



Service placement strategy for joint network selection and resource scheduling in edge computing

Junwei Xu¹ · Ruijuan Zheng¹  · Lei Yang² · Muhua Liu¹ · Jianqiang Song¹ · Mingchuan Zhang¹ · Qingtao Wu¹

Accepted: 15 March 2022 / Published online: 4 April 2022

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract

In the edge computing, service placement refers to the process of installing service platforms, databases, and configuration files corresponding to computing tasks on edge service nodes. In order to meet the latency requirements of new types of applications, service placement in edge computing becomes critical. The service placement strategy must be carried out in accordance with the relevant tasks within the program. However, previous research has paid little attention to related tasks within the application. If the service placement strategy does not consider task relevance, the system will frequently switch services and cause serious system overhead. In this paper, we mainly study the problem of service placement in edge computing. At the same time, we considered the issue of network access point selection during data transmission and the dependencies of task execution. We propose a Dynamic Service Placement List Scheduling (DSPLS) algorithm based on dynamic remaining task service time prediction. We conducted relevant simulation experiments, and our algorithm took the least amount of time to complete the task.

Keywords Edge computing · Network access point · Relevant tasks · Service placement

1 Introduction

Edge computing is a new computing paradigm that pushes resources such as computing, storage, and services from a centralized cloud to close to the data source [1]. Under normal circumstances, the long transmission distance of the traditional

Ruijuan Zheng, Lei Yang, Muhua Liu, Jianqiang Song, Mingchuan Zhang and Qingtao Wu contributed equally to this work.

✉ Ruijuan Zheng
zhengruijuan@haust.edu.cn

Extended author information available on the last page of the article

cloud computing architecture leads to a large network delay in the Internet, which cannot meet the delay-sensitive applications such as augmented reality [2], virtual reality [3], vehicle-in-the-car Internet systems [4], and smart grids [5]. In addition, the Internet has gradually started to use the IPV6 protocol in recent years, providing the most basic network communication conditions for the Internet of Things [6]. According to Gartner's survey, the Internet infrastructure will support 10 billion smart devices to access the Internet in 2020, an increase of 82% compared to Gartner's 2018 forecast [7]. It is estimated that by 2025, about 50 billion smart devices will be connected to the Internet [8]. These smart devices include vehicles, wearable devices, measuring sensors, household appliances, healthcare, and industrial products [9]. The data generated by these smart devices poses a crucial challenge to communication and computing technology [10]. The multiple data types generated by these applications are more suitable for processing in edge computing [11].

New types of applications that have emerged in recent years have higher requirements for time delay. One of the current challenges in edge computing is the placement of edge servers and service entities [12]. The prerequisite for the edge server to process user requests is the deployment of corresponding services. As the main deployment plan, edge servers are deployed on cellular base stations to provide users with final services [13]. In the research on service placement [14], in order to ensure service quality, the service placement model is usually divided into two parts: user offloading tasks and system service processing tasks. The service placement strategy must be carried out in accordance with the relevant tasks within the program [15]. Therefore, service placement in edge computing becomes critical. At the same time, because edge devices have the characteristics of independence and heterogeneity, how to schedule the resources required to provide services is also a problem that must be considered in edge computing.

In the work of recent years [16, 17], in order to ensure the quality of service (QoS), the problem of service placement has been solved from the aspect of resource utilization. In reality, edge computing is more used as a micro data center [18], and each edge has at least one network access point (AP) as a means of access (for example, a base station or a WiFi hotspot). In order to meet the needs of delay-sensitive applications, some studies have paid attention to the impact of network performance on the quality of service [19, 20]. In a multi-access environment, a suitable transmission network link must be considered before data transmission [21]. The heterogeneity between different service nodes leads to differences in the performance of providing services. In terms of service deployment, high-load service nodes will affect the performance of the service [22]. In previous work, research on service placement ignored the impact of network latency on service performance. In some studies that have paid attention to network access point selection, they have not paid attention to the load situation of service nodes.

