Dynamic Load Balancing Algorithm Based on Optimal Matching of Weighted Bipartite Graph

WEI HOU, LIMIN MENG, XUQING KE, AND LINGJUN ZHONG
College of Information Engineering, Zhejiang University of Technology, Hangzhou 310000, China
Corresponding authors: Wei Hou (weihou@zjut.edu.cn) and Limin Meng (mlm@zjut.edu.cn)
This work was supported in part by the National Natural Science Foundation of China under Grant 61871349.

ABSTRACT When the server cluster is processing concurrent task requests, if the performance difference among servers is not fully considered, task allocation will be unreasonable, which will lead to an increase in task making span and a decrease in cluster resource utilization. As one of the core technologies of server cluster, load balancing is used to balance the load of each server by allocating tasks to each server through an algorithm before the task processing. Therefore, this paper proposes a dynamic load balancing algorithm based on optimal matching of weighted bipartite graph. First, we constructed a bipartite graph with servers and tasks as vertices. The management server collects the load indicators of each server in the cluster in real time, using the real-time processing rate of each server as the load indicator. Each edge of the bipartite graph is determined by comparing the expected completion time of the tasks with the load of each server. The degree of matching between each task amount and each server load capacity is defined as the weight matrix of the edges, and the bipartite graph is weighted to construct a weighted bipartite graph. The Kuhn-Munkres algorithm was used to solve the optimal matching of the weighted bipartite graph, and the optimal assignment of tasks to servers was achieved based on the result of the optimal matching. The proposed algorithm fully considers the differentiation of each task amount and each server load capacity. By building a server cluster example and conducting comparison experiments, it is demonstrated that the algorithm can achieve load balancing of the server cluster and improve the resource utilization efficiency of the cluster, while offsetting the extra time overhead caused by the algorithm.

INDEX TERMS Load balancing, server cluster, bipartite graph matching, optimal matching, request scheduling.

I. INTRODUCTION
Nowadays, there is an era of high-speed communication network developments. With the gradual enrichment of network service content and the growth of a variety and number of smart devices, the Internet has become an important and indispensable tool in life. According to a report on emerging technology trends released in the U.S., more than 100 billion devices are expected to be connected to networks by 2045. According to a white paper published by Seagate and the Internet Data Center, the volume of data worldwide will reach 163 ZB by 2025, and more than a quarter of this data will be real-time communication data.

With this comes an exponential increase in concurrent access requests, and as a large number of requests are queued and stacked in the task pool, the traditional single server processing mode will cause an increase in make-span of tasks, congestion, overload and other problems. In serious cases, this can also lead to a server crash. This not only poses a huge challenge to the processing performance and data security of the platform server, but also causes an irreversible negative impact on the user experience.

Therefore, the industry mostly uses multiple servers to form a server cluster system. Multiple servers within the system work together to process parallel requests from users and provide external services [1]. The server cluster system can quickly switch to the remaining servers for response and data backup when a single server crash [2]. It improves the
computing response speed and ensures the stability of the system and the security of data. The server cluster system can reduce the average waiting time of users, reduce the energy consumption of the system, and expand the throughput of the system. There are some performance differences between different servers in the cluster system. If the tasks assigned to each server do not consider this problem, some servers will run under a high load, resulting in increased task processing time or failure to complete tasks, and some servers will run under load for a long time, resulting in a waste of system resources. This will affect the advantages of server cluster. One of the key issues in server cluster system is how to reasonably distribute tasks to each service node under the premise of meeting the maximum service demand and ensuring service quality. Load balancing technology is a key technology for solving this problem. A reasonable load balancing algorithm can make the server resources more efficient, even, and reasonable, thus improving the system performance, increasing the throughput, and improving the flexibility and stability of the system.

By considering the complexity of different task requests and the performance differences of each server in the cluster, this study proposes a dynamic load balancing algorithm based on the optimal matching of weighted bipartite graph. In the proposed algorithm, we first construct a bipartite graph with the server and task nodes as vertices. The ratio of task volume to actual task completion time was used as the load indicator for the server [3]. The load indicator of each server in the server cluster was collected in real time by the management server. The expected completion time of each task is compared with the real-time load of each server to determine each edge of the bipartite graph. The edge weight matrix of the bipartite graph is generated by calculating the matching matrix between the task volume and the load capacity of each server, the weighted bipartite graph to construct the weighted bipartite graph. The Kuhn-Munkres algorithm [4] with depth-first traversal is used to find the maximum weight match of the weighted complete bipartite graph, which is the optimal matching of the original weighted bipartite graph. Finally, according to the optimal matching results, the task was assigned to the designated server for processing. In this manner, real-time scheduling of tasks and dynamic load balancing of the server cluster can be achieved.

The main contributions of this paper can be summarized as follows.

(i) Design of scheduling model based on proposed algorithm. The model realizes the real-time scheduling of task requests, and distributes tasks to the server cluster for execution reasonably. Improve the efficiency and stability of the server cluster.

(ii) Propose a dynamic load balancing algorithm based on the optimal matching of weighted bipartite graph. The assignment of tasks to servers is accomplished by calculating the optimal matching results such that the load of the server cluster is relatively balanced.

(iii) Build a server cluster test example according to the proposed algorithm. Two experiments were conducted to demonstrate that the proposed algorithm had better load balancing effect.

The rest of this paper is organized as follows. In Section II, some related works are reviewed. Section III gives the problem statement, establishes the scheduling model and evaluation index of the proposed algorithm. Section IV introduces the algorithm flow in detail, analyzes the time complexity of the proposed algorithm. Section V conducts extensive experiments to evaluate the practical effects of the proposed algorithm by building test examples. Finally, in Section VI, the work of this paper is concluded and gives some possible directions for future work.

