Multi-node Task Scheduling Algorithm for Edge Computing Based on Multi-Objective Optimization

Zheng Shi¹, Zhiguo Shi*¹

¹School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, Beijing, 100083, P.R. China
*Corresponding author’s e-mail: szg@ustb.edu.cn

Abstract. In edge computing, nodes are highly dynamic and resources are unbalanced. Due to the need to ensure near real-time response of tasks, when task scheduling is required, the resource availability of the target node needs to be estimated. It prevents the node's computing resources from being exhausted or the node from failing or going offline, when the task is pushed to the target node. In this paper, by studying multi-node task scheduling technology, a multi-objective optimization model is established, while considering the impact of completion time, energy consumption and load balancing on task scheduling. The task scheduling problem is transformed into a bidding model, and the offloading location of subtasks is determined in real time to meet the requirements of delay-sensitive tasks. Finally, simulation experiments are used to obtain the availability of the technology for multi-node task scheduling, which provides new ideas for task scheduling in edge computing.

1. Introduction
The current traditional cloud computing shows a trend from the center to the edge. The core of network applications spreads from the central server to the edge computing device. By being close to the data source, it reduces the amount of data sent to the cloud, saves bandwidth resources, reduces the transmission delay and the load of the core network, and improves the real-time processing capacity. It solves the single point failure and load balancing problems effectively, and improves the overall computing power of the network greatly[1]. It makes full use of the computing resources and computing power in the peer-to-peer network, decomposes the tasks with large computing workload, and schedules them to the preferred node for distributed computing, which will greatly improve the utilization efficiency of available computing resources and overall execution efficiency of workflow. In order to use the other computing nodes of edge computing to process the delay-sensitive tasks, it is necessary to schedule the tasks appropriately to ensure the load balance of the whole edge computing network and complete the task as soon as possible.

The remainder of this paper is organized as follows. Section 2 reviews some related studies. In Section 3, this paper presents an edge computing task scheduling technology based on multi-objective optimization. The effectiveness of the algorithm is further validated through simulation experiments in Section 4. Finally, Section 5 summarizes the research work of this paper.

2. Related work
As an important research content in edge computing, computing task scheduling has attracted the attention of researchers since the concept of edge computing was proposed. At present, there have been many related research results. The main task of task scheduling offload strategy is to research whether
the task generated by the user terminal needs to be offloaded and where the task is scheduled. This section understands the research direction and hotspots of task scheduling at this stage by analyzing the current research status of task scheduling strategies.

For the task scheduling problem in traditional cloud computing, literature [1][2][3] proposed a heuristic algorithm to shorten the total task completion time. It models the target. After normalization, the linear weighting method is used to construct the objective function. And the constructed objective function is used to improve pheromone updating strategies to avoid trapping in the local optimum in the ant colony algorithm. However, the above algorithms do not consider the transmission delay, and the calculation of the heuristic algorithm requires high computing power of the device, which is difficult to implement in edge computing. Literature [10] combines scheduling and multi-objective optimization problems, and proposes a chemical reaction multi-objective optimization algorithm based on Pareto optimal comparison strategy. Inspired by the chemical reaction processing mechanism, it uses four chemical reaction operators for DAG scheduling solution evolutionary search. This method has better optimal search performance. Literature [11] proposed a hybrid algorithm GA-DE based on genetic algorithm and differential algebra. The three factors of task completion time, cost and load balancing are used as constraints to construct the fitness function. The algorithm has better global search capability and faster convergence speed, and makes full use of resources to achieve load balancing. Literature [13] proposed a multi-objective task scheduling model based on execution time and execution cost. Based on this model, a Cloud Scalable Multi-Objective Cat Swarm Optimization Based Simulated Annealing (CSMCSOSA) was proposed, which can use dynamically changing clouds computing resources, while minimizing task processing time and cost. But heuristic algorithms have high computing power requirements for scheduling nodes and are difficult to implement at edge nodes. Literature [9] proposes a multi-objective integrated task scheduling algorithm in cloud computing, provides an optimization scheme based on multiple target execution time, execution cost and load balance, which improves CPU utilization. But it uses the exhaustive method to seek the optimal solution ,which has high time complexity. Literature [12] considers the Pareto trade-off between cost and completion time, and proposes a multi-objective optimization algorithm NSGA-II for cloud computing workflow scheduling. NSGA-II has found a reasonable solution in weighing task completion time and cost. It is superior to a single target solution. However, this research is limited to the NSGA-II algorithm, which has large limitations and is difficult to be widely used.

