Joint Data Placement and Service Deployment in Distributed Cloud-Edge Environment

Pengwei Wang, *Member, IEEE*, Jingtan Jia, Chao Fang, Guobing Zou, *Member, IEEE*, and Zhijun Ding, *Senior Member, IEEE*

***Abstract*—How to efficiently deploying the service components of a data-intensive application on cloud and edge servers to minimize its latency is one of the main challenges for service providers. Most existing studies consider either service deployment or data placement, rather than their joint optimization. This work considers the driving relationship between data and services in a heterogeneous environment including remote cloud and nearby edge servers, and aims to obtain a desired data placement and service deployment scheme while meeting user requirements for service quality. First, we formulate the problem and decouple data placement from service deployment by polynomial reduction. Then, a priority-based data placement strategy is proposed, which can generate a data placement scheme. After that, the original problem is transformed into a classical assignment problem, and a service deployment strategy based on an improved Hungarian algorithm is proposed to obtain a service deployment scheme. Then, a dynamic adjustment strategy based on response weight is proposed to dynamically adjust the data placement and service deployment scheme in order to reduce response latency, and obtain the final scheme. Finally, a series of comparative experiments were conducted, pitting our algorithms against several baseline and SOTA algorithms. The results show that the proposed algorithms, in comparison to other algorithms, is capable of generating superior data placement and service deployment schemes to significantly reduce response latency.**

***Index Terms*—Cloud computing, edge computing, data placement, service deployment, Hungarian algorithm.**

I. Introduction

SINCE the early 2000s, cloud computing has emerged as a new frontier of software and application delivery, rapidly moving beyond traditional inner systems to become reliable, scalable, and cost-effective IT solution. However, with the increase of IoT devices at the network edge, the scale of data to be processed by data centers is increasing, which pushes the network bandwidth requirements to the limit. Thus, edge computing emerged by aiming to move computing from data centers to the edge of a network [1], utilizing smart objects, mobile phones or network gateways to perform tasks and provide services on behalf of cloud. By moving services to the edge, content caching, service delivery, service storage, and IoT management can be provided, resulting in faster response and data transfer. But distributing logic among different network nodes brings new problems and challenges.

In addition to cloud computing and edge computing, a series of applications supported by the Internet have also developed by leaps and bounds. In order to be highly reliable and scalable, modern application software adopts a service-oriented architecture [2], [3]. As users have higher and higher requirements for Quality-of-Service (QoS) of their applications, the functions of service components that constitute applications become more and more complex. Data storage and computing capabilities of service components themselves are also increasing. Moreover, there may be enforced associations and temporal sequences among them [4]. In edge and cloud computing environments, application providers can rent computing and storage resources to deploy their applications, thereby providing highly-available and low-latency services to their application users [5].

Existing studies related to service deployment often consider a single aspect, but rarely consider data placement and service deployment jointly. However, in the real world, there are many types of applications, such as computing-intensive applications, and data-intensive ones [6]. For the latter [7], most service components are driven by data, which means that simply considering service deployment separately in heterogeneous networks is not the best decision. In addition, the integration of cloud and edge resources [8] can combine the powerful storage and computing capability of cloud computing and the low latency processing capability of edge computing. Their collaboration enables them to adapt to broader application scenarios and exert more powerful functions.

In this work, we study the problem of deploying data-intensive applications in a cloud-edge collaborative environment in conjunction with the driving relationship between data and services, and combining data placement and service deployment, aiming to reduce the response latency of application. We aim to make the following new contributions to the field of cloud/edge computing.

- We propose a priority-based data placement strategy (PDPS) that optimizes data placement by considering the demands of services for data and the proximity of edge resources.
- By transforming the original problem into a classical assignment problem, we propose a service deployment strategy based on an improved Hungarian algorithm (HA-SDS) to obtain a service deployment scheme.

Received 18 July 2024; revised 10 May 2025; accepted 5 June 2025. Date of publication 4 July 2025; date of current version 8 August 2025. This work was partially supported in part by Ant Group through CCF-Ant Research Fund and in part by the National Natural Science Foundation of China (NSFC) under Grant 61602109. *(Jingtan Jia and Chao Fang contributed equally to this work.) (Corresponding author: Pengwei Wang.)*

Pengwei Wang, Jingtan Jia, and Chao Fang are with the School of Computer Science and Technology, Donghua University, Shanghai 201620, China (e-mail: wangpengwei@dhu.edu.cn; xjtx0523@gmail.com; zihuanxue2@gmail.com).

Guobing Zou is with the School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China (e-mail: gbzou@shu.edu.cn).

Zhijun Ding is with the School of Computer Science and Technology, Tongji University, Shanghai 201804, China (e-mail: dingzj@tongji.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TSC.2025.3586092, provided by the authors.

Digital Object Identifier 10.1109/TSC.2025.3586092

- A dynamic adjustment strategy based on response weight (RW-DAS) is introduced to dynamically adjust the data placement and service deployment scheme jointly in order to reduce response latency.
- By leveraging the driving relationship between data and services, we jointly optimize data placement and service deployment, effectively reducing the response latency of data-intensive applications in cloud-edge collaborative environments.

Next section briefs the related work. Section III presents our research motivation and the current challenges associated with this problem. Section IV gives system models and problem formulation. Then, our proposed algorithms are presented in Section V. Their effectiveness are verified through extensive experiments in Section VI. Finally, Section VII concludes this work.

II. Related Work

Since most edge servers have limited resources, it is likely to increase data processing latency, if reasonable data storage is not provided. Therefore, Xia et al. [9] presented a Lyapunov optimization-based method to solve the problem of collaborative edge data caching. Xia et al. [10] proposed an approximation algorithm, in order to solve the EDD problem. The studies consider cloud data centers and edge environments jointly to solve the related problems of data caching. A method is proposed to cache data based on information entropy theory [11]. A balanced placement algorithm is proposed based graph partition [12]. Multi-cloud storage is a framework for cloud storage. Wang et al. [13] presented an architecture for multi-cloud storage in order to reduce cost and improve availability. Similarly, Wang et al. [14] proposed an adaptive architecture for multi-cloud data placement, in order to solve the challenge of dynamically storing users' data according to time-varying access patterns. Liu et al. [15] proposed a genetic algorithm-based method to select cloud instances among multiple clouds. Cao et al. [16] presented an algorithm based on NSGA-II and a multi-group strategy, so as to help users select suitable services to store their data in multi-cloud and edge environments. Wang et al. [17] proposed a data temperature model to explore the spatial and temporal attributes of data and their changing trend, and presented a data temperature-based multi-cloud storage strategy. Cost and makespan are the most concerned issues for workflow scheduling [18], [19], [20]. Wang et al. [21] proposed an immune-based PSO algorithm to implement makespan-driven workflow scheduling among multiple clouds. Regarding the challenge of multi-cloud placement of data copies, Chikhaoui et al. [22] improved NSGA-II to lower execution time.

In an edge computing environment, service providers can deploy their application instances on edge servers to provide low latency service. Li et al. [23] explored an integrated manner of optimizing both partitioning and replication without distinguishing replica's roles. Then, they proposed LDP to conduct the optimization of replica placement. In the highly distributed, dynamic and volatile edge environment, the robustness of services deployed on edge servers has been largely ignored. Chen et al. [24] provided an integer programming-based approach to solve it, and an approximate algorithm to efficiently find near-optimal solutions for large-scale problems. To ensure the system's performance, there are at least two major challenges to cope with: 1) how to offload the training jobs with multiple data source nodes, and 2) how to allocate the limited resources on each edge server among training jobs. Wang et al. [25]

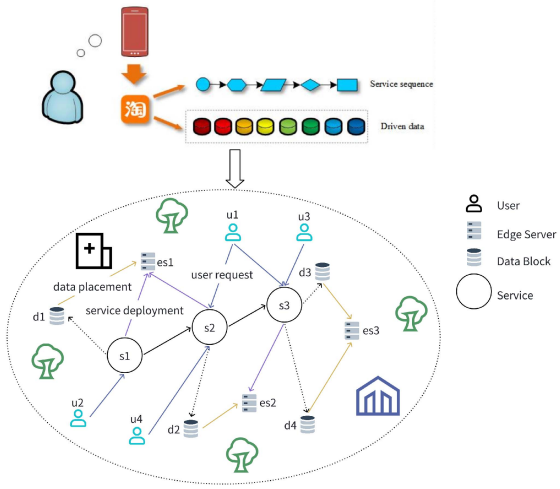


Fig. 1. A typical example of data placement and service deployment of the data-intensive application.

combined edge computing and deep learning, proposing an efficient approximate algorithm based on reformulation and randomized rounding technique. Wu et al. [26] proposed a general parallel discovery strategy to improve the performance of parallelization for single-node algorithms. Xu et al. [27] and Lei et al. [28] both considered Lyapunov optimization algorithm. The former proposed OREO for mobile edge computing. The latter transformed the long-term optimization problem into a series of real-time optimization problems that only require of the current time slot information. Considering user mobility and network load balance, Wang et al. [29], [30] proposed a base station selection algorithm based on user mobility prediction, which included a user mobility prediction algorithm and a selection strategy. Each microservice decoupled from the application can be packaged into a docker image, and each microservice instance is a docker container.

For data-intensive applications, data placement and service deployment need to be jointly considered. Wang et al. [31] analyzed the geographical characteristics of cloud centers with the help of a clustering algorithm, and proposed a multi-cloud data placement initialization strategy for spatial crowdsourcing tasks. Wei and Wang [32] proposed a popularity-based data placement method by mapping both data items and edge servers to a virtual plane and performing data placement and retrieval based on their virtual coordinates, aiming to minimize data access latency and balance the load. Jin et al. [33] optimized the distribution of data in edge storage systems by combining erasure coding technology, while also considering data access frequency and node load to ensure efficient data access performance while reducing storage overhead. Bahreini et al. [4] proposed a data-intensive service edge deployment scheme based on genetic algorithm to minimize the response time under storage constraints and load balancing conditions. Chen et al. [7] solved a multi-component application placement problem in edge computing environment by designing an efficient heuristic online algorithm, and proposed a mixed integer linear programming formulation. Hao et al. [34] achieved a more efficient service placement strategy in complex industrial CPS environments by combining deep reinforcement learning techniques with traditional optimization methods. Fan et al. [35] utilizes the Lagrangian relaxation method to optimize whether each base station (BS) should deploy specific types of services,

combined with resource allocation on edge and cloud servers. Peng et al. [36] proposed a service deployment method based on a submodular optimization algorithm, which iteratively adjusts the placement of microservices and their request routing paths to minimize resource consumption and latency.

However, the above-mentioned studies have not well considered the driving relationship between data and services. They only considered the service deployment while ignoring the related data placement issue; or they focused solely on data placement without considering the corresponding service deployment needs. Such a one-sided approach may lead to suboptimal overall performance. Therefore, this work focuses on data-intensive applications that combine the driving relationship between data and services in a cloud-edge environment so as to reduce the response latency of application requests.

