electronics

MDPI

Article

Task Offloading in Real-Time Distributed Energy Power Systems

Ningchao Wu [1,*], Xingchuan Bao [2], Dayang Wang [3], Song Jiang [3], Manjun Zhang [1] and Jing Zou [4]

[1] State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China; zhangmanjun@bupt.edu.cn
[2] State Grid Laboratory of Electric Power Communication Network Technology, State Grid Smart Grid Research Institute Co., Ltd., Nanjing 210003, China; boxicha@sina.com
[3] Information and Communication Branch of State Grid Jiangsu Electric Power Co., Ltd., Nanjing 210024, China; wangdy1@js.sgcc.com.cn (D.W.); jsong@js.sgcc.com.cn (S.J.)
[4] State Grid Economic and Technological Research Institute Co., Ltd., Beijing 102200, China; lailaizou@163.com
* Correspondence: wuningchao@bupt.edu.cn

Abstract: The distributed energy power system needs to provide sufficient and flexible computing power on demand to meet the increasing digitization and intelligence requirements of the smart grid. However, the current distribution of the computing power and loads in the energy system is unbalanced, with data center loads continuously increasing, while there is a large amount of idle computing power at the edge. Meanwhile, there are a large number of real-time computing tasks in the distributed energy power system, which have strict requirements on execution deadlines and require reasonable scheduling of multi-level heterogeneous computing power to meet real-time computing demands. Based on the aforementioned background and issues, this paper studies the real-time service scheduling problem in a multi-level heterogeneous computing network of distributed energy power systems. Specifically, we consider the divisibility of tasks in the model. This paper presents a hierarchical real-time task-scheduling framework specifically designed for distributed energy power systems. The framework utilizes an orchestrating agent (OA) as the execution environment for the scheduling module. Building on this, we propose a hierarchical selection algorithm for choosing the appropriate network layer for real-time tasks. Further, we develop two scheduling algorithms based on greedy strategy and genetic algorithm, respectively, to effectively schedule tasks. Experiments show that the proposed algorithms have a superior success rate in scheduling compared to other current algorithms.

Keywords: distributed energy power system; real-time scheduling; greedy algorithm; genetic algorithm

Citation: Wu, N.; Bao, X.; Wang, D.; Jiang, S.; Zhang, M.; Zou, J. Task Offloading in Real-Time Distributed Energy Power Systems. *Electronics* **2024**, *13*, 2747. https://doi.org/10.3390/electronics13142747

Academic Editor: Dario Di Cara

Received: 14 May 2024
Revised: 5 July 2024
Accepted: 10 July 2024
Published: 12 July 2024

1. Introduction

With the rapid development of new energy technologies and the wide application of renewable energy, distributed energy power systems have become some of the main development directions in the power industry. However, the integration of distributed energy brings many challenges to the traditional power system, including energy management, scheduling optimization, system stability, etc. For instance, the consumption and grid integration of new energy introduces a large amount of real-time data collection, analysis, and decision-making tasks, which require efficient real-time scheduling to ensure the stable operation of the power system and the balance of supply and demand. The real-time scheduling of computing tasks, as one of the key links in the optimization of distributed energy power systems, has a significant impact on the efficiency and stability of the system's operation.

However, the distribution of the current power system's computing resources is characterized by multi-level heterogeneity, from the headquarters' data center to provincial and municipal data centers, and then to edge and terminal sides. Not only is the distribution of computing resources uneven, but there is also a significant difference between

various types of storage, communication, and computing resources. This distributed and heterogeneous characteristic, although providing rich computing resources for the power system, also leads to the problem of low resource utilization efficiency. A large portion of the computing load is concentrated in the computing centers of big cities, while the edge computing resources scattered around are mostly idle. This situation not only limits the efficient use of energy by the power system but also hinders the process of intelligent management of the power system.

Faced with this challenge, we need to rethink how to schedule computing tasks in the distributed energy power system. Through analyzing the current distribution of computing resources and the characteristics of computing tasks, we find that different tasks have varying demands for resources and time constraints. In this context, the real-time system described in our paper offers a perspective by highlighting the dynamic nature of such environments. Unlike traditional real-time systems, it underscores how tasks are not only generated in real-time but also how the various resources of the nodes change dynamically over time [1–3]. This understanding allows us to fully leverage the multi-level heterogeneous characteristics of computing resources in distributed energy power systems, thus more effectively meeting these differentiated demands and improving the completion rate of real-time tasks. For example, energy data collection and processing tasks have lower computing resource demands but more urgent time constraints; energy storage control tasks have higher computing resource demands but more relaxed time constraints; and energy-precise prediction and regulation tasks have both high computing resource demands and urgent time constraints. For time-sensitive computing tasks, distributing them to lower-level edge nodes closer to users, the computing pressure on the computing centers can be dispersed, and task latency can be reduced. For tasks with high computing resource demands and relaxed time constraints, handing them over to higher-level data centers for execution, efficient resource utilization can be ensured. For tasks that require high computing power and have urgent time constraints, task splitting is needed. Considering the divisibility of tasks, they are decomposed into several subtasks, harnessing the scattered heterogeneous computing resources in the distributed energy power system to collaboratively complete the entire task.

Therefore, researching and designing computing task-scheduling algorithms in distributed energy power systems is of great importance. It meets the differentiated resource and time requirements of computing tasks within these systems and improves the completion rate of real-time tasks.

Researchers have extensively studied the computing task-scheduling problem across different computing environments. Singh et al. explored the scheduling problem in homogeneous nodes [4]. Fizza et al. introduced the deadline requirements of tasks to enhance the timeliness and accuracy of the scheduling strategy [5]. Yang et al. proposed an energy-saving computing framework that demonstrates how to share computing resources among multiple neighboring assisting nodes to determine the optimal scheduling decision for task nodes, thereby optimizing resource utilization and improving energy efficiency [6]. Furthermore, Yang et al. expanded the research field by proposing a real-time algorithm named DEBTS, aimed at achieving systemic performance balance while focusing on two key indicators: service latency and energy consumption [7]. However, these studies did not consider the heterogeneous characteristics of computing nodes.

