information

MDPI

Article

Resource Allocation and Pricing in Energy Harvesting Serverless Computing Internet of Things Networks

Yunqi Li [1,*] and Changlin Yang [2]

[1] School of Computer Science, Zhongyuan University of Technology, Zhengzhou 450007, China
[2] School of Software Engineering, Sun Yat-sen University, Zhuhai 528406, China
* Correspondence: liyunqi@zut.edu.cn

Abstract: This paper considers a resource allocation problem involving servers and mobile users (MUs) operating in a serverless edge computing (SEC)-enabled Internet of Things (IoT) network. Each MU has a fixed budget, and each server is powered by the grid and has energy harvesting (EH) capability. Our objective is to maximize the revenue of the operator that operates the said servers and the number of resources purchased by the MUs. We propose a Stackelberg game approach, where servers and MUs act as leaders and followers, respectively. We prove the existence of a Stackelberg game equilibrium and develop an iterative algorithm to determine the final game equilibrium price. Simulation results show that the proposed scheme is efficient in terms of the SEC's profit and MU's demand. Moreover, both MUs and SECs gain benefits from renewable energy.

Keywords: edge computing; resource allocation; energy management; game theory

Citation: Li, Y.; Yang, C. Resource Allocation and Pricing in Energy Harvesting Serverless Computing Internet of Things Networks. *Information* **2024**, *15*, 250. https://doi.org/10.3390/info15050250

Academic Editor: Aneta Poniszewska-Maranda

Received: 5 April 2024
Revised: 22 April 2024
Accepted: 23 April 2024
Published: 29 April 2024

1. Introduction

The emergence of cloud computing has overcome the resource limitations of small-scale smart devices, thereby allowing devices with insufficient computing resources to offload complex tasks to central clouds through wireless networks and leveraging the nearly unlimited computing resources of cloud servers for processing [1]. However, with the rapid proliferation and development of Internet of Things (IoT) technology, emerging applications such as autonomous driving and artificial intelligence continue to emerge. These tasks typically require significant computing resources and are highly sensitive to task execution latency [2]. A large number of computationally intensive, latency-sensitive tasks will lead to congestion in cloud computing communication channels, thus affecting system performance [3]. Additionally, since cloud servers are often far from the user end, this further increases the latency of data transmission.

To address the latency issues associated with task offloading, mobile edge computing (MEC) has garnered extensive attention from researchers and has become one of the most popular paradigms [4]. MEC extends the services provided by the cloud to the edge of a network, where servers are deployed near end users. Consequently, tasks from mobile users (MUs) can be computed at a nearby server. Advantageously, MUs do not necessarily have to offload their tasks to the cloud, which may be located far away. Instead, edge servers provide MUs with a quick response and help conserve the energy of MUs [5]. Hence, MEC plays a critical role in providing latency-sensitive services and forms a key part of future 5G/6G networks [6].

A key concern when operating servers is their energy consumption. In particular, the energy consumption of the information and communication technology (ICT) sector has continued to rise in recent years [7] in accordance with increasing traffic. As reported in [8], the energy attributed to the production and operation of devices and infrastructure amounted to 14% of the global greenhouse emission in 2016. To this end, many works have considered different ways to minimize the carbon footprint of network equipment [9].

For example, in [10], the authors survey works that consider base stations powered by renewable energy. Another example is [11], where the authors reviewed energy-aware hardware and software for edge computing. Further, for MEC, works such as [12] have considered powering servers with a renewable energy source. Further, in [13], the authors studied distributed servers powered by solar and power grid.

Many works assume that servers have all the functions required by tasks offloaded by MUs. This, however, becomes an issue, as servers have limited storage. This means it becomes impractical to store all the possible functions required by various tasks/applications. Further, resources, e.g., processing and memory, may not be used efficiently if applications or functions run infrequently. These facts motivate the serverless computing paradigm [14] or function-as-a-service (FaaS). A provider, e.g., Amazon, manages the infrastructure for FaaS and orchestrates the resources required to run functions/tasks. Furthermore, it charges users based on the amount of resources, e.g., CPU and memory, used by their functions. On the other hand, a developer only needs to specify the necessary functions and services. Advantageously, the developer does not have to be concerned with the management of resources or servers. These responsibilities are handled by a provider who will automatically scale computing resources according to the resource requirements of functions [15]. The serverless computing paradigm has also been extended to the edge or so-called serverless edge computing (SEC) [16]. This feature is particularly attractive to Internet of things (IoT) applications, where devices may operate infrequently. This means a server does not have to preload all possible functions in order to support IoT devices [14]. It helps facilitate applications, such as auto driving [17], that require quick responses. Furthermore, it helps overcome issues with the current cloud computing paradigm, including its inability to meet requirements such as low latency, location awareness, and mobility support. This avoids transmitting tasks to the cloud that may lead to congestion or incur large delays [18].

In this paper, we continued the research from our previous work [19] and delved deeper into the pricing issues of serverless edge computing. We consider an SEC system whereby an operator manages one or more servers. These servers execute offloaded tasks from one or more MUs. Furthermore, they are powered by *both* the grid and a renewable energy source. A request from an MU is regarded as an event. When an MU sends a request to a server, it will trigger one or more specific functions or containers on the server. If the required functions are not available, the server must download them from the cloud. This incurs an additional cost for the server. Finally, it returns the result to the MU that made the request.

Figure 1 shows an example SEC system. The server is equipped with solar panels and batteries to collect and store solar energy to provide energy for the server. There are three different functions, f_1, f_2, and f_3, which correspond to three different function modules. Specifically, MU 1 and MU 5 require f_1, MU 2 and MU 4 require f_2, and MU 3 requires f_3. The tasks of MU 1 can be completed by using function f_1, which is stored by server 1. For tasks of MU 2, MU 3, and MU 4, functions f_2 and f_3 stored by servers 2 and server 3 are used. However, for task requests of MU 5, the required function f_1 is not available on server 3. Therefore, server 3 downloads function f_1 from the cloud to complete the task request of MU 5.

This paper focuses on resource allocation and pricing in an SEC system with multiple MUs that compete for server resources. It considers server resources that vary over time due to their spatiotemporal workload and energy arrivals. Hence, there needs to be an incentive mechanism to allocate resources on servers so that an MU with a limited budget can buy the resources it needs, and at the same time, appropriate incentives need to be given to the server to provide services to the MUs. Henceforth, we make the following contributions:

1. We consider a pricing problem between multiple servers and MUs. We model the interaction between servers and MUs as a Stackelberg game where MUs are followers. These MUs determine their own demand strategy, which represents the number of resources required by the MUs for each server. Servers are leaders that set their price according to the response of the MUs in order to maximize their profit.

2. We decompose the resource allocation and pricing problem between multiple servers and MUs into a group of subproblems, in which leaders can determine the price of their resources, and followers can choose appropriate demand strategies according to the leader's price. We have proved the existence of game equilibrium. In addition, we have developed an iterative algorithm to find the equilibrium price of the server. The equilibrium solution of all the above subproblems constitutes the equilibrium solution of the original problem.
3. We evaluate the Stackelberg equilibrium price under various energy consumption costs and study the impact of SEC prices on MU's demand strategy.

In the next section, we review prior works. Then, Section 3 formalizes our system. After that, Section 4 models our problem as a Stackelberg game. In Section 5, we introduce an iterative algorithm. Section 6 shows our evaluation, which is followed by the conclusions in Section 7.