III. Motivation

In the era of distributed cloud-edge computing, data placement and service deployment have become cornerstones of efficient application operation. Data placement involves determining where to store application data across cloud and edge servers, while service deployment focuses on allocating computational tasks to the most suitable nodes. These two processes are not independent; instead, they are deeply interrelated and play a critical role in ensuring low latency, optimal resource utilization, and high-quality service delivery.

A microservices-based application consists of multiple microservices, each of which can be executed by several available candidate edge servers. Taking an e-commerce application as an example: when we shop on a client browser, we first search for the desired products through various site search services; then add the items to the shopping cart and proceed with payment, which can be completed by calling services from Alipay, WeChat Pay, or PayPal. Afterward, we can review and rate the purchased items. In modern e-commerce platforms, personalized recommendations, inventory management, and facial recognition payments are three core functional modules that rely on efficient interaction between data and services to provide an excellent user experience. Fig. 1 above illustrates the relationships among services (s_1, s_2, s_3), data blocks (d_1, d_2, d_3, d_4), and edge servers (es_1, es_2, es_3). The personalized recommendation service (s_1) analyzes user behavior data (such as browsing history and purchase records) to predict user preferences and recommend products they might be interested in. In addition, in order to achieve better recommendation results, the system may require additional user contextual data or social data (d_1). Inventory management service (s_2) is responsible for tracking and updating the stock status of products. It handles transaction history data (d_2), which includes product details and sales information for each order. Effective inventory data management enables timely stock replenishment, preventing out-of-stock situations while also avoiding excessive inventory tying up funds. For large-scale e-commerce platforms, inventory data is typically distributed across multiple warehouses and geographic locations. Facial recognition payment service (s_3) involves face database data (d_3) and payment-related data (d_4). Face database data is used for user identification, whereas payment-related data contains specific payment information. Given the sensitivity of this data, it cannot be directly coupled with the services. This architecture design not only enhances system flexibility and scalability but also provides users with a more secure and seamless shopping experience.

These services are highly interdependent and rely on data distributed across various locations. If data placement and service deployment are not properly aligned, this can lead to latency in recommendation generation or transaction processing, directly impacting the overall user experience. Given the separation of services and data, along with the frequent need for data access by services, the joint optimization of data placement and service deployment becomes extremely important. However, the above-mentioned studies have not well considered and combined the driving relationship between data and services for application deployment, and optimize the service deployment and data placement jointly.

Therefore, this work focuses on data-intensive applications that combine the driving relationship between data and services in a cloud-edge environment so as to reduce the response latency of application requests. In the preliminary conference version [37], we formulated the service deployment and data placement problem and decoupled the original problem to obtain separate solutions, but not the joint optimization of them. Moreover, in the process of decoupling, we simplified the original problem and ignored some details, which leaves room for further optimization of the results. Therefore, in this extended work, we further propose a dynamic adjustment strategy based on response weight to iteratively optimize the results of the previous work, aiming to obtain a better solution.

IV. Basic Definitions and Problem Formulation

A. System Model

1) Heterogeneous Network Model: A heterogeneous network contains a remote cloud C and an edge cloud $E = \{e_k\}_{k=1}^{H}$ (a set of H edge servers). It is represented as a DAG graph $G(\mathbb{V}, \mathbb{E})$, where $\mathbb{V} = \{\mathbb{v}_1, \mathbb{v}_2, \ldots, \mathbb{v}_{H+1}\}$, and $\mathbb{E} = \{\mathbb{e}_1, \mathbb{e}_2, \ldots, \mathbb{e}_g, \mathbb{e}_{g+1}, \cdots \mathbb{e}_{g+H}\}$. Vertices $\mathbb{v}_1 \sim \mathbb{v}_H$ represent H edge servers and vertex $\mathbb{v}_{H+1}$ represents the remote cloud. Edges $\mathbb{e}_1 \sim \mathbb{e}_g$ represent totally g edges among edge servers, and $\mathbb{e}_{g+1} \sim \mathbb{e}_{g+H}$ represent H edges between the remote cloud and edge servers.

For each edge server e_k, it has index k, storage capacity o_k, computing capacity p_k $(unit/ms)$, memory capacity m_k and geographic coordinates $\mathbb{z}_\mathrm{k} = (\mathbb{x}_\mathrm{k}, \mathbb{y}_\mathrm{k})$. Similarly, for the remote cloud C, it has storage capacity o^C, computing capacity p^C $(unit/ms)$, memory capacity m^C, and geographic coordinates $\mathbb{z}^\mathrm{C} = (\mathbb{x}^\mathrm{C}, \mathbb{y}^\mathrm{C})$. Then, for any e_k, we have the following constraints: $o^C \gg o_k, p^C \gg p_k, m^C \gg m_k$

2) Application Model: An application A is composed of M services that have different functions and uses N different data blocks. Among them, each data block and service has an index. The execution of each service should be in order, and may be driven by some data blocks. We suppose that a data block can only drive one service, but a service may be driven by several (including zero) different data blocks. Therefore, we have $A = (D, S, R, F)$, where the specific meanings of each component are as follows.

1) $D = \{d_i\}_{i=1}^{N}$ denotes a set of N different data blocks. Each data block d_i has a quadruple (i, d_i^o, d_i^s, d_i^e), where i is its index, d_i^o is its size, d_i^s is the service which is driven by d_i, and d_i^e represents the edge server where d_i is placed. Next, two binary variables r_{ij} and x_{ik} are introduced.

$$r_{ij} = \begin{cases} 1, & \text{service } s_j \text{ is driven by data block } d_i \\ 0, & \text{other} \end{cases} \quad (1)$$

$$x_{ik} = \begin{cases} 1, & \text{data block } d_i \text{ is placed on edge server } e_k \\ 0, & \text{other} \end{cases} \tag{2}$$

The symbol "| |" is defined to obtain the index. If $d_i^s = s_j$ (if and only if $r_{ij} = 1$), then $|d_i^s| = j$. Similarly, if $d_i^e = e_k$ (if and only if $x_{ik} = 1$), then $|d_i^e| = k$. To sum up, a data placement scheme can be represented as an $N \times H$ matrix $X = [x_{ik}]_{i=1,k=1}^{i=N,k=H}$, and each data block corresponds to a row vector $X_i = [x_{ik}]_{k=1}^{k=H}$ ($\sum_{k=1}^{H} x_{ik} = 1$).

2) $S = \{s_j\}_{j=1}^{M}$ denotes a set of M services that have different functions. Each service s_j has a quintuple $(j, s_j^m, s_j^p, s_j^d, s_j^e)$, where j is the index, s_j^m represents the memory required by s_j, s_j^p $(unit)$ represents the number of instructions that s_j contains, s_j^d represents the set of data blocks that can drive s_j, and s_j^e is the edge server where s_j is deployed. Combining the binary variable r_{ij}, if $s_j^d = \{d_i | r_{ij} = 1\}$, then $|s_j^d| = \{i | r_{ij} = 1\}$. Another binary variable y_{jk} is introduced. If $s_j^e = e_k$ (if and only if $y_{jk} = 1$), then $|s_j^e| = k$.

$$y_{jk} = \begin{cases} 1, & \text{service } s_j \text{ is deployed on the server } e_k \\ 0, & \text{other} \end{cases} \tag{3}$$

The symbol "‖ ‖" is defined for counting. When $\|s_j^d\| = \varepsilon$, it means that service s_j is driven by ε data blocks, and $\sum_{i=1}^{N} r_{ij} = \varepsilon$. To sum up, a service deployment scheme can be represented as an $M \times H$ matrix $Y = [y_{jk}]_{j=1,k=1}^{j=M,k=H}$, and each service corresponds to a row vector $Y_j = [y_{jk}]_{k=1}^{k=H}$ ($\sum_{k=1}^{H} y_{jk} = 1$).

3) $R = [r_{ij}]_{i=1,j=1}^{i=N,j=M}$ represents the driving relationship between data and service, which is an $N \times M$ matrix. r_{ij} is the same as that in (1). Each data block d_i corresponds to a row vector $R_i = [r_{ij}]_{j=1}^{M}$ ($\sum_{j=1}^{M} r_{ij} = 1$), which represents the driving relationship between data block d_i and all services. Similarly, each service s_j corresponds to a column vector $R_j = [r_{ij}]_{i=1}^{N}$ ($\sum_{i=1}^{N} r_{ij} = \|s_j^d\| \geq 0$), which represents the driving relationship between service s_j and all data blocks.

4) $F = \{\langle s_i, s_j \rangle \mid i, j \in [1, M]\}$ represents the execution order between services. $\langle s_i, s_j \rangle$ means that s_j can be executed after s_i has been finished. Symbol $\overrightarrow{f}$ is used to record the execution order of all services ($\overrightarrow{f_0}$ is 'null'). $\overrightarrow{f_\varphi} = s_j$ ($j \in [1, M], \varphi \in [1, M]$) means that s_j is the φth service to be executed, and $|\overrightarrow{f_\varphi}| = j$, $|s_j| = \varphi$.

3) Response Latency: To simplify the model, the response latency $\mathbb{T}$ of a data-intensive application in heterogeneous network is defined as the period from the execution of its first service to the completion of its last service. Regarding each service, it contains three parts: *request data*, *service execution*, and *output result*. The user's location is not considered in this paper, and thus we can ignore the transmission delay between a user and application.

Assumption 1. (Service execution conditions): There are two preconditions for service s_j to be executed.

- The previous service $s_{j'} = \overrightarrow{f_{|s_j|-1}}$ ($|s_j| > 1$) has already finished and transmitted the result to service s_j (if $|s_j| = 1$, this constraint can be ignored).
- All of $\|s_j^d\|$ data blocks in set s_j^d have been successfully requested and obtained.

Assumption 2. (Data request mechanism): The data request mechanism for service $s_j = (j, s_j^m, s_j^p, s_j^d, s_j^e)$ $(\|s_j^d\| = \varepsilon \geqslant 0) \wedge (s_j^e = e_k)$ is as follows:

1) When all the ε data blocks required by service s_j are placed on the same edge server e_k, s_j can be executed directly, and the data transmission time can be ignored.
2) If a required data block d_i $(r_{ij} = 1)$ is not placed on edge server e_k, we find the nearest edge server $e_{k'}$ to check whether this data block exists, and transmit it if so.
3) If the required data block is neither placed on the same edge server e_k nor on the nearest edge server $e_{k'}$, the required data block is requested directly from remote cloud C.

