(t)}{B_i} \tag{12}$$

and the lifespan which, at time t, is described as:

$$l_i(t) = \frac{b_i(t)}{p_i^t(t)} \tag{13}$$

Therefore the proposed algorithms can be applied both by considering as reference parameter Equation 12 and Equation 13. Given any two nodes $i, j \in \mathcal{N}$ s.t. $i \neq j$ at time t we can encode that, for example, j is in a "better" state with respect to i both as: (i) j has a higher residual battery percentage and therefore $b_p^j(t) > b_i^p(t)$ or (ii) j has a greater lifespan than i and therefore $l_j(t) > l_i(t)$. In the algorithms that we are going to present in this section, we encode the function "GetEnergyState()" which retrieves the battery percentage or the residual lifespan, in the text we will refer to it as energy reference value $e_i(t)$ which can assume the values $b_i^p(t)$ or $l_i(t)$ according to the specific experiment.

A. Random Choice

The first algorithm that we present is based on random choice. The random approach in used in balancing algorithms may seem a little bit controversial, however, there is a wide literature that shows that the approach has a beneficial effect, especially when the number of nodes increases [20]. In order to appreciate the effect of randomness we do not have to forward tasks blindly to random nodes but when a task arrives at time t we probe d random one node asking their state and then if a node is found in a "better" state, the task is forwarded [21] to it otherwise it is executed locally. Algorithm 1 shows the proposed algorithm based on the random choice. The flow of the algorithm is the following, suppose that the algorithm is executed by node i:

1) first of all, we pick d random nodes from the neighbours of the current node i;
2) then for each of the picked nodes we explicitly ask them the state;
3) if a node j with a better state is found, then the task is forwarded to it;
4) in any other case, the task is executed locally.

Algorithm 1 Random Choice Balancer

```
Ensure:
  neighsIPs ← list of neighbor nodes' IP from configuration
Require:
  task ← task to be executed
  d ← number of neighbor nodes to probe

  function SCHEDULE(task, d)
      selfState ← GetEnergyState()
      [1. Pick d random neighbor nodes]
      randomNodes ← pickdRandomNodes(neighsIPs, d)
      [2. Retrieve the state of the picked nodes]
      for ip ∈ randomMachines do
          randomState ← GetEnergyState(ip)
          [3. Check if the probed node has a better state]
          if randomState > selfState then
              [3a. If yes, forward the task to it]
              ForwardTask(task, ip)
              return
          end if
      end for
      [4. If no node in a better state is found, then execute the task locally]
      Execute(task)
  end function
```

The purpose of the algorithm is to avoid the probing of all the neighbor nodes which may introduce latency and relies on the probability of finding a node that is in a better state. The mathematical model and the assumption behind this idea are well-studied in the above-mentioned works.

We remark that, in this particular approach, we are not interested to find directly the migration ratios $m_{ij}(t)$ from

any node $i, j \in \mathcal{N}$ since we apply a random decision to each task that arrives to the node.

B. Ratio approach

The second approach is specifically designed to find the migration ratios $m_{ij}(t)$ which describe, for any two nodes $i, j \in \mathcal{N}$ the percentage of the arrival task rate to node i, λ_i, that is forwarded to node j.

The proposed approach divides the time into rounds of τ seconds. When the round time is hit the migration ratios for a given node $i \in \mathcal{N}$ are updated. Moreover, to improve the stability of the found ratio, due to the distribution of tasks' arrivals (which we will assume to be Poissonian during the experiments), we introduce a tolerance coefficient ϵ which is used in the criterium for establishing if a node is in a better state. In particular, given any two nodes $i, j \in \mathcal{N}$, we consider a node j to be in a better state of i if

$$e_j(t) > e_i(t)(1+\epsilon) \tag{14}$$

The fundamental idea of the algorithm is that, at each round, we make each node i equally forward all its incoming tasks at rate λ_i to all the neighbors which are in a better state, if they exist. Then, if performing this action will cause a certain neighbor node to worsen its state, the node will reset the migration ratio towards it. Even if the behavior is all or nothing we expect the energy reference value (battery or lifespan) to be equalized and maximized over time.

