electronics

MDPI

Article

Research on Multi-DAG Satellite Network Task Scheduling Algorithm Based on Cache-Composite Priority

Zhiguo Liu [1,†], Luxi Zhang [1,†], Lin Wang [2,*], Xiaoqi Dong [1] and Junlin Rong [1]

[1] Communication and Network Key Laboratory, Dalian University, Dalian 116622, China; liuzhiguo@dlu.edu.cn (Z.L.); 15242561513@163.com (L.Z.); dongxiaoqi3360@163.com (X.D.); rongjunlin@s.dlu.edu.cn (J.R.)
[2] College of Environment and Chemical Engineering, Dalian University, Dalian 116622, China
* Correspondence: wanglin@dlu.edu.cn
† These authors contributed equally to this work.

Abstract: The problem of multiple DAGs sharing satellite constellation resources has gradually attracted widespread attention. Due to the limited computing resources and energy consumption of satellite networks, it is necessary to formulate a reasonable multi-DAG task scheduling scheme to ensure the fairness of each workflow under the premise of considering latency and energy consumption. Therefore, in this paper, we propose a multi-DAG satellite network task scheduling algorithm based on cache-composite priority under the Software-Defined Networking satellite network architecture. The basic idea of this algorithm lies in the DAG selection phase, where not only are task priorities computed but also the concept of fair scheduling is introduced, so as to prevent the excessively delayed scheduling of low-priority DAG tasks. In addition, the concept of public subtasks is introduced to reduce the system overhead caused by repetitive tasks. The experimental results show that the hybrid scheduling strategy proposed in this paper can meet the demand of DAG scheduling and improve the degree of task completion while effectively reducing the task latency and energy consumption.

Keywords: Software-Defined Networking; satellite network; multi-DAG scheduling; fairness

check for updates

Citation: Liu, Z.; Zhang, L.; Wang, L.; Dong, X.; Rong, J. Research on Multi-DAG Satellite Network Task Scheduling Algorithm Based on Cache-Composite Priority. *Electronics* **2024**, *13*, 763. https://doi.org/10.3390/electronics13040763

Academic Editor: Hideaki IIduka

Received: 4 December 2023
Revised: 22 January 2024
Accepted: 25 January 2024
Published: 15 February 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Sixth-generation technology combines fifth-generation technology with satellite networks to achieve global coverage and is considered a cheap and fast internet technology. The goal of 6G technology is to provide mobile users with multimedia and networked all-weather information services [1–3]. Applying edge computing technology to both ground networks and satellite networks endows the network with characteristics such as low latency, low energy consumption, high speed, and real-time insight into wireless networks [4]. This will undoubtedly be extensively utilized in the architecture of 6G networks. The introduction of edge computing technology into the satellite network, that is, the extension of the cloud computing platform to the network edge, can provide users with heterogeneous computing resources and enable users to obtain computing services anywhere in the world, thereby improving user service experience and reducing redundant network traffic [5].

Due to the limited computing power and energy consumption of Mobile Edge Computing (MEC) servers, only a portion of service requests can be deployed at the same time, and the amount of computing resources allocated to each service is also limited. End users wish to offload as many tasks as possible to satellite edge computing nodes to reduce transmission latency and enhance user experience. To make full use of the resources of satellite MEC servers and enhance user experience, reasonable task deployment algorithms have become a key focus of current research. Wang B et al. proposed a system cost model for Low-Earth-Orbit (LEO) satellite edge computing systems, which can be represented as a mixed-integer nonlinear programming problem. The model decomposes the problem into

computation offloading and resource allocation and then proposes the Joint Computation Offloading and Resource Allocation (JCORA) strategy. Experimental results show that this model can significantly reduce the average cost of a system [6]. Wang Y et al. proposed a game theory approach to optimize the computational offloading strategy in satellite edge computing, constructing a computational offloading game framework. When response time and energy consumption based on computing tasks are taken as indicators of optimized performance, the average cost of equipment can be significantly reduced [7]. Tang Q et al. proposed a hybrid cloud and edge computing LEO satellite (CECLS) network with a three-layer computing architecture and studied the computing offload decision to minimize the total energy consumption of ground users [8]. However, some new applications have emerged that require a large amount of computing resources. For example, virtual reality and HD video require a large amount of computing resources for rendering and video encoding and decoding [9]. Autonomous vehicles rely on large amounts of computing resources for automatic control [10]. Deep learning tasks deployed on IoT devices rely on large amounts of computing resources for model training [11,12]. The above tasks cannot be supported by a single MEC node due to their huge computing loads. These tasks can be optimized and selected for deployment to the remote cloud through algorithms. However, due to the long transmission distance between the end user and the cloud server, the transmission delay is relatively large.

Therefore, when allocating tasks, they are no longer deployed in the remote cloud, but are split into multiple subtasks that are assigned to satellite nodes, which can make full use of the satellite resources, thus significantly reducing the latency of task execution. In this paper, the correlation of subtasks is represented by a directed acyclic graph (DAG), and the DAG graph is always single-input and single-output. The essence of multi-DAG task scheduling is to reasonably arrange subtasks with priority relationships, map them to appropriate nodes for execution, and achieve optimization in terms of resources or execution time. The main focus of this paper is the multi-DAG task scheduling problem within the integrated Software-Defined Networking (SDN) satellite network architecture. In the DAG selection phase, we not only calculate task priorities but also incorporate the concept of fair scheduling to prevent potential excessive delays in the scheduling of low-priority DAG tasks. In addition, in multi-DAG task scheduling, there are some public subtasks, and these public subtasks can reuse the same processing results. In this study, we take advantage of the extensive coverage characteristics of satellite networks to repetitively use these public subtasks, thereby enhancing performance. The main contributions of this paper are as follows:

- In the SDN satellite task deployment network architecture, the idea of splitting the tasks that the MEC server cannot afford into DAG subtasks is proposed.
- The concept of public subtasks is introduced, combined with the SDN controller, adding a public task management module and reducing the system overhead caused by repetitive subtasks, thereby improving resource utilization.
- This paper proposes a multi-DAG satellite network task scheduling algorithm based on composite priority. Compared with the Fairness scheduling algorithm and LB-HEFT scheduling algorithm, when introducing public subtasks, the task completion time and task energy consumption of this algorithm increase by nearly 20%.

The remainder of this paper is organized as follows. In Section 2, the relevant research on DAG task scheduling is introduced. In Section 3, we propose a multi-DAG task scheduling algorithm based on cache-composite priority. In Section 4, we carry out a simulation. Section 5 presents a summary of the article and research prospects.

