**Journal of Database Management**
Volume 34 • Issue 1

# Computation Offloading Method for Large-Scale Factory Access in Edge-Edge Collaboration Mode

Junfeng Man, Hunan First Normal University, China
Longqian Zhao, Nanjing University of Aeronautics and Astronautics, China
Bowen Xu, Virginia Polytechnic Institute and State University, USA*
Cheng Peng, Hunan University of Technology, China
Junjie Jiang, Hunan University of Technology, China
Yi Liu, Hunan University of Technology, China

## ABSTRACT

Large-scale manufacturing enterprises have complex business processes in their production workshops, and the edge-edge collaborative business model cannot adapt to the traditional computation offloading methods, which leads to the problem of load imbalance. For this problem, a computation offloading algorithm based on edge-edge collaboration mode for large-scale factory access is proposed, called the edge and edge collaborative computation offloading (EECCO) algorithm. First, the method partitions the directed acyclic graphs (DAGs) on edge server and terminal industrial equipment, then updates the tasks using a synchronization policy based on set theory to improve the accuracy effectively, and finally achieves load balancing through processor allocation. The experimental results show that the method shortens the processing time by improving computational resource utilization and employs a heterogeneous distributed system to achieve high computing performance when processing large-scale task sets.

## KEYWORDS

Computing Offloading, DAG Segmentation, DAG Synchronization, Edge-Edge Collaboration, Processor Allocation

## INTRODUCTION

Industrial cloud-edge collaboration is the trending networking paradigm involving seamless connectivity in heterogeneous industrial environments to aggregate distributed industrial devices into a shared resource pool and mobilizes these industrial devices for local manufacturing. In the

DOI: 10.4018/JDM.318451
*Corresponding Author

edge-side cloud collaborative computing in industrial environments, information flows are a dynamic exchange process that includes industrial device-to-network, industrial device-to-infrastructure, and industrial device-to-device to enable collaborative data sensing and information analysis (Cai et al., 2022; Mustafa et al., 2022; Xu & Zhu, 2022).

The purpose of computation offloading is to obtain the best system throughput and high-performance computing. Traditional computation offloading algorithms have limitations in the processing and analysis of high-dimensional and high-precision data. The minimum completion time algorithm, for example, significantly improves the total task completion rate, but the resources computed by the task scheduler and the maintenance cost of task information are high. The particle swarm optimization algorithm can achieve better cloud computing results, but this algorithm's main disadvantage is that it does not consider the user budget. Market-driven computation offloading algorithms feature low cost and short response times but lack scalability and long completion time. Tasks in industrial scenarios have the characteristics of multiplicity, multimodality, and small batches. For example, in the sheet metal industry, the factory focuses on the automatic cutting, punching, forming, shearing, and nesting processes for sheet metal parts. Usually, there are 20 to 200 jobs available for scheduling the sheet metal industry's production queue. Therefore, it is necessary to realize multi-site collaboration and multi-task parallel in the upstream and downstream of the work chain to realize the factory floor's intelligent integrated application. In the coming decade, more and more factories adopting the industry 4.0 model will develop worldwide. At the same time, along with the current situation of increasing task types and task sizes, there is an urgent need to give a computation offloading method for large-scale factory access in the edge-edge collaboration mode, to improve the efficiency of task execution effectively, scientifically reduce the procurement funds of enterprises, and avoid the waste of limited resources, which is the focus of this research and the problem to be solved in this paper (Materwala et al., 2022; Sheikh Sofla et al., 2022; Sun et al., 2022; Zhang et al., 2022).

This paper presents a reliable solution for collaborative operation scenarios between edge-side servers and cloud-side servers deployed in large factories. This paper proposes a task scheduling method for large-scale factory access under cloud collaborative computing architecture by learning the previous research results to solve the above problems. The main contributions of this paper are as follows:

1. A task division method in the edge-edge collaboration mode is proposed to solve computation offloading's difficulties due to the variability in practical scenarios.
2. A computation offloading method for large-scale factory access in edge-edge collaboration mode is proposed to solve high response delays caused by the increase of terminal industrial devices.
3. The simulation results verify the feasibility of the architecture in real industrial scenarios.

The rest of this article is organized as follows. The second part is related work. The third part is the problem description. This part mainly introduces the typical edge-edge collaborative computing architecture under the current industrial scenarios and leads to the edge-edge collaboration model proposed in this paper. The fourth part mainly proposes a computation offloading algorithm for large-scale factory access in edge-side collaborative mode. This algorithm can optimize the response latency of industrial terminal devices, achieve load balancing and improve the resource utilization of the server at the edge, so that it solves various problems arising from the computational offloading process and improve the robustness of the system through. The fifth part provides a detailed description of the EECCO algorithm and presents the algorithm performance by time complexity and space complexity. The sixth part discusses experiments and results, mainly verifying the feasibility and effectiveness of the EECCO algorithm and analyzing the simulation results. The last section concludes the whole paper.

## RELATED WORK

The computation offloading problems in the actual manufacturing workshop are systematic, dynamic, and random. The goal of such problems is to identify an approach that, at each decision stage, specifies how to allocate available resources among competing task requests to optimize the system's performance. The computation offloading requirements in the actual manufacturing workshop are closely related to the complex systems in the dynamic environment, making the traditional computation offloading method unable to adapt to the business model under the cloud manufacturing scenarios, leading to the problem of load imbalance (Li et al., 2022; Liu et al., 2022).

In recent years, artificial intelligence has been widely developed in manufacturing, transportation, finance, medicine, and other fields. However, with machine learning and deep learning technologies in key application fields such as cloud manufacturing and self-driving, researchers explore advanced edge-edge collaboration methods to ensure real-time, precision, and high interaction efficiency. Khan et al. (2020) designed an optimal computation offloading algorithm based on integer linear optimization that allows each mobile device to dynamically select execution modes: local execution, offloading execution, and deletion of tasks, thus achieving significant energy improvements. Yang et al. (2020) designed a lightweight linear programming algorithm based on an integrated architecture that can effectively reduce industrial equipment's energy consumption and cloud computing cost by transforming the offloading problem into an energy cost minimization problem to achieve full utilization of resources in edge computing. Chen et al. (2022) proposed a composite integer nonlinear programming algorithm, which aimed at the computational offloading problem in mobile edge computing systems, using convex optimization to derive the closed-form of the resource allocation solution, transforming it into an integer linear programming problem, Therefore, it has certain effectiveness and advantages in reducing delay and energy consumption. The above solution is often only suitable for solving integer linear programming problems, which require that all or part of the decision variables are assumed to be non-negative integers. However, in the practical industry, the solution is too idealistic, and the integer solution obtained by rounding is usually inaccurate. Because of the above algorithm's limitations, Liao et al. (2021) designed a distributed offloading strategy based on a binary coded genetic algorithm that splits the multi-user multi-server problem into two phases, server selection and computational offloading, for solving the problem to obtain an adaptive offloading decision. Hussein and Mousa (2020) proposed a meta-heuristic computing offloading algorithm based on the ant colony algorithm and the particle swarm algorithm. They considered the influence of load balancing on the offloading strategy, thus significantly reducing the response time of Internet of Things applications and improving the service quality. Topcuoglu et al. (2002) presented an algorithm named Critical-Path-on-a-Processor (CPOP) for workflow scheduling over heterogeneous processors with the bounded number. Xu et al. (2022) proposed a computing offloading strategy for the gene-ant colony fusion algorithm, which makes up for the shortcomings of a single heuristic algorithm applied to computing offloading, improves the efficiency of the algorithm, and realizes the efficient use of edge-side base station resources. Due to the large number of parameters involved in the ant colony algorithm, the initial value is easy to be selected improperly. The use of a single ant colony algorithm tends to fall into local optimum and has low processing efficiency when dealing with large-scale combinatorial problems, followed by not solving continuous problems well. Most of the relevant parameters, such as pheromones, are selected by personal experience without sufficient theoretical basis. Due to its limitations, Cha et al. (2021) designed a virtual edge formation algorithm by predicting the inter-vehicle link duration to efficiently utilize the sporadic free computing resources around the smart vehicles. Laroui et al. (2021) proposed a service offloading technique in virtual mobile edge computing, which utilizes the computation offloading method of deep reinforcement learning to process the IoT network accessed by large-scale industrial equipment to obtain effective offloading decisions. Zhu et al. (2018) proposed the local-based partial reasonable task schedule construction (LoPRTC) to deal with a more complex multi-heterogeneous MEC server scenario shared by multiple mobile devices to ensure time sensitivity. However, the above

scheme is usually only applicable to the mobile edge calculation scene. In the complex industrial scene, industrial devices are more often fixedly installed in factories, and it needs to be discussed on a case-by-case basis. Given the above algorithms' limitations, Luo et al. (2021) proposed a new architecture that automatically offloads user tasks in mobile edge computing scenarios, which utilizes drones to cache data generated IoT devices dynamically and enables flexibility in computation offloading using blockchain technology. Wang et al. (2021) proposed a joint UAV-based heuristic power and quality-of-experience algorithm that jointly optimizes the UAV offload delay, transmit power, and UAV layout, thus ensuring the quality of experience for different priorities endpoints. Guo et al. (2018) proposed a game-theoretic greedy approximation offloading algorithm (GT-GAOA) to ensure better service quality and quality of experience and other diverse requirements in multi-user ultra-dense edge server scenarios. Zhang et al. (2022) proposed a dual auction-based common task offloading scheme and a Stackelberg game-based mining task offloading scheme to cope with the high computational overhead of the blockchain mining process, thereby achieving efficient resource allocation. However, the above scheme only considers how mobile IoT devices' tasks in industrial scenarios are offloaded to the edge-side servers, and there is no reasonable classification of computation offloading strategies within the edge-side server clusters. For the above algorithms' limitations, Zhu et al. (2021) proposed a computational offloading algorithm based on deep reinforcement learning to accelerate the learning process by adjusting the number of candidate positions, which utilizes minimizing the offloading cost to achieve faster convergence and dynamic adjustment during the learning process, thus significantly reducing the running time. Zhang et al. (2021) proposed a distributed task-loading algorithm based on multiple intelligences and load balancing, which aims to effectively reduce the response latency of all users and improve the robustness and scalability of the system by introducing load balancing coefficients. Mekala et al. (2022) proposed a two-step service offloading method based on deep reinforcement learning to reduce the cost of edge servers through a DRL-influenced resource and ensemble analysis model, thereby achieving high resource utilization and low energy consumption. However, as another important machine learning method, deep reinforcement learning emphasizes how to take corresponding actions based on the environment to maximize the expected benefits. In this mode, the input sample data feeds back to the model, but does not directly give correct conclusions like other methods. The feedback of deep reinforcement learning only detects the model, and the model will gradually make adjustments after receiving stimuli similar to rewards or punishments. Due to inefficient data sampling, complicated reward function design, difficulty reproducing operational results, and unavoidable local optima.

The EECCO algorithm used in this paper can effectively divide the DAG by calculating the task priority, calculation offload tolerance and DAG critical path in the edge-edge cooperative mode. This step can improve the processing of large-scale combination problems in complex industrial scenarios. Efficiency makes up for the lack of theoretical basis for the selection of pheromone by Liao, Hussein and others, and cannot solve the continuous problem well. It also solves the problem that Wang et al. did not reasonably classify the computing offloading strategy in the edge server cluster. Second, in the EECCO algorithm, this paper synchronizes multiple DAGs in the edge-edge cooperative mode. This step uses CGSC and EGSC to construct, schedule and maintain the DAG, thus realizing the task flow with the change of space. Dynamic execution makes up for the lack of flexibility in the computing offloading process by Cha, Laroui and others, which greatly improves the computing offloading efficiency in any complex industrial scenario. Reasonable allocation, the proposed IHEFT algorithm process is mainly composed of three stages: weight allocation stage, task priority allocation stage, and processor selection. Through the EECCO algorithm, the efficiency of task execution is effectively improved, and the capital cost of the enterprise is greatly reduced.

