Resource-Aware Dynamic Scheduling for Tasks With Deadline Constraints on Edge Computing Systems

Wenbiao Cao, Xiaoyong Tang, Tan Deng, Ronghui Cao, and Keqin Li, *Fellow, IEEE*

***Abstract*—The proliferation of various IoT devices has brought about diverse computing requests. Scheduling delay-sensitive tasks to edge nodes closer to data sources can help alleviate core network congestion and improve system quality of service (QoS). However, with the dynamic computing requirements of changing scenarios and the imbalanced performance of limited heterogeneous edge resources, resource competition among multiple tasks has become increasingly fierce. This resource competition leads to inefficient services and performance fluctuations in edge scheduling systems. The key lies in dynamically matching task requirements and limited heterogeneous resources to improve resource utilization efficiency. To overcome this challenge, we propose a resource-aware task grouping scheduling strategy (RATGS) based on our proposed group-based and shared-state edge scheduling framework, aiming to improve the overall service quality of edge computing systems. We perform extensive evaluation on multiple metrics using realistic workloads and real-world traces. The experimental results demonstrate that RATGS improves the task completion rate by 7.56%∼50.1% before the deadline and improves the efficiency of resource utilization by 17.7%∼94.8% compared with existing baseline strategies. In addition, RATGS performed second best in terms of average completion time.**

***Index Terms*—Edge computing, task scheduling, deadline constraint, resource utilization.**

I. Introduction

With the development of artificial intelligence Internet of Things (AIoT), multi-access edge computing (MEC) as a promising paradigm has attracted more attention from researchers [1], [2]. It migrates some computing requests accessing the cloud center to edge servers closer to the data source. This helps to alleviate the congestion of the core network and ensure the growing quality of service requirements, especially for applications with low latency and high reliability [3], [4]. However, the distributed heterogeneous characteristics of computing resources, the increasingly diverse dynamic requirements, and the limited nature of resources pose challenges to efficient scheduling at the edge [5], [6].

How to dynamically adapt tasks to limited computing resources is the key to improving edge system performance. First, edge nodes have obvious heterogeneity in terms of resource type, performance, and architecture. They usually deploy limited computing services, such as virtual reality, face recognition, and object detection [7], [8]. However, applications deployed at edge usually have strict deadline requirements [9], [10]. The contradiction between the performance differences between nodes and the deterministic requirements of deadlines may lead to an imbalance in computing load. Specifically, when there is a powerful node in the area, the gateways that do not interact with information will prioritize the tasks to the powerful nodes, which will affect the overall performance and resource utilization of the system.

Second, resource competition is widely present in edge networks. Tasks from different network links compete for limited computing resources, which involves the priority of resource allocation. It is worth noting that cloud edge service providers provide computing services based on service level agreements (SLAs) with users. Violating task deadlines will have a negative impact on the user's QoE and reflect the QoS of system [11], [12], [13], [14]. Moreover, this impact will fluctuate with changes in the computing environment. For example, in computing scenarios related to train stations and high-speed rail stations, violating the deadline of face recognition tasks during peak hours may cause greater QoE loss than during idle periods. Therefore, resources should be prioritized for tasks that are more beneficial to the current computing environment, aiming to improve users' long-term satisfaction with system services [15].

Besides, task scheduling should comprehensively consider task requirements and resource status changes to achieve dynamic scheduling under resource constraints [16], [17], [18]. In real-edge scheduling scenarios, the execution information of tasks is uncertain before they are offloaded to computing nodes. When multiple tasks with deadline constraints arrive simultaneously, different scheduling schemes for these tasks will produce varying results. As shown in Fig. 1, all three scheduling schemes can meet or miss the deadlines of t_1, t_2, t_3, and t_4. The third scheduling scheme is obviously the best. However, when tasks go online in chronological order and information is unknown, it is difficult to find the optimal schedule immediately. The optimal scheme can be achieved by adjusting the execution order later, such as from Scheme 1 to 3. Second, the state of

Received 18 August 2024; revised 29 June 2025; accepted 24 August 2025. Date of publication 28 August 2025; date of current version 8 December 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62372064, in part by Hunan Provincial Natural Science Foundation of China under Grant 2025JJ50347 and Grant 2025JJ50409, and in part by Hunan Province Science and Technology Innovation Plan Project under Grant YLS-2025-ZY03023. Recommended for acceptance by J. Ren.*(Corresponding author: Xiaoyong Tang.)*

Wenbiao Cao, Xiaoyong Tang, Tan Deng, and Ronghui Cao are with the School of Computer and Communications Engineering, Changsha University of Science & Technology, Changsha, Hunan 410114, China (e-mail: tangxy@csust.edu.cn).

Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA.

Digital Object Identifier 10.1109/TCC.2025.3603782

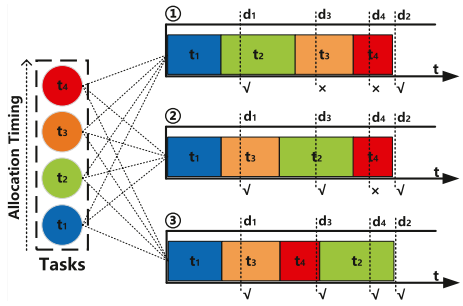

Fig. 1. An example for multiple tasks scheduling.

resources is constantly changing. As shown in Fig. 2, static scheduling leads to idle resources in two time periods, while through dynamic task adjustment, the system can further utilize resource fragmentation and improve resource utilization.

Overall, our key insight is that improving the efficiency of resource utilization is more important than simply improving utilization. The system should complete as many requests as possible with limited resources. Thus, this paper focuses on how to effectively schedule deadline-constrained tasks to improve the service quality of edge scheduling systems. This paper is an extension of our conference paper [11], and its main contributions are as follows:

- First, we build a *group-based and shared-state resource management model*. The model abstracts heterogeneous physical resources and logically organizes computing resources into multiple computing groups. Each computing group provides a similar type of service. The state of computing units is shared among multiple computing groups, reflecting their collaborative management.
- Second, we mathematically formulate the deadline-constrained multi-task scheduling problem on multiple machines with resource competition, aiming to maximize the utilization efficiency of limited resources.
- Third, we propose a resource-aware task grouping scheduling strategy (RATGS). The strategy includes three parts: task regrouping and priority response model, a resource-aware greedy scheduling algorithm, and a task adjusting method.
- Fourth, we employ realistic workloads and real-world traces to evaluate our strategy. Experimental results show that RATGS can adapt to the changes of dynamic resource status and achieve higher overall quality of service for edge scheduling systems.

The rest of this paper is organized as follows: we summarize the related work in Section II. We introduce the system model and problem formulation in Section III. Then, we present our strategy in Section IV. Experimental evaluation and discussion of results are presented in Section V, and the related work is in Section VI. Finally, we conclude this paper in Section VII.

II. System Model and Problem Formulation

This section first describes the multi-access heterogeneous edge computing system model and architecture, shown in Fig. 3. This architecture mainly consists of a Network Model, Edge Resource Management and Scheduling Model, and Task Arrival and Execution Model. Then, we present the mathematical formulation of the problem and conduct the complexity analysis. The key notations used in this paper are explained in Table I.

TABLE I
Explanation of Notations Used in This Article

Notation	Description
E_i	The i-th edge computing node
S_i	The set of services in E_i
E_i^c	The execution parallelism of E_i
E_i^m	The memory capacity of E_i
f_i	The baseline computing power of E_i
E_i^*	The i-th node in the summary node set E^*
$b_{i,j}(\tau)$	The bandwidth between E_i and E_j at time τ
E_{cloud}	The remote cloud
ρ	The packet loss rate caused by network jitter
α_{cloud}	The propagation delay from edge to cloud
β_{cloud}	The communication bandwidth between edge to cloud
X_h	The h-th computing group
$x_{h,k}$	The k-th computing unit in h-th computing group
$f_{h,k}$	The baseline computing power of $x_{h,k}$
$m_{h,k}$	The actual memory required to run $x_{h,k}$
$st_{h,k} \in \{0,1\}$	The state of $x_{h,k}$
R^*	The local resource matrix maintained on E_i^*
D_j	The device offloading task t_j
rs_j	The task t_j is released by D_j time rs_j
t_j^{kind}	The requested type of service of task t_j
t_j^{load}	The computing amount of t_j
t_j^{data}	The amount of data uploaded by t_j
$t_j^{arrival}$	The arrival time of t_j
$t_j^{deadline}$	The deadline of t_j
T_j^{delay}	The total delay of t_j
T_j^{up}	The upload delay of t_j
T_j^{down}	The time of returning results to user devices
$\alpha_{j,i}^*$	The propagation delay from D_j to the nearest E_i^*
T_j^{trans}	The transmission delay from D_j to the nearest E_i^*
$\beta_{j,i}^*$	The bandwidth between D_j and E_i^*
T_j^{rsp}	The response time of t_j
$T_j^{interTrans}$	The internal transmission time for t_j
$T_j^{process}$	The execution time of t_j
T_j^{wait}	The waiting time of t_j
$T_{j,*}^{wait}$	The time waiting to be dispatched on E_i^*
$T_{j,h,k}^{wait}$	The waiting time executed on $x_{h,k}$
ϑ_j	The execution status of t_j
w_h	The SLA between service provider and user for service s_h
φ_j^h	The benefit of system after executing t_j in group X_h
T_j^{slack}	The slack time of t_j from Current time to deadline
rw_j	The task completion reward of t_j

A. System Model

1) Network Model: We consider a three-layer network architecture diagram of cloud-edge-end. As shown in Fig. 4, a series of computing nodes $E_i \in \{E_1, E_2, \ldots, E_n\}$ such as gateways (access points), local servers, small data centers, and even high-performance terminal devices are geographically distributed. The gateway is bound to the base station or edge computing node. In this system, the gateway-bound node is defined as the summary nodes E^*, managing one or more edge computing nodes in nearby geographical areas. Multiple summary nodes maintain static routing tables between each other. The bandwidth

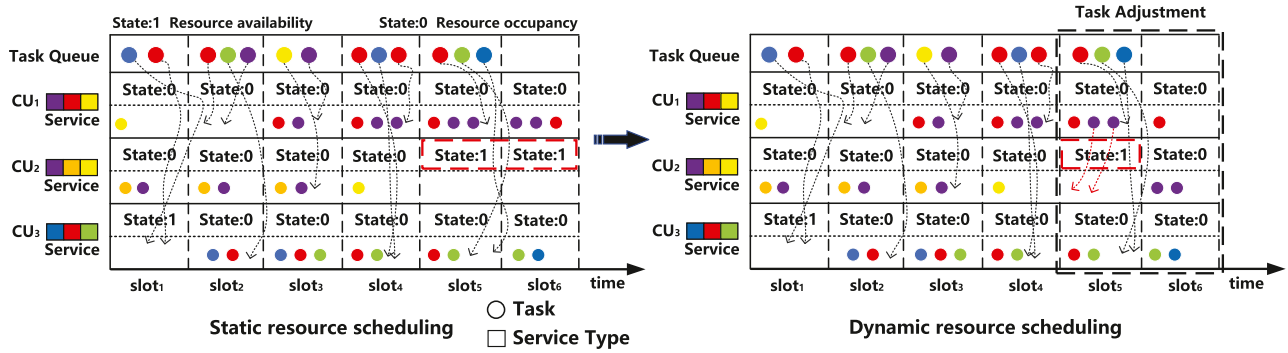

Fig. 2. An example of dynamic resource status and scheduling diagram.