In summary, the response latency $\mathbb{T}$ can be defined as:

$$\mathbb{T} = T^{req} + T^{exec} + T^{out} \tag{4}$$

T^{req} is the sum of network latency for all services in the data request phase, i.e.,

$$T^{req} = \sum\nolimits_{j=1}^{M} t_j^{req} \tag{5}$$

t_j^{req} represents the network latency of s_j requesting the required data blocks, which includes propagation delay and transmission delay, i.e.,

$$t_j^{req} = \sum\nolimits_{i \in |s_j^d|} \left(t_{ij}^{pro} + t_{ij}^{trans}\right) \tag{6}$$

t_{ij}^{pro} represents the propagation delay between edge servers s_j^e and d_i^e. Let λ denotes the propagation rate between edge servers, and λ^C denotes the propagation rate between edge server and remote cloud. Then, We use euclidean distance to represent the distance between two servers: $l(\S, \dagger) = \sqrt{(\S.lat - \dagger.lat)^2 + (\S.long - \dagger.long)^2}$. We have:

$$t_{ij}^{pro} = \begin{cases} \frac{l(\mathbb{z}_{|d_i^e|}, \mathbb{z}_{|s_j^e|})}{\lambda}, & \text{service } s_j \text{ requests data } d_i \text{ from edge, and } |d_i^e| \neq |s_j^e| \\ 0, & \text{service } s_j \text{ requests data } d_i \text{ from edge, and } |d_i^e| = |s_j^e| \\ \frac{l(\mathbb{z}^C, \mathbb{z}_{|s_j^e|})}{\lambda^C}, & \text{service } s_j \text{ requests data } d_i \text{ from the remote cloud} \end{cases} \tag{7}$$

t_{ij}^{trans} is the transmission delay between s_j^e and d_i^e. Let b denotes the network bandwidth between edge servers, and b^C denotes the network bandwidth between edge server and remote cloud. We have:

$$t_{ij}^{trans} = \begin{cases} \frac{d_i^o}{b}, & \text{service } s_j \text{ requests data } d_i \text{ from edge servers, and } |d_i^e| \neq |s_j^e| \\ 0, & \text{service } s_j \text{ requests data } d_i \text{ from edge servers, and } |d_i^e| = |s_j^e| \\ \frac{d_i^o}{b^C}, & \text{service } s_j \text{ requests data } d_i \text{ from the remote cloud} \end{cases} \tag{8}$$

T^{exec} represents the sum of latency for all services in service execution phase, i.e.,

$$T^{exec} = \sum\nolimits_{j=1}^{M} t_j^{exec} \tag{9}$$

t_j^{exec} is the execution time of the service s_j that is deployed on edge server $e_k = s_j^e$ $(y_{jk} = 1)$, i.e.,

$$t_j^{exec} = \frac{s_j^p}{p_{|s_j^e|}} \tag{10}$$

T^{out} represents the sum of latency for all services in output result phase, i.e.,

$$T^{out} = \sum_{\varphi=1}^{M-1} t_{jj'}^{out} \tag{11}$$

$t_{jj'}^{out}$ represents the network latency of s_j outputting and transmitting its result, which includes propagation delay $t_{jj'}^{pro}$ and transmission delay $t_{jj'}^{trans}$, i.e.,

$$t_{jj'}^{out} = t_{jj'}^{pro} + t_{jj'}^{trans} \tag{12}$$

where $j = |\overrightarrow{f_{\varphi}}|$, and $j' = |\overrightarrow{f_{\varphi+1}}|$ $(\varphi \in [1, M-1])$. Let $u_{jj'}$ represents the amount of resulting data that service s_j outputs.

$$t_{jj'}^{pro} = \begin{cases} \frac{l(_{\mathbb{Z}|s_j^e|}, _{\mathbb{Z}|s_{j'}^e|})}{\lambda}, & |s_j^e| \neq |s_{j'}^e| \\ 0, & |s_j^e| = |s_{j'}^e| \end{cases} \tag{13}$$

$$t_{jj'}^{trans} = \begin{cases} \frac{u_{jj'}}{b}, & |s_j^e| \neq |s_{j'}^e| \\ 0, & |s_j^e| = |s_{j'}^e| \end{cases} \tag{14}$$

Therefore, $\mathbb{T}$ can be re-expressed as:

$$\begin{aligned} \mathbb{T} &= \sum_{j=1}^{M} t_j^{req} + \sum_{j=1}^{M} t_j^{exec} + \sum_{\varphi=1}^{M-1} t_{jj'}^{out} \\ &= \sum_{j=1}^{M} \sum_{i \in |s_j^d|} \left(t_{ij}^{pro} + t_{ij}^{trans}\right) + \sum_{j=1}^{M} \frac{s_j^p}{p_{|s_j^e|}} \\ &\quad + \sum_{\varphi=1}^{M-1} \left(t_{jj'}^{pro} + t_{jj'}^{trans}\right) \end{aligned} \tag{15}$$

B. Problem Formulation

This work aims to find a satisfactory service deployment and data placement scheme for data-intensive applications, so as to minimize response latency. The problem to be solved can be formulated as follows.

$$\min \quad \mathbb{T} \tag{16}$$

$s.t.$

$$\sum_{k=1}^{H} x_{ik} = 1, \forall i \tag{17a}$$

$$\sum_{k=1}^{H} y_{jk} = 1, \forall j \tag{17b}$$

$$(d_i^o \leq o_{|d_i^e|}) \wedge \left(\sum_{i=1}^{N} d_i^o \leqslant \sum_{k=1}^{H} o_k\right), \forall i \tag{17c}$$

$$\left(s_j^m \leq m_{|s_j^e|}\right) \wedge \left(\sum_{j=1}^{M} s_j^m \leqslant \sum_{k=1}^{H} m_k\right), \forall j \tag{17d}$$

$$\left(o^C \gg o_k\right) \wedge \left(p^C \gg p_k\right) \wedge \left(m^C \gg m_k\right), \forall k \tag{17e}$$

Constraints (17a) indicates that only one copy is placed on edge servers for a data block, and (17b) indicates that only one instance is deployed on edge servers for each service. (17c) represents storage resource constraint, and (17d) represents memory resource constraint. Constraint (17e) means that the remote cloud has more powerful capabilities than edge servers.

V. Problem Solving

Let $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$ denote the three parts of (15), i.e.,

$$\begin{cases} \mathcal{P}_1 = \sum_{j=1}^{M} \sum_{i \in |s_j^d|} (r_{ij}^{pro} + t_{ij}^{trans}) \\ \mathcal{P}_2 = \sum_{j=1}^{M} \frac{s_j^p}{p_{|s_j^e|}} \\ \mathcal{P}_3 = \sum_{\varphi=1}^{M-1} (t_{jj'}^{pro} + t_{jj'}^{trans}) \end{cases} \tag{18}$$

Then, the problem to be solved can be further expressed as:

$$\begin{aligned} A: \quad & \min \quad \mathcal{P}_1(X,Y) + \mathcal{P}_2(Y) + \mathcal{P}_3(Y) \\ & s.t. \ \text{(17a), (17b), (17c), (17d), (17e)} \end{aligned} \tag{19}$$

This work aims to solve problem A such that the final data placement scheme X^{opt} and service deployment scheme Y^{opt} can meet the requirement of low response latency. It's important to note that because data placement and service deployment are NP-hard problems, which has been proved in [7], [38] [39], the algorithms we proposed below to obtain data placement scheme X^{opt} and service deployment scheme Y^{opt} are heuristic.

A. Problem Decoupling

In order to decouple the above problem, we introduce two functions $\mathcal{P}_1{}'$ and $\mathcal{P}_1{}''$. Problem A can be reduced to the following problem $B1$ if we randomly generate a feasible service deployment scheme Y^{viable}. We prove that in Appendix.

$$\begin{aligned} B1: \quad & \min \quad \mathcal{P}_1{}'(X, Y^{viable}) \\ & s.t. \ \text{(17a), (17b), (17c), (17d), (17e)} \end{aligned} \tag{20}$$

Then, we can obtain a data placement scheme X^{sub} by solving problem $B1$. Finally, by substituting X^{sub} into the original problem A, problem A can be transformed into problem $B2$.

$$\begin{aligned} B2: \quad & \min \quad \mathcal{P}_1{}''(X^{sub}, Y) + \mathcal{P}_2(Y) + \mathcal{P}_3(Y) \\ & s.t. \ \text{(17a), (17b), (17c), (17d), (17e)} \end{aligned} \tag{21}$$

B. Data Placement Problem

In this section, we focus on solving problem $B1$. First, Y^{viable} is generated. Then, we propose PDPS to obtain X^{sub}.

Let $L = [l_{kk'}]_{k=1,k'=1}^{k=H,k'=H}$ denote a $H \times H$ distance matrix, where $l_{kk'}$ is the euclidean distance between edge servers e_k and $e_{k'}$. If $k = k'$, then $l_{kk'} = \infty$. Let $L^C = [l_k^C]_{k=1}^{k=H}$ denote the distance matrix about remote cloud, where l_k^C is the euclidean distance between edge server e_k and remote cloud. Moreover, let $W = [w_{\varrho k}]_{\varrho=1,k=1}^{\varrho=2,k=H}$ denote a $2 \times H$ residual capacity matrix. Then, row vector $W_1 = [w_{1k}]_{k=1}^{k=H}$ represents the remaining storage capacity of edge servers, and $W_2 = [w_{2k}]_{k=1}^{k=H}$ represents the remaining memory capacity of edge servers.

Rule 1 (Placement): For a data block d_i, edge server e_k satisfies the placement condition indicating that its remaining storage capacity meets the demand of this data block, i.e., $w_{1k} \geqslant d_i^o$.

Rule 2 (Deployment): For service s_j, edge server e_k satisfies the deployment condition indicating that its remaining memory capacity meets the demand of this service, i.e., $w_{2k} \geqslant s_j^m$.

The main idea of PDPS algorithm is that each data block is preferentially placed on edge server e_k, where the service driven by this data is deployed on. If e_k does not satisfy the placement condition, this data block is placed on $e_{k'}$ which is the closest to e_k. If $e_{k'}$ cannot satisfy the placement condition either, this data block is placed on $e_{k''}$ from the remaining edge servers, which has the smallest storage capacity satisfying the placement condition. In addition, a full copy of all data blocks D are also stored in the remote cloud. The pseudo-code and time complexity analysis of PDPS algorithm is in Appendix.

C. Service Deployment Problem

In this section, we focus on solving problem $B2$. Assuming that the data size is very small in the output result phase,

then, the network latency $\mathcal{P}_3 = \sum_{\varphi=1}^{M-1}(t_{jj'}^{pro} + t_{jj'}^{trans})$ among services is essentially negligible, i.e., $\mathcal{P}_3(Y) = 0$. Thus, $B2$ can be transformed into $C1$:

$$C1: \quad \min \quad \mathcal{P}_1''(X^{sub}, Y) + \mathcal{P}_2(Y)$$
$$s.t. \quad (17a), (17b), (17c), (17d), (17e) \tag{22}$$

Problem $C1$ is a classical assignment problem [4], [40], which can be solved by using the Hungarian algorithm [41]. The original Hungarian algorithm requires that services and edge server should have a strict one-to-one relationship. However, the number of services and edge servers is not necessarily equal, and multiple services may be deployed on the same edge server. Therefore, we make modifications on the basis of the Hungarian algorithm and propose HA-SDS to obtain Y^{sub}.

Let $\Lambda = [\tau_{jk}]_{j=1,k=1}^{j=M,k=H}$ denote an $M \times H$ cost matrix, where $\tau_{jk} = \mathcal{P}_1''(X^{sub}, Y) + \mathcal{P}_2(Y)$ is the latency cost when service s_j is deployed on edge server e_k. If the deployment condition is not satisfied, then $\tau_{jk} = \infty$.