## PROMBLEM DESCRIPTION

With the rapid development of industrial intelligence technology and the explosive growth of IoT terminal data, the analysis and processing of massive data increase industrial cloud platforms'

operational burden. The emergence of edge-cloud collaborative computing makes full use of all computing resources on the entire link, bringing into play the advantages of different devices and providing strong technical support for the popularization and development of intelligent manufacturing.

## DAG Task Description

According to the industrial IoT environment requirements, the edge-cloud collaboration mainly consists of the cloud component composed of the remote server cluster, the edge component composed of the edge server cluster, and the terminal component composed of the terminal cluster. Figure 1 depicts the working schematic under the edge-cloud collaboration computing architecture. In terms of specific operations, the cloud component is responsible for processing non-real-time, long-period data and completes the full lifecycle management of edge-side and end-side applications. The edge-side components significantly reduce the data transmission delay and meet the real-time data requirements of low-latency services. The terminal device is mainly responsible for data collection and real-time control of industrial equipment. The "End-Edge-Cloud" collaborative computing provides greater possibilities for precise decision-making and dynamic optimization of the industrial Internet.

It is assumed that there are n factories in which $k_1, k_2, \ldots, k_n$ DAG tasks are run on the deployed terminal devices and $q_1, q_2, \ldots, q_n$ DAG tasks are run on the edge server, respectively, and they cooperate with DAG tasks running on the cloud server to form the end-to-side cloud collaborative computing architecture. When the *i* th factory collaborates with the cloud server, the number of tasks in the cloud server cluster is $\alpha_1, \alpha_2, \ldots, \alpha_p$ ; The number of tasks in the edge server cluster is $\beta_1^i, \beta_2^i, \ldots, \beta_q^i$ And the number of tasks in the terminal device cluster is $\gamma_1^i, \gamma_2^i, \ldots, \gamma_k^i$, so the number of all tasks in the cloud environment is $\sum_{i=1}^{p} \alpha_i = p_i$ .The number of all tasks in the edge environment was $\sum_{j=1}^{q} \beta_j^i = q_i$ ;The number of all tasks in the terminal environment is $\sum_{j=1}^{k} \gamma_j^i = k_i$. Using the edge-cloud collaborative computing model, a task cannot be interrupted when it is not scheduled, and the task cannot be split into several smaller tasks. Figure 2 depicts a multi-DAG task graph in a cloud-side collaborative computing mode. There are p DAG task graphs in cloud components, and $q_1, q_2, \ldots, q_n$ task graphs in edge-end components, which run in n edge servers deployed in factories.

For simplicity, Figure 3 depicts three DAG task diagrams under the cloud-side collaborative computing architecture. DAG1 under the cloud component includes three tasks, and DAG2 under the edge component includes five tasks. DAG3 includes 2 tasks, and 3 DAG task graphs include a total of 10 tasks. The circle represents the task node, the background color of the node in the cloud component,

**Figure 1.**
**Schematic diagram of the working process under the end-side cloud collaborative computing architecture**

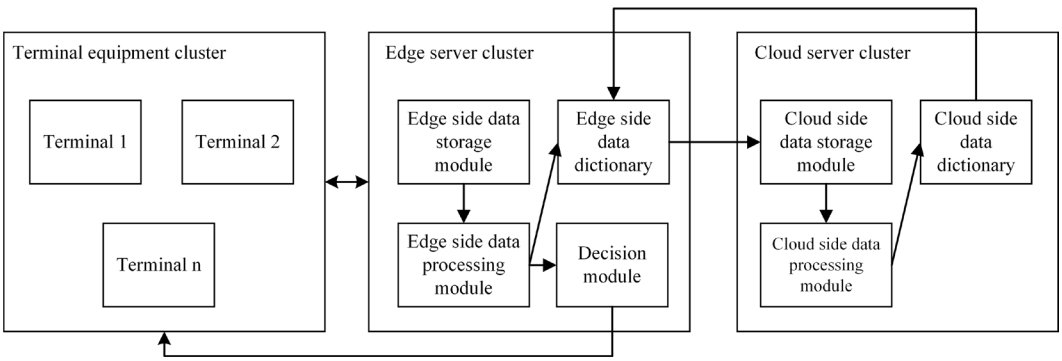
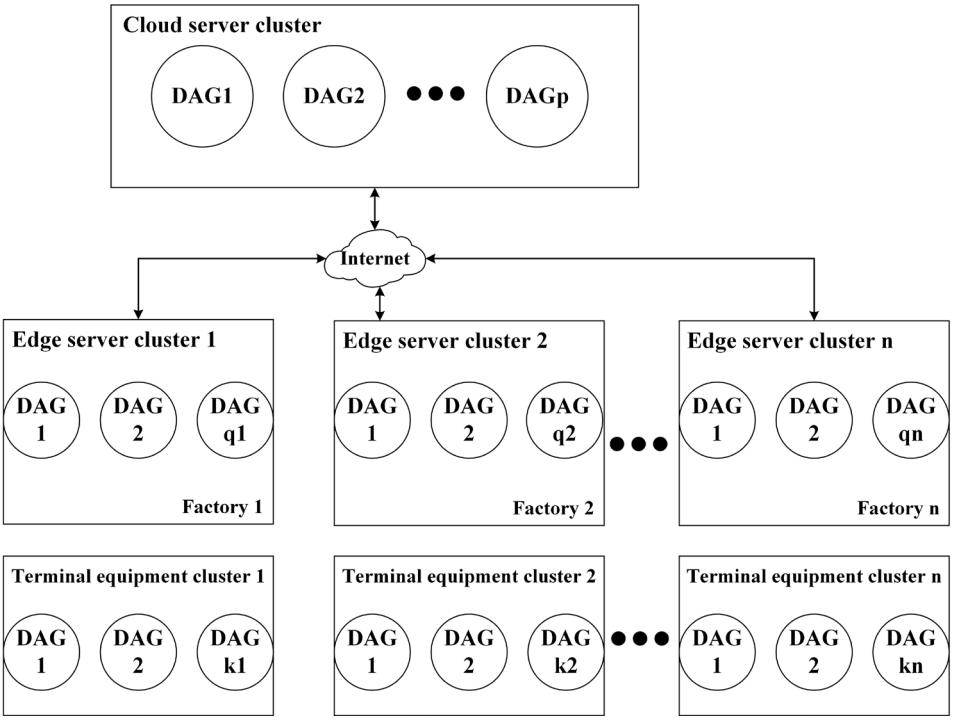


**Figure 2.**
**End-side cloud collaborative computing architecture**

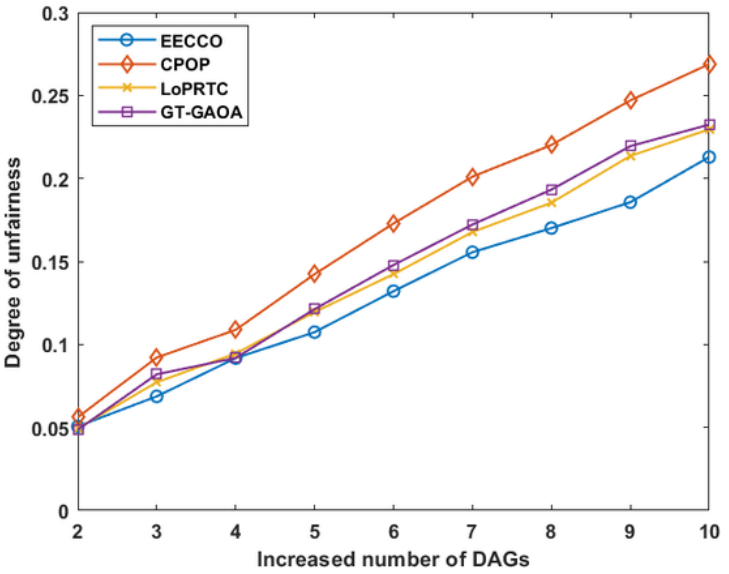



the edge part, and the terminal part are filled with green, white, and yellow respectively, the letter in the circle represents the number of the task node, and the task node number in the cloud part and the edge part is respectively used Uppercase and lowercase letters, node numbers are not case-sensitive, for example, the letter a in the circle represents task node $t_a, t_a = t_A$. The solid edge with an arrow indicates the event edge, and the number on edge indicates the event calculation overhead between the two tasks; The number on the side represents the communication delay between the two tasks. Each DAG includes 1 entry task node and 1 exit task node. Considering the communication delay of different tasks, for example, the communication delay of cloud computing is about 100*ms*. The communication delay of small data centers is about 10~40*ms*, the communication delay of routers is about 5*ms*, and the communication delay between terminal devices is about 5*ms*. It is about 1~2*ms* (Martinelli, 2018). We use the time instruction to obtain the event calculation overhead between the two tasks.

## Resources and Environment

Compute offloading can efficiently compute resources and speed up computation for resource-constrained end devices or single edge-side servers running compute-intensive applications. The offloading algorithm is to find a set of offloading schemes that allow the application's processing speed to be efficiently increased based on the task's deadlines and resource requirements. Figure 4 shows the edge-side collaboration model's organization diagram, which consists of a cloud component, an edge-side component, and an endpoint component. The cloud component and the edge component mainly consist of multiple performance heterogeneous physical machines, and each physical machine is virtualized with multiple virtual machines. Assuming that there are $P_1, P_2, ..., P_m$ physical machine nodes, each physical machine is composed of $p_1, p_2, ..., p_k$ virtual machines with heterogeneous performance and the physical machines are interconnected through

**Figure 3.**
**Three DAG task diagrams under the end-side cloud collaborative computing architecture**




a network. On each physical machine, task execution and communication can be executed at the same time, the communication overhead between tasks allocated to virtual machines is 0, and task execution is non-preemptive.

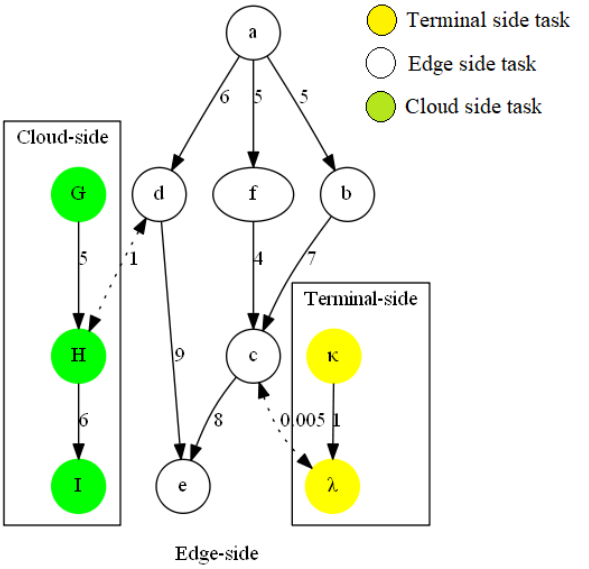
## Computing Offloading Target

DAG computation offloading is to offload the tasks in a single edge server or offload the terminal device calculations to the processor in the edge server for execution in the edge-side collaboration mode. The goal of computation offloading includes the following aspects:

1. The execution time of the entire task set on the processor makespan as small as possible;
2. Design a flat uninstall strategy for the edge server, instead of unified management through the cloud server;
3. The value $AWT = \frac{\sum_{j=1}^{n}\left(ST_j - AT_j\right)}{n}$ of all tasks in the entire task set from the arrival time $AT_i$ to the start execution time $ST_i$ should be as small as possible;
4. *Slack* is a measure of a computation offloading algorithm's robustness, reflecting the degree of uncertainty in the processing time of a task generated by a computation offloading algorithm (Yuan et al., 2022). The definition of *Slack* is shown in Equation 1. Where *M* represents the span of the DAG makespan, *n* represents the number of tasks, $b_{level}$ represents the length of the longest path from task $t_i$ to the exit, and $t_{level}$ represents the length of the longest path from the entry node to the task $t_i$.

$$Slack = \frac{\left[\sum_{t_i \in T} M - b_{level}\left(t_i\right) - t_{level}\left(t_i\right)\right]}{n} \quad (1)$$

**Figure 4.**
**The organizational chart of the edge-edge collaboration model**




5. *Unfairness (S)* is an important indicator used to measure the unfairness of the multi-DAG scheduling algorithm *S* (Mahdi et al., 2022). *Unfairness(S)* is defined as shown in Formula 2, where *A* is the set of a given multi-DAG, *AvgSlowdown* is the average of all *Slowdown*, and $Slowdown$ reflects the degree of hysteresis of the DAG, defined as $Slowdown\left(a\right) = M_{multi}\left(a\right) / M_{own}\left(a\right)$.