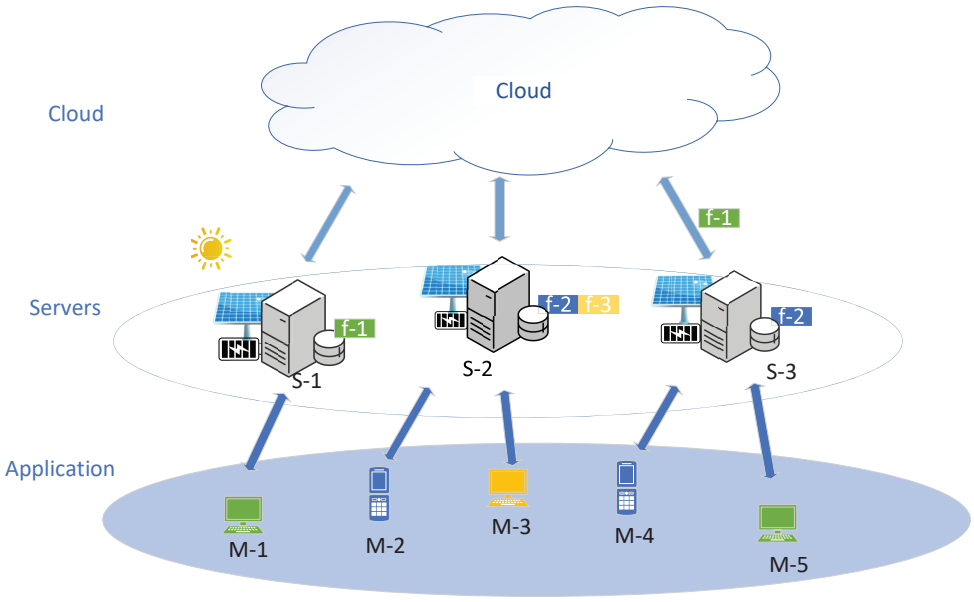

Figure 1. An example serverless edge system with multiple energy harvesting servers and MUs.

2. Related Works

Our research overlaps with (i) MEC works with energy harvesting (EH) servers, (ii) task scheduling in an SEC system, and (iii) pricing strategies in MEC or SEC systems. As it will become clear in our discussion, prior works either do not consider EH servers, SEC or function-as-a-service (FaaS), or/and pricing between MU, EH servers, and the cloud.

To date, many works have considered edge computing with EH servers, see [20–22] and the references therein. In general, their research aims are to develop algorithms/strategies to collaboratively make use of renewable energy. For example, the work in [23] considered the collaboration between EH servers when executing tasks. Their goal was to minimize the cost of using the cloud and energy purchased from the grid. The work in [24] developed an algorithm to predict energy and task arrivals to optimize the number of virtual machines instantiated at an EH server. The authors of [25] developed an algorithm to dispatch requests from end devices to EH servers operating in a fog computing system. Reference [12] considered optimizing the amount of offloaded workload by end users and the number of active servers. Similarly, in [21], the authors considered task assignments to EH servers and the migration of tasks to better exploit the different energy levels of servers. The main distinctions to our work are as follows. First, works such as [23,24] are focused on minimizing grid usage to supplement the energy of servers. Furthermore, these works do not consider SEC. The work in [26] considered SEC, where a controller orchestrates task

assignments and execution of functions at EH devices. In contrast, we consider a pricing problem concerning MUs, EH servers, and the cloud.

A number of works have considered task scheduling in SEC systems. The authors of [27] considered dispatching requests from IoT devices to a pool of servers instead of having devices assigned to a specific server. Similarly, the work in [28] considered the decentralized routing of requests to edge servers, where an access router is responsible for determining the best executioner or edge server of a request. In [29], the authors considered the routing of requests to minimize the number of function cold starts. On the other hand, in [30], servers used an auction algorithm to determine whether to store and run functions and also to decide whether to run a request. In [31,32], the authors considered embedding a directed acyclic graph (DAG), where functions represent nodes, and links denote dependencies. They also considered the routing of traffic between functions. In [32], the authors considered servers with random processing times and time-varying channel conditions from a user to a server. Many works have considered allocating resources for containers that run tasks. For example, the work in [33] used reinforcement learning (RL) to adapt resources over time. Similarly, in [34], a multiagent RL approach was used by edge servers, with each managed by a different owner and having different computing resources, to compete for tasks from IoT devices. On the other hand, the work in [35] determined functions that should be kept *warm* and their resources.

Different from the aforementioned works, in [26], the authors considered a scheduler that orchestrates the placement of functions, allocates required resources, and routes requests to EH edge nodes. All of the aforementioned works do not consider EH servers. Furthermore, except for [32,34], all the works do not consider the energy consumption of devices or/and servers. Lastly, they do not consider a pricing problem.

There are very limited works on pricing in SEC systems. Note that providers such as Amazon employ a pay-per-use model where users are charged based on function usage and allocated resources. In [36], Gupta et al. introduced a pricing scheme that helps a provider schedule jobs according to a user utility that relates to delay. In [37], an SEC provider may rent edge servers to help deliver better quality of service to its clients. We note that there are some works that consider MEC systems. For example, in [38–40], the problem is to determine a pricing strategy for servers that sell their resources to end users who then determine how much server resources to purchase. These works, however, do not consider EH servers or FaaS.

3. System Model

We consider an SEC system consisting of multiple servers and MUs. Define $\mathcal{K}$ as the set of servers indexed by $k \in \{1, \ldots, |K|\}$. These servers have EH capability and are also connected to mains electricity. The set of MUs is $\mathcal{M}$, which is indexed by $i \in \{1, \ldots, |M|\}$. Let $\mathcal{F}$ denote the set of all functions. The nth function is denoted as f_n; its size is α_n. The cloud has all functions in $\mathcal{F}$. We divide time into T time slots, with each indexed by t. The server and MUs are connected through a wireless channel; we assume uplinks and downlinks are allocated an orthogonal channel. Table 1 summarizes our notations.

Each MU i requires a set of functions denoted as $\mathcal{D}_i$, where $\mathcal{D}_i \subseteq \mathcal{F}$, and we have $\mathcal{D}_i = \{f_1, \ldots, f_{|\mathcal{D}_i|}\}$. Furthermore, MU i has a budget of B_i. Define Γ_k as the subset of functions maintained on server k, where $\Gamma_k \subset \mathcal{F}$. Moreover, we have $\Gamma_k = \{f_1, \ldots, f_{|\Gamma_k|}\}$. We use $\gamma_k = |\Gamma_k|$ to denote the number of functions at server k. When server k requires one or more functions that are not in Γ_k, it downloads them from the cloud. Note, there is no collaboration between servers.

Server k advertises price p_k for its services. This price is a function of the cost associated with the energy, the computation, and the budget of MUs. More specifically, servers are charged at a cost of p_t to purchase *energy* from the grid when it experiences an energy outage. The *computation* cost of a server corresponds to the cost to download functions from the cloud. Specifically, for any server k, if MU i demands a function that is not included in

Γ_k, i.e., $\Gamma_k \cap \mathcal{D}_i \neq \emptyset$, then server k downloads functions $\mathcal{D}_i \backslash \Gamma_k$ from the cloud. The per unit price of a function charged by the cloud is p_c.

Table 1. List of notations.

1.	**Set**
$\mathcal{M}$	The set of users
$\mathcal{K}$	The set of servers
$\mathcal{F}$	The set of all functions
Γ_k	The set of functions maintained on server k
$\mathcal{D}_i$	The set of functions needed by MU i
T	The set of time slot
W	Area of server's solar panels
2.	**Variables**
ω_k	CPU frequency of server k
B_i	The budget of MU i
p_k	The unit price of one function on server k
p_c	The unit price of one function on cloud
p_t	Electricity price at time slot t
$E_{i,k}$	Energy consumed by server k to execute MU i demands
$r_{i,k}$	Transmission rate from server k to MU i
$\hat{\rho}$	Transmission power of server k
$E^r_{i,k}$	Energy for transmitting results from server k to MU i
t^r_i	The size of output of the task from MU i

3.1. Energy Harvesting

Each server is equipped with a solar panel that has size W cm^2 and a rechargeable battery. We use b_k^{max} to denote the maximum energy at the server k, while b_k^t represents the energy stored by server k at the beginning of each time slot t. We use b_k^{max} to denote the maximum energy at the server k. Then the energy level at the server k in time slot t is [26]

$$b_k^t = \min(b_k^{max}, \max(0, b_k^{t-1} + g_k^t - \sum_{i \in \mathcal{M}} E^o_{i,k})). \tag{1}$$

where $\sum_{i \in \mathcal{M}} E^o_{i,k}$, and g_k^t denote the consumed and harvested energy and harvesting of server k at time slot t, respectively. In Section 3.2, we detail the energy consumption $\sum_{i \in \mathcal{M}} E^o_{i,k}$.