2. Related Research

Most current research on task scheduling involves splitting tasks into subtasks. Due to the clear dependency relationship between subtasks, the current research mainly focuses on DAG task scheduling. Topcuoglu H et al. proposed a method to generate a scheduling list, arrange the list according to task priority, select the task with the highest priority

for scheduling according to the list, and use a strategy to allocate tasks to appropriate processors in order to minimize the predefined objective function. This method is the most well known HEFT algorithm [13]. Arabnejad H et al. improved the HEFT algorithm by not only considering the completion time of the current task, but also predicting the completion time of the subsequent tasks and including comprehensive considerations to obtain better scheduling decisions [14]. Mahmoud H et al. took load balancing as the optimization objective and proposed the LB-HEFT algorithm based on the HEFT algorithm. This algorithm improved the performance of load balancing by 72.59% [15]. Senapati D et al. used the actual communication time between tasks to replace the average communication time in the task priority evaluation phase and accurately analyzed the impact of heterogeneous data transmission time on task priority. When matching edge nodes for tasks, the influence of the entire DAG task completion time on the current task matching process was explored recursively, and a novel pruning mechanism was developed to prune the search process in order to reduce the time complexity of the algorithm [16]. Hamza Djigal et al. introduced a forward-looking function during the determination of task priorities and processor matching phase. This involved considering the priority of the current task and its immediate successor tasks to adjust the priority of the current task. Additionally, the matching of the current task to a server was guided by evaluating the minimum completion time of the entire task [17]. Zhao Y et al. further extended this forward-looking function by introducing a pre-scheduling phase before determining task priorities and server matching. In this phase, the PEFT algorithm [18] was used to preplan a feasible solution for task scheduling, and this solution was used as a guide to analyze the impact of the entire DAG task completion time on task priority determination and server matching, further reducing the makespan [19]. In addition, evolutionary algorithms are also suitable for solving DAG task scheduling problems. Keshanchi B et al. proposed a DAG task scheduling problem with the optimization goal of processing latency and used a genetic algorithm to solve it [20].

The above papers mainly studied how to schedule a single DAG with multiple resources. These methods have been widely used and are basically mature. However, with the increasing demand for low cost and resource-sharing application systems, the scheduling problem of multi-DAG tasks needs to be solved. Zhao H et al. proposed the well-known Fairness algorithm. The basic idea is to calculate and compare the slowdown values of different DAGs after scheduling tasks, then select the DAG with the largest slowdown value from multiple DAGs, and use the corresponding single-DAG scheduling algorithm for scheduling. This process is repeated until all tasks in all DAGs are scheduled. Experimental results demonstrate that the Fairness scheduling algorithm effectively improves the relative fairness of multiple tasks [21]. Zhu Y et al. proposed an intelligent scheduling method for multi-DAG jobs using deep reinforcement learning, called Smart-mDAG. It is a job-specific scheduling method that adjusts the scheduling policy based on diverse dependencies to minimize the makespan [22]. The above algorithm is suitable for situations where the priorities of individual DAGs are not defined, but in practice, the priorities of individual tasks are often different. Tian G et al. proposed an algorithm called Backfill, which effectively solved this problem. The algorithm was improved based on the Fairness scheduling algorithm, which can simultaneously consider multiple different types of DAG scheduling requirements, including factors such as arrival time and priority [23]. In addition to algorithms that improve fairness, other algorithms can also solve multi-DAG scheduling problems. Zhang Y et al. proposed an online task scheduling optimization method for DAG-based requests and modeled the scheduling procedure as a Markov decision process. A temporal-difference learning-based mechanism was adopted to learn an optimal task allocation strategy at each decision stage [24]. Cai L et al. proposed a fault-tolerant DAG task scheduling algorithm to minimize the response latency experienced by tasks. After formulating the DAG task scheduling problem, a context-aware greedy task scheduling algorithm was proposed. Then, to address edge server failure events, a dependency-aware task rescheduling algorithm was designed [25]. Lin Z et al. proposed a

multi-DAG scheduling algorithm based on reinforcement learning. They first utilized a convolutional neural network with graph attention to fully process and encode scheduling information. Then, a fully connected neural network selected appropriate nodes from the DAG application for execution. Finally, a heuristic approach was employed to repeatedly allocate processors to the selected nodes based on the earliest completion time [26].

In summary, current multi-DAG task scheduling primarily focuses on task execution time and resource utilization. However, unlike ground networks, satellite networks equipped with edge computing capabilities face challenges of limited computational resources and constrained energy (the energy consumption is directly related to the satellite's lifespan). Therefore, when integrating multi-DAG task scheduling with satellite networks, factors such as task latency and energy consumption should be considered. Furthermore, there is currently a lack of research on the multi-DAG task scheduling problem in the context of integrated SDN satellite networks.

3. System Model

3.1. Satellite Network Model

As shown in Figure 1, the SDN satellite task deployment network architecture is composed of an LEO satellite network and a Geostationary-Earth-Orbit (GEO) synchronous satellite network [27]. LEO satellites form a space access network and provide wireless access rights to sparse terminal devices on the ground, forming the forwarding layer of the SDN satellite network. On the other hand, LEO satellites can deploy edge computing nodes to directly provide services to end users, thereby significantly shortening the response delay of terminal devices. Benefiting from their broad coverage characteristics, GEO synchronous satellites can be used as the control layer of the SDN satellite network to maintain network topology and status information and perform routing using the OpenFlow protocol [28]. In the SDN-based satellite network architecture, LEO satellites can quantify their own computing resources and communication resources, uploading the current resource status to the SDN controller [29].

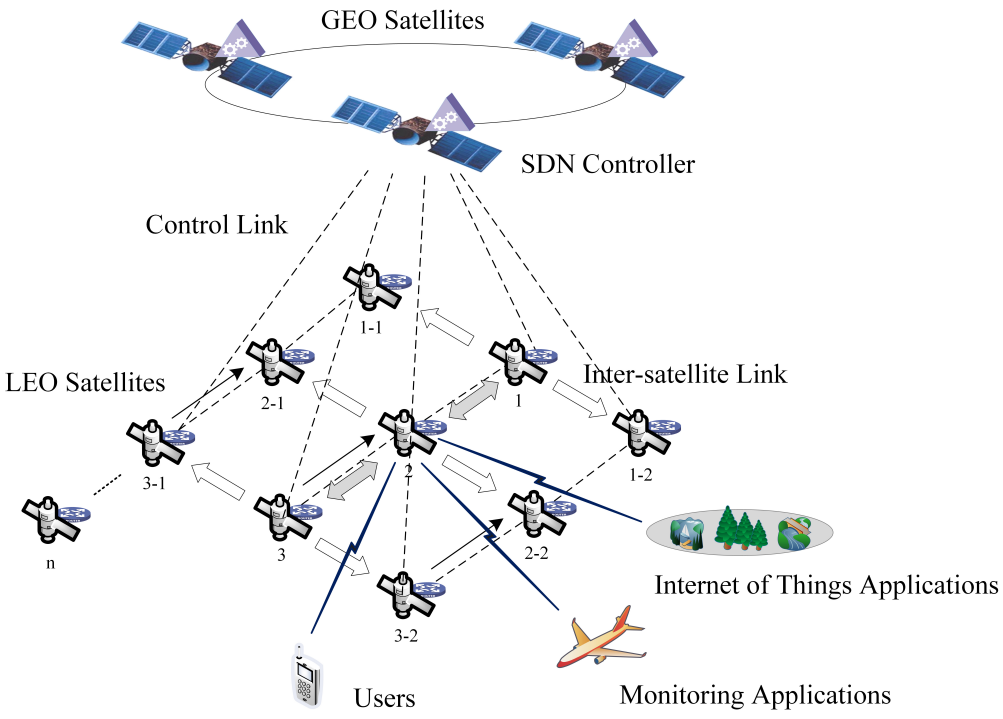


Figure 1. SDN satellite task deployment network architecture.