The main idea of HA-SDS are as follows. The numbers of services and edge servers are compared first. When $M = H$, the original Hungarian algorithm is applied. When $M < H$, we add $H - M$ virtual services. The cost matrix is denoted as Λ', and the values of the last $H - M$ rows are all set to 0. Then, the standard Hungarian is applied to cost matrix Λ'. When $M > H$, the cost matrix Λ is divided into ξ ($\xi = \lceil M/H \rceil$) small $H \times H$ matrices according to the number of edge servers, denoted as $\Lambda_1 \sim \Lambda_\xi$. If the number of services in the last small matrix Λ_ξ is less than the number of edge servers, a corresponding number of virtual services are added to it, and the related latency cost is set to 0. Then, the standard Hungarian algorithm is applied to each small matrix in order. The pseudo-code and time complexity analysis of HA-SDS algorithm is in Appendix.

D. Joint Data Placement and Service Deployment

By PDPS and HA-SDS, we can obtain X^{sub} and Y^{sub}. However, in the process of problem decoupling, we use the strategies of problem reduction and simplification. Moreover, in the process of finding Y^{sub}, although we consider the communication latency between data blocks and services, as well as the execution latency of services, we assume that the data size is very small in the output result phase to neglect the communication latency among services. Therefore, the obtained X^{sub} and Y^{sub} can be further optimized. Next, we propose RW-DAS, so as to adjust X^{sub} and Y^{sub}, and obtain the optimized data placement scheme X^{opt} and optimized service deployment scheme Y^{opt}. This algorithm possesses the versatility to be seamlessly integrated into various data placement and service deployment algorithms, yielding superior data placement and service deployment schemes.

Let $\Delta = [\sigma_j]_{j=1}^{M}$ denote the response weight matrix, where $\sigma_j = t_j^{req} + t_j^{exec} + t_{jj'}^{out}$ is the response weight of service s_j. For the same data placement scheme and service deployment scheme, the service that has the largest response weight is called a bottleneck service.

Rule 3 (Data location exchange): If there is $(w_{1k} + d_i^o \geq d_{i'}^o) \wedge (w_{1k'} + d_{i'}^o \geq d_i^o)$, then data block d_i placed on edge server e_k and data block $d_{i'}$ placed on edge server $e_{k'}$ satisfy the location exchange condition.

Rule 4 (Service location exchange): If there is $(w_{2k} + s_j^m \geq s_{j'}^m) \wedge (w_{2\,k'} + s_{j'}^m \geq s_j^m)$, then service s_j deployed on edge server e_k and service $s_{j'}$ deployed on edge server $e_{k'}$ satisfy the location exchange condition.

The main idea of RW-DAS is as follows:

- The bottleneck service $s_{\mathbb{j}}$, which is obtained based on X^{sub} and Y^{sub}, is redeployed on other edge servers that satisfy Rule 2 or exchanged locations with other services that satisfy Rule 4. If a lower response latency $\mathbb{T}$ can be obtained, the location of this service is changed, and Y^{sub} is updated.
- Each bottleneck data d_i ($r_{ij} = 1$) which drives bottleneck service $s_{\mathbb{j}}$ is re-placed on other edge servers that satisfy Rule 1 or exchanged locations with other data blocks that satisfy Rule 3. Specifically, the replace and position exchange ideas are similar to PDPS's. If a lower response latency $\mathbb{T}$ can be obtained, the location of this data block is changed, and X^{sub} is updated.
- The above two steps are iterated until a lower response latency cannot be obtained. Then output X^{opt} and Y^{opt}.

The pseudo-code of RW-DAS is shown in Algorithm 1.

Line 1 initializes the distance matrices L and L^C, the remaining capability matrix W, the response latency $\mathbb{T}^{sub}$, and the response weight matrix Δ. Lines 2-3 initialize X^{opt}, Y^{opt}, $\mathbb{T}^{opt}$, the iteration stop flag $stop$, and the number of iterations $count$. Line 6 indicates that the bottleneck service and the bottleneck data blocks of the current iteration are obtained. Lines 7-11 indicate that the bottleneck service can be redeployed on other edge servers. Lines 12-16 indicate that the bottleneck service can be exchanged with other services. Lines 18-30 indicate that each bottleneck data block can be replaced on other edge servers or exchanged with other data blocks. Lines 31-35 indicate that if the current iteration yields a better result, then this result is saved; otherwise, the iteration is terminated.

We can prove the convergence of RW-DAS algorithm based on the monotonicity of the objective function and the finiteness of the optimization space. During each iteration, RW-DAS ensures that the response latency $\mathbb{T}$ decreases strictly monotonically by redeploying bottleneck services or adjusting the locations of bottleneck data, satisfying $\mathbb{T}^{count+1} \leq \mathbb{T}^{count}$. Additionally, since the response latency T is non-negative and the search space for data placement and service deployment is finite, the algorithm is guaranteed to converge to a local optimal solution within a finite number of iterations, satisfying $\mathbb{T}^{count+1} = \mathbb{T}^{count}$.

Time Complexity Analysis: The RW-DAS algorithm, in each iteration, identifies the bottleneck service and the data driving this bottleneck service, and deploys/places them on new edge servers provided that this action can reduce the response latency. The most time-consuming step is to swap the location of the data driving the service with the data on the target servers, which include only the server hosting the service and the server closest to the one hosting the service. Therefore, the time complexity is $O(C \cdot N^2)$, where C is the number of iterations. However, in practice, the actual time complexity is far less than $O(C \cdot N^2)$, because the number of data driving a specific service is much smaller than N, and the number of data placed on a single server is also much smaller than N.

VI. Experiments and Analysis

We implement the proposed PDPS, HA-SDS and RW-DAS in a Python 3.7.12 environment. All the experiments are

Algorithm 1: Dynamic Adjustment Strategy Based on Response Weight.

Input: N, M, H, C and each d_i, s_j, e_k, X^{sub} and Y^{sub}
Output: X^{opt}, Y^{opt}, $\mathbb{T}^{opt}$, $count$
Initialize: $L, L^C, W, \mathbb{T}^{sub}, \Delta$
$X^{opt} \leftarrow X^{sub}$; $Y^{opt} \leftarrow Y^{sub}$; $\mathbb{T}^{opt} \leftarrow \mathbb{T}^{sub}$;
$stop = 0$; $count = 0$;
while $stop == 0$ **do**
  $count++$;
  $\mathbb{j} \leftarrow |\max\{\sigma_j\}_{j=1}^{M}|$; $\mathbb{I} \leftarrow |s_{\mathbb{j}}^{d}|$;
  for $k = 1, \ldots, H$ **do**
    if $(flag_{\mathbb{j}k}^{deployment}) \wedge (flag^{lower\mathbb{T}^{sub}})$ **then**
      update $s_{\mathbb{j}}^{e}$, $\mathbb{T}^{sub}$, Y^{sub}, W, Δ;
    end
  end
  for $j = 1, \ldots, M$ **do**
    if $(flag_{\mathbb{j}j}^{exchange}) \wedge (flag^{lower\mathbb{T}^{sub}})$ **then**
      update $s_{\mathbb{j}}^{e}$, s_{j}^{e}, $\mathbb{T}^{sub}$, Y^{sub}, W, Δ;
    end
  end
  $k = |s_{\mathbb{j}}^{e}|$; $k' \leftarrow \min\{l_{kk'}\}_{k'=1}^{k'=H}$;
  for $\mathbb{i} \in \mathbb{I}$ **do**
    for $\mathbb{k} \in \{k, k'\}$ **do**
      if $(flag_{\mathbb{i}\mathbb{k}}^{placement}) \wedge (flag^{lower\mathbb{T}^{sub}})$ **then**
        update $d_{\mathbb{i}}^{e}$, $\mathbb{T}^{sub}$, X^{sub}, W, Δ;
      else
        for $(i \in |D|) \wedge (x_{i\mathbb{k}} = 1)$ **do**
          if $(flag_{\mathbb{i}i}^{exchange}) \wedge (flag^{lower\mathbb{T}^{sub}})$ **then**
            update $d_{\mathbb{i}}^{e}$, d_{i}^{e}, $\mathbb{T}^{sub}$, X^{sub}, W, Δ;
          end
        end
      end
    end
  end
  if $\mathbb{T}^{opt} > \mathbb{T}^{sub}$ **then**
    $X^{opt} \leftarrow X^{sub}$; $Y^{opt} \leftarrow Y^{sub}$; $\mathbb{T}^{opt} \leftarrow \mathbb{T}^{sub}$;
  else
    $stop = 1$;
  end
end
Return X^{opt}, Y^{opt}, $\mathbb{T}^{opt}$, $count$;

performed on an AMD Ryzen 7 4800H with Radeon Graphics 2.90 GHz processor, 16.0 GB RAM and Windows 10 operating system. Due to the lack of suitable edge server information and benchmark data sets, we automatically generate different datasets through program control by referring to [7], [42]. In our experiments, the most important issue is the driving relationship between data and services, which has a great impact on the results. Therefore, in our datasets, the indexes of data-driven services are randomly generated, which ensures that some services do not need to request data, and some services need to request more than one data. See Table I for details on parameter settings.

A. Baseline Algorithms

The proposed method (named as PDPS-HA-RW) consists of three stages: 1) PDPS is used to get a data placement scheme. 2) HA-SDS is applied to obtain a service deployment scheme. 3) RW-DAS is used to adjust the above data placement scheme and service deployment scheme, so as to obtain the final data placement and service deployment scheme.

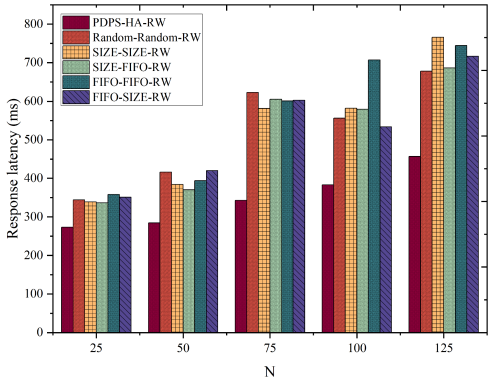


Fig. 2. Comparison of latency with changing the number of data blocks.

To evaluate the effectiveness of different strategies in our proposed methodology, we implement three baseline strategies, Random, SIZE, and FIFO. By combining different baseline strategies and RW-DAS, five combination strategies, Random-Random-RW, SIZE-SIZE-RW, SIZE-FIFO-RW, FIFO-FIFO-RW, FIFO-SIZE-RW can be obtained.

- *Random:* It randomly selects the available edge servers for data placement and service deployment.
- *SIZE:* The largest data block is preferentially placed on the edge server with the smallest storage capacity which can meet the placement condition. The service with largest computing demand is preferentially deployed on the edge server with the largest computing power, which can meet the deployment condition.
- *FIFO:* According to the execution order of services, the corresponding data blocks are placed on the edge server with the smallest storage capacity that meets the placement condition. According to the execution order of services, the first service is preferentially deployed on the edge server with the largest computing power that meets the deployment condition.

