A Dichotomous Repair-Based Load-Balanced Task Allocation Strategy in Cloud-Edge Environment

Zekun Hu, Pengwei Wang(✉), Peihai Zhao, and Zhaohui Zhang

School of Computer Science and Technology, Donghua University, Shanghai, China
{wangpengwei,peihaizhao,zhzhang}@dhu.edu.cn

Abstract. Load balancing is a hot issue in the current cloud-edge environment. However, due to the characteristics of edge computing, load balancing needs to be better integrated with edge devices and edge networks to provide higher performance and reliability. The presence of a large number of overloaded nodes may lead to load imbalance and thus affect the efficiency of nodes. To solve this problem, the key is how to allocate tasks to the appropriate resources. To this end, this work proposes a dichotomous task allocation policy Dichotomous Repair (DREP) to achieve efficient task allocation and overall load balancing of edge nodes in cloud-edge environment. The proposed policy consists of five steps: grouping, adjustment, filtering, greed and repair. The dichotomous policy is adopted to generate the initial allocation scheme according to the number of edge nodes, and then the overloaded and underloaded nodes are repaired by the subsequent two-stage repair policy to maintain the load balance. Finally, through extensive experiments, we evaluate the proposed method and the results show that it outperforms other algorithms in terms of workload balancing.

Keywords: Load balancing · Task allocation · Dichotomous · Edge computing · Cloud-edge

1 Introduction

Cloud-edge computing is a hot research direction at this stage [1], which is a new architectural model that combines the advantages of cloud computing and edge computing. The cloud center can make up for the lack of computing resources at the edge, while the edge nodes can provide computation and services with lower latency [2]. However, the overall performance may be affected due to improper task allocation in edge computing. Specifically, if the resources requested by a task exceed the remaining resources of a node, the task cannot be assigned to that node, thus affecting the efficiency of subsequent allocations. Neglecting the capacity and workload of nodes can also lead to poor node performance, slow task processing, and unbalanced data center load, and the advantages of cloud-edge computing cannot be realized.

H. Gao et al. (Eds.): CollaborateCom 2023, LNICST 561, pp. 128–148, 2024.
https://doi.org/10.1007/978-3-031-54521-4_8

In order to achieve load balancing in data centers [3], it is crucial to efficiently assign tasks to the appropriate nodes. Therefore, an efficient task allocation strategy is needed to assign user-submitted task requests to the most appropriate nodes to improve node efficiency and reduce user waiting time while maintaining load balancing. However, there is still room for improvement in the current research on load balancing task allocation strategies, mainly focusing on the following aspects. First, the existing scheduling strategies can be further improved in terms of flexibility, so that they can be better adjusted dynamically according to the real-time load situation and resource status. Second, current research focuses mainly on the load situation of nodes, and pays relatively little attention to the unique characteristics and demands of tasks themselves. Such methods may bring some imbalance in resource allocation, thus failing to achieve better load balancing.

In order to better improve the shortcomings of previous work and solve the problem of unbalanced workload of edge nodes, this paper proposes a new task scheduling algorithm DREP, which employs a bisection-based idea to flexibly deal with task allocation and improve the adaptability of the system. By successively bisecting the current task set, a better initial task allocation scheme is obtained, which provides a good basis for subsequent load balancing, and self-healing is performed in combination with the node's capacity and load, which fully utilizes the node's resources and improves the overall efficiency. Specifically the main contributions of this paper are as follows:

- A load balancing algorithm is proposed, which fully considers the processing capacity of nodes as well as the load sizes of different tasks, and achieves a highly consistent load balancing effect by dynamically adjusting the task allocation of each edge node. It is worth emphasizing that this algorithm not only performs well in homogeneous environments, but is also applicable to heterogeneous environments.
- With the consideration of task size and the number of edge nodes, this paper proposes a grouping strategy based on the dichotomous idea to dichotomize the task set appropriately according to the number of edge nodes, thus forming a good initial allocation scheme.
- We propose a two-stage repair strategy to adjust task allocation. By moving suitable tasks from the overloaded nodes to the underloaded nodes, the strategy achieves a highly consistent edge node load rate and effectively improves the overall performance.

The rest of this paper is organized as follows. Section 2 briefly introduces the current work on task allocation methods and load balancing in edge computing and cloud computing environments. Section 3 illustrates the imperfections of current work through an example. Section 4 describes the relevant issues and formalizes them. Section 5 describes in detail the design and implementation process of algorithm. Section 6 presents the experimental results, which demonstrate the effectiveness of proposed algorithm. We conclude the paper in Sect. 7.

2 Related Work

This section reviews two aspects related to our research, including task allocation and load balancing in cloud and edge computing.

2.1 Task Allocation

Task allocation refers to the assignment of user-submitted task requests to heterogeneous available resources. Appropriate task allocation not only improves resource utilization, but also minimizes makespan by executing the assigned tasks in a shorter period of time. Therefore, more research is needed in area of cloud task scheduling or task allocation in order to efficiently map tasks to available resources and to improve quality of service (QoS) parameters.

In order to focus on the energy consumption problem caused by task allocation in cloud computing, Gai et al. [4] proposed a management model in a heterogeneous cloud environment, which can effectively reduce the energy consumption in mobile cloud systems. Zhou et al. [5] proposed an improved ITSA algorithm to enhance the resource utilization and QoS in cloud environment by binding the size tasks to form task pairs through the gain value of task exchange and then scheduling with greedy policies. However, their approach is limited to scenarios where the task sizes do not differ much. Panda et al. [6] introduced a multi-cloud environment to break the limited resources of a single cloud during peak demand, and proposed an improved Min-Min algorithm and Max-Min algorithm, which is divided into three phases: matching, allocation and scheduling. Ali et al. [7] divided the tasks into five categories, each with similar properties, and based on the execution time of tasks, the ones with shorter execution time are scheduled first. Scheduling is divided into two steps, first determining the category of a task and then identifying a task in that category. Er-raji and BenaNaoufal [8] proposed a multi-parameter-based global task scheduling strategy for the priority task scheduling problem in distributed data centers in cloud computing. Zhang and Zhou [9] proposed a two-stage scheduling method based on a two-stage scheduling approach in order to improve the scheduling performance and efficiency of cloud. In the first stage, a Bayesian classifier is used to classify tasks with historical scheduling data to create a certain number of virtual machines to save time, and in the second stage, a dynamic scheduling algorithm is used to match with specific virtual machines.