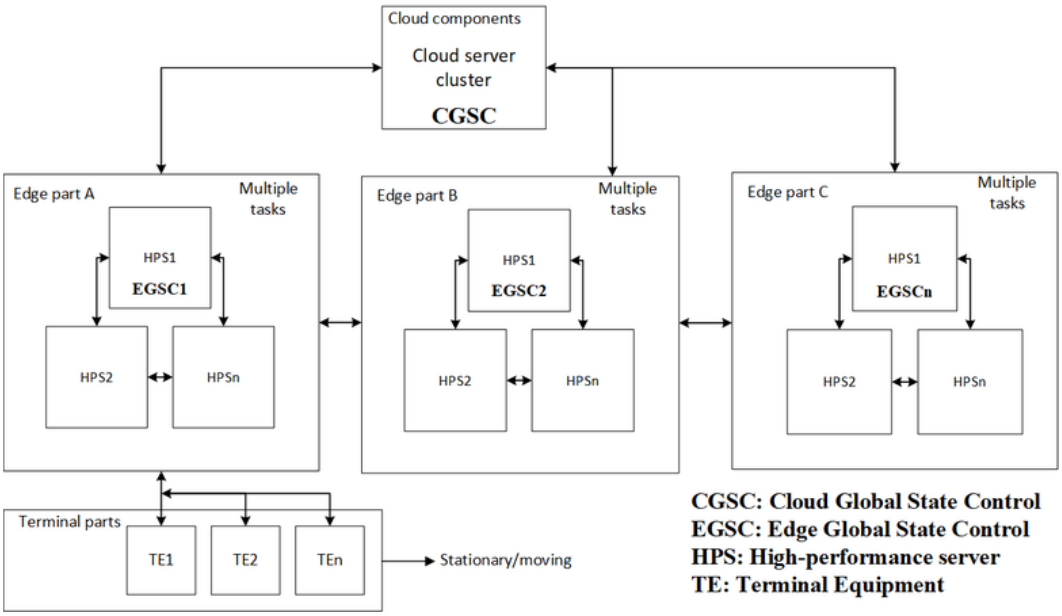
$$Unfairness\left(S\right) = \sum_{\forall a \in A} \left| Slowdown\left(a\right) - Avgslowdown \right| \tag{2}$$

## EDGW-EDGE COLLABORATIVE COMPUTATION OFFLOADING MODEL

In practical industrial application scenarios, tasks mainly consist of the static class known tasks and dynamic class unknown tasks. The priority of the static class of known tasks is determined at design time. Its priority varies with the dynamic class of unknown tasks. The tasks in this class satisfy the resource and latency requirements, i.e., the tasks can be successfully offloaded. The priority of the dynamic class of unknown tasks is determined during the runtime and changes continuously. There is some risk in the computational offloading strategy for this class of tasks. Both known tasks with static classes and unknown tasks with dynamic classes need to design corresponding computing offloading strategies to ensure efficient execution of tasks and full utilization of resources. Figure 5 shows the computing offload architecture in the edge-edge collaboration mode:

As can be seen from Figure 5, the edge-edge collaboration mode includes three central bodies: cloud component, edge component, and terminal component. The cloud component is mainly responsible for updating and synchronizing information states, such as computing offloading and task execution. The edge end components mainly include internal computing offloading and inter-computing offloading. The efficient utilization of resources can be achieved by coordinating

**Figure 5.**
**Computing offload architecture in the edge-side collaboration mode**




offloading strategies. Terminal components are classified according to industrial equipment's position state, including static calculation offloading of fixed position and dynamic calculation offloading of variable position. The three central bodies assume different functions based on different geographical spaces and roles of different devices and jointly promote the development of artificial intelligence.

## DAG Segmentation in Edge-Edge Collaboration Mode

Currently, heterogeneous computing systems play an increasingly important role in dealing with complex industrial production problems. Heterogeneous computer clusters support the execution of parallel applications to achieve the goal of completing tasks quickly. The Improved Critical Path on a Processor (ICPOP) algorithm proposed in this paper comprises three stages: computing task priority, computing offload tolerance, and computing DAG critical path.

## Task Priority

The computation offloading strategy in the edge-edge collaboration mode includes task offloading in edge components and task offloading in terminal components. Assume that an edge component $\delta$ is composed of $m$ edge servers, and each edge server virtualizes $k$ virtual machines. Each virtual machine runs $\tau\ DAG^{Edge}$, and each $DAG^{Edge}$ is composed of φ edge tasks. The end task will interact with $\tau'$ and $DAG^{Terminal}$ in the terminal part, and the adjacency matrix M corresponding to the DAG in the edge end part and the terminal part is shown in Formula 3:

$$M^{\frac{Edge}{Terminal}}\left[i,j\right]=\begin{cases}0 & if\ \ i=j\\ weight\left(i,j\right) & if\ \ i\neq j\ \ and\ \left(i,j\right)\in E\\ \infty & if\ \ i\neq j\ \ and\ \left(i,j\right)\notin E\end{cases} \tag{3}$$

From the adjacency matrix *M*, the sum of the weight values of the incoming edges of each task node $v_i$ can be calculated and inserted into the set *S*, as shown in Formula 4:

$$S^{\overline{Edge}}_{Terminal} = S^{\overline{Edge}}_{Terminal} \cup \left\{ \sum_i M^{\overline{Edge}}_{Terminal} \left( v_i \right) \right\} \tag{4}$$

In the industrial application scenario, due to the particularity of the industrial equipment itself, taking the aircraft as an example, there are many gears, bearings, blades, and other key mechanical components in the aircraft's powertrain. The parameter rights are obtained by analyzing the collected data through machine learning. Value *W*, so that each type of task's importance can be determined in advance (Wang & Leelapatra, 2022). Assume that each virtual machine may have operational failures and thus cannot complete the task. Assuming that these failures occur independently with probability *p*, Formula 5 indicates that the benefits of task processing are maximized:

$$\begin{aligned} &maximizes\ \pi_i^{\overline{Terminal}^{Edge}} = \sum_{i=1}^{\varphi} t_i \left(1 - p^{h_i}\right) \\ &subject\ \ to \sum_{i=1}^{h} h_i \le \varphi\ \ and\ \ h_i \ge 0 \end{aligned} \tag{5}$$

Combining Formula 3 and Formula 4, the priority $P_i$ of each task node $v_i$ of the DAG in the edge end part and the terminal part is given, as shown in Formula 6:

$$P_i^{\overline{Terminal}^{Edge}} = \frac{\sum w}{\left\| W \right\|} \times S_i \times \pi_i \tag{6}$$

*w* represents the weight of each parameter. It can be seen from Formula 6 that when the value of $P_i^{Edge/Terminal}$ is larger, the task node needs to be allocated more computing resources in time, thereby increasing the speed of task processing.

## Task Offload Tolerance

Before performing task uninstallation, it is necessary to evaluate the currently available resources fully, the uninstall task's size, and the energy consumed by executing local applications. Assuming that the communication delay between edge components is $\theta_1$; the communication delay between edge servers in the edge component is $\theta_2$; the communication delay between virtual machines in the edge server is $\theta_3$; the communication delays of the edge component and the terminal component is $\theta_4$. For the static known task set $t_i^{Edge/Terminal}$, the task set running time in a virtual machine is $time_i^{Edge}$, and the static known task set $t_i^{Edge}$ in the edge, component is separately stored in the same edge server by means of task offloading. Run in other virtual machines, the time is $time_i^{\text{Edge}'}$, $i = \left\{1, \ldots, \varphi\right\}$, and run in the virtual machines in other edge servers, and the time is $time_i^{\text{Edge}''}$, $i = \left\{1, \ldots, k\right\}$; The task set runs in a terminal for $time_i^{Terminal}$, and the static known task set $t_i^{Terminal}$ in the terminal component is run on the virtual machine in the edge server by task offloading, and its time is

$time_i^{\text{Terminal}'},\ i=\{1,\ldots,\varphi'\times k\times m\}$. Then the internal task offload tolerance $o$ of the known static tasks in the edge end parts and the terminal parts are shown in Formula 7 and 8:

$$o^{Edge}=\begin{cases}same\ \ server & ,\max\left\{time_i^{\text{Edge}'}\right\}-time_j^{Edge}<\theta_3,\ i=\{1,\ldots,k\}\\ other\ \ server & ,\ \max\left\{time_i^{\text{Edge}''}\right\}-time_j^{Edge}<\theta_2+\theta_3,\ i=\{1,\ldots,k\}\\ 0 & ,\ others\end{cases} \tag{7}$$

$$o^{Terminal}=\begin{cases}Edge\ \ server & ,\max\left\{time_i^{\text{Terminal}'}\right\}-\ time_j^{Terminal}<\theta_4, i=\{1,\ldots,k\}\\ 0 & ,others\end{cases} \tag{8}$$

The *same server* refers to the edge task set $t_i^{Edge}$ can only uninstall tasks on virtual machines in the same edge server, and the *other server* refers to the edge task set $t_i^{Edge}$ that can be other parts of the same edge. Uninstall tasks in the edge server. *Edge server* refers to the terminal task set $t_i^{Terminal}$ that can uninstall tasks to the virtual machine in the edge component for execution, and 0 means that the task set is recommended to be executed locally. For the dynamic unknown task $t_i'$, where the size of the input data of the task set is *s*, the average unit processing time of the data is *c*, and num is the number of tasks in the task set, then the internal task offloading tolerance of the dynamic unknown task is $o'$ As shown in Formula 9 and 10:

$$o^{Edge'}=\begin{cases}same\ \ server & ,\dfrac{s\times c}{num}-time_j^{Edge}<\theta_3,\ i=\{1,\ldots,k\}\\ other\ \ server & ,\ \dfrac{s\times c}{num}-time_j^{Edge}<\theta_2+\theta_3,\ i=\{1,\ldots,k\}\\ 0 & ,\ others\end{cases} \tag{9}$$

$$o^{Terminal'}=\begin{cases}Edge\ \ server & ,\dfrac{s\times c}{num}-\ time_j^{Terminal}<\theta_4,\ i=\{1,\ldots,k\}\\ 0 & ,others\end{cases} \tag{10}$$

In smart manufacturing production scenarios, industrial equipment is required to have the ability to self-organize and collaborate to meet flexible production. However, it also puts forward higher requirements for the flexible mobility and differentiated business processing capabilities of industrial equipment. Suppose that the residence time of a mobile device in a factory near the edge part is $\vartheta$, and its pre-stay time is $\vartheta_0$, and *m* is the amount of data calculation. Then the inter-task offloading tolerance $o''$ of the mobile device is shown in Formula 11:

$$o''=\begin{cases}1 & ,\dfrac{\left(M1-m\right)}{M1\times T1}-\dfrac{\vartheta}{\vartheta_0}\times\dfrac{\left(M2-m\right)}{M2\times T2}>0\\ & ,\dfrac{\left(M1-m\right)}{M1\times T1}-\dfrac{\left(M2-m\right)}{M2\times T2}>0\\ 0 & ,others\end{cases} \tag{11}$$

Where *M1* is the size of the remaining resources of the edge-end component, *M2* is the size of the remaining resources of the components adjacent to the edge-end component, *T1* is the communication delay between the mobile device and the edge-end component, and *T2* is the mobile device and the edge-end component. The communication delay of the adjacent components of the component, *T1* and T2, will change with the mobile device's location.