In the work by Ale et al., the authors considered a single-layer mobile edge computing (MEC) system aimed at dynamically processing computing tasks generated by Internet of Things (IoT) devices in a time-varying operational environment with various requirements [8]. Chen et al. discussed a distributed computation offloading strategy in a vehicular edge computing environment based on the Deep Q-learning Network (DQN), focusing solely on a single-layer network structure, specifically vehicle-to-vehicle (V2V) [9]. Elgendy et al. addressed the simultaneous computation offloading problem in mobile edge computing (MEC) systems using the reinforcement learning algorithm Q-learning to reduce the total overhead of time and energy [10]. Zhang et al. proposed a task-offloading method

for Internet of Vehicles (IoV) edge computing based on deep reinforcement learning, aimed at achieving lower task delay and energy consumption [11]. Karimiafshar et al. proposed a dynamic request scheduling algorithm that minimizes energy consumption and timeliness through the Lyapunov optimization technique [12]. Li et al. presented an adaptive queue weight (AQW) resource allocation and real-time offloading technique in a heterogeneous computing environment [13]. Adhikari et al. worked on reducing the waiting time for latency-sensitive tasks and minimizing the starvation of low-priority tasks by using multi-level feedback queues [14]. Although these studies have made significant progress, they mostly focus on single-layer or large-scale computing node scheduling strategies, which do not apply to the multi-level architecture of distributed energy power systems.

Progress has also been made in multi-level network scheduling. Zhang and Yu proposed an artificial bee colony algorithm to solve the task-collaborative offloading problem in multi-layer edge networks [15]. Chekired et al. proposed a multi-tier fog cloud architecture and classified tasks into low-priority and high-priority [16]. Wang et al. and others proposed task-offloading strategies for multi-level heterogeneous architecture [17–20]. Zhou et al. considered a hierarchical network hybrid computational offloading scheme, allowing edge users to offload workloads through various communication methods [21]. Ren et al. provided a solution named HT3O to solve the scheduling problem in large-scale UAV-assisted MEC systems in dynamic environments, introducing the concept of multi-level network scheduling [22]. These studies are dedicated to minimizing response time, providing a new direction for the development of multi-layer scheduling strategies. However, the tasks targeted by the above work do not have deadlines, failing to address the real-time scheduling problem of time-critical tasks in multi-level distributed energy power systems.

Based on the above analysis, to our knowledge, few studies have considered the hierarchy of three or more layers of heterogeneous networks while also taking into account the deadlines of real-time tasks. In summary, the real-time computing task-scheduling problem in multi-level distributed energy power systems is a challenging issue. This paper aims to improve the overall performance of distributed energy power systems through real-time computational task scheduling and designs an efficient real-time computing task-scheduling algorithm.

This paper is organized as follows: Section 2 presents the system model, including the modeling of multi-level heterogeneous computing power networks in distributed energy systems, real-time tasks, and scheduling problems. Section 3 proposes our solution to the scheduling problem, which includes a scheduling framework for real-time tasks and introduces a hierarchical selection algorithm to fully utilize the properties of hierarchical networks. On this basis, scheduling strategies based on greedy algorithms and genetic algorithms are presented. Section 4 covers the model simulation and results analysis, comparing our solution with other baseline algorithms, and discussing the impact of task load, node delay, and changes in network topology on the success rate of real-time task scheduling. Section 5 summarizes the conclusions and outlines future prospects.

2. System Model

This section first models the multi-level heterogeneous computing network in the distributed energy power system. It then models real-time tasks based on their storage, communication, computing, and deadline requirements. Subsequently, a model for the real-time task-scheduling problem is proposed, transforming the real-time task-scheduling issue into a MINLP (mixed-integer nonlinear programming) problem.

2.1. Network Model

The computing network in distributed energy power systems exhibits a multi-level structure, with the central cloud at the top and edge nodes close to users at the bottom. The higher the level of the node, the richer its storage, communication, and computing resources, but the higher its latency due to distance from users. Conversely, nodes closer to

the bottom have fewer resources but lower latency. Additionally, the computing network is characterized by heterogeneity, not only between different levels but also within the same level, specifically manifested as the variance in various resources. In this paper, the N set represents all computing nodes, N^k represents the set of nodes at level k, and N_n represents the n-th computing node. A virtual node N_0 is introduced, assumed to have infinite resources and zero latency. If a task is scheduled to N_0, it indicates scheduling failure. Each computing node possesses heterogeneous computing resources, and this paper considers the joint scheduling of computing, storage, and communication resources, with each computing node defined as a quadruple:

$$N_n = (S_n, B_n, C_n, D_n) \tag{1}$$

For node N_n, S_n represents the node's storage resources, B_n represents the node's network bandwidth, and the bandwidth attribute of a node is described as a resource that supports task execution, similar to storage resources. This means that bandwidth is one of the resources consumed during task execution. C_n represents the node's computing rate, and D_n is the average time delay experienced when transmitting a task from the user to the node. According to the multi-level nature: if $N_n \in N^k, N_j \in N^{k+1}$, then $S_n < S_j$. $B_n < B_j$, $C_n < C_j$, $D_n < D_j$.

2.2. Task Model

Each user may initiate one or more tasks, represented by the set U, with U_u representing the u-th task. Each task is considered to have simultaneous storage, communication, and computing demands, and each real-time computing task has a deadline. Additionally, considering a sequence of task flows, each task also has an attribute of arrival time, which is uncertain. Therefore, each task is modeled as a quintuple, as follows:

$$U_u = (s_u, b_u, o_u, d_u, a_u) \tag{2}$$

For task U_u, s_u represents the storage resources required to execute the task (in MB), b_u represents the communication resources required (in Mbps), o_u represents the amount of computation (in million instructions, or MI), d_u represents the task's deadline, and a_u represents the task's arrival time. A workflow may include multiple tasks, with possible dependencies between tasks. For example, if task A depends on task B, task A may require the output of task B as input, and the start time of task A must be later than the end time of task B. Tasks form a Directed Acyclic Graph (DAG), termed as a task group.