With the development of technology, edge computing is integrated into people's life as a new technology. In order to get low latency and efficient services, users can assign tasks to nearby edge nodes.

Therefore how to assign task scheduling to edge nodes becomes an important issue. Su et al. [10] proposed an improved NSGA-II algorithm to solve the task scheduling of edge computing, using a dynamic adaptive strategy to adjust the crossover rate and mutation rate to improve the population search efficiency, which improves the search efficiency while maintaining the population diversity and finally obtains a set of optimal solution sets. Shen [11] proposed a hierarchical task scheduling strategy for the latency cost problem, with an edge node

layer for processing simple tasks and dividing them into three priority levels and a frog server layer for processing complex tasks and returning accurate computation results. To make full use of the limited computational resources, Yang and Poellabauer [12] prioritized tasks based on attributes such as task CPU and combine reinforcement learning for task scheduling, but the proposed scheme is based on too many good assumptions and constrained to a practical system.

2.2 Load Balancing

The goal of load balancing is to ensure that the state of each host in cloud computing or edge computing is balanced to achieve high availability of computing platform as a whole. While making full use of computing resources, it ensures that each host can achieve optimal business performance and work efficiency.

Load balancing in the cloud refers to the distribution of load to the active components associated with the processing tasks. Ruan et al. [13] proposed a strategy to calculate the optimal work utilization of hosts, dividing hosts into different blocks according to different PPRs, and keeping each host in the optimal block through VM allocation and migration framework to achieve the best balance between host utilization and energy consumption, However, because of the energy consumption constraint, there is still a large variation in load between hosts. In [14–17], the problem of consolidation of virtual machines is investigated for consolidating overloaded or underloaded hosts through virtual machine placement and migration strategies so as to keep the workload of hosts in an appropriate range. Zhou et al. [18] solved the problem of reducing high energy consumption in cloud data centers with minimal service level agreement (SLA) violations by proposing two new adaptive energy-aware algorithms for overloaded and underloaded vm's and considering their application types. In another paper [5], Zhou proposed a task-based load balancing problem for cloud environments by swapping the gain values and binding the one with the smallest gain value and the one with the largest gain value to form a task pair for the purpose of load balancing, but it is not applicable to scenarios with large differences in task loads.

Although edge computing makes up for the shortcomings of cloud computing, however, edge nodes are usually heterogeneous and have relatively weak computing power. How to fully utilize the resources of edge nodes, reasonably allocate tasks, and achieve load balancing in edge computing is also a problem. Dong et al. [19] proposed a deployment policy HEELS based on the analysis of heuristic task clustering method and firefly swarm optimization algorithm, so as to solve the long-term load balancing problem of edge computing federated cloud as a whole. Li et al. [20] combined greedy algorithm with genetic algorithm to minimize the number of edge servers while ensuring load balancing among edge servers and QoS requirements of users. Li et al. [21] classified edge nodes into three categories based on their attributes: light, normal, and heavy, and intermediated nodes assign new tasks to light nodes to achieve load balancing. Dong et al. [22] studied the load balancing problem of federated cloud-edge data centers and proposed a task deployment strategy JCETD based on pruning algorithm

and deep reinforcement learning in order to achieve efficient deployment and overall load balancing of cloud-edge tasks.

Task scheduling has been considered for load balancing, both in cloud and edge environments. However, due to the complexity and diversity of scheduling, load balancing algorithms may not be flexible enough to effectively deal with complex task scheduling problems or even resource scarcity. In addition, most of the current research focuses on making servers host the same number of tasks or minimizing the number of servers to achieve load balancing without considering task size, node capacity, and resource availability, which leads to significant differences in server load and cannot achieve an approximate consistency.

3 Motivation Example

In this section we use an example to compare the differences between the algorithms. The scenario is to process several input tasks ti which are assumed to have no sequential relationship and are not dependent on each other, and task allocation is operated by the proxy server, including the calculation of total number of resources for the tasks.

We use Table 1 to map the resource requirements for each task. Here we consider CPU resources and memory resources, and both are weighted to get the total requested resources. For example, task 1 requires 100 CPU resources and 92 memory resources, for a total of 96 resources.

Table 1. Resource requirements for each task in motivation example.

Task	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
CPU	100	10	20	30	42	90	1	6	55	76
Memory	92	26	46	32	88	106	1	36	113	108
Total requested resources	96	18	33	31	65	98	1	21	84	92

Table 2. Computing resources for edge nodes.

Edge Nodes	e_1	e_2	e_3
CPU	120	200	250
Memory	140	200	350
Total Resources	130	200	300

The tasks need to be processed at the edge nodes, and there are three edge nodes. Different edge nodes have different computational resource capabilities. We use Table 2 to represent the computational resources of each edge node, and the resource requirements for each task on each node should not exceed the amount of resources that can be provided by that node. For edge node e_1, it has a total CPU resource of 120 units and a total memory resource of 140 units.

In cloud and edge computing environments, greed is a simple and effective approach. In this example, we obtained a task allocation scheme by greedy policy as $\{t_3, t_7, t_8\}$, $\{t_2, t_4, t_5\}$, $\{t_1, t_9\}$ with the load factor of edge nodes as 42.31%, 57%, and 60% with a standard deviation of 0.077. In addition, we also tried an improved ITSA [5] algorithm by combining task binding to form task pairs for allocation scheduling, resulting in allocation schemes of $\{t_6, t_7\}$, $\{t_1, t_2\}$, $\{t_4, t_8, t_9, t_{10}\}$ with load ratios of 76.15%, 57% and 76% for edge nodes, respectively, with a standard deviation of 0.0899.

The existing task allocation schemes do not consider the capacity and load of the nodes, resulting in unsatisfactory load balancing. In contrast, a better allocation scheme $\{t_6\}$, $\{t_1, t_3\}$, $\{t_2, t_4, t_5, t_7, t_8, t_9\}$ can result in edge nodes with load ratios of 70%, 64.5%, and 73.3% with a standard deviation of only 0.036. A small value of standard deviation indicates a more balanced load distribution across edge nodes. This means that under this distribution scheme, the difference in the load carried by each edge node is very small and the load is more evenly distributed among the nodes. Therefore, the purpose of this paper is to propose an algorithm to generate a better task allocation scheme that results in better load balancing. The details of algorithm will be given in the subsequent sections.