II. RELATED WORK

The load balancing algorithm needs to consider the complexity of the tasks as well as the processing capacity of each server in the cluster, and distribute the tasks to the servers in a reasonable manner. On the premise of ensuring effective task processing, the load of each server is relatively balanced. It avoids the load skewing phenomenon [5] that occurs when a few servers are running under a high load, whereas the rest are running under a low load. Load balancing algorithms can be divided into static load balancing algorithms and dynamic load balancing algorithms, according to the request distribution policy [6], [7]. The static load balancing algorithm pre-sets the load balancing policy based on the overall situation of the cluster servers, and distributes the task requests according to the established policy when the servers are running. The balancing policy was fixed during the running process. This method is more suitable for situations involving a single scene and fewer tasks. Unable to cope with the more complex request processing system. Typical static load balancing algorithms include the Weighted Round-Robin (WRR) algorithm [8] and the Destination Hashing Scheduling algorithm. The dynamic load balancing algorithm dynamically adjusts the distribution policy based on the real-time operation of the servers. This method is more suitable for complex cluster system and more tasks. Compared with the static load balancing algorithm, it is more effective, more flexible, and has better dynamic adaptability to the system. However, dynamically adjusting the distribution policy also introduces additional overhead. Typical dynamic load balancing algorithms include the Consistent Hashing algorithm [9] and the Least Connections (LCS) algorithm. These algorithms lack the consideration of performance differences between servers and the complexity of task requests; therefore, they cannot accurately judge the load situation of each server. For the load balancing of the server cluster system, the distribution strategy should be dynamically adjusted by considering several indicators such as the complexity of task requests, the load situation of each server, and the overall load of the system.
First of all, paper [10] analyzes and compares the existing main load balancing algorithms, and points out that load balancing algorithms can effectively improve the resource utilization, response rate and throughput of server clusters, proving that a reasonable load balancing algorithm is the key to improve the system service performance. Many researchers have proposed a series of load balancing algorithms to solve the problems of overload, load skewing, low efficiency and resource optimization in the system. The work in [11] aims the problem of excessive gateway traffic load in IoT by using multiple gateways to share the traffic, proposing a queuing theory-based analysis method to evaluate the performance of multi-gateway IoT systems, and using a load balancing strategy based on multi-criteria decision making in multiple gateways to achieve the relative consistency of global load. But they do not consider the complexity differences of different task requests. The authors in [12], aims the problem of application performance degradation due to the use of multiple GPUs in heterogeneous environments, proposed a dynamic load balancing algorithm for multi-objective decision making. Constructed two objective decision-making models of performance priority and performance energy balance, and dynamically exchanged the models during the execution of the algorithm, which can make more effective use of system resources. But the algorithm model lacks the differentiation research of energy consumption indicators and the impact of different service performance on the balancing effect. The work in [13] in order to solve the load imbalance problem caused by each node of the server in the cloud center under the high concurrency of task requests, proposed a load balancing strategy with dynamic weight adjustment to improve the Weighted Round-Robin algorithm. By considering the server hardware performance and real-time load situation, static weight and dynamic weight are designed and the final weight is calculated by combining the two. But the algorithm model does not consider the complexity difference of different tasks. The work in [14] aims the lack of dynamic scheduling algorithms in Kubernetes cluster and the incomplete optimization of cluster resources, proposed a multi-resource load balancing algorithm based on cooperative game theory. The dynamic scheduling of tasks is achieved by monitoring the actual resource usage of servers in real time. A cooperative game model is established for load balancing based on the indicators of CPU, memory, network bandwidth and disk IO of the clustered servers. But they do not consider the extra time cost caused by the cooperative game model. Paper [15] aims the inefficiency caused by the Round-Robin algorithm used by the Flink engine in heterogeneous clusters, proposed a smooth weighted round-robin task scheduling algorithm and a task scheduling algorithm based on ant colony algorithm to solve the cluster load imbalance problem during operation. But they only consider the performance difference of service nodes in the definition of weight, and do not consider the difference between different task requests.

Most of the above researches have been done from the perspective of server cluster performance optimization, without fully considering the complexity differences of task requests and the matching with different performance servers. This is also the research motivation of this paper.

The optimal matching of weighted bipartite graph means that each edge in the bipartite graph has a weight value, and finding a set of edges that maximizes the total weight value while ensuring that each vertex is connected to a unique edge. It is often used to solve problems such as optimal assignment of tasks [16], [17], resource allocation [19], [20], chemical interactions [21], [22], video information summary extraction [23], data matching [24] and collaborative sense optimization of composite events [25]. The work in [22] in order to solve the alignment problem of protein-protein interaction in biological systems, the initial alignment was created using a weighted bipartite graph matching technique to calculate the similarity scores and extended to obtain the final alignment, thus obtaining more similar regions. Paper [16] applied the weighted bipartite graph matching to the high-complexity flight service assignment problem and combined with the shortest-path algorithm to find an optimal solution. For the server cluster system, the maximum weight matching of weighted bipartite graph can be used to solve the problem of how to reasonably assign tasks to each server for processing and ensure the overall efficiency of the system. Paper [17] applied weighted bipartite graph matching to the digital twin technology for multi-energy systems to improve the performance of the computing nodes by allocating computing task blocks to each computing node evenly. The authors in [18] in order to solve the problem of scattered computing resources and limited resources of single computing node in the edge computing architecture, tasks are allocated to different edge data centers through the matching of weighted bipartite graphs and combined with particle swarm optimization algorithms to allocate tasks to specific docker containers for execution.

In this paper, optimal matching of weighted bipartite graphs is applied to load balancing of server cluster. Compared with the above researches, this paper uses the more intuitive real-time processing speed of the server as the load indicator. By calculating the matching degree between different tasks and different servers as the weight value, the load balancing between each server is realized from the perspective of task allocation.

### III. ALGORITHM SCHEDULING MODEL

#### A. PROBLEM STATEMENT

Define the graph $G = (J, S, E)$ is a bipartite graph consisting of the nonempty set $J$ and the nonempty set $S$ as vertices and the set $E$ as the set of edges. The set $J = \{J_1, J_2, J_3, \cdots, J_m\}$ represents the $m$ independent pending task requests and the set $S = \{S_1, S_2, S_3, \cdots, S_n\}$ represents the $n$ servers in the server cluster. The process of distributing a task $J_i$ to a server $S_j$ is defined as an edge $e_{ij} = (J_i, S_j)$ of the bipartite graph, the set of all edges is called the set of edges.
A bipartite graph is shown in Figure 1. The core of the proposed balancing algorithm based on optimal matching of weighted bipartite graph is relatively balanced.