3.2. Multi-DAG Task Deployment Process

In certain DAG tasks, there are public subtasks that can reuse the same processing results, such as the recognition of remote sensing images for the same geographical location. These public subtasks only need to be executed once, thereby reducing unnecessary resource consumption. To meet the requirements of DAG satellite network task scheduling, public subtasks can be processed uniformly. We deploy a task decomposition module, task deployment management module, and public task management module in the SDN controller. The task decomposition module primarily decomposes incoming tasks, storing the decomposed tasks in the DAG pool. The task deployment management module provides edge computing node monitoring and task deployment functions. The public task management module is primarily used to identify and execute public subtasks in the DAG pool. Due to the extensive coverage of GEO synchronous satellites, they naturally provide an advantage for deploying public subtasks. A schematic diagram of multi-DAG task deployment is shown in Figure 2.

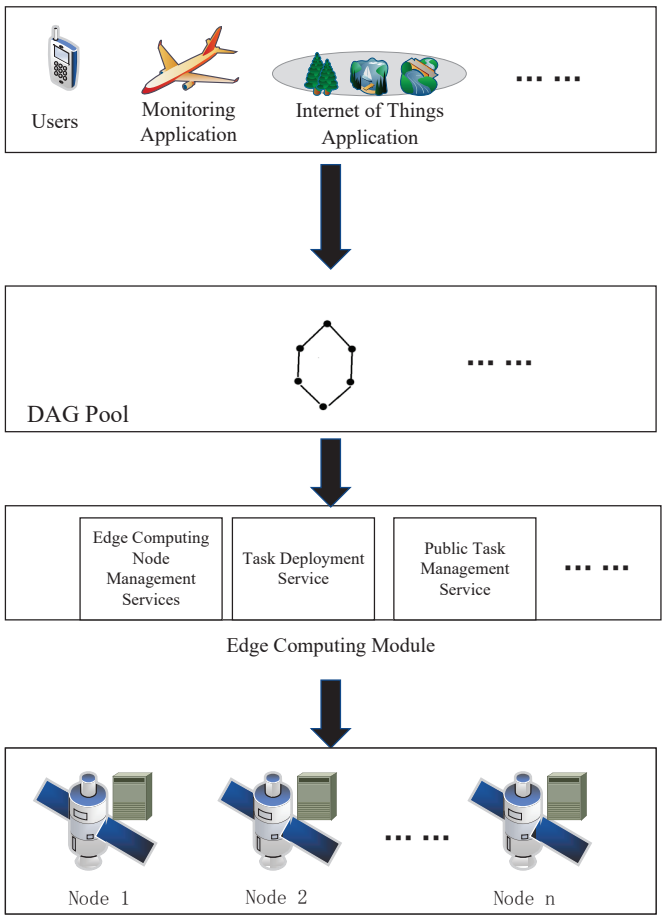


Figure 2. Multi-DAG task deployment diagram.

Multiple different users on the ground send task deployment requests to their corresponding coverage satellites. The receiving satellites uniformly send task information to the SDN controller. The task decomposition module of the SDN controller decomposes the tasks into DAG subtasks and stores them in the current DAG pool. The public task management module in the SDN controller scans the subtasks in the DAG pool. When a subtask has been executed a certain number of times, it is marked, and its execution result is cached on the GEO satellite. This cached result is then made available as a public resource for all tasks. Then, based on the resource information obtained from satellite nodes, the task deployment module schedules multi-DAG tasks through an algorithm to select the optimal deployment locations. The SDN controller sends the corresponding control information

to the LEO satellite and forwards tasks to the specified deployment locations through a flow table.

3.3. Cost Model

Assuming that multiple users on the ground initiate different service requests, these service requests are represented as $a = \{application_1, application_2, \ldots, application_n\}$, where n represents the number of requests. Thus, the set of service request identifiers can be represented as $N = \{1, 2, \ldots, n\}$. As the tasks in the service requests are too large, they need to be divided into multiple subtasks. The relationship between subtasks is represented by a directed acyclic graph (DAG). An example DAG is illustrated in Figure 3. The DAG is represented by the binary tuple $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_s\}$ represents the subtask node set of the DAG, $s = |V|$ represents the total number of subtasks, $E = \{e_{i,j} \mid e_{i,j} =< v_i, v_j >, e_{i,j} \in V \times V\}$ represents the directed edges of the DAG, and each edge connects two different subtasks at its two ends. If $e_{i,j} \in E$, it means that there is a dependency between subtask v_i and subtask v_j, whereby subtask v_j can only be executed after the successful completion of subtask v_i.

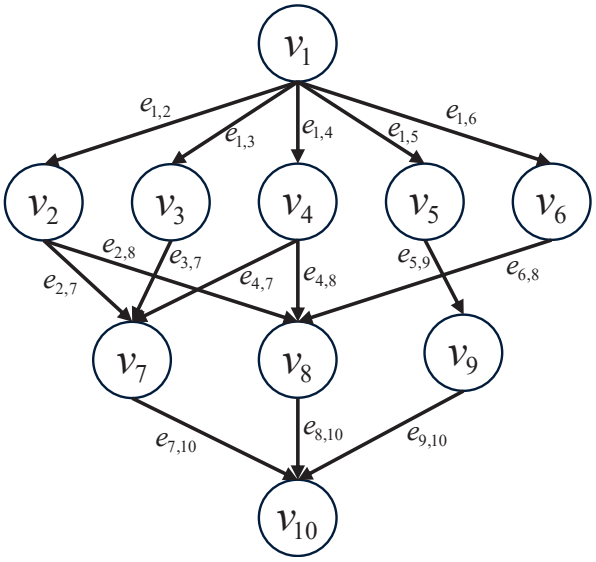


Figure 3. DAG example diagram (s = 10).

Let $P = \{p_1, p_2, \ldots, p_m\}$ represent the set of edge computing nodes, where $m = |P|$ denotes the number of edge computing nodes. We use $w_{i,j}$ to represent the calculation delay of subtask v_i on the edge computing node p_j. $w_{i,j}$ can be expressed as

$$w_{i,j} = \frac{L_i}{h_j} \tag{1}$$

where L_i is the calculation amount of the subtask v_i, represented by million instructions (MI), and h_j is the processing power of the current edge computing node p_j, represented by million instructions per second (MIPS).

Then, the energy consumption $\Theta_{i,j}$ of this execution can be expressed as

$$\Theta_{i,j} = w_{i,j} \times P_j^{work} \tag{2}$$

where P_j^{work} represents the power of the edge computing node p_j in operation.

Therefore, we define the matrix $W = [w_{i,j}]_{s \times m}$ as the computation latency of the subtask set on all edge computing nodes. Similarly, the matrix $J = [\Theta_{i,j}]_{s \times m}$ is defined as the energy consumption of the task set on all edge computing nodes.