4 Concept and Proposed Model

We will define the system model and some key definitions in this section.

4.1 Models and Definitions

In cloud-edge environment, the task allocation problem can be briefly described as follows: the agent server collects n independent computational tasks and allocates them to a joint data center composed of m edge nodes and 1 cloud, considering both heterogeneous and homogeneous environments of nodes, and solves the load balancing problem of edge nodes through high-quality allocation schemes and policies.

Definition 1. In cloud-edge environment, the agent server collects n task requests which form a set $T = \{t_1, t_2, \ldots, t_i \ldots, t_n\}$, where t_i denotes the *i-th* task. Assume that the n tasks are independent of each other and have no priority or time restrictions, and each task is represented by $t_i(R_i^{cpu}, R_i^{mem})$, where R_i^{cpu} denotes the CPU resources required by the task and R_i^{mem} denotes the memory resources required by the task.

Definition 2. In cloud-edge environment, assume that there are m edge nodes. Define the edge node set $E = \{e_1, e_2, \ldots, e_j \ldots, e_m\}$, where e_j denotes the *j-th* edge node. Assume that the capacity of edge node is denoted by $e_j(S_j^{cpu}, S_j^{mem})$, where S_j^{cpu} denotes the CPU capacity of edge node j and S_j^{mem} denotes the memory capacity of edge node *j*.

Definition 3. In order to distribute the task sets collected by the agent server to the federated data center at cloud-edge, it is necessary to ensure that the resources available at the edge nodes can accommodate the current task sets. The total resources of edge node are expressed as:

$$Q_j = \alpha S_j^{cpu} + \beta S_j^{mem} \tag{1}$$

$$\alpha + \beta = 1 \tag{2}$$

The amount of used resources of edge node j is expressed as follows:

$$L_j = \sum_{i=1}^{k} R_i^j \tag{3}$$

$$R_i^j = \alpha R_i^{cpu} + \beta R_i^{mem} \tag{4}$$

where R_i^j denotes the total requested resources for the *i-th* task on the *j-th* edge node, k denotes the number of tasks assigned to that edge node. We refer to [22] for evaluating CPU and memory, α is the weight of CPU, and β is the weight of memory. In this paper we consider that the task's CPU and memory requirements are considered equally important, and thus α and β are equal.

Definition 4. Load balancing refers to the balanced degree of load distribution on each node, and this paper uses the standard deviation of resource load ratio of edge nodes to express the degree of load balancing, where the resource load ratio of edge node j is expressed as:

$$G_j = \frac{L_j}{Q_j} \tag{5}$$

The resource load ratio of each edge node can be calculated by (1) (2)(3) (4) (5), and then the load balancing degree of edge data center can be calculated by using the standard deviation formula.

$$W = \sqrt{\frac{1}{m}\sum_{j=1}^{m}(G_j - \frac{1}{m}\sum_{j=1}^{m} G_j)^2} \tag{6}$$

The problem of this paper can be formalized:

$$\begin{aligned} & min \sqrt{\frac{1}{m}\sum_{j=1}^{m}(G_j - \frac{1}{m}\sum_{j=1}^{m} G_j)^2} \\ s.t. C1 \quad & \sum_{i=1}^{k} R_i^{cpu} \le S_j^{cpu} \quad \forall j \in m \\ C2 \quad & \sum_{i=1}^{k} R_i^{mem} \le S_j^{mem} \quad \forall j \in m \end{aligned} \tag{7}$$

In summary, the ultimate goal of this paper is to achieve load balancing in the edge data center under the condition that the resource constraints of each edge node are satisfied. That is, to make W as small as possible.

4.2 Problem Statement

In cloud-edge environments, task requests need to be assigned to different edge nodes and cloud center for processing. However, when the amount of resources required for a task request exceeds the remaining resources of a certain node, it will lead to an overload problem and reduce the computational and operational capacity of that data center, thus failing to achieve the desired load balancing. Therefore, when facing large-scale task requests and limited computing resources in edge computing centers, different task allocation schemes and resource allocation strategies will produce different computing efficiency and load balancing states. Optimizing task allocation schemes is an important means to achieve high-quality computing services and load balancing. Therefore, this paper designs a high-quality task allocation scheme to solve the problem of load imbalance of edge nodes.

5 Load Balancing DREP Algorithm

5.1 System Architecture Design

In our model, the process of task assignment is mainly grouping, adjustment, filtering, greed and repair in five parts, and Fig. 1 shows the operation flow of our proposed model. Different from other algorithms, our algorithm can take into account both homogeneous and heterogeneous environments.

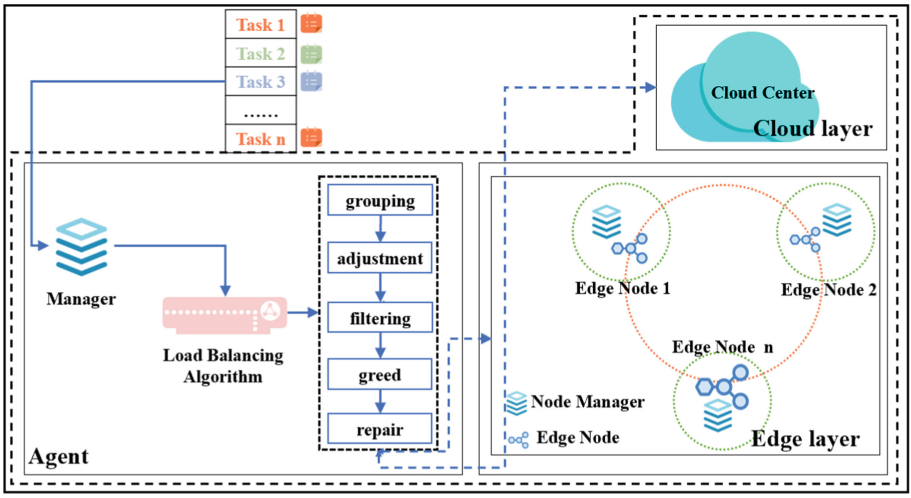

Fig. 1. The framework of DREP.