Algorithm 2 shows the complete pseudocode of the proposed approach which is triggered every τ seconds. The flow of operations is the following:

1) probe all the neighbor nodes for their state and store into a list the nodes for which the condition in Equation 14 is true. For any node j that is found to not have a better state, the ratio is reset;
2) update the ratios towards all nodes in the list by distributing the traffic totally and equally.

C. Ratio approach with adaptive step

The last algorithm that we present follows from Algorithm 2 and it has the same purpose of finding the migration ratios $m_{ij}(t)$ for a given node i, a similar approach has been applied in case of leveling of latency [22]. Even in this approach, we reason with rounds of duration τ seconds and at the end of the round each node updates its migration ratios. The difference with the previous algorithm is that, first of all, any given node $i \in \mathcal{N}$ computes the average energy reference value (which can be the battery percentage or the lifespan) between itself and its neighbors, then it increases by a step size α the migration ratio m_{ij} to the nodes whose state is above the average and decreases it by a step size α to the nodes whose state is lesser than the average, and this is done by always keeping $m_{ij}(t)$ positive. Since arrivals are in general not exactly periodic, as in the previous approach, we define a node that is balanced when

$$e_i^a(t)(1-\epsilon) \leq e_i(t) \leq e_i^a(t)(1+\epsilon) \tag{15}$$

Algorithm 2 Ratio Balancer with Equidistribution

```
Ensure:
  neighsIPs ← list of current node's neighbours
  ratios ← array of floats one for each neighbor in neighsIPs
  τ ← round time
  lastRatioUpdate ← the timestamp of last ratios update
Require:
  task ← task to be executed
  ε ← the better state tolerance value

  function UPDATERATIOSEQUIDISTRIBUTION(neighsIPs, ε)
      list ← [ ]
      selfState ← GetEnergyState(self)
      [1. Loop over every neighbor]
      for ip ∈ neighsIPs do
          neighbourState ← GetEnergyState(ip)
          [2. Check if the current neighbor has a (1 + ε) better state]
          if neighbourState > selfState · (1 + ε) then
              [2a. If yes then save its IP to a list]
              list.append(ip)
          else
              [2b. Otherwise, reset the ratio to it]
              ratios[ip] ← 0
          end if
      end for
      if len(list) > 0 then
          [3. Equidistribute the λ rate to all neighbors in a better state]
          value ← 1/len(list)
          for ip ∈ list do
              ratios[ip] ← value
          end for
      end if
      return ratios
  end function
```

We call this zone of size $2\epsilon e_i^a(t)$, a tolerance zone and a node for which Equation 15 holds at time t does not need to alter any migration ratio. Therefore the algorithm is triggered only if a node is outside the zone.

Algorithm 3 shows the pseudocode of the proposed approach. The flow of operations is the following:

1) compute the average of the energy reference value $e_i(t)$, then the upper and the lower limits of the tolerance zone;
2) at this point we check the condition for which the current node i's state is below the tolerance zone and the migration ratios are not zero, this means that the node is forwarding an amount of traffic that is excessive and causes it to go below the zone[4]. If the condition is true then we decrease all the positive migration ratios by the step size α.
3) after reducing the ratios, check if the current state is above the lower limit of the tolerance zone and if this is true we do not further step since the node cannot increase the ratios and we assume it in balance;
4) if the node reaches this point then it is below the tolerance zone and the ratios must be altered, in particular for going in balance we
   a) reduce by α the ratios towards every node neighbor $j \in \mathcal{N}$ that is below the balance zone, because for

[4]This because the more the traffic is forwarded the less is the energy consumption of the node and therefore the higher is the energy reference value (battery or lifespan).

going in balance it needs to increase their energy reference value $e_j(t)$;

b) increase by α the ratios towards the nodes which are above the balance zone, in this way, we increase the energy reference value of the current node for decreasing the one of the node to which the traffic is offloaded.