3.2. Energy Consumption

The energy consumption of a server consists of three parts: (i) tasks computation, (ii) communication of results to a MU, and (iii) download the required function modules from the cloud. Let $\omega_{i,k}$ indicate the computing resources allocated by server k to MU i, and the maximum computing resource on server k is ω_k. The energy incurred by server k to process functions requirement from MU i is [26]

$$E_{i,k} = \kappa \frac{\omega_{i,k}}{\omega_k}, \tag{2}$$

where κ is the constant power consumption of a function module calculated by the server k. Let $E^r_{i,k}$ be the energy required by server k to send results to MU i, and t^r_i is the size of the result. We assume all servers have consistent transmission power $\hat{\rho}$. Then, we can write $E^r_{i,k}$ as

$$E^r_{i,k} = \hat{\rho} \frac{t^r_i}{r_{i,k}}, \tag{3}$$

where $r_{i,k}$ is the data transmission rate. We use E_{k,f_i} to denote the energy required by the server k to download function f_i from the cloud. Therefore, the total energy consumption of server k to execute tasks from MU i is

$$E^o_{i,k} = E_{i,k} + E^r_{i,k} + E_{k,f_i}. \tag{4}$$

In practice, the servers usually communicate with the cloud via fiber links with fast transmission rates and low transmission power. Hence, in this paper, we ignore the energy consumption for downloading functions and assume $E_{k,f_i} = 0$ in Equation (4).

4. Problem Formulation

Our aim is to set the price advertised by servers to maximize the overall revenue of servers and ensure that the MUs execute their tasks. Briefly, servers seek to maximize their revenue by setting a suitable price, while the MUs determine their own demands based on the price of the server and their budgets.

To this end, we model this pricing problem as a Stackelberg game [41]. A Stackelberg game is a strategic interaction between two players: a leader and a follower. The leader first makes decisions that consider the potential reactions of the follower. Then the follower makes their decisions based on the leader's decision. In our case, servers are leaders, where they determine the price of each function and advertise the price to MUs. On the other hand, the MUs are followers that determine their own demands based on the price of servers.

In this section, we establish a mathematical model to analyze the details of the Stackelberg game. By solving this Stackelberg game, we can obtain the optimal price for the server, as well as the optimal demands decisions for the MUs. These help maximize the benefits of the entire system.

4.1. Follower Strategy

Let $x_{i,k}$ denote the number of functions that MU i requires from server k. Initially, we have $x_{i,k} = |\mathcal{D}_i|$. We apply the utility function introduced in [42] for a fair resource allocation, where

$$U_i = B_i \sum_{k \in \mathcal{K}} \log(\sigma_i + x_{i,k}), \quad \forall k \in \mathcal{K}. \tag{5}$$

Here, σ_i is a constant, and $\sigma_i = 1$ [42]. The aim of each MU i is to maximize U_i by adjusting the number of function modules $x_{i,k}$ required from the server k. We thus have the following model

$$\max \quad U_i \tag{6}$$

$$\begin{aligned} \text{s.t.} \quad & \sum_{k \in \mathcal{K}} x_{i,k} p_k \leq B_i \\ & x_{i,k} \geq 0, \quad \forall k \in \mathcal{K}. \end{aligned} \tag{7}$$

The constraint (7) ensures that the total cost incurred by MU i does not exceed its budget B_i and that the number of function modules required by MU i is non-negative. Recall that the demands of MUs are independent and not in conflict with each other. We first discuss a scenario with N MUs and two servers. We then release this to a general scenario with $|\mathcal{K}|$ servers.

4.1.1. Two Servers

When there are N MUs and two servers, we have the following subproblem and rewrite problem (6) as

$$\max_{\{x_{i,1}, x_{i,2}\}} \quad U_i \tag{8}$$

$$\begin{aligned} \text{s.t.} \quad & \sum_{k=1}^{2} x_{i,k} p_k \leq B_i \\ & x_{i,1}, x_{i,2} \geq 0. \end{aligned} \tag{9}$$

In this subproblem, $x_{i,1}$ and $x_{i,2}$ represent the number of function modules required by MU i on two serverless edge servers. The objective is to maximize MU i's utility function U_i in Equation (8) while ensuring that the total cost incurred by MU i does not exceed its budget B_i. The constraints also include that the number of function modules required by MU i is non-negative; see constraint (9).

In order to find the optimal solution to the problem defined by Equations (8) and (9), we adopt the Karush–Kuhn–Tucker (KKT) conditions. KKT conditions are a method for assessing optimality in constrained optimization problems, thus employing Lagrange multipliers to handle constraints. Using Lagrange's multipliers λ_1, λ_2, and λ_3 for constraint Equation (9), we have

$$L_M = B_i \sum_{f \in \mathcal{F}} \log(\sigma_i + x_{i,k}) + \lambda_1 (B_i - \sum_{k=1}^{2} x_{i,k} p_k) + \lambda_2 x_{i,1} + \lambda_3 x_{i,2}. \tag{10}$$

The KKT conditions of the problem in Equations (8) and (9) are presented as follows:

$$\frac{\partial L_M}{\partial x_{i,1}} = \frac{B_i}{\sigma_i + x_{i,1}} - \lambda_1 p_1 + \lambda_2 = 0 \tag{11}$$

$$\frac{\partial L_M}{\partial x_{i,2}} = \frac{B_i}{\sigma_i + x_{i,2}} - \lambda_1 p_2 + \lambda_3 = 0 \tag{12}$$

$$\lambda_1 (B_i - \sum_{k=1}^{2} x_{i,k} p_k) = 0 \tag{13}$$

$$\lambda_2 x_{i,1} = 0 \tag{14}$$

$$\lambda_3 x_{i,2} = 0 \tag{15}$$

$$\lambda_1 > 0, \lambda_2, \lambda_3, x_{i,1}, x_{i,2} \geq 0. \tag{16}$$

The optimal solution of Equations (8) and (9) can be categorized into four cases, with each corresponding to a specific set of variable values. Specifically, we have the following:

(1) Case 1: $x_{i,1} > 0$, and $x_{i,2} > 0$. In this case, MU i requires functions from both servers. Therefore, we have $\lambda_2 = \lambda_3 = 0$ and, substituting into (11) and (12), we obtain

$$x_{i,k} = \frac{B_i}{\lambda_1 p_k} - \sigma_i, \ \forall i \in \mathcal{M}, \ k = 1, 2, \tag{17}$$

By substituting (17) into (13), we have the final demand from both servers as

$$x_{i,k} = \frac{B_i + \sigma_i \sum_{k=1}^{2} p_k}{2 p_k} - \sigma_i, k = 1, 2. \tag{18}$$

(2) Case 2: $x_{i,1} > 0$, and $x_{i,2} = 0$. In this case, MU i requires functions from server 1. Therefore, we have $\lambda_2 = 0$ and, substituting into (11) and (12), we have

$$x_{i,1} = \frac{B_i}{\lambda_1 p_1} - \sigma_i, \tag{19}$$

Substituting $x_{i,1}$ into (13), we obtain

$$\frac{1}{\lambda_1} = \frac{B_i + p_1 \sigma_i}{B_i}, \tag{20}$$

Using (20) in (19), we have the number of required functions from server 1 as

$$x_{i,1} = \frac{B_i + \sigma_i(p_1 + p_2)}{2p_1} - \sigma_i. \tag{21}$$

(3) Case 3: $x_{i,1} = 0$, and $x_{i,2} > 0$. Similar to case 2, in this case, MU i requires functions from server 2. We have its demand as

$$x_{i,2} = \frac{B_i + \sigma_i(p_1 + p_2)}{2p_2} - \sigma_i. \tag{22}$$

(4) Case 4: $x_{i,1} = 0$, and $x_{i,2} = 0$. In this case, MU i does not require functions from any server. However, if $B_i > 0$, then $\lambda_1 = 0$, which does not satisfy Equation (16). Therefore, when $x_{i,1} = x_{i,2} = 0$, we have $B_i = 0$.