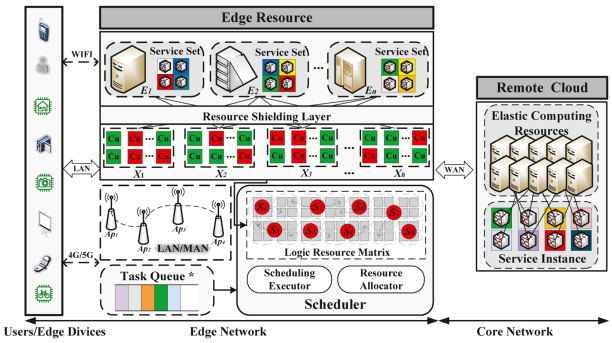

Fig. 3. The multi-access edge computing systems architecture.

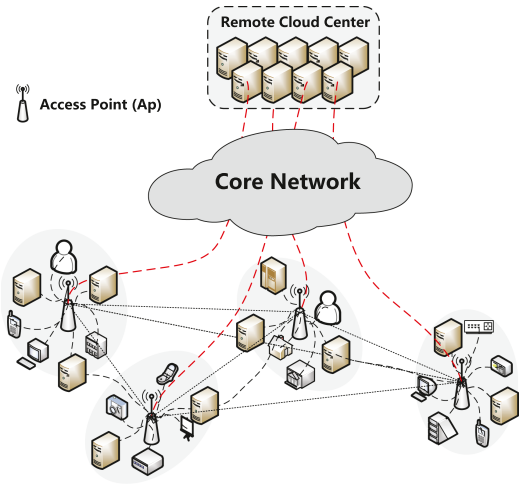

Fig. 4. The cloud-edge-end three-layer network architecture.

between edge computing nodes is represented as

$$B(\tau) = \begin{bmatrix} b_{1,1}(\tau) & b_{1,2}(\tau) & \cdots & b_{1,J}(\tau) \\ b_{2,1}(\tau) & b_{2,2}(\tau) & \cdots & b_{2,J}(\tau) \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1}(\tau) & b_{n,2}(\tau) & \cdots & b_{n,J}(\tau) \end{bmatrix}, \quad (1)$$

where, $b_{i,j}(\tau)$ represents the bandwidth between E_i and E_j at time τ. Additionally, the remote cloud E_{cloud} is regarded as a special server with sufficient computing resources. By flexibly launching instances, almost all requests can be responded to by E_{cloud}. We assume that executing task in E_{cloud} is *non-blocking* but will suffer from significant communication delays. Through WIFI, Ethernet, or 4G/5G mobile networks, user device $D_j \in \{D_1, D_2, \ldots, D_m\}$ can offload computing requests to the nearest edge access point (AP).

2) Edge Resource Management Model: In this part, we propose a *group-based and shared-state resource management model*. This model constructs a *resource shielding layer* that abstracts locally distributed heterogeneous resources into service groups based on service types [14] and shares the node status between local summary nodes [19].

Specifically, we first assume that all edge computing nodes are equipped with multi-core, multi-threading, and even multiple heterogeneous resources. This implies that tasks can be processed in parallel. The computing resource of E_i can be represented as $Resource(E_i) =< E_i^c, E_i^m, S_i, f_i >$, where S_i, E_i^c, E_i^c and f_i represents the set of services, execution parallelism, memory capacity, and baseline computing power, respectively. Then, we redefine the edge computing unit (CU). We logically divide an edge computing node E_i into multiple computing slots based on E_i^c. These computing slots are configured with different operating environments and services and divided into multiple groups according to their service types, represented as $X_h = \{x_{h,1}, x_{h,2}, \ldots, x_{h,k}\}$. The k-th computing unit in h-th computing group is defined as $x_{h,k} =< f_{h,k}, m_{h,k}, st_{h,k} >$. $f_{h,k}, m_{h,k}$, and $st_{h,k} \in \{0, 1\}$ represent the benchmark performance, the actual memory required to run the service, and the state of $x_{h,k}$, respectively. The available resources of the h-th computing group are represented as $Xnum(h) = \sum_{k=1}^{K} |x_{h,k}|_{st_{h,k}=1}, \forall x_{h,k} \in X_h$. Next, we implement a resource shielding layer in each summary node to isolate the mapping of tasks to underlying resources while only focusing on the scheduling between computing services and tasks. The underlying resource management is transparent to task scheduling. Finally, the heterogeneous computing resources are defined as a two-dimensional logical resource matrix R^*, represented as $R^* = [X_1, X_2, \ldots, X_h]$.

Note that edge computing nodes are not interconnected in pairs. Moreover, an edge computing node may be managed by two or even more summary nodes, sharing resource states. Therefore, each summary node only maintains the resource

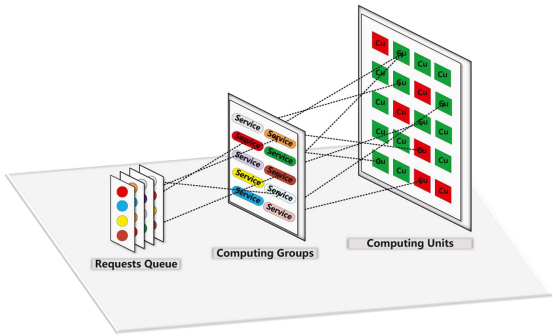

Fig. 5. The one-to-one mapping relationship for request-service and one-to-many mapping relationship for service-computing resource.

matrix of the local edge network. Moreover, the scheduling scenario and the actual environment are changing dynamically. Hence, the service set deployed on the edge computing node will also be adjusted dynamically. Excessive service deployment adjustment may cause system instability, so periodic adjustment is necessary. The deployment of service sets between nodes may overlap, which ensures the reliability of the service. Resource abstraction also helps to cope with unpredictable changes in resource status, such as node failures, changes in network load, etc. Moreover, this approach is highly scalable and can utilize the idle shared computility of edge devices [20] to improve the system capacity.

To further improve scheduling efficiency, we adopt a *two-stage scheduling framework*. Specifically, we deploy a sub-scheduler for each computing group and a global scheduling executor for summary node E_i^*. The sub-scheduler maintains a *node performance table* for the computing group based on historical information and is responsible for scheduling requests assigned to the computing group. The global scheduling executor is responsible for handling scheduling conflicts caused by online resource competition and making adjustment decisions based on the synchronization status information shared from other nodes. We also deploy a state monitor on each edge computing node E_i. The state monitor maintains a *mapping table* that records the relationship between the summary node and the instance running on the node. Through this fine-grained scheduling, we achieve a one-to-one mapping between tasks and computing services, and a one-to-many mapping between services and computing units, as shown in Fig. 5. It means that each computing unit can execute a maximum of one task at a time. By updating the resource matrix in real-time, we can adaptively adjust the scheduling plan according to the resource status.

3) Task Arrival and Execution Model: Multiple user devices randomly offload computing requests to the edge system. We assume that these tasks are independent and indivisible modules of applications, such as speech and face recognition, crowdsensing, and object detection [21]. We set that each E_i^* maintains a dispatching buffer queue, denoted to Q_i^*.

Once a task t_j arrives at AP_i, it will be placed in Q_i^* to wait for a response and then dispatched to an appropriate node $x_{h,\overline{k}}$ by the scheduler on E_i^*. Based on the computing requirements of t_j, its execution will be scheduled using a virtual machine or container as a carrier. The requirement information of t_j can be expressed as $Req(t_j) =< t_j^{kind}, t_j^{load}, t_j^{data}, t_j^{arrival}, t_j^{deadline} >$, where $t_j^{kind}, t_j^{load}, t_j^{data}, t_j^{arrival}$, and $t_j^{deadline}$ represent the requested type of service, the computing amount, the amount of transferred data, the arrival time, and its expected deadline, respectively. The resource constraints to running a service are expressed as

$$\sum_{h=1}^{H}\sum_{k=1}^{K} f_{h,k} + f_{h,\overline{k}} \le \sum_{i=1}^{n} E_i^c f_i, \forall x_{h,k}|_{st_{h,k}=0}, h \in S_i, \tag{2}$$

$$\sum_{h=1}^{H}\sum_{k=1}^{K} m_{h,k} + m_{h,\overline{k}} \le \sum_{i=1}^{n} E_i^m, \forall x_{h,k}|_{st_{h,k}=0}, h \in S_i. \tag{3}$$

We assume that the edge device offloads t_j at time rs_j. The total delay of t_j includes three parts: $T_j^{delay} = T_j^{up} + T_j^{rsp} + T_j^{down}$. Here, T_j^{up}, T_j^{rsp}, and T_j^{down} represent the upload delay, the response time, and the time of returning the result to the user device, respectively.

The upload delay includes two parts: $T_j^{up} = \alpha_{j,i}^* + T_j^{trans}$. Here, $\alpha_{j,i}^*$ represents the propagation delay from D_j to the nearest E_i^*. It is related to the transmission distance and environment. T_j^{trans} represents the transmission delay, expressed as $T_j^{trans} = t_j^{data}/\beta_{j,i}^*$. $\beta_{j,i}^*$ represents the bandwidth between D_j and E_i^*. The amount of data returned from edge nodes to edge devices for execution results is usually small, so transmission delay can be almost ignored. We only consider its propagation delay and it usually takes a very small average value.

The response time of t_j includes three parts: $T_j^{rsp} = T_j^{wait} + T_{j,\overline{k}}^{interTrans} + T_{j,h,\overline{k}}^{process}$. Here, T_j^{wait} includes the time waiting to be dispatched on E_i^* and the waiting time executed on $x_{h,\overline{k}}$, which are denoted to $T_{j,*}^{wait}$ and $T_{j,h,\overline{k}}^{wait}$. $T_{j,\overline{k}}^{interTrans}$ represents the time to send t_j from E_i^* to $x_{h,\overline{k}}$, expressed as $T_{j,\overline{k}}^{interTrans} = t_j^{data}/b_{i,k}(\tau), x_{h,\overline{k}} \in E_k$. $T_{j,h,k}^{process}$ represents the execution time of t_j on $x_{h,k}$.

In addition, if the current resource state of the edge network cannot satisfy the requirement of t_j, t_j will be dispatched to a remote cloud center for execution. We assume that scheduling t_j to the cloud will be executed immediately without waiting time. Therefore, the total delay of scheduling t_j to the remote cloud center can be expressed as

$$T_{j,cloud}^{delay} = 2\alpha_{cloud} + (1+\rho)\frac{t_j^{data}}{\beta_{cloud}} + \frac{t_j^{load}}{f_{cloud}} + T_{j,*}^{wait}. \tag{4}$$

Here, ρ is the packet loss rate caused by network jitter during task transmission. α_{cloud} and β_{cloud} represent the propagation delay and communication bandwidth from edge to cloud. To better represent the state of t_j, we introduce a variable $\vartheta_j \in \{-1, 0, 1\}$ to indicate the execution status of t_j, represented as

$$\vartheta_j = \begin{cases} 1, & rs_j + T_j^{delay} \le t_j^{deadline}, \\ -1, & rs_j + T_j^{delay} > t_j^{deadline}, \\ 0, & \text{otherwise}. \end{cases} \tag{5}$$

We set $\vartheta_j = 1$ if t_j is completed before the deadline, and set $\vartheta_j = -1$ otherwise. Besides, we set $\vartheta_j = 0$ if t_j is interrupted or abandoned.