Tasks (edges) known that each server can process one or several of these in processing different tasks (edges weight $w$ of length $m$). The management server has a set of task queues $S$ (set $S$), from which $n$ tasks are taken out. It is known that each server can process one or several of these tasks (edges $e_{ki}$), and each server has different efficiency for processing different tasks (edges weight $w_{ki}$). Find an allocation scheme to distribute the $n$ tasks to the $n$ servers, so that each server is assigned a task that is different from the other servers (edge set $H$). The efficiency of the server cluster as a whole is maximized and the load on the $n$ servers is relatively balanced.

### B. SCHEDULING MODEL

The scheduling model of the proposed dynamic load balancing algorithm based on optimal matching of weighted bipartite graph is shown in Figure 1. The core of the proposed algorithm is deployed on the task assignment management server. The management server processes each arriving task in the task pool in the form of a stream. After determining the assignment result for each task according to the proposed algorithm, the task is assigned to a server in the cluster for processing. The whole system consists of three parts: task request, task assignment management server and task execution server cluster. The task request part places the service requests of the task queue in the task pool waiting to be allocated for execution. The task assignment management server part consists of task volume estimation module, server cluster load indicator monitoring module, calculation module and task assignment module. The task volume estimation module is responsible for estimating the volume of the task to be assigned and the expected time to complete the task processing. The server cluster load indicator monitoring module is responsible for collecting the load of each server in the cluster in real time, and updating and storing it in the management server. The calculation module is the core part of the algorithm, which constructs a bipartite graph by comparing the expected completion time of the task with the load of each server. The edge weight matrix of the bipartite graph is generated by calculating the match between the task volume of each task and the load capacity of each server.

The task assignment module sends the task to the designated server for processing based on the optimal matching result obtained by the calculation module. The task execution server cluster part consists of each server, which is responsible for completing the execution of each task, calculating the load indicator of the server in real time and feeding back to the management server.

Server load indicator is related to the server’s own CPU utilization, memory capacity and occupancy, disk read/write rate and other parameters, external network bandwidth,
latency, and others will also have an impact [27]. In order to reduce the computational complexity, this paper uses a more intuitive task processing rate to represent the load of the server [28]. The load indicator for each server in the cluster is defined as
\[ v_i = \frac{r}{t_{\text{start}} - t_{\text{end}}} \]
where \( r \) is the volume of tasks currently executed by the server, \( t_{\text{start}} \) is the start time of the task execution, and \( t_{\text{end}} \) is the end time of the task execution. A larger \( v_i \) indicates a shorter task execution time and a higher service rate, which proves a better server performance and a lower server load. For the server cluster, the average value of the load indicator is defined as:
\[ \bar{v} = \frac{1}{n} \sum_{i=1}^{n} v_i \]
where \( n \) is the number of servers in the server cluster. Thus, the variance of the load indicator for each server in the server cluster can be calculated as:
\[ \sigma^2 = \frac{1}{n} \sum_{i=1}^{n} (v_i - \bar{v})^2 \]
where the variance \( \sigma^2 \) indicates the dispersion of the load indicator of each server in the server cluster. The smaller \( \sigma^2 \) proves that the load of each server in the cluster is more balanced.

IV. ALGORITHM PROCESS
In the proposed algorithm, we build a bipartite graph with tasks and servers as vertices, construct each edge of the bipartite graph according to the expected completion time of tasks and the load of each server, and calculate the matching degree of each task volume and each server to obtain the weight matrix of the edges. Finally, the Kuhn-Munkres algorithm is used to find the optimal matching of the weighted bipartite graph, and the tasks are assigned to the servers for processing according to the optimal matching result. The general flow of the proposed algorithm is shown in Figure 2.

The whole algorithm process is mainly completed by the management server and the server cluster together. Firstly, there are a series of task queues in the task pool with a certain amount of task requests waiting to be executed, which is represented by the set \( J \). There are \( n \) servers in the server cluster, which is represented by the set \( S = \{ S_1, S_2, S_3, \ldots, S_n \} \). The server records the task amount \( r \), the task execution start time \( t_{\text{start}} \) and the task execution end time \( t_{\text{end}} \) when the task is executed. The current load indicator \( v_i \) of the server is calculated by (2), and the load indicator \( v_i \) is fed back to the management server. The management server collects and updates the load indicators of each server for storage in real time, and the load indicators of each server are represented by the set \( V = \{ v_1, v_2, v_3, \ldots, v_n \} \). The initial load indicator for each server is calculated by sending a test task \( J_0 \) to each server from the management server. The task complexity of the test task \( J_0 \) is \( C_0 \), the task volume is \( R_0 \), and the time consumption is \( T_0 \), where the task complexity is determined by the actual number of cycles in the task execution.

After getting the initial load indicator of each server, the management server takes out \( n \) tasks from the task pool with the same number of servers, which is represented by the set \( J = \{ J_1, J_2, J_3, \ldots, J_n \} \). The task volume of each task is calculated by the task volume estimation module in the management server, which is represented by the set \( R = \{ R_1, R_2, R_3, \ldots, R_n \} \). The expected completion time of each task is represented by the set \( T = \{ T_1, T_2, T_3, \ldots, T_n \} \). For a task \( J_k \) in the task set \( J \), its task volume \( R_k \) and expected completion time \( T_k \) are calculated as:
\[ R_k = \frac{C_k}{C_0} R_0 \]
\[ T_k = \beta \frac{R_k}{R_0} T_0 \]
where \( R_0 \) is the volume of test task \( J_0 \), \( C_0 \) is the complexity of test task \( J_0 \), \( C_k \) is the complexity of test task \( J_k \), and \( T_0 \) is the actual running time of test task \( J_0 \). \( \beta \) is an empirical value for algorithm tuning. In practice, the processing time of the task does not expand proportionally with the increase of the task complexity. Therefore, we introduce the parameter \( \beta \) and increase the expected completion time \( T_k \) of the task by adjusting the value of the parameter \( \beta \), so that more matching results can be obtained in the subsequent steps.