2.3. Problem Model

The scheduling problem involves assigning certain tasks to appropriate computing nodes, using a set of 0/1 variables to indicate which computing nodes a task is scheduled to. y_{nu} represents whether node n processes task u, with 0 indicating no and 1 indicating yes. However, for some tasks with high demands for computing, bandwidth, and storage resources, and with tight time constraints, it is not feasible to schedule them to high-level nodes or a single bottom-level node because high-level nodes have higher latency and low-level nodes do not have enough resources. Therefore, these tasks need to be split into subtasks for execution by lower-level nodes. Tasks or subtasks must satisfy the following constraints. As specified in Equation (3), each task should be served by at least one node.

$$\sum_{n \in N} y_{nu} \geq 1, \quad u \in U \tag{3}$$

Each task can be split into subtasks for multiple nodes to execute; that is, each node executes a certain proportion of the task. r_{nu} represents the proportion of task u processed by node n. We assume that tasks can be arbitrarily split [20,23,24]. For example, in the context of a distributed energy system, tasks such as inspecting equipment like transformers may involve data from hundreds or even thousands of devices. In this case, we consider that

such a task can be arbitrarily divided. Only when a task is assigned to a node can that node process the task, Thus, the feasibility constraint is given by Equation (4), for $u \in U, n \in N$, as follows:

$$r_{nu} = \begin{cases} 0 & \text{if } y_{nu} = 0, \\ x & \text{if } y_{nu} = 1 \text{ and } 0 < x \leq 1, \end{cases} \tag{4}$$

Assuming the communication and storage resources required for the split subtasks are proportional to r_{nu}. Each subtask will consume a part of the computing node's storage and communication resources, but will not exceed the node's total storage and communication resources. The constraints for storage and communication resources are specified in Equations (5) and (6):

$$\sum_{u \in U} r_{nu} s_u \leq S_n \quad n \in N \tag{5}$$

$$\sum_{u \in U} r_{nu} b_u \leq B_n, \quad n \in N \tag{6}$$

Every task should be executed completely, as specified in Equation (7):

$$\sum_{n \in N} r_{nu} = 1 \tag{7}$$

Each node can serve multiple tasks, and p_{nu} represents the proportion of computing power allocated by node n to task u, which can be envisioned as each node having a multi-core CPU. The computing power allocated for tasks should not exceed the total computing power of the node, as specified in Equation (8):

$$\sum_{u \in U} p_{nu} \leq 1 \quad n \in N \tag{8}$$

The feasibility constraint is given by Equation (9), for $u \in U, n \in N$, as follows:

$$p_{nu} = \begin{cases} 0 & \text{if } y_{nu} = 0, \\ x & \text{if } y_{nu} = 1 \text{ and } 0 < x \leq 1, \end{cases} \tag{9}$$

This paper studies the scheduling problem of real-time computing tasks, considering the tasks' deadlines. Since the split subtasks are executed in parallel, the total completion time should equal the completion time of the latest subtask, as specified by Equations (10) and (11), for $u \in U, n \in N$:

$$\theta_{nu} = \begin{cases} \frac{r_{nu} o_u}{p_{nu} C_n} + D_n & \text{if } y_{nu} = 1, \\ 0 & \text{if } y_{nu} = 0 \end{cases} \tag{10}$$

$$\theta_u = \max_{n \in N} \theta_{nu} \tag{11}$$

where θ_{nu} represents the execution time for node n to process task u, θ_u represents the total completion time for task u, and D_n is the node's inherent delay.

Each task needs to wait in the queue for some time before being scheduled, a_u represents the arrival time of task i, and w_u represents the waiting time of the task in the queue. The completion time of each task should be less than its given deadline, otherwise, it is considered a scheduling failure, as specified by Equation (12).

$$a_u + w_u + \theta_u \leq d_u \tag{12}$$

The goal of real-time task scheduling is to maximize the scheduling success rate. The virtual node N_0 has infinite computing power and zero latency. If a task cannot be scheduled before its deadline, it is handed over to the virtual node. If a task is scheduled to N_0, it represents task failure. Therefore, the goal of the real-time task-scheduling problem is

transformed into minimizing the number of tasks scheduled to the virtual node, as specified by Equation (13), the variable y_{N_0u} indicates whether task u is scheduled on node N_0.

$$\min \sum_{u \in U} y_{N_0 u} \tag{13}$$

Because the calculation of delay involves division and maximization, the model proposed is classified as a mixed-integer nonlinear programming (MINLP) problem, which is NP-hard. This complexity persists even with a small number of tasks and nodes, making it challenging to obtain an exact solution. In Section 3, we introduce the layered selection algorithm along with two scheduling algorithms, aiming to derive sufficiently good approximate solutions. The scheduling algorithm detailed in this paper could be augmented by integrating with node-level scheduling algorithms, such as the earliest deadline first (EDF) algorithm. This combination can improve task allocation efficiency and resource utilization.

3. Scheduling Algorithms

This section first introduces a hierarchical real-time computing task-scheduling framework for distributed energy power systems, incorporating an orchestrating agent (OA) as the execution environment for the scheduling module. Following this, a hierarchical selection algorithm is proposed to choose the appropriate network layer for real-time tasks. Based on this, scheduling algorithms based on both a greedy strategy and a genetic algorithm are subsequently introduced to accomplish task scheduling.

3.1. Real-Time Task-Scheduling Framework

This paper proposes a layered real-time task-scheduling framework node model, as shown in Figure 1, where each node is equipped with an OA. The OA serves as the execution environment for the scheduling algorithm, responsible for collecting information from all nodes and the tasks received by the node, as well as implementing the scheduling plan output by the scheduling algorithm.

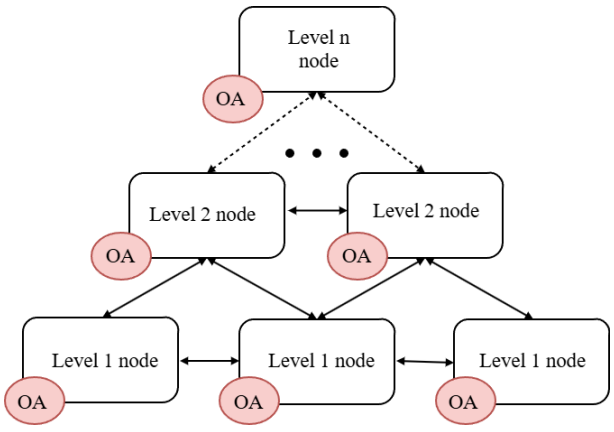

Figure 1. Scheduling framework.

The scheduling process is shown in Figure 2. Each OA possesses several of the modules. The OAs in the first-level nodes have a task waiting queue for accepting tasks, while a higher-level decision node has a scheduling module responsible for running the scheduling algorithm. The information communication module is responsible for communication between nodes and can utilize real-time communication protocols such as MQTT (message queuing telemetry transport). As shown in Figure 1, nodes within the same level and nodes between adjacent levels can communicate directly, while nodes between non-adjacent levels can communicate through relay nodes in intermediate levels. When a user task arrives, it will be handed over to the OA of a bottom-layer node closer to the user, adding it to the OA's waiting queue.