4.1.2. $|\mathcal{K}|$ Servers

Thus, using (18), (21), and (22), we can formulate the demand of N MUs and K servers as

$$x_{i,k} = \frac{B_i + \sigma_i \sum_{k \in \mathcal{K}} p_k}{K p_k} - \sigma_i. \tag{23}$$

4.2. Leader Strategy

The previous section analyzed how MUs formulate their demand strategies based on the given price p_k by the server. This subsection considers how to maximize the utility of serverless edge servers. As the leader in the Stackelberg game, servers profit by providing services to MUs. To maximize utility, servers need to set appropriate prices for the services they provide. Since MUs tend to acquire resources from servers with lower prices, servers operate in a noncooperative and competitive relationship with each other. The utility function of server K can be expressed as

$$U_k(p_k, p_{-k}) = p_k \sum_{i \in \mathcal{M}} x_{i,k}. \tag{24}$$

where p_{-k} is the price matrix of the other SEC server except server k. Server k aims to select a proper price vector p_k to maximize its utility. Therefore, server k's utility maximization problem can be expressed as

$$\max \quad U_k(p_k, p_{-k}). \tag{25}$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{K}} x_{i,k} \leq \gamma_k \qquad p_k \geq 0, \quad \forall k \in \mathcal{K}. \tag{26}$$

The constraints in (26) ensure that the number of functions that server k can provide to all MUs does not exceed γ_k. They also ensure that the price of server k, i.e., p_k, must be non-negative. Because the price must be a non-negative, then by substituting Equation (23) into the problem defined by Equation (25), this yields

$$\max \quad U_k(p_k, p_{-k}) = p_k \sum_{i \in \mathcal{M}} x_{i,k} \tag{27}$$

$$s.t \quad \sum_{k \in \mathcal{K}} \frac{B_i + \sigma_i \sum_{k \in \mathcal{K}} p_k}{K p_k} - \sigma_i \leq \gamma_k \qquad p_k \geq 0, k \in \mathcal{K}, \tag{28}$$

The objective function Equation (27) can be expressed as

$$U_k(p_j, p_{-j}) \quad = p_k \sum_{i \in \mathcal{M}} \frac{B_i + \sigma_i \sum_{k \in \mathcal{K}} p_k}{K p_k} - \sigma_i, \tag{29}$$

If the MUs do not require any function, i.e., $x_{i,k} = 0$, Equation (23) implies that

$$B_i + \sigma_i \sum_{g \in \mathcal{K}, g \neq k} p_g \geq \sigma_i (K-1) p_k, \tag{30}$$

Therefore, in order to achieve $x_{i,k} > 0$ for server k, its price must satisfy

$$p_k \leq \left(\frac{B_i + \sigma_i (\sum_{g \in \mathcal{K}, g \neq k} p_g)}{\sigma_i (K-1)} \right). \tag{31}$$

Equation (31) ensures that the server's pricing strategy considers the influence of other servers while meeting the demand of the MUs.

4.3. The Existence of Stackelberg Equilibrium

We now prove that the Nash equilibrium of the price optimization game between servers and MUs always exists and is unique. By letting Y_k be the set of all values that satisfy Equation (31), we have the following theorem.

Theorem 1. *In the Stackelberg game between servers and MUs, there is a unique Nash equilibrium.*

Proof. (1) In the Stackelberg game, the strategy space of all players is product space $\mathrm{Y} = \mathrm{Y}_1 \times \mathrm{Y}_2 \ldots \mathrm{Y}_k$, where $p_k \in \mathrm{Y}_k$, and $k \in \mathcal{K}$. Thus, Y is a convex function and is a nonempty subset of a Euclidean space R^K.

(2) U_k is continuous in p_k. The second derivative of U_k to p_k is expressed as

$$\frac{\partial^2 U_K}{\partial p_k} = 0, \forall k \in \mathcal{K}. \tag{32}$$

Therefore, U_k is concave in p_k. Thus, Nash equilibrium exists in our Stackelberg game. □

5. Iterative Price Update Algorithm

To achieve the equilibrium state of the game outlined above, we propose an iterative pricing algorithm. Initially, each server randomly selects a price from the range computed by using Equation (31). The server's revenue is derived from the total price paid by the MUs minus its cost. The cost encompasses the energy expense and purchasing function modules from the cloud. In particular, the energy consumption expenses correspond to current energy usage, energy acquired from natural sources, and grid electricity tariffs.

If the server selects a price that falls below its cost, it suffers profit losses. Conversely, if the chosen price is excessively high, it may struggle to attract MU consumption. The pseudocode of the iterative price update algorithm is shown in Algorithm 1. Each server updates the unit price in each iteration according to its own energy level and resource pool.

Initially, each server calculates its price p'_k based on the energy stored in its own storage b_k^t at the current time slot t. Let ϑ be a given threshold; if a server whose energy b_k^t surpasses threshold ϑ, the server opts to provide services to MUs using its own stored energy rather than purchasing from the grid. This decision helps in reducing costs. Therefore, the pricing of server k is updated as follows:

$$p'_k = p_k - E^o_{i,k} p_t. \tag{33}$$

After the server updates its price, the MUs also need to update demand using Equation (23). If the resulting $x_{i,k} < |\mathcal{D}_i|$, then MU i discards the last few functions $|\mathcal{D}_i| - x_{i,k}$, because it does not have enough budget to afford them; otherwise, it remains $x_{i,k} = |\mathcal{D}_i|$.

Algorithm 1 Iterative price update algorithm

Require: b_k^t, p_k, $\mathcal{D}_i$, p_c
Ensure: p_k^*, $x'_{i,k}$
1: $x_{i,k} = |\mathcal{D}_i|$
2: **if** $b_k^t \geq \vartheta$ **then**
3: Reduce the price of server k using Equation (33), obtain p'_k
4: Calculate demands $x'_{i,k}$ using Equation (23)
5: Update demands $\mathcal{D}'_i$
6: **end if**
7: **if** $\mathcal{D}_i \subseteq \Gamma_k$ **then**
8: Retain the price $p_k^* = p'_k$ of server k
9: **else**
10: Increase the price of server k using Equation (34), obtain p_k^*
11: Calculate demands $x'_{i,k}$ using Equation (23)
12: Update the demands of $\mathcal{D}'_i$
13: **end if**
14: Return p_k^*

Next, server k checks demands from MUs, i.e., $\mathcal{D}_i$, and its resource pool Γ_k. If $\mathcal{D}_i \subseteq \Gamma_k$, this means server k already has all the required function modules within its resource pool to fulfill the MU's requirements. Then, the server retains the price p'_k and advertises to MUs. Otherwise, the server needs to download functions from the cloud, and thus the price p'_k increases. In particular, the updated price p_k^* is calculated by

$$p_k^* = p'_k + (|\mathcal{D}_i/\Gamma_k|)p_c \tag{34}$$

Finally, all MUs determine their demand strategies $\{x_{i,k}\}$ based on the prices $\{p_k\}$ provided by servers, as per Equation (23). These strategies encompass each MU's resource purchasing decisions regarding all servers. In subsequent time slots, all servers continually update their prices iteratively based on the MUs' demands and their own energy.