Considering the randomness, all algorithms are executed five times and the results are averaged. In addition, let γ denotes the response latency reduction rate after applying the RW-DAS, i.e.,

$$\gamma = \left(\frac{\mathbb{T} \text{ without RW } - \mathbb{T}^{+} \text{ after using RW}}{\mathbb{T} \text{ without RW}} \right) \times 100\% \quad (23)$$

1) Change the Number of Data Blocks: Fix $M = 100$, $H = 150$, and change N. As shown in Fig. 2, our algorithm can always get the lowest response latency and the fastest convergence speed compared with other methods. In addition to our algorithm, when $N < 75$, SIZE-FIFO-RW performs relatively well. When $N > 75$, the advantages of Random-Random-RW and FIFO-SIZE-RW begin to emerge slowly. This is because as N increases, the driving relationship between data and services becomes more and more complex, and service deployment using SIZE performs better than using FIFO. When N is small, γ of SIZE-FIFO-RW always remains at a high position. As the value of N gradually increases, the γ of FIFO-SIZE-RW algorithm begins to increase, almost approaching 30% at $N = 100$, and the corresponding response time reaches the lowest among the five baseline combination algorithms. This shows that part of the credit for this excellent performance belongs to our RW algorithm, and the FIFO-SIZE itself may not perform so well.

TABLE I
PARAMETER VALUES

Variable name	Parameter name	Parameter value or range
data block	data block size (MB)	[200, 800]
service	memory capacity required for service (MB)	[200, 400]
	the number of computing units included in the service (unit)	[300, 800]
	the amount of data transferred between services (MB)	150
edge server	storage capacity (MB)	[500, 1000]
	memory capacity (MB)	[400, 800]
	computing capacity (unit/ms)	[500, 1000]
	longitude	[112.087, 118.841]
	latitude	[29.852, 33.648]
	propagation rate λ (km/ms)	200
	bandwidth b (MB/ms)	300
remote cloud	storage capacity (MB)	10 0000
	memory capacity (MB)	10 0000
	computing capacity (unit/ms)	10 0000
	longitude	116.425
	latitude	39.906
	propagation rate λ^C (km/ms)	250
	bandwidth b^C (MB/ms)	400

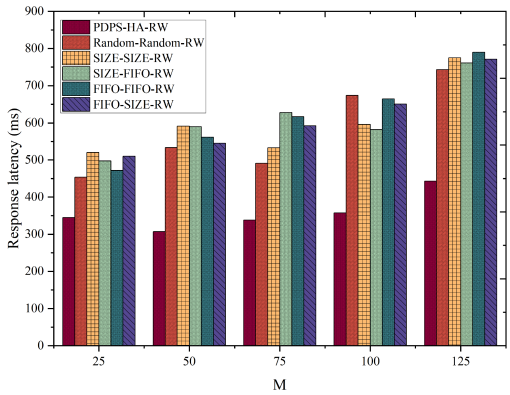


Fig. 3. Comparison of latency with changing the number of services.

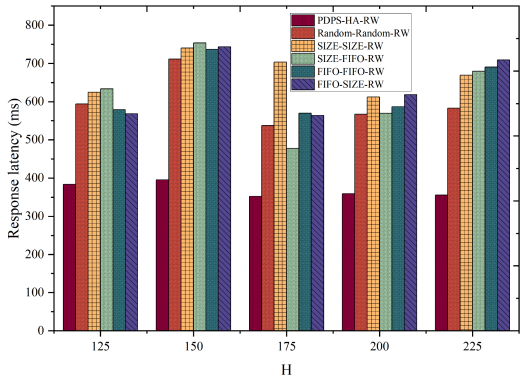


Fig. 4. Comparison of latency with changing the number of edge servers.

In addition, when $N = 75$, the performance of the five baseline combination algorithms is relatively flat, and the corresponding γ has a sudden downward change, but all remain around 5%. The reason is that the driving relationship between the data block and the server is relatively balanced at this time, the storage resources and computing resources in the entire heterogeneous network are in a relatively reasonable resource competition state, and the load balance of the system tends to be stable.

2) Change the Number of Services: Fix $N = 100$, $H = 150$, and change M. Combined with Fig. 3, our algorithm can always get the lowest response latency compared with other algorithms, and the convergence speed is always stable in a faster state. Besides, when $M \leq 75$, Random-Random-RW performs the best among the five baseline combined algorithms. Random-Random-RW always maintains the highest γ in the first half. This shows that part of its good performance is due to our RW algorithm, and its Random-Random algorithm itself may not be good. In addition, the other four baseline combination algorithms are always in a state of mutual game, until $M = 125$, they tend to be in equilibrium with each other. This is because the number N of data blocks is always fixed at 100. When $M =$ 25, 50, 75, and 100, the number of services is always less than or equal to the number of data blocks. The driving relationship has a great influence on the experimental results.

3) Change the Number of Edge Servers: Fix $N = 100$, $M =$ 100, and change H. Combined with Fig. 4, our algorithm can always get the lowest response latency compared with other algorithms, and the convergence speed is always stable in a faster state. In addition, Random-Random-RW performs well on the whole. The FIFO-FIFO-RW and FIFO-SIZE-RW have obvious advantages in the early stage, and the later advantages are gradually replaced by the SIZE-SIZE-RW and SIZE-FIFO-RW. Especially when $H = 175$, the response latency of SIZE-FIFO-RW is significantly lower than the response latency of the other four baseline combination algorithms, and the corresponding γ value exceeds 35%, which means that it achieves good result at the same time. There is a considerable time cost, and part of this advantage is obtained with the help of our RW algorithm. At the same time, our algorithm also obtains the lowest response time, but the time cost paid in the RW stage is almost zero.

4) Change Storage Capacity of Edge Servers: Fix the other parameters, and change the storage capacity of edge servers to [600,1100], [700,1200], [800,1300], [900,1400], [1000,1500], so as to verify the impact of storage capacity on different methods. Combining Fig. 5, our algorithm PB-HA-RW always maintains the lowest response latency and has the fastest convergence speed. Besides, Random-Random-RW performed the best among the five baseline combined algorithms, but its γ was consistently higher than 25%, and even exceeded 35% twice. This means that although the algorithm obtains a lower response latency, its convergence speed in the RW stage is slower, but if the RW algorithm is removed, the effect of Random-Random algorithm itself may not be so brilliant. In addition, SIZE-FIFO-RW performs the worst among the five baseline combination algorithms, while the corresponding FIFO-SIZE-RW algorithm performs well. This is because the values of the three parameters

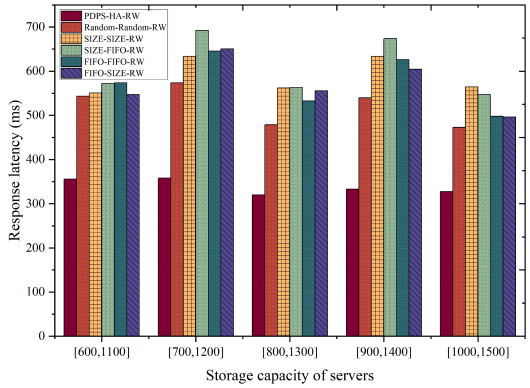


Fig. 5. Comparison of latency with changing storage capacity of edge servers.

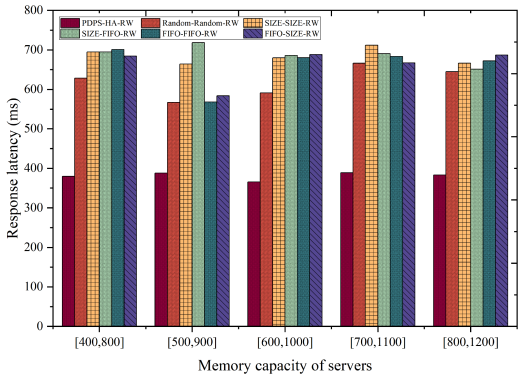


Fig. 6. Comparison of latency with changing memory capacity of edge servers.

N, M, and H are fixed at 100, 100, and 150 respectively. The influence of the driving relationship between the data block and the service is dominant in the whole experiment. Compared with FIFO, SIZE is more suitable for the service deployment stage. And FIFO is more suitable for the data placement stage.

5) Change Memory Capacity of Edge Servers: Fix the other parameters, and change the memory capacity of edge servers. Combining Fig. 6, our algorithm can always obtain the lowest response latency and the fastest convergence in the RW stage. Furthermore, Random-Random-RW performs the best among the five baseline combined algorithms. However, its γ always remains the highest, which indicates that the algorithm has the slowest convergence speed and the longest time-consuming in the RW stage. The overall performance of the other four baseline combination algorithms is relatively stable. This is because as the memory capacity of the server increases, a single edge server can deploy multiple services at the same time, and the probability of the service and the data driving it on the same edge server increases, and the driving relationship has less influence on the experimental results. Therefore, the difference between SIZE and FIFO in the data placement stage and the service deployment stage is not obvious. When the memory capacity of the server is in the range of [500, 900], SIZE-SIZE-RW and SIZE-FIFO-RW perform poorly. This is because our data set is randomly generated through program control, and the driving relationship in the corresponding data set at this stage is relatively tense, while the overall memory capacity of the server is relatively small, and the driving relationship still occupies a dominant position.

6) Change Computing Capacity of Edge Servers: Fix the other parameters, and change the computing capacity of edge servers. Combining Fig. 7, our algorithm can always get the lowest response latency compared to other algorithms, and the convergence speed is always the fastest. In addition, the overall performance of Random-Random-RW is better. As the computing capacity of the server increases, FIFO-SIZE-RW begins to show its advantages. But when the computing power of the server tends to be saturated, SIZE-SIZE-RW obtains a lower response latency, while FIFO-SIZE-RW performs the worst. This is because when the computing capacity of the server increases, the driving relationship between the data and the service has little effect on the experimental results. The SIZE performs better in the service deployment stage, and FIFO performs better in the data placement stage.

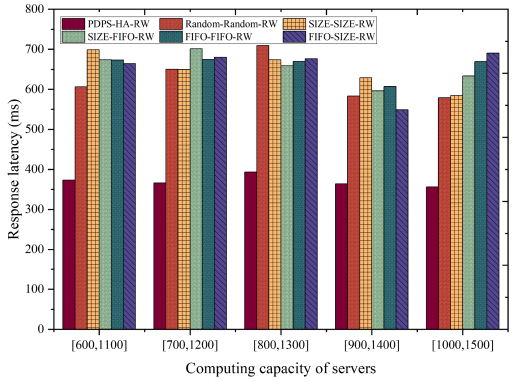


Fig. 7. Comparison of latency with changing computing capacity of edge servers.

TABLE II
DATASET SCALE SETTING

Dataset Name	Dataset Size		
	N	M	H
I	50	50	70
II	100	100	120
III	150	150	180
IV	200	200	240
V	250	250	300

B. Ablation Study

To further evaluate the effectiveness of different strategies in our proposed method, we have also conducted some ablation studies. By replacing our proposed strategies with different baseline strategies, new methods are obtained as comparison. Then, by fixing other parameters, and changing the number of data blocks, services, and edge servers respectively, we can obtain five data sets with different scale (Table II).