After getting the expected completion time of the tasks to be assigned with the load of each server in the cluster, the management server starts to construct the bipartite graph. Firstly, the task set \( J \) and the server set \( S \) are respectively used as the vertices of the bipartite graph to construct an

![FIGURE 2. The flow chart of the proposed algorithm.](image-url)
edgeless bipartite graph, denoted as graph $G_1 = (J, S, E)$, where $E = \emptyset$, as shown in Figure 3.

For task $J_k$ and server $S_i$, the management server estimates the actual execution time $t_{ki}$ of the task at the current server based on the stored set of load indicators $V$. The actual execution time $t_{ki}$ is calculated as follows:

$$t_{ki} = \frac{R_k}{v_i}$$

where $R_k$ is the volume of the task $J_k$, and $v_i$ is the load indicator of the server $S_i$. If $t_{ki} \leq T_k$, it means that server $S_i$ is able to process task $J_k$. Connecting vertex $J_k$ and vertex $S_i$ in the graph means obtaining an edge $e_{ki} = (J_k, S_i)$ in the bipartite graph. If $t_{ki} > T_k$, it means that server $S_i$ is unable to complete the processing of task $J_k$ with the current load situation. The management server traverses all tasks and servers to obtain the edge set $E$. According to (1), the $e_{ki}$ in the edge set $E$ is quantized to two values of 0 and 1, and the edge matrix $E_{mn}$ is obtained. From this, an edged bipartite graph can be obtained, representing which servers each task can be executed on, denoted as graph $G_2 = (J, S, E)$, as shown in Figure 4.

By traversing $n$ tasks and $n$ servers when building the edge set, the node with the largest task volume is defined as $J_I$, and its task volume is $R_I$. The node with the largest server load indicator is defined as $S_1$, and its load indicator is $v_1$. The matching degree between node $J_I$ and $S_1$ is set as 1, which means that for edge $e_{1I} = (J_I, S_1)$, its weight $w_{1I} = 1$. The weight $w_{ki}$ of the remaining edges connected by each task node $J_k$ and each server node $S_i$ is defined as:

$$w_{ki} = \begin{cases} 
1, & k = I \& i = I \\
\frac{(R_k - R_I)(v_i - v_I)}{R_I v_I}, & k \neq I \& i \neq I \\
0, & \text{others}
\end{cases}$$

where $R_k$ is the volume of task $J_k$, $v_i$ is the load indicator of server $S_i$. Thus, the edge weight matrix $W_{nn}$ of the bipartite graph $G_2$ can be obtained as:

$$W_{nn} = \begin{bmatrix} 
w_{11} & w_{12} & \cdots & w_{1n} \\
w_{21} & w_{22} & \cdots & w_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
w_{n1} & w_{n2} & \cdots & w_{nn}
\end{bmatrix}$$

By combining the weight matrix $W_{nn}$ with the bipartite graph $G_2$, a weighted bipartite graph is obtained, which is still denoted as $G_2$. According to the theorem, for any weighted bipartite graph, it is always possible to add some vertices and edges with zero weight to transform it into a weighted complete bipartite graph, and there is always a perfect matching, which corresponds to the maximum weight matching of the original weighted bipartite graph. This is also the optimal matching result sought by the proposed algorithm. A complete bipartite graph means that for every vertex in the set $J$ is connected with an edge to every vertex in the set $S$. Therefore, for the later calculation, the weighted bipartite graph needs to be converted to the weighted complete bipartite graph. For the proposed algorithm, in the case where the number of task vertices is the same as the server vertices, some edges with weight of zero can be added to make it a weighted complete bipartite graph. In the case where the number of task vertices is less than the server vertices, some task vertices and edges with weight of zero are added to make it a weighted complete bipartite graph. A weight of zero means that the server node is unable to process the task node. In practical calculations, the weighted complete bipartite graph can be expressed by the Hadamard product of the edge weight matrix $W_{nn}$ and the edge matrix $E_{mn}$. Define the weighted complete bipartite graph as $G_3 = (J, S, E)$, as shown in Figure 5, where the dotted lines represent the edges with added weights of zero.

After the management server obtains the weighted complete bipartite graph $G_3$, it finds the optimal matching of the...
original weighted bipartite graph $G_2$ according to the Kuhn-Munkres algorithm of depth-first traversal, the specific steps are as follows:

**A. STEP 1: FIND THE EQUAL SUBGRAPH $K_L$ OF THE WEIGHTED COMPLETE BIPARTITE GRAPH $G_3$**

Define $l(J)$ as the set of labeling $N_k$ of each vertex in the task set $J$, and $l(S)$ as the set of labeling $Z_i$ of each vertex in the server set $S$. The vertex labeling is a mapping from the vertex set to the set of positive integers. And $w_{ki}$ is the weight of edge $e_{ki} = (J_k, S_i)$, by (8), we know that $0 \leq w_{ki} \leq 1$. Since the vertex labeling is a positive integer, all the elements in the weight edge matrix $W_{an}$ are scaled up by a thousand times equally after keeping three decimal places. If each edge $e_{ki}$ in the weighted complete bipartite graph $G_3$ can satisfy the following conditions:

$$l(J_k) + l(S_i) \geq w_{ki} \quad (10)$$

Then this set of vertex labeling is a set of feasible vertex labeling of the weighted complete bipartite graph $G_3$, denoted as $L$, where $N_k \in l(J), Z_i \in l(S) \cap l(J), l(S) \in L$. The feasible vertex labeling as shown in Figure 6.

Define the edge set $E_L$ as:

$$E_L = \{e_{ki} \in E(G_3) | l(J_k) + l(S_i) = w_{ki}\} \quad (11)$$

where $E(G_3)$ is the edge set of weighted complete bipartite graph $G_3$. After removing the edges that do not satisfy the above conditions, the spanning subgraph with $E_L$ as the edge set is the $L$ equal subgraph of $G_3$, denoted as graph $K_L$, as shown in Figure 7.