5.2 Algorithm Implementation

Overall Process. Figure 2 depicts a flow of our algorithm as a whole, covering five key steps. First, we group the tasks based on the number of edge nodes, and then reconcile the groups with each other to avoid getting trapped in a local optimal solution. Then, we filter out some sets of tasks that do not meet the constraints and assign them to suitable edge nodes in a greedy manner. Finally, the capacity and load of edge nodes are combined to repair so that the load ratio of each node is close to the mean value. The purpose of the grouping is to get a good initial distribution scheme. An adjustment mechanism is introduced to avoid falling into a local optimum among the groups. Then for the tasks that cannot be carried in the nodes need to be eliminated and greedily assigned to other nodes that can carry them. Finally, in order to achieve load balancing, the load rate of each edge node is approximated to be the same using a repair strategy.

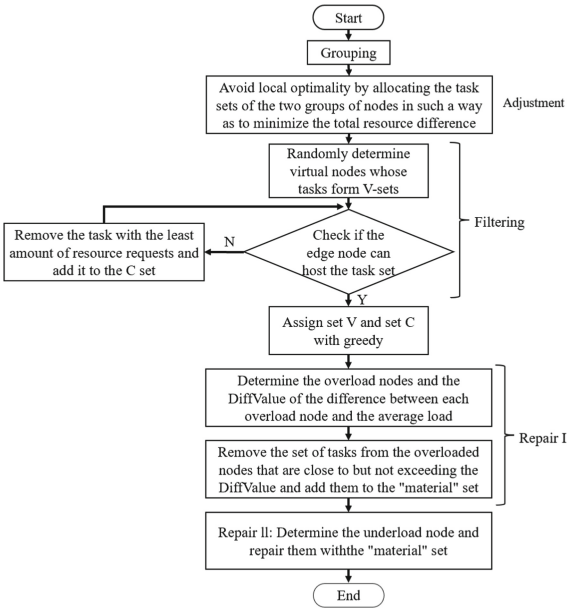

Fig. 2. Flow chart of the proposed method.

Grouping. As mentioned earlier, the first step of our proposed DREP algorithm is grouping, which is used to generate a good initial allocation scheme. The algorithm performs grouping through the dichotomous idea, and the corresponding pseudo-code representation is given in Algorithm 1. The inputs include the set of tasks, initial conditions, and termination conditions. The goal of grouping is to divide the set of tasks into two sub-task sets with the smallest difference in the total amount of requested resources, similar to the improved 01 backpack problem, in which we analogize tasks to items, but we care only about the weight of the items, not their value, so the core of the problem is how to choose the

items so that the weight of the backpack is closest to half of the total item weight. This is a classical dynamic programming problem. The core mechanism of the algorithm is that for each task, if the remaining capacity of the backpack is less than the weight of the current item, it can only be chosen not to put it in. Conversely, it can be considered to put in, choosing the larger of these cases according to the properties of dynamic programming. Where *dp[i][j]* means the maximum gain that can be obtained for the first i items with capacity j.

Algorithm 1: Group

Input: $Task, EndNum, Num$
Output: $Taskassignment$
if $Task.size == 1$ *or* $EndNum == Num$ **then**
  $Result$.append($Task$)
  return
end
for $i = 0$ *to* $Task.size$ **do**
  $Sum+ = Task[i]_{Load}$
end
Initialize the parameter dp
for $i = 1$ *to* $Task.size + 1$ **do**
  for $j = 1$ *to* $Sum/2 + 1$ **do**
    if $j >= Task[i-1]_{Load}$ **then**
      $dp[i][j] = max(dp[i-1][j], dp[i-1][j - Task[i-1]_{Load}] + Task[i-1]_{Load})$
    else
      $dp[i][j] = dp[i-1][j]$
    end
  end
end
The set of $Task$ can be divided into $Task1$ and $Task2$ according to the dp
$Num = Num + 1$
Group $(Task1, Num, StartNum)$
Group $(Task2, Num, StartNum)$

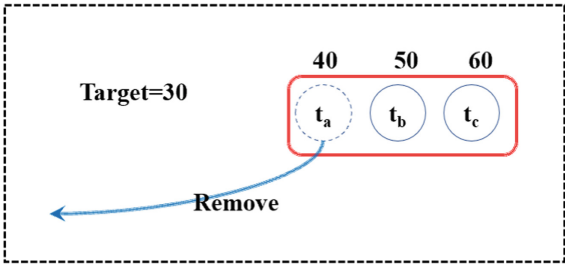

Fig. 3. Example for remove.

Repair. In order to keep the load ratio of each edge node close to the mean, we propose a two-stage repair algorithm. The first phase quickly reduces the load to near-average by eliminating some tasks from the overloaded nodes; the

eliminated task set and other tasks that do not get assigned become the second phase "materials" that is then assigned to the underloaded nodes. The repair idea is to eliminate or select the most suitable task set from a set in order to bring the node to the desired load ratio. The most suitable task set is the one for which the total amount of requested resources should converge to, but not exceed the target value. Here, the problem is likewise analogous to the 01 backpack problem, with the difference that the weight of the backpack no longer converges to half of the total item weight, but to the target value TargetValue. We give the pseudo-code representation in Algorithm 2. The input of the algorithm consists of task set Task of the overloaded node and the target value TargetValue, and the output is the "materials" task set RepairTaskSet core. The difference with Algorithm 1 is that there may be no tasks on the overloaded node that converge to and do not exceed the target value, as shown in the Fig. 3, In this case, the task that requests the least amount of resources is eliminated by default to ensure that the load on the overloaded node can be reduced.

Suppose there are n edge nodes and m tasks. First determine the number of bisections, our approach is to find a power of 2 closest to the number of nodes n, expressed as $n + x = 2^a$, where x serves to keep n near the smallest power of 2, so the number of bisections is a, so $a = \log(n+x)$. In each bisection operation, a dynamic planning operation is performed to minimize the difference between the resources of the two groups of tasks divided, with the outer layer traversing the tasks sequentially, and the inner loop determining inside by constantly changing the capacity (from 1 to $s/2$) whether the minimization of the difference can be produced, which is performed a total of $m * \frac{s}{2}$ times, thus the total time complexity is $O(\log(n+x)ms)$. Similarly, we can calculate the time complexity of the other steps, and overall the time complexity of DREP is $O(n^2)$.