Algorithm 3 Ratio Balancer with Adaptive Step

```
Ensure:
  neighsIPs ← list of current node's neighbours
  ratios ← array of floats one for each neighbor in neighsIPs
  τ ← round time
Require:
  task ← task to be executed
  ε ← the better state tolerance value
  α ← the migration ratios' step size

  function UPDATERATIOSADPTSTEP(ratios, neighsIPs, ε, α)
    [1. Compute the average energy value and the tolerance zone limits]
    states ← GetEnergyStates(neighsIPs)
    avgEnergy ← sum(states) / len(states)
    avgHigh ← avgEnergy · (1 + ε)
    avgLow ← avgEnergy · (1 − ε)

    [2. Check if the current state is above the average and ratios must be reduced]
    selfState ← GetEnergyState(self)
    if selfState > avgHigh then
      for ip ∈ neighsIPs do
        [2a. Reduce of step α every positive ratio]
        if states[ip] > 0 then
          ratios[ip] ← ratios[ip] - α
        end if
      end for
    end if
    [3. Check current state is above the low limit of the tolerance zone, in that case, the node is balanced]
    if selfState > avgLow then
      return ratios
    end if
    for ip ∈ neighsIPs do
      [4. Reduce the ratio to nodes that are below the tolerance zone]
      if states[ip] ≤ avgLow then
        ratios[ip] ← ratios[ip] - α
      end if
      [5. Increase the ratio to nodes that are above the tolerance zone]
      if states[ip] > avgHigh and sum(ratios) < 1.0 then
        ratios[ip] ← ratios[ip] + α
      end if
    end for
    return ratios
  end function
```

V. Experimental Results

The algorithms presented in Section IV have been implemented in the P2PFaaS framework [6] which we envisioned and implemented for testing distributed scheduling and load balancing algorithms. The paradigm used as a task model is the Function-as-a-Service (FaaS), therefore we envision that every node $i \in \mathcal{N}$ generates a rate λ_i of function execution requests per second, then the scheduling decision is made per each request upon its generation. We suppose each node to run at maximum $K = 4$ requests in parallel with no queue, therefore if a node decides to execute a request locally and it is executing exactly four requests, the new upcoming request is automatically rejected, thus increasing the r metric.

The framework has been installed on 11 Raspberry Pi 4, however, since the devices had a not real solar panel and battery attached we simulated them. We introduced a new module in the framework which simulates the energy discharge by periodically reading the actual CPU usage of the board and reducing the capacity of a virtual battery. The CPU time gives an accurate estimation of the device's current load and therefore it has been used as $\rho_i(t)$ in the Equation 4. The energy module implements instead Equation 5 with ticks that update the battery triggered every second. From real measures we assumed, in every experiment and without loss of generality, that the idle power of a board as $P_{\text{IDLE}} = 2.5W \ \forall i \in \mathcal{N}$ and the power consumption for a unit of load as $P_{\text{WORK}} = 0.025W \ \forall i \in \mathcal{N}$, in this way, the boards in full load have a total power absorption of $5W$. The capacity of the batteries and the initial energy as been instead set to $B_i = 10\text{Wh} \ \forall i \in \mathcal{N}$. Finally, the arrival rates match the node index, starting from $i = 1$ to $i = 11$, $\lambda_i = i$.

A. Discharge

The first experiment that we carried out assumes that the contribution of the solar panels is absent, i.e. $s_i(t) = 0 \ \forall i \in \mathcal{N}, \forall t \geq 0$ in Equation 5. Table II shows the results of the experiment in the case in which no algorithm is applied to the system and then for every proposed algorithm applied both to the battery capacity ($e_i(t) = b_i(t)$) and to lifespan ($e_i(t) = l_i(t)$). What we can observe is that the approach that equally distributes the traffic among all the neighbors which are in a better state (Algorithm 2) when applied to lifespans is the one which makes the $e_i(t)$ more stable (since the variance $\sigma = 0.04$ is minimized) and which minimizes the gap between the first and the last nodes that run out of battery ($d_g = 3$s) and also maximizes the lifespan of the first node which goes offline (with 2 hours, 27 minutes and 22 seconds). This is because the random approach generates an excessive variance and since it is random it is less precise in selecting the node for which the migration ratio must be increased, then the approach with the adaptive step is instead too slow in adjusting the migration ratios and therefore forwarding all the traffic all at once is the approach which performs better. Then we need to highlight two aspects of these results. First of all, the approaches which use the lifespan are as expected the ones which maximize the d_f metric because they have the advantage of exactly working with the metric. The second aspect regards the probability of a task being rejected which is r. In general, none of our approaches focused directly on that particular metric, because the objective of all the algorithms was to extend the minimum lifespan, however, an approach cannot extend the lifespan by rejecting the tasks and thus reducing the energy consumption. This kind of reasoning opens for a multi-objective study which is left as future work, however, the approach which is resulted to be the better still gives a fair $r = 16.5\%$ with respect to 13.2% which is the case in which no load balancing is applied. This happens because it can be seen as a partial side-effect of