**B. STEP 2: FIND A PERFECT MATCHING $M$ OF THE EQUAL SUBGRAPH $K_L$**

The Hungarian algorithm is executed in the equal subgraph $K_L$. Take any one matching $Q = \{q_{ki} | k, i = 1, 2, 3, \cdots, n\}$ of $K_L$. The matching means the set consisting of some non-adjacent edges in the bipartite graph. In the task vertex set $J$, let the set of unsaturated vertices in the matching $Q$ be $A$. If $A = \emptyset$, the matching $Q$ is a perfect matching $M$ of the equal subgraph $K_L$, then the algorithm turns to the Step 4. The saturated vertex means that the vertex has and only has one edge connected to it. If $A \neq \emptyset$, then take any unsaturated vertex $A_{x_i}$ in the set $A$ and define $F = \{A_{x_i}\}$ and $O = \emptyset$. The set of adjacent vertices connected to the edges of $F$ is defined as $D(F), D(F) \in S$. Take an adjacent vertex $S_y$ of $A_{x_i}, S_y \in D(F) - O$, if $S_y$ is saturated vertex, set $q_{xy} = (A_{x_i}, S_y), q_{xy} \in Q$, update $F = F \cup \{A_{x_i}\}, O = O \cup \{S_y\}$, and find the other adjacent vertices of $A_{x_i}$. If $S_y$ is unsaturated vertex, obtain an augmenting path $p_{xy} = (A_{x_i}, S_y)$, let $Q = Q \oplus p_{xy}, A = A - A_{x_i}$, and continue to search for the augmenting path of the remaining unsaturated vertices in the set $A$ until $A = \emptyset$.

At this time, the matching $Q$ is a perfect matching $M$ of the equal subgraph $K_L$, then the algorithm turns to the Step 4. For the possible case of $D(F) = O$ in the algorithm, which means that there are unsaturated vertices and no edges connected to them, it is necessary to turn to the Step 3 to find another equal subgraph $K'_{L'}$ of the weighted complete bipartite graph $G_3$.

**C. STEP 3: FIND ANOTHER EQUAL SUBGRAPH $K'_{L'}$ OF THE WEIGHTED COMPLETE BIPARTITE GRAPH $G_3$**

Firstly, each vertex labelling $l'(u)$ in the weighted complete bipartite graph $G_3$ is modified according to the following equation and verified according to (10).

$$a_{l} = \min \{l(J_k) + l(S_i) - w_{ki} | J_k \in F, S_i \in S - O\} \quad (12)$$

$$l'(u) = \begin{cases} l(u) - a_l, & u \in F \\ l(u) + a_l, & u \in O \\ l(u), & \text{others} \end{cases} \quad (13)$$

From this, we can compute a new set of feasible vertex labelling $L'$. According to (11), a new edge set $E(G_3)$ is found to generate a new equal subgraph $K'_{L'}$. Then replace $L$ in the first step with $L'$, replace $K_L$ in the first step with $K'_{L'}$. Then turn to the step 2 to find the perfect matching $M$ of the equal subgraph $K_{L'}$.

**D. STEP 4: FIND THE OPTIMAL MATCHING $H$ OF THE WEIGHTED BIPARTITE GRAPH $G_2$**

By repeatedly performing the above steps, the perfect matching result $M$ of $K_L$ is solved, which is the optimal matching result of the weighted complete bipartite graph $G_3$. After deleting the edges and their vertices with weight of zero, the optimal matching result $H = \{h_{ki} | k, i = 1, 2, 3, \cdots, n\}$
TABLE 1. The time complexity of Algorithm 1.

| Lines       | Complexity Description | Description |
|-------------|-------------------------|-------------|
| Lines 6-10  | O(n^2)                  | Calculate the edge set and weight matrix |
| Lines 14-21 | O(n^2)                  | Find the augmenting path, total n times |
| Total       | O(n^2)+O(n × n^2)=O(n^3)+O(n)=O(n^3) | Augment the matching |

of the original weighted bipartite graph \( G_2 \) is obtained, as shown in the following figure with bold lines.

Finally, the management server sends the task to the specified server for processing based on the optimal matching \( H \), where each edge \( h_{ki} = (j_k, s_i) \) represents the assignment of task \( j_k \) to the server \( s_i \).

The management server repeats the above algorithm process and assigns all tasks in the task pool to each server for processing based on the optimal matching result. Under the condition of ensuring the effective processing of tasks, according to the edge weight matrix set by the proposed algorithm, by fully considering the matching degree between each task volume and each server load indicator, it can ensure that the tasks with higher task volume are assigned to the server with lower load for processing.

Therefore, it can effectively guarantee the priority matching and fast processing of key tasks. At the same time, it can effectively balance the load among servers while maximizing the overall efficiency of the server cluster. Thus, realizing the task scheduling and load balancing of the server cluster under high task concurrency.

The specific process of dynamic load balancing algorithm based on optimal matching of weighted bipartite graph is shown in Algorithm 1, and the time complexity of Algorithm 1 is shown in Table 1.

Through the analysis of the flow of the whole algorithm, the time complexity of the proposed algorithm is \( O(n^3) \). Among them, the edge set \( E \) and the weight matrix \( W_{nn} \) required to construct the weighted bipartite graph \( G_2 \) need to be traversed over \( n \) task nodes and \( n \) server nodes in the computation, so the time complexity is \( O(n^2) \). When computing the perfect matching \( M \) of the weighted complete bipartite graph \( G_3 \), since the perfect matching \( M \) has \( n \) edges, the number of times to find the augmenting path is \( n \). The time complexity of finding the augmenting path each time is \( O(n^2) \), so the time complexity of finding the augmenting path is \( O(n × n^2) = O(n^3) \). The time complexity of modifying the parameters is \( O(n^2) \). The time complexity for augmenting the matching is \( O(n) \). Therefore, the total time complexity of the proposed algorithm is \( O(n^3) \).

### V. SIMULATION

According to the model framework of the proposed algorithm, a test example of server cluster system is established for testing the performance of the proposed algorithm. The test system consists of a task pool, a management server and a server cluster composed of four servers. The program of management server and server cluster is written through the Java. There are task queues of a certain length in the test system.
task pool, and the complexity of each task obeys a uniform distribution in the interval (1, 10). The task pool will send the task stream to the management server in real time. The management server program specifically contains the task volume estimation module, the load indicator receiving module, the task assignment module and the core of the proposed algorithm. The server cluster program contains the task processing module, the load indicator calculation module and the load indicator feedback module. The configuration parameters of each server as shown in Table 2.