1) Change the Strategy in the First Stage: From Fig. 8, it can be found that our algorithm PDPS-HA-RW performs the best on the five datasets. Although the other three algorithms Random-HA-RW, SIZE-HA-RW, and FIFO-HA-RW also showed good results when the data set was small. As the scale of the data set increased, the response latency obtained by our algorithm is lower and lower than the other three algorithms, and the effect is more obvious. When the data set is large, although the convergence speed of our algorithm in the third stage (RW) is always higher than SIZE-HA-RW and FIFO-HA-RW, but γ is always stable below 0.5% for almost negligible time. In addition, although SIZE-HA-RW and FIFO-HA-RW perform well on dataset I, they take too much time to converge in the RW stage. And, from another perspective, the good performance of these two algorithms is partly due to our RW algorithm, and

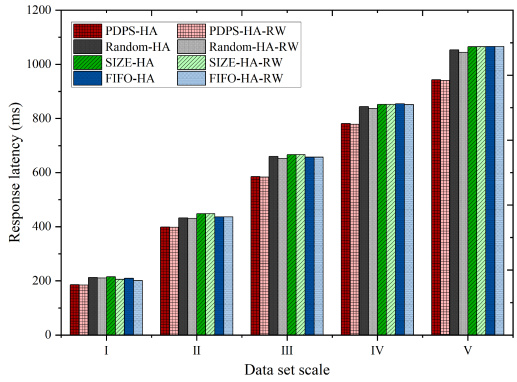


Fig. 8. Ablation studies of changing the strategy in the first stage.

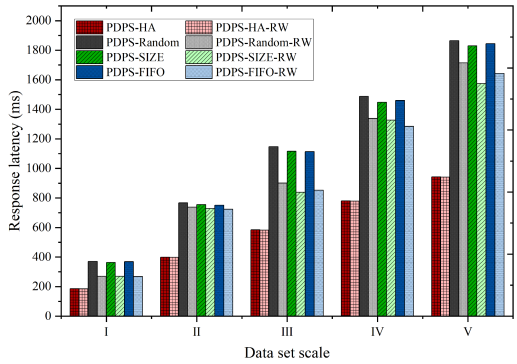


Fig. 9. Ablation studies of changing the strategy in the second stage.

SIZE-HA and FIFO-HA themselves may not be outstanding. Overall, compared with the other three baseline algorithms, the PB algorithm can make our algorithm get lower response latency.

2) Change the Strategy in the Second Stage: From Fig. 9, it can be found that our algorithm PDPS-HA-RW performs the best on the five datasets. Although the other three algorithms PDPS-Random-RW, PDPS-SIZE-RW, and PDPS-FIFO-RW also showed good results when the data set was small. As the scale of the data set increased, the response latency obtained by our algorithm is lower and lower than the other three algorithms, and the effect is more obvious. In addition, the γ of our algorithm in the third stage (RW) is almost zero, which means that our algorithm has achieved near-optimal when the second stage (HA) is completed. The other three baseline combination algorithms all take too much time to converge in the RW stage. Among them, γ obtained by PDPS-SIZE-RW in the RW stage on the III reached a high of 25%, which shows that part of the credit for the algorithm's good results on this data set belongs to us RW algorithm. Overall, our HA algorithm not only achieves lower response latency, but also has the best convergence effect compared to the other three baseline combination algorithms.

C. Comparison With SOTA Algorithms

To further evaluate the performance of our proposed method, we compared it with several SOTA algorithms, DESGA [7], hJTORA [43], EISPA [20] and AES-JDR [36]. Their time complexities are $O(n^2)$, $O(n^3)$, $O(n^3)$ and $O(n^2)$, respectively.

1) Change the Number of Data Blocks, Services and Edge Servers Respectively: From Fig. 10(a), when N is small, the performance of hJTORA, DSEGA, EISPA and AES-JDR is almost the same. However, with the increase of N, the data-driven relationship between data and services gradually becomes more complex, and the performance of AES-JDR is better. After applying the RW, AES-JDR-RW has a more obvious optimization effect than hJTORA-RW, DSEGA-RW and EISPA-RW. From Fig. 10(b), as M increases, both before and after applying RW, AES-JDR performs slightly better than hJTORA, DSEGA and EISPA. From Fig. 10(c), AES-JDR is always better than hJTORA, DSEGA and EISPA. Moreover, no matter how H changes, our algorithm and these SOTA algorithms are always in a small fluctuation state on the whole. After applying RW, AES-JDR-RW significantly outperforms hJTORA-RW, DSEGA-RW and EISPA-RW.

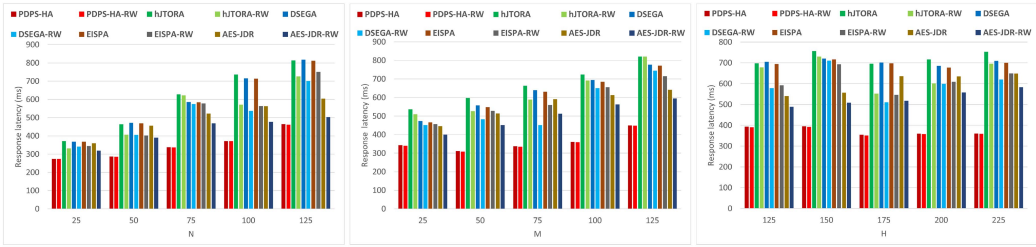
Combined with Fig. 10(a)–(c), our algorithm is not greatly affected by the three parameters of N, M, and H, and whether applying RW or not, our algorithm can obtain the lowest response latency. This is because our algorithm jointly optimizes data placement and service deployment in a distributed cloud-edge environment to minimize response latency. However, these SOTA algorithms did not take this into account. As a result, after applying RW to optimize their data placement and service deployment schemes, the response latency is significantly reduced.

2) Change Storage, Memory and Computing Capacity of Edge Servers Respectively: From Fig. 11(a), as the storage capacity of edge servers increases, AES-JDR consistently outperforms hJTORA, DSEGA and EISPA. However, when the storage capacity of the edge server reaches saturation, hJTORA-RW performs better than DSEGA-RW, EISPA-RW and AES-JDR-RW with the optimization of the RW algorithm. From Fig. 11(b) and (c), AES-JDR is always better than hJTORA, DSEGA and EISPA. After applying RW, AES-JDR-RW significantly outperforms hJTORA-RW, DSEGA-RW and EISPA-RW.

Combining with Fig. 11(a)–(c), our algorithm is not greatly affected by the change of edge server capability, and whether applying RW or not, our algorithm both obtain the lowest response latency. The performance of hJTORA, DSEGA, EISPA and AES-JDR may be affected by the driving relationship between data and service. Since our dataset is randomly generated by program control, there may be some small differences in the driving relationship between datasets, which leads to fluctuations in the results of these algorithms, but these fluctuations are acceptable within the range. In addition, after the application of RW algorithm, the response latencies of AES-JDR-RW, DSEGA-RW, hJTORA-RW and EISPA-RW were significantly reduced.

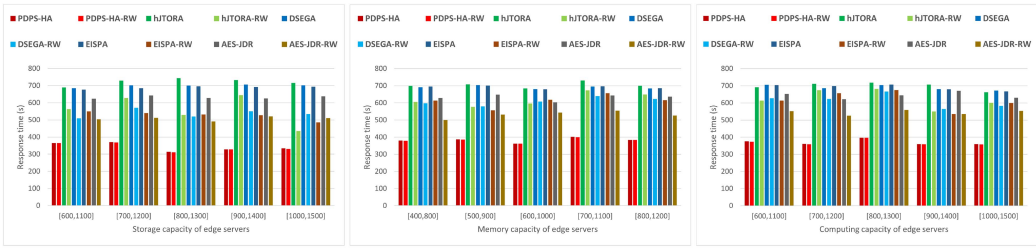
3) Running Time and Resource Consumption: In real-world applications, particularly in resource-constrained environments, the computational overhead of algorithms becomes critical. Therefore, to thoroughly evaluate algorithm performance, it is essential to prioritize computational efficiency and conduct a comprehensive assessment of resource consumption. Under Dataset III in Table II, we conducted multiple experiments to obtain the running time, CPU utilization, and memory usage of the algorithms. The scale of this dataset is already quite large, as indicated by the comparative algorithm papers we referenced and the results of our practical research.

The experimental results are presented in Table III. Under comparable conditions of CPU utilization and memory usage, PDPS-HA-RW demonstrates significant advantages in running time performance. As elaborated in Section VI-C2, this is primarily because the PDPS-HA algorithm inherently approaches near-optimal solutions, requiring only minimal iterations during the RW phase to achieve convergence. Furthermore, PDPS-HA



(a) (b) (c)

Fig. 10. Comparison of latency with changing the number of N, M, H: (a) changing the number of data blocks N; (b) changing the number of services M; (c) changing the number of edge servers H.



(a) (b) (c)

Fig. 11. Comparison of latency with changing capabilities of edge servers: (a) changing the storage capacity; (b) changing the memory capacity; (c) changes the computing capacity.

TABLE III
COMPARISON OF RUNNING TIME AND RESOURCE CONSUMPTION

Algorithm	Running Time (s)	CPU (%)	Memory (MB)
PDPS-HA-RW	5.99	93.93	120.99
hJTORA-RW	109.51	96.50	120.52
DSEGA-RW	215.76	96.40	120.47
EISPA-RW	193.939	96.033	120.57
AES-JDR-RW	100.66	96.404	120.44

TABLE IV
COMPARISON OF STANDARD DEVIATION

Algorithm	Standard Deviation (ms)
PDPS-HA-RW	14.27
hJTORA-RW	16.78
DSEGA-RW	63.64
EISPA-RW	49.92
AES-JDR-RW	45.48

exhibits superior time complexity compared to other algorithms. In contrast, alternative approaches employing PSO, GA, or other methods necessitate multiple optimization iterations during the RW phase, consequently resulting in substantially prolonged execution times.

When the total amount of resources is sufficient but constrained, the data placement strategy of hJTORA may lead to suboptimal or unbalanced resource allocation—this occurs because the strategy prioritizes selecting the first server that meets the conditions, which can negatively impact the initial environment for subsequent service deployment. Furthermore, during the service deployment phase, the feasible space for move and exchange operations is significantly reduced, making it difficult to effectively optimize bottleneck services. The GA for service deployment in DSEGA may face a lack of diversity among feasible solutions during population initialization. At the same time, the effectiveness of evolutionary operations such as mutation is reduced, leading to a higher risk of premature convergence, or requiring more iterations to find a satisfactory solution. As a result, the quality and stability of the solutions may be worse compared to when resources are abundant. For EISPA, resource constraints reduce the effective search space, leading to a situation where a large number of particles representing service deployment strategies may fall into infeasible regions. This decreases the diversity of particles during the initialization phase and increases the risk of converging to local optima. With the feasible solution space narrowed down, the algorithm frequently encounters resource boundaries when updating particle positions and velocities, further limiting its exploration capability. In the first stage of AES-JDR, when high-quality node resources are in short supply, the rounding step may force the deployment of microservices in suboptimal locations, leading the solution quality close to the theoretical worst-case scenario. The routing optimization in the second stage heavily relies on the deployment quality of the previous stage. When critical microservices are sparsely deployed or located in unfavorable positions, the flexibility of adaptive routing is significantly reduced.