In order to improve edge computing performance, the system must consider both network transmission and related service deployment [23]. In an ideal state, the user's offloading task is to select the computing node closest to him and with better network communication conditions. As shown in Fig. 1, in order to avoid excessive communication distance leading to excessive transmission delay, users



Fig. 1 Users selects the nearest edge cloud offloading task

only offload tasks to adjacent computing nodes. If the service placement strategy does not consider task relevance, the system will frequently switch services and cause serious system overhead. We use a directed acyclic graph (DAG) to represent dependencies within tasks. The relationship between the corresponding tasks is represented by the points and edges in the graph [24, 25]. We considered a scenario consisting of edge service nodes and network access points. As shown in Fig. 2, in our system model, users may be in multiple areas covered by base station signals, but the transmission signal status of each base station is different. When the user needs to request the service cloud edge, the base station with good communication status should be selected to upload the task. The tasks are uploaded to the edge cloud. The edge cloud counts the status of each node and uploads the status of each node and task type to the core cloud. The core cloud provides a service placement plan based on the task type submitted by the edge cloud and the status of each edge node, and services will be required. The part of parallel computing is divided into different service nodes in the edge cloud for processing.

Our main contributions in this paper are as follows:

1. The purpose of our proposed edge computing service placement model is to reduce the total delay of task processing and complete task requests submitted by users faster.
2. We consider the impact of network access point selection and task segmentation deployment on service placement strategies when users submit tasks.
3. We put the service processing flow into the DAG graph, and predict the remaining service time based on the remaining task volume.
4. We propose a dynamic remaining task service time prediction based on the Dynamic Service Placement List Scheduling (DSPLS) algorithm. We conducted

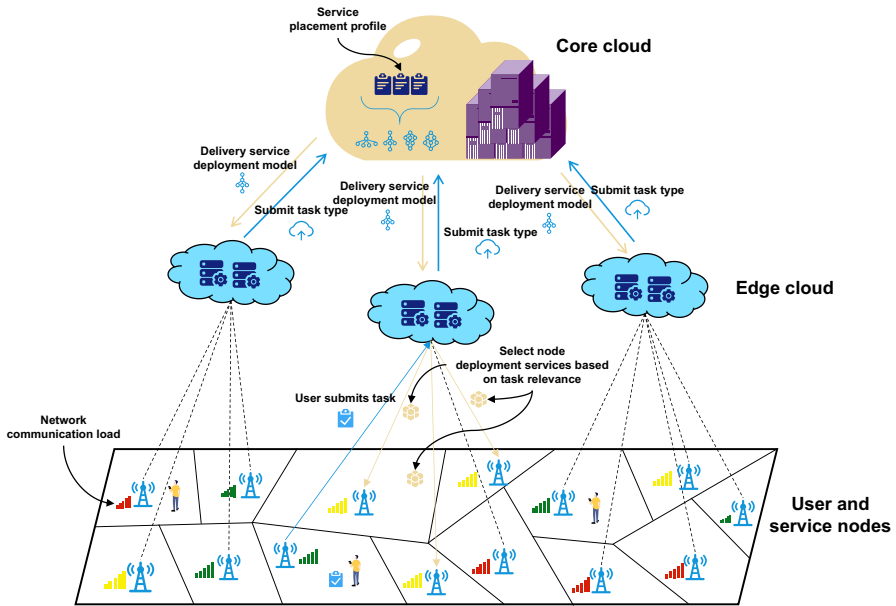


Fig. 2 The user selects a channel with good communication status to upload the task to the edge cloud. Edge cloud upload task type to obtain service deployment model. The edge cloud selects the appropriate node to deploy the service in the edge service node

relevant simulation experiments, and our algorithm took the least amount of time to complete the task.

The rest of the paper is arranged as follows. We introduce related work in Sect. 2. In Sect. 3, we introduce our system model. The algorithm is introduced in Sect. 4. In Sect. 5, we introduce related experiments and analyze the results of the experiments. In Sect. 6, we conclude this paper.

2 Related work

In traditional cloud computing, the placement of services is a relatively common problem, and it has been extensively studied. In cloud computing, the method of optimizing the placement of services reduces the time of accessing services and the communication delay to achieve network balance [26]. In order to solve the problem of limited edge computing resources, this work [27] proposed a mode of enabling edge cloud and centralized cloud cooperation for service placement. In the field of mobile edge computing, user mobility and service migration and placement have become another hot issue of research. There are some studies on the placement of edge computing services, focusing on device mobility. The work [28] is that the solution of the service placement problem of device mobility is formulated as a Markov decision process (MDP). In work [29], a service deployment strategy for user self-management was proposed. This strategy combines

user preferences to optimize the users perceived delay and service migration costs, and transforms the service deployment problem as a contextual Multi-armed Bandit (MAB) problem. The article considers task scheduling in cloud computing using dynamic scheduling [30]. However, these studies only consider the placement of services and ignore the influence of the network on the overall effect of the program.