### TABLE 2. The configuration parameters of each server.

| Server   | Detailed Parameters                                      |
|----------|----------------------------------------------------------|
| Management Server | 4 Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz; 16 GB Memory |
| Server 1  | 2 Intel(R) Pentium(R) G645 CPU @ 2.90GHz; 4 GB Memory   |
| Server 2  | 4 Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz; 8 GB Memory |
| Server 3  | 4 Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz; 8 GB Memory |
| Server 4  | 8 Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz; 16 GB Memory |

### A. EXPERIMENTAL 1: TUNING THE ALGORITHM ACCORDING TO THE EMPIRICAL VALUE $\beta$

The proposed algorithm calculates the expected task completion time by comparing it with the test task complexity. In practice, the processing time of a task is not simply proportional to the task complexity. Therefore, the empirical value $\beta$ is set. The value of $\beta$ is related to the determination of the edge set $E$ in the bipartite graph, which represents whether the server $S_i$ can process the task $J_k$. By adjusting the value of $\beta$, more matching results can be obtained, and the optimal matching results can be found from them. It directly affects the assignment of tasks and the load balancing effect of the proposed algorithm. Therefore, based on the test example established in this paper, the proposed algorithm is tuned by varying the value of the empirical value $\beta$. During the experiment, the management server completes the assignment of tasks to servers by sending a task queue of 500, 1000 and 1500 tasks to the management server. The load indicator of each server in the server cluster is recorded every 5 seconds after running for 20 seconds, calculate the variance of the load indicator of the server cluster, and record the time consuming to complete all tasks at the same time. In this way, we compare the influence of different values of $\beta$ on the running time and load balancing effect of the proposed algorithm. For the accuracy of the experimental results, each experiment was repeated three times, and the average value of the three experiments was taken to represent the final experimental results. The results of experiment 1 are shown in Figure 9-11 and Table 3.

According to the results of experiment 1, when the system completes the assignment and processing of 500 tasks, the variance of the load indicator of $\beta = 1.5$ is lower than the case of other values in most of the time. When the system completes the assignment and processing of 1000 tasks, the four cases fluctuate substantially in the early stage, and in the middle and late stage the variance of the load indicator with $\beta = 1.5$ is mostly lower than that of the other values. When the system completes the assignment and processing of 1500 tasks, the variance of the load indicator with $\beta = 1.5$ is relatively lower than that of the other values in the first stage when the fluctuations of all four cases are large. In the middle and late stage, the variance of the load indicator with $\beta = 1.5$ is lower than that of the other values in most of the time, and the variance is relatively stable in the lower area. According to Table 3, in terms of the assignment and processing time to complete different number of tasks, the time consuming for $\beta = 1.5$ is lower than the other cases. In summary, when $\beta = 1.5$, the proposed algorithm can ensure relatively stable server cluster load and less time consuming to complete the task. Therefore, the value of $\beta$ is 1.5 in the experiments after this paper.

### B. EXPERIMENTAL 2: COMPARISON OF LOAD BALANCING EFFECTS OF DIFFERENT ALGORITHMS

In the test example of the server cluster system, the load balancing effect of the proposed algorithm is compared with...
that of other load balancing algorithms. During the experiment, the management server completes the assignment of tasks to servers by sending a task queue of 500, 1000 and 1500 tasks to the management server. The load indicator of each server in the server cluster is recorded every 5 seconds after running for 20 seconds, calculate the variance of the load indicator of the server cluster, and record the time consuming to complete all tasks at the same time. We select the Maximum Matching of Bipartite Graph algorithm (BGM), the Least Connections algorithm (LCS), and the Weighted Round-Robin algorithm (WRR) for comparison experiments with the proposed algorithm (WBGM). In order to evaluate the additional time complexity imposed on the system by introducing the proposed algorithm, we also set up a control check group without using any algorithm (CK). For the accuracy of the experimental results, each experiment was repeated three times, and the average value of the three experiments was taken to represent the final experimental results. The results of experiment 2 are shown in Figure 12-14 and Table 4.

According to the results of Experiment 2, when the system completes the assignment and processing of 500 tasks, the variance of the load indicator of proposed algorithm is lower than other algorithms most of the time. When completing the allocation and processing of 1000 and 1500 tasks, the fluctuations of all four algorithms in the early stage are large. In the middle and late stage, the proposed algorithm can quickly stabilize the variance of the load indicator in the lower area, and it is lower than the other algorithms most of the time. When the number of tasks gradually increases, the variance of the load indicator of the proposed algorithm gradually decreases. The variance of the load indicator of the CK group fluctuates greatly because no algorithm is used. According to Table 4, the proposed algorithm consumes less time in most cases. Compared with the BGM algorithm, we increase the consideration of matching the task volume with the server load, so the complexity of the proposed algorithm is slightly higher. However, the time consuming of the proposed algorithm is lower than it, which also proves that the matching degree of tasks and servers considered in the proposed algorithm can improve the server efficiency. The LCS algorithm consumes less time in some cases due to its lower algorithm complexity and is comparable to the proposed algorithm in terms of overall time consumption,
but its balancing effect is worse than the proposed algorithm. Since the WRR algorithm assigns tasks to the server for processing based on fixed weights, it lacks consideration of the real-time load on the server, so it consumes more time than the rest of the algorithms. Experiments with a control check group show that the introduction of the proposed algorithm does not add too much time complexity to the system. When the number of tasks increases, this algorithm can even offset the additional time complexity introduced and reduce the task processing time. In summary, compared with other algorithms, the proposed algorithm can obtain better load balancing effect on the premise of ensuring server processing efficiency, especially when the number of concurrent tasks gradually increases.

