information

Article

SPMOO: A Multi-Objective Offloading Algorithm for Dependent Tasks in IoT Cloud-Edge-End Collaboration

Liu Liu [1], Haiming Chen [1,2,*] and Zhengtao Xu [3]

[1] Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo 315211, China; Lliu661004@163.com
[2] Zhejiang Provincial Key Laboratory of Mobile Network Application Technology, Ningbo University, Ningbo 315211, China
[3] Chu Kochen Honors College, Zhejiang University, Hangzhou 310063, China; xuzhengtao@zju.edu.cn
* Correspondence: chenhaiming@nbu.edu.cn

Abstract: With the rapid development of the internet of things, there are more and more end devices, such as wearable devices, USVs and intelligent automobiles, connected to the internet. These devices tend to require large amounts of computing resources with stringent latency requirements, which inevitably increases the burden on edge server nodes. Therefore, in order to alleviate the problem that the computing capacity of edge server nodes is limited and cannot meet the computing service requirements of a large number of end devices in the internet of things scenario, we combined the characteristics of rich computing resources of cloud servers and low transmission delay of edge servers to build a hybrid computing task-offloading architecture of cloud-edge-end collaboration. Then, we study offloading based on this architecture for complex dependent tasks generated on end devices. We introduce a two-dimensional offloading decision factor to model latency and energy consumption, and formalize the model as a multi-objective optimization problem with the optimization objective of minimizing the average latency and average energy consumption of the task's computation offloading. Based on this, we propose a multi-objective offloading (SPMOO) algorithm based on an improved strength Pareto evolutionary algorithm (SPEA2) for solving the problem. A large number of experimental results show that the algorithm proposed in this paper has good performance.

Keywords: internet of things (IoT); cloud-edge-end collaboration; task offloading; delay; energy consumption

Citation: Liu, L.; Chen, H.; Xu, Z. SPMOO: A Multi-Objective Offloading Algorithm for Dependent Tasks in IoT Cloud-Edge-End Collaboration. *Information* **2022**, *13*, 75. https://doi.org/10.3390/info13020075

Academic Editor: Corinna Schmitt

Received: 12 January 2022
Accepted: 2 February 2022
Published: 5 February 2022

1. Introduction

With the rapid development of various internet-of-things (IoT) fields, such as intelligent traffic, intelligent home and intelligent manufacturing [1–4], the number of various IoT devices (e.g., wearable devices, unmanned surface vehicles (USVs) and intelligent automobiles [5–7] etc.) has increased significantly. According to Cisco, there are now more than 30 billion mobile IoT devices worldwide, generating about 2.5 EB of data per day, which often requires further processing and analysis. However, IoT devices have limited computing power and storage resources because they are mostly small, battery-powered and equipped with sensors. So, many latency-sensitive computing tasks generated in real time, such as face recognition, virtual reality (VR) and augmented reality (AR) [8–10], are present difficulties in guaranteeing users' real-time experience when executed on local devices. These tasks are usually passed to the cloud, which costs time and computation power in maintaining the long-distance connection [11]. However, since the ultra-long-distance communication between IoT devices and remote clouds in real-world scenarios requires a large amount of bandwidth resources, sending all locally generated tasks to remote clouds for processing would bring serious problems, such as high latency and network congestion.

To alleviate these problems, one of the more effective approaches today is to offload all complex computing tasks from the local device to a nearby edge server. The edge servers

are geographically closer to the local devices, they feature low communication costs and short response times. SDN can be set up at the edge layer to manage network traffic and resources with more flexibility and scalability support [12]. This not only ensures the real-time requirements of latency-sensitive tasks, but also reduces the energy consumption of the local devices. However, due to cost and scale constraints, the computing resources on edge servers are not unlimited. When a large number of tasks are offloaded to edge servers at the same time or when resource-poor local devices require more computing and storage resources, the edge servers can be overloaded and cannot effectively meet the complex application requirements.

Since these single-computing paradigms cannot solve all offload problems, there is a need to jointly consider the heterogeneity of cloud and edge computing to build a cloud-edge-end-collaborative computing architecture. A white paper from ETSI illustrates that cloud and edge computing are highly complementary, and that cloud-edge-end-collaborative computing can provide better computing performance and transport performance than cloud or edge computing alone [13]. However, it remains a challenge to effectively offload in a cloud-edge-end-computing environment due to resource heterogeneity, the diversity of user requirements, network complexity and task dependencies. At present, a part of the related work does not consider the complexity of tasks in realistic scenarios and only considers each computing task as an indivisible whole when performing offloading [14–21], which is obviously unreasonable. Another part of the related work considers the complexity of the tasks and partitions the tasks, but the complexity of calculating the communication cost between subtasks makes it extremely difficult to perform offloading research in cloud-edge-end architectures [22–26], so their work is often performed in two-tier offloading architectures (cloud–edge architecture or edge–end architecture, etc.). Therefore, in this paper, we propose a multi-objective optimal offloading algorithm for the offloading problem of complex tasks in IoT cloud-edge-end-computing architectures, aiming to optimize both the average latency and average energy consumption through the offloading of subtasks and the scheduling of resources in a heterogeneous environment. The main contributions of this paper are as follows:

1. We integrate the characteristics of edge computing and cloud computing, and then fully integrate and, based on heterogeneous computing resources, establish an IoT cloud-edge-end-collaborative computing offload architecture.
2. Defining tasks as a directed, acyclic graph (DAG) composed of a set of nodes with interdependent subtasks, we then establish latency and energy consumption models by introducing two-dimensional offloading factors.
3. Based on the delay and energy consumption models, the offloading problem is formalized as a multi-objective optimization problem with the objective of minimizing the average delay and average energy consumption of task offloading, and a multi-objective offloading (SPMOO) algorithm based on an improved strength Pareto evolutionary algorithm (SPEA2) is proposed for solving the problem.

The rest of the paper is organized as follows. Section 2 discusses the related work; Section 3 presents the system model and the analysis of the problem; Section 4 describes the proposed multi-objective task offloading algorithm in detail; in Section 5 we perform extensive simulations and analyze different aspects; finally, Section 6 concludes the paper and discusses future work.

2. Related Work

To address the important challenges of working together with IoT systems and emerging computing paradigms, some researchers have proposed many effective task offloading algorithms to maximize application service performance from different perspectives. Based on whether the computing tasks are divisible or not, the tasks involved in the current related offloading research work are divided into two main categories: indivisible independent tasks and complex tasks consisting of multiple subtasks with data or logical dependencies.