In another part of the research, researchers have also noticed the impact of network performance on the service placement system. In addition to considering the computing and storage capabilities of service nodes, this paper [31] also considers network performance, and designs a polynomial time algorithm to solve the service placement problem. In this work [32], network performance is also considered, and an iterative-based service placement algorithm is proposed. The local-Greedy-Gen (LGG) algorithm [33] is proposed in the article, the purpose of which is to store the service content required by the user on the server closest to the user. Also in the related research of service deployment, the article proposes a method based on distributed data flow (DDF) [34] to provide the required services for the program. The article [35] proposes the Edge-ward module placement (EWMP) algorithm for service placement in the research on edge computing and fog computing. In the research on service placement in edge computing in recent years, it is proposed to deploy services based on the reliability of edge computing nodes. This article considers the high availability of services from the perspective of redundancy. The author proposes that some unexpected situations in complex environments may cause edge nodes to be unavailable. The author proposes an Reliable Redundant Services Placement (RRSP) [36] algorithm for service placement decision based on node reliability. Because the service placement environment proposed by the author is rather special, the optimal network transmission line is not considered.

The scheduling of service placement plays a vital role in overall program performance. We consider corresponding service placement for different nodes and introduce DAG graphs for research. For task processing in DAG graphs, there have been extensive researches in cloud computing. In the related research in the field of scheduling, the two scheduling algorithms of short job priority (SJF) [37] and first-come-first-served (FCFS) [38] are the most well-known classic scheduling algorithms. In this work [39], proposed a scheduling algorithm based on genetic algorithm according to the priority of tasks in the graph. For the heterogeneity of computing devices, in the work [40] proposed a new scheduling algorithm called HEFT in order to enhance the function of the heterogeneous earliest completion time (HEFT) algorithm. This paper proposes a (SDBATS) algorithm [41] that takes into account the heterogeneity of tasks and significantly reduces the overall execution time of a given application. In the research of DAG scheduling in heterogeneous environment, the author propose a new and efficient Ant-Colony based Low Delay Scheduling (ACLDS) [42] algorithm. In order to reduce the communication delay of service placement, we choose to select the network before placing the service. Secondly, in order to reduce the time overhead of switching services, according to the split of tasks, predict the remaining service time of the task on different computing nodes. We propose a Dynamic Service

Placement List Scheduling (DSPLS) algorithm based on network selection and service placement prediction.

3 System model and problem formulation

3.1 System model

The edge computing system we consider has a series of network access points NA (for example, wireless network, wired network, 4G/5G, etc.) and U users. Each edge cloud can be accessed through multiple network access points NA . We use NA and U to represent AP/edge computing cloud and users. In order not to lose generality, the system operates within a larger working time range, and we use time slots to work. Timeline representation $T = \{0, 1, 2, \dots, T\}$. In an edge cloud, we use lightweight virtualization solutions (for example, virtual machines, containers, etc.) [43] for service placement on each computing node. These lightweight virtualization programs can provide services such as computing, storage, and database. At time $t \in T$, user $u \in U$ has multiple base stations to access

Table 1 Notations used in the paper

Notation	Definition
U	A series of users in edge computing
NA	A series of network access points in edge computing
E	A series of edge computing cloud
$\gamma_u(t)$	User u selectable network access point
$\mu_{ju}(t)$	Whether to choose j network access point to connect to the edge cloud (=1) or not (=0)
$v_{iu}(t)$	Whether user u service is placed on edge cloud i (=1) or not (=0)
$\mu(t)$	Vector of a $\mu_{ju}(t)$ decision variables
$v(t)$	Vector of a $v_{iu}(t)$ decision variables
C_j	The capacity of each NA point in the edge cloud
E_p	Computing power of each edge service node
D_q	Queuing delay for access to the edge cloud network
D_c	Total communication delay to access the service
$B_{ij}(t)$	Communication delay between nodes i and j
$r_u(t)$	Provide resources needed for user u
$d_u(t)$	User u demand for network access point resources
J	Job consists of a series of tasks m
P	A series of edge servicing nodes
N	Number of edge service nodes
$l_{p_i, p_j}(t)$	The communication distance between two service nodes p_i, p_j
$c(p_i, p_j)$	The time when node p_i transmits data to p_j
$w(p_n, m_i)$	The servicing time of task m_i on service node p_n