Aiming at the demand for real-time optimization in edge computing, literature [5][6] designed an optimal decision engine for QOS, which can effectively offload computing services to optimize throughput and delay. Literature [7] provides a game theory algorithm for computing offload, describes the structure of equilibrium distribution, proves the existence of equilibrium distribution, and provides a polynomial time dispersion algorithm to calculate equilibrium. But the complexity is higher. Literature [8] for multi-user collaborative computing scenarios, considering the user's selfish characteristics, based on Lyapunov optimization theory and peer-to-peer file paying off incentive scheme, proposes a multi-user collaborative computing distributed online task migration scheme. In the delay network it ensures user participation and collaborative processing tasks in a multi-hop manner. Compared with the existing single-hop centralized collaboration solution, it can significantly reduce system energy consumption and increase system throughput.

For the problem of load balancing, Literature [4] proposes a new task scheduling method (I-FASC) for the characteristics of tasks and resources, and then proposes an improved genetic algorithm (I-FA) firework detection mechanism by introducing an explosion radius, which reduces the task execution time and makes load balancing in a short time.

3. Task scheduling algorithm

Under the condition of high dynamic connection and resource imbalance, for common computing tasks, we need to optimize the distributed computing scheduling strategy, improve the processing performance of distributed computing task components, and reduce the overhead of computing task migration. When the edge service node application is sensitive to real-time service performance, the task processing
performance optimization is the goal, and the calculation task migration cost is the constraint, to achieve performance optimization at a controllable cost, to meet the real-time response needs of the edge service node application. Similarly, when the edge service node expects to reduce the overhead (such as reducing energy consumption or the resource or cost of the information resource management service center), the goal is to minimize the migration overhead of the computing task, and optimize the scheduling decision of the computing task with the user's minimum performance guarantee as the constraint. In addition, if the computing resource node expects to optimize task processing performance and overhead at the same time, it can assign different weights to the performance and overhead according to the preference of the edge service node. And it optimizes the weighted objective function of performance and overhead to achieve the goal of multi-objective collaborative optimization of performance and overhead.

For the demand of real-time response to task scheduling in edge computing, previous studies have mostly optimized single targets such as load or delay, and had less consideration of the energy limitation of equipment in edge computing. And some studies used ant colony algorithm and other heuristic algorithms are offline algorithms. The algorithm response speed is slow. It is difficult to adapt to the highly dynamic edge computing network. And the computing power of the dispatch center has high requirements. The scheduling algorithm for multi-objective optimization based on the bidding model proposed in this study is an online algorithm, which can meet the near real-time requirements of edge computing, and at the same time for multi-objective optimization, taking into delay, energy and load balance.

3.1. Multi-objective optimization model establishment

In order to solve the problem of low computing power or energy in edge computing, and to make better use of the idle resources of the surrounding edge devices, collaborative computing with the surrounding edge devices is an important way to improve resource utilization and reduce local execution energy consumption and delay. Therefore, how to schedule a task to a suitable computing device is a prerequisite for computing migration. If the selected node is highly mobile, it cannot communicate with each other due to too far distance during the execution of the task. So the task fails and the performance experience of the user is reduced. The selected node can not complete its original task due to power or computing power or make the delay of completing the task increase. It will have a great impact on the task.

Therefore, for the user who wants to offload the task, how to ensure that when the task is completed, the choice of the appropriate terminal to run the task is an important issue. For the node to provide services, how to guarantee the foundation of original task, it is the most important issue for the service node which participates in the task to choose the task that has the greatest benefit to itself from the tasks that are offloaded.

Figure 1 is a schematic diagram of a computing offload scenario. As shown in the figure, when requesting node 1 performs the migration of a computing task, it first detects the available edge computing nodes through communication technology. In order to detect a more suitable offload node, node 4 will act as the relay node to continue to detect the available edge computing nodes around. So through information aggregation, the current user can get all the surrounding nodes which can provide services and the communication time is below the threshold.