### DAG Critical Path

The earliest possible start time of a task j is represented by $t_{ES}\left(j\right)$, and any task can only start after all its predecessor tasks are completed. The earliest completion time of the task $t_i$ is represented by $t_{EF}\left(j\right)$. It represents the completion time that the task can reach according to the earliest start time, and its calculation Formula is:

$$\begin{cases} t_{ES}\left(j\right)=0 \\ t_{ES}\left(j\right)=max_k\left\{t_{ES}\left(k\right)+t\left(k\right)\right\} \\ t_{EF}\left(j\right)=t_{ES}\left(j\right)+t\left(j\right) \end{cases} \tag{12}$$

The latest start time of a task *j* is represented by $t_{LS}\left(j\right)$, which represents the latest time that task *j* must start without affecting the completion of the entire task on schedule. The latest completion time of task *j* is represented by $t_{LF}\left(j\right)$, which represents the completion time that the task can start at the latest time and its calculation formula is:

$$\begin{cases} t_{LS}\left(j\right)=max_k\left\{t_{LS}\left(k\right)-t\left(k\right)\right\} \\ t_{LF}\left(j\right)=t_{LS}\left(j\right)+t\left(j\right) \end{cases} \tag{13}$$

Formula 11 is a recursive process from the starting point to the endpoint; Formula 12 is a systematic process from the endpoint to the starting point. This paper uses Formula 11 and Formula 12 to implement the critical path algorithm to find the critical path $CP=\left\{cp_1,cp_2,\ldots,cp_m\right\},m\in N$ of the DAG merged graph. In this paper, the DAG separation graph is defined as the Critical Tasks Set (CTS) and the Non-Critical Tasks Set (NCTS), where the task set types are divided into edge-side collaborative tasks Edge-Side Tasks (EST), and the Terminal-Side Tasks (TST) is composed, and its definition is shown in Formula 14:

$$\begin{cases} DAG=CTS+NCTS \\ CTS=CTS_{EST}+CTS_{TST} \\ NCTS=NCTS_{EST}+NCTS_{TST} \\ \left\{CTS,NCTS\right\}=nEST+mTST,n\in N,m\in N \end{cases} \tag{14}$$

The DAG task's critical path is the longest path from entering the task to the exit node. Each task on this path provides the lowest cost for all critical paths. According to Formula 11, the edge end part and the DAG in the end part shown in Figure 3 are segmented in this paper. Figure 6 shows the segmentation diagram of the DAG in the edge-side collaboration mode. The path formed by the red arrow is the critical path of the DAG. The degree of each subgraph is {2,0,0,1}.

## Multi-DAG Synchronization in Edge-To-Side Collaboration Mode

The edge-side collaboration mode adopted in this paper is responsible for the construction, scheduling, and maintenance of DAG through Cloud Global State Controller (CGSC) and Edge Global State Controller (EGSC). CGSC is responsible for cloud components and edge components; EGSC is responsible for tasks in edge components and terminal components. Figure 7 shows the multi-DAG synchronization method in the edge-side collaboration mode. Aiming at 5G+ industrial intelligence integration to broaden the application scenarios, the data-driven optimized closed loop is used as the key to real-time decision-making through EGSC; for the actual scenarios of high-speed mobile industrial production lines, through CGSC regards decision-driven optimization closed loop as the key to real-time data processing (Darwish, 2022).

Due to the high computational complexity of the algorithm, multiple task links, and extended business processes are some of the most complex problems in the industrial field. The task running status includes successful execution, execution blockage, and execution failure.

Since there is data interaction between the edge part and the terminal part that the EGSC is responsible for, its task execution state $Status^{Edge}$ is shown in Formula 15, where the set $A_t^{Edge}$ represents the *DAG* in the edge part at time *t*, which can be represented by $A_t^{Edge} = \{a, b, c, \ldots, \alpha, \beta, \ldots\}$. $\{a, b, c, \ldots\}$ represents the task in the edge end part, $\{\alpha, \beta, \ldots\}$ represents the task that the terminal part is unloaded to the edge end part. $A_{t+1}^{Edge}$ represents $DAG'$ in the edge end component at time *t+1*.

**Figure 6.**
**DAG segmentation diagram in edge-side collaboration mode**

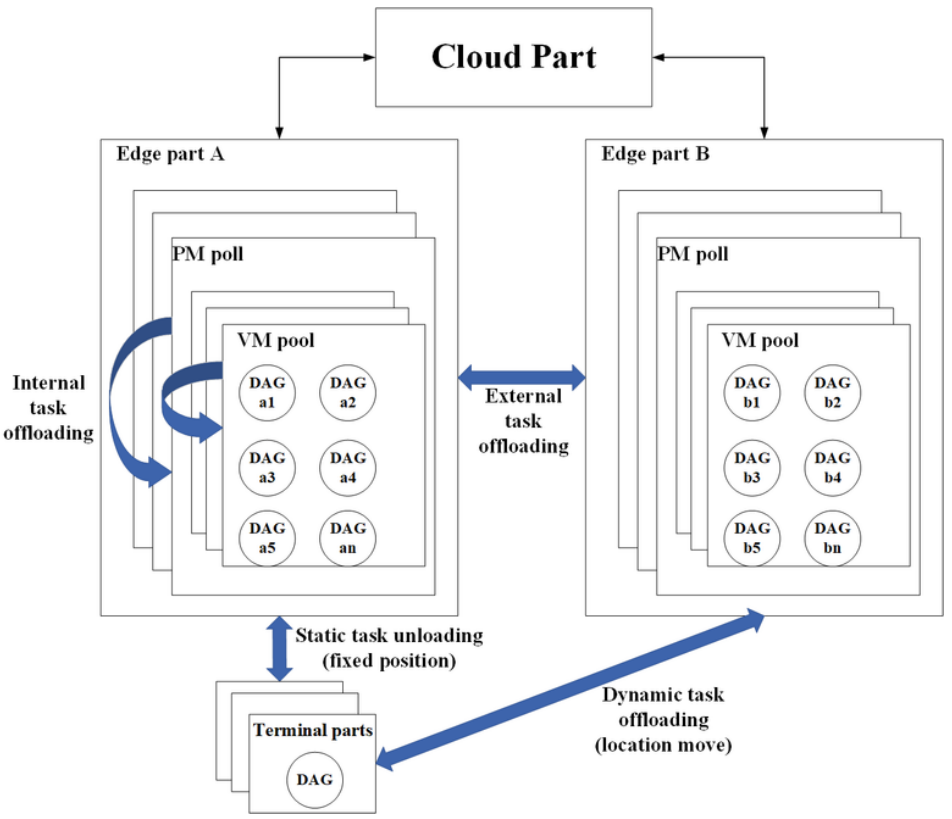
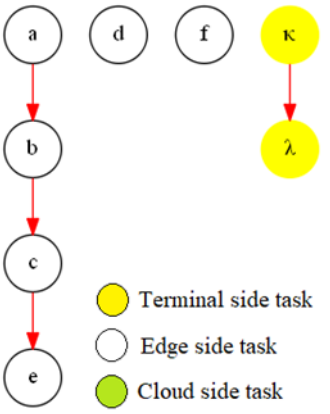


**Figure 7.**
**Multi-DAG synchronization method in edge-side collaboration mode**




$$Status^{Edge} = \begin{cases} execution\ \ succeed & ,A_t^{Edge} = A_t^{Edge} - A_{t+1}^{Edge} \\ execution\ \ blocking & ,A_t^{Edge} \supset A_{t+1}^{Edge} \\ execution\ \ failed & ,\varnothing = A_t^{Edge} - A_{t+1}^{Edge} \end{cases} \tag{15}$$

According to Formula 14, the task synchronization strategy © of the EGSC in the lower edge part at different times can be obtained, as shown in Formula 16:

$$\Omega = \begin{cases} B^{Edge} = \varnothing & ,execution\ \ succeed \\ B^{Edge} = A_t^{Edge} \oplus A_{t+1}^{Edge} & ,execution\ \ blocking \\ B^{Edge} = A_t^{Edge} & ,execution\ \ failed \end{cases} \tag{16}$$

Where $B^{Edge}$ represents the *DAG* after EGSC synchronization in the edge end component at time *t+1*. Since there is data interaction between the edge components that CGSC is responsible for, its task synchronization strategy is shown in Formula 17, where the set $\left| \begin{matrix} Cloud \\ i \end{matrix} \right.$ is expressed as the DAG set in the *i*-th edge component, which can be used as $\left| \begin{matrix} Cloud \\ i \end{matrix} \right. = \left\{ A_t^{Edge}, \ldots, A_{t+n}^{Edge}, \ldots, B_t^{Edge} \right\}$. $\left\{ A_t^{Edge}, \ldots, A_{t+n}^{Edge} \right\}$ represents the task execution sequence in the *i*-th edge end component, $\left\{ B_t^{Edge} \right\}$ represents the task of offloading other edge end components to the edge end component set.

$$C^{Cloud} = \bigcup_i^{\infty} \max_t \left\{ x \right\}, x \in \left| \begin{matrix} Cloud \\ i \end{matrix} \right. \tag{17}$$

Where $C^{Cloud}$ represents the union of the latest task completed in each edge component in CGSC. DAG update is realized by data collaboration between tasks in $C^{Cloud}$ and $B^{Edge}$ of the edge end component so that the task flow can be dynamically executed as the space changes.