2.1. Offloading Strategies for Independent Tasks

Since the dependency between subtasks is not involved, the offloading problem of independent tasks is much easier than that of dependent tasks. Therefore, there are many studies on the offloading strategy of independent tasks at present.

The study [15] proposed an online task scheduling algorithm for improving the total weighted response time (WRT) of all jobs. The study [16] considered the optimization problem of minimizing the long-term average delay of an IoT fog–cloud system and applied the Lyapunov drift-plus-penalty method to solve it. However, the optimization of long-term average delay may lead to performance imbalance among different tasks or time slots. The study [17] proposed an online method that attempts to maximize the number of tasks that meet the deadline while minimizing the average completion time (ACT) of the tasks through joint scheduling of network and computing resources. The study [18] proposed an energy-efficient dynamic task offloading (EEDTO) algorithm with the goal of minimizing energy consumption and task response time. This algorithm could adaptively select the optimal computation location for each task without requiring future system information as a priori knowledge. The study [19] transformed the dynamic performance optimization problem of service deployment for each user during system operation into a multi-arm bandit problem (MAB), which was then solved using a contextual Thompson sampling learning approach. The study [20] proposed a distributed, deep-learning method. They used a parallel deep neural network (DNN) to input the output of task load and offload decisions and updated the DNN parameters with the newly generated data.

2.2. Offloading Strategies for Dependent Tasks

The tasks generated by IoT devices in real scenarios are usually composed of a number of subtasks with logical or data dependencies. Any subtask must start execution only after all its predecessors have been executed; and, when executing these subtasks at different locations, data transfer between subtasks with dependencies is required, which can incur communication costs. Therefore, achieving optimal offloading decisions should not only focus on user requirements, complex access networks and resource utilization, but also take into account the dependencies between subtasks.

The study [27] proposed a greedy task graph partitioned offloading algorithm in order to minimize task communication overhead and energy consumption, which offloaded tasks at the device layer, cloudlet layer and cloud layer based on the computing power of the device and the type of task. However, the algorithm easily fell into local optimal solutions. The study [22] proposed an adaptive content-aware task scheduling (CATSA) framework. The approach was based on a multi-criteria technique to determine the way each task pair matches each computing paradigm and then selected an optimal solution from the solution set using a random search method. However, this work did not consider the resource heterogeneity between different computing layers. The study [28] used a long-short-term memory (LSTM) approach to predict the data size for each task. Based on the predicted values, the latency of each user task was optimized. However, this work simply partitioned the tasks without considering the dependencies between subtasks. The study [23] considered response time, reliability, and task execution cost in an edge cloud environment, and then formulated the offloading problem as an NP-hard, multi-objective optimization problem based on then on-dominated sorting genetic algorithm (NSGA-II) [29] and simulated binary crossover and polynomial variation to provide an offloading policy in the Pareto frontier. However, this work did not consider the power consumption model. The study [24] designed an online recurring score matrix-based heuristic algorithm (RSM-H) using a network state prediction method for the offloading problem of stateful, dataflow applications. However, this work only offloaded to the edge cloud and did not consider the power consumption model. The study [30] quantified the execution overhead as a weighted sum of task completion time and data transfer energy consumption and used this as an optimization objective to determine the execution location of a task. However, this work ignored the queueing order of tasks and the waiting execution time.

3. System Model and Problem Formulation

3.1. Overview

Since cloud servers have powerful computing and storage resources and rich applications, while edge servers have advantages of low communication cost, short response time,and strong network adaptability, we establish an IoT cloud-edge-end-collaborative computing offload architecture by combining the characteristics of both, fully integrating and invoking heterogeneous computing resources, and offloading tasks to the appropriate location based on the requirements of different applications in terms of computing power, energy cost and latency. As shown in Figure 1, the architecture consists of three layers, the end-device layer, edge layer and cloud layer. The end-device layer consists of a wide variety of IoT devices with sensors. These devices are usually small in size, have limited battery capacity, and have only limited computing and storage capabilities. So, to further process the data, these devices transmit the task data to the edge or cloud layers through wireless access points (APs) or base stations (BSs) [31]. The edge layer mainly consists of some lightweight edge servers that can provide low-latency computing services at the edge of the network. Meanwhile, to avoid overload, the edge servers can transfer tasks to the cloud servers for processing via wired links. The cloud layer has many cloud servers with rich computing and storage resources, and these servers provide services for users in different areas at the same time. However, the sharing among multiple users and the long transmission distance of connections to other layers lead to high communication cost for data transmission.

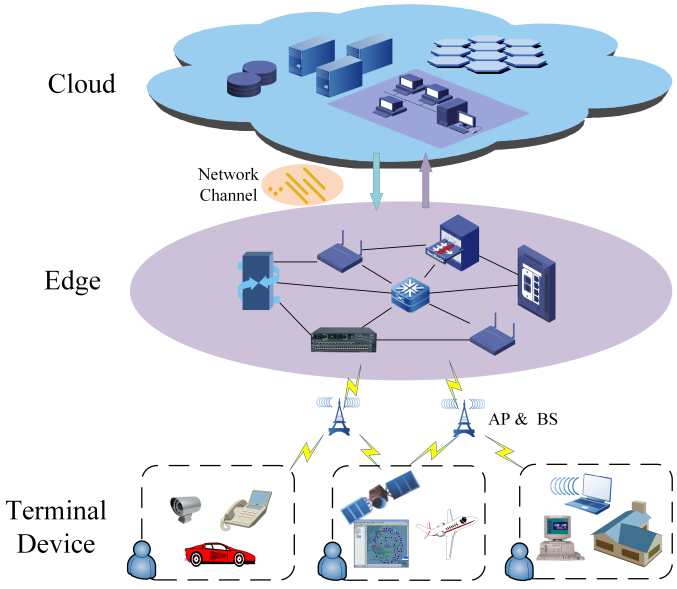


Figure 1. cloud-edge-end-collaborative computing offload system architecture.

In this three-tier computing offload architecture, it is assumed that a total of $U = \{1, 2, 3, \ldots, u\}$ computing tasks are generated at the end device layer and sent to the edge layer, with $K = \{1, 2, 3, \ldots, k\}$ edge server nodes and $S = \{1, 2, 3, \ldots, s\}$ cloud server nodes available to provide services. Then we partition the computing tasks from end devices into multiple subtasks with dependencies, which are offloaded to edge servers or cloud services for execution.