the edge cloud through NA , denoted as $\gamma_u(t)$. The $jobs = \{0, 1, 2, \dots, m\}$ that users offload to the edge cloud consist of multiple tasks. For ease of reference, the notations used in the paper are listed in Table 1.

3.2 Network access point selection model

In each time slot t , the user needs to select the network access point before offloading the task. Here we use a binary variable to $\mu_{ju}(t)$ represent the network selection. When $\mu_{ju}(t) = 1$, it means that the user selects the $j \in \gamma_u(t)$ network access point for edge cloud access, and $\mu_{ju}(t) = 0$ otherwise. It is worth noting that the user can only select one network access point for access in each time slot t , so the following constraints are given for $\mu_{ju}(t)$:

$$\sum_{j \in \gamma_u(t)} \mu_{ju}(t) = 1, \forall u \in U, \tag{1}$$

$$\mu_{ju}(t) \in \{0, 1\}, \forall j \in \gamma_u(t), \forall u \in U. \tag{2}$$

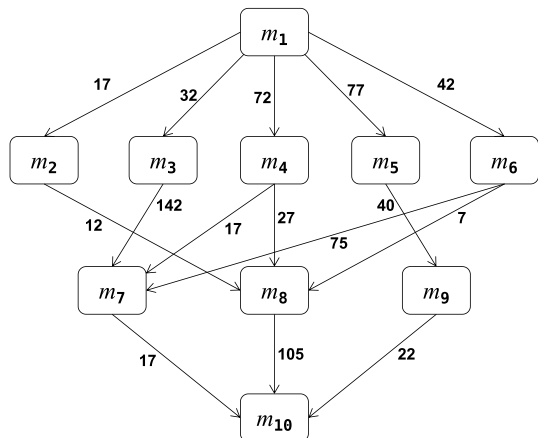
At any time, the network access point selected by the user cannot exceed the resource limit of its network access point.

$$\sum_{u \in U} d_u(t) \mu_{ju}(t) \leq C_j, \forall j \in \gamma_u(t). \tag{3}$$

3.3 Service placement model

Due to the difference in the computing capacity and storage capacity of each computing node, the length of time a task is served on different computing nodes is different. We use the service time matrix $W = J * P$ to list all possible service time

Fig. 3 DAG-based job example



mapping relationships between computing nodes p_n and tasks m_i . Table II describes an example of the service time matrix of the relevant task graph in Fig. 3. Assuming that the computing node set $P = \{p_1, p_2, p_3\}$ consists of 3 computing nodes, the elements of the mapping task J_1 and the computing node p_1 indicate that it takes time 55 to provide services for the task m_1 on the computing node p_1 .

As we all know, user u must access the edge cloud through the network access point NA . For each user u , the corresponding service is provided in the edge cloud i selected by the user. In addition, the user's network access point selection is not necessarily related to the service placement location. The service of user u can be placed on any edge cloud. However, service placement is meaningful only when user $u \in U$ accesses edge cloud $i \in E$ through network access point $j \in \gamma_u(t)$ in time slot t . Similar to the network access model, we establish a service placement model.

$$\sum_{i \in E} v_{iu}(t) = 1, \forall u \in U, \tag{4}$$

$$\sum_{u \in U} r_u(t)v_{iu}(t) \leq E_q, \forall i \in E, \tag{5}$$

$$v_{iu}(t) \in \{0, 1\}, \forall i \in E, \forall u \in U. \tag{6}$$

Ensure that services are provided by certain edge clouds. In the time slot t , the service allocated to the edge cloud cannot exceed the processing capacity of the edge cloud where it is located. We use Eq. 5 to express it. Equation 6 is used to indicate whether to place the service of user $u \in U$ on the edge cloud $i \in E$.