### Task Allocation in the Edge-Edge Collaboration Mode

The Improved Heterogeneous Earliest Finish Time (IHEFT) algorithm process proposed in this paper comprises three stages: weight distribution stage, task priority distribution stage, and processor selection stage. Assume that *R* is used to represent the collection of processor resources: $R = \left\{CPUs, GPUs, FPGAs\right\}$. *GPUs* are the most widely used accelerators, and *FPGAs* can provide better performance-to-power ratios. They are used in multiple applications, including high-performance computing. They deliver superior performance for a wide range of applications, including high-performance computing (Bobda et al., 2022; Mendes et al., 2022). Use undirected graph $RG = \left(T, C\right)$ to describe the computing performance of tasks on different processors, where *T* represents the task node, $C = \left\{c_{11}, c_{12}, ..., c_{nm}\right\}, n \in N^*, m \in N^*$ represents the computing performance of tasks on different processors, where *n* represents the number of tasks and *m* represents the number of processors. Then the average computing performance of the task $\overline{compute}$ is shown in Formula 18:

$$\overline{compute_j} = \frac{\sum_{i=1}^{m} c_{ji}}{m}, j = 1,2,\ldots n \tag{18}$$

In the task priority assignment stage, its primary purpose is to solve the problem of establishing task priority lists in multi-source heterogeneous scenarios. In actual industrial scenarios, both edge tasks and terminal tasks have the characteristics of multiple task types, large scales, and strong associations. Therefore, the original weights of the edges between tasks cannot accurately reflect the priority of tasks in the edge-side collaboration mode. The task graph's priority in the edge-side collaboration mode needs to be determined according to the sum of the weights of the DAG merged graph's critical paths. According to the priority value, the DAG in the edge-side collaboration mode is sorted in descending order to form a task graph list, which is convenient for calculation and offloading in the later stage. The priority $rank\left(D_k\right)$ of the task graph $D_k$ in the edge-side collaboration mode is shown in Formula 19:

$$rank\left(D_k\right) = \sum_{i=1}^{m} cp_i, k = 1,2,\ldots,n \tag{19}$$

The priority $rank\left(D_k\right)$ of the task graph $D_k$ in the edge-edge collaboration mode is equal to the sum of the weights of all event edges on the graph's critical path. In the subsequent computation offloading operation, the processor resources will be allocated first from the task graph with the higher priority in the task graph list. The corresponding path list is constructed according to the task graph list, and the priority $rank\left(p_k\right)$ of path $p_k$ is shown in Formula 20:

$$rank\left(p_k\right) = \sum_{t_i \in p_k} \left(t_i + e_{i,succ(i)}\right) \tag{20}$$

Due to the high computational complexity of the algorithm, multiple task links, and extended business processes are some of the most complex problems in the industrial field. The task running status includes successful execution, execution blockage, and execution failure.

In the DAG segmentation graph in the edge-side collaboration mode, there are edge terminal graphs $\xi$, with a degree value of $o = \left\{o_1, o_2, ..., o_\xi\right\}$; there are terminal subgraphs ζ, with a degree

value of $o = \left\{o_1, o_2, ..., o_\zeta\right\}$. From this, the number of processors $\varpi_i$ required for the task graph $DAG_i$ to calculate the offload execution is shown in Equation 21:

$$w_j = \begin{cases} 1 & ,o_j = 0 \\ 1 & ,o_j \in ETS, \varpi_i = \begin{cases} \sum_{j=1}^{\xi} w_j & ,w_j \in EST \\ \sum_{j=1}^{\zeta} w_i & ,w_j \in TST \end{cases} \\ o_j & ,other \end{cases} \tag{21}$$

After the segmentation operation and the resource collection type determination, the processor allocation stage starts to allocate resources to the task based on the resource collection type identified by each segmentation (Azhar et al., 2022; Yang & Deyu, 2017). The task set after DAG segmentation and synchronization selects and schedules processors according to the computation offloading strategy in the IHEFT algorithm.

## EDGE-EDGE COLLABORATIVE COMPUTATION OFFLOADING ALGORITHM

The EECCO algorithm proposed in this paper is to make full use of the edge and terminal resources to achieve efficient task calculation by dividing the edge DAG task set and the terminal DAG task set through DAG; DAG synchronization; The processor allocates three steps to achieve. Among them, DAG synchronization reduces the time overhead caused by processing redundant tasks.

### DAG Segmentation Algorithm

Assuming that the *k*-th factory under the edge-side collaboration mode is adopted, where the deployed edge server and terminal components perform N and M tasks, respectively, then the DAG segmentation algorithm is shown in Algorithm 1:

The second line of the algorithm traverses the DAG in the edge part and the terminal part of obtaining the number of task nodes. The fourth line obtains the priority of each task node in the edge part and the terminal part. Lines 5-16 perform DAG in the segmentation process, tasks belonging to the critical path are placed in the critical task set, and tasks not belonging to the critical path are placed in the non-critical task set. The time complexity of running the algorithm in each factory is O(n), and the space complexity of the algorithm is *O(1)*. As tasks increase, their advantages become more apparent.

### DAG Synchronization Algorithm

We use set theory to synchronize the state information of the DAG segmentation graph obtained above. The tasks after synchronization are divided into two categories: critical task set and non-critical task set. The synchronization principle is shown in Algorithm 2:

The second line of the algorithm traverses each DAG segmentation graph's task nodes and calculates each task's average computing performance on the processor. Lines 3-5 calculate the execution status of each DAG in the edge part, and the fourth line is based on each. Each task's execution status is synchronized with the corresponding strategy, and the 8th line performs a merge update operation for the DAG in the cloud component. The algorithm's time complexity is *O(n)*, and the space complexity of the algorithm is *O(1)*. As tasks increase, their advantages become more obvious.

### Processor Allocation Algorithm

We use the IHEFT algorithm to perform processor allocation operations on the DAG segmentation-synchronization graph obtained above and sort the DAG task graph according to the critical path weight $rank\left(D_k\right)$, to obtain the priority list of different tasks in the edge-side collaboration mode;

**Algorithm 1.**
**DAG segmentation algorithm**

Input: $DAG_k^{Edge}, DAG_k^{Terminal}$

Output: $CTS_k^{Edge}, NCTS_k^{Edge}, CTS_k^{Terminal}, NCTS_k^{Terminal}$

1: function Partitioning ( $DAG_k^{Edge}, DAG_k^{Terminal}$ ):
2: *N*, *M* ⟵ order tasks based on level
3: while *N*, *M* is not empty, do
4: $P_i^{Edge/Terminal} = \frac{\sum w}{\left\| W \right\|} \times S_i \times \pi_i$
5: if it is an edge task
6: if it is a static class known task
7: $o^{Edge} = \begin{cases} same\,server & ,\max\left\{time_i^{\text{Edge}'}\right\} - time_j^{Edge} < \theta_3, i = \left\{1,\ldots,k\right\} \\ other\,server & ,\max\left\{time_i^{\text{Edge}''}\right\} - time_j^{Edge} < \theta_2 + \theta_3, i = \left\{1,\ldots,k\right\} \\ 0 & ,others \end{cases}$
8: else
9: $o^{Edge'} = \begin{cases} same\,server & ,\frac{s \times c}{num} - time_j^{Edge} < \theta_3, i = \left\{1,\ldots,k\right\} \\ other\,server & ,\frac{s \times c}{num} - time_j^{Edge} < \theta_2 + \theta_3, i = \left\{1,\ldots,k\right\} \\ 0 & ,others \end{cases}$
10: else
11: if it is industrial equipment in a mobile scenario
12: $o'' = \begin{cases} 1 & ,\frac{\left(M1-m\right)}{M1 \times T1} - \frac{\vartheta}{\vartheta_0} \times \frac{\left(M2-m\right)}{M2 \times T2} > 0 \\ & ,\frac{\left(M1-m\right)}{M1 \times T1} - \frac{\left(M2-m\right)}{M2 \times T2} > 0 \\ 0 & ,others \end{cases}$
13: else if it is a dynamic unknown task
14: $o^{Terminal} = \begin{cases} Edge\,server & ,\max\left\{time_i^{\text{Terminal}'}\right\} - time_j^{Terminal} < \theta_4, i = \left\{1,\ldots,k\right\} \\ 0 & ,others \end{cases}$
15: else
16: $o^{Terminal'} = \begin{cases} Edge\,server & ,\frac{s \times c}{num} - time_j^{Terminal} < \theta_4, i = \left\{1,\ldots,k\right\} \\ 0 & ,others \end{cases}$
17: $if\left(t_i \in CP\left(DAG^{Edge/Terminal}\right)\right)$
18: add task $t_i$ in $CTS^{Edge/Terminal}$
19: remove task $t_i$ from N or M
20: else
21: add task $t_i$ in $NCTS^{Edge/Terminal}$
22: remove task $t_i$ from N or M
23: end else
24: end if
25: end while
26: return $CTS_k^{Edge}, NCTS_k^{Edge}, CTS_k^{Terminal}, NCTS_k^{Terminal}$
27: end function

**Algorithm 2.**
**DAG synchronization algorithm**

Input: $CTS_k^{Edge}, NCTS_k^{Edge}$

Output: $CTS^{Edge'}, NCTS^{Edge'}, C^{Cloud}$

1: function Partitioning( $CTS_k^{Edge}, NCTS_k^{Edge}$ ):
2: N ⟵ order tasks based on level
3: while N is not empty, do
3: $A_t^{Edge}$
$CTS_k^{Edge}, NCTS_k^{Edge}$ based on level
4: $\begin{matrix} \vdots \\ Cloud \\ i,k \end{matrix} = \bigcup_{i,k}^{\infty} \left\{ CTS_k^{Edge}, NCTS_k^{Edge} \right\}$
4: if it is EGSC
5: $Status^{Edge} = \begin{cases} execution\,succeed & , A_t^{Edge} = A_t^{Edge} - A_{t+1}^{Edge} \\ execution\,blocking & , A_t^{Edge} \supset A_{t+1}^{Edge} \\ execution\,failed & , \varnothing = A_t^{Edge} - A_{t+1}^{Edge} \end{cases}$
6: $\copyright = \begin{cases} B^{Edge} = \varnothing & , execution\,succeed \\ B^{Edge} = A_t^{Edge} \oplus A_{t+1}^{Edge} & , execution\,blocking \\ B^{Edge} = A_t^{Edge} & , execution\,failed \end{cases}$
7: else
8: $C^{Cloud} = \bigcup_{i,k}^{\infty} \max_t \left\{ x \right\}, x \in \begin{matrix} \vdots \\ Cloud \\ i,k \end{matrix}$
9: end if
10: end else
11: end while
12: return $C^{Cloud}, CTS_k^{Edge'}, NCTS_k^{Edge'}$
13: end function

The tasks in each edge part are sorted by $rank\left(p_k\right)$ and the tasks are distributed to $\varpi$ processors. The processor allocation algorithm is shown in Algorithm 3:

The algorithm starts the function from the first line, the second line traverses each DAG split-synchronization graph's task nodes and calculates each task's average computing performance on the processor. The third line calculates the sum of the weights of the edges of the critical path of the DAG merged graph. Lines 4-8 calculate the priority of the DAG segmentation-synchronization graph path. Line 9 is sorted in descending order according to the priority of the task graph. The 10th line is sorted according to the descending order of the path priority. The 11th line assigns the task to the best on the processor. Line 12 returns related information such as the mapping task's processor set, and line 13 ends the function. The algorithm's time complexity is *O(1)*, and the space complexity of the algorithm is *O(1)*. With the increase of tasks, its advantages become more evident.

## SIMULATION EXPERIMENT AND RESULT ANALYSIS

### Purpose

In order to verify the EECCO algorithm proposed in this paper, the performance of the proposed EECCO algorithm was compared with similar offloading algorithms Local-Based Partial Reasonable

**Algorithm 3.**
**Processor allocation algorithm**

Input: $CTS_k^{Edge'}, NCTS_k^{Edge'}, CTS_k^{Terminal}, NCTS_k^{Terminal}$

Output: $rank(D_k)_j$, $rank(p_k)_j$, $\varpi_j$

1: function IHEFT( $CTS_k^{Edge'}, NCTS_k^{Edge'}, CTS_k^{Terminal}, NCTS_k^{Terminal}$ ):

2: $\overline{compute_j} = \frac{\sum_{i=1}^{m} c_{ji}}{m}$, $i \in \{1,2,...,\xi+\zeta\}$

3: $rank(D_k)_j = \sum_{i=1}^{m} cp_i$, $i \in \{1,2,...,\xi+\zeta\}$

4: if the task $t_i$ is the last task, then

5: the rank value of $t_i$ = its average execution time

6: else:

7: $rank(p_k)_j = \sum_{t_i \in p_k} \left( t_i + e_{i,succ(i)} \right)$

8: end if

9: Sort the DAG in a scheduling list by descending order of $rank(D_k)_j$ values

10: Sort the tasks in a scheduling list by descending order of $rank(p_k)_j$ values

11: Assign task $t_i$ to the best processor base on $\varpi_j$ list

12: return Set of processors with the mapping tasks, $rank(D_k)_j$, $rank(p_k)_j$

13: end function

Task Schedule Construction (LoPRTC) and Game-theoretic Greedy Approximation Offloading Algorithm (GT-GAOA) under the same experimental conditions (Hussain et al., 2020; Xu et al., 2019), mainly comparing task span *Makespan*, task average waiting for time *AWT* and average *Slack* value.