B. Problem Formulation

Our research focuses on the scheduling of deadline-constrained tasks in edge computing. A key issue is how to improve the efficiency of resource utilization, that is, to complete more requests within a limited time frame and under limited resources, especially prioritizing to complete more urgent tasks to maximize the benefits of edge computing systems. As mentioned in the introduction, different users perceive delays differently and therefore have different tolerances for deadline violations. Thus, the benefits of the system can also reflect the long-term satisfaction of users with the system services. The goal of our research is to maximize the quality of system service and minimize the loss of user experience.

To more clearly express the overall benefits of the system, we first introduce the concept of "penalty" and "reward" to express the impact of task completion on user satisfaction with the system service. Here, the "penalty" and "reward" are defined as the influencing factors on the long-term satisfaction of system service (LTSS). The LTSS is also used to reflect the QoE. If a task is completed before its deadline, the system will receive a positive reward. Otherwise, the system will receive a negative reward (penalty). Then, we set a dynamic processing urgency for all delay-sensitive tasks. As we mentioned in the introduction (Section I), the computing task of object detection is superior to other modules on complex roads. Thus, we define initial processing weight factors for different types of tasks, defined as $w \in \{w_1, w_2, \ldots, w_h\}$, which is determined by the SLAs between service providers and users. Next, to adapt to the changes in different scenarios, we assume that the weight factors of tasks are dynamically changing. The benefit of t_j in computing group X_h is expressed as

$$\varphi_j^h = w_h \times exp\left(-\frac{T_j^{slack}}{t_j^{deadline} - rs_j}\right). \tag{6}$$

Here, T_j^{slack} represents the slack time of t_j from arrival time to deadline, denoted as $T_j^{slack} = t_j^{deadline} - t_j^{arrival}$. In real scenarios, we can use performance prediction techniques to predict the computing requirements and the urgency of tasks in dynamic environments. Meanwhile, we introduce the variable to represent the allocation of t_j, expressed as

$$y_{h,k}^j \in \{0, 1\}, \tag{7a}$$

$$\sum_{h=1}^{H}\sum_{k=1}^{K} y_{h,k}^j \leq 1, \forall j. \tag{7b}$$

If t_j is assigned to $x_{h,k}$, $y_{h,k}^j = 1$. Otherwise, $y_{h,k}^j = 0$. We use a request-resource pair $< t_j, x_{h,k} >$ to represent the allocation result of t_j. The task completion reward of t_j for system can be expressed as

$$rw_j = y_{h,k}^j \vartheta_j \varphi_j^h. \tag{8}$$

Finally, the LTSS of system can expressed as

$$LTSS = \sum_{h=1}^{H}\sum_{j=1}^{m} rw_j. \tag{9}$$

Note that distributed resources in edge environments and limitations of service deployment increase the overhead of matching available resources to tasks. This overhead is unacceptable for some urgent tasks. Therefore, the system should provide different quality of service for tasks. We use the completion rate before deadline (CRD) to comprehensively express this service quality. It can be expressed as

$$CRD = \frac{NC(\vartheta_j)_{\vartheta_j=1,\forall j}}{NC(\vartheta_j)_{\vartheta_j\in\{-1,0,1\},\forall j}}. \tag{10}$$

where, NC represents the number of tasks. Through the system's scheduling, more tasks that meet their deadlines indicate an improved user experience, which means that the system's positive benefits are higher. Therefore, the optimization process of maximizing the task completion rate before the deadline (CRD) includes maximizing the long-term satisfaction of the system service (LTSS).

Ultimately, the main goal of this paper (optimizing the utilization efficiency of limited resources) can be paraphrased as maximizing CRD and $LTSS$ as follows.

$$Maximize \quad CRD, LTSS. \tag{11}$$

$$\text{s.t.} \quad \text{Eq.}(2) \sim (3), (5), (7\text{a}) \sim (7\text{b}).$$

C. Problem Complexity Analysis

In this part, we analyze the deadline-constrained multi-task scheduling problem on multiple machines with resource competition.

Theorem 1: The deadline-constrained multi-task scheduling problem on multiple machines with resource competition is NP-hard.

Proof: First, in our model, multiple tasks are offloaded to the edge system and then assigned to multiple machines. The problem of assigning multiple tasks on multiple machines under resource constraints to has been shown to be NP-hard [9], [22]. This scheduling problem is also included in our study. In fact, optimizing the number of tasks that meet their deadlines is a subproblem of optimizing the sum of completion times because it aims to optimize the completion time of each task within a finite range to ensure that the total completion time reaches an optimal range value. Thus, we conclude that the multi-task scheduling problem on multiple machines with deadline constraints is still an NP-hard problem.

Second, our study also includes a grouping-based multi-task scheduling problem to maximize the system service benefits under resource competition. To prove that this problem is NP-hard, we derive it from the multidimensional multiple-choice knapsack problem (MMKP) promoted by 0-1 knapsack [23], a problem that has been proven to be NP-hard. In the MMKP problem, the backpack is divided into h backpacks H_i and each

backpack contains a load-bearing W_k $(k = 1, 2, 3, \ldots, h)$. At the same time, there are n types of items, and each type of item has several S_i $(i = 1, 2, \ldots n)$, each S_i with a non-negative value $V_{i,j}$ $(j = 1, 2, \ldots)$. There is also a weight constraint here, that is, each item loaded into a different backpack has a different weight $w_{i,j,k}$. Our goal is to select at most one item from each of the H backpacks while maximizing the total benefits while meeting system resource constraints of (2) and (3). This problem can be formalized as

$$\max \sum_{i=1,2,3..}^{h} \sum_{j \in H_i} V_{i,j} x_{i,j} \tag{12}$$

Subject to

$$\begin{cases} \sum_{i=1,2,\ldots}^{h} \sum_{j \in H_i} w_{i,j,k} x_{i,j} \leq W_k \\ \sum_{j \in H_i} x_{i,j} \leq 1 \quad \forall i, \\ x_{i,j} \in \{0, 1\} \quad \forall i, j. \end{cases}$$

The system maps multiple tasks to a computing group. At the time slot of each system action, each sub-scheduler of the computing group selects a task to execute in parallel from the computing group. Then, these tasks are selected to be assigned to a computing unit and occupy a resource of the computing unit. Here, because a computing group may deploy multiple computing services, the allocation of physical resources in the computing group may overlap at the underlying level. Therefore, a task can be assigned to multiple heterogeneous computing nodes, occupying different lengths of computing resource segments, namely $w_{i,j,k}$ mentioned above. Moreover, if the status of the current resource cannot meet the computing requirements of these tasks, we will obtain the current optimal scheduling plan through task adjustment. At the same time, the optimization goal of our scheduling is to maximize the reward of system service. Although rw_j after completing each task is either positive or negative, maximizing long-term satisfaction of system services for users and maximizing long-term benefits in MMKP problems still belong to the same optimization problem. Thus, we deduce that the problem of grouping-based multi-task scheduling to maximize the long-term satisfaction of system service also is an NP-hard problem. □

III. Resource-Aware Task Grouping Scheduling Strategy

In this section, we propose a resource-aware task grouping scheduling strategy (RATGS) to solve task scheduling problems with deadline constraints. Our strategy is divided into three main parts. The first phase is to develop a task regrouping and priority sorting model, aiming to balance efficiency and fairness for the response of tasks, as shown in Fig. 6. Then, we propose a resource-aware greedy scheduling algorithm. The algorithm quickly matches the appropriate execution location for the task based on the resource status. Finally, a task adjusting method is proposed to solve the online resource competition between tasks and further improve the resource utilization efficiency. Their details will be explained in the following sections.

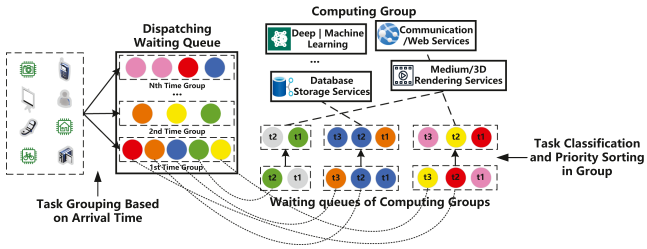

Fig. 6. The process of regrouping and sorting for multiple tasks.

Algorithm 1: Tasks Regrouping and Priority Sorting.

Input: Resource Matrix R^*, Service Set S, A set of tasks T
Output: $NewQueue$

for each $i \in T$ **do**
  if $t_i^{kind} \in S$ **then**
    Initialize array of p
    mapping t_i into Computing Group X_h from R^*
    $PreQueue \leftarrow$ obtain wait queue of X_h
    for each $j \in PreQueue$ **do**
      $\bar{rp}_j \leftarrow$ Calculate $\bar{rp}_j$ of $PreQueue[j]$ by (14)
      $p[j] \leftarrow$ Calculate $p[j]$ by (13)
    end for
    $NewQueue \leftarrow$ Sort tasks of $PreQueue$ by p
  else
    $y_{h,k}^{j} = 0$
    Scheduling t_j to E_{cloud} by (4)
  end if
end for

A. Task Regrouping and Priority Response Model

At any time, a summary node may accept multiple computing requests. To adapt our proposed group-based and shared-state resource management model, it is necessary to group tasks based on their computational requirements. This also helps to quickly match them with available resources. In fact, scheduling tasks to a service instance for execution may incur cold start overhead for service startup (this is beyond the scope of our discussion, as we assume that all service instances at the edge are pre-warmed in advance). Therefore, scheduling tasks to local "home" nodes helps services start quickly without having to pull images from remote repositories.

To further manage tasks with online random arrival patterns, we divide tasks into multiple timing groups according to their arrival timing. The summary node responds to the tasks in a timing group in turn, and then classifies and maps them to the computing group X_h. Moreover, considering the different computing requirements of tasks, it is necessary to prioritize tasks within X_h.

Specifically, as shown in Fig. 6, we first determine the request information associated with each task. Then, we subsequently group them based on their arrival time. Next, we sequentially classify these tasks in each time series group into the corresponding computing groups. Finally, we calculate their priorities and insert them into the waiting queue of the corresponding

computing group. The priority weight of t_j is represented as

$$p_j = \frac{\overline{rp_j}}{T_j^{slack}(\tau)}. \tag{13}$$

where $T_j^{slack}(\tau) = t_j^{deadline} - \tau$. τ represents the current time. Due to the heterogeneity of resources, the remaining execution time of t_j cannot be directly and accurately calculated online. So, we determine the priority of t_j by introducing the average execution time on K slot nodes in the computing group it belongs to, represented as

$$\overline{rp_j} = \frac{\sum_{k=1}^{K} f_k}{k \times t_j^{load}}. \tag{14}$$

Note that the computing power f_k of K slot nodes is obtained from the historical *node performance table* maintained by each sub-scheduler. According to this equation, it is evident that for tasks with the same deadline, the larger the p_j, the less reserved time for task execution and return, and the more urgent the response demands.