VI. CONCLUSION
In this paper, we propose a dynamic load balancing algorithm based on the optimal matching of weighted bipartite graph to solve problems in server cluster system. This is mainly due to the unreasonable assignment of tasks caused by not fully considering the performance differences among servers, which leads to problems of prolonged task completion processing time, decreased cluster load efficiency and load skewing. By calculating the matching degree of each task volume and each server load capacity, the weighted bipartite graph with task nodes and server nodes as vertices is constructed by fully considering the differentiation of task volume and server load capacity. The Kuhn-Munkres algorithm is used to find the optimal matching of the weighted bipartite graph, and the task assignment module achieves the optimal assignment of tasks to servers according to the optimal matching. The simulation results show that, compared with the BGM, LCS and the WRR algorithms, the proposed algorithm can improve the overall performance of the server cluster and better balance the load among the servers while ensuring the task processing efficiency. When the number of concurrent tasks is gradually increased, the proposed algorithm can achieve better load balancing effect. However, as the number of servers in the cluster increases, the efficiency of the proposed algorithm decreases; thus, improving the Kuhn-Munkres algorithm to reduce the time complexity is an important optimization direction for the proposed algorithm. For the overload problem generated by a server during task processing, the proposed algorithm dynamically adjusts the weights using real-time feedback load indicators, reduces the complexity of tasks assigned to that server or stops sending tasks to that server to solve the problem. This problem can also be solved using the load migration technology, which is our next research direction.

REFERENCES
[1] B. Li, J. Shang, M. Dong, and Y. He, “Research and application of server cluster load balancing technology,” in Proc. IEEE 4th Inf. Technol. Netw. Electron. Autom. Control Conf. (ITNEC), Jun. 2020, pp. 2622–2625, doi: 10.1109/ITNEC48623.2020.9850862.
[2] A. U. Rehman, Z. Ahmad, A. I. Jehangiri, M. A. Alaa Anzy, M. Othman, A. I. Umam, and J. Ahmed, “Dynamic energy efficiency resource allocation strategy for load balancing in fog environment,” IEEE Access, vol. 8, pp. 199829–199839, 2020.
[3] L. Rao, X. C. Tang, and Z. J. Hou, “Research on load balancing strategy for server cluster,” Comput. Modernization, vol. 209, no. 1, pp. 29–32, 2013.
[4] D. Hernandez, J. M. Cecilia, C. T. Calafate, J.-C. Cano, and P. Manzonii, “The Kuhn–Munkres algorithm for efficient vertical takeoff of UAV swarms,” in Proc. IEEE 93rd Veh. Technol. Conf. (VTC-Spring), Apr. 2021, pp. 1–5, doi: 10.1109/VTC2021-Spring51267.2021.9448873.
[5] F. Lin, J. L. Duan, C. W. Li, and J. C. Jiang, “Research on dynamic load balancing algorithm for C-V2X edge server,” Comput. Eng., vol. 46, no. 12, pp. 201–206, 2020.
[6] M. Junaid, A. Sohail, R. N. B. Rais, A. Ahmed, O. Khalid, I. A. Khan, S. S. Hussain, and N. Ejaz, “Modeling an optimized approach for load balancing in cloud,” IEEE Access, vol. 8, pp. 173208–173226, 2020.
[7] Z. Wen, G. Li, and G. Yang, “Research and realization of Nginx-based dynamic feedback load balancing algorithm,” in Proc. IEEE 3rd Adv. Inf. Technol. Electron. Autom. Control Conf. (IAEAC), Oct. 2018, pp. 2541–2546, doi: 10.1109/IAEAC.2018.8577911.
[8] W. Wang and G. Casale, “Evaluating weighted round Robin load balancing for cloud web services,” in Proc. 16th Int. Sym. Symbolic Numeric Algorithms Sci. Comput., Sep. 2014, pp. 393–400, doi: 10.1109/SYNASC.2014.59.
[9] J. Li, Y. Nie, and S. Zhou, “A dynamic load balancing algorithm based on consistent hash,” in Proc. 2nd IEEE Adv. Inf. Manag., Communicat., Electron. Autom. Control Conf. (IMCCE), May 2018, pp. 2387–2391, doi: 10.1109/IMCCE.2018.8469341.
[10] H. Rai, S. K. Ojha, and A. Nizarov, “Comparison study of load balancing algorithm,” in Proc. 2nd Int. Conf. Adv. Comput. Commun. Netw. (ICACCN), Dec. 2020, pp. 852–856, doi: 10.1109/ICACCN51052.2020.9362905.
[11] F. Banaie, M. Hossein Yaghmaei, S. A. Hosseini, and F. Tashtrian, “Load-balancing algorithm for multiple gateways in fog-based Internet of Things,” IEEE Internet Things J., vol. 7, no. 8, pp. 7043–7053, Aug. 2020.
[12] A. Cabrera, A. Acosta, F. Cabrera, and V. Blanco, “A dynamic multi-objective approach for load balancing in heterogeneous systems,” IEEE Trans. Parallel Distrib. Syst., vol. 31, no. 10, pp. 2421–2434, Oct. 2020, doi: 10.1109/TPDS.2020.2989869.
[13] C. Tan, X. Tan, L. Hu, and W. Xiong, “Dynamic weight load balancing method of cloud-center based on Nginx,” J. Chongqing Posts Telecommun. Univ. Natural Sci., vol. 33, no. 6, pp. 991–998, Dec. 2021.
[14] H. D. Li, X. Z. Zhang, L. X. Wang, H. Liu, P. C. Wang, and J. Z. Du, “Multi-node cooperative game load balancing strategy in the Kubernetes cluster,” J. Xian Univ., vol. 48, no. 6, pp. 16–22, Dec. 2021.
[15] Z. F. Wang, Y. H. Zhao, and G. R. Wang, “Research and implementation of load balancing algorithm in heterogeneous Flink cluster,” Chin. J. Electron., vol. 31, no. 4, pp. 3174–3187, Sep. 2022, doi: 10.1109/TNSM.2022.3167934.
[16] X. Tang, Y. Ding, J. Lei, H. Yang, and Y. Song, “Dynamic load balancing method based on optimal complete matching of weighted bipartite graph for simulation tasks in multi-energy system digital twin applications,” Energy Rep., vol. 8, pp. 1423–1431, Apr. 2022.
[17] S. S. Shao, S. D. Liu, K. Li, S. You, and H. J. Qiu, “LBA-ECA load balancing algorithm based on weighted bipartite graph for edge computing,” Chin. J. Electron., vol. 31, no. 4, pp. 1–12, Jul. 2022.
[18] W. Hou et al.: Dynamic Load Balancing Algorithm Based on Optimal Matching of Weighted Bipartite Graph