![Figure 1. The calculation offload scenario](image-url)
In the research scenario of task scheduling, it is assumed that there are \( L \) user equipments that need to perform a task migration request expressed as \( Q = \{q_1, q_2, q_3, ..., q_L\} \). Moreover, during the task migration process, the edge computing nodes that provide services have the same type of computing and communication properties as the nodes that request the tasks. So \( M \) edge computing nodes that provide services are represented as \( S = \{s_1, s_2, s_3, ..., s_M\} \). For each of the service nodes, there are only limited computing resources and communication resources. Here we represent the resource status of each service node \( s \) as \( \{C_s, B_s\} \), where \( C_s \) and \( B_s \) represent their size of computing resources and size of wireless communication resources. Each node can provide related offload services to nodes that communicate with it (direct communication and indirect communication). And the service node cannot affect its own tasks and maintain enough power.

For each task generated by the task request node, here we build a task model. Each subtask can be expressed as \( T_i = \{d_i, c_i, t_{i}^{\text{max}}\}, i \in N \), where \( N \) is the set of subtasks, \( d_i \) represents the size of the input data, \( c_i \) is the computing power required to complete the subtask (the number of CPU clocks or cycles required to complete the task), and \( t_{i}^{\text{max}} \) is the maximum delay limit to complete the task. For the convenience of research, it is assumed that the task dependency in task queue scheduling has no effect on scheduling. In order to select the appropriate service node for offloading, the requesting end can determine the priority of each task for each factor, and then adjust the task model through the weight coefficient to provide corresponding basis for future offloading decisions.

In this research scenario, each service node pays a certain "fee" for the task, i.e. makes a quotation. In order to obtain the maximum benefit for itself, the requesting node needs to select the service node with the most profit for each task to schedule the task.

Assuming that the requesting node obtains the topology map of the service node through detection, when the requesting node selects an appropriate unloading location in the service node, it needs to calculate the quotation made by the service node according to the characteristics of the calculation task and the topological relationship of the relevant service node. And then based on the quotation it gets corresponding ranking of service nodes. The definition of appropriate quotations are important prerequisites for task scheduling.

The main purpose of scheduling tasks to the edge service node for execution is to ensure task completion time, reduce energy consumption for executing tasks, and improve user experience when a single node has insufficient computing power. Therefore, when defining quotations based on tasks and node status, factors such as time, energy consumption, and communication delay should be considered comprehensively, and multiple objectives should be optimized. After the above analysis, the quotation of the service node uses the weighted sum of the task completion time, energy consumption and load balance of the task scheduled to the edge node.

The task completion time includes the two-way communication delay \( T_{l,\text{trans}} \) from the request node to the service node and the task calculation time \( T_{l,\text{exe}} \). the task completion time is as follows:

\[
T_l = T_{l,\text{exe}} + T_{l,\text{trans}}
\]  

(1)

The two-way communication delay includes the sending time \( T_{l,\text{in}} \), receiving time \( T_{l,\text{out}} \) and queuing time \( T_{l,\text{que}} \) of task \( l \), and the formulas are as follows:

\[
T_{l,\text{trans}} = T_{l,\text{in}} + T_{l,\text{out}} + T_{l,\text{que}}
\]  

(2)

\[
T_{l,\text{in}} = \frac{c_{l,\text{in}}}{b_j}
\]  

(3)

\[
T_{l,\text{out}} = \frac{c_{l,\text{out}}}{b_j}
\]  

(4)

Where \( c_{l,\text{in}} \) is the file size when task \( l \) is submitted; \( c_{l,\text{out}} \) is the file size when task \( l \) is executed and output; \( b_j \)-the bandwidth to service node \( j \).

Assume that the number of cpu cycles required for each calculation task follows an exponential random distribution with an average value of \( \rho \). The computing power of the base station \( j \) is determined by its cpu frequency and is denoted as \( f_j \). Therefore, if the base station \( j \) processes a computing task
with an arrival rate of $\lambda$, the dwell time of the task, i.e. the queuing time $T_{i,que}$ can be obtained by an M/M/1 queue model. The formulas are as follows:

$$T_{i,que} = \frac{\lambda}{\left(\frac{f_j}{\rho} \cdot \left(\frac{f_j}{\rho} - \lambda\right)\right)}$$

(5)

The execution time of a task is related to the computing resource of the service node $j$ (cpu computing capability $f_j$) and the calculation amount of the task $i$ (the number of clock cycles required to execute the task $c_i$). The formulas are as follows:

$$T_{i,exe} = \frac{c_i}{f_j}$$

(6)