After regrouping and prioritization, the sub-scheduler of each computing group schedules tasks to an appropriate node in parallel according to the priority. Algorithm 1 outlines the pseudocode of the task regrouping and priority sorting.

B. Resource-Aware Greedy Scheduling Algorithm

The computing resources are limited and dynamically change over time. The replica status of computing resources is updated in real time. Therefore, the decision of task scheduling should adapt to the current changes in resource status. In this part, we propose a resource-aware scheduling algorithm that aims to improve resource utilization efficiency while ensuring that tasks are completed before their deadlines. The pseudocode of the algorithm is shown in Algorithm 2.

First, Each sub-scheduler schedules the first task of each computation group in parallel. For the scheduling of task t_j, if there is an idle computing unit directly available and the resource status satisfies (2) and (3), then t_j is greedily scheduled to the node with the shortest completion time (including internal transmission delay and processing time) (Steps 2-9 in Algorithm 2). Otherwise, the current resource utilization rate is close to full load and the state of computing resources is tight. Thus, we sequentially preallocate t_j to each slot node of X_h and calculate the earliest completion time, expressed as

$$EFT(x_{h,k}, t_j) = EAT(x_{h,k}) + T_{j,h,k}^{wait} + T_{j,h,k}^{process}. \tag{15}$$

Here, we assume that $x_{h,k}$ obtains the next idle time slot at time $EAT(x_{h,k})$ and define an array $EFT[]$ to store the earliest completion time (Steps 10-13 in Algorithm 2). If the minimum value in $EFT[]$ meets the deadline of t_j, we dispatch t_j to a node with the earliest completion time and mark it with an allocation pair $< t_j, x_{h,\overline{k}} >$ at time ea_j. Once the current resource status meets its deadline, resource allocation follows the $FCFS$ strategy (Steps 14-16 in Algorithm 2). Otherwise, if there is no node that meets the deadline, we call Algorithm 3 to try to find a feasible solution. When Algorithm 3 also cannot find a feasible solution, schedule t_j to the E_{cloud} (Steps 17-24 in Algorithm 2).

Algorithm 2: Resource-Aware Greedy Scheduling Algorithm.

Input: Task t_j, Computing Group X_h
Output: Task-Computing node allocation pair $< t_j, x_{h,\overline{k}} >$

1: Initialize array of $AvaNodes$, EFT, EAT as $\emptyset$
2: **for** each $x_{h,k} \in X_h$ **do**
3: **if** $x_{h,k}$ is idle && satisfy (2) and (3) **then**
4: $AvaNodes \leftarrow$ add $x_{h,k}$, $Xnum(h)$ ++
5: **end if**
6: **end for**
7: **if** $Xnum(h) \neq 0$ **then**
8: $x_{h,\overline{k}} \leftarrow$ Select $Min\{T_{j,h,k}^{process} + T_{j,k}^{interTrans}\}$ from $AvaNodes$
9: Obtain allocation pair $< t_j, x_{h,\overline{k}} >$, $y_{h,\overline{k}}^j = 1$
10: **else**
11: $EAT \leftarrow$ Update $EAT(x_{h,k})$ of X_h
12: $EFT \leftarrow$ Calculate $EFT(x_{h,k}, t_j)$ of X_h by (15)
13: $x_{h,\overline{k}} \leftarrow$ Select a node with $Min\{EFT[]\}$
14: **if** $EFT(x_{h,\overline{k}}, t_j) \leq t_j^{deadline}$ **then**
15: Obtain allocation pair $< t_j, x_{h,\overline{k}} >$, $y_{h,\overline{k}}^j = 1$
16: Dispatching t_j to computing node $x_{h,\overline{k}}$
17: **else**
18: Preallocation t_j to $x_{h,\overline{k}}$ and calling Algorithm 3
19: **if** Algorithm 3 exist a feasible solution **then**
20: Obtain allocation pair $< t_j, x_{h,\overline{k}} >$, $y_{h,\overline{k}}^j = 1$
21: Dispatching t_j to $x_{h,\overline{k}}$ and executing task adjusting
22: **else**
23: $y_{h,\overline{k}}^j = 0$
24: Scheduling t_j to E_{cloud} by (4)
25: **end if**
26: **end if**
27: **end if**

C. Task Adjusting Method

Due to the possibility of overlapping resource coverage for each logical computing group, the process of parallelly scheduling multiple tasks is inevitably accompanied by resource competition. Thus, we propose a task adjusting algorithm for this problem. Task adjusting contains two core ideas: the "bin packing" strategy and the power of "waiting". The former dynamically allocates resource fragments based on the resource status to improve resource utilization, while the latter considers the dynamic changes of resources and balances the availability of resources with the completion time. The pseudocode of task adjusting is shown in Algorithm 3.

Specifically, we define the resource allocation plan for a series of tasks as $Scheme$. When appending a new task to a node's $Scheme$ and its deadline cannot be met, we first try to insert it into the original scheme without affecting the timely completion of other tasks. This insertion sacrifices the average

Algorithm 3: Task Adjusting Algorithm.

Input: Task t_j, Original scheme $PreScheme$ on $x_{h,\bar{k}}$
Output: an optimal schedule $BestScheme$

```
1: Initialize array of RLT as ∅
2: Index ← Calculate the position Index of PreScheme
3: Divide PreScheme into AQ(t_k) and NAQ(t_k) by Index
4: RST ← Calculate RST_{t_k} of PreScheme by (17)
5: if exist a solution satisfying (16) && (18) then
6:     BestScheme ← Insert t_j into Index of PreScheme
7: else
8:     PreCRD ← Calculate CRD of PreScheme by (10)
9:     PreRW ← Calculate LTSS of PreScheme by (9)
10:    NewScheme ← Insert t_j into Index of PreScheme and reschedule tasks in AT*(t_k)
11:    NCRD ← Calculate CRD of NewScheme by (10)
12:    NRW ← Calculate LTSS of NewScheme by (9)
13:    if exist a solution satisfying (19) then
14:        Calling Algorithm 4 to reschedule t*_k ∈ AT*(t_k)
15:        BestScheme ← NewScheme
16:    else
17:        Search a appropriate task from NAQ(t_k)
18:        if exist a solution satisfying (21) then
19:            BestScheme ←replace the task for PreScheme
20:        end if
21:    end if
22: end if
```

time to complete tasks but completes more tasks before the deadline. However, when we have to abandon some tasks due to resource competition, prioritize resources for tasks that are more beneficial to the LTSS of the system. This algorithm attempts to obtain a better scheduling scheme ($BestScheme$) than the original scheme ($PreScheme$) from a global perspective. The $BestScheme$ must satisfy the following two conditions:

Condition 1: $BestScheme$ must complete the same or more tasks before deadlines than $PreScheme$.

Condition 2: $BestScheme$ must achieve higher LTSS from completing tasks than $PreScheme$.

$$\begin{cases} CRD_{BestScheme} \geq CRD_{PreScheme}, \\ LTSS_{BestScheme} > LTSS_{PreScheme}. \end{cases} \tag{16}$$

The algorithm seeks feasible solutions through three methods in sequence as follows. An example of the three sub-methods is shown in Fig. 7.

Direct Insertion (DI): As shown in Fig. 7(a), we try to complete more tasks before the deadline by changing the order of task execution. For a task t_j that cannot meet the deadline, we first use a first-fit bin-packing to find an insertion position for t_j in the execution queue $PreScheme$ that meets the deadline. We define this insertion as $Index$. In fact, inserting t_j into a certain position of $PreScheme$ only affects the tasks arranged behind t_j and delays their completion time. Therefore, $PreScheme$

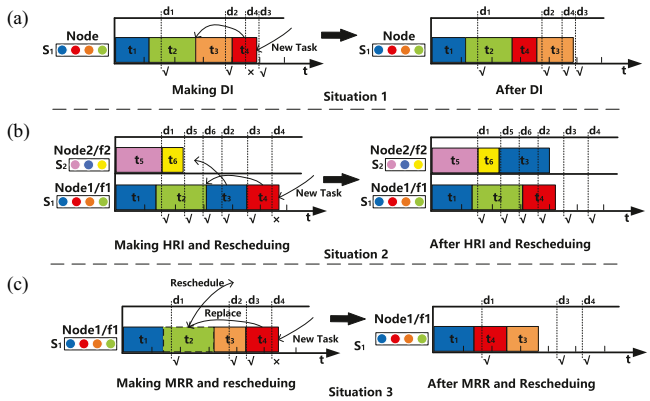

Fig. 7. The example of task adjusting.

can be divided into two parts: a) the affected queue $AQ(t_k)$ and b) the unaffected queue $NAQ(t_k)$. Here, t_k represents the tasks that have been arranged in the order in $PreScheme$. We assume that all tasks except t_j in $PreScheme$ can be completed before the deadline. Then, we calculate the remaining slack time of all tasks in $AQ(t_k)$. The remaining slack time is defined as

$$RST(t_k) = t_k^{deadline} - (ea_k + rp_k). \tag{17}$$

where, ea_k represents the earliest time to allocate resources for t_k and rp_k represents the estimated remaining execution time of t_k. As shown in (18), if the remaining execution time of t_j is less than the minimum remaining slack time of all tasks in $AQ(t_k)$, there exists an optimal scheduling solution for executing DI. It means that inserting a task into $PreScheme$ at position $Index$ only delays the completion time of tasks in $AQ(t_k)$ without violating their deadlines. This solution must also satisfy *Condition 1* and *Condition 2*. Finally, we update ea_k for all tasks in $AQ(t_k)$.

$$rp_j \leq \min\left\{RST_{t_k \in AQ(t_k)}(t_k)\right\}. \tag{18}$$

Highest Reward Insertion (HRI): If there is no feasible solution for DI that adheres to (18), we attempt to find a scheduling solution by adjusting the position of task execution. First, we calculate the number of tasks completed before the deadline and the total reward from $PreScheme$. Then, we try to insert t_j into the position $Index$ of $PreScheme$. We define $NewScheme$ to represent the current scheduling plan. Next, we calculate the number of tasks completed before the deadline and the total reward from $NewScheme$. This kind of insertion will inevitably lead to some tasks in $AQ(t_k)$ not being completed before the deadline. However, the load of computing nodes is in a dynamic process of change. It is possible that a node's resources are still occupied at the last moment, but are released at the next moment. Moreover, due to the constraint in (16), an optimization solution will result in at most only one task is affected. Thus, we can introduce a rescheduling mechanism to solve this problem. We use $AT^*(t_k)$ to represent the set of affected tasks in $AQ(t_k)$. We reschedule the affected task in $AQ(t_k)$ (Steps 7-15 in Algorithm 3). The algorithm pseudocode of task rescheduling is shown in

Algorithm 4: Task Rescheduling Algorithm.