We divide time into multiple short intervals, and only one scheduling operation is completed within each interval. At the beginning of each interval, a decision node performs information synchronization through an information communication module. This involves obtaining information from all nodes and from tasks that are in the waiting queues. Since the data volume of synchronization information is much smaller than that of the tasks, the synchronization time can be negligible.

Once synchronization is complete, the node information collected by the decision node can be considered valid for that time interval. All scheduling decisions for tasks with pending assignments are made by the scheduling module within the OA of the decision node. The scheduling results are then distributed to the nodes receiving the tasks. Finally, the tasks are forwarded to the target nodes for execution through the scheduling result execution module. Because the scheduling module is activated periodically, and although some tasks may finish early within a given period, the freed resources will be utilized in the next period. Since the intervals are short, the waste of resources caused by this can be considered negligible.

Upon receiving a task group with dependencies, the receiving node will maintain the in-degree of each task within the group. At the start of each time period, the node will only synchronize the task information of those tasks with an indegree of 0 to the decision node. After receiving notification that a task has been completed, the receiving node will adjust the indegrees of the remaining unscheduled tasks based on the dependency graph, and then synchronize the information of tasks with an indegree of 0 to the decision node in the next moment, continuing until all tasks in the group are completed or a task times out.

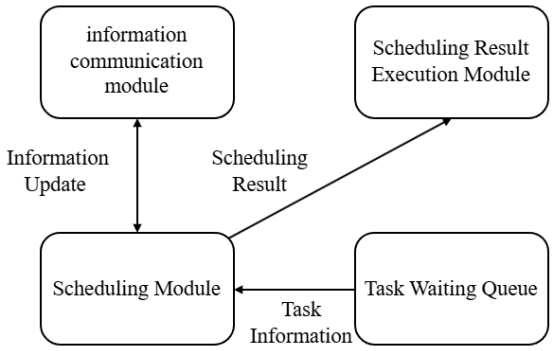

Figure 2. Scheduling process.

3.2. Layered Selection Algorithm

The input to the scheduling algorithm consists of tasks and node information from the scheduling queue, and its output is the scheduling plan, i.e., a set of tasks that should be allocated to which computing nodes for execution. The problem proposed in this paper is a MINLP problem, which is an NP-hard problem, it is impossible to obtain the optimal solution within a reasonable time when the number of nodes and tasks is large. For the real-time scheduling problem in a distributed energy power system computing network, which involves a large number of tasks and multiple nodes.

The scheduling algorithm needs to fully utilize the multi-level heterogeneous computing network resources to meet the diversified needs of tasks. This paper decomposes the scheduling process into two steps, first utilizing the hierarchical nature of the computing network to propose a layered selection algorithm, attempting to allocate tasks to appropriate levels, and then running the scheduling algorithm, as described in Sections 3.3 and 3.4, for the allocated level.

Referring to Algorithm 1 in the pseudo-code, the layered selection algorithm initially extracts tasks from the waiting queue one by one, while updating each task's waiting time. Then, the algorithm checks whether the task can start scheduling. If the task has already exceeded its deadline, then the task fails.

The decision node's OA has a scheduling queue for each layer of the computing network. Tasks are first added to the scheduling queue of the highest layer of the node for scheduling. Since the higher-level nodes have more resources while the lower-level nodes have fewer resources, the algorithm prefers to assign tasks to higher-level nodes. This paper assumes that each task can only be processed by nodes on the same layer. For each layer's scheduling queue, the specific scheduling algorithm (see the following two subsections) is run. If a task cannot be allocated to a suitable node at that layer, i.e., it cannot be completed before its deadline, then the task is passed to the scheduling queue of the next layer. The scheduling between layers is independent of each other. If the task still cannot be completed by the lowest layer's node, then the task is re-added to the waiting queue.

Algorithm 1: Layered selection algorithm for task scheduling

Input: Tasks and node information
Output: Scheduling plan
Initialize scheduling queues for each level of the computing network;
Initialize *schedulableTasks* as empty;
for *each task j in the waiting queue* **do**
    Update the waiting time of task *j*;
    if *task j times out* **then**
        Mark task *j* as failed execution, **continue**;
    end
    Add task *j* to *schedulableTasks*;
end
for $k \leftarrow$ *number of network levels* **to** 1 **do**
    Add *schedulableTasks* to the scheduling queue of level *k*;
    Execute the scheduling algorithm on nodes of the computing network at level *k*;
    set *schedulableTasks* as empty;
    schedulableTasks $\leftarrow$ tasks that failed to schedule at level *k*;
end
Re-add tasks in *schedulableTasks* to the waiting queue;

3.3. Scheduling Algorithm Based on the Greedy Strategy

The greedy algorithm is a method used to solve optimization problems. It makes the optimal choice at each step, representing an efficient algorithm. In the context of task scheduling, this algorithm aims to achieve overall resource allocation and scheduling optimization through a series of locally optimal choices. The core idea of the greedy algorithm is that each choice is the best under the current state, meaning that at each decision point, the algorithm selects the option that is most beneficial to the current state, without considering future states and consequences.

As shown in Algorithm 2, the greedy algorithm sorts the tasks in the scheduling queue from the nearest to the farthest deadline. If the deadlines are the same, tasks with smaller volumes have priority. The design of the sorting strategy is based on the following analysis: (1) Prioritizing tasks that are not urgent may cause urgent tasks to miss their deadlines. (2) If tasks with larger volumes are prioritized, these larger tasks will occupy more time, causing smaller tasks to miss their deadlines. Next, the algorithm sequentially selects an appropriate node at that layer for each task according to the sorted order, preferably choosing nodes with more remaining computing resources to ensure a balanced load of computing resources among the nodes. The allocation ratio of the computing power of nodes follows Equation (14), where, $\{u' \mid u' < u\}$ represents all tasks before task u.

$$p_{nu} = 1 - \sum_{\{u' | u' < u\}} p_{nu'} \tag{14}$$

Algorithm 2: Scheduling algorithm based on the greedy strategy

Input: The scheduling queue and node information for this layer
Output: The scheduling scheme for this layer

```
Sort tasks by deadline and workload;
Add nodes to the priority queue q;
for each task in order after sorting do
    while the task has not met resource requirements do
        if q is empty then
            The task scheduling fails, break;
        end
        Pop a node n from q;
        if n's computation capability is zero then
            continue;
        end
        Allocate computing resources according to Equation (14);
        Allocate computation ratio according to Equation (15);
    end
    if task scheduling is successful then
        Output the scheduling scheme for this task;
        Update node information;
    end
    Re-add the popped node to q;
end
```