In this paper, these interdependent subtasks are defined architecturally as a directed acyclic graph (DAG) $G = (V, E)$ as shown in Figure 2. Where the vertex $V = \{i_u \mid i_u = 1_u, 2_u, 3_u, \ldots, n_u\}$ represents the services or microservices used by all subtasks of the task u. The modularity of the services or microservices makes it possible to deploy each service or microservice on various types of hardware, virtualized devices and specific libraries, thus solving the compatibility problem. The edge $E = \{(i_u, j_u) \mid i_u, j_u \in V\}$ represents the dependency between subtask i_u and subtask j_u, and $w^u_{i,j}$ indicates the size of data to be

transferred between i_u and j_u, which will incur additional communication cost when these two subtasks are executed at different locations when data transfer is required.

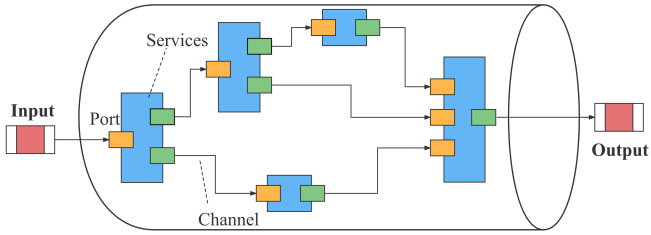


Figure 2. Task execution pipeline model.

Meanwhile, we introduce the offloading decision factor $x_{u,i} \in \{0,1\}$ to indicate whether the subtask i_u is offloaded or not, and the offloading location factor $x_{u,i} \in \{-1,0,1\}$ to indicate the specific location where the subtask i_u is offloaded. When $x_{u,i} = 0$ indicates that subtask i_u is executed locally, at this time $y_{u,i} = -1$ indicates that i_u is not offloaded. When $x_{u,i} = 1$, $y_{u,i} = 0$ indicates that subtask i_u is offloaded to the edge node, and when $x_{u,i} = 1$, $y_{u,i} = 1$ indicates that subtask i_u is offloaded to the cloud node. For convenience, some important symbols adopted in this paper and their description are listed in Table 1.

Table 1. Important symbols used in the paper and their description.

Symbols	Description
U	the total number of tasks
K	the total number of edge servers
S	the total number of cloud servers
N	the total number of subtasks
u	index of tasks
k	index of edge devices
s	index of cloud devices
i	index of subtasks
pre_i	index of the predecessor of the subtask
$x_{u,i}$	the offloading decision factor to indicate whether the subtask i_u is offloaded or not
$y_{u,i}$	the offloading location factor to indicate the specific location where the subtask i_u is offloaded
$w^u_{i,j}$	the size of data to be transferred between i_u and j_u
D^u_i	the CPU resources required to execute subtask i_u
$t^{st}_{u,i}$	the time when the subtask i_u started to be executed
$t^{ed}_{u,i}$	the time when the subtask i_u finished to be executed
f_{local}	the CPU frequency of local device
f_k	the CPU frequency of edge server
f_s	the CPU frequency of cloud server
$r_{u,k}$	the transmission rate between the end device and the edge server
$r_{u,s}$	the transmission rate between the end device and the cloud server
$r_{k,s}$	the transmission rate between the edge server and the cloud server
B	wireless transmission channel bandwidth
H_u	channel gain
c_u	transmission power of the end device for wireless transmission
$d^{\eta}_{u,k}$	path loss between the end device and the edge server
$d^{\eta}_{u,s}$	path loss between the end device and the cloud server
P^u_i	the set of all predecessor nodes of subtask i_u
T^{total}_u	the total delay required for the execution of task u
μ_u	the CPU energy factors of the end device
μ_k	the CPU energy factors of the edge server
E^{cal}_u	the computing energy consumption of the task u
E^{trans}_u	the transmission energy consumption of the task u
E^{total}_u	the total energy consumption

Here is an example which takes USVs as end devices for intelligent water quality monitoring application. The application breaks through the limits of conventional manual sampling monitoring, realizing the collection of changes of water quality, water temperature and other data in a certain area, and can regularly report the regional water quality analysis results. As shown in Figure 3, the application includes the following four parts: information collection and inspection, water quality level analysis and evaluation, historical water quality data query and water-quality distribution visualization. Among them, information collection and inspection can only be run locally, while the other three parts can be run locally, on the edge or cloud. The specific task offload execution position is determined by the scheduler according to the system state and task configuration information.

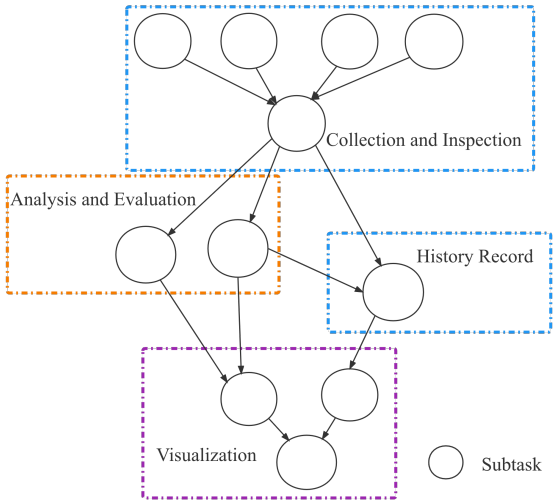


Figure 3. USVs intelligent water quality monitoring.

3.1.1. Delay Model

Since each subtask has a time to start execution and a time to finish execution, in this paper we use $t_{u,i}^{st}$ to denote the subtask i_u start execution time and $t_{u,i}^{ed}$ to denote the subtask finish execution time. Then we denote the delay required for subtask i_u to be executed at the local device as $\frac{D_i^u}{f_{local}}$, the delay required at the edge server node as $\frac{D_i^u}{f_k}$, and the delay required at the cloud server node as $\frac{D_i^u}{f_s}$, where D_i^u denotes the CPU resources required to execute subtask i_u, f_{local} denotes the CPU frequency of local device, f_k denotes the CPU frequency of edge server, and f_s denotes the CPU frequency of cloud server. From this we can obtain:

$$t_{u,i}^{ed} = t_{u,i}^{st} + (1 - x_{u,i}) \cdot \frac{D_i^u}{f_{\text{local}}} + x_{u,i} \cdot \left((1 - y_{u,i}) \cdot \frac{D_i^u}{f_k} + y_{u,i} \cdot \frac{D_i^u}{f_s} \right) \tag{1}$$

Since the end device layer generally communicates with the edge server and the cloud server through wireless access points (APs) or small base stations (BSs), the transmission rate $r_{u,k}$ between the end device and the edge server can be obtained according to Shannon's formula as:

$$r_{u,k} = B \log_2 \left(1 + \frac{H_u c_u d_{u,k}^{\eta}}{\sigma^2} \right) \tag{2}$$

where B denotes the channel bandwidth of wireless transmission, H_u denotes the channel gain of the end device communicating through the wireless channel or base station, c_u denotes the transmitting power of the end device for wireless transmission, and $d_{u,k}^{\eta}$ denotes the path loss between the end device and the edge server. Similarly, the transmission rate $r_{u,k}$ between the end device and the cloud server can be expressed as:

$$r_{u,s} = B\log_2\left(1 + \frac{H_u c_u d_{u,s}^{\eta}}{\sigma^2}\right) \tag{3}$$

since edge servers usually communicate with cloud servers through wired links, we denote the transmission rate between them as $r_{k,s}$.