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{s1} & w_{s2} & \cdots & w_{sm} \end{bmatrix} \tag{3}$$

$$J = \begin{bmatrix} \Theta_{11} & \Theta_{12} & \cdots & \Theta_{1m} \\ \Theta_{21} & \Theta_{22} & \cdots & \Theta_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \Theta_{s1} & \Theta_{s2} & \cdots & \Theta_{sm} \end{bmatrix} \tag{4}$$

Assuming that in a satellite network, the communication speed between any two edge computing nodes is the same, we can use the transmission cost between two nodes as the communication cost. Each edge in the DAG corresponds to a communication cost. Let $C = \{c_{i,j} \mid e_{i,j} \in E, c_{i,j} > 0\}$ represent the set of communication costs for the entire DAG, where $e_{i,j}$ represents the directed edge from subtask v_i to its successor subtask v_j, and $c_{i,j}$ is the weight of $e_{i,j}$. The communication cost can be expressed as

$$c_{i,j} = t_{i,j} \times P_{pass} \tag{5}$$

where $t_{i,j}$ represents the communication latency between edge computing nodes i and j, and P_{pass} represents the transmission power between satellites.

Adopting the concept of upward rank values proposed by the classical HEFT algorithm, sorting the subtask priorities $rank_k(v_i)$ for each DAG, $rank_k(v_i)$ can be expressed as

$$rank_k(v_i) = \alpha\, \overline{w}_i + \beta\, \overline{\Theta}_i + \max(c_{i,j} + rank_k(v_j)) \tag{6}$$

where $\overline{w}_i$ represents the average execution time of the subtask v_i on different edge computing nodes, and $\overline{\Theta}_i$ represents the average energy consumption of the subtask v_i on different edge computing nodes. $\alpha, \beta \in [0, 1]$, and $\alpha + \beta = 1$, where α and β represent the balance ratio parameters for the average completion time and average energy consumption, respectively. $c_{i,j}$ represents the communication cost from the subtask v_i to the successor subtask v_j. A higher weight for $rank_k(v_i)$ indicates a higher priority for scheduling, where k represents the service request's identifier. For the above variables, normalization is required when performing summation.

The above design is specifically intended for regular DAG subtasks. Public subtasks, due to their uniqueness, are uniformly identified and executed by the public task management module within the GEO satellite. When executing the batch task set $a = \{application_1, application_2, \ldots, application_n\}$, preprocessing is uniformly carried out by the public task management module of the SDN controller. If a subtask is executed more than two times, it will be marked, and the execution results of the subtask will be cached in the GEO satellite for public use by all tasks. Setting a time parameter T, if a public subtask has not been executed beyond the time T, the record for that subtask is removed from the public task management module. We assume that the public subtasks are represented by v_i^{common}, and the set of public subtasks is represented by $Common$.

3.4. Algorithm Design

Different DAG workflows submitted by various users have distinct task requirements. When multiple tasks are simultaneously offloaded to edge servers, considering the limited computing resources of edge servers, allocating computing resources reasonably based on task priorities becomes a crucial issue. Defining any $DAG_k (1 \leq k \leq n)$, the priority of $application_k$ can be expressed as

$$T_k = \frac{1}{1 + e^{t_k^{stat} + t_k - systemtime}} \tag{7}$$

where t_k represents the maximum tolerated execution time, t_k^{start} represents the moment when the task accepts scheduling, and *systemtime* represents the current system time.

To prevent potential excessive delays in scheduling low-priority DAG tasks, in addition to considering task priorities, this paper introduces the concept of fair scheduling during the DAG selection phase, that is, the tasks in the DAG task queue with the greatest relative lag degree are always scheduled first. In the fair DAG selection step, to address fairness issues in multi-DAG applications, the relative lag degree can be expressed as

$$Slowdown_M(k) = 1 - \frac{M_{own}(k_{v_i})}{M_{muti}(k_{v_i})} \tag{8}$$

where $M_{own}(k_{v_i})$ represents the completion time of a subtask v_i when DAG_k is scheduled independently, and $M_{muti}(k_{v_i})$ represents the completion time of subtasks v_i when DAG_k is scheduled in a multi-DAG scenario. Additionally, to prevent excessive waiting times for low-priority tasks, we introduce the waiting time $wait_k$ since the last scheduling of the current DAG_k. The final total scheduling priority can be expressed as

$$Y_k = Slowdown_M(k) + wait_k + T_k \tag{9}$$

The task scheduling algorithm for DAG satellite networks based on cache-composite priority is shown in Algorithm 1.

Algorithm 1 DAG Satellite Network Task Scheduling Algorithm Based on Cache-Composite Priority

Input: DAG set a to be scheduled, available resource set P
Output: Scheduling scheme S_{result} for all DAGs in set a

Apply Formula (6) on the resource set P to calculate the priority of all subtasks for each DAG in the DAG set a to be scheduled, and save the results to S_{own}
Mark each DAG as unscheduled and place it into the unscheduled set U
Initialize the priority T_k for all DAGs in the unscheduled set U according to Formula (7) and sort them in descending order
Preprocess all DAGs, find the public subtask set *Common*, and cache the results
while U $\neq \emptyset$ **do**
    Take out the first DAG task a_k from the set U
    According to the result stored in S_{own}, take out the subtask v_i with the highest current priority from a_k
    if $v_i \in Common$ **then**
        Request the SDN controller to obtain the task result from the cache
    else
        Record the completion time of the scheduling subtask v_i as d, and save the scheduling result to S_{result}
    end if
    if v_i is the last task of the current DAG **then**
        Delete task a_k from U
    else
        $M_{own}(k_{v_i})$ is the scheduling time of v_i in S_{own}, d is the value of $M_{muti}(k_{v_i})$. Update the priority Y_k of DAGs in the set U according to Formula (9)
        Re-sort all DAGs in set U in descending order according to the value of Y_k
    end if
end while

Next, we analyze the overhead of scheduling Algorithm 1 in the system. Firstly, the complexity of calculating the priorities of all subtasks in the set of DAGs to be scheduled is $O(ns \log(ns))$, where ns represents the number of subtasks. Secondly, the complexity of reordering all DAGs in the set U based on priority is $O(n^2 \log n)$, where n represents the

number of DAGs. Therefore, the complexity of the Cache-Composite priority algorithm is $O(\max(ns\log(ns), n^2\log n))$.

4. Simulation Experiment

This simulation experiment utilized NS2 simulation software to simulate the information transmission network. In the simulation software, the LEO satellites constellation in this paper adopted the Iridium constellation, a diagram of which is shown in Figure 4. The constellation structure consists of 66 satellites orbiting in six polar circular orbits, at an altitude of approximately 780 km, to ensure global coverage. Additionally, the constellation composed of three GEO satellites can cover the whole Iridium constellation. The SDN controller was deployed for the GEO satellite constellation. The SDN controllers of the three GEO satellites synchronized their data with each other and provided services for the LEO satellites covered by them. The SDN controller can monitor satellites within the Iridium constellation and schedule multi-DAG tasks received by the LEO satellites (DAG scheduling can be achieved through software programming and then deployed to the SDN controllers on the GEO satellites through the north interface), achieving global optimization. The Iridium communication network can provide reliable communication connections globally, enabling the SDN controller to effectively manage and schedule DAG tasks. This is particularly helpful for scenarios requiring task distribution and data transmission across different geographical regions. The processing speed of edge computing nodes, the workload of subtasks, the power of edge computing nodes, and the transmission power between satellites were set [30] as listed in Table 1.