If a task cannot be executed by a single node, the task's division needs to be considered. Each node should take on as much of the task's computing requirements as possible, and the proportion of the current task completed by each node is calculated using Equation (15).

$$r_{nu} = \min\{\frac{S_n}{s_u}, \frac{B_n}{b_u}, \frac{(d_{ur} - D_n)C_n}{o_u}, r_{nur}\} \tag{15}$$

where the first term in the parentheses represents the proportion of the remaining storage resources of the current node to the storage resources required by the task. The second term represents the proportion of the remaining communication resources of the current node to the communication resources required by the task. The third term represents the maximum proportion of the task that the current node can complete before the deadline of the task. Here, d_{ur} represents the remaining time of the task, equal to the deadline d_u minus the current time. The fourth term, r_{nur}, represents the proportion of unallocated computing task volume remaining after the task volume distribution by previous nodes. The minimum value of the four terms is the proportion of the task completed by the current node.

The complexity of the algorithm is $O(NM + N \log N)$, where N represents the number of nodes, M represents the number of tasks, and $N \log N$ denotes the complexity of sorting.

3.4. Scheduling Algorithm Based on Genetic Algorithm

The genetic algorithm transforms the solution of a problem into chromosomes and simulates natural selection, iteratively approaching the optimal solution [25–27]. The genetic algorithm mainly consists of initialization, selection, crossover, and mutation steps. Among these, the fitness function plays the role of natural selection. The lower the fitness function value, the closer it is to the problem's optimal solution and, thus, the corresponding chromosome should have a higher probability of mating and producing more offspring. The algorithm design is detailed below. Algorithm 3 is the pseudocode for the genetic algorithm.

Algorithm 3: Scheduling algorithm based on the genetic algorithm

Input: The scheduling queue and node information for this layer
Output: The scheduling scheme for this layer

```
Set the total population set as P Initialize population P;
for i = 1 to iteration_num do
    Calculate the fitness of chromosomes in population P (see Section 3.4.3);
    Set new population P' to empty;
    Retain a certain number of elites to add to P';
    for j = 1 to (population_size − n)/2 do
        Select a pair of chromosomes C1 and C2 using tournament selection;
        C1 and C2 crossover to produce offspring C1' and C2';
        Mutate C1' and C2' separately, add to P';
    end
    Set P = P';
end
Extract the chromosome C with the lowest fitness;
Construct a feasible solution for C (see Section 3.4.5);
Translate the feasible solution into a scheduling plan and output;
```

3.4.1. Chromosome Construction

The first step is to translate the problem's solution into the expression of a chromosome. Each chromosome contains several genes, each representing a node, with each gene recording all task allocation schemes for the corresponding node. Notably, the last gene corresponds to a virtual node, recording tasks that failed to be scheduled. Each task scheme includes the task's information, the completed task proportion r_{nu}, and the allocated computing resource proportion p_{nu}. Through this method, the solution to the problem is encoded, with each chromosome corresponding to a solution to the scheduling problem.

3.4.2. Population Initialization

Initially, each task is simply assigned to a random node (including the virtual node). Then, the r_{nu} in the scheme is set to 1 (meaning task splitting is not considered initially), as each node can perform multiple tasks, and initially, the computing power ratio p_{nu} is evenly distributed among tasks within the same node.

3.4.3. Fitness Function

Given the model's numerous constraints, some constraints need to be transformed into a fitness function. If a constraint is violated, a penalty will be given in the fitness function. Firstly, each node will tally the total storage, communication, and computing resources occupied by the allocated tasks. δ_1 indicates the penalty for violating resource constraints, increasing by 1 for each task that violates these constraints.

δ_2 represents the penalty for tasks not completed within their deadline. For each task, Δt represents the difference between the task's projected completion time and its deadline. If Δt is less than 0, meaning the task is completed within the deadline, no penalty is given. If δ_2 is greater than 0, indicating the task exceeded its deadline, a penalty is needed. Moreover, the more the time exceeded, the larger the penalty. A sigmoid function is used to achieve this effect (Equation (16)), where this penalty is smaller than the penalties for resource constraints and task completion constraints, hoping the genetic algorithm can make as many attempts as possible (even though it may exceed the task's time requirements).

$$\delta_2 = -0.2 + \frac{1}{1 + e^{-\Delta t}} \tag{16}$$

δ_3 represents the penalty for scheduling failure, referring to tasks assigned to the virtual node. Similar to the idea in the greedy algorithm, if a task is more urgent, the penalty for scheduling failure is larger; conversely, if a task's deadline is more lenient, the penalty is smaller. An exponential function is used to achieve this effect (see Equation (17)). For each task in the virtual node, where 'now' represents the current moment, and d_i is the task's deadline.

$$\delta_3 += 0.5 + 2^{-(di-now)} \tag{17}$$

Ultimately, the fitness of the chromosome is denoted by δ, calculated by Equation (18). The smaller δ is, the more adapted the chromosome is to the environment, and the more likely it is to mate and produce offspring.

$$\delta = \delta_1 + \delta_2 + \delta_3 \tag{18}$$

3.4.4. Genetic Operators

The selection operator in this paper uses the tournament selection method as well as elite selection. The crossover operator employs k-point crossover. Two gene loci are randomly selected, exchanging all genes between these two loci on the two chromosomes, essentially swapping the scheduling schemes for the same node regarding the chosen loci. Due to the unique encoding method of the genetic algorithm designed in this paper, careful design of the mutation operator is needed to match the background of the problem to be solved. Firstly, there is the computing power allocation mutation. Each node will randomly mutate the computing power allocation ratio p_{nu} within the range of $(0, 1]$ for the tasks currently assigned to it, which can increase or decrease. Next is the split mutation. As previously mentioned, when a single node cannot complete a task, it needs to be split. Upon mutation, a task is divided into two equally sized sub-tasks A and B, with A assigned to the node and B assigned to another node (non-virtual). At this point, the computing power ratio p_{nu} is randomly reallocated for the two sub-tasks. Tasks in the virtual node cannot undergo split mutation. Note that through sufficient iterations, any proportion of task division can be achieved. The design of this mutation method actually uses a binary approach to approximate the optimal split ratio. Moreover, this splitting naturally meets the task completion constraint (Equation (7)). Lastly, there is the transfer mutation, which randomly assigns a task to another node. After being transferred to another node, the computing power ratio is also randomly reallocated.