Input: Task t_k^*, Resource Matrix R^*, pre-allocation $y_{h,\overline{k}}^j$
Output: New allocation pair $< t_k^*, x_{h,k'} >$
1: Initialize array of EFT, $AvaNodes$ as $\emptyset$
2: $AvaNodes \leftarrow$ append all nodes from X_h
3: Remove $x_{h,\overline{k}}$ from $AvaNodes$
4: **for** each $i \in AvaNodes$ **do**
5: $EFT[i] \leftarrow$ Calculate $EFT(AvaNodes[i], t_k^*)$
6: **end for**
7: $x_{h,k'} \leftarrow$ Find a node with $Min\{EFT[i]\}$
8: $TransTime \leftarrow$ Calculate the time from $x_{h,\overline{k}}$ to $x_{h,k'}$
9: **if** $EFT(x_{h,k'}, t_k^*) + TransTime \leq t_j^{deadline}$ **then**
10: Update allocation pair $< t_k^*, x_{h,k'} >$, $y_{h,k'}^j = 1$
11: $rp_k^* \leftarrow$ Calculate rp_k^* by by (20)
12: Rescheduling t_k^* to computing node $x_{h,k'}$
13: Update ea_k^* of t_k^*
14: **else**
15: $y_{h,k'}^{k^*} = 0$
16: Scheduling t_k^* to E_{cloud} by (4)
17: **end if**

Algorithm 4.

$$\begin{cases} rp_j > \min\left\{RLT_{t_k \in AQ(t_k)}(t_k)\right\}, \\ CRD_{NewScheme} \geq CRD_{PreScheme}, \\ LTSS_{NewScheme} > LTSS_{PreScheme}. \end{cases} \tag{19}$$

Note that regardless of whether the rescheduled task finds a feasible node, $NewScheme$ is still considered a better scheme because it satisfies both *Condition 1* and *Condition 2*. Thus, if $NewScheme$ is better than $PreScheme$, then it satisfies (19), and we have found an optimized solution. Besides, considering the heterogeneity of resources, the rescheduling of t_j from node $x_{h,k}$ to node $x_{h,k'}$ will lead to a change in execution time and generate a certain transmission delay. The change of rp_j is expressed as

$$rp_j = \frac{f_{h,k} * rp_j}{f_{h,k'}}. \tag{20}$$

An example of HRI is shown in Fig. 7(b). We can see that inserting t_j into a position of $Index$ results in a task not meeting the deadline in $AQ(t_k)$. However, the task still meets the deadline after being rescheduled to another node.

Maximum Reward Replacement (MRR): If there is no feasible solution for both DI and HRI, we attempt to replace an ongoing task or a task in $NAQ(t_k)$. Specifically, replacing a task t_k to satisfy the deadline of a new task must meet the *Condition 1*. Meanwhile, a feasible solution of MRR also needs to meet the *Condition 2* and another $Condition3$ as follows:

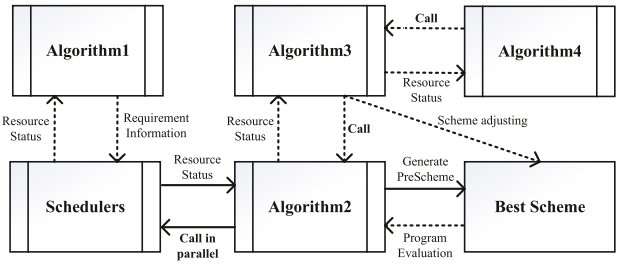

Fig. 8. The relationship diagram between algorithms.

Condition 3: $NewScheme$ achieves a lower average completion time (ACT).

$$\begin{cases} t_k \in NAQ(t_k) \\ ACT_{NewScheme} < ACT_{PreScheme}, \\ LTSS_{NewScheme} \geq LTSS_{PreScheme}, \\ CRD_{NewScheme} \geq CRD_{PreScheme}. \end{cases} \tag{21}$$

If $NewScheme$ meets (25), it is conducive to providing more possible optimizations for subsequent tasks, such as reducing the total completion time of subsequent tasks and facilitating the insertion of more small tasks. So it is also a solution that can optimize overall scheduling performance. Finally, it is worth noting that task replacement is preemptive. For the replaced task, we still try to reschedule it to meet its deadline as much as possible. An example of MRR is shown in Fig. 7(c).

D. Time Complexity Analysis

As shown in Fig. 8, the system first calls Algorithm 1 to match computing groups for multiple tasks and sort them before inserting them into their queues. In this process, computing group mapping can be implemented through key-value pairs, and the time complexity of searching and sorting n tasks does not exceed $O(n^2)$. Algorithm 2 searches for a suitable node among m nodes for task t_j. Its operations include searching for idle nodes and iterative calculations, and its time complexity does not exceed $O(n^2)$. Once the current resource load cannot meet the needs of t_j, the system will call Algorithm 3 to adjust the scheme. Algorithm 3 runs three sequential and mutually exclusive solution phases in sequence. The most complex operation is iterative traversal of the queue and search, and its time complexity does not exceed $O(n^2)$. Algorithm 4 is called by Algorithm 3. Its time complexity is mainly reflected in the iteration of node search and resource reallocation, and its time complexity does not exceed $O(n^2)$ at most. In summary, the time complexity of the entire strategy workflow is $O(n^2)$.

IV. Experimental Setup and Performance Evaluation

In this section, we implement a small prototype system to evaluate our proposed RATGS through a testbed and experimental simulations driven by a real production trace.

TABLE II
SETTING OF SYSTEM PROTOTYPE

Node Type	Configuration
User Device	Simulated by 8 task dispatchers
Access point	Raspberry Pi 4B@1.5GHz, 2GB Ryzen 7-6800H@3.2GHz, 16GB Intel i5-7200@2.5GHz, 12GB
Edge Server	Intel i5-12400@2.5GHz, 8GB Intel i9-12900H@2.5GHz, 16 GB-NVIDIA GeForce RTX 3060, 6GB

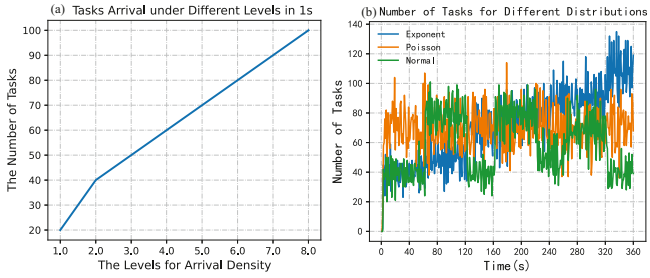

Fig. 9. Datasets of varying density and distribution.

A. Experimental Setup

1) Testbed Implementation: We develop D-ECCS, an edge-cloud collaborative scheduling system. This system was deployed on a testbed, which includes user devices, access points, edge servers, and a remote cloud. The edge device is simulated by eight *task dispatchers*. We use a Raspberry Pi 4B and a laptop with Ryzen 7-6800H (it also acts as an edge server) as access points and connect to three edge servers independently or not, as shown in Table II. To further constrain the heterogeneity of bandwidth, we use traffic control tools to set the network bandwidth between different machines to 100 Mbps$\sim$ 500Mbps. Besides, we built a three-node cloud cluster connected to the edge system. The system architecture is shown in Part A of Chapter II. To enable our system to capture the different expected execution times of tasks, we deployed a *node performance table* on each sub-scheduler to record the execution time of tasks on different nodes. By querying this table, the sub-scheduler can predict the expected execution of tasks. Note that the execution time of a task can be determined within a small error range based on its input metadata and hardware configuration. We use Kafka to decouple scheduling decisions and resource allocation.

2) Workloads: We selected several real applications written in Python to evaluate our strategy, such as image processing, ALU logic, float operation, face matching, etc. In addition, we conduct extensive simulations based on real traces from Alibaba [24] to fully evaluate our strategy. We set that task generation have different densities and distributions. Therefore, By modifying task arrival/end timestamps in the trace, we constructed two datasets. The first contains eight subsets with different task arrival densities in the range 1$\sim$8, as shown in Fig. 9(a). Each subset contains approximately 10,000 tasks. The second contains three subsets with different task arrival distributions, as shown in Fig. 9(b). Each subset contains approximately 25,000 tasks. Considering resource heterogeneity, the execution time range was calibrated from these datasets. Task execution times were then mapped to the average computing capacity of heterogeneous nodes, deriving the mean computational load per task. Task types were adapted from the trace metadata, with initial processing weights assigned based on dataset-provided priority levels, dynamically adjusted during scheduling. The remaining simulation evaluation parameters are shown in Table III.

TABLE III
PARAMETER SETTINGS

Parameters	Value
Task type	$1 \sim 8$
Initial processing weight factor for w_h	$1.0 \sim 2.0$
Execution time of real applications	$50 \sim 500$ ms
Computing amount of simulation tasks	$0.2 \sim 1$ TFLOPs
Uplink data size in simulation t_j^{data}	$625 \sim 2500$ KB
Deadlines for tasks in simulation	$450 \sim 1400$ ms
Bandwidth between edge servers in simulation $b_{i,j}$	100Mbps $\sim$ 300Mbps
Propagation delay between edge devices $\alpha_{j,i}^*$	$1 \sim 10$ ms
Baseline computing capacity of each f_i	$1.5 \sim 3.2$ TFLOPS/s

3) Evaluation Metrics and Baselines: We utilize four indicators: task completion rate before the deadline (CRD), the long-term satisfaction of system services (LTSS), the average completion time (ACT), and the resource utilization to evaluate our strategy.

We reproduced two scheduling strategies named $Dedas$ [9] and RTH^2S [25] (it is represented as RTH2S in subsequent analysis). $Dedas$ finds a suitable insertion position for the task by traversing the node's schedule in sequence. When a task cannot be inserted into a position that meets the deadline, the strategy considers the total completion time as a cost factor to replace a task to obtain the optimal schedule (Tasks that do not share link resources cannot be replaced). RTH2S ensures that tasks are completed before deadlines by layering heterogeneous nodes, prioritizing tasks using a three-level priority queue and EDF method, and splitting urgent tagging tasks. Besides, we construct multiple combined baseline strategies to effectively compare RATGS, including LSPT (LeastLoad + SRPT), LEDF (LeastLoad + EDF), and SF (Self + FCFS). The baseline *resource allocation strategies* include EDF (Earliest Deadline First), SRPT (Shortest Remaining Processing Time) [26], and FCFS (First Come First Served). The baseline *dispatching strategies* include *LeastLoad* (dispatch tasks to the edge server with shortest waiting queue) and *Self* (dispatch tasks to the node with earliest completion time).