For each subtask, its earliest start execution time is mainly limited by the completion execution time of all its predecessor nodes, and if the subtask is not executed at the same location as its predecessor nodes, the time consumed by data transmission is also considered, which can be expressed as:

$$t_{u,i}^{st} = \max_{pre\in P_i^u}\left\{\begin{array}{l} t_{u,pre_i}^{ed} + x_{u,i}\cdot\left(1 + x_{u,i}\oplus x_{u,pre_i}\right)\left(y_{u,i} - y_{u,pre_i}\right)^2\cdot\frac{w_{pre\to i}^u}{r_{k,s}} \\ +\left(x_{u,i}\oplus x_{u,pre_i}\right)\left(\left(y_{u,i} + y_{u,pre_i} + 1\right)\cdot\frac{w_{pre\to i}^u}{r_{u,s}} + \left(y_{u,i} + y_{u,pre_i}\right)^2\cdot\frac{w_{pre\to i}^u}{r_{u,k}}\right) \end{array}\right\} \tag{4}$$

where, t_{u,pre_i}^{ed} denotes the completion execution time of the predecessor nodes of subtask i_u, P_i^u denotes the set of all predecessor nodes of subtask i_u, and $w_{pre\to i}^u$ denotes the amount of data to be transferred between subtask i_u and its predecessor nodes.

To facilitate the representation of the total delay of the task computation, we abstracted the input and output of the task as two subtasks i_0^u and i_{n+1}^u, which do not need to consider the amount of computation, placed at the beginning and end of the task, respectively, so that the total delay required for the execution of task u can be expressed as:

$$T_u^{total} = t_{u,n+1}^{st} - t_{u,0}^{st} \tag{5}$$

3.1.2. Energy Consumption Model

When the task is executed in the cloud server, the energy required is often negligible compared to the abundant computing resources of the cloud service nodes themselves, so this paper mainly considers the energy consumption of three parts, local computing energy, edge server node computing energy and communication transmission energy. Among them, the computing energy consumption of the task u can be expressed as:

$$E_u^{cal} = \sum_{i=0}^{N}(\mu_u\cdot(1 - x_{u,i})\cdot D_i^u + \mu_k\cdot x_{u,i}\cdot(1 - y_{u,i})\cdot D_i^u), \tag{6}$$

where, μ_u and μ_k denote the CPU energy conversion factors of the end device and the edge server node, respectively.

The energy consumption generated by data transmission during the execution of task u includes the communication energy consumption $E_u^{trans}(local, edge)$ between the local device and the edge server, the communication energy consumption $E_u^{trans}(local, cloud)$ between the local device and the cloud server and the communication energy consumption $E_u^{trans}(edge, cloud)$ between the edge server and the cloud server, which can be expressed as:

$$E_u^{trans}(local, edge) = \sum_{i=1}^{N+1}\sum_{pre}^{P_i^u}\left(x_{u,i}\oplus x_{u,pre_i}\right)\left(y_{u,i} + y_{u,pre_i}\right)^2\cdot\frac{w_{pre\to i}^u}{r_{u,s}}\cdot c_{u,s} \tag{7}$$

$$E_u^{trans}(local, cloud) = \sum_{i=1}^{N+1}\sum_{pre}^{P_i^u}\left(x_{u,i}\oplus x_{u,pre_i}\right)\left(y_{u,i} + y_{u,pre_i} + 1\right)\cdot\frac{w_{pre\to i}^u}{r_{u,s}}\cdot c_{u,k} \tag{8}$$

$$E_u^{trans}(edge, cloud) = \sum_{i=1}^{N+1}\sum_{pre}^{P_i^u}x_{u,i}\cdot\left(1 + x_{u,i}\oplus x_{u,pre_i}\right)\left(y_{u,i} - y_{u,pre_i}\right)^2\cdot\frac{w_{pre\to i}^u}{r_{k,s}}\cdot c_{k,s} \tag{9}$$

Therefore, the total communication energy consumption generated by performing data transfer during the execution of task u is

$$E_u^{trans} = E_u^{trans}(local, edge) + E_u^{trans}(local, cloud) + E_u^{trans}(edge, cloud). \tag{10}$$

Finally, the total energy consumption required during the computation and offloading performed by task u can be expressed as:

$$E_u^{total} = E_u^{cal} + E_u^{trans}. \tag{11}$$

3.2. Problem Formulation

Considering the real scenario where there is a conflict between the objectives of minimizing the average delay and minimizing the average energy consumption of task computation offload, if we use the weighted sum aggregation method to assign weights to the objectives, we can only obtain a single optimal solution, for which it is often difficult to meet the diverse needs of different users or service providers in a resource-variant environment. Therefore, in order to better meet the needs of users and providers, and to improve user experience and resource utilization as much as possible, we formalize the problem as a multi-objective optimization problem, where the optimization objective is to minimize the average delay and average energy consumption of task computation offloading. According to Equations (5) and (11), the optimization objective can be expressed as:

$$\min \frac{1}{U} \sum_{u=1}^{U} T_u^{total} \tag{12}$$

$$\min \frac{1}{U} \sum_{u=1}^{U} E_u^{total} \tag{13}$$

$$\begin{aligned} s.t. \quad & C1: x_{u,i} \in \{0,1\}, y_{u,i} \in \{-1,0,1\}, \forall u \in [1,U], \forall i \in [0,N+1] \\ & C2: x_{u,0} = 0, x_{u,N+1} = 0, \forall u \in [1,U] \\ & C3: t_{u,i}^{st} \geq \max t_{u,pre_i}^{ed}, \forall pre \in P_i^u, \forall u \in [1,U], \forall i \in [1,N+1] \\ & C4: \sum_{u=1}^{U} \sum_{i=1}^{N} x_{u,i} \cdot \left[(1 - y_{u,i}) \cdot \mathfrak{M}(\tau - t_{u,i}^{st}) \cdot \mathfrak{M}\left(t_{u,i}^{st} + \frac{D_i^u}{f_k} - \tau \right) \right. \\ & \left. + y_{u,i} \cdot \mathfrak{M}(\tau - t_{u,i}^{st}) \cdot \mathfrak{M}\left(t_{u,i}^{st} + \frac{D_i^u}{f_s} - \tau \right) \right] \leq K + S \end{aligned} \tag{14}$$

Constraint C1 in Equation (14) indicates that for any subtask can only be executed on the edge server node or on the cloud server node. Constraint C2 indicates that for any task u, its input and output are at the local device. Constraint C3 indicates that each subtask must complete all its predecessors before it can start execution. Constraint C4 indicates that any edge server node and cloud server node cannot process multiple subtasks simultaneously in parallel. In this paper, we divide the time duration into a sequence of $T = \{1, 2, 3, \ldots, \tau\}$ time slots, and for each time slot, the number of offloaded subtasks cannot exceed the sum of the number of edge servers and cloud servers.