3.4.5. Constructing a Feasible Solution

Chromosomes must be converted into solutions. Since these solutions might violate constraints, they need to be modified to satisfy these constraints, thereby constructing a set of feasible solutions. Initially, tasks are assigned to the virtual node, which directly indicates that the task has failed to be scheduled. Then, each node checks if any task violates the node's resource constraints. If the required computing power ratio exceeds the node's maximum computing power ratio, then the allocation will be readjusted to meet the computing resource constraints, evenly distributing the remaining computing power among the remaining tasks. If communication, storage, or time constraints are violated, then the task fails, releasing the occupied resources and transferring them to the next task. Due to task splitting, if any sub-task of a task fails, then the entire task is considered failed. Through this construction of feasible solutions, solutions that meet constraints are created and then handed over to the OA for scheduling.

3.4.6. Complexity Analysis

The complexity of the algorithm is $O(CPL + CP \log P)$, where C represents the number of iterations, P represents the population size, and L represents the encoding length of each chromosome (related to the number of tasks and nodes). $CP \log P$ represents the sorting complexity in elite selection.

4. Simulation and Results Analysis

This section first introduces the experimental and simulation environment, then analyzes and compares the scheduling success rates of the proposed algorithm and related comparison algorithms under changes in task load, node latency, and network structure.

4.1. Experiment and Simulation Environment

The experiments were conducted on an AMD 8-core, 3.2 GHz, 16 GB server, using Java jdk1.8 for coding.

The experimental environment simulates a four-layer hierarchical network, where the top layer (fourth layer) is a central cloud node. The model and algorithms can be extended to any number of layers. There are eight level-one nodes, four level-two nodes, one level-three node, and one level-four node. The latency from users to level-one nodes is 2 ms, to level-two nodes is 6 ms, to level-three nodes is 12 ms, and to level-four nodes is 137 ms. The storage resources of level-1 nodes range from 1 GB to 8 GB, communication resources range from 10 Mbps to 200 Mbps, and computing resources range from 1000 MIPS to 2500 MIPS, with the capabilities of nodes at other levels detailed in Table 1.

Table 1. Node Attributes.

Layer	Storage/GB	Bandwidth/Mbps	Computation/MIPS	Latency/ms
1	1, 8	200, 500	1000, 2000	2
2	10, 50	200, 500	2500, 4000	6
3	500, 800	500, 800	6000, 10,000	12
4	2000, 3000	5000, 10,000	70,000, 100,000	137

Each task requires storage resources ranging from 0.15 GB to 2.5 GB, communication resources ranging from 0.1 Mbps to 100 Mbps, and computational workload ranging from 100 MI to 8500 MI, with deadlines ranging from 0 to 200 ms.

The population size of the genetic algorithm is fixed at 1500, with an elitism rate set to 0.3, computing power allocation mutation rate set to 0.3, splitting mutation rate set to 0.2, transfer mutation rate set to 0.3, and the number of iterations set to 100 generations. In the experiment, each attribute of every task is independently and uniformly randomly selected from a given range.

4.2. Result Evaluation

This paper compares the layered real-time task-scheduling algorithm based on the greedy strategy (abbreviated as greedy) and the layered real-time task-scheduling algorithm based on the genetic algorithm (abbreviated as genetic) with the longest time first (LTF) and workload first (WLF) algorithms, analyzing the scheduling success rates of each algorithm under different scenarios. The scheduling success rate (SR) refers to the ratio of successfully scheduled tasks to the total number of tasks; we conducted ten independent experiments and calculated the average SR from these experiments.

The LTF algorithm prioritizes tasks with the longest deadlines, assigning them to the fastest computing nodes. The WLF algorithm prioritizes tasks with smaller workloads, assigning them to the fastest computing nodes to minimize response time.

Figure 3 shows the change in scheduling success rates of the four algorithms as the task load increases. As illustrated in the figure, the scheduling success rates of all four algorithms decrease as task load increases, due to resource contention among multiple tasks, causing some tasks to not be completed within their deadlines. However, compared to the LTF and WLF algorithms, the greedy and genetic algorithms demonstrate superior performance. Both consider task splitting, allowing tasks with high resource demands and urgent deadlines to be completed on time. Additionally, unlike LTF, which schedules tasks in descending order of deadlines, the greedy algorithm does so in ascending order, preventing urgent tasks from missing their deadlines due to long waits. The WLF algorithm

performs slightly better than LTF, scheduling tasks in ascending order of computational requirements, preventing small tasks from missing their deadlines while waiting for large tasks to complete.

Figure 3 also shows that the genetic algorithm performs similarly to the greedy algorithm when the task volume is 50 and 100, but outperforms the greedy algorithm when the task volume reaches 150. As the task volume continues to increase, the genetic algorithm surpasses the greedy algorithm. This is because when the task volume is below 150, the computing tasks relative to node resources are not fully saturated, so both algorithms perform well. As the task volume increases to 250, the greedy algorithm shows significant advantages. This is due to the increasing solution space of the Genetic algorithm with the rise in task volume, making it challenging for the Genetic algorithm to find the optimal solution within a limited number of iterations. Meanwhile, the greedy algorithm is less sensitive to increases in task volume, thus maintaining good performance under high task load conditions. In summary, for smaller task loads (as in the experimental setup of this paper with 150 tasks), the genetic algorithm demonstrates superior performance. In contrast, the greedy algorithm is less sensitive to increases in task volume, making it more suitable for handling larger task loads.

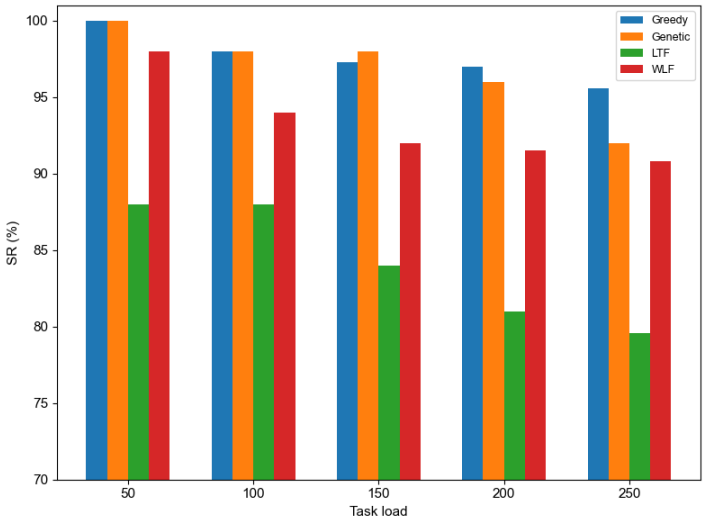

Figure 3. Scheduling framework.