| Algorithm | Task-500(ms) | Task-1000(ms) | Task-1500(ms) |
|-----------|--------------|---------------|---------------|
| WBGM      | 55675        | 113003        | 167865        |
| BGM       | 56039        | 114066        | 167949        |
| LCS       | 56112        | 112881        | 168042        |
| WRR       | 59661        | 119408        | 174698        |
| CK        | 56082        | 112736        | 168157        |
[20] X. Zhou, L. Yang, and D. Yuan, “Bipartite matching based user grouping for grouped OFDM-IDMA,” *IEEE Trans. Wireless Commun.*, vol. 12, no. 10, pp. 5248–5257, Oct. 2013, doi: 10.1109/TWC.2013.090413.130097.

[21] P. Sarkar and K. Giri, “An intelligent technique to find bicliques and its application to optimum matching problem,” in *Proc. Int. Conf. Emerg. Trends Inf. Technol. Eng. (ic-ETITE)*, Feb. 2020, pp. 1–6, doi: 10.1109/ic-ETITE47903.2020.61.

[22] A. Mir, M. Naghibzadeh, and N. Saadati, “INDEX: Incremental depth extension approach for protein–protein interaction networks alignment,” *Biosystems*, vol. 162, pp. 24–34, Dec. 2017.

[23] S. Angadi and V. Naik, “Static video summarization—A minimum edge weight bipartite graph matching approach,” in *Proc. IEEE Int. Conf. Comput. Graph., Vis. Inf. Secur. (CGVIS)*, Nov. 2015, pp. 100–105, doi: 10.1109/CGVIS.2015.7449901.

[24] K. Lu, R. Ji, J. Tang, and Y. Gao, “Learning-based bipartite graph matching for view-based 3D model retrieval,” *IEEE Trans. Image Process.*, vol. 23, no. 10, pp. 4553–4563, Oct. 2014, doi: 10.1109/TIP.2014.2343460.

[25] J. Liu, F. Yuan, J. Wang, and X. Lu, “Efficient matching of multi-modal sensing nodes for collaborative sense optimization of composite events,” *IEEE Access*, vol. 7, pp. 47185–47196, 2019.

[26] R. Pavithra, S. Priyadharsini, P. and I. G. Hemanandhini, “Image matching using weighted graph matching algorithm,” in *Proc. 7th Int. Conf. Adv. Comput. Commun. Syst. (ICACCS)*, Mar. 2021, pp. 1–5, doi: 10.1109/ICACCS51430.2021.9442037.

[27] P. Zhang and M. Zhou, “Dynamic cloud task scheduling based on a two-stage strategy,” *IEEE Trans. Automat. Sci. Eng.*, vol. 15, no. 2, pp. 772–783, Apr. 2018.

[28] L. Zhou, L. M. Meng, L. P. Zhou, and W. Jiang, “Dynamic load balancing algorithm based on maximum matching of bipartite graph,” *Chin. High Technol. Lett.*, vol. 30, no. 8, pp. 798–804, Aug. 2020.

**WEI HOU** was born in Nanyang, Henan, China, in 1998. He received the B.S. degree from the Communication University of Zhejiang, Hangzhou, China, in 2020. He is currently pursuing the M.S. degree with the Information Engineering College, Zhejiang University of Technology, Hangzhou. His research interests include load balancing, the Internet of Things, and multimedia communication.

**LIMIN MENG** received the B.E., M.E., and Ph.D. degrees from Zhejiang University, Hangzhou, China, in 1984, 1998, and 2003, respectively. She is currently a Professor with the College of Information Engineering, Zhejiang University of Technology, Hangzhou, and the Director of the Zhejiang Provincial Key Laboratory of Communication Networks and Applications. Her main research interests include wireless communication and networks, streaming media transmission, and the IoT communications. She is a member of the China Computer Federation (CCF).

**XUQING KE** was born in Huangshan, Anhui, China, in 1999. He received the B.S. degree from the Zhejiang University of Technology, Hangzhou, China, in 2020, where he is currently pursuing the M.S. degree with the Information Engineering College. His research interests include the Internet of Things, multimedia communication, and embedded technology.

**LINGJUN ZHONG** was born in Hangzhou, Zhejiang, China, in 1995. He received the B.S. degree from the Huzhou College, Huzhou, China, in 2018. He is currently pursuing the M.S. degree with the Information Engineering College, Zhejiang University of Technology, Hangzhou. His research interests include multimedia communication, SLAM, and inertial navigation.

---

**WEI HOU** was born in Nanyang, Henan, China, in 1998. He received the B.S. degree from the Communication University of Zhejiang, Hangzhou, China, in 2020. He is currently pursuing the M.S. degree with the Information Engineering College, Zhejiang University of Technology, Hangzhou. His research interests include load balancing, the Internet of Things, and multimedia communication.

**LIMIN MENG** received the B.E., M.E., and Ph.D. degrees from Zhejiang University, Hangzhou, China, in 1984, 1998, and 2003, respectively. She is currently a Professor with the College of Information Engineering, Zhejiang University of Technology, Hangzhou, and the Director of the Zhejiang Provincial Key Laboratory of Communication Networks and Applications. Her main research interests include wireless communication and networks, streaming media transmission, and the IoT communications. She is a member of the China Computer Federation (CCF).

**XUQING KE** was born in Huangshan, Anhui, China, in 1999. He received the B.S. degree from the Zhejiang University of Technology, Hangzhou, China, in 2020, where he is currently pursuing the M.S. degree with the Information Engineering College. His research interests include the Internet of Things, multimedia communication, and embedded technology.

**LINGJUN ZHONG** was born in Hangzhou, Zhejiang, China, in 1995. He received the B.S. degree from the Huzhou College, Huzhou, China, in 2018. He is currently pursuing the M.S. degree with the Information Engineering College, Zhejiang University of Technology, Hangzhou. His research interests include multimedia communication, SLAM, and inertial navigation.