We conclude this section with the runtime complexity of the iterative price update algorithm. To calculate the initialization price p'_k and MU demand strategy $x'_{i,k}$ based on the energy level of each server, i.e., lines 3–5, the time complexity is $O(K)$. In lines 7–12, the algorithm updates the price of the server p^*k and calculates the demand strategy $x'i,k$ for the MUs. Recall that the size of the resource pool of each server is $|\Gamma_k|$, and the time complexity of each iteration is $O(K|\Gamma_k|)$. The convergence of the algorithm is related to the number of iterations. In this paper, we denote the number of iterations as N. Therefore, the total running time complexity of the iterative price update algorithm is $O(K|\Gamma_k|N)$.

6. Simulations

In this section, we evaluate how MUs choose the best demand strategy according to the given price and how the serverless edge server determines a price that maximizes its revenue according to the MU's budget and its own cost. As far as we know, there are few studies on the resource allocation and pricing of serverless edge computing [37,43]. However, due to differences in background information and optimization objectives, it would be inappropriate to directly compare the method proposed in this paper with previous work. Our simulations were conducted using MATLAB 2021 on an AMD Ryzen 7 CPU @ 2.30 GHz with an 8 GB RAM laptop.

In the experiments, we assumed that there were three servers and five MUs. Firstly, we considered five different MUs, each with a different budget and required functions. This was to simulate the differences between different MUs and also to evaluate the scenarios where the server needs to handle different demands. Similar with the works in [43–45], where they predefine the budget of MUs, we also fixed the budget of each MU as $B_1 = 5$, $B_2 = 10$, $B_3 = 14$, $B_4 = 18$, and $B_5 = 22$. On the other hand, we assumed that there were three

servers, each with different computing resources and capabilities. The computing resources were represented by $\gamma_1 = 10$, $\gamma_2 = 15$, and $\gamma_3 = 20$. Additionally, the servers' computing capabilities were denoted by $\omega_1 = 0.5$ GHz, $\omega_2 = 2$ GHz, and $\omega_3 = 1.5$ GHz. Furthermore, we set the transmission power of the three servers to be $\hat{\rho} = 5$ W and the transmission rate from server to MU to be $r_{i,k} = 5$ Kbps. Additionally, we set parameters $\kappa = 0.8$, $\vartheta = 15$, and $h_t = 2$ per Wh. Note that, as shown in [46–48], in practice, the servers are usually statically deployed. The parameters of the server are fixed and can be predetermined.

In this section, we first demonstrate the impact of server prices on MU utility and the countereffect of MU budgets on server prices. Next, we assume that servers are equipped with natural energy harvesting devices such as solar panels, and the energy collection from solar panels is from real solar radiation data [49].

6.1. Stackelberg Game

In Figures 2–5, we demonstrate the impact of an MU's budget on the entire system by simply changing the budget B_1 for MU 1. Figure 2 shows the optimal prices set by servers when the budget of MU 1 changed from 5 to 40. We see from Figure 2 that the pricing of a server is closely related to the number of resources in its own resource pool. A server with more cached function modules incurs lower fees, and vice versa. For example, S_1 charged higher than S_2, and S_2 charged higher than S_3.

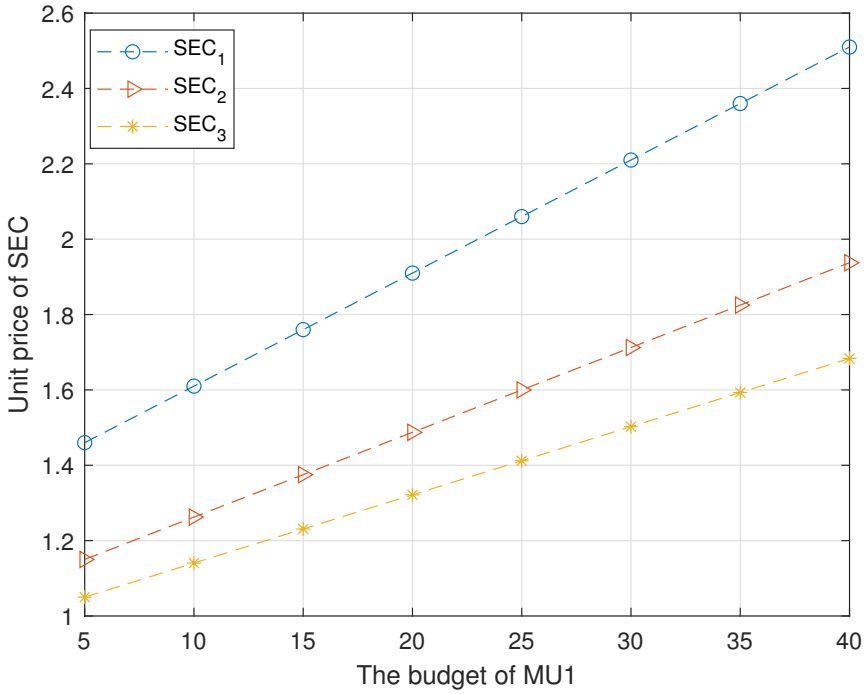

Figure 2. Server's price at equilibrium.

Figure 3 shows the demand of each MU when Nash equilibrium has been achieved. The demand of MU_1 was increased because the budget B_1 increased. On the contrary, the other MU's demand decreased because the prices set by the server increased. This reflects the sensitivity of MUs to resource purchase decisions and their behavior of adjusting demand strategies based on server pricing. Figure 4 depicts the MU's utilities at Nash equilibrium and is related to Figure 3.

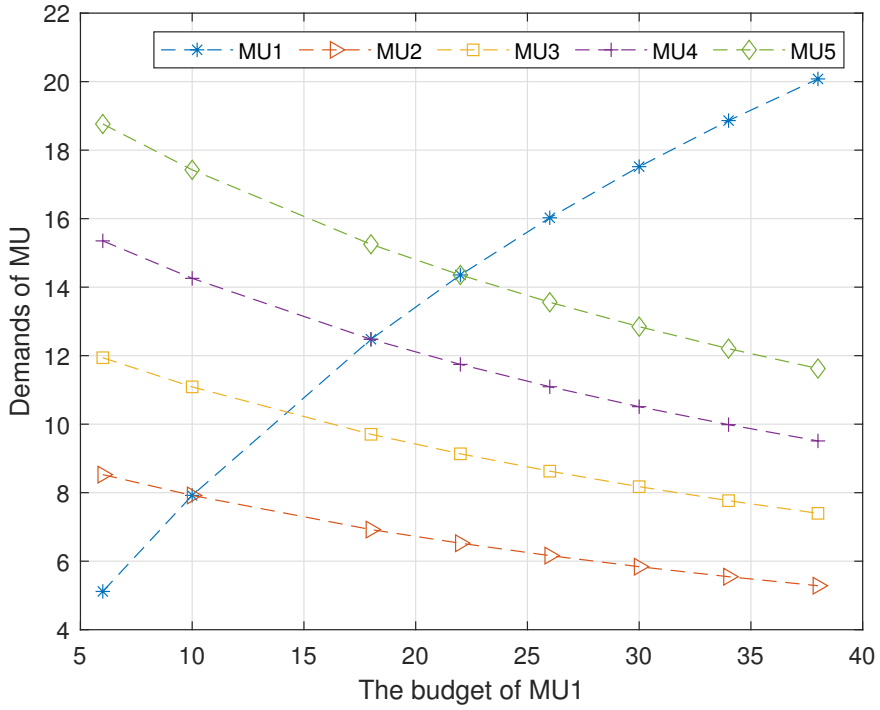

Figure 3. MUs' demands at equilibrium.