Figure 4 the impact of node delay changes on scheduling success rates. By fixing the number of tasks at 100 and multiplying each node's delay by a delay coefficient ranging from 1 to 5, the greater the coefficient, the greater the transmission delay between nodes. As shown in Figure 4, the scheduling success rates of all four algorithms decline as node delay increases. This is because the rise in node delay causes more tasks to be unable to be executed and completed before their deadlines. The LTF algorithm, scheduling tasks in ascending order of deadlines, causes some urgent tasks to miss their deadlines due to long waits. The WLF algorithm performs slightly better than LTF, with the greedy algorithm showing excellent performance. The genetic algorithm performs similarly to the greedy algorithm when the node delay coefficient is small, but underperforms as node delay increases. This is due to the rise in node delay preventing high-layer nodes from meeting the requirements of some urgent tasks, leading to more tasks being backlogged to lower-layer nodes, thereby increasing the load on lower-layer nodes and causing a decline in scheduling success rates.

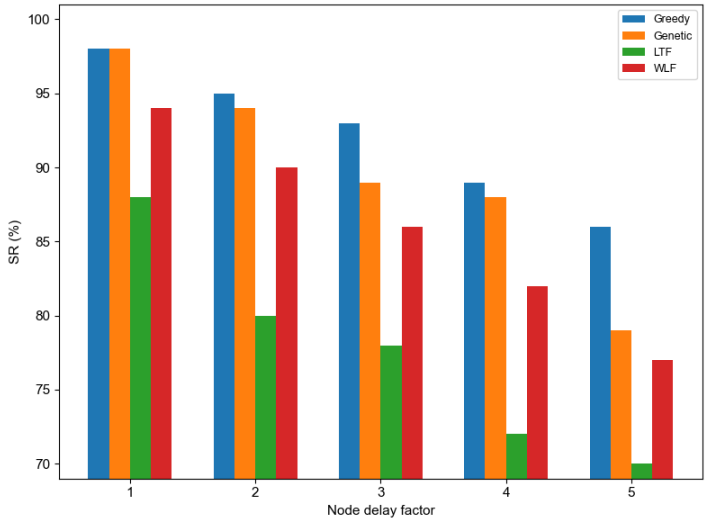

Figure 4. Scheduling framework.

Figure 5 shows the effect of changes in the number of network layers on scheduling success rates. As shown, the scheduling success rates of all four algorithms tend to increase as the hierarchical structure increases. With more low-latency nodes closer to users being added to the network structure, the completion of some urgent tasks becomes possible. When there is only a central cloud (a single-layer network), the scheduling success rates of all four algorithms are at a lower level (52%). This section evaluates the performance of various algorithms under changes in task load, node delay, and network structure. Under various conditions, compared to the baseline algorithm, the layered real-time task-scheduling algorithm based on greedy strategy and the layered real-time task-scheduling algorithm based on the genetic algorithm proposed in this paper demonstrates superior performance. The genetic algorithm performs better than the greedy algorithm when the task volume is small but is inferior under large task volumes.

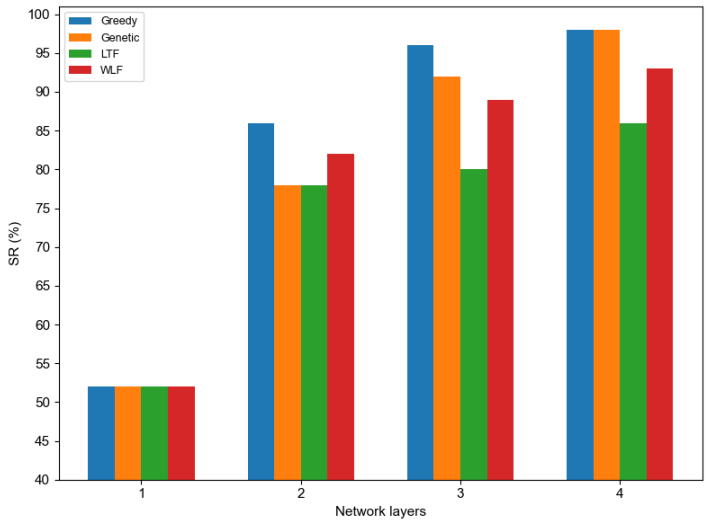

Figure 5. Scheduling framework.

5. Conclusions

This study discusses the problem of real-time computational task scheduling in multi-tier heterogeneous computing networks within distributed energy power systems. This paper models the multi-tier heterogeneous computing network and the real-time computational tasks mathematically, transforming the scheduling problem into a MINLP problem.

It proposes a computing network scheduling architecture based on an orchestration agent; based on a hierarchical selection algorithm, it introduces a task-scheduling algorithm based on a greedy strategy and another based on a genetic algorithm. Experiments show that the proposed algorithms can fully utilize the resources of the multi-tier heterogeneous computing network, meet the diverse needs of tasks, and improve the success rate of real-time task scheduling. We plan to conduct future work experiments in more real-life environments and to verify and enhance the applicability and stability of algorithms through more rigorous theoretical derivation.