3.4 Network queuing delay model

The network access point changes with time and users. In some densely populated areas, certain network access points will become popular options, causing some network access points to overload. The increase in queuing delay will greatly reduce the quality of service of the application. Through research and analysis, we introduce queuing theory [44] to model the queuing delay of the core backbone network. The queue delay of the network access point NA for a series of users U in a time slot t is expressed by the following formula:

$$D_q(\mu(t)) = \sum_{u \in U} \sum_{i \in E} \mu_{ju}(t) \frac{1}{C_J - \sum_{u \in U} d_u(t)\mu_{ju}(t)}, \tag{7}$$

where C_J is the resource amount of NA J . In particular, network access point $j \in \gamma_u(t)$ satisfies the Eq. 7. Otherwise Eq. 7 could be 0 when $j \notin \gamma_u(t)$. In order to ensure that Eq. 7 is true, we assume that the resources owned by C_J meet the needs of users at all times.

3.5 Predictive service placement model in edge cloud

We consider that when users decide to offload job to an edge cloud, we process different tasks in the job on different computing nodes in the cloud. These tasks are executed in a sequential or parallel manner, so we introduce a DAG diagram to represent the service placement process in a single edge cloud.

3.5.1 Start and end tasks

First of all, we define two computing tasks, namely the start task and the last task. According to graph theory, when the in-degree is 0, we represent the starting task:

$$\text{deg}^-(m) = 0, \forall m \in J. \quad (8)$$

According to graph theory, the out-degree is 0 and we denote the starting task:

$$\text{deg}^+(m) = 0, \forall m \in J. \quad (9)$$

3.5.2 Earliest service placement time

Define the earliest time when task m starts to provide services on computing node p as the earliest service placement time (*ESPT*). $ESPT(m_i, p_n)$ represents the earliest time when task m can start providing services on computing node p , where m_i is one of the components of task m , and p_n is the edge node n serving the task. In practice, the earliest time to start providing services also depends on the completion time of previous tasks. In Eq. 10, we express *ESPT* as the completion time of the previous task plus the communication time of the previous task's output data to this node. $pre(m_i)$ is a series of pre-tasks of m_i . $PCT(m_i)$ is completion time of all predecessors of m_i .

$$ESPT(m_i, p_n) = \max \{PCT(m_i) + c(p_j, p_i)\}, \forall m_i \in J, \forall p_i, p_j \in P. \quad (10)$$

If the two tasks are on the same compute node, then $c(p_j, p_i) = 0$. Indicates that there is no network delay overhead between the data processed by the previous task and the data required by the next task.

3.5.3 Earliest service completion time

In order to better represent the service time of the computing node, we also define the earliest service completion time (*ESCT*). $ESCT(m_i, p_n)$ indicates the earliest completion time when the service provided by the computing node p_n satisfies the computing task m_i .

$$ESCT(m_i, p_n) = ESCT(m_j, p_n) + w(p_n, m_i), \forall m_i, m_j \in J, \forall p_n \in P. \quad (11)$$

3.5.4 Service scheduling length

The service scheduling length of the job, *workspan*, is the total time to provide all related task services. Workspan is calculated by Eq. 12 and is used to represent the actual completion time of the final task of the job in the DAG graph. $PCT(m_{finaltask})$ represents the actual final task completion time:

$$workspan = \max \{PCT(m_{finaltask})\}. \quad (12)$$

3.5.5 Total latency in edge cloud

The complete route (CR) is the longest path in the DAG graph, from the start task to the final task. Each edge on this path represents the transmission delay required for the job. Therefore, the total communication delay of a job in the edge cloud is expressed as follows:

$$c_v(m_i) \sum_{i,j=1}^P c(p_j, p_i), \forall m \in J, \forall p \in P. \quad (13)$$

All calculation delays of a job in the edge cloud are expressed as follows:

$$w_v(m_i) \sum_{p=1}^P w(p_n, m_i), \forall m \in J, \forall p \in P. \quad (14)$$

3.6 Problem formulation

Combined with the queue delay of the user selecting the communication path on the core network, the communication delay of working in the cloud, and the calculation delay of the task calculated at each node. We express the problem of network selection and predictive service placement as follows:

$$\begin{aligned} \min \sum_{t=1}^T D(v(t), \mu(t)) &= \sum_{t=1}^T (D_q(\mu(t)) + cv(t) + wv(t)) \\ \text{s.t. } &(1)(3)(2)(4)(5)(6). \end{aligned} \quad (15)$$