4. The Proposed SPMOO Algorithm

For the multi-objective optimization problems formulated in Section 3.2, the more frequently used solution algorithms are NSGA-II, MOEA/D, SPEA2, and etc [29,32–34]. In this paper, a multi-objective offloading algorithm (SPMOO) based on the improved strength Pareto evolutionary algorithm (SPEA2) is proposed considering the parallel processing mechanism and global optimization. Since the standard SPEA2 algorithm has high randomness and poor ability in local search, we add local search set to enhance the local search ability of the algorithm based on the original algorithm, and we also adopt intergenerational crossover and individual similarity judgment strategies to improve popu-

lation diversity and convergence speed. Therefore, our algorithm is able to obtain a set of Pareto-optimal solutions for user or supplier selection in a short time.

As shown in Algorithm 1, we initialize the population P_0 with size M and create an empty external archive set A_0 and an empty local search set L_0, both with size M'. Then we calculate the fitness of all individuals in the population and external archive set and local search set, and save all the non-dominated solution sets in them to the next generation of external archive set A_{t+1} and local search set L_{t+1}. If the number of individuals in the external archive set exceeds M' at this time, truncate the operation; if the number of individuals does not reach, we select some of the dominant liberation from P_t and A_t into A_{t+1} and L_{t+1}. After that, the local search update is performed for L_{t+1}, and the individuals from the external archive set A_{t+1} at this time are selected into the mating pool for crossover and mutation and the results are kept into the next generation population P_{t+1} to recalculate the fitness, thus iterating. Finally, the non-dominated solution set in A_{t+1} is taken as the final output of the offloading policy.

Algorithm 1: SPMOO

Input: M (population size), M' (archive size), T (maximum number of iterations)
Output: Q (offloading strategy)

```
Initialize the first generation P_0, empty external set A_0 and local search set L_0 ;
t = 0 ;
Calculate fitness values of individuals in P_t, A_t and L_t;
Copy all nondominated individuals from P_t, A_t and L_t to A_{t+1} ;
for t < T do
    if P_t + A_t > M' then
        Clip A_{t+1} by truncation operation ;
    else
        Add dominated individuals from P_t, A_t and L_t to A_{t+1} ;
    L_{t+1} = A_{t+1} ;
    Conduct the binary tournament selection on A_t and A_{t+1} for filling the mating
     pool ;
    Crossover and mutation operation ;
    Perform a partial search update on L_t ;
    Set A_{t+1} as the next generation ;
    t = t + 1 ;
Set Q to the set of offloading strategy vectors represented by the nondominated
 individuals in A_{t+1} ;
final ;
return Q;
```

4.1. Fitness Calculation

To avoid the situation that individuals have the same fitness, the SPEA2 algorithm calculates the number of dominated solutions for each individual in the population and external archive set and defines it as the strength value $S(i)$, and based on this, the original fitness $R(i)$ is defined to represent the number of dominated solutions for individual i. The larger the $R(i)$, the more individuals dominate individual i.

$$S(i) = |\{j \mid j \in P_t + A_t \wedge i \succ j\}| \tag{15}$$

$$R(i) = \sum_{j \in P_t + A_t, j \succ i} S(i) \tag{16}$$

When most of the individuals are non-dominated, it is difficult to distinguish them by $R(i)$ alone, so the density value $D(i)$ is introduced, and the density value is calculated using

the K-nearest neighbor method, where the density value of any individual is a function of the distance of its k neighboring individuals.

$$D(i) = \frac{1}{\sigma_i^k + 2} \tag{17}$$

$$k = \sqrt{M + M'} \tag{18}$$

In the above equations, σ_i^k represents the Euclidean distance between individual i and its adjacent individual k. Finally, the fitness of individual i is obtained:

$$F(i) = R(i) + D(i). \tag{19}$$

4.2. Local Search Update

In order to avoid falling into local optimum, we replicate the external archive set to create a local search set, and then rank the individuals in this set according to their fitness and perform domain search on some of them with higher fitness. First, the decision factor of the selected individuals is the center of the circle, and then the search is performed within the determined search range, where the search radius and search density can be set by ourselves, and the higher the radius and density, the stronger the local search capability, and at the same time the computing effort will increase. Generally speaking, individuals with high adaptability will be closer to the Pareto optimal solution boundary, and the local search operation for these individuals can effectively improve the search efficiency of the algorithm.

4.3. Intergenerational Crossover

The standard SPEA2 algorithm uses tournament selection to select individuals from an external archive to generate a mating pool. As the number of individuals in the non-dominated solution gradually increases, this approach reduces the genetic composition of the resulting next generation because it has a high probability of selecting similar individuals to generate a mating pool. Intergenerational crossover uses tournament selection with replacement to select an individual from the current generation external archive set while selecting an individual from the previous generation archive set. Compared to the original crossover, intergenerational crossover generates both good and bad individuals, so we keep only the best solutions from both intergenerational and original crossovers.

On the other hand, we evaluate the similarity of individuals before performing a crossover, and only perform a crossover operation on two individuals when the difference between them exceeds the maximum threshold of similarity. Many related works use the number of identical gene positions of chromosomes as individual similarity, which is simple to operate but has some drawbacks. For binary coding, there may be many identical gene positions in two chromosomes corresponding to the same position, but their corresponding actual values may be very different, which may easily cause the Hamming cliff phenomenon [35]. Therefore, this paper adopts the Euclidean distance of individuals to be crossed as the measure of individual similarity. Its corresponding formula is:

$$\text{Crossover } = \begin{cases} 1, & D_{ij} \geq D_{\max} \\ 0, & D_{ij} < D_{\max} \end{cases}, \tag{20}$$

where, D_{ij} denotes the Euclidean distance between individuals to be crossed and D_{max} denotes the set maximum similarity threshold.