balancing the load for increasing the lifespan, indeed, a node offload parts of its traffic to nodes that have a greater energy availability, probably because they have carried out less work with respect to nodes that forwards the traffic. This particularly holds when the energy consumption per unit of load (P_{WORK}) is the same for every node, which is our specific case.

Balancers	Based on	$\sigma \downarrow$	$d_g \downarrow$	$r \downarrow$	$d_f \uparrow$
No Balancing	-	140	00:30:40	**13.2%**	02:14:17
Random (Alg. 1)	battery	1.47	00:02:02	15.7%	02:20:22
Random (Alg. 1)	lifespan	0.16	00:00:29	19.5%	02:20:44
Ratio E. (Alg. 2)	battery	0.25	00:00:44	34%	02:14:52
Ratio E. (Alg. 2)	lifespan	**0.04**	**00:00:03**	16.5%	**02:27:22**
Ratio A. (Alg. 3)	battery	2.75	00:01:55	16.5%	02:10:10
Ratio A. (Alg. 3)	lifespan	4.81	00:00:19	15.3%	02:10:53

TABLE II
COMPARISON OF THE PROPOSED ENERGY BALANCING ALGORITHMS DURING DISCHARGE UNTIL THE SHUTDOWN OF ALL THE NODES WITH NO RECHARGE DURING THE EXPERIMENT. VALUES OF σ ARE INTENDED TO BE MULTIPLIED BY 10^{-3}.

B. Discharge and Solar Recharge

The purpose of this second experiment is to measure how the proposed algorithms behave when, as it could happen in a real environment, the batteries are recharged during the day according to the solar activity. The amount of energy harvested by the solar panels used in the experiment has been taken from real home-designed solar panels[5] over 3 days of activity, the data were re-scaled to match 3 hours under the conditions of the experiment in which we suppose that each node is attached to its own polycrystalline solar panel with a rated maximum power of 14W[6]. In this process, we suppose that the differences in the traces are due to the different geographical positions of the nodes.

Figure 2 shows the energy harvested by three solar panels. The traces have been applied to the 11 nodes by using the following fashion: nodes 1, 4, 7 and 10 follow the trace no. 1 in Figure 2a; nodes 2, 5, 8 and 11 follow the trace no. 2 in Figure 2b; nodes 3, 6 and 9 follow the trace no. 3 in Figure 2c. The configuration of the node batteries remains the same as the ones in the previous test (initial capacity $B_i = 10$Wh, $P_{\text{IDLE}} = 2.5$W and $P_{\text{WORK}} = 0.025$). The arrival rate to the node is also the same as the previous experiment (Section V-A).

Table III shows the final results of the experiment which has been conducted on all the algorithms as the previous one. The purpose of the experiment is to show how the algorithms react when the energy in the battery can also unpredictably increase. In this case, the random approach based on lifespan is the one that maximizes the minimum lifespan d_f (3 hours, 46 minutes and 38 seconds) and the variance $\sigma = 1.02$, this is because when energy is more dynamic the random guess is able to better capture the overall behavior of the system. Instead, the algorithm based on equidistribution of traffic (Algorithm 2 based on lifespan) minimizes the time gap d_g but shows an unacceptable task rejection probability r. This is because the all-or-none approach is not able to follow a dynamic energy behavior.