Author Contributions: Conceptualization, N.W.; data collection, D.W.; formal analysis, S.J.; investigation, X.B.; writing—original draft preparation, M.Z.; writing—review and editing, J.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Science and Technology Project of State Grid, project No. 5700-202318269A-1-1-ZN.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: X.B. is employed by the State Grid Smart Grid Research Institute Co., Ltd., D.W. and S.J. are employed by the Information and Communication Branch of State Grid Jiangsu Electric Power Co., Ltd., and J.Z. is employed by the State Grid Economic and Technological Research Institute Co., Ltd. The remaining authors, N.W. and M.Z., declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Ku, Y.-J.; Baidya, S.; Dey, S. Adaptive computation partitioning and offloading in real-time sustainable vehicular edge computing. *IEEE Trans. Veh. Technol.* **2021**, *70*, 13221–13237. [CrossRef]
2. Wang, J.; Zhu, K.; Chen, B.; Han, Z. Distributed clustering-based cooperative vehicular edge computing for real-time offloading requests. *IEEE Trans. Veh. Technol.* **2021**, *71*, 653–669. [CrossRef]
3. Wang, X.; Ning, Z.; Wang, L. Offloading in internet of vehicles: A fog-enabled real-time traffic management system. *IEEE Trans. Ind. Inform.* **2018**, *14*, 4568–4578. [CrossRef]
4. Singh, A.; Auluck, N.; Rana, O.; Jones, A.; Nepal, S. RT-SANE: Real time security aware scheduling on the network edge. In Proceedings of the International Conference on Utility and Cloud Computing, Austin, TX, USA, 5–8 December 2017; pp. 131–140.
5. Fizza, K.; Auluck, N.; Azim, A. Improving the schedulability of real-time tasks using fog computing. *IEEE Trans. Serv. Comput.* **2019**, *15*, 372–385. [CrossRef]
6. Yang, Y.; Wang, K.; Zhang, G.; Chen, X.; Luo, X.; Zhou, M.T. Maximal energy efficient task scheduling for homogeneous fog networks. In Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Honolulu, HI, USA, 16–19 April 2018; pp. 274–279.
7. Yang, Y.; Zhao, S.; Zhang, W.; Chen, Y.; Luo, X.; Wang, J. DEBTS: Delay energy balanced task scheduling in homogeneous fog networks. *IEEE Internet Things J.* **2018**, *5*, 2094–2106. [CrossRef]
8. Ale, L.; Zhang, N.; Fang, X.; Chen, X.; Wu, S.; Li, L. Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning. *IEEE Trans. Cogn. Commun. Netw.* **2021**, *7*, 881–892. [CrossRef]
9. Chen, C.; Zhang, Y.; Wang, Z.; Wan, S.; Pei, Q. Distributed computation offloading method based on deep reinforcement learning in ICV. *Appl. Soft Comput.* **2021**, *103*, 107108. [CrossRef]
10. Elgendy, I.A.; Zhang, W.Z.; He, H.; Gupta, B.B.; Abd El-Latif, A.A. Joint computation offloading and task caching for multi-user and multi-task MEC systems: Reinforcement learning-based algorithms. *Wirel. Netw.* **2021**, *27*, 2023–2038. [CrossRef]
11. Zhang, D.; Cao, L.; Zhu, H.; Zhang, T.; Du, J.; Jiang, K. Task offloading method of edge computing in internet of vehicles based on deep reinforcement learning. *Clust. Comput.* **2022**, *25*, 1175–1187. [CrossRef]
12. Karimiafshar, A.; Hashemi, M.R.; Heidarpour, M.R.; Toosi, A.N. An energy-conservative dispatcher for fog-enabled IIoT systems: When stability and timeliness matter. *IEEE Trans. Serv. Comput.* **2021**, *16*, 80–94. [CrossRef]
13. Li, L.; Guan, Q.; Jin, L.; Guo, M. Resource allocation and task offloading for heterogeneous real-time tasks with uncertain duration time in a fog queueing system. *IEEE Access* **2019**, *7*, 9912–9925. [CrossRef]
14. Adhikari, M.; Mukherjee, M.; Srirama, S.N. DPTO: A deadline and priority-aware task offloading in fog computing framework leveraging multilevel feedback queueing. *IEEE Internet Things J.* **2019**, *7*, 5773–5782. [CrossRef]
15. Zhang, W.; Yu, J. Task offloading strategy in mobile edge computing based on cloud-edge-end cooperation. *J. Comput. Res. Dev.* **2022**, *65*, 1–14.

16. Chekired, D.A.; Khoukhi, L.; Mouftah, H.T. Industrial IoT data scheduling based on hierarchical fog computing: A key for enabling smart factory. *IEEE Trans. Ind. Inform.* **2018**, *14*, 4590–4602. [CrossRef]
17. Wang, P.; Zheng, Z.; Di, B.; Song, L. HetMEC: Latency-optimal task assignment and resource allocation for heterogeneous multi-layer mobile edge computing. *IEEE Trans. Wireless Commun.* **2019**, *18*, 4942–4956. [CrossRef]
18. El Haber, E.; Nguyen, T.M.; Assi, C. Joint optimization of computational cost and devices energy for task offloading in multi-tier edge-clouds. *IEEE Trans. Comms.* **2019**, *67*, 3407–3421. [CrossRef]
19. Peixoto, M.L.M.; Genez, T.A.; Bittencourt, L.F. Hierarchical scheduling mechanisms in multi-level fog computing. *IEEE Trans. Serv. Comput.* **2021**, *15*, 2824–2837. [CrossRef]
20. Sun, Y.; He, Q. Computational offloading for MEC networks with energy harvesting: A hierarchical multi-agent reinforcement learning approach. *Electronics* **2023**, *12*, 1304. [CrossRef]
21. Zhou, H.; Long, Y.; Gong, S.; Zhu, K.; Hoang, D.T.; Niyato, D. Hierarchical multi-agent deep reinforcement learning for energy-efficient hybrid computation offloading. *IEEE Trans. Veh. Technol.* **2022**, *72*, 986–1001. [CrossRef]
22. Ren, T.; Niu, J.; Dai, B.; Liu, X.; Hu, Z.; Xu, M.; Guizani, M. Enabling efficient scheduling in large-scale UAV-assisted mobile-edge computing via hierarchical reinforcement learning. *IEEE Internet Things J.* **2021**, *9*, 7095–7109. [CrossRef]
23. Qiu, X.; Zhang, W.; Chen, W.; Zheng, Z. Distributed and collective deep reinforcement learning for computation offloading: A practical perspective. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *32*, 1085–1101. [CrossRef]
24. Zhan, Y.; Guo, S.; Li, P.; Zhang, J. A deep reinforcement learning based offloading game in edge computing. *IEEE Trans. Comput.* **2020**, *69*, 883–893. [CrossRef]
25. Katoch, S.; Chauhan, S.S.; Kumar, V. A review on genetic algorithm: Past, present, and future. *Multimed. Tools Appl.* **2021**, *80*, 8091–8126. [CrossRef] [PubMed]
26. Lambora, A.; Gupta, K.; Chopra, K. Genetic algorithm-A literature review. In Proceedings of the International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), Faridabad, India, 14–16 February 2019; pp. 380–384.
27. Haldurai, L.; Madhubala, T.; Rajalakshmi, R. A study on genetic algorithm and its applications. *Int. J. Comput. Sci. Eng.* **2016**, *4*, 139.