## Simulated Environment

Based on the simulator toolkit provided by SimGrid, a simulation environment for heterogeneous multi-core processors is built (Cornebize & Legrand, 2022). The computer used in the experiment is configured as Intel Core i5-7200U CPU @ 2.5GHz 2.7GHZ Dual-core processor, 8GB of RAM.

## Analysis of Calculation and Offloading Process

The edge-edge collaboration mode is adopted. Assuming the tasks running in the edge server and terminal industrial equipment are shown in Figure 8, the following will analyze the segmentation, synchronization, and resource scheduling process of the multi-DAG task graph realized by the EECCO algorithm under the distributed and heterogeneous computing environment.

Figure 8 shows the DAG task diagram in the edge-edge collaboration mode. The edge server performs 10 tasks; Terminal industrial equipment performs 1 task. Table 1 and Table 2 respectively show the calculation events of tasks in DAG1 on terminal industrial equipment and DAG2 on edge parts on different processors, in which the unit of communication cost and execution time is *s*.

According to the division principle, the DAG task division diagram can be obtained according to Algorithm 1, as shown in Figure 9; the processor allocation principle can be obtained according to algorithm 3 to obtain the processor set of the mapping task, as shown in Table 4:

**Figure 8.**
**Dag task diagram using edge-side collaboration mode**

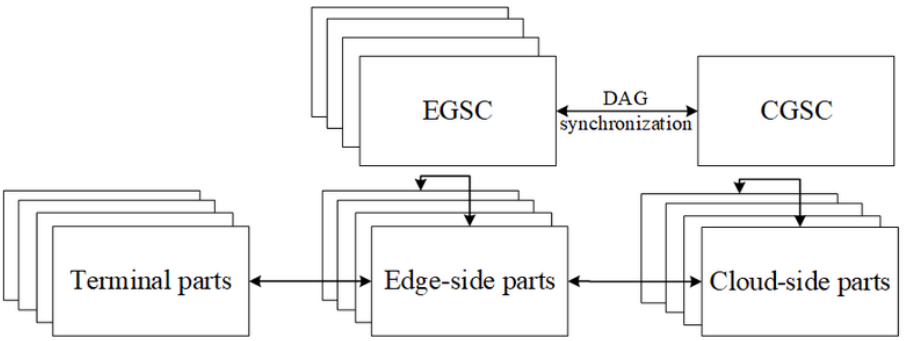


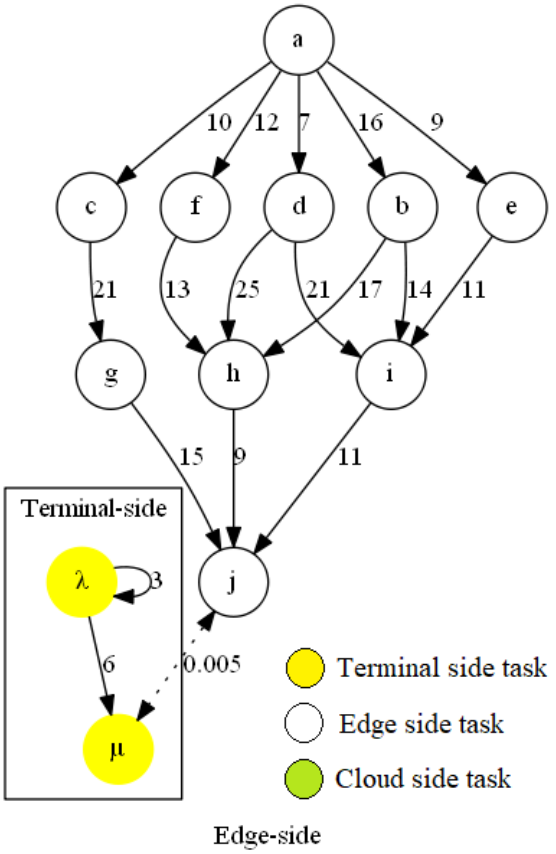

**Table 1.**
**The scheduling time of the DAG1 task set on the processor set**

| Task | $P_1^{Terminal}$ |
|---|---|
| $\lambda$ | 8 |
| $\mu$ | 6 |

**Table 2.**
**The calculation time of DAG2 task set on processor set**

| Task | $P_1^{ES1}$ | $P_2^{ES1}$ | $P_3^{ES1}$ | $P_1^{ES2}$ | $P_2^{ES2}$ | $P_3^{ES2}$ |
|---|---|---|---|---|---|---|
| $a$ | 9 | 11 | 4 | 6 | 8 | 1 |
| $b$ | 8 | 14 | 13 | 2 | 11 | 10 |
| $c$ | 6 | 8 | 14 | 3 | 6 | 11 |
| $d$ | 8 | 3 | 12 | 6 | 1 | 9 |
| $e$ | 7 | 8 | 5 | 4 | 5 | 2 |
| $f$ | 8 | 11 | 4 | 6 | 8 | 1 |
| $g$ | 2 | 10 | 6 | 1 | 7 | 3 |
| $h$ | 1 | 6 | 9 | 3 | 3 | 6 |
| $i$ | 13 | 7 | 15 | 10 | 4 | 12 |
| $j$ | 16 | 2 | 11 | 13 | 2 | 9 |
| $\lambda$ | 4 | 2 | 3 | 1 | 1 | 2 |
| $\mu$ | 2 | 2 | 3 | 1 | 1 | 1 |

**Figure 9.**
**DAG segmentation diagram using edge-edge collaboration mode**




## Analysis of Results

The time complexity of the EECCO algorithm is *O(n)*, and the actual data collected on the site of a blower is taken as the analysis target. Compared with the traditional centralized traversal method, the EECCO algorithm has a good advantage. Figure 10 shows the impact of an increase in the number of edge servers on the computation time overhead.

According to Equation 13, the DAG task segmentation graph's subgraph has a degree of {5,3,1}. In the edge-edge collaboration mode, edge tasks are allocated to 5 processors and terminal tasks to 1 processor. After the task set is unloaded by the EECCO algorithm, the corresponding relationship between the task and the processor is shown in Figure 12. Figure 11(b) shows that the same task set uses a naive CPOP algorithm as a computational offload strategy. Figure 12(a) and 12(b) respectively represent the offloading of tasks by LoPRTC and GT-GAOA algorithms for the same task set. As shown in Figure 11, the average task execution time of the EECCO algorithm is 42s, and that of the simple CPP algorithm is 57s. At the same time, it can be seen that the EECCO algorithm can reduce the number of processors when the task execution time is reduced. In Figure 11(a), the number of processors used at the edge is 6. In Figure 11(b), the number of edge processors used is 3.As can be seen from Figure 12, the average task execution time of the LoPRTC algorithm is 53s, and that of the GT-GAOA algorithm is 48s.In Figure 12(a), the number of edge end processors used is 6; In Figure 12(b), the number of edge processors used is 4.

**Figure 10.**
**The impact of the increase in the number of edge servers on computing time overhead**

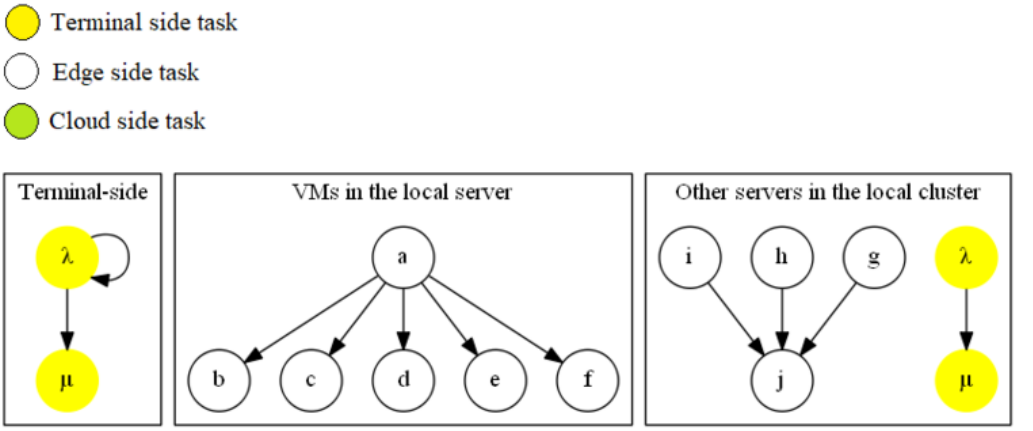
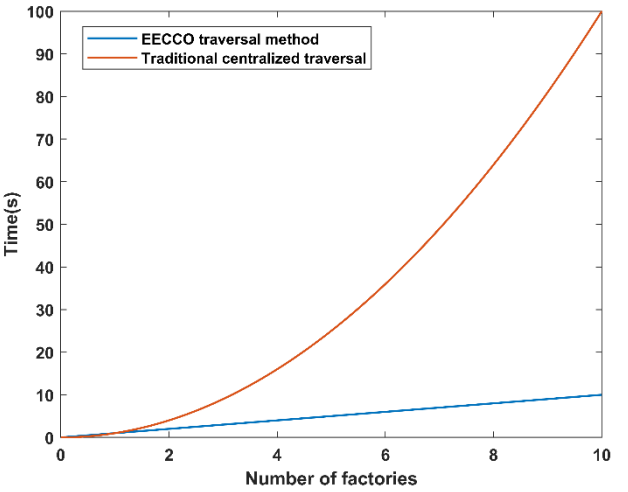


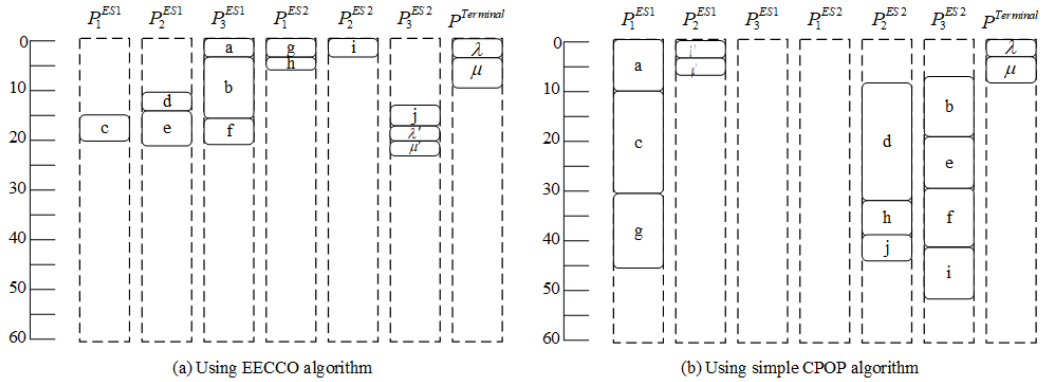

**Figure 11.**
**The impact of different network environments on the CECTS algorithm**




System processor utilization is an essential indicator of real-time system performance, representing the system's time characteristics and task status (Wu et al., 2022). Table 3 shows the average utilization rate of processor resources of the three algorithms. It can be seen from the figure that the EECCO algorithm has the highest average utilization rate of processors, which can give full play to the internal resources of the processor.