4.4. Algorithm Complexity Analysis

The computational complexity of the NSGA-II, MOEA/D and SPEA2 algorithms are $O(rN^2)$, $O(rNT)$ and $O(rN \log N)$. The computational complexity of SPMOO proposed in this paper is $O(rN \log N)$, where r is the number of objective functions and N is the population size. The evaluation population set P, external archive A and empty local search set L_0 are taken into consideration simultaneously in SPMOO, so $N = M + 2M'$. Compared

with the SPEA2 algorithm, the SPMOO algorithm optimizes the evolutionary selection mechanism without increasing time complexity, which reduces mutation and crossover operations of optimal solutions and accelerates the iteration speed.

5. Simulation Results

In this section, we conducted a large number of experiments to verify and evaluate our algorithm. Firstly, we describe the experimental setup and several benchmark methods. Then, we do some comparison with the benchmark methods and finally we analyze and evaluate the experimental results.

5.1. Simulation Settings

In the simulation experiments, we set the number of endpoint devices to 40 by default. Since the number of edge layer nodes is usually much smaller than the number of endpoint layers, we set the number of servers to eight by default. The execution time of the tasks varies from device to device, since they have different processing capabilities. All parameters used in the experiments are shown in Table 2.

Table 2. Parameters and values.

Parameter	Value	Parameter	Value
U	[20, 60]	f_s	20 GHz
K	[2, 10]	D_i^u	[1000, 1500] megacycles
S	[1, 2]	$w_{i,j}^u$	[200, 500] kB
N	[4, 12]	c_u	[1, 2] W
B	10 MHz	H_u	[2, 4]
f_u	[0.5, 1] GHz	$d_{u,k}^{\eta}$	[0.0001, 0.0005]
f_k	[2, 10] GHz	σ	30 dBm

We compare the proposed SPMOO algorithm with the following three benchmark algorithms, which are described as follows.

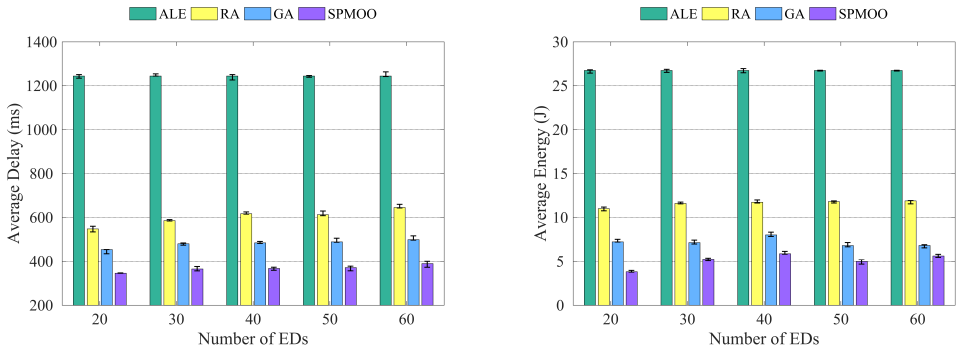
ALE (All Local Execution), which does not consider the resources of edge nodes and cloud nodes, with all tasks processed locally on the device.

RA (Random Allocation), which randomly places all subtasks, except those that can only be executed locally, to the edge or cloud nodes.

GA (Genetic Algorithm), which simulates the natural biological evolution mechanism and performs efficient, parallel random global search and optimization for all policies, and adaptively controls the search process to find the optimal solution.

5.2. Analysis and Evaluation

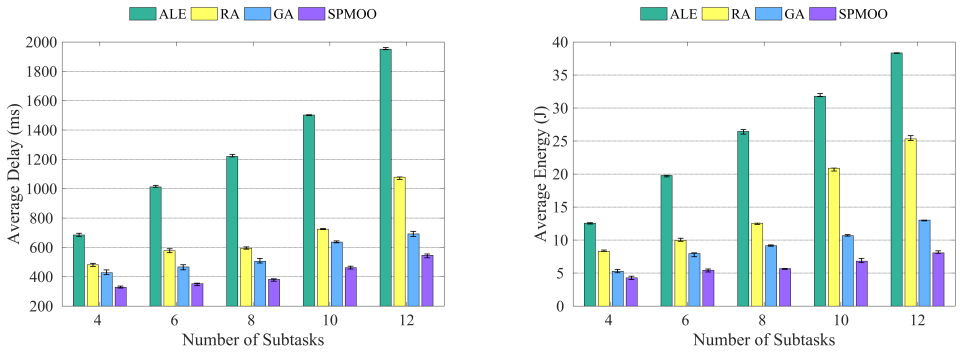
From Figure 4a, we can see that the average delay of the four algorithms does not change much when the number of end devices increases, and the average delay of the SPMOO algorithm proposed in this paper is the lowest among the four algorithms. This is mainly due to the full consideration of the environmental factors in the subtask offloading process, such as the computing power of the edge server and the communication laws between subtasks and nodes, which make our algorithm more accurate and efficient in performing task offloading. The traditional GA algorithm does not introduce methods such as local search strategy and intergenerational crossover strategy, which leads to a gap between it and the algorithm proposed in this paper in terms of constraint processing. The RA algorithm does not use an optimization algorithm for the node offloading location, so it performs poorly in terms of both average delay and average energy consumption. The analysis of the average energy consumption in Figure 4a is similar to the analysis in Figure 4b.



(**a**) (**b**)

Figure 4. Comparison of the four algorithms when the number of end devices is {20, 30, 40, 50, 60}; (**a**) Average delay; (**b**) Average energy consumption.

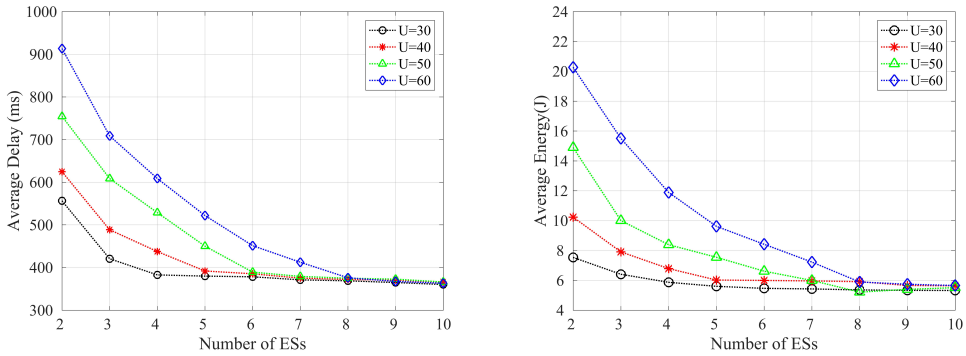
It can be seen in Figure 5a,b that when the number of end devices is kept constant and the number of subtasks increases, the average latency of all four algorithms increases accordingly, while the SPMOO algorithm proposed in this paper has the best results.