According to the model we established, the total delay of task processing under this model includes three parts: data transmission delay, communication delay between

components within edge cloud, and service time provided by edge nodes. where $D_q(\mu(t))$ is the queuing delay mentioned in Eq. 7, that is, the link delay of data transmission, $cv(t)$ is the communication delay of the task component in the edge cloud, which is defined by Eq. 13, and $wv(t)$ is the time for the task component to request resources at the edge computing node, which is Eq. 14. For each part of the modeling of the service placement problem, there are corresponding constraints, namely *s.t.*(1)(3)(2)(4)(5)(6), and our goal is to minimize the total delay under the condition that the constraints are satisfied.

4 Algorithm

When the user's work needs to be offloaded to the edge cloud, a decision needs to be made based on the total delay. It is known that the placement problem of minimizing latency for computing tasks is NP-hard [45]. The following aspects affect the performance of service placement: (1) Queue waiting time on the backbone network. (2) The communication between nodes that provide services in edge computing nodes. (3) Time to provide services on different computing nodes. (4) Provide constraint relationships between service nodes in the edge cloud. (5) The number of nodes serving in parallel in the edge cloud. (6) The number of nodes that provide associated services in the edge cloud.

In this section, we propose a Dynamic Service Placement List Scheduling (DSPLS) algorithm based on dynamic remaining task service time prediction. The algorithm provides services for tasks in the DAG graph at different nodes, determines the execution order and parallelism between tasks, and minimizes the calculation and propagation delay in the edge cloud. The algorithm calculates the service time of each node task and predicts all execution paths of the task. After that, the algorithm selects the largest task execution path and provides services at the computing node in order to optimize the execution path length of the entire job. After providing service for the previous task, the remaining path length of the task to be serviced must be updated. Therefore, the service provided by the computing node is dynamically updated according to the previous task scheduling. The algorithm is mainly composed of four components.

Component one: The transmission delay of the backbone network accounts for. The transmission delay of the data of user offloading job in the backbone network accounts for a relatively large amount of the total delay. The status of the backbone network D_q is represented by the queues in the network.

Component two: The remaining time of the task service is calculated. The remaining service time of task m should be calculated after m starts to be served, after considering all task m constraints, and then calculate the total calculation and transmission overhead.

Component three: The task to be scheduled needs to be serviced. According to task scheduling, service placement is performed in sequence. According to the calculated remaining service time of the task, select the computing node where the service is placed for the next task and start the service.

Component four: Select computing nodes for task service scheduling. For the task to be scheduled, the algorithm determines the computing node assigned to the task and the time to provide the service.

Algorithm 1 Dynamic service placement list scheduling algorithm

Input: Network load status D_q , task m_i .

Output: Service placement strategy for m_i .

- 1: $minD_q(\mu(t)) = \sum_{u \in U} \sum_{i \in E} \mu_{ju}(t) \frac{1}{C_J - \sum_{u \in U} d_u(t) \mu_{ju}(t)}$
 - 2: Generate service time matrix.
 - 3: $PRSS(m_i, p_n) \leftarrow \max\{\alpha_{i,n}, \beta_i\}$.
 - 4: Create an empty service ready status list, and put all start tasks into this table.
 - 5: **while** service ready status list is not empty **do**
 - 6: **for** each task m_i in service ready status list **do**
 - 7: $ASPL(m_i) \leftarrow \frac{\sum_{p_n \in P} ESPL(m_i, p_n)}{N}$.
 - 8: **end for**
 - 9: $m_i \leftarrow \max_{argpath \in SRSL} \{ASPL(m_j)\}$.
 - 10: $p_m \leftarrow \min_{argp_i \in P} \{ESPL(m_i, p_i)\}$.
 - 11: Update service ready status list.
 - 12: **end while**
-