The three algorithms were used to schedule 10 DAG task graphs, respectively. Table 4 shows the comparison of three algorithms' *Slack* values in the case of 3-core processors. Three algorithms were adopted to schedule DAG tasks, and scheduling was carried out for the different number of DAG task models. The span *Makespan* values, average waiting time *AWT* and average *Slack* values of all tasks were obtained, as shown in Figure 13(a),13(b), and 13(c).

**Figure 12.**
**Computational offload diagram of LoPTRC and GT-GAOA Algorithms**

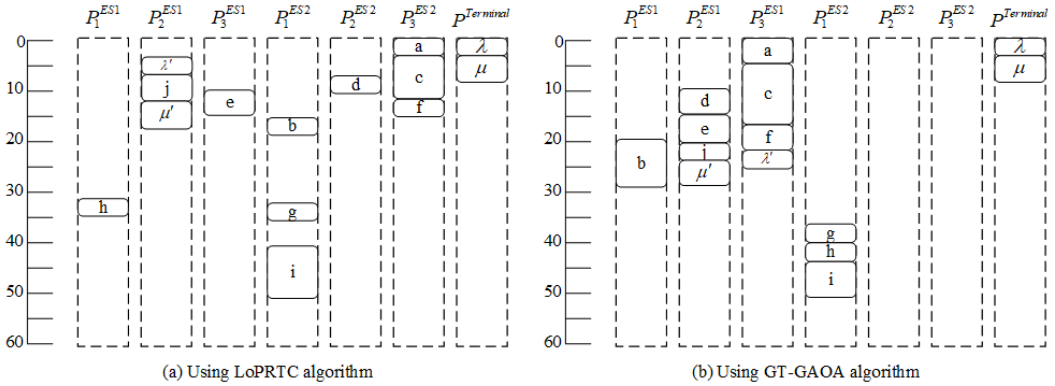



(a) Using EECCO algorithm (b) Using simple CPOP algorithm

**Table 3.**
**The average utilization of processor resources of the three algorithms**

| Algorithms | Average processor utilization at the edge (%) |
|---|---|
| EECCO | 67.3 |
| CPOP | 35.7 |
| LoPRTC | 36.5 |
| GT-GAOA | 36.3 |

**Figure 13.**
**Comparison of average waiting time, Makespan, and average Slack value of the four algorithms**




(a) Using LoPRTC algorithm (b) Using GT-GAOA algorithm

According to the experimental data, both will increase correspondingly with the DAG number increase in terms of the span of task offloading and average Slack. In general, the EECCO algorithm's performance is the best, followed by the LoPRTC and GT-GAOA algorithm, and the CPOP algorithm's performance is the worst. The EECCO algorithm reduces task offloading span by 26.6% on average compared with the CPOP algorithm from the experimental data. EECCO algorithm is 21.4% lower than the LoPRTC algorithm on average. Compared with GT-GAOA, the EECCO algorithm reduces by 10.91% on average. In terms of average Slack, the EECCO algorithm is 23.5% lower than the

**Table 4.**
***Slack* value of CECTS, CPOP, LoPRTC, GT-GAOA algorithm**

| DAG | EECCO | CPOP | LoPRTC | GT-GAOA |
|---|---|---|---|---|
| 1 | 1.42 | 1.92 | 1.62 | 1.67 |
| 2 | 2.69 | 3.45 | 2.98 | 3.11 |
| 3 | 3.77 | 6.03 | 4.99 | 5.49 |
| 4 | 5.23 | 7.23 | 6.22 | 6.20 |
| 5 | 6.60 | 9.65 | 8.03 | 8.34 |
| 6 | 8.41 | 11.80 | 9.67 | 10.24 |
| 7 | 10.09 | 13.88 | 11.51 | 12.01 |
| 8 | 11.19 | 15.27 | 12.79 | 13.50 |
| 9 | 12.29 | 17.19 | 14.79 | 15.38 |
| 10 | 14.27 | 18.76 | 15.95 | 16.33 |

CPOP algorithm. EECCO algorithm is 11.13% lower than the LOPRTC algorithm on average. The EECCO algorithm reduces by 10.90% on average compared with the GT-GAOA algorithm. The average waiting time of the EECCO algorithm is reduced by 58.9% compared with that of the CPOP algorithm. EECCO algorithm is 57.1% lower than the LoPRTC algorithm on average. EECCO algorithm is 65.7% lower than the GT-GAOA algorithm on average.

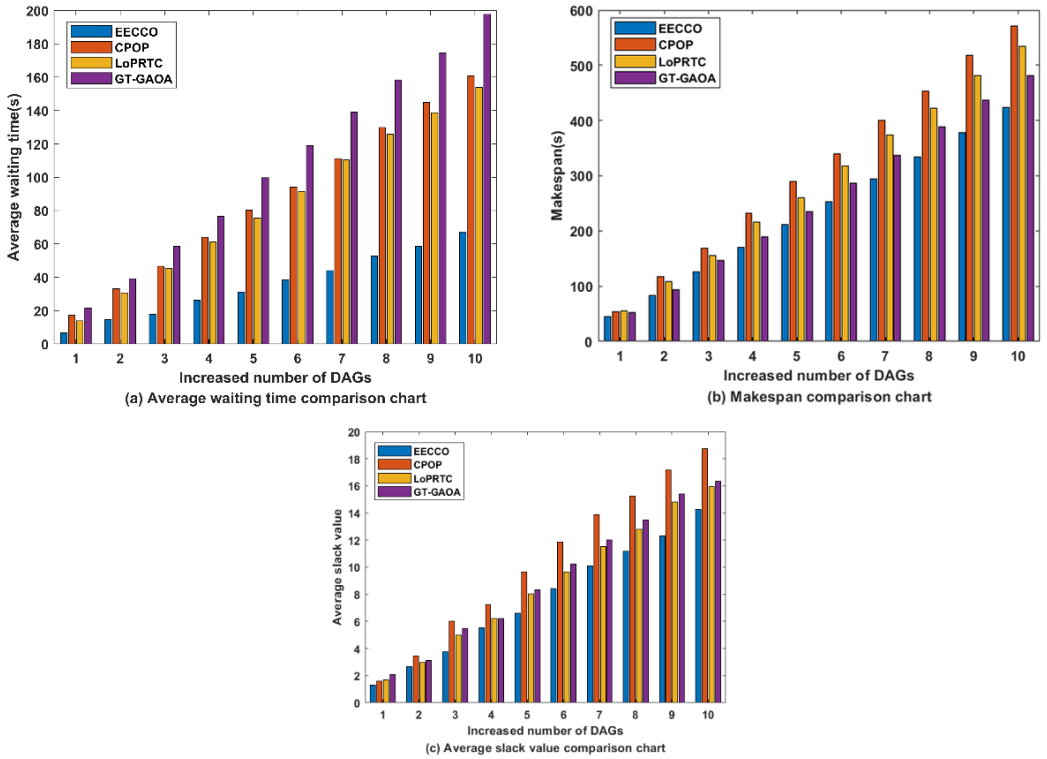
The offloading algorithm's fairness indicates the reliability of multiple DAG task offloading algorithms and is an important indicator that reflects whether the algorithm can reasonably handle tasks with different priority levels. Figure 14 shows the degree of fairness of the EECCO, CPOP, LoPRTC, and GT-GAOA algorithms.

The EECCO algorithm is superior to CPOP and GT-GAOA algorithm in terms of average task waiting time and average *Slack* value through the above experimental analysis. Simultaneously, in terms of algorithm fairness, the EECCO algorithm has high fairness and maintains an excellent steady-state with the increase of DAG number.

## DISCUSSION

With the continuous development of network information technology, the new industrial model led by smart manufacturing has started to be widely used. The existing computing offloading methods tend to lead to congestion caused by heavy load on an edge server. Meanwhile, the existing studies assume that each task is independent of each other, without considering the possible dependency between tasks. As a result, a large amount of idle computing time is not fully utilized on the computing nodes, which wastes the computing resources of the nodes and increases the task execution delay. In this paper, a computation offloading method for large-scale factory access in edge-edge collaboration mode is given for cloud manufacturing scenarios. The computation offloading problem in complex scenarios is solved by three steps, task graph partitioning, synchronization, and processor scheduling. Simulation results show that the EECCO algorithm proposed in this paper can reduce the time overhead of processing complex tasks compared to other offloading algorithms and enable an edge-edge collaborative approach to offload tasks to the appropriate processor in an overall increase in the speed of task processing. Associated tasks can be processed more efficiently in the case of resource-constrained large-scale edge-side servers. It solves the problem of difficult unloading due to the difference of actual scenarios and the high response delay caused by the increase of terminal industrial equipment.

**Figure 14.**
**The degree of fairness of EECCO, CPOP, LoPRTC, and GT-GAOA algorithms**




## CONCLUSION

China has now become an industrial power. Most of the methods used in manufacturing workshops are to provide dedicated production lines or rigid assembly lines. This solidified production method can cope with a certain amount of production, but in the face of large-scale production. For the production demand of the manufacturing workshop, the traditional production method is far from enough. Therefore, the method of computing offloading of large-scale factory access in the edge-edge collaborative mode in this paper is based on the fact that the demand of the manufacturing factory is greater than the actual production volume. For the manufacturing industry of large-scale manufacturing production mode, the problem of resource layout and allocation is indispensable. In order to solve the problem of collaborative operation between edge servers deployed in large-scale manufacturing workshops, a side-by-side collaborative computing architecture is proposed. The task scheduling method of large-scale factory access realizes the effective allocation of resources. In fact, the machine tools and equipment seen in ordinary workshops work in a discrete form, but in order to combine the needs of the task, this simple configuration cannot be realized. The EECCO algorithm divides the equipment and equipment into more levels. It has realized multi-level resource allocation, and changed the discrete working state into a multi-dynamic process working state. At present, many large-scale manufacturing workshops have an increasing demand for production flexibility, which is basically through the production process. Various methods to increase the flexibility of mass manufacturing production, such as the cluster layout of traditional manufacturing workshops, etc. Considering the existence of specific requirements, unexpected events during equipment operation, and the impact of unknown users on the system, the next step will be to improve and refine the

requirements analysis model, evaluate the effectiveness of the scheme in this paper after obtaining actual data, and further improve the performance of the algorithm. In the future, the task offloading scheme for multi-intelligence collaborative operation will be studied in depth to complement the work in this paper. At the same time, the next stage can start to carry out the technical landing, to realize the computing unloading platform. At present, due to the limitations of hardware and software and other factors, it is difficult to achieve the implementation of technology.

## ACKNOWLEDGMENT

This work was supported in part by the National Key Research and Development Program of China, under Grant 2019YFB2102002; in part by the National Natural Science Foundation of China, under Grant 62176122, 61871432, 62001217; in part by A3 Foresight Program of NSFC, under Grant No. 62061146002; the Natural Science Foundation of Hunan Province, grant numbers 2020JJ4275 and 2021JJ50049.

## CONFLICTS OF INTEREST

The authors declare no conflict of interest.