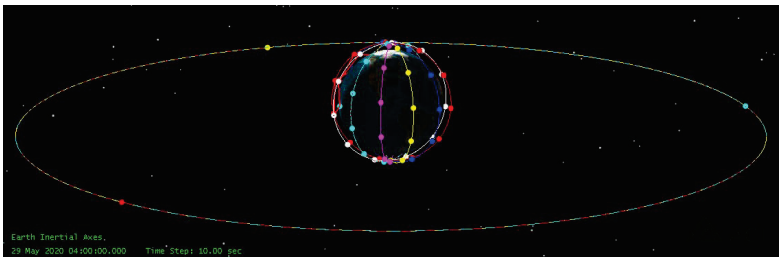


Figure 4. Iridium constellation diagram.

Table 1. System simulation parameters.

Parameter	Value
Edge computing node processing speed h_j	200∼300 MIPS
Subtask workload L_i	20∼100 MI
Edge computing node power P_j^{work}	300∼350 W
Inter-satellite transmission power P_{pass}	1∼4 W

4.1. Multi-DAG Task Completion Time and Energy Consumption Experiment

To validate the performance of the proposed multi-DAG task scheduling algorithm, in the simulation experiments, we randomly generated various edge computing nodes (m = 3, 5, 7, 15) and multiple random DAGs (n = 5, 10, 15) in the satellite network for testing and conducted 50 experiments on the Composite priority algorithm, Fairness algorithm, LB-HEFT algorithm, and Cache-Composite priority algorithm, respectively. The comparison diagram of average completion time and total energy consumption is shown in Figures 5 and 6.

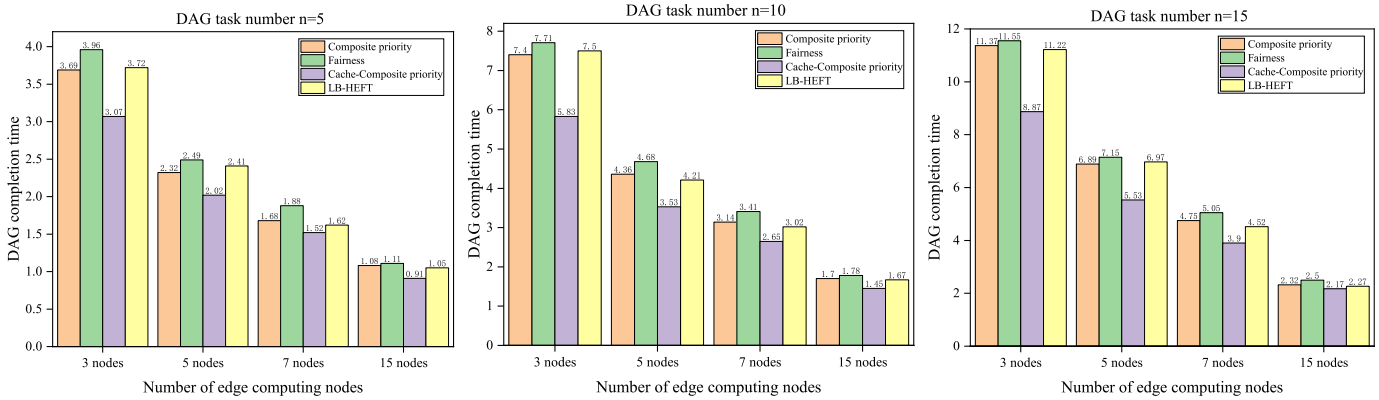


Figure 5. Average completion time comparison diagram.

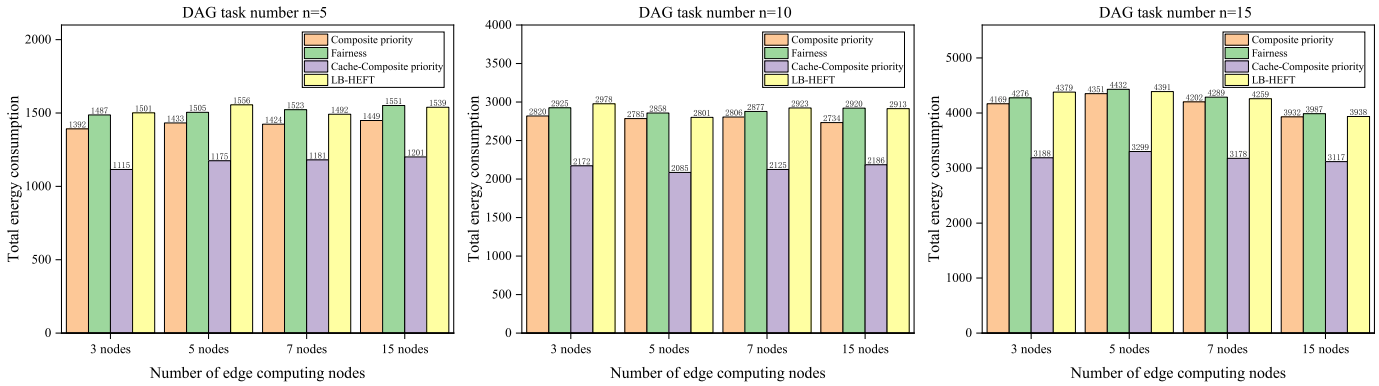


Figure 6. Total energy consumption comparison diagram.

The comparison diagram of DAG completion time is shown in Figure 5, where the completion time of the Composite priority algorithm and LB-HEFT algorithm is superior to that of the Fairness algorithm, showing an improvement of nearly 5% in performance. As the number of tasks increased, the gap gradually decreased. The reason for this phenomenon is that the Composite priority algorithm needs to consider the priority of each DAG task. When the number of tasks increased with a constant number of edge computing nodes, the weight of the DAG task priority increased, thus affecting the results. The introduction of the Cache-Composite priority algorithm significantly improved the performance, with the completion time performance increasing by nearly 20%. The total energy consumption comparison diagram is shown in Figure 6. Due to the Composite priority algorithm considering not only completion time but also energy consumption factors, its energy consumption was superior to that of the Fairness algorithm, resulting in a reduction in the energy consumption of nearly 7%. Similarly, the performance of the Composite priority algorithm with caching was enhanced by almost 25% compared to the Fairness algorithm. Because the LB-HEFT algorithm does not optimize for energy consumption, its energy consumption performance was comparable to that of the Fairness algorithm.

In summary, the performance of the Composite priority algorithm and LB-HEFT algorithm showed a slight improvement compared to that of the Fairness algorithm in terms of both DAG completion time and energy consumption. Building upon the Composite priority algorithm, the introduction of a satellite SDN controller public task management module employing the Cache-Composite priority algorithm, which identifies and caches public subtasks, resulted in a significant performance improvement compared to the Fairness algorithm.

4.2. Multi-DAG Task Scheduling Fairness Experiment

In this study, we needed to conduct experiments on the scheduling fairness of the four algorithms. According to previous research [21], the unfairness of the scheduling algorithm is defined as

$$Unfairness_M(S) = \sum_{\forall k \in a} |Slowdown_M(k) - AvgSlowdown_M| \tag{10}$$

where $AvgSlowdown_M$ represents the average $Slowdown_M$ value for all DAGs, and the smaller the value of $Unfairness_M(S)$, the higher the scheduling fairness of the algorithm. The unfairness of the four algorithms is shown in Figure 7.