4) Standard Deviation: Similarly, to verify the stability of the algorithm, we conducted multiple experiments and performed standard deviation analysis on the results. We selected the results from Dataset III in Table II for presentation. It can be observed from the Table IV that our algorithm achieves the smallest standard deviation, which demonstrates superior stability. As can be seen from Figs. 10, 11, Tables III, and IV, our algorithm not only produces the solution with the lowest response latency, but also demonstrates significant superiority in terms of actual running time, resource consumption, and stability.

5) Effectiveness of RW: From Figs. 8, 9, 10, and 11, our proposed RW-DAS, namely the third-stage algorithm RW, can be seamlessly integrated into various algorithms to effectively optimize the existing data placement and service deployment schemes with a notably significant optimization effect on response latency. It is worth noting that the RW algorithm has a higher degree of adaptation to the algorithms with faster

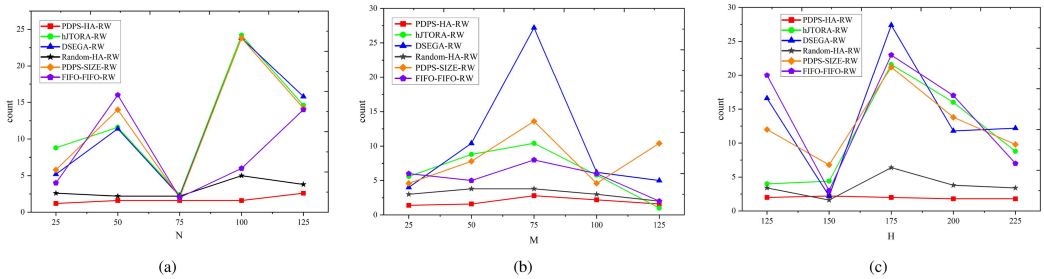


Fig. 12. Comparison of efficiency *count* with changing of N, M, H: (a) changing the number of data blocks N; (b) changing the number of services M; (c) changing the number of edge servers H.

convergence (such as our HA algorithm), and while achieving better optimization results, the time cost is negligible. For other algorithms with slower convergence speed (such as the three baseline algorithms), the more significant the optimization effect, the higher the time cost.

We also noticed that although the performance of PDPS-HA has been improved by RW, the effect is not very obvious. This is because the PDPS-HA we proposed is good enough and has something in common with RW. In the data placement stage, PDPS is less likely to fall into local optimum than other algorithms. In the subsequent service deployment stage, the cost matrix of HA is generated based on the data request latency of the service and the service execution latency, which is similar to the response weight. Other algorithms require adjustments during the RW phase, but the PDPS-HA algorithm has already preemptively achieved these adjustments during the HA phase. Taking these points into account, it is evident that the PDPS-HA is already quite close to the optimal solution, thus requiring minimal iterations during the RW phase to converge to the optimum.

D. Efficiency Experiment and Analysis

In Sections VI-A, VI-B, and VI-C, we compared with the five baseline combination algorithms and several SOTA algorithms to perform effect and parameter experiments, and also performed ablation experiments for different strategies of the proposed method, and showed that our method performs best in terms of both response latency and convergence. Since the third-stage optimization algorithm (RW) performs well on other comparison algorithms and our first two-stage algorithm (PDPS-HA), we will go further in this section to verify the efficiency of the RW strategy. We selected three baseline combination algorithms Random-HA-RW, PDPS-SIZE-RW, FIFO-FIFO-RW that performed well in previous experiments, and also hJTORA-RW and DSEGA-RW for comparison.

1) Change the Number of Data Blocks, Services and Edge Servers Respectively: As shown in Fig. 12(a), as N increases, our algorithm always maintains the lowest number of iterations compared to the other five algorithms. This shows that we do not need to spend a lot of time on the RW stage, and the approximate optimal solution has been obtained when the previous two stages of PDPS-HA are completed. The performance of Random-HB-RW is sub-optimal, and the *count* is always stable below 5 times. Combined with the analysis in the previous Section VI-B2, we can see that most of the credit for this performance is due to our HA algorithm. When $N = 75$, the *count* of all algorithms is maintained at about 2.5, that is, it has basically converged when RW is not used. The reason is that the driving relationship between data blocks and services is relatively balanced at this time, there is a relatively reasonable resource competition between storage resources and computing resources in the entire heterogeneous network, and the load balance of the system tends to be stable. From Fig. 12(b), our algorithm performs the best, the Random-HA-RW algorithm performs sub-optimally, and DSEGA-RW and PDPS-SIZE-RW perform poorly. This shows when the driving relationship between data and service is relatively tense, DSEGA and PDPS-SIZE are prone to fall into local optimum. After applying our RW algorithm, a better data placement and service deployment scheme can be obtained at a certain time cost. From Fig. 12(c), our algorithm has obvious advantages over other algorithms, and Random-HB-RW performs sub-optimal. Furthermore, hJTORA-RW performs well until $H < 175$, but performs poorly when $H \geq 175$. The DSEGA-RW iterates as many as 27 times in the RW stage when $H = 175$. It means a great cost of time for lower response latency. When $H = 150$, the *count* of all algorithms is stable at around 5, which means that it has basically converged before the RW algorithm is used. The reason is that the number of edge servers is basically saturated at this time, there is a relatively reasonable resource competition between storage resources and computing resources in the entire heterogeneous network, and the load balance of the system tends to be stable. Looking at Fig. 12(a)–(c), our algorithm has basically converged after the first two stages are completed, and the obtained solution is approximately optimal, which makes us not need to spend too much time in the RW stage for further optimization.

2) Change Storage, Memory and Computing Capacity of Edge Servers Respectively: From Fig. 13(a), as the storage capacity increases, our algorithm always maintains the lowest number of iterations compared to the other five algorithms. This shows that we do not need to spend a lot of time on the optimization of the RW stage, and the approximate optimal solution has been obtained when the two stages of PDPS-HA are completed. The performance of Random-HA-RW is sub-optimal, and the *count* is always stable at around 5. In addition, when the storage capacity tends to be saturated, the count of PDPS-SIZE-RW reaches nearly 45 times. This shows that the PDPS-SIZE algorithm is easy to fall into the local optimum when the storage capacity of the edge server is large enough. From Fig. 13(b), our algorithm performs the best, and Random-HA-RW performs the second. As the memory capacity of edge servers increases, the *count* of other algorithms are basically maintained below 10. At this time, the probability of placing multiple services on an edge server is high, and the driving relationship between data and services has little influence on the experimental results, so that

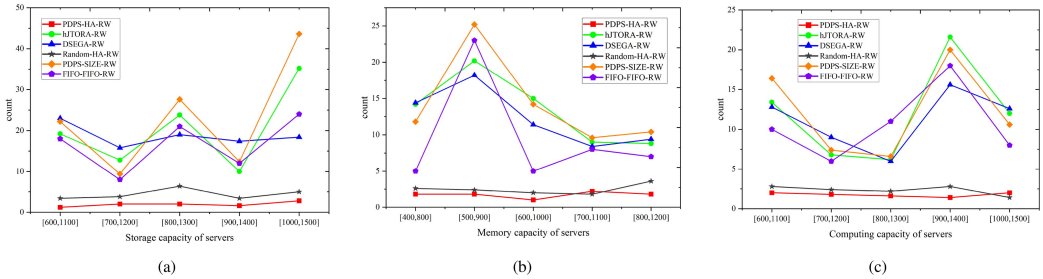


Fig. 13. Comparison of efficiency *count* with changing capabilities of edge servers: (a) changing the storage capacity; (b) changing the memory capacity; (c) changes the computing capacity.

these algorithms can already obtain good solutions in the first two stages. From Fig. 13(c), our algorithm has obvious advantages over other algorithms, Random-HA-RW is sub-optimal, and other algorithms are always in a state of mutual game. But on the whole, the two SOTA algorithms perform worse and worse in terms of *count* as the computing capacity of edge servers increases. This shows that as the computing capacity increases, hJTORA and DSEGA are easy to fall into local optimum, and they spend a great deal of time in the RW stage to obtain a lower response latency. Looking at Fig. 13(a), (b), and (c), our algorithm has basically converged after the first two stages are completed, and the obtained solution is approximately optimal, which makes us not need to spend too much time in the RW stage for further optimization.

VII. Conclusion

Most existing studies about application deployment consider either service deployment or data placement, rather than their joint optimization. Differing from them, this study considers the driving relationships between data and services for data-intensive applications in a cloud-edge collaborative environment, and aims to obtain a desired data placement and service deployment scheme so as to ensuring the quality of services. We formulate the problem and propose a priority-based data placement strategy to identify a data placement scheme. Next we transform the problem into a classical assignment problem, and propose a service deployment strategy based on an improved Hungarian algorithm to find a service deployment scheme. Then, in order to reduce the response latency, we propose a response-weight-based a dynamic adjustment strategy to dynamically adjust the data placement scheme and service deployment scheme to obtain the final solution. Finally, we perform a series of experiments to prove that our proposed methodology can obtain lower response latency than other compared methods. Different composite patterns among service components and more complex associations between data and services should be considered as our next research.

Acknowledgment

A preliminary conference version of this paper appeared in Proceedings of the IEEE International Conference on Web Services (ICWS 2022).

References

[1] L. Silva et al., "Computing paradigms in emerging vehicular environments: A review," *IEEE/CAA J. Automatica Sinica*, vol. 8, no. 3, pp. 491–511, Mar. 2021.

[2] P. Wang, Z. Ding, C. Jiang, M. Zhou, and Y. Zheng, "Automatic web service composition based on uncertainty execution effects," *IEEE Trans. Serv. Comput.*, vol. 9, no. 4, pp. 551–565, Jul./Aug. 2016.

[3] P. Wang, Z. Ding, C. Jiang, and M. Zhou, "Constraint-aware approach to web service composition," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 44, no. 6, pp. 770–784, Jun. 2014.

[4] T. Bahreini and D. Grosu, "Efficient placement of multi-component applications in edge computing systems," in *Proc. 2nd ACM/IEEE Symp. Edge Comput.*, 2017, pp. 1–11.

[5] S. Wang, M. Zafer, and K. K. Leung, "Online placement of multi-component applications in edge computing environments," *IEEE Access*, vol. 5, pp. 2514–2533, 2017.

[6] H. Yuan, J. Bi, and M. Zhou, "Spatiotemporal task scheduling for heterogeneous delay-tolerant applications in distributed green data centers," *IEEE Trans. Automat. Sci. Eng.*, vol. 16, no. 4, pp. 1686–1697, Oct. 2019.

[7] Y. Chen, S. Deng, H. Ma, and J. Yin, "Deploying data-intensive applications with multiple services components on edge," *Mobile Netw. Appl.*, vol. 25, no. 2, pp. 426–441, 2020.