Figure 5 shows the revenue of each server at Nash equilibrium. It is worth noting that although S_3 set the lowest price, its revenue was the highest, which was 95% higher than S_1. This is because it cached the most number of function modules.

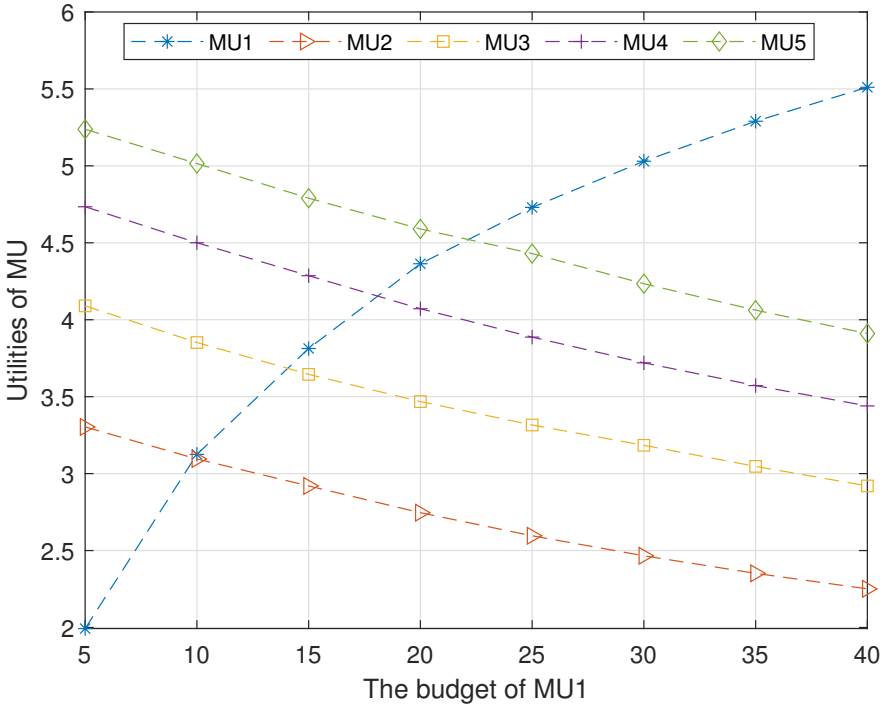

Figure 4. MUs' utilities at equilibrium.

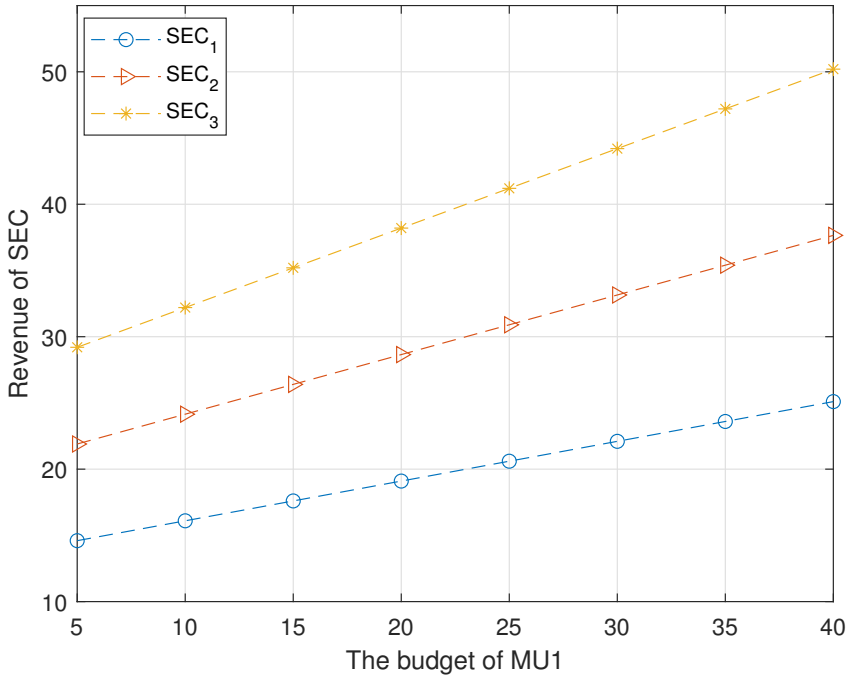

Figure 5. Servers' revenues at equilibrium.

6.2. Impact of Energy Harvesting on Price

Figure 6 shows the harvested energy of the three servers in 24 h according to the solar radiation [49]. The server obtains energy through solar panels during the day and stores the remaining energy in the battery.

Figure 7 shows the residual energy at each server and their price. We see that the harvested energy at server S_1 was not sufficient for its operation, i.e., its battery capacity was near zero for all day. In addition, server S_1 cached the minimum number of function modules. Therefore, the price of S_1 was higher than both S_2 and S_3. On the other hand, during the daytime when solar radiation intensity is high, server S_3 had a lower price than S_2. This is because when the energy was sufficient, i.e., both servers S_2 and S_3 did not need to purchase from the grid, the server that cached a lower number of function modules set a higher price for purchasing from the function from the cloud. We also see that the prices of S_2 and S_3 fluctuated during the daytime. This is because the server has a lower price in the daytime; hence, there are more demands from MUs, which in turn increases the number of function modules purchased from the cloud.

In conclusion, while our experiments comprehensively unveil the influence of the Stackelberg games on resource allocation and pricing within serverless edge computing, there are limitations. For instance, our model assumed fixed parameters for the transmission power and rate, as well as constant power consumption, which may not fully capture the dynamic nature of real-world scenarios influenced by environmental conditions, hardware constraints, and communication protocols. Furthermore, the instantaneous steady state assumption simplifies the model but may oversimplify actual system behavior.

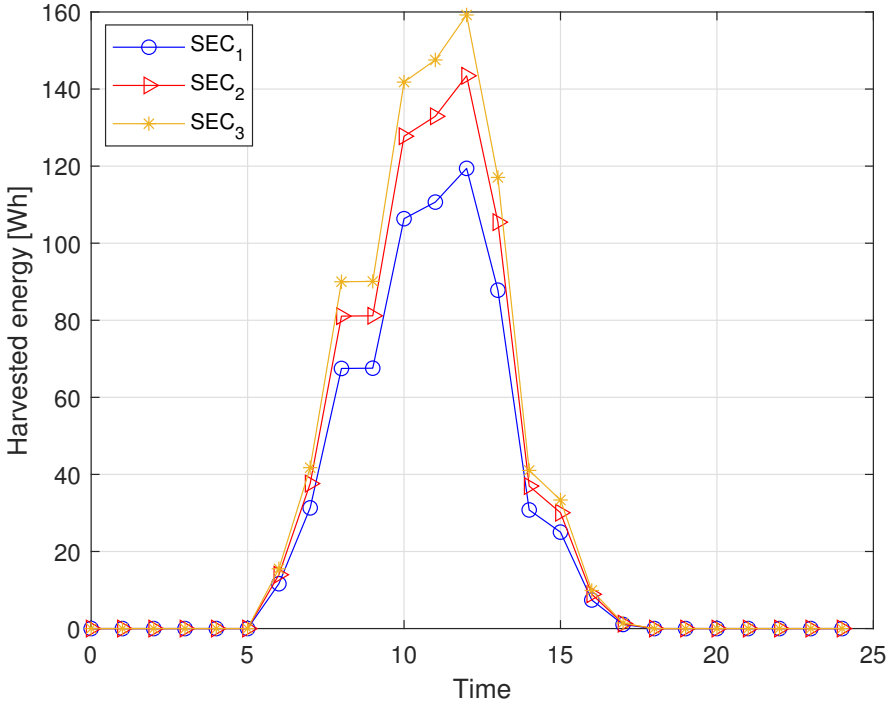

Figure 6. Energy collected by server in 24 h.