Algorithm 2: Pick_Tasks

Input: $Task, TargetValue$
Output: $RepairTaskSet$
for $i = 1$ *to* $Task.size + 1$ **do**
  for $j = 1$ *to* $TargetValue + 1$ **do**
    if $j >= Task[i]_{Load}$ **then**
      $dp[i][j] = max(dp[i-1][j], dp[i-1][j - Task[i]_{Load}] + Task[i]_{Load})$
    else
      $dp[i][j] = dp[i-1][j]$
    end
  end
end
$SetT$ stores the set of tasks that match the $TargetValue$
if $SetT.size$!= 0 **then**
  $RepairTaskSet = RepairTaskSet + SetT$
else
  Find the task T with the lowest load value
  $RepairTaskSet.add(T)$
end
return $RepairTaskSet$

5.3 Running Example

In this section we will use a concrete example to illustrate our algorithmic ideas.

First, we group the task set according to the number of edge nodes. To determine the number of dichotomies, we use the method of finding the smallest power of 2 that can contain the number of edge nodes and perform x dichotomies. For example, our example has three edge nodes, so x equals 2. That is, the task set is divided into four groups, each group corresponding to one edge node assignment scheme, as shown in Fig. 4, as follows: $\{t_2,t_4,t_{10}\}$,$\{t_1,t_3\}$,$\{t_5,t_9\}$,$\{t_6,t_7,t_8\}$. The extra set we assume is assigned to virtual nodes, which will be processed in the later filtering steps.

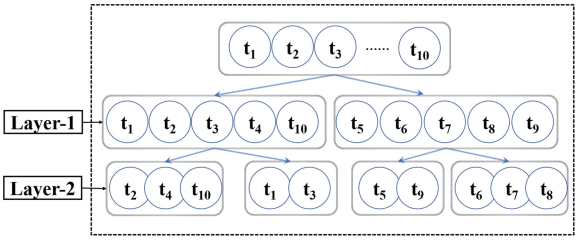

Fig. 4. Step 1: grouping.

In the second step, we take the adjustment step to avoid local optima in the process of generating sub-task sets. Specifically, a set of tasks in a frontier node is combined with a set of tasks in a trailing edge node, and the set of tasks within the two groups is redistributed to obtain an allocation that minimizes the difference in total resource requirements, as shown in the Fig. 5, In this example, the reconciled results change from $\{t_2,t_4,t_{10}\}$,$\{t_1,t_3\}$,$\{t_5,t_9\}$,$\{t_6,t_7,t_8\}$ to $\{t_4,t_6,t_7\}$,$\{t_1,t_3\}$,$\{t_5,t_9\}$,$\{t_2,t_8,t_{10}\}$.

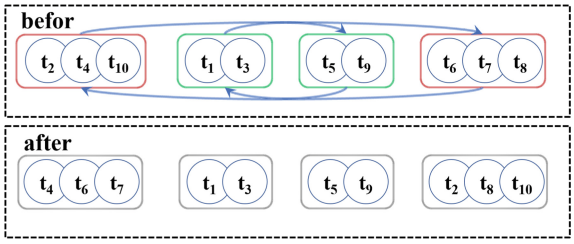

Fig. 5. Step 2: adjustment.

Next, we need to filter out the set of tasks that cannot satisfy the resource constraints and solve the set of tasks on the virtual nodes. For the virtual nodes,

we directly select them using a random strategy, assuming that edge node 4 is a virtual node. Then, we check each edge node in turn for the presence of insufficient resources, where the amount of resources that the node can provide is less than the amount of resources required by the task. If it exists, we sort the set of tasks on the edge nodes in ascending order of the total requested resources and then eliminate the tasks one by one until all resource constraints are satisfied. As shown in Fig. 6. In this example, only edge node 1 cannot satisfy the constraint, and the set of tasks belonging to it is sorted into t_7, t_4,t_6 in ascending order, and t_7 is eliminated to satisfy the constraint. After filtering, we get the task set as $\{t_4,t_6\}$,$\{t_1,t_3\}$,$\{t_5,t_9\}$, The remaining $\{t_2,t_7,t_8,t_{10}\}$ are considered as the set of tasks to be assigned.

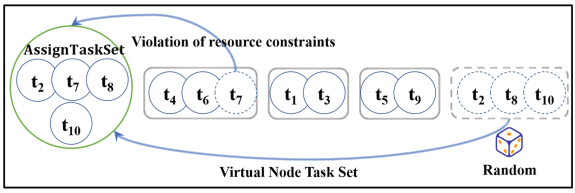

Fig. 6. Step 3: filtering.

Then what we expect is to achieve a load-rate balance, not to carry the same amount of resources, so edge nodes with more capacity should carry more tasks. Therefore, we use a greedy strategy to sequentially assign the set of tasks generated by filtering to the edge node with the lowest current load factor subject to the resource constraints. In this example, the current load rates of edge nodes are 99.23%, 64.5%, and 49.7%, so the first element t_2 in the set of tasks generated by filtering is added to the edge node 3 with the lowest current load rate, and then the load rate is updated and the process is cycled. The results after greedy are $\{t_4,t_6\}$,$\{t_1,t_3\}$,$\{t_2,t_5,t_7,t_8,t_9\}$. As shown in Fig. 7.

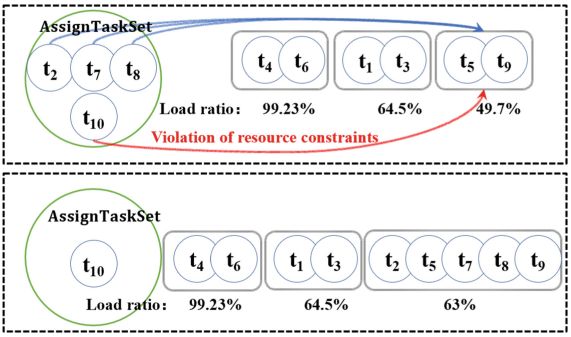

Fig. 7. Step 4: greed.