The energy consumption $E_i$ scheduled for execution of task $i$ should include the energy consumption $E_{exe}$ of the computing task of the service node and the energy consumption $E_k$ consumed by the node due to communication forwarding. The energy consumption for task execution is shown in the following formula. Where $n$ represents the number of relay nodes from the request node to the service node. Therefore, the energy consumption of task execution is the sum of the communication energy consumption of all relay nodes and the computational energy consumption of service nodes performing tasks.

$$E_i = \sum_{k=1}^{n} E_{k,trans} + E_{exe}$$

(7)

The energy consumption $E_{exe}$ of the calculation task $i$ is mainly related to the calculation power $P_j$ of the calculation node $j$ and the task execution time $T_{i,exe}$.

$$E_{exe} = T_{i,exe}P_j$$

(8)

The energy consumption $E_{k,trans}$ of the transmission task $i$ is only related to the data transmission amount $d_i$, the data receiving bandwidth $b_k$, and the data receiving power $P_k$.

$$E_{k,trans} = \frac{d_i}{b_k}P_k$$

(9)

For multi-objective optimization, load balancing must also be considered. Load balancing is usually used to describe the load of a virtual machine cluster. It is an important indicator for measuring the efficiency of virtual machines. This paper establishes a load balancing evaluation model. Use $p_j$ to represent the performance of the service node $j$. The default weight of the service node is as follows:

$$W_d(j) = \frac{p_j}{\sum_{j=1}^{n} p_j}$$

(10)

$U_c(j)$, $U_m(j)$, $U_d(j)$ and $U_b(j)$ respectively represent the occupancy rate of the service node's cpu, memory, disk I/O and network bandwidth. Different indicators have different effects on node performance, $K_c$ $K_m$ $K_d$ $K_b$ respectively represent the impact factors of CPU, memory, disk I/O and network bandwidth on the server, and $K_c + K_m + K_d + K_b = 1$. The real-time load ratio $W_N(j)$ of the performance of each node is as follows:

$$W_N(j) = K_c * U_c(j) + K_m * U_m(j) + K_d * U_d(j) + K_b * U_b(j)$$

(11)

In the Least Connections method, when a request is distributed, the number of connections increases by 1. And when the request processing is completed, the number of connections decreases by 1. The improved model in this paper will also consider the scheduled The number of connections of the task $N_j = \{N_1, N_2 ... N_n\}$, the number of connections of each node is as follows:

$$W_c(j) = \frac{N_j}{\sum_{j=1}^{n} N_j}$$

(12)
When the load of the service node is heavy, the response time of the request becomes longer. So the response delay is also an important factor to measure the real-time load of the service node. Extract the last task request time of each service node in the cycle, \( T_j = \{T_1, T_2, \ldots, T_n\} \), the proportion of service node response time is as follows:

\[
W_t(j) = \frac{T_j}{\frac{1}{n} \sum_{j=1}^{n} T_j}
\]  

(13)

A mathematical model is established based on the real-time performance of the nodes, the number of connections and the response time. And the weight of each server node is calculated as follows:

\[
W_f = (W_n(j) + W_c(j) + W_t(j)) \times W_d(j)
\]  

(14)

The smaller \( W_f \), the lower the service node load.

The task execution time takes into account the computing power of the service node and the communication delay time between the request node and the service node; the energy consumption function also considers the energy consumption of the service node and the energy consumption of the relay node; the load balancing function considers the service node load balancing problem. Therefore, when defining the quotation function, it is reasonable and feasible to combine the three to form a metric. Therefore, this article defines the quote function of the service node as shown in the following formula:

\[
O_i = \alpha E_i + \beta T_i + \gamma W_j
\]  

(15)

Among them, \( \alpha + \beta + \gamma = 1 \), \( \alpha, \beta, \gamma \) are weight coefficients. And the adjustment of these three coefficients can reflect the importance of different task delays or energies in measuring quotes. From this formula, it can be concluded that the lower the task completion time, execution energy consumption, and load balancing index, the lower the offer made to the service node. The task scheduling node establishes a quotation ranking for each service node through tasks. The lower the quotation, the more inclined the task scheduling node is to schedule tasks to run on the service node.

3.2. Bidding model optimization and adaptive algorithm

When the requesting node wants to schedule tasks to run on other service nodes, the most important question is which service node the requesting node chooses for offload execution. If there are many request tasks and the service node resources are limited, how to schedule the tasks? In response to these two problems, this section proposes a task scheduling optimization strategy based on the bidding model.