[8] C. Ding, A. Zhou, Y. Liu, R. N. Chang, C.-H. Hsu, and S. Wang, "A cloud-edge collaboration framework for cognitive service," *IEEE Trans. Cloud Comput.*, vol. 10, no. 3, pp. 1489–1499, Jul./Aug. 2022.

[9] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Online collaborative data caching in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 2, pp. 281–294, Feb. 2021.

[10] X. Xia, F. Chen, Q. He, J. C. Grundy, M. Abdelrazek, and H. Jin, "Cost-effective app data distribution in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 31–44, Jan. 2021.

[11] P. Wang, Y. Zhao, and Z. Zhang, "Joint optimization of data caching and task scheduling based on information entropy for mobile edge computing," in *Proc. IEEE Int. Conf. Parallel Distrib. Process. Appl. Big Data Cloud Comput., Sustain. Comput. Commun., Social Comput. Netw. (ISPA/BDCloud/SocialCom/SustainCom)*, 2021, pp. 460–467.

[12] Y. Yang, X. Li, H. Khalajzadeh, X. Liu, X. Ji, and F. Qian, "Data placement cost optimization and load balancing for online social networks," in *Proc. IEEE 7th Int. Conf. Adv. Cloud Big Data*, 2019, pp. 162–167.

[13] P. Wang, C. Zhao, W. Liu, Z. Chen, and Z. Zhang, "Optimizing data placement for cost effective and high available multi-cloud storage," *Comput. Informat.*, vol. 39, no. 1/2, pp. 51–82, 2020.

[14] P. Wang, C. Zhao, Y. Wei, D. Wang, and Z. Zhang, "An adaptive data placement architecture in multicloud environments," *Sci. Program.*, vol. 2020, pp. 1–12, 2020.

[15] W. Liu, P. Wang, Y. Meng, G. Zou, and Z. Zhang, "A novel algorithm for optimizing selection of cloud instance types in multi-cloud environment," in *Proc. IEEE 25th Int. Conf. Parallel Distrib. Syst.*, 2019, pp. 167–170.

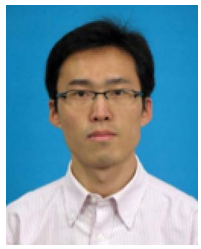
[16] E. Cao, P. Wang, C. Yan, and C. Jiang, "A cloudedge-combined data placement strategy based on user access regions," in *Proc. IEEE 6th Int. Conf. Big Data Inf. Analytics*, 2020, pp. 243–250.

[17] P. Wang, Y. Wei, and Z. Zhang, "Optimizing data placement in multi-cloud environments considering data temperature," in *Proc. Int. Conf. Artif. Intell. Secur.*, 2021, pp. 167–179.

[18] H. Li, G. Xu, D. Wang, M. Zhou, Y. Yuan, and A. Alabdulwahab, "Chaotic-nondominated-sorting owl search algorithm for energy-aware multi-workflow scheduling in hybrid clouds," *IEEE Trans. Sustain. Comput.*, vol. 7, no. 3, pp. 595–608, Jul.-Sep. 2022.

[19] Q. Wu, M. Zhou, Q. Zhu, Y. Xia, and J. Wen, "MOELS: Multiobjective evolutionary list scheduling for cloud workflows," *IEEE Trans. Automat. Sci. Eng.*, vol. 17, no. 1, pp. 166–176, Jan. 2020.

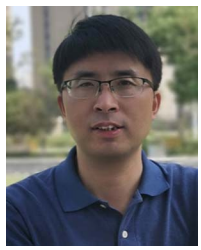
[20] Y. Wang and X. Zuo, "An effective cloud workflow scheduling approach combining PSO and idle time slot-aware rules," *IEEE/CAA J. Automatica Sinica*, vol. 8, no. 5, pp. 1079–1094, May 2021.
[21] P. Wang, Y. Lei, P. R. Agbedanu, and Z. Zhang, "Makespan-driven workflow scheduling in clouds using immune-based PSO algorithm," *IEEE Access*, vol. 8, pp. 29281–29290, 2020.
[22] A. Chikhaoui, L. Lemarchand, K. Boukhalfa, and J. Boukhobza, "Stornir, a multi-objective replica placement strategy for cloud federations," in *Proc. 36th Annu. ACM Symp. Appl. Comput.*, 2021, pp. 50–59.
[23] B. Li et al., "READ: Robustness-oriented edge application deployment in edge computing environment," *IEEE Trans. Serv. Comput.*, vol. 15, no. 3, pp. 1746–1759, May/Jun. 2020.
[24] F. Chen, J. Zhou, X. Xia, H. Jin, and Q. He, "Optimal application deployment in mobile edge computing environment," in *Proc. IEEE 13th Int. Conf. Cloud Comput.*, 2020, pp. 184–192.
[25] H. Wang, X. Chen, H. Xu, J. Liu, and L. Huang, "Joint job offloading and resource allocation for distributed deep learning in edge computing," in *Proc. IEEE 21st Int. Conf. High Perform. Comput. Commun.; IEEE 17th Int. Conf. Smart City; IEEE 5th Int. Conf. Data Sci. Syst.*, 2019, pp. 734–741.
[26] P. Wu, W. Yang, H. Wang, and L. Huang, "GDS: General distributed strategy for functional dependency discovery algorithms," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2020, pp. 270–278.
[27] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 207–215.
[28] Z. Lei, H. Xu, L. Huang, and Z. Meng, "Joint service placement and request scheduling for multi-SP mobile edge computing network," in *Proc. IEEE 26th Int. Conf. Parallel Distrib. Syst.*, 2020, pp. 27–34.
[29] H. Wang, Y. Li, A. Zhou, Y. Guo, and S. Wang, "Service migration in mobile edge computing: A deep reinforcement learning approach," *Int. J. Commun. Syst.*, vol. 106, 2020, Art. no. e4413.
[30] H. Wang, Y. Li, X. Zhao, and F. Yang, "An algorithm based on Markov chain to improve edge cache hit ratio for blockchain-enabled IoT," *China Commun.*, vol. 17, no. 9, pp. 66–76, 2020.
[31] P. Wang, Z. Chen, M. Zhou, Z. Zhang, A. Abusorrah, and A. C. Ammari, "Cost-effective and latency-minimized data placement strategy for spatial crowdsourcing in multi-cloud environment," *IEEE Trans. Cloud Comput.*, vol. 11, no. 1, pp. 868–878, Jan.-Mar. 2023.
[32] X. Wei and Y. Wang, "Popularity-based data placement with load balancing in edge computing," *IEEE Trans. Cloud Comput.*, vol. 11, no. 1, pp. 397–411, Jan.-Mar. 2023.
[33] H. Jin, R. Luo, Q. He, S. Wu, Z. Zeng, and X. Xia, "Cost-effective data placement in edge storage systems with erasure code," *IEEE Trans. Serv. Comput.*, vol. 16, no. 2, pp. 1039–1050, Mar./Apr. 2023.
[34] Y. Hao, M. Chen, H. Gharavi, Y. Zhang, and K. Hwang, "Deep reinforcement learning for edge service placement in softwarized industrial cyber-physical system," *IEEE Trans. Ind. Informat.*, vol. 17, no. 8, pp. 5552–5561, Aug. 2021.
[35] W. Fan et al., "Collaborative service placement, task scheduling, and resource allocation for task offloading with edge-cloud cooperation," *IEEE Trans. Mobile Comput.*, vol. 23, no. 1, pp. 238–256, Jan. 2024.
[36] K. Peng, L. Wang, J. He, C. Cai, and M. Hu, "Joint optimization of service deployment and request routing for microservices in mobile edge computing," *IEEE Trans. Serv. Comput.*, vol. 17, no. 3, pp. 1016–1028, May/Jun. 2024.
[37] J. Jia and P. Wang, "Low latency deployment of service-based data-intensive applications in cloud-edge environment," in *Proc. IEEE Int. Conf. Web Serv.*, 2022, pp. 57–66.
[38] X. Du, S. Tang, Z. Lu, J. Wu, K. Gai, and P. C. K. Hung, "A novel data placement strategy for data-sharing scientific workflows in heterogeneous edge-cloud computing environments," in *Proc. IEEE Int. Conf. Web Serv.*, 2020, pp. 498–507.
[39] Y. Wang, Z. Cao, X. Zhang, H. Zhou, and W. Li, "Clustering-based algorithm for services deployment in mobile edge computing environment," in *Proc. IEEE 25th Int. Conf. Parallel Distrib. Syst.*, 2019, pp. 963–966.
[40] Y. Zhao, P. Wang, H. Huang, and Z. Zhang, "Temperature matrix-based data placement using improved Hungarian algorithm in edge computing environments," in *Proc. Parallel Distrib. Comput., Appl. Technol.*, 2022, pp. 237–248.
[41] H. Zhu and M. Zhou, "Efficient role transfer based on Kuhn–Munkres algorithm," *IEEE Trans. Syst., Man, Cybern.-Part A: Syst. Humans*, vol. 42, no. 2, pp. 491–496, Mar. 2011.
[42] S. Deng et al., "Optimal application deployment in resource constrained distributed edges," *IEEE Trans. Mobile Comput.*, vol. 20, no. 5, pp. 1907–1923, May 2021.
[43] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.



Pengwei Wang (Member, IEEE) received the PhD degree in computer science from Tongji University, Shanghai, China, in 2013. He finished his postdoctoral research work with the Department of Computer Science, University of Pisa, Italy, in 2015. Currently, he is an associate professor with the School of Computer Science and Technology, Donghua University, Shanghai, China. His research interests include cloud and edge computing, services computing, and serverless. He has published more than 100 papers on premier international journals and conferences, including *IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Services Computing*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *IEEE Transactions on Automation Science and Engineering* etc.

Jingtan Jia received the bachelor's degree from Agricultural University of Hebei, in 2020 and the master's degree from Donghua University, in 2023. Her research interests include service computing, cloud computing and edge computing.

Chao Fang received the bachelor's degree from Zhejiang Sci-Tech University, in 2023. He is currently working toward the master's degree with the School of Computer Science and Technology, Donghua University. His research interests include serverless computing, cloud computing and edge computing.



Guobing Zou (Member, IEEE) received the PhD degree in computer science from Tongji University, Shanghai, China, 2012. He is a professor and Vicec Dean of the School of Computer Engineering and Science, Shanghai University, Shanghai, China. He has worked as a visiting scholar in the Department of Computer Science and Engineering, Washington University in St. Louis from 2009 to 2011, USA. His current research interests mainly focus on services computing, edge computing, data mining and intelligent algorithms, recommender systems. He has published more than 90 papers on premier international journals and conferences.

Zhijun Ding (Senior Member, IEEE) received the MS degree in computer application technology from the Shandong University of Science and Technology, Taian, China, in 2001, and the PhD degree in computer application technology from Tongji University, Shanghai, China, in 2007. He is currently a professor with the School of Computer Science and Technology, Tongji University. He has published more than 100 papers in domestic and international academic journals and conference proceedings. His research interests are in formal engineering, Petri nets, services computing, and mobile Internet.