Finally, we combined the capacity of edge nodes and the current task allocation to achieve load balancing through a two-phase repair strategy. As shown in Fig. 8. In the first stage, we fix overloaded nodes (nodes above the mean value) and remove part of task set based on the difference between the load ratio of the nodes and the mean value, so that their total requested resources converge to but are not higher than the difference. If such a combination does not exist, the task with the smallest total requested resources is removed by default. In this example, only the load ratio of edge node 1 exceeds the mean value, and the task with the smallest total requested resources t_4 is removed by default. In the second stage, we repair the nodes below the mean value, and the "materials" for repair comes from the set of tasks removed in the first stage and the remaining set of tasks to be allocated in the greedy step ($\{t_4,t_{10}\}$). The difference in resources is calculated based on the mean value, and some tasks from the set of "materials" that are close to but not higher than the target resources are added to the node. For the second edge node, the node needs only 22.16 resources to reach the mean value, but there are no tasks in the task set that satisfy the condition, so the node gives up the repair. The third edge node requires 37.73 resources to reach around the mean value, and we find a task t_4 in the task set that satisfies the condition and assign it to this node. With such a repair strategy, we successfully keep the load ratio of each edge node at or near the mean value and achieve the purpose of load balancing.

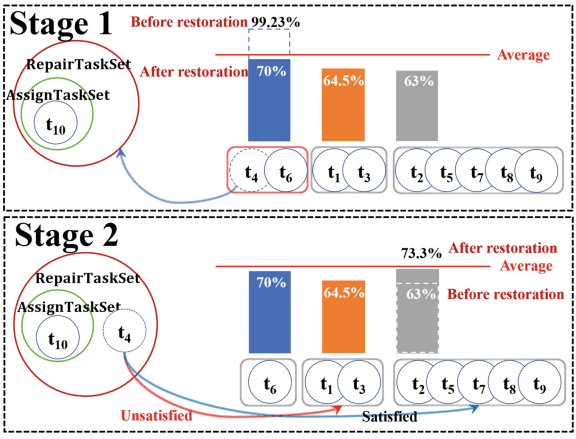

Fig. 8. Step 5: repair.

With our optimization scheme, the final set of tasks $\{t_6\}$, $\{t_1,t_3\}$, $\{t_2,t_4,t_5,t_7,t_8,t_9\}$ are assigned to edge nodes and taskt_{10} is assigned to the cloud respectively. The reduction in load balancing degree is 56.6% compared to random policy, 53.2% compared to greedy policy, and 59.96% compared to ITSA algorithm. Therefore, our scheme has smaller standard deviation and more balanced load.

6 Performance Evaluation and Analysis

In this section, we mainly evaluate the performance of the DREP algorithm and verify the effectiveness of the algorithm. We have selected four policy-generated schemes, namely greedy, random, ITSA and DVMC, for experimental comparison with the DREP algorithm proposed in this paper, and then most graphically display the final simulation results.

Greedy policy: Assign tasks to the edge node with the lowest current load factor each time until no compute resources are available. After that, the remaining tasks are assigned to the cloud center.

Random policy: Each task in the task set is randomly assigned in the edge node until there are no available compute resources. After that, the remaining tasks are assigned to the cloud center.

ITSA: Considering both small and large tasks, for each task in the task set, the task with the lowest load value and the task with the highest load value are bound together to form a task pair, and the task pair is assigned to the edge node with the lowest load ratio [5].

DVMC: The task loads are sequentially assigned to edge nodes after sorting them in decreasing order. Through overload detection, the overloaded nodes are adjusted and the task with the highest load value is preferred for removal from the overloaded node. If the node is still overloaded after one migration, the task with the next highest load value is selected for removal. This process is repeated until the node is no longer overloaded [23].

6.1 Evaluation Metrics

The experimental metrics in this paper are mainly compared in terms of load balancing degree and max/average. Through simulation experiments, the effectiveness of the DREP algorithm proposed in this paper is finally proved.

Load balancing is an important indicator to measure the degree of load balance of each edge node, and also reflects the overall performance of the data center. From Eq. 6, the load balance degree of edge data center can be calculated, and the goal is to make it as small as possible.

Max/average also measures the load balance of the data center, where max refers to the edge node with the highest load rate and average refers to the average load rate of all edge nodes. Max/average is closer to 1, which means the load is more balanced, and the optimal value is 1, which means the load rates of all edge nodes are the same.

We constructed two experimental environments, using setting 1 to simulate a homogeneous environment and setting 2 to simulate a heterogeneous environment. And Table 3 shows the parameter settings for our experiments.

Table 3. Parameterization of the experiment

Variable Name	Parameter	Parameter Value
task	R_i^{cpu} (m)	[1, 1000]
	R_i^{mem} (MB)	[1, 1000]
edge	S_j^{cpu} (m)	[5000, 50000]
	S_j^{mem} (MB)	[5000, 30000]

- Setting 1: 500 tasks with 10 edge nodes of the same capability.
- Setting 2: 500 tasks with 15 edge nodes of all different capabilities.

6.2 Heterogeneous Environments

Under setting 2, we compared the load balancing degree of the five algorithms, and the load rate of each edge node of the five algorithms is shown in Fig. 9. The more balanced the load, the smoother the line, and the line of the DREP algorithm tends to be straight, so the overall performance of the DREP algorithm is the best. Greedy algorithm performs well in load balancing but it has low overall load factor. This is because it will assign the tasks that do not fulfill the requirements of the nodes to the cloud for processing, without considering other edge nodes.

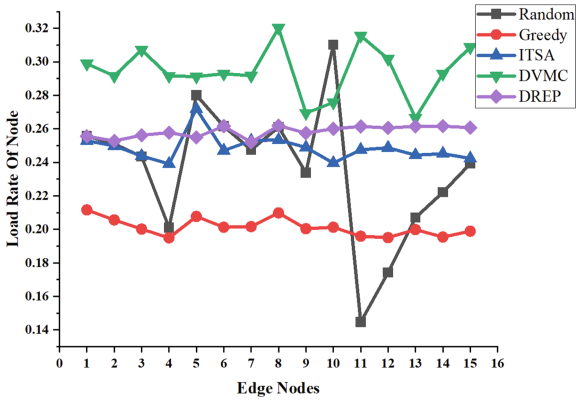

Fig. 9. Comparison on load rate.

In order to compare the advantages and disadvantages among the algorithms more clearly, we calculated the standard deviation of the five algorithms, as shown in Fig. 10. By comparing with the other four algorithms, the standard deviation of the load rate of each node of the DREP algorithm proposed in this paper is the smallest only 0.0032, because the DREP algorithm can assign the

tasks to the most suitable edge nodes according to their sizes, keeping their load ratios on similar levels.

Because there is a random strategy in our proposed algorithm, so we conducted 10 times more comparisons with the current suboptimal greedy algorithm. The results are shown in Fig. 11, as can be seen from the figure, the standard deviations of our proposed algorithms all stay around 0.004, which are lower than the results of the greedy strategy.