**Formalizing the bidding model.** This section studies the task scheduling process based on the idea of bidding process. In the bidding model, there are two roles: bidder and inviter (those who call for bid). In this model, the inviter provides labor and quotation, and the bidder selects the inviter with the lowest bid by quotation. The bidding model of task scheduling problem is shown in the figure below:

![Bidding model for task scheduling](image)

Figure 2. Bidding model for task scheduling
The inviter in the figure is an edge computing node that provides computing. Here, we can express the computing resources as a service node to provide an idle VM (virtual machine), that is, the lease time for the bidder to sell his idle VM. The bidder refers to each task in the node requesting to offload the task. The bidder purchases the running time of the VM of the service node by bidding. The inviter uses the bidding function $O_i$ of the service node in the previous section for his bidding standard, that is, the formula (15) as shown. When the bidder receives bids from multiple inviters, the bidder selects the inviter with the lower bid.

Before starting to calculate the quotation, first arrange the tasks in descending order according to the required amount of calculation, and then perform the quotation calculation. This is a balanced scheduling idea. In order to achieve balanced scheduling between tasks and nodes, it can schedule long tasks to nodes with stronger processing power as far as possible.

In the bidding process, consider a situation where the service node, i.e., the inviter, bids through the price of formula (15) in each bidding process. But when multiple tasks are scheduled, it is assumed that task $i$ is scheduled to node $j$. It is difficult to obtain the load of node $j$ immediately after task scheduling. And querying the load of node $j$ again will also occupy bandwidth and increase delay. Therefore, this paper adopts a real-time adaptive price compensation strategy for load balancing. When scheduling tasks for multiple tasks at a time, after node $j$ bids for task $i$ and wins the bid, the real-time adjustment of the bid of node $j$ afterwards, according to the estimate completion time of task $i$ increases the bid of node $j$. That is, the load of the representative node $j$ increases. It reduces the possibility of the node $j$ winning the bid, and improves the load balance of the entire system. As shown in the following formula:

$$O_{i+1} = \alpha E_{i+1} + \beta (T_{i+1} + T_i) + \gamma W_{i+1}$$  \hspace{1cm} (16)

The previous research analyzes that by transforming the task scheduling problem into a bidding problem. The bidder's price modeling during the bidding process is established. And the price compensation strategy is used to improve the load balancing. Figure 3 shows the specific flow chart of the bidding process.

First, Initialize bidders and inviters. The bidder refers to the request to offload the scheduled task (generated by the request node), and the tasks are arranged in descending order. The bidder is the edge computing node that the inviter can communicate through D2D and provides computing resources. The labor force for the bidding is the computing resource provided by the computing node. Second, the inviter first quotes the tasks provided by the bidder according to formula (15). Third, the bidder bids. And each inviter makes a quotation for the task provided. The bidder selects the transaction according to the bid price provided by the inviter. The inviter with the lower price wins the bid. If the bidder and the inviter agree with each other, the bid is successful. Fourth, if during the bidding process, each bid wins a bid, the inviter adjusts the bid for the next task through the price compensation strategy (16) and rejoins the next round of bidding process to bid. Fifth, if the transaction is successful, the bidder and the inviter reach a transaction, requesting the node to schedule the task to run on the service node represented by the inviter. If the transaction fails, you can choose to conduct the next bidding, or abandon the bidding.
Begin

Initialize bidders and tenderers

the tenderer quotes the tasks provided by the bidder

The bidder selects winning bidder

Quote again

Bid success

Trade success

End

Figure 3. Bidding process

After the above analysis, the solution to the problem of computing task scheduling proposed in this paper is the task migration process based on the bidding model, that is, when the requesting node chooses node to offload, it selects the task offloading node through the bidding process in order to solve the scheduling of task offloading by multiple users problem. Described below is the complete process of task scheduling.

First, the node that needs to offload the task directly or indirectly detects the edge computing nodes available within n relay nodes through communication, and then constructs the communication topology map between the request node and each service node. The selection of service nodes should satisfy that the nodes have sufficient computing resources and energy, and the mobility is low. Second, Then the scheduling terminal sorts the tasks in descending order, calculates the calculation delay and energy consumption of the task at each candidate service node according to the task’s calculation amount and output data and other attributes, and then calculates the load balancing degree according to the current state of the candidate service node. At the same time, it is necessary to estimate the communication delay and energy consumption with the candidate service node according to the shortest path algorithm of the graph. Third, taking the task scheduling problem as the bidding process. Use the request node and service node as bidders and inviters, and select the unloading location according to the bidding process established in the previous section. Fourth, according to the result of the previous bidding, the requesting node offloads the task to the service node for execution. If an unexpected bidding fails, it can choose to abandon the bidding or proceed to the next bidding process. Fifth, the requesting node offloads the calculation task to the service node selected through the bidding process for execution. Sixth, the service node executes the scheduled task and returns the processing result to the requesting node.