B. Testbed Results

We evaluate the three indicators of CRD, LTSS, and ACT using real requests on the testbed. Each *task dispatcher* deploys a type of application respectively, offloading $0 \sim 1$ task at intervals of 50ms $\sim$ 100ms. Each request includes the metadata of task input, the expected deadline, and a flag indicating whether the scenario is expedited. Note that the service is always set to start in pre-warm mode. Fig. 10(a) shows the performance comparison. Compared with other strategies, RATGS increases the completion rate before deadline by 13.5% $\sim$ 50.1% and achieves the highest system benefit, which means that RATGS completes

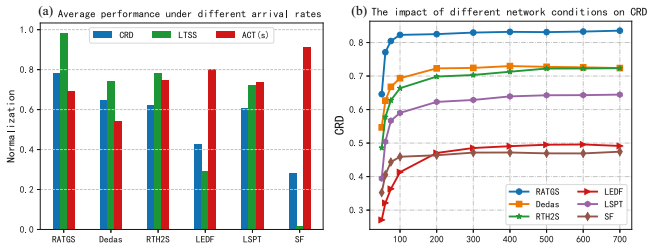

Fig. 10. Impact of testbed experiments.

more urgent tasks with the same resource consumption. In addition, to evaluate the reliability of strategies, we conducted strategy performance tests under different network conditions, as shown in Fig. 10(b). Under the same test conditions, when the bandwidth reaches a certain threshold, the performance of all strategies tends to be stable.

C. Simulation Results

In this section, we demonstrate the performance comparison of RATGS and multiple scheduling strategies under four indicators through the evaluation results.

Fig. 9(a).

1) Influence of Task Arrival Density: The arrival density of tasks affects the probability of resource competition between multiple tasks, thereby affecting the efficiency of scheduling strategies. The higher the density level, the more tasks arrive at the same time, as shown in Fig.

From Fig. 11, we can observe that before the task density is less than 4, the performance of all strategies on the metrics remains relatively stable. As the task density increases, the performance gap of the algorithm begins to manifest. RATGS outperforms Dedas, RTH2S, LSPT, LEDF, and SF by an average of 7.77%, 10.12%, 17.04%, 32.9%, and 22.62% in CRD respectively. In terms of LTSS, RATGS, Dedas, RTH2S, LSPT, LEDF, and SF achieve system benefits of 3969.6, 2869.5, 3235.3, 2486.9, 269.7, and 1491.3, respectively. In terms of ACT, RATGS, Dedas, RTH2S, LSPT, LEDF, and SF are 0.43, 0.37, 0.53, 0.51, 0.62, and 0.58, respectively.

The preemptive scheduling strategy obviously has a clear advantage. When the task density exceeds 4, the performance degradation rate of non-preemptive strategies is notably faster than that of preemptive strategies such as LEDF and SF. SRPT emerges as an excellent resource allocation strategy, capable of minimizing resource gaps and thus performing exceptionally well in scheduling small tasks. Consequently, even under resource-constrained conditions, LSPT maintains robust overall performance. RTH2S achieves higher system completion efficiency by aligning tasks of varying urgency levels with suitable resources; however, its approach of splitting urgent tasks leads to an increase in ACT. Meanwhile, Dedas outperforms all strategies in terms of ACT, likely due to its proactive discarding of "blocking" tasks, which effectively improves ACT.

2) Influence of Task Distribution: To more realistically reflect the performance impact brought about by changes in task arrival patterns, we conduct experiments based on three task distributions, as shown in Fig. 9(b).

Fig. 12 shows the comparison of CRD performance over time under three distributions. Our proposed RATGS continuously optimizes the scheduling scheme through the dynamic perception of resources and performs best in CRD indicators. Fig. 13 shows the overall results on CRD, LTSS and ACT indicators. RATGS outperforms Dedas, RTH2S, LSPT, LEDF, and SF by an average of 7.56%, 6.4%, 11.1%, 34.59%, and 33.89% in terms of CRD, respectively. In terms of LTSS, RATGS, Dedas, RTH2S, LSPT, LEDF, and SF achieve the average system benefits by 7731.9, 5831.8, 6462, 5515.6, -2150.5, and -1918.7, respectively. In terms of ACT, RATGS, Dedas, RTH2S, LSPT, LEDF, and SF are 0.51, 0.39, 0.59, 0.54, 0.69, and 0.65, respectively. RATGS lags behind Dedas ACT indicator but outperforms other strategies. This may be because RATGS sacrifices the ACT of tasks to meet more deadlines, which is an effective and meaningful trade-off. EDF is an intuitive offline non-preemptive single-machine scheduling algorithm that only considers deadlines. In contrast, the RTH2S strategy combines the EDF approach with preemption to achieve better performance. Besides, SF is a simple and stable, but it cannot capture the dynamic matching of resources and task requirements. In contrast, Dedas adapts to the state matching of resources and computing requirements through "out-of-order insertion" and replacement. However, Dedas always tries its best to traverse the search, which in turn limits the performance upper limit of scheduling.

3) Influence of Number of Edge Servers: The number of edge servers is another key factor affecting the quality of service and the performance of scheduling strategies. Fewer computing units may lead to a higher deadline miss rate, which can be expressed as 1-CRD. As computing units increases, tasks can be scheduled to more locations, which will ease resource competition and reduce the deadline miss rate. Thus, we add computing units to the system one by one to observe the performance changes of the strategy.

We can observe from Fig. 14 that the deadline missing rate of all scheduling strategies decreases as the number of computing units increases. In contrast, the deadline miss rate of our proposed RATGS decays the fastest. With 22 computing units, RATGS completes 99.9 percent of tasks before deadlines, while all other strategies require more computing units. There are two points worth noting here. First, SF also completed 99.9 percent of the tasks before deadlines when computing unit increased to 22. This is because as resources become abundant, the selfish scheduling strategy begins to show advantages. Second, the strategies with load-balancing, such as LSPT and LEDF, can only complete about 80 percent of the tasks before deadlines, regardless of the increase in computing units. The reason may be that resource heterogeneity causes a mismatch between the computational requirements of tasks and resource performance. This also proves that resource heterogeneity is a key factor affecting scheduling performance.

Figs. 15 and 16 show the performance comparison on LTSS and ACT. When the number of computing units reaches 30, the performance of each scheduling strategy tends to be stable, but when it is less than 22, the performance of RATGS is

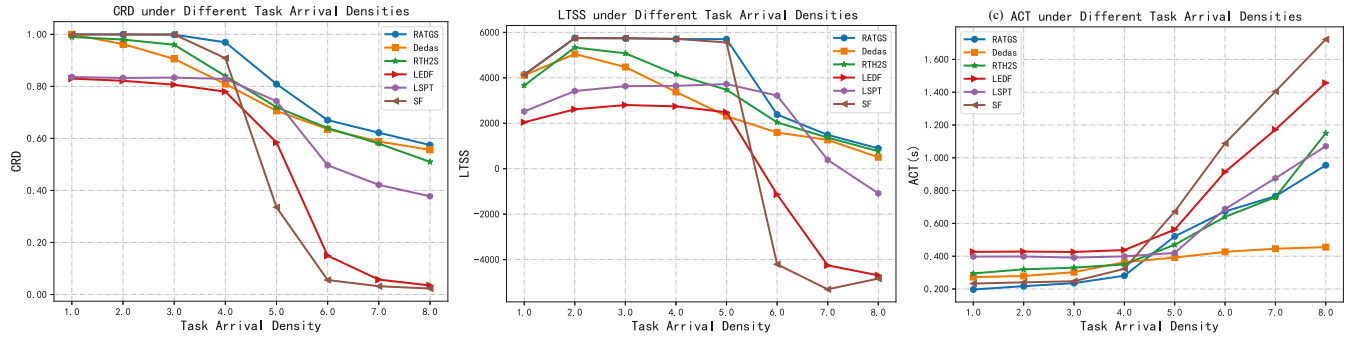

Fig. 11. Impact of task arrival dentisy.

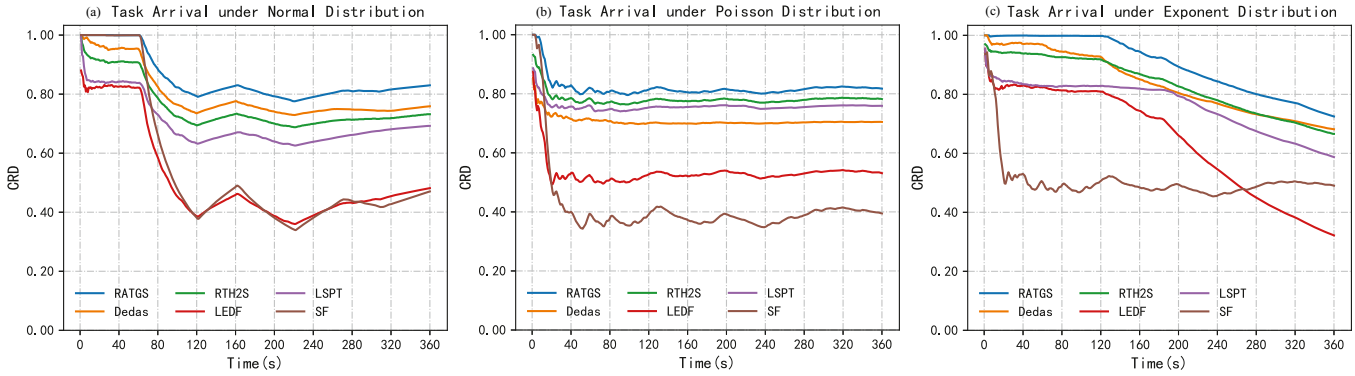

Fig. 12. Impact of task distribution on CRD over time.

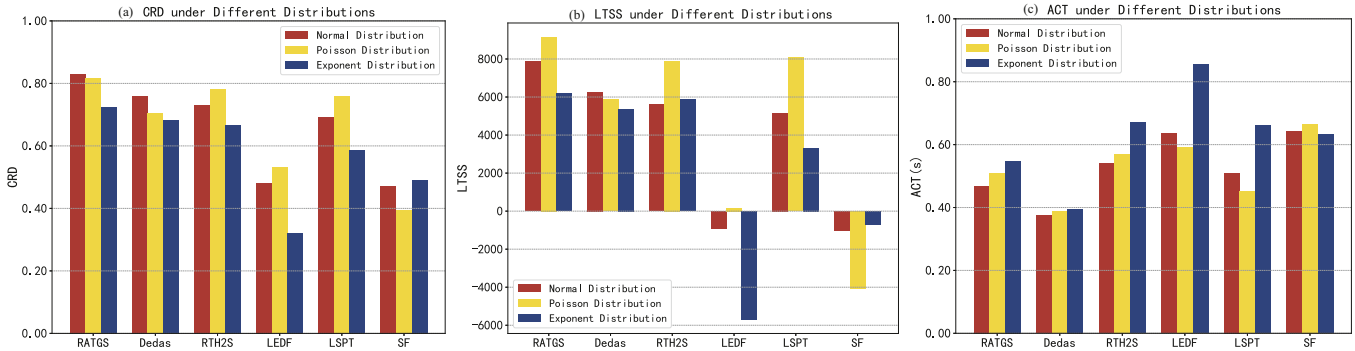

Fig. 13. Impact of task distribution.