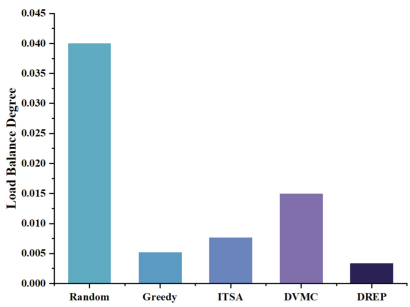

Fig. 10. Compare on load balancing under setting 2.

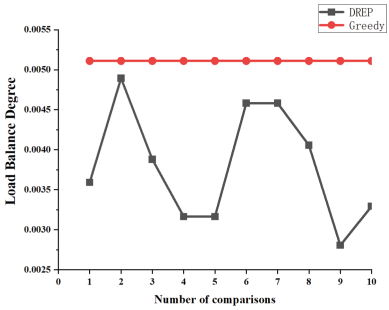

Fig. 11. Comparison of DREP and Greedy Strategies.

Figure 12 represents the comparison of max/average for the five algorithms. The most ideal case ratio is 1, which represents the same load ratio for all edge nodes. And the max/average value of our proposed algorithm is only 1.013, which is already very close to the most ideal load case. This is followed by greedy, ITSA, DVMC and random, respectively.

Figure 13 shows a comparison of the load balancing of the five algorithms with increasing task volume under setting 2. Our proposed algorithm always maintains a low standard deviation because our algorithm takes into account both the capacity and the load of each edge node in a fully heterogeneous environment, which makes it more accurate to filter or eliminate the most suitable task set to maintain the load balance of each edge node. When the task volume is 100–300, the gap between the DREP algorithm and the greedy and ITSA algorithms is not obvious, but as the task volume increases, the advantages of the DREP algorithm keep coming out and the load balancing becomes more and more effective. This is because as the task volume increases, the richer the set of "materials" in the repair phase becomes, so the possibility of load-rate balancing at each edge node becomes greater.

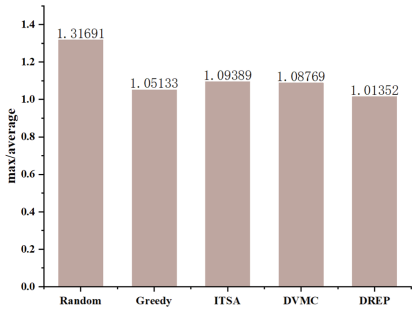

Fig. 12. Comparison on max/average under setting 2.

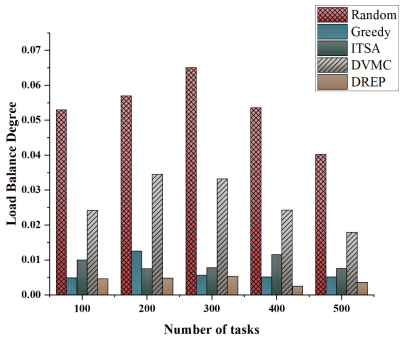

Fig. 13. Comparison of load balancing degree under setting 2.

6.3 Homogeneous Environment

We also verified the effectiveness of the DREP algorithm in a homogeneous environment, where we compared the five algorithms in terms of both the degree of load balancing and max/average under setting 1. Figure 14 shows that in terms of load balancing degree, the standard deviation of the DREP algorithm is only 0.0005, which is 68.65% lower than the next best greedy algorithm; Fig. 15 shows that in terms of max/average, the result of the DREP algorithm is only 1.005, which is very close to the ideal case.

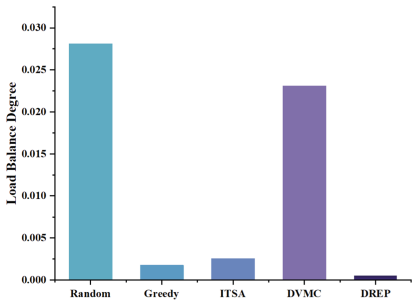

Fig. 14. Comparison on load balancing degree under setting 1.

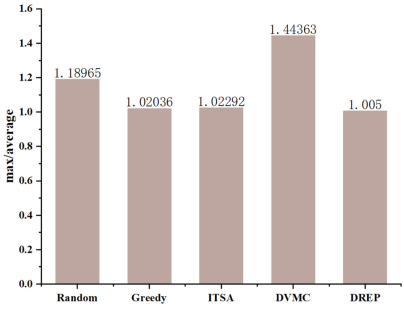

Fig. 15. Comparison on max/average under setting 1.

In setting 1, we compare the load balancing performance of five algorithms with different number of tasks, as shown in Fig. 16. When the number of tasks is 100, the four algorithms perform similarly except for the random algorithm, where the ITSA algorithm has the best load balancing effect and the DREP algorithm has the second best effect. The reason for this situation is that when the number of tasks is small and the requested resources do not differ much, the

load of a single task on a node does not have a great impact on the load balancing. Therefore, the greedy algorithm maintains load balancing by assigning tasks to the node with the lowest current load rate. In this case, the DREP algorithm is unable to repair the load ratio of each edge node because there is too little or no "materials" for the repair phase. However, as the number of tasks increases, the DREP algorithm has the advantage of being able to keep the standard deviation around 0.0004.

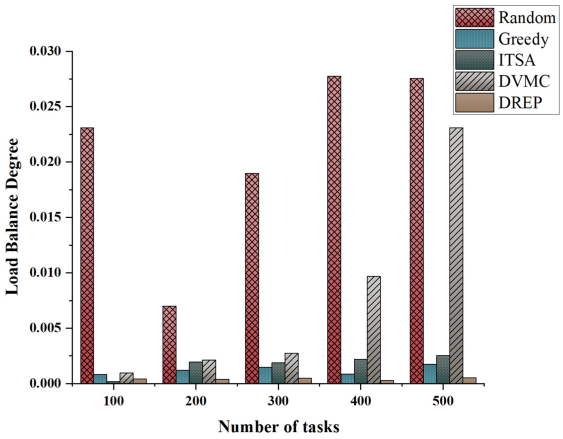

Fig. 16. Comparison on load balancing degree under setting 1.