(**a**) (**b**)

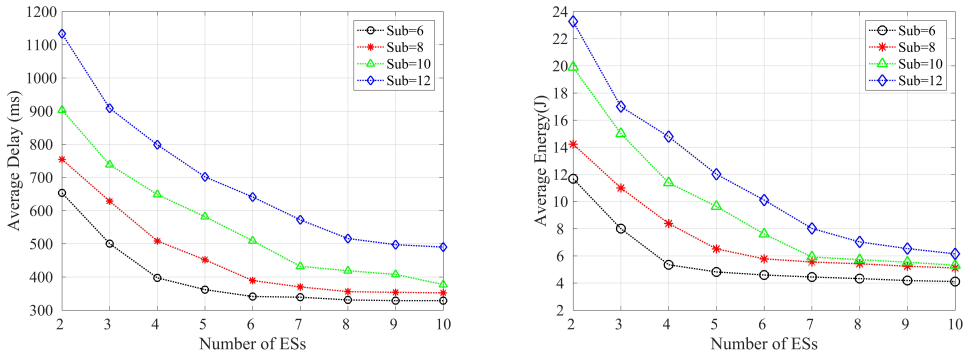
Figure 5. Comparison of the four algorithms when the number of subtasks is {4, 6, 8, 10, 12}; (**a**) Average delay; (**b**) Average energy consumption.

Figure 6a,b show the graphs of average delay and average energy consumption with the number of edge servers when the number of end devices is 10, 15, 20 and 25, respectively. We can see that the latency and energy consumption slowly decrease with the increase of the number of edge servers, and when the number of edge servers reaches eight, the average latency and average energy consumption can basically reach an optimal state. Therefore, in order to improve the utilization of resources and allocate them rationally, we can occupy as few edge servers' resources as possible while ensuring the latency and energy consumption requirements. Figure 7a,b show the graphs of average delay and average energy consumption with the number of edge servers when the number of subtasks is 6, 8, 10 and 12, respectively. Similarly, the average delay and average energy consumption can be seen to decrease and stabilize as the number of edge servers increases.



(**a**) (**b**)

Figure 6. Variation of the average delay and the average energy consumption with the number of edge servers when the number of end devices is {30, 40, 50, 60} respectively; (**a**) Average delay; (**b**) Average energy consumption.



(**a**) (**b**)

Figure 7. Variation of the average delay and the average energy consumption with the number of edge servers when the number of subtasks is {6, 8, 10, 12} respectively; (**a**) Average delay; (**b**) Average energy consumption.

6. Conclusions

In this paper, we study the multi-objective optimization task offloading problem with resource-constrained edge servers in an IoT environment. To address this problem, we construct a cloud-edge-end-collaborative computing offload architecture consisting of end device layer, edge layer and cloud layer. Based on this, we construct a latency and energy consumption model by introducing two-dimensional offloading decision factors, and formalize the model as a multi-objective optimization problem with the optimization objective of minimizing the average latency and average energy consumption of task offloading, and then propose a multi-objective offloading (SPMOO) algorithm based on an improved strength Pareto evolutionary algorithm (SPEA2) for solving the problem. Extensive experiments show that our proposed method can significantly reduce the overall average delay and average energy consumption of the system. However, our method is less adaptable to new scenarios and takes a lot of time to get new offload decisions when the type of offload tasks or device resources change. Therefore, in future work we will further explore how to get offload decisions faster in dynamically changing heterogeneous scenarios by combining the feature of deep reinforcement learning that builds on prior experience to significantly accelerate the learning of new tasks.

Author Contributions: Data curation, L.L.; methodology, L.L.; funding acquisition, H.C.; formal analysis, Z.X.; visualization, L.L.; supervision, H.C.; writing—original draft, L.L.; writing—review and editing, H.C. and Z.X. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Ningbo Natural Science Foundation grant number 2021J090.

Institutional Review Board Statement: Not Applicable, the study does not involve humans or animals.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not Applicable, the study does not report any data.

Conflicts of Interest: The authors declare that there is no conflict of interest regarding the publication of this paper.