4.1 Calculation of remaining service time

First, assign a weight to each task service, which is the total computing service time and communication time of subsequent services. In the system, we create a predicted remaining service schedule (PRSS) to maintain the weight value of each task on different computing nodes. In particular, the PRSS table is composed of N (different tasks) rows and M (different computing nodes) columns. $PRSS(m_i, p_n)$ represents the estimated remaining time required for all subsequent tasks of service task m_i if the task service is allocated to the computing node p_n . The weight of the service task m_i is closely related to the number of subsequent tasks to be serviced and the available computing nodes. Therefore, we use the Eq. 16 to express as:

$$\alpha_{i,n} = \max \left[\min_{m_j \in sub(m_i)} PRSS(m_j, p_n) + w(p_n, m_j) + c(t_i, t_j) \right], \tag{16}$$

where $sub(m_i)$ is the subsequent task set of task m_i , and let

$$\beta_i = \frac{\sum_{m_j \in sub(m_i)} \frac{\sum_{p_n \in P} PRSS(m_j, p_n)}{N}}{N}. \tag{17}$$

Note that no matter which computing node the final task is served on, the weight of the task is 0. Therefore, for any $p_t \in P$, $PRSS(m_{end}, p_n) = 0$.

The weight of the service task m is calculated by Eq. 18:

$$PRSS(m_i, p_n) = \max \{ \alpha_{i,n}, \beta_i \}, \forall m_i \in M, \forall p_n \in P. \quad (18)$$

From the entry task service to the exit task service, the algorithm DSPLS recursively calculates the weight value of each computing node and obtains the PRSS table.

4.2 The task to be scheduled needs to be serviced

It is not possible to schedule task m_i and provide services unless all predecessor tasks of task m_i have completed the scheduling and services. If task m_i can be serviced, we call task m_i in a service-ready state. We create a task service ready state list (SRSL) to maintain all tasks in the service ready state. At the beginning, only the start task in the DAG graph is in the service ready state. The service ready status list at this time only contains the initial start task. We calculate the earliest service placement time (ESPT) for each task in the service ready state list. The ESPT of task m_i on calculate node p_n , $ESPT(m_i, p_n)$ calculated by Eq. 10. The estimated service path length (ESPL) when task m_i is served on computing node p_n , $ESPL(m_i, p_n)$ can be calculated by Eq. 19:

$$ESPL(m_i, p_n) = ESPT(m_i, p_n) + w(m_i, p_n) + PRSS(m_i, p_n), \quad (19)$$

$$\forall m_i \in M, \forall p_n \in P.$$

For each task, services may be provided at any computing node $p_n \in P$. For the task m_i in the service ready state list, the average service path length (ASPL) of m_i is expressed $ASPL(m_i)$ by Eq. 20:

$$ASPL(m_i) = \frac{\sum_{p_n \in P} ESPL(m_i, p_n)}{N}, \forall m_i \in M. \quad (20)$$

Because the task chooses different service paths, there will be great delay differences. Therefore, the task m_i with the largest service path in the service ready list is selected for service scheduling first, which is expressed as:

$$m_i = \max [ASPL(m_j)], path \in SRSL. \quad (21)$$

In the process of the entire job being serviced, priority is given to the ESPT related tasks to provide services, so the entire ESPT will change with the previous task service scheduling. Therefore, in the service scheduling process of related tasks, the scheduling selection of task services will be dynamically changed.

4.3 Select computing nodes for service placement for tasks to be scheduled

We assign task to be scheduled m_i to computing node p_m to provide services. Such scheduling will shorten the task service path length, thereby reducing the total service time.

$$p_m = \min [ESPL(m_i, p_t)], p_t \in P. \quad (22)$$

If multiple edge computing nodes provide services for task m_i and can obtain the same minimum service path length, we will randomly select one of these edge computing nodes to provide services for task m_i . The actual service start time of task m_i is determined by Eq. 11.

After completing the assignment of computing nodes to the task m to provide services, other tasks may be ready to receive services. Therefore, we have updated the service readiness list.

4.4 Time complexity analysis

The algorithm consists of four parts, network access point selection, remaining service time, service scheduling and services node selection. In algorithm DSPLS, the core network selection mainly depends on the queue length in the network. The length of the queue depends on the number of tasks $O(P * J)$. For the remaining service time of the job, the algorithm DSPLS recursively calculates the estimated service time for each task on different computing nodes from the outgoing task to the last entry task of the DAG-based job, which requires time $O(P * J^