Table 1 describes the task scheduling algorithm based on the bidding model. The algorithm first obtains candidate edge computing nodes that can be offloaded for the task of the requesting node, and then according to the bidding model, each candidate node bids for the task. In the bidding process, the service node with the lowest bid wins the bid, then assigns the task to the node, then adjusts the load balancing degree of the node in real time, adjusts the quotation after the node, and then bids for the next task until all service nodes There are tasks or tasks that have no candidate nodes to assign. Finally, if there are tasks that are not successful in bidding, re-evaluate bids until all tasks are assigned to the
service node. Suppose the number of tasks to be offloaded is n, and the number of request nodes is m. The main step of the algorithm is to estimate the quotation of each candidate node for each task. Therefore, the time complexity is O(mn).

Table 1. Task scheduling algorithm based on bidding model.

| Algorithm 1: Task scheduling algorithm |
|----------------------------------------|
| **Input:** task set T to be scheduled and service set S |
| **Output:** tasks and service nodes whose tasks are offloaded |
| 1: sort (T) |
| 2: for taski in T: |
| 3: Si = getService(taski) |
| 4: for sj in Si: |
| 5: pricesj = quote (taski, sj) |
| 6: n = select(sj) |
| 7: price sj = price sj * pow (δ, n) |
| 8: if pricesj < pricesj-1 |
| 9: bid(taski) = sj |
| 10: final |
| 11: return (taski, bid(taski)) |

4. Experimental results and analysis

4.1. Experimental environment.
CloudSim was launched in 2009 by the Cloud Computing and Distributed Systems Laboratory of the University of Melbourne, Australia. A major advantage of CloudSim is that it supports cloud computing simulation experiments on a single physical node, and supports both Windows and Linux operating systems. Through the current mature virtualization technology, the resources of the data center are virtualized into resource pools, providing data-based The modeling and simulation functions of the center's virtualized cloud. The open source CloudSim platform can enable researchers to focus on specific algorithm problems and promote the rapid development of cloud computing task scheduling algorithms.

4.2. Contrast algorithm.
SSA (Sequenced Scheduling Algorithm): assign tasks to all VMs in sequence, and then, when reallocating newly arrived tasks, target the first VM until all tasks are completed. The SSA algorithm can ensure that the number of tasks performed by each VM is approximately the same to achieve load balancing. FFCSA (First Come First Service Algorithm): According to the first-come-first-served way, after the task is assigned to the VM, the new task is directly assigned to the VM that completes the task first, until all are completed task. The FCFSA algorithm can reduce the idle time of VMs, but VMs with strong processing power may be overloaded and unable to achieve balanced load distribution. BSA (Balance Scheduling Algorithm): arrange tasks in descending order by size (Million Instructor), and then arrange VMs in descending order by processing capacity (MIPS), after reordering task sets and VM sets, And then call the SSA algorithm to schedule the task. The main goal of the BSA algorithm is to schedule long tasks to VMs with stronger processing power, and VMs with weaker processing power execute short tasks in order to achieve balanced scheduling between tasks and VMs, but this will reduce the part of the VM Resource utilization.
4.3. Experimental results.
The figure below shows the task completion time and energy consumption. The eight tasks arrived four times at times 0, 220, 440, and 660. As can be seen from Figure 4, when the task volume is small, the completion time of each algorithm is not much different. With the increase of the task volume, the completion time of each algorithm increases, the algorithm begins to show advantages, in 32 tasks When both are completed, this algorithm is 6.6% faster than the second fastest FCFSA algorithm.

![Figure 4. Task completion time](image)

It can be seen from Figure 5 that the total energy consumption of the algorithm to complete all tasks, including calculation energy consumption and transmission energy consumption, is the lowest. This is mainly because the algorithm has the fastest task completion time. According to the built-in energy consumption model of CloudSim, the lowest total energy consumption is calculated.