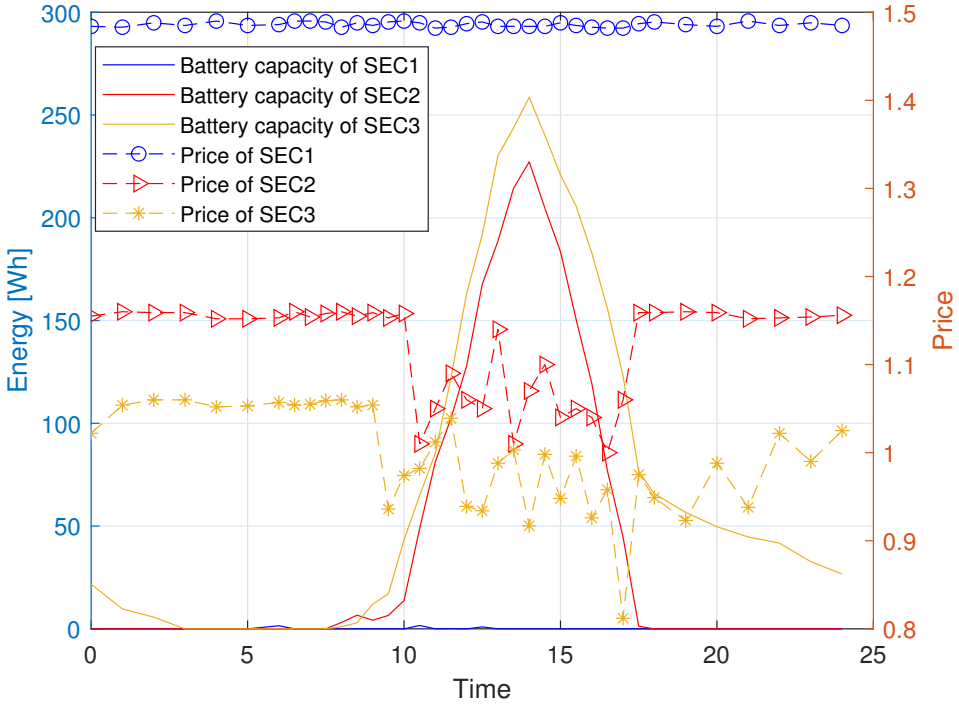

Figure 7. Residual energy and price in 24 h.

7. Conclusions

In this paper, we studied the resource allocation and pricing between MUs and energy harvesting servers. We formulated the problem as a Stackelberg game and presented the pricing tradeoff strategy. In addition, we considered exploiting renewable energy on servers and calculated the optimal pricing via a proposed iterative algorithm. The experimental results show that the proposed mechanism is effective.

In future work, we will prioritize the refinement of adaptive models that can dynamically adjust to the ever-changing conditions of real-world scenarios. This involves conducting thorough and comprehensive comparisons with existing methods to discern the relative strengths and weaknesses of our proposed approach. Moreover, we recognize the significance of conducting sensitivity analyses to gauge the reliability and robustness of our algorithm under a spectrum of varying conditions. In forthcoming research, we plan to delve deeper into assessing the impact of various factors, including server capacity, server load, MU demand, energy supply stability, and MU density, on the performance of our algorithm. By scrutinizing these factors, we aim to ascertain the extent to which our algorithm can maintain its reliability and robustness across diverse operational environments.

Author Contributions: Writing—original draft, Y.L.; Writing—review & editing, C.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

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

References

1. Voorsluys, W.; Broberg, J.; Buyya, R. Introduction to cloud computing. In *Cloud Computing: Principles and Paradigms*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2011; pp. 1–41.
2. Naha, R.K.; Garg, S.; Georgakopoulos, D.; Jayaraman, P.P.; Gao, L.; Xiang, Y.; Ranjan, R. Fog computing: Survey of trends, architectures, requirements, and research directions. *IEEE Access* **2018**, *6*, 47980–48009. [CrossRef]
3. Moghaddam, F.F.; Ahmadi, M.; Sarvari, S.; Eslami, M.; Golkar, A. Cloud computing challenges and opportunities: A survey. In Proceedings of the 2015 1st International Conference on Telematics and Future Generation Networks (TAFGEN), Kuala Lumpur, Malaysia, 26–28 May 2015; IEEE: New York, NY, USA, 2015; pp. 34–38.
4. Corcoran, P.; Datta, S.K. Mobile-Edge Computing and the Internet of Things for Consumers: Extending cloud computing and services to the edge of the network. *IEEE Consum. Electron. Mag.* **2016**, *5*, 73–74. [CrossRef]
5. Liu, Y.; Han, Y.; Zhang, A.; Xia, X.; Chen, F.; Zhang, M.; He, Q. QoE-aware Data Caching Optimization with Budget in Edge Computing. In Proceedings of the IEEE International Conference on Web Services (ICWS), Virtual, 5–10 September 2021; pp. 324–334.
6. Taleb, T.; Samdanis, K.; Mada, B.; Flinck, H.; Dutta, S.; Sabella, D. On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1657–1681. [CrossRef]
7. Gambín, Á.F.; Rossi, M. A sharing framework for energy and computing resources in multi-operator mobile networks. *IEEE Trans. Netw. Serv. Manag.* **2019**, *17*, 1140–1152. [CrossRef]
8. Belkhir, L.; Elmeligi, A. Assessing ICT global emissions footprint: Trends to 2040 and recommendations. *J. Clean. Prod.* **2018**, *177*, 448–463. [CrossRef]
9. Clemm, A.; Westphal, C. Challenges and Opportunities in Green Networking. In Proceedings of the IEEE 8th International Conference on Network Softwarization (NetSoft), Milan, Italy, 27 June–1 July 2022; pp. 43–48.
10. Hu, S.; Chen, X.; Ni, W.; Wang, X.; Hossain, E. Modeling and Analysis of Energy Harvesting and Smart Grid-Powered Wireless Communication Networks: A Contemporary Survey. *IEEE Trans. Green Commun. Netw.* **2020**, *4*, 461–496. [CrossRef]
11. Jiang, C.; Fan, T.; Gao, H.; Shi, W.; Liu, L.; Cerin, C.; Wan, J. Energy Aware Edge Computing: A Survey. *Comput. Commun.* **2020**, *151*, 556–580. [CrossRef]
12. Xu, J.; Chen, L.; Ren, S. Online Learning for Offloading and Autoscaling in Energy Harvesting Mobile Edge Computing. *IEEE Trans. Cog. Commun. Netw.* **2017**, *3*, 361–373. [CrossRef]
13. Cecchinato, D.; Berno, M.; Esposito, F.; Rossi, M. Allocation of Computing Tasks In Distributed MEC Servers Co-Powered By Renewable Sources And The Power Grid. In Proceedings of the IEEE ICASSP, Barcelona, Spain, 4–8 May 2020; pp. 8971–8975.
14. Baresi, L.; Mendonça, D.F. Towards a serverless platform for edge computing. In Proceedings of the IEEE International Conference on Fog Computing (ICFC), Prague, Czech Republic, 24–26 June 2019; pp. 1–10.
15. McGrath, G.; Brenner, P.R. Serverless computing: Design, implementation, and performance. In Proceedings of the IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), Atlanta, GA, USA, 5–8 June 2017; pp. 405–410.