## REFERENCES

Azhar, M. W., Pericas, M., & Stenström, P. (2022). Task-RM: A resource manager for energy reduction in task-parallel applications under quality of service constraints. [TACO]. *ACM Transactions on Architecture and Code Optimization*, *19*(1), 1–26. doi:10.1145/3494537

Bobda, C., Mbongue, J. M., Chow, P., Ewais, M., Tarafdar, N., Vega, J. C., Eguro, K., Koch, D., Handagala, S., Leeser, M., Herbordt, M., Shahzad, H., Hofste, P., Ringlein, B., Szefer, J., Sanaullah, A., & Tessier, R. (2022). The future of FPGA acceleration in datacenters and the cloud. [TRETS]. *ACM Transactions on Reconfigurable Technology and Systems*, *15*(3), 1–42. doi:10.1145/3506713

Cai, J., Fu, H., & Liu, Y. (2022). Deep reinforcement learning-based multitask hybrid computing offloading for multiaccess edge computing. *International Journal of Intelligent Systems*, *24*(11), 6222–6243. doi:10.1002/int.22841

Cha, N., Wu, C., Yoshinaga, T., Ji, Y., & Yau, K.-L. A. (2021). Virtual edge: Exploring computation offloading in collaborative vehicular edge computing. *IEEE Access: Practical Innovations, Open Solutions*, *9*, 37739–37751. doi:10.1109/ACCESS.2021.3063246

Chen, H., Deng, S., Zhu, H., Zhao, H., Jiang, R., Dustdar, S., & Zomaya, A. Y. (2022). Mobility-Aware Offloading and Resource Allocation for Distributed Services Collaboration. *IEEE Transactions on Parallel and Distributed Systems*, *33*(10), 2428–2443. doi:10.1109/TPDS.2022.3142314

Cornebize, T., & Legrand, A. (2022). Simulation-based optimization and sensibility analysis of MPI applications: Variability matters. *Journal of Parallel and Distributed Computing*, *166*, 111–125. doi:10.1016/j.jpdc.2022.04.002

Darwish, R. (2022). A congestion-aware decision-driven architecture for information-centric Internet-of-Things applications. *International Journal of Computers and Applications*, *44*(4), 324–337. doi:10.1080/1206212X.2020.1738088

Guo, H., Liu, J., Zhang, J., Sun, W., & Kato, N. (2018). Mobile-edge computation offloading for ultradense IoT networks. *IEEE Internet of Things Journal*, *5*(6), 4977–4988. doi:10.1109/JIOT.2018.2838584

Hussain, A., Manikanthan, S., Padmapriya, T., & Nagalingam, M. (2020). Genetic algorithm based adaptive offloading for improving IoT device communication efficiency. *Wireless Networks*, *26*(4), 2329–2338. doi:10.1007/s11276-019-02121-4

Hussein, M. K., & Mousa, M. H. (2020). Efficient task offloading for IoT-based applications in fog computing using ant colony optimization. *IEEE Access: Practical Innovations, Open Solutions*, *8*, 37191–37201. doi:10.1109/ACCESS.2020.2975741

Khan, P. W., Abbas, K., Shaiba, H., Muthanna, A., Abuarqoub, A., & Khayyat, M. (2020). Energy efficient computation offloading mechanism in multi-server mobile edge computing—An integer linear optimization approach. *Electronics (Basel)*, *9*(6), 1010. doi:10.3390/electronics9061010

Laroui, M., Ibn-Khedher, H., Ali Cherif, M., Moungla, H., Afifi, H., & Kamel, A. E. (2021). SO-VMEC: Service offloading in virtual mobile edge computing using deep reinforcement learning. *Transactions on Emerging Telecommunications Technologies*, 4211.

Li, X., Huang, L., Wang, H., Bi, S., & Zhang, Y.-J. A. (2022). An Integrated Optimization-Learning Framework for Online Combinatorial Computation Offloading in MEC Networks. *IEEE Wireless Communications*, *29*(1), 170–177. doi:10.1109/MWC.201.2100155

Liao, Z., Peng, J., Xiong, B., & Huang, J. (2021). Adaptive offloading in mobile-edge computing for ultra-dense cellular networks based on genetic algorithm. *Journal of Cloud Computing*, *10*(1), 1–16.

Liu, Y., Liu, C., Liu, J., Hu, Y., Li, K., & Li, K. (2022). Mobility-Aware and Code-Oriented Partitioning Computation Offloading in Multi-Access Edge Computing. *Journal of Grid Computing*, *20*(2), 1–15. doi:10.1007/s10723-022-09599-x

Luo, S., Li, H., Wen, Z., Qian, B., Morgan, G., Longo, A., Rana, O., & Ranjan, R. (2021). Deep Learning and Blockchain with Edge Computing for 5G-Enabled Drone Identification and Flight Mode Detection. *IEEE Network*, *35*(1), 124–129. doi:10.1109/MNET.011.2000222

Mahdi, H. F., Alwan, M. H., Al-bander, B., & Sameen, A. Z. (2022). A Comparision of Node Detection Algorithms Over Wireless Sensor Network. *International Journal of Interactive Mobile Technologies*, *16*(7), 38–53. doi:10.3991/ijim.v16i07.24609

Martinelli, N. (2018). Getting to know StarlingX: The high-performance edge cloud software stack. *SuperUser.* https://superuser.openstack.org/articles/starlingx-overview/

Materwala, H., Ismail, L., Shubair, R. M., & Buyya, R. (2022). Energy-SLA-aware genetic algorithm for edge–cloud integrated computation offloading in vehicular networks. *Future Generation Computer Systems*, *135*, 205–222. doi:10.1016/j.future.2022.04.009

Mekala, M., Dhiman, G., Srivastava, G., Nain, Z., Zhang, H., Viriyasitavat, W., & Varma, G. (2022). A DRL-Based Service Offloading Approach Using DAG for Edge Computational Orchestration. *IEEE Transactions on Computational Social Systems*, 1–9. doi:10.1109/TCSS.2022.3161627

Mendes, F., Tomás, P., & Roma, N. (2022). Decoupling GPGPU voltage-frequency scaling for deep-learning applications. *Journal of Parallel and Distributed Computing*, *165*, 32–51. doi:10.1016/j.jpdc.2022.03.004

Mustafa, E., Shuja, J., Jehangiri, A. I., Din, S., Rehman, F., Mustafa, S., Maqsood, T., & Khan, A. N. (2022). Joint wireless power transfer and task offloading in mobile edge computing: A survey. *Cluster Computing*, *25*(4), 2429–2448. doi:10.1007/s10586-021-03376-3

Sheikh Sofla, M., Haghi Kashani, M., Mahdipour, E., & Faghih Mirzaee, R. (2022). Towards effective offloading mechanisms in fog computing. *Multimedia Tools and Applications*, *81*(2), 1997–2042. doi:10.1007/s11042-021-11423-9 PMID:34690529

Sun, Y., Wang, H., & Zhang, C. (2022). Balanced Computing Offloading for Selfish IoT Devices in Fog Computing. *IEEE Access: Practical Innovations, Open Solutions*, *10*, 30890–30898. doi:10.1109/ACCESS.2022.3160198

Topcuoglu, H., Hariri, S., & Wu, M.-Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, *13*(3), 260–274. doi:10.1109/71.993206

Wang, M., & Leelapatra, W. (2022). A Review of Object Detection Based on Convolutional Neural Networks and Deep Learning. [ISJET]. *International Scientific Journal Of Engineering And Technology*, *6*(1), 1–7.

Wang, Q., Gao, A., & Hu, Y. (2021). Joint power and QoE optimization scheme for multi-UAV assisted offloading in mobile computing. *IEEE Access: Practical Innovations, Open Solutions*, *9*, 21206–21217. doi:10.1109/ACCESS.2021.3055335

Wu, M., Chen, Q., & Wang, J. (2022). Toward low CPU usage and efficient DPDK communication in a cluster. *The Journal of Supercomputing*, *78*(2), 1852–1884. doi:10.1007/s11227-021-03942-x

Xu, D., & Zhu, H. (2022). Legitimate Surveillance of Suspicious Computation Offloading in Mobile Edge Computing Networks. *IEEE Transactions on Communications*, *70*(4), 2648–2662. doi:10.1109/TCOMM.2022.3151767

Xu, F., Qin, Z., Ning, L., & Zhang, Z. (2022). Research on computing offloading strategy based on Genetic Ant Colony fusion algorithm. *Simulation Modelling Practice and Theory*, *118*, 102523. doi:10.1016/j.simpat.2022.102523

Xu, J., Li, X., Liu, X., Zhang, C., Fan, L., Gong, L., & Li, J. (2019). Mobility-aware workflow offloading and scheduling strategy for mobile edge computing. International Conference on Algorithms and Architectures for Parallel Processing, Yang, L., Zhong, C., Yang, Q., Zou, W., & Fathalla, A. (2020). Task offloading for directed acyclic graph applications based on edge computing in Industrial Internet. *Information Sciences*, *540*, 51–68.

Yang, S., & Deyu, Q. (2017). Study on static task scheduling based on heterogeneous multi-core processor. *2017 International Conference on Computer Network, Electronic and Automation (ICCNEA)*. IEEE.

Yuan, Y., Yi, C., Chen, B., Shi, Y., & Cai, J. (2022). *A Computation Offloading Game for Jointly Managing Local Pre-Processing Time-Length and Priority Selection in Edge Computing. IEEE Transactions on Vehicular Technology*. IEEE.

Zhang, D., Cao, L., Zhu, H., Zhang, T., Du, J., & Jiang, K. (2022). Task offloading method of edge computing in internet of vehicles based on deep reinforcement learning. *Cluster Computing*, *25*(2), 1175–1187. doi:10.1007/s10586-021-03532-9

Zhang, K., Gui, X., Ren, D., Du, T., & He, X. (2022). Optimal pricing-based computation offloading and resource allocation for blockchain-enabled beyond 5G networks. *Computer Networks*, *203*, 108674. doi:10.1016/j.comnet.2021.108674

Zhang, Z., Li, C., Peng, S., & Pei, X. (2021). A new task offloading algorithm in edge computing. *EURASIP Journal on Wireless Communications and Networking*, *2021*(1), 1–21. doi:10.1186/s13638-021-01895-6

Zhu, D., Liu, H., Li, T., Sun, J., Liang, J., Zhang, H., Geng, L., & Liu, Y. (2021). Deep reinforcement learning-based task offloading in satellite-terrestrial edge computing networks. 2021 IEEE Wireless Communications and Networking Conference (WCNC), Zhu, T., Shi, T., Li, J., Cai, Z., & Zhou, X. (2018). Task scheduling in deadline-aware mobile edge computing systems. *IEEE Internet of Things Journal*, *6*(3), 4854–4866. doi:10.1109/JIOT.2018.2874954

*Junfeng Man, works in the School of Computer Science, Hunan First Normal University, Executive member of CCF Computer Application Special Committee, executive director of Hunan Computer Society, evaluation expert of national key R&D projects, vice president of Zhuzhou Big Data and Artificial Intelligence Industry Association, and young backbone teacher in Hunan Province. The main research directions are industrial big data, industrial Internet, and industrial software.*

*Longqian Zhao, PhD student at Nanjing University of Aeronautics and Astronautics, research direction is device-edge-cloud collaborative computing.*

*Bowen Xu, Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, current research interests include networking, swarm UAV networks, IoT networks with applications.*

*Cheng Peng, Associate professor. He received the M.E.and the Ph.D. degree in the School of Information Science and Engineering, Central South University, Chang Sha, China. he is a post-doctor in the automation and control major of Central South University. His current research interests include big data analysis, industrial equipment health analysis, and software engineering.*

*Junjie Jiang is an undergraduate student at Hunan University of Technology, his research direction is device-edge-cloud collaborative computing.*

*Yi Liu is at the Hunan University of Technology, Research interests include digital manufacturing and industrial big data.*