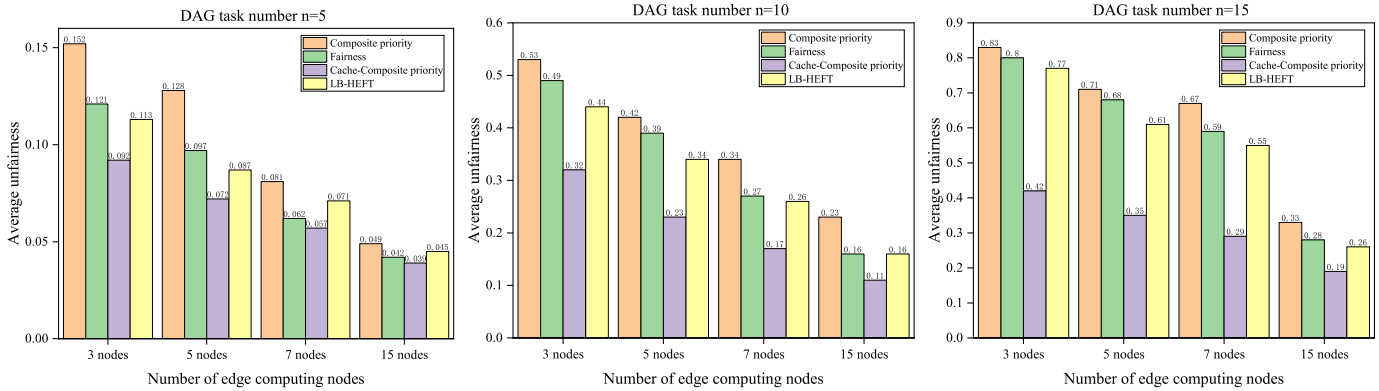


Figure 7. Unfairness degree comparison diagram.

The unfairness comparison diagram is shown in Figure 7. In the evaluation of unfairness performance, the average unfairness increased with the growth in DAG task numbers and decreased with the increase in node numbers. The performance of the Composite priority algorithm was the poorest. The reason for this is that the priority of the original DAG task increased during multi-DAG scheduling, which affected the performance of the algorithm compared with the Fairness algorithm. The LB-HEFT algorithm's performance was slightly better than that of the Fairness algorithm, while the performance of the Cache-Composite priority algorithm was the best. Due to the introduction of the concept of fair scheduling in the Cache-Composite priority algorithm, the difference in unfairness between the two algorithms became more pronounced compared to the Fairness algorithm. The unfairness of the Cache-Composite priority algorithm was reduced by 42.1%.

4.3. Multi-DAG Task Load Balancing Experiment

Tables 2–5, respectively, represent the number of tasks assigned by of the multiple edge computing nodes (m = 3, 5, 7, 15) during random multi-DAG generation. To assess the load balancing performance of the four algorithms, a performance analysis was conducted, and we defined the load balancing factor as

$$\sigma = \sqrt{\frac{\sum_{i=1}^{m} | \ avgload - load_i \ |}{m}} \tag{11}$$

where $avgload$ represents the average load of the batch of nodes; $load_i$ represents the load of node i; and the smaller the factor, the better the load performance. The load balancing factor for the Fairness algorithm, Composite priority algorithm, Cache-Composite priority algorithm, and LB-HEFT algorithm can be obtained from Tables 2–5, as shown in Figure 8.

Table 2. Number of node tasks (edge computing node m = 3).

	Fairness	Composite Priority	Cache-Composite Priority	LB-HEFT
Node 1	42	35	45	40
Node 2	44	47	39	39
Node 3	32	36	34	39

Table 3. Number of node tasks (edge computing node m = 5).

	Fairness	Composite Priority	Cache-Composite Priority	LB-HEFT
Node 1	30	27	27	24
Node 2	24	25	24	24
Node 3	20	24	21	25
Node 4	21	18	24	24
Node 5	23	24	22	23

Table 4. Number of node tasks (edge computing node m = 7).

	Fairness	Composite Priority	Cache-Composite Priority	LB-HEFT
Node 1	21	16	18	17
Node 2	17	18	17	16
Node 3	21	19	19	16
Node 4	15	17	17	17
Node 5	17	19	17	17
Node 6	12	15	16	18
Node 7	15	14	14	17

Table 5. Number of node tasks (edge computing node m = 15).

	Fairness	Composite Priority	Cache-Composite Priority	LB-HEFT
Node 1	8	8	8	8
Node 2	9	9	8	8
Node 3	8	8	8	8
Node 4	7	8	8	8
Node 5	7	9	9	8
Node 6	9	8	8	8
Node 7	7	8	8	8
Node 8	8	7	7	7
Node 9	9	7	7	7
Node 10	7	8	8	8
Node 11	9	7	8	8
Node 12	8	8	7	8
Node 13	7	8	8	8
Node 14	8	7	8	8
Node 15	7	8	8	8

From Figure 8, it can be observed that the load performance of the LB-HEFT algorithm was the best, followed by that of the Cache-Composite priority algorithm proposed in this paper. Both algorithms outperformed the Fairness algorithm and the Composite priority algorithm in terms of load balancing performance.

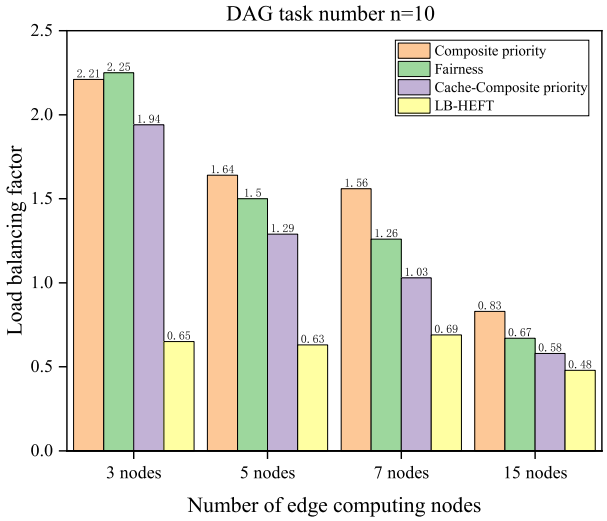


Figure 8. Load balancing factor comparison diagram.

4.4. Multi-DAG Task Completion Degree Experiment

To verify whether the algorithm can meet the DAG scheduling requirements, that is, whether it can meet the maximum tolerable latency of each task, the definition of task completion degree is introduced: that is, the ratio of the number of random DAG tasks meeting their respective maximum tolerable latency to the total number of random DAG tasks. The task completion degree comparison is shown in Tables 6–9.

Table 6. Task completion degree comparison (edge computing node m = 3).

	Fairness	Composite Priority	Cache-Composite Priority	LB-HEFT
Task count n = 5	3/5	4/5	5/5	4/5
Task count n = 10	4/10	6/10	8/10	5/10
Task count n = 15	5/15	8/15	14/15	7/15

Table 7. Task completion degree comparison (edge computing node m = 5).