16. Kjorveziroski, V.; Filiposka, S.; Trajkovik, V. Iot serverless computing at the edge: A systematic mapping review. *Computers* **2021**, *10*, 130. [CrossRef]
17. Liu, S.; Liu, L.; Tang, J.; Yu, B.; Wang, Y.; Shi, W. Edge Computing for Autonomous Driving: Opportunities and Challenges. *Proc. IEEE* **2019**, *107*, 1697–1716. [CrossRef]
18. Ahmed, E.; Rehmani, M.H. Mobile edge computing: Opportunities, solutions, and challenges. *Future Gener. Comput. Syst.* **2017**, *70*, 59–63. [CrossRef]
19. Li, Y.; Liu, J.; Jiang, B.; Yang, C.; Wang, Q. Cost Minimization in Serverless Computing with Energy Harvesting SECs. In Proceedings of the 2023 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), Beijing, China, 14–16 June 2023; pp. 1–7. [CrossRef]
20. Patros, P.; Spillner, J.; Papadopoulos, A.V.; Varghese, B.; Rana, O.; Dustdar, S. Toward Sustainable Serverless Computing. *IEEE Internet Comput.* **2021**, *25*, 42–50. [CrossRef]
21. Gu, L.; Cai, J.; Zeng, D.; Zhang, Y.; Jin, H.; Dai, W. Energy efficient task allocation and energy scheduling in green energy powered edge computing. *Future Gener. Comput. Syst.* **2019**, *95*, 89–99. [CrossRef]
22. Li, W.; Yang, T.; Delicato, F.C.; Pires, P.F.; Tari, Z.; Khan, S.U.; Zomaya, A.Y. On Enabling Sustainable Edge Computing with Renewable Energy Resources. *IEEE Commun. Mag.* **2018**, *56*, 94–101. [CrossRef]
23. Zhang, J.; Zhang, B.; Liu, J.; Han, Z. A Joint Offloading and Energy Cooperation Scheme for Edge Computing Networks. In Proceedings of the IEEE ICC, Seoul, Republic of Korea, 16–20 May 2022; pp. 5537–5542.
24. Perin, G.; Berno, M.; Erseghe, T.; Rossi, M. Towards Sustainable Edge Computing Through Renewable Energy Resources and Online, Distributed and Predictive Scheduling. *IEEE Trans. Netw. Serv. Manag.* **2022**, *19*, 306–321. [CrossRef]
25. Karimiafshar, A.; Hashemi, M.R.; Toosi, A.N. A request dispatching method for efficient use of renewable energy in fog computing environments. *Future Gener. Comput. Syst.* **2021**, *114*, 631–646. [CrossRef]
26. Aslanpour, M.S.; Toosi, A.N.; Cheema, M.A.; Gaire, R. Energy-aware resource scheduling for serverless edge computing. In Proceedings of the 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid), Taormina, Italy, 16–19 May 2022; pp. 190–199.
27. Cicconetti, C.; Conti, M.; Passarella, A. Uncoordinated access to serverless computing in MEC systems for IoT. *Comput. Netw.* **2020**, *172*, 107184–107190. [CrossRef]
28. Cicconetti, C.; Conti, M.; Passarella, A. A Decentralized Framework for Serverless Edge Computing in the Internet of Things. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 2166–2180. [CrossRef]
29. Sethunath, M.; Peng, Y. A joint function warm-up and request routing scheme for performing confident serverless computing. *High Confid. Comput.* **2022**, *2*, 100071. [CrossRef]
30. Bermbach, D.; Bader, J.; Hasenburg, J.B.; Pfandzelter, T.; Thamsen, L. AuctionWhisk: Using an auction-inspired approach for function placement in serverless fog platforms. *Wiley J. Softw. Pract. Exp.* **2022**, *52*, 1143–1169. [CrossRef]
31. Deng, S.; Zhao, H.; Xiang, Z.; Zhang, C.; Jiang, R.; Li, Y.; Yin, J.; Dustdar, S.; Zomaya, A.Y. Dependent Function Embedding for Distributed Serverless Edge Computing. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 2346–2357. [CrossRef]
32. Xie, R.; Gu, D.; Tang, Q.; Huang, T.; Yu, F.R. Workflow Scheduling in Serverless Edge Computing for the Industrial Internet of Things: A Learning Approach. *IEEE Trans. Ind. Inform.* **2022**, *19*, 8242–8252. [CrossRef]
33. Benedetti, P.; Femminella, M.; Reali, G.; Steenhaut, K. Reinforcement Learning Applicability for Resource-Based Auto-scaling in Serverless Edge Applications. In Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), Pisa, Italy, 21–25 March 2022; pp. 674–679.
34. Tang, Q.; Xie, R.; Yu, F.R.; Chen, T.; Zhang, R.; Huang, T.; Liu, Y. Distributed Task Scheduling in Serverless Edge Computing Networks for the Internet of Things: A Learning Approach. *IEEE Internet Things J.* **2022**, *9*, 19634–19648. [CrossRef]
35. Ko, H.; Pack, S. Function-Aware Resource Management Framework for Serverless Edge Computing. *IEEE Internet Things J.* **2023**, *10*, 1310–1319. [CrossRef]
36. Gupta, V.; Phade, S.; Courtade, T.; Ramchandran, K. Utility-based resource allocation and pricing for serverless computing. *arXiv* **2020**, arXiv:2008.07793.
37. Xie, R.; Tang, Q.; Qiao, S.; Zhu, H.; Yu, F.R.; Huang, T. When serverless computing meets edge computing: Architecture, challenges, and open issues. *IEEE Wirel. Commun.* **2021**, *28*, 126–133. [CrossRef]
38. He, Z.; Sun, Y.; Feng, Z. Research on Resource Allocation of Autonomous Swarm Robots Based on Game Theory. *Electronics* **2023**, *12*, 4370. [CrossRef]
39. Chen, Y.; Li, Z.; Yang, B.; Nai, K.; Li, K. A Stackelberg game approach to multiple resources allocation and pricing in mobile edge computing. *Future Gener. Comput. Syst.* **2020**, *108*, 273–287. [CrossRef]
40. Xiong, Z.; Feng, S.; Niyato, D.; Wang, P.; Han, Z. Optimal Pricing-Based Edge Computing Resource Management in Mobile Blockchain. In Proceedings of the IEEE ICC, Kansas City, MO, USA, 20–24 May 2018.
41. Basar, T.; Olsder, G.J. Dynamic noncooperative game theory, ser. In *Classics in Applied Mathematics*; SIAM: Philadelphia, PA, USA, 1999.
42. Basar, T.; Srikant, R. Revenue-maximizing pricing and capacity expansion in a many-users regime. In Proceedings of the IEEE INFOCOM, New York, NY, USA, 23–27 June 2002; Volume 1, pp. 294–301.
43. Tütüncüoğlu, F.; Dán, G. Joint Resource Management and Pricing for Task Offloading in Serverless Edge Computing. *IEEE Trans. Mob. Comput.* **2023**, 1–15.

44. Li, K. A game theoretic approach to computation offloading strategy optimization for non-cooperative users in mobile edge computing. *IEEE Trans. Sustain. Comput.* **2018**. [CrossRef]
45. Liu, C.; Li, K.; Liang, J.; Li, K. COOPER-MATCH: Job offloading with a cooperative game for guaranteeing strict deadlines in MEC. *IEEE Trans. Mob. Comput.* **2019**. [CrossRef]
46. Yuan, X.; Xie, Z.; Tan, X. Computation offloading in uav-enabled edge computing: A stackelberg game approach. *Sensors* **2022**, *22*, 3854. [CrossRef] [PubMed]
47. Liu, X.; Zheng, J.; Zhang, M.; Li, Y.; Wang, R.; He, Y. A Game-Based Computing Resource Allocation Scheme of Edge Server in Vehicular Edge Computing Networks Considering Diverse Task Offloading Modes. *Sensors* **2023**, *24*, 69. [CrossRef] [PubMed]
48. Liu, B.; Xu, H.; Zhou, X. Resource allocation in wireless-powered mobile edge computing systems for internet of things applications. *Electronics* **2019**, *8*, 206. [CrossRef]
49. Measurement and Instrumentation Data Center (MIDC). Available online: https://midcdmz.nrel.gov/ (accessed on 4 April 2024).