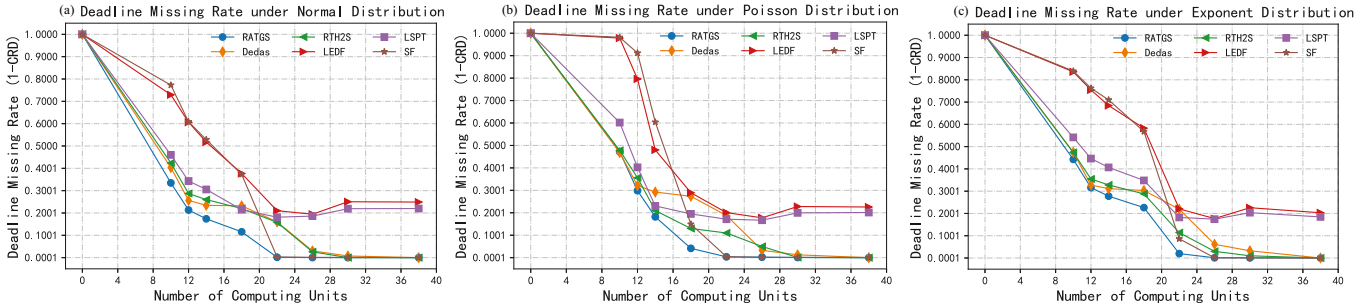

Fig. 14. Impact of computing units on deadline missing rate under three task distributions.

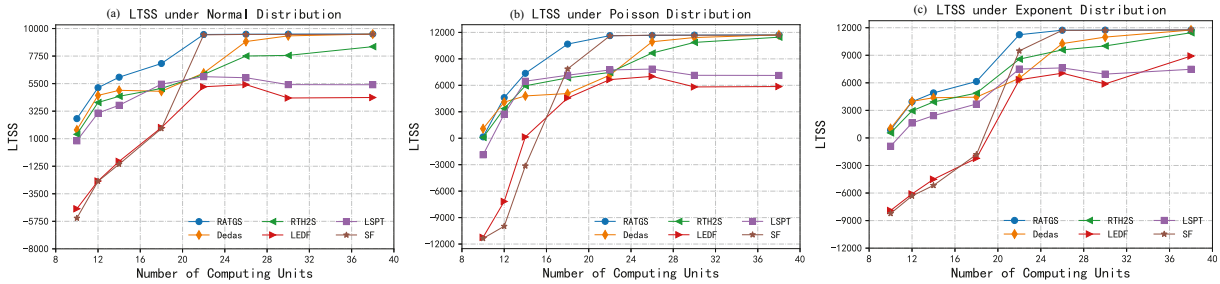

Fig. 15. Impact of computing units on LTSS under three task distributions.

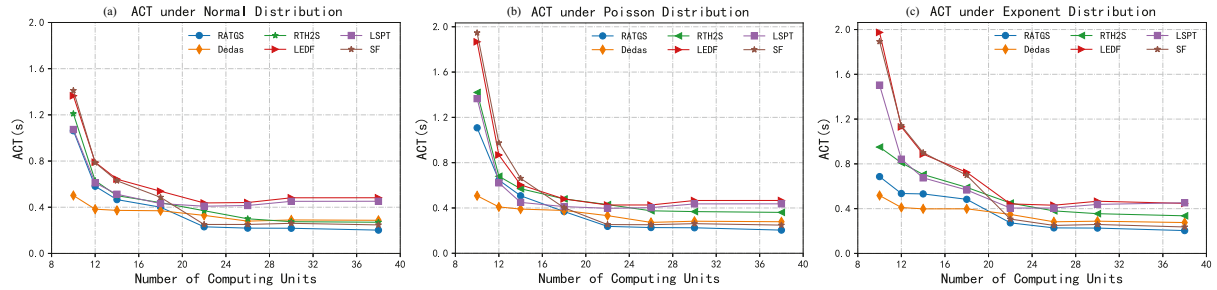

Fig. 16. Impact of computing units on ACT under three task distributions.

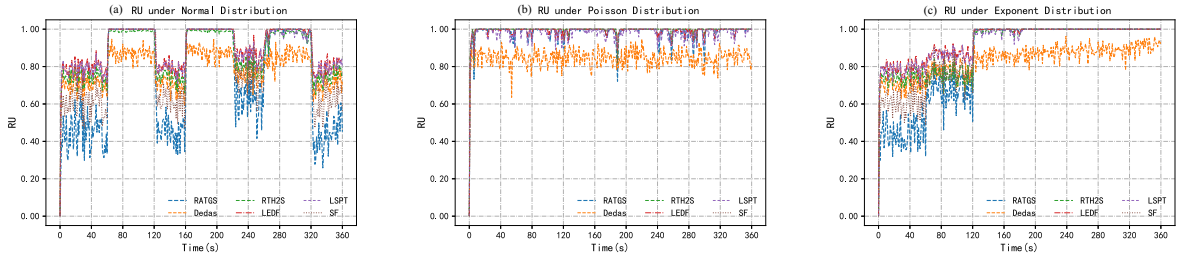

Fig. 17. Resource utilization rate under different distributions.

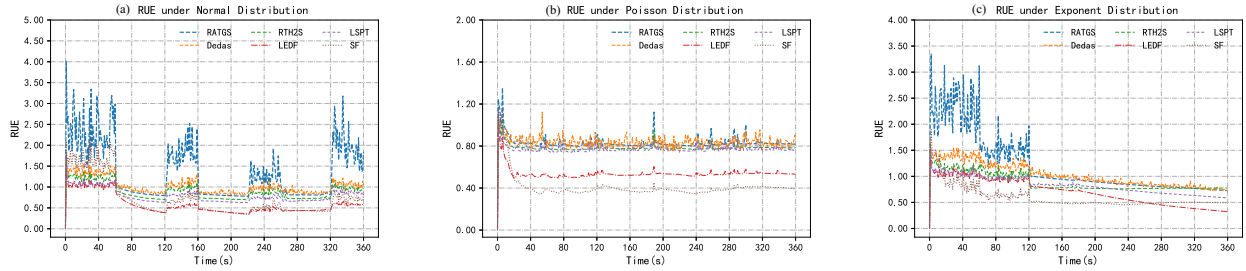

Fig. 18. Resource utilization efficiency under different distributions.

better than all strategies in LTSS indicator. Dedas still maintains overall optimality in ACT indicator. However, as the number of computing units exceeds 30 and gradually increases, RATGS begins to perform slightly better on ACT than Dedas. This is because the increase in computing units provides greater space and choices for task adjustment to achieve optimal scheduling.

4) Resource Utilization: Resource utilization rate (RU) is also a key indicator to reflect the performance of scheduling strategies in resource-constrained edge computing. We advocate using resource utilization efficiency (RUE) to evaluate the performance, which represents the number of tasks completed before the deadlines by a percentage unit of resource utilization, as shown in (22). The higher RUE indicates that the strategy is more efficient in utilizing limited resources.

$$RUE(\tau) = \frac{CRD(\tau)}{RU(\tau)}. \tag{22}$$

As shown in Figs. 17 and 18. load-balancing strategies (e.g., LSPT and LEDF) achieve the highest overall RU, maintaining an average rate exceeding 80% across all three task distributions.

RTH2S exhibits strong performance in RU and RUE due to its effective matching of heterogeneous tasks to resources. Notably, RATGS demonstrates near-unity RU during high task-arrival density periods but the lowest RU during low-density intervals (Fig. 17(a)–(c)). However, Fig. 18 reveals that RATGS achieves the highest RUE precisely in these contrasting phases such as $0 \sim 60$in Fig. 17(a), $180 \sim 190$in Fig. 17(b), and $80 \sim 120$in Fig. 17(c). It proves that RATGS effectively leverages limited resources to complete more tasks with less resource consumption.

V. Related Work

The optimization problem of edge task scheduling has been widely studied in recent years. These studies mainly focus on offloading from devices to edge servers, task dispatching among edge servers, and resource allocation on edge servers.

Task Dispatching and Scheduling in Edge. Han et al. first proposed an online task dispatching and scheduling algorithm OnDisc based on a general model [27]. Considering the fluctuations of the network, Meng et al. first considered the management of network resources and proposed an online dispatching and scheduling algorithm called Dedas to reduce the loss rate of deadlines [9]. Yuan et al. proposed an online task dispatching and fair scheduling algorithm OTDS [28] to cope with the dynamic characteristics of network and server loads. The uncertainty of request dispatching also reflects the dynamic characteristics of server load. To solve the "load evacuation" problem, Deng et al. proposed a staged task scheduling strategy [29]. Meanwhile, the utility function of task scheduling also shows diversity. Therefore, Zhang et al. proposed an online job dispatching and scheduling strategy O4A [30] under the heterogeneous utility coexistence. However, these studies usually set the processing time of tasks to a specific value without considering the dynamic computing requirements. The increasingly diverse computing requests also bring new challenges. Priority-based scheduling is an intuitive approach to deal with this problem. Peng et al. proposed a decentralized method DoSRA [31] that considers task diversity and priority. Besides, some research [32], [33] [34] have extensively studied the priority of task diversity in different scenarios such as vehicle edge computing (VEC) and unmanned aerial vehicles (UAVs), etc.

Heterogeneous Scheduling with Deadline Constraints. Zhu et al. considered a multi-user and multi-server scenario, jointly considered resource heterogeneity and task deadlines, and proposed two approximate algorithms LoPRTC and LoPRTC-MMD [35]. Kaur et al. considered the hierarchical heterogeneity of fog node resources and task deadlines and proposed a real-time heterogeneous hierarchical scheduling algorithm RTH^2S [25]. Azizi et al. proposed two semi-greedy strategy-based algorithms PSG and PSG-M to address the challenges of executing heterogeneous and delay-sensitive IoT tasks [36]. Gedawy et al. studied a cluster edge scheduling system RAMOS. The system adopts a multi-objective, resource-aware task allocation and scheduling strategy to minimize latency and improve energy efficiency [37]. Nevertheless, most of these studies adopt static resource allocation methods, without considering the impact of online resource competition between tasks on the performance of scheduling.

Resource-aware dynamic task scheduling. Considering the dynamic nature of computing demand, Ma et al. proposed a dynamic task scheduling algorithm WiDaS based on Lyapunov optimization technology [17]. Xu et al. studied the problem of dynamic computing power and task coordination scheduling between edge nodes and proposed an adaptive mechanism. The mechanism uses a greedy decision-making method to optimize task scheduling and dynamically adjust computing resources according to changes in user requests [18]. Zhao et al. proposed a three-stage iterative resource allocation strategy to address the challenge of multi-task competition for communication resources in edge computing [3]. However, they cannot guarantee the "optimal loss" when resource competition cannot be avoided and causes a loss of system service quality. Game theory may be an excellent solution to resource competition. Niu et al. described resource competition in heterogeneous edge computing as a non-cooperative stochastic game problem and proposed a multi-agent meta-PPO algorithm using meta-learning [38]. In addition, using deep reinforcement learning theory to optimize edge task scheduling is also a popular solution [39]. Han et al. designed a scheduling framework called Kais based on multi-agent deep reinforcement learning to improve the long-term throughput of request processing [40]. Wei et al. used a reinforcement learning algorithm to solve the joint optimization problem of resource placement and task scheduling in the dynamic state of edge servers to improve the QoS of mobile users or maximize the platform utility [41]. Tuli et al. proposed a real-time scheduler based on A3C to adaptively make dynamic decisions [42]. Nonetheless, most of these studies rely on offline models to guide scheduling decisions, which cannot timely reflect the actual dynamic changes in edge computing and the impact of the joint changes of resources on the scheduling strategy. Thus, they are only suitable for specific scenarios.