Balancers	Based on	$\sigma \downarrow$	$d_g \downarrow$	$r \downarrow$	$d_f \uparrow$
NoScheduler	-	1.57	01:29:44	**11.6** %	03:21:50
Random (Alg. 1)	battery	1.03	00:42:14	14.7 %	03:46:01
Random (Alg. 1)	lifespan	**1.02**	00:39:49	16.2 %	**03:46:38**
Ratio E. (Alg. 2)	battery	1.08	**00:38:00**	38.6 %	03:22:46
Ratio E. (Alg. 2)	lifespan	1.15	00:43:14	18.2 %	03:25:36
Ratio A. (Alg. 3)	battery	1.17	00:51:51	26.3 %	03:18:02
Ratio A. (Alg. 3)	lifespan	1.06	00:47:12	23.2 %	03:20:23

TABLE III
COMPARISON OF THE PROPOSED ENERGY BALANCING ALGORITHMS DURING DISCHARGE UNTIL THE SHUTDOWN OF ALL THE NODES WITH SIMULATED RECHARGE DURING THE EXPERIMENT FOLLOWING THE SAME CURVES. VALUES OF σ ARE INTENDED TO BE MULTIPLIED BY 10^{-3}.

Figure 3 shows the trace of $e_i^m(t)$ regarding the battery capacity over time ($e_i(t) = b_i(t)$) when no balancing algorithm is used, while Figure 4 shows the same trace when the best approach is used, which is Algorithm 2 with energy reference value the lifespan of nodes ($e_i(t) = l_i(t)\ \forall i \in \mathcal{N}$). This illustrates how the proposed algorithm can reduce over time the variance between the batteries thus extending the lifespan of 25 minutes with respect to the case in which no balancing is used.

VI. CONCLUSIONS

In this paper, we presented three decentralized load balancing algorithms that level the energy consumption of the nodes thus extending the lifespan of the nodes and therefore of the service in Green Edge Computing environments. Indeed, we show a brief mathematical model of the system, the pseudocode of the approaches and the experimental results carried out in a cluster of 11 Raspberry Pis with simulated batteries and solar panels. However, further investigations are needed, especially regarding the mathematical model which can be further developed with the communication impact on energy for finding if and under which conditions a solution exists, even when considering particular topologies that do not match a fully connected graph.

ACKNOWLEDGEMENTS

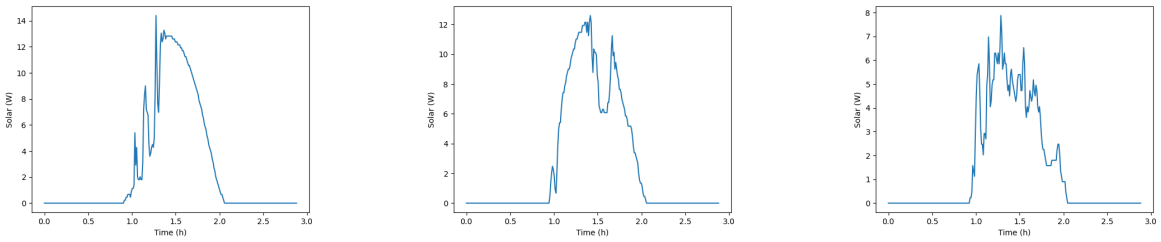
The authors would also like to thank the former master's student Marco Ciancia for his implementation and report of the proposed algorithms in a cluster of Raspberry Pi 4 boards by using the P2PFaaS Framework [6].

REFERENCES

[1] B. Qureshi, K. Kawlaq, A. Koubaa, B. Saeed, and M. Younis, "A commodity sbc-edge cluster for smart cities," in *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, 2019, pp. 1–6.

[5]These panels, Jinko Solar JKM410M-72H, have an efficiency of 20% in standard test conditions (STC) and in our tests, 20 panels harvested at maximum 7kW of power.

[6]A low-cost polycrystalline panel has an efficiency of 12% (in STC) this means that we can suppose that each node is attached to a panel of 45×30cm.



(a) Trace no. 1 (day 1). (b) Trace no. 2. (day 2) (c) Trace no. 3. (day 3)

Fig. 2. Traces of the power harvested by home-sized solar panels normalized to match panels compatible with Edge Computing, assuming a 12V, 30×40cm polycrystalline panel with a maximum harvesting power of 40W. The traces recorded during three days have been normalized to match the three hours of the benchmark.