	Fairness	Composite Priority	Cache-Composite Priority	LB-HEFT
Task count n = 5	5/5	5/5	5/5	5/5
Task count n = 10	6/10	8/10	10/10	8/10
Task count n = 15	9/15	12/15	14/15	11/15

Table 8. Task completion degree comparison (edge computing node m = 7).

	Fairness	Composite Priority	Cache-Composite Priority	LB-HEFT
Task count n = 5	5/5	5/5	5/5	5/5
Task count n = 10	7/10	9/10	10/10	10/10
Task count n = 15	10/15	14/15	15/15	14/15

Table 9. Task completion degree comparison (edge computing node m = 15).

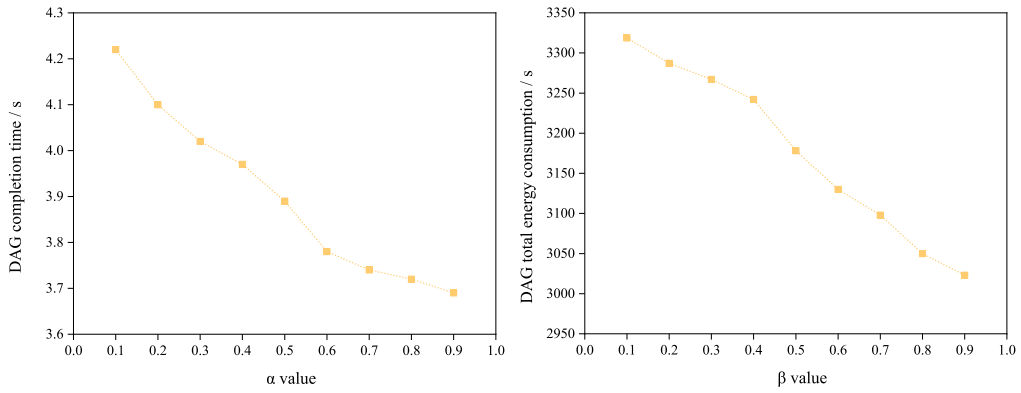
	Fairness	Composite Priority	Cache-Composite Priority	LB-HEFT
Task count n = 5	5/5	5/5	5/5	5/5
Task count n = 10	10/10	10/10	10/10	10/10
Task count n = 15	14/15	15/15	15/15	15/15

Tables 6–9 show the task completion degree comparisons of the Fairness algorithm, Composite priority algorithm, Cache-Composite priority algorithm, and LB-HEFT algorithm executing multiple random DAGs (n = 5, 10, 15) on edge computing nodes (m = 3, 5, 7, 15). In Table 6 (m = 3), the differences are pronounced. The task completion degree of the Fairness algorithm was lower than that of the Composite priority algorithm, LB-HEFT algorithm, and Cache-Composite priority algorithm. All four algorithms exhibited a declining trend in task completion degree, with the Fairness algorithm showing a significant fluctuation as task numbers increased. With the increase in edge computing nodes, the task completion degree of the Cache-Composite priority algorithm consistently exceeded 90%. The reason for this is that the Fairness algorithm did not consider the tolerance latency of the task, and the Cache-Composite priority algorithm added this factor. Consequently, during task scheduling, the priority of DAG tasks increased with time, which influenced the results.

4.5. Multi-DAG Influence Parameter Experiment

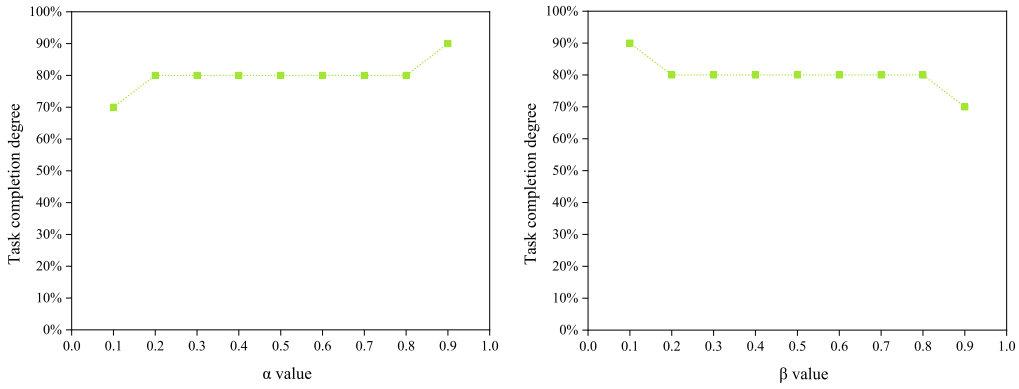
As shown in Figure 9, we conducted experiments focusing on the balance ratio parameter in Formula (6). These experiments were conducted by using the Cache-Composite Priority algorithm to execute multiple random DAGs (n = 15) on edge computing nodes (m = 7). Figure 9a demonstrates that as the value of α increased, the DAG completion time gradually decreased when scheduling subtasks. This is because in Formula (6), an increase in the value of α results in a higher weighting of delay factors, which ultimately leads to a reduction in the DAG completion time. Figure 9b shows that as the value of β increased, the DAG total energy consumption gradually decreased when subtasks were scheduled. This is because in Formula (6), an increase in the value of β results in a higher weighting of energy consumption factors, which ultimately leads to a reduction in the DAG total energy consumption.

The relationship between the balance ratio parameter and task completion degree is illustrated in Figure 10. In order to explain the experimental results more clearly, this experiment was conducted by using the Cache-Composite Priority algorithm to execute multiple random DAGs (n = 10) on edge computing nodes (m = 3). Figure 10a demonstrates that the task completion degree exhibited a gradual increase with the increase in the α value. The reason for this is that as the α value increased, the task completion time was reduced, thereby improving the task completion degree. Figure 10b shows that the task completion degree exhibited a gradual decrease with the increase in the β value. The reason for this is that an increase in the β value reduced the energy consumption, thereby affecting task execution efficiency and resulting in a reduced task completion degree.



(a) α value - DAG completion time relationship diagram (b) β value - DAG total energy consumption relationship diagram

Figure 9. Diagram of balance ratio parameter's relationship with DAG completion time and DAG total energy consumption.



(a) α value - task completion degree relationship diagram (b) β value - task completion degree relationship diagram

Figure 10. Balance ratio parameter and task completion degree relationship diagram.

5. Conclusions

In satellite edge computing, due to the limited computing capacity and energy consumption of MEC servers, individual MEC nodes may struggle to handle the computational load for certain tasks. To address this issue, this paper split large tasks into DAG tasks and proposed a multi-DAG scheduling algorithm with composite priority. This algorithm effectively resolves the challenge of deploying large tasks to remote clouds, thereby improving user response latency and reducing system energy consumption. At the same time, in the satellite edge computing system, the concept of public subtasks was introduced to reduce system overhead from repetitive tasks. A multi-DAG task scheduling algorithm based on cache-composite priority was proposed, and compared to the LB-HEFT algorithm, it demonstrated improvements of nearly 20% in completion time performance and around 25% in energy consumption performance. Additionally, the Cache-Composite priority algorithm reduced the unfairness by 42.1% and significantly enhanced the task completion degree. However, the performance of the Cache-Composite priority algorithm in load balancing was not ideal and requires further research and optimization.