![Figure 5. Total task energy consumption](image)

The following figure is the time required for the algorithm function to run. The simulation scheduling algorithm will traverse the task once, so BSA, SSA, and FCFSA are all constant time complexity. The time complexity of this algorithm is O(n), and n is the number of offloadable node. The actual running time of this algorithm is three to four times that of other algorithms, but the time is in the order of microseconds, and the impact on the task completion time is much smaller than the effect of the scheduling results. And the running time of the algorithm is much shorter than other heuristic algorithms, which satisfies the constraint of the limited computing power of the edge computing nodes.
As can be seen from Figure 7, the memory footprint of the algorithm function has not increased much compared to the BSA, SSA, and FCFSA algorithms, and the space complexity is an order of magnitude, fully satisfying the limited memory space limitation of edge computing nodes. Therefore, this algorithm can be deployed on edge computing nodes without affecting performance.

5. Conclusions
In order to simultaneously consider the three optimization goals of task completion time, energy consumption, and load balancing during task scheduling in an edge computing environment with limited node computing power, a multi-node optimization-based multi-node task scheduling technology based on multi-objective optimization is proposed. A multi-objective optimization unified model is used to optimize the bidding model, and an adaptive balanced scheduling strategy is used to optimize the bidding strategy in the bidding process so that the tasks are more evenly scheduled to each node. Experiments were conducted in the CloudSim simulation environment. The experiments proved the effectiveness of the algorithm, and the complexity satisfies the constraints of running on edge devices. In actual situations, task scheduling is more complicated. In the future, the dependencies between tasks and complex arrival situations will be considered.
References

[1] Maray M, Jhumka A, Chester A, et al. (2019) Scheduling Dependent Tasks in Edge Networks. In: 2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC). London. pp. 1-4.

[2] Lee J, Kim J, Pack S, et al. (2019) Dependency-Aware Task Allocation Algorithm for Distributed Edge Computing. In: 2019 IEEE 17th International Conference on Industrial Informatics (INDIN). Helsinki. pp. 1511-1514.

[3] Sun P, Zhang H, Ji H, et al. (2018) Task Allocation for Multi-APs with Mobile Edge Computing. In: 2018 IEEE/CIC International Conference on Communications in China (ICCC Workshops). Beijing. pp. 314-318.

[4] Wang S, Zhao T, Pang S. (2020) Task Scheduling Algorithm Based on Improved Firework Algorithm in Fog Computing. IEEE Access, 8: 32385-32394.

[5] Apat H K. (2019) An Optimal Task Scheduling Towards Minimized Cost and Response Time in Fog Computing Infrastructure. In: 2019 International Conference on Information Technology (ICT). Bhubaneswar. pp. 160-165.

[6] Samanta A, Chang Z, Han Z. (2018) Latency-oblivious distributed task scheduling for mobile edge computing. In: 2018 IEEE Global Communications Conference (GLOBECOM). Abu Dhabi. pp. 1-7.

[7] Jošilo S, Dán G. (2018) Decentralized scheduling for offloading of periodic tasks in mobile edge computing. In: 2018 IFIP Networking Conference (IFIP Networking) and Workshops. Zurich. pp. 1-9.

[8] Zhang Q, Gui L, Hou F, et al. (2020) Dynamic Task Offloading and Resource Allocation for Mobile Edge Computing in Dense Cloud RAN. IEEE Internet of Things Journal, 7: 3282-3299.

[9] Langhnoja H K, Joshiyara H A. (2019) Multi-Objective Based Integrated Task Scheduling In Cloud Computing. In: 2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA). Coimbatore. pp. 1306-1311.

[10] Xiao X, Li Z. (2019) Chemical Reaction Multi-Objective Optimization for Cloud Task DAG Scheduling. IEEE Access, 7: 102598-102605.

[11] Li Y, Wang S, Hong X, et al. (2018) Multi-objective task scheduling optimization in cloud computing based on genetic algorithm and differential evolution algorithm. In: 2018 37th Chinese Control Conference (CCC). Wuhan. pp. 4489-4494.

[12] Shishido H Y, Estrella J C, Toledo C F M. (2018) Multi-objective optimization for workflow scheduling under task selection policies in clouds. In: 2018 IEEE Congress on Evolutionary Computation (CEC). Rio de Janeiro. pp. 1-8.

[13] Gabi D, Zainal A, Ismail A S, et al. Scalability-Aware scheduling optimization algorithm for multi-objective cloud task scheduling problem. In: 2017 6th ICT International Student Project Conference (ICT-ISPC). Skudai. pp. 1-6.