VI. Conclusion

Nowadays, edge computing has attracted more attention from researchers in various fields. A key challenge lies in enhancing the overall service quality of resource-constrained edge computing systems. Considering the more complex edge computing environment than cloud computing, we propose a resource-aware task grouping scheduling strategy (RATGS). This strategy considers resource competition among multiple tasks and dynamic changes in the computing environment, aiming to enhance the overall quality of service in edge systems. We conduct extensive comparative experiments. The results demonstrate the effectiveness of our proposed RATGS, especially in the resource-constrained edge computing environment. In particular, RATGS achieves higher efficiency of resource utilization rather than a simple resource utilization rate. Although our proposed RATGS is proven to be effective, our work also has some limitations. The main limitation is that our scheduling strategy currently does not consider the dependencies between tasks. The second limitation is that we ignore the energy consumption caused by hot starts of

service instances during task adjustment and the service delay caused by cold starts in actual environments.

ACKNOWLEDGMENT

A preliminary version of the article was presented in The 23th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2023) [11].

REFERENCES

[1] H. Chen, Y. Wen, Y. Huang, C. Xiao, and Z. Sui, "Edge computing enabling Internet of Ships: A survey on architectures, emerging applications, and challenges," *IEEE Internet of Things J.*, vol. 12, no. 2, pp. 1509–1528, Jan. 2025.

[2] X. Deng, J. Yin, P. Guan, N. N. Xiong, L. Zhang, and S. Mumtaz, "Intelligent delay-aware partial computing task offloading for multiuser industrial Internet of Things through edge computing," *IEEE Internet of Things J.*, vol. 10, no. 4, pp. 2954–2966, Feb. 2023.

[3] Z. Zhao, J. Shi, Z. Li, J. Si, P. Xiao, and R. Tafazolli, "Multiobjective resource allocation for mmWave MEC offloading under competition of communication and computing tasks," *IEEE Internet of Things J.*, vol. 9, no. 11, pp. 8707–8719, Jun. 2022.

[4] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 1–37, 2019.

[5] D. Lan et al., "Task partitioning and orchestration on heterogeneous edge platforms: The case of vision applications," *IEEE Internet of Things J.*, vol. 9, no. 10, pp. 7418–7432, May 2022.

[6] Z. J. K. Abadi, N. Mansouri, and M. Khalouie, "Task scheduling in fog environment—Challenges, tools & methodologies: A review," *Comput. Sci. Rev.*, vol. 48, 2023, Art. no. 100550.

[7] B. Wang, C. Wang, W. Huang, Y. Song, and X. Qin, "Security-aware task scheduling with deadline constraints on heterogeneous hybrid clouds," *J. Parallel Distrib. Comput.*, vol. 153, pp. 15–28, Jul. 2021.

[8] H. Zhou, Z. Zhang, D. Li, and Z. Su, "Joint optimization of computing offloading and service caching in edge computing-based smart grid," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1122–1132, Second Quarter, 2023.

[9] J. Meng, H. Tan, X.-Y. Li, Z. Han, and B. Li, "Online deadline-aware task dispatching and scheduling in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1270–1286, Jun. 2020.

[10] K. Gao, J. Song, C. Zhou, Z. Zheng, and S.-W. Jeon, "Task offloading and scheduling under hard deadlines in vehicular edge computing systems," *IEEE Trans. Veh. Technol*, vol. 74, no. 6, pp. 10029–10034, Jun. 2025.

[11] X. Tang, W. Cao, T. Deng, C. Xu, and Z. Zhu, "A grouping-based multi-task scheduling strategy with deadline constraint on heterogeneous edge computing," in *Proc. 23th Int. Conf. Algorithms Architect. Parallel Process.*, Springer, 2023, pp. 468–483.

[12] J. Li et al., "Maximizing user service satisfaction for delay-sensitive IoT applications in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 5, pp. 1199–1212, May 2022.

[13] C. Avasalcai, C. Tsigkanos, and S. Dustdar, "Resource management for latency-sensitive IoT applications with satisfiability," *IEEE Trans. Serv. Comput.*, vol. 15, no. 5, pp. 2982–2993, Sep./Oct. 2022.

[14] Y. Ren, S. Shen, Y. Ju, X. Wang, W. Wang, and V. C. Leung, "EdgeMatrix: A resources redefined edge-cloud system for prioritized services," in *Proc. 2022 IEEE Conf. Comput. Commun.*, 2022, pp. 610–619.

[15] A. Hazra, A. Kalita, and M. Gurusamy, "Meeting the requirements of Internet of Things: The promise of edge computing," *IEEE Internet of Things J.*, vol. 11, no. 5, pp. 7474–7498, Mar. 2024.

[16] Y. Laili, F. Guo, L. Ren, X. Li, Y. Li, and L. Zhang, "Parallel scheduling of large-scale tasks for industrial cloud–edge collaboration," *IEEE Internet of Things J.*, vol. 10, no. 4, pp. 3231–3242, Feb. 2023.

[17] X. Ma, A. Zhou, S. Zhang, Q. Li, A. X. Liu, and S. Wang, "Dynamic task scheduling in cloud-assisted mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 2116–2130, Apr. 2023.

[18] Y. Xu, L. Chen, Z. Lu, X. Du, J. Wu, and P. C. Hung, "An adaptive mechanism for dynamically collaborative computing power and task scheduling in edge environment," *IEEE Internet of Things J.*, vol. 10, no. 4, pp. 3118–3129, Feb. 2023.

[19] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: Flexible, scalable schedulers for large compute clusters," in *Proc. 8th ACM Eur. Conf. Comput. Syst.*, 2013, pp. 351–364.

[20] Q. Ma, Y. Qin, C. Zhu, L. Gao, and X. Chen, "Joint resource trading and task scheduling in edge-cloud computing networks," *IEEE Trans. Netw.*, vol. 33, no. 3, pp. 994–1008, Jun. 2025.

[21] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource scheduling in edge computing: A survey," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 4, pp. 2131–2165, Fourth Quarter 2021.

[22] J. Remy, "Resource constrained scheduling on multiple machines," *Inf. Process. Lett.*, vol. 91, no. 4, pp. 177–182, 2004.

[23] A. Mkaouar, S. Htiouech, and H. Chabchoub, "Solving the multiple choice multidimensional Knapsack problem with ABC algorithm," in *Proc. 2020 IEEE Congr. Evol. Computation*, 2020, pp. 1–6.

[24] Aliababa-clusterdata, 2023. [Online]. Available: https://github.com/alibaba/clusterdata

[25] A. Kaur, N. Auluck, and O. Rana, "Real-time scheduling on hierarchical heterogeneous fog networks," *IEEE Trans. Serv. Comput.*, vol. 16, no. 2, pp. 1358–1372, Mar./Apr. 2023.

[26] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 455–466, 2015.

[27] Z. Han, H. Tan, X.-Y. Li, S. H.-C. Jiang, Y. Li, and F. C. Lau, "OnDisc: Online latency-sensitive job dispatching and scheduling in heterogeneous edge-clouds," *IEEE/ACM Trans. Netw.*, vol. 27, no. 6, pp. 2472–2485, Dec. 2019.

[28] H. Yuan, G. Tang, X. Li, D. Guo, L. Luo, and X. Luo, "Online dispatching and fair scheduling of edge computing tasks: A learning-based approach," *IEEE Internet of Things J.*, vol. 8, no. 19, pp. 14985–14998, Oct. 2021.

[29] S. Deng, C. Zhang, C. Li, J. Yin, S. Dustdar, and A. Y. Zomaya, "Burst load evacuation based on dispatching and scheduling in distributed edge networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 8, pp. 1918–1932, Aug. 2021.

[30] C. Zhang et al., "Online dispatching and scheduling of jobs with heterogeneous utilities in edge computing," in *Proc. 21st Int. Symp. Theory Algorithmic Found. Protocol Des. Mobile Netw. Mobile Comput.*, 2020, pp. 101–110.

[31] Q. Peng, C. Wu, Y. Xia, Y. Ma, X. Wang, and N. Jiang, "DoSRA: A decentralized approach to online edge task scheduling and resource allocation," *IEEE Internet of Things J.*, vol. 9, no. 6, pp. 4677–4692, Mar. 2022.

[32] H. Hao, C. Xu, W. Zhang, S. Yang, and G.-M. Muntean, "Joint task offloading, resource allocation, and trajectory design for multi-UAV cooperative edge computing with task priority," *IEEE Trans. Mobile Comput.*, vol. 23, no. 9, pp. 8649–8663, Sep. 2024.

[33] X. Wang et al., "Augmented intelligence of things for priority-aware task offloading in vehicular edge computing," *IEEE Internet of Things J.*, vol. 11, no. 22, pp. 36002–36013, Nov. 2024.

[34] N. Qiao, S. Yue, Y. Zhang, and J. Ren, "PoPeC: PAoI-centric task offloading with priority over unreliable channels," *IEEE/ACM Trans. Netw.*, vol. 32, no. 3, pp. 2376–2390, Jun. 2024.

[35] T. Zhu, T. Shi, J. Li, Z. Cai, and X. Zhou, "Task scheduling in deadline-aware mobile edge computing systems," *IEEE Internet of Things J.*, vol. 6, no. 3, pp. 4854–4866, Jun. 2019.

[36] S. Azizi, M. Shojafar, J. Abawajy, and R. Buyya, "Deadline-aware and energy-efficient IoT task scheduling in fog computing systems: A semi-greedy approach," *J. Netw. Comput. Appl.*, vol. 201, no. 1, pp. 1–13, 2022.

[37] H. Gedawy, K. Habak, K. A. Harras, and M. Hamdi, "RAMOS: A resource-aware multi-objective system for edge computing," *IEEE Trans. Mobile Comput.*, vol. 20, no. 8, pp. 2654–2670, Aug. 2021.

[38] L. Niu et al., "Multi-agent meta-reinforcement learning for optimized task scheduling in heterogeneous edge computing systems," *IEEE Internet of Things J.*, vol. 10, no. 12, pp. 10519–10531, Jun. 2023.

[39] P. Li, Z. Xiao, H. Gao, X. Wang, and Y. Wang, "Reinforcement learning based edge-end collaboration for multi-task scheduling in 6G enabled intelligent autonomous transport systems," *IEEE Trans. Intell. Transp. Syst.*, to be published, doi: 10.1109/TITS.2024.3525356.

[40] Y. Han, S. Shen, X. Wang, S. Wang, and V. C. Leung, "Tailored learning-based scheduling for Kubernetes-oriented edge-cloud system," in *Proc. 2021 IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.

[41] X. Wei, A. M. Rahman, D. Cheng, and Y. Wang, "Joint optimization across timescales: Resource placement and task dispatching in edge clouds," *IEEE Trans. Cloud Comput.*, vol. 11, no. 1, pp. 730–744, First Quarter, 2023.

[42] S. Tuli, S. Ilager, K. Ramamohanarao, and R. Buyya, "Dynamic scheduling for stochastic edge-cloud computing environments using A3C learning and residual recurrent neural networks," *IEEE Trans. Mobile Comput.*, vol. 21, no. 3, pp. 940–954, Mar. 2022.