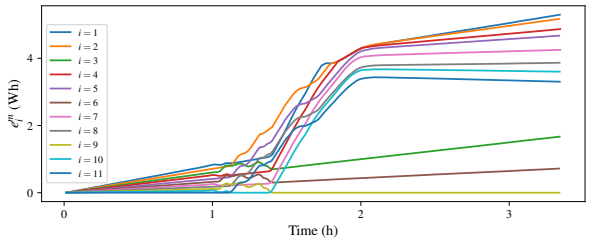


Fig. 3. Trace of $e_i^m(t)$ for every node during the experiment in which no balancing algorithm is used but with energy harvesting from solar panels. The chart shows how the batteries' energy diverges over time.

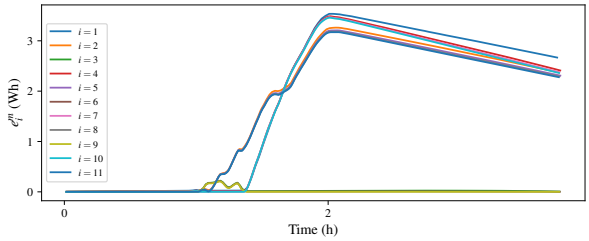


Fig. 4. Trace of $e_i^m(t)$ for every node during the experiment with Algorithm 1 is used with energy reference value the lifespan of nodes ($e_i(t) = l_i(t)\ \forall i \in \mathcal{N}$) and with energy harvesting from solar panels. The chart shows how the batteries' energy is more aligned with respect to the case in which no balancing is used.

[2] F. Kaup, S. Hacker, E. Mentzendorff, C. Meurisch, and D. Hausheer, "The progress of the energy-efficiency of single-board computers," *Tech. Rep. NetSys-TR-2018-01*, 2018.

[3] W. Li, T. Yang, F. C. Delicato, P. F. Pires, Z. Tari, S. U. Khan, and A. Y. Zomaya, "On enabling sustainable edge computing with renewable energy resources," *IEEE Communications Magazine*, vol. 56, no. 5, pp. 94–101, 2018.

[4] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[5] Z. Chen, Q. He, L. Liu, D. Lan, H.-M. Chung, and Z. Mao, "An artificial intelligence perspective on mobile edge computing," in *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*, 2019, pp. 100–106.

[6] G. Proietti Mattia and R. Beraldi, "P2pfaas: A framework for faas peer-to-peer scheduling and load balancing in fog and edge computing," *SoftwareX*, vol. 21, p. 101290, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711022002084

[7] R. Beraldi and G. P. Mattia, "On off-grid green solar panel supplied edge computing," in *2022 IEEE 19th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*, 2022, pp. 794–799.

[8] M. N. Adil, R. Khan, M. A. Almaiah, M. H. Binsawad, J. Ali, A. Saaidah, and Q. T. H. Ta, "An efficient load balancing scheme of energy gauge nodes to maximize the lifespan of constraint oriented networks," *IEEE Access*, vol. 8, pp. 148 510–148 527, 2020.

[9] S. L. Sampayo, J. Montavont, and T. Noël, "elobaps: Towards energy load balancing with wake-up radios for iot," in *Ad-Hoc, Mobile, and Wireless Networks*, M. R. Palattella, S. Scanzio, and S. Coleri Ergen, Eds. Cham: Springer International Publishing, 2019, pp. 388–403.

[10] S. A. Ali and C. Sevgi, "Energy load balancing for fixed clustering in wireless sensor networks," *2012 5th International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, 2012.

[11] Z. Ning, X. Kong, F. Xia, W. Hou, and X. Wang, "Green and sustainable cloud of things: Enabling collaborative edge computing," *IEEE Communications Magazine*, vol. 57, no. 1, pp. 72–78, 2019.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, 2001.

[21] R. Beraldi and G. Proietti Mattia, "Power of random choices made efficient for fog computing," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2020.

[22] G. Proietti Mattia, A. Pietrabissa, and R. Beraldi, "A load balancing algorithm for equalising latency across fog or edge computing nodes," *IEEE Transactions on Services Computing*, pp. 1–12, 2023.