Author Contributions: Conceptualization, Z.L. and L.Z.; methodology, Z.L. and L.Z.; software, L.Z. and L.W.; validation, L.Z. and X.D.; investigation, L.Z., L.W. and J.R.; writing—original draft preparation, Z.L.; writing—review and editing, Z.L., L.Z., L.W., X.D. and J.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The processed data required to reproduce these findings cannot be shared as they also form part of an ongoing study.

Conflicts of Interest: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Dang, S.; Amin, O.; Shihada, B. What should 6G be? *Nat. Electron.* **2020**, *3*, 20–29. [CrossRef]
2. Khan, L.U.; Saad, W.; Niyato, D. Digital-twin-enabled 6G: Vision, architectural trends, and future directions. *IEEE Commun. Mag.* **2022**, *60*, 74–80. [CrossRef]
3. Zhang, H.; Chen, A.; Li, Y.; Long, K. Key technologies of 6G mobile network. *J. Commun.* **2022**, *43*, 189–202.
4. Zhang, H.; Liu, R.; Kaushik, A. Satellite Edge Computing with Collaborative Computation Offloading: An Intelligent Deep Deterministic Policy Gradient Approach. *IEEE Internet Things J.* **2023**, *10*, 9092–9107. [CrossRef]
5. Tang, Q.; Xie, R.; Liu, X.; Zhang, Y. Satellite-ground collaborative network integrating MEC: Architecture, key technologies and challenges. *J. Commun.* **2020**, *41*, 162–181.
6. Wang, B.; Feng, H.; Huang, D. A joint computation offloading and resource allocation strategy for LEO satellite edge computing system. In Proceedings of the 2020 IEEE 20th International Conference on Communication Technology (ICCT), Nanning, China, 28–31 October 2020; pp. 649–655.

7. Wang, Y.; Yang, J.; Guo, X. A game-theoretic approach to computation offloading in satellite edge computing. *IEEE Access* **2019**, *8*, 12510–12520. [CrossRef]
8. Tang, Q.; Fei, Z.; Li, B. Computation offloading in LEO satellite networks with hybrid cloud and edge computing. *IEEE Internet Things J.* **2021**, *8*, 9164–9176. [CrossRef]
9. Hsiang, E.L.; Yang, Z.; Yang, Q. AR/VR light engines: Perspectives and challenges. *Adv. Opt. Photonics* **2022**, *14*, 783–861. [CrossRef]
10. Hakak, S.; Gadekallu, T.R.; Maddikunta, P.K.R. Autonomous Vehicles in 5G and beyond: A Survey. *Veh. Commun.* **2023**, *39*, 100551. [CrossRef]
11. Dong, S.; Wang, P.; Abbas, K. A survey on deep learning and its applications. *Comput. Sci. Rev.* **2021**, *40*, 100379. [CrossRef]
12. Shlezinger, N.; Whang, J.; Eldar, Y.C. Model-based deep learning. *Proc. IEEE* **2023**, *111*, 465–499. [CrossRef]
13. Topcuoglu, H.; Hariri, S.; Wu, M.Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* **2002**, *13*, 260–274. [CrossRef]
14. Arabnejad, H.; Barbosa, J.G. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Trans. Parallel Distrib. Syst.* **2013**, *25*, 682–694. [CrossRef]
15. Mahmoud, H.; Thabet, M.; Khafagy, M.H. An efficient load balancing technique for task scheduling in heterogeneous cloud environment. *Clust. Comput.* **2021**, *24*, 3405–3419. [CrossRef]
16. Senapati, D.; Sarkar, A.; Karfa, C. HMDS: A makespan minimizing DAG scheduler for heterogeneous distributed systems. *ACM Trans. Embed. Comput. Syst. (TECS)* **2021**, *20*, 1–26. [CrossRef]
17. Djigal, H.; Feng, J.; Lu, J. Task scheduling for heterogeneous computing using a predict cost matrix. In Proceedings of the Workshop Proceedings of the 48th International Conference on Parallel Processing, Kyoto, Japan, 5–8 August 2019; pp. 1–10.
18. Verma, P.; Maurya, A.K.; Yadav, R.S. A survey on energy-efficient workflow scheduling algorithms in cloud computing. *Softw. Pract Exper.* **2023**, 1–46. [CrossRef]
19. Zhao, Y.; Cao, S.; Yan, L. List scheduling algorithm based on pre-scheduling for heterogeneous computing. 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. 588–595.
20. Keshanchi, B.; Souri, A.; Navimipour, N.J. An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: Formal verification, simulation, and statistical testing. *J. Syst. Softw.* **2017**, *124*, 1–21. [CrossRef]
21. Zhao, H.; Sakellariou, R. Scheduling multiple DAGs onto heterogeneous systems. In Proceedings of the Proceedings 20th IEEE International Parallel and Distributed Processing Symposium, Rhodes, Greece, 25–29 April 2006; Volume 14.
22. Zhu, Y.; Hu, B. Smart-mDAG: An intelligent scheduling method for multi-DAG jobs. In Proceedings of the 2021 International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Republic of Korea, 20–22 October 2021; pp. 110–115.
23. Tian, G.; Xiao, C.; Xu, Z.; Xiao, X. Hybrid Scheduling Strategy for Multi-DAG Workflow in Heterogeneous Distributed Environment. *J. Softw.* **2016**, *23*, 15.
24. Zhang, Y.; Zhou, Z.; Shi, Z. Online scheduling optimization for DAG-based requests through reinforcement learning in collaboration edge networks. *IEEE Access* **2020**, *8*, 72985–72996. [CrossRef]
25. Cai, L.; Wei, X.; Xing, C.; Zou, X.; Zhang, G. Failure-resilient DAG task scheduling in edge computing. *Comput. Netw.* **2021**, *198*, 108361. [CrossRef]
26. Lin, Z.; Li, C.; Tian, L.; Zhang, B. A scheduling algorithm based on reinforcement learning for heterogeneous environments. *Appl. Soft Comput.* **2022**, *130*, 109707. [CrossRef]
27. Liu, Z.; Dong, X.; Wang, L.; Feng, J.; Pan, C. Satellite network task deployment method based on SDN and ICN. *Sensors* **2022**, *22*, 5439. [CrossRef]
28. Dilshodov, A.; Xayitboev, E. 5G tarmog'ida dasturiy ta'minot aniqlangan tarmoq (SDN) va OPENFLOW protokoli. *Eng. Probl. Innov.* **2023**, 75–76.
29. Chen, C.; Liao, Z.; Zhu, Y. Hierarchical domain-based multicontroller deployment strategy in SDN-enabled space-air-ground integrated network. *IEEE Trans. Aerosp. Electron. Syst.* **2022**, *58*, 4864–4879. [CrossRef]
30. Wang, Y.; Zhang, J.; Zhang, X. A computation offloading strategy in satellite terrestrial networks with double edge computing. In Proceedings of the 2018 IEEE international conference on communication systems (ICCS), Chengdu, China, 19–21 December 2018; pp. 450–455.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.