References

1. Mehmood, Y.; Ahmad, F.; Yaqoob, I.; Adnane, A.; Imran, M.; Guizani, S. Internet-of-Things-Based Smart Cities: Recent Advances and Challenges. *IEEE Commun. Mag.* **2017**, *55*, 16–24. [CrossRef]
2. Yu, W.; Liang, F.; He, X.; Hatcher, W.G.; Lu, C.; Lin, J.; Yang, X. A Survey on the Edge Computing for the Internet of Things. *IEEE Access* **2018**, *6*, 6900–6919. [CrossRef]
3. Mallapuram, S.; Ngwum, N.; Yuan, F.; Lu, C.; Yu, W. Smart City: The State of the Art, Datasets, and Evaluation Platforms. In Proceedings of the 2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS), Wuhan, China, 24–26 May 2017; pp. 447–452. [CrossRef]
4. Wang, K. Migration strategy of cloud collaborative computing for delay-sensitive industrial IoT applications in the context of intelligent manufacturing. *Comput. Commun.* **2020**, *150*, 413–420. [CrossRef]
5. Arias, O.; Wurm, J.; Hoang, K.; Jin, Y. Privacy and Security in Internet of Things and Wearable Devices. *IEEE Trans.-Multi-Scale Comput. Syst.* **2015**, *1*, 99–109. [CrossRef]
6. Zhang, J.; Dai, M.; Su, Z. Task Allocation with Unmanned Surface Vehicles in Smart Ocean IoT. *IEEE Internet Things J.* **2020**, *7*, 9702–9713. [CrossRef]
7. Lee, I.; Lee, K. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Bus. Horiz.* **2015**, *58*, 431–440. [CrossRef]
8. Adjabi, I.; Ouahabi, A.; Benzaoui, A.; Taleb-Ahmed, A. Past, Present, and Future of Face Recognition: A Review. *Electronics* **2020**, *9*, 1188. [CrossRef]
9. Premsankar, G.; Di Francesco, M.; Taleb, T. Edge Computing for the Internet of Things: A Case Study. *IEEE Internet Things J.* **2018**, *5*, 1275–1284. [CrossRef]
10. Hu, L.; Tian, Y.; Yang, J.; Taleb, T.; Xiang, L.; Hao, Y. Ready Player One: UAV-Clustering-Based Multi-Task Offloading for Vehicular VR/AR Gaming. *IEEE Netw.* **2019**, *33*, 42–48. [CrossRef]
11. Ning, H.; Farha, F.; Mohammad, Z.N.; Daneshmand, M. A Survey and Tutorial on "Connection Exploding Meets Efficient Communication" in the Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 10733–10744. [CrossRef]
12. Bouras, M.A.; Farha, F.; Ning, H. Convergence of Computing, Communication, and Caching in Internet of Things. *Intell. Converg. Netw.* **2020**, *1*, 18–36. [CrossRef]
13. Reznik, A.; Murillo, L.M.C.; Fang, Y.; Featherstone, W.; Filippou, M.; Fontes, F.; Giust, F.; Huang, Q.; Li, A.; Turyagyenda, C.; et al. *Cloud RAN and MEC: A Perfect Pairing*; ETSI White Paper; ETSI: Sophia Antipolis, France, 2018; pp. 1–24.
14. Hu, X.; Wang, L.; Wong, K.K.; Tao, M.; Zhang, Y.; Zheng, Z. Edge and Central Cloud Computing: A Perfect Pairing for High Energy Efficiency and Low-Latency. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 1070–1083. [CrossRef]
15. Han, Z.; Tan, H.; Li, X.Y.; Jiang, S.H.C.; Li, Y.; Lau, F.C.M. OnDisc: Online Latency-Sensitive Job Dispatching and Scheduling in Heterogeneous Edge-Clouds. *IEEE/ACM Trans. Netw.* **2019**, *27*, 2472–2485. [CrossRef]
16. Li, L.; Guo, M.; Ma, L.; Mao, H.; Guan, Q. Online Workload Allocation via Fog-Fog-Cloud Cooperation to Reduce IoT Task Service Delay. *Sensors* **2019**, *19*, 3830. [CrossRef]
17. Meng, J.; Tan, H.; Li, X.Y.; Han, Z.; Li, B. Online Deadline-Aware Task Dispatching and Scheduling in Edge Computing. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *31*, 1270–1286. [CrossRef]
18. Wu, H.; Wolter, K.; Jiao, P.; Deng, Y.; Zhao, Y.; Xu, M. EEDTO: An Energy-Efficient Dynamic Task Offloading Algorithm for Blockchain-Enabled IoT-Edge-Cloud Orchestrated Computing. *IEEE Internet Things J.* **2021**, *8*, 2163–2176. [CrossRef]
19. Ouyang, T.; Li, R.; Chen, X.; Zhou, Z.; Tang, X. Adaptive User-managed Service Placement for Mobile Edge Computing: An Online Learning Approach. In Proceedings of the IEEE Conference on Computer Communications (INFOCOM), Paris, France, 29 April–2 May 2019; pp. 1468–1476. [CrossRef]
20. Wu, H.; Zhang, Z.; Guan, C.; Wolter, K.; Xu, M. Collaborate Edge and Cloud Computing with Distributed Deep Learning for Smart City Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 8099–8110. [CrossRef]
21. Alfakih, T.; Hassan, M.M.; Gumaei, A.; Savaglio, C.; Fortino, G. Task Offloading and Resource Allocation for Mobile Edge Computing by Deep Reinforcement Learning Based on SARSA. *IEEE Access* **2020**, *8*, 54074–54084. [CrossRef]

22. Lakhan, A.; Li, X. Content Aware Task Scheduling Framework for Mobile Workflow Applications in Heterogeneous Mobile-Edge-Cloud Paradigms: CATSA Framework. In Proceedings of the 2019 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom), Xiamen, China, 16–18 December 2019; pp. 242–249. [CrossRef]
23. De Maio, V.; Kimovski, D. Multi-objective Scheduling of Extreme Data Scientific Workflows in Fog. *Future Gener. Comput. Syst.* **2020**, *106*, 171–184. [CrossRef]
24. Ding, S.; Yang, L.; Cao, J.; Cai, W.; Tan, M.; Wang, Z. Partitioning Stateful Data Stream Applications in Dynamic Edge Cloud Environments. *IEEE Trans. Serv. Comput.* **2021**, 1, 1. [CrossRef]
25. Wang, S.; Ding, Z.; Jiang, C. Elastic Scheduling for Microservice Applications in Clouds. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 98–115. [CrossRef]
26. Zhang, T.; Chen, W. Computation Offloading in Heterogeneous Mobile Edge Computing with Energy Harvesting. *IEEE Trans. Green Commun. Netw.* **2021**, *5*, 552–565. [CrossRef]
27. Naouri, A.; Wu, H.; Nouri, N.A.; Dhelim, S.; Ning, H. A Novel Framework for Mobile-Edge Computing by Optimizing Task Offloading. *IEEE Internet Things J.* **2021**, *8*, 13065–13076. [CrossRef]
28. Miao, Y.; Wu, G.; Li, M.; Ghoneim, A.; Al-Rakhami, M.; Hossain, M.S. Intelligent Task Prediction and Computation Offloading based on Mobile-Edge Cloud Computing. *Future Gener. Comput. Syst.* **2020**, *102*, 925–931. [CrossRef]
29. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197. [CrossRef]
30. Sun, H.; Yu, H.; Fan, G.; Chen, L. Energy and Time Efficient Task Offloading and Resource Allocation on the Generic IoT-Fog-Cloud Architecture. *Peer-to-Peer Netw. Appl.* **2020**, *13*, 548–563. [CrossRef]
31. Liu, H.; Eldarrat, F.; Alqahtani, H.; Reznik, A.; de Foy, X.; Zhang, Y. Mobile Edge Cloud System: Architectures, Challenges, and Approaches. *IEEE Syst. J.* **2018**, *12*, 2495–2508. [CrossRef]
32. Zitzler, E.; Laumanns, M.; Thiele, L. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*; Eidgenössische Technische Hochschule Zürich (ETH): Zürich, Switzerland, 2001. [CrossRef]
33. Zhang, Q.; Li, H. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Trans. Evol. Comput.* **2007**, *11*, 712–731. [CrossRef]
34. Xu, X.; Wu, Q.; Qi, L.; Dou, W.; Tsai, S.B.; Bhuiyan, M.Z.A. Trust-Aware Service Offloading for Video Surveillance in Edge Computing Enabled Internet of Vehicles. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 1787–1796. [CrossRef]
35. Strasser, S.; Goodman, R.; Sheppard, J.; Butcher, S. A New Discrete Particle Swarm Optimization Algorithm. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Denver, CO, USA, 20–24 July 2016; pp. 53–60. [CrossRef]