In other aspects, we conducted the same experiments in a homogeneous environment. We compared the load balancing degree as well as the Max/average of the five algorithms, and since the homogeneous environment is not the focus of this paper, we only describe the experimental results here. The experimental results show that our proposed DREP algorithm consistently achieves the best performance and maintains the most balanced load. Therefore, the DREP algorithm is also applicable and performs better in a homogeneous environment.

In summary, our proposed DREP algorithm outperforms other algorithms overall and performs well in load balancing. According to the experimental results, our method is more suitable for scenarios with a larger number of tasks, because each edge node can have more choices in the repair phase and can better keep the load ratio around the mean by picking the most suitable task set according to its own load.

7 Conclusion

In this paper, we propose a task allocation strategy DREP based on the idea of dichotomous repair, which aims to achieve load balancing among edge nodes and improve the quality of service. Specifically, a dichotomous strategy is used

to generate a good initial allocation scheme based on the number of edge nodes, and then a two-stage repair strategy is proposed for overloaded and underloaded nodes to adjust the load ratio of edge nodes to a relatively approximate level. By comparing the experimental results of the other four allocation algorithms, the DREP algorithm shows better performance and can balance the load among the nodes well and improve the quality of service. In order to further improve the performance and load balancing of cloud-edge computing, we plan to analyze and study from the perspective of distributed multi-cloud and edge. Under the premise of providing high-quality services to users, we will reasonably schedule tasks to multi-cloud and edge to maximize the benefits of data centers and users.

Acknowledgements. Pengwei Wang is the corresponding author. This work was partially supported by the National Natural Science Foundation of China (NSFC) under Grant 61602109, DHU Distinguished Young Professor Program under Grant LZB2019003, Shanghai Science and Technology Innovation Action Plan under Grant 22511100700, Fundamental Research Funds for the Central Universities.

References

1. Shi, W., Cao, J., Zhang, Q., et al.: Edge computing: vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016)
2. Zhu, T., Shi, T., Li, J., et al.: Task scheduling in deadline-aware mobile edge computing systems. IEEE Internet Things J. **6**(3), 4854–4866 (2018)
3. Chen, W., Liu, B., Huang, H., et al.: When UAV swarm meets edge-cloud computing. The QoS perspective. IEEE Network **33**(2), 36–43 (2019)
4. Gai, K., Qiu, M., Zhao, H.: Energy-aware task assignment for mobile cyber-enabled applications in heterogeneous cloud computing. J. Parallel Distrib. Comput. **111**, 126–135 (2018)
5. Zhou, Z., Wang, H., Shao, H., et al.: A high-performance scheduling algorithm using greedy strategy toward quality of service in the cloud environments. Peer Peer Netw. Appl. **13**, 2214–2223 (2020)
6. Panda, S.K., Gupta, I., Jana, P.K.: Task scheduling algorithms for multi-cloud systems: allocation-aware approach. Inf. Syst. Front. **21**, 241–259 (2019)
7. Ali, H.G.E.D.H., Saroit, I.A., Kotb, A.M.: Grouped tasks scheduling algorithm based on QoS in cloud computing network. Egypt. Inform. J **18**(1), 11–19 (2017)
8. Er-raji, N., Benabbou, F.: Priority task scheduling strategy for heterogeneous multi-datacenters in cloud computing. Int. J. Adv. Comput. Sci. Appl. **8**(2) (2017)
9. Zhang, P., Zhou, M.: Dynamic cloud task scheduling based on a two-stage strategy. IEEE Trans. Autom. Sci. Eng. **15**(2), 772–783 (2017)
10. Su, C., Gang, Y., Jin, C.: Genetic algorithm based edge computing scheduling strategy. In: 2021 4th International Conference on Data Science and Information Technology, Shanghai, China (2021)
11. Shen, X.: A hierarchical task scheduling strategy in mobile edge computing. Internet Technol. Lett. **4**(5), e224 (2021)
12. Yang, J., Poellabauer, C.: SATSS: a self-adaptive task scheduling scheme for mobile edge computing. In: 2021 International Conference on Computer Communications and Networks (ICCCN), Athens, Greece, pp. 1–9 (2021)

13. Ruan, X., Chen, H., Tian, Y., et al.: Virtual machine allocation and migration based on performance-to-power ratio in energy-efficient clouds. Future Gener. Comput. Syst. **100**, 380–394 (2019)
14. Wang, H., Tianfield, H.: Energy-aware dynamic virtual machine consolidation for cloud datacenters. IEEE Access **6**, 15259–15273 (2018)
15. Zahedi Fard, S.Y., Ahmadi, M.R., Adabi, S.: A dynamic VM consolidation technique for QoS and energy consumption in cloud environment. J. Supercomput. **73**(10), 4347–4368 (2017)
16. Saadi, Y., El Kafhali, S.: Energy-efficient strategy for virtual machine consolidation in cloud environment. Soft Comput. **24**(19), 14845–14859 (2020)
17. Liang, B., Dong, X., Wang, Y., Zhang, X.: Memory-aware resource management algorithm for low-energy cloud data centers. Future Gener. Comput. Syst. **113**, 329–342 (2020)
18. Zhou, Z., Abawajy, J., Chowdhury, M., et al.: Minimizing SLA violation and power consumption in Cloud data centers using adaptive energy-aware algorithms. Future Gener. Comput. Syst. **86**, 836–850 (2018)
19. Dong, Y., Xu, G., Ding, Y., et al.: A 'joint-me' task deployment strategy for load balancing in edge computing. IEEE Access **7**, 99658–99669 (2019)
20. Li, X., Zeng, F., Fang, G., et al.: Load balancing edge server placement method with QoS requirements in wireless metropolitan area networks. IET Commun. **14**(21), 3907–3916 (2020)
21. Li, G., Yao, Y., Wu, J., et al.: A new load balancing strategy by task allocation in edge computing based on intermediary nodes. EURASIP J. Wirel. Commun. Netw. **2020**(1), 1–10 (2020)
22. Dong, Y., Xu, G., Zhang, M., et al.: A high-efficient joint 'cloud-edge' aware strategy for task deployment and load balancing. IEEE Access **9**, 12791–12802 (2021)
23. Sissodia, R., Rauthan, M.S., Barthwal, V.: A multi-objective adaptive upper threshold approach for overloaded host detection in cloud computing. Int. J. Cloud Appl. Comput. (IJCAC) **12**(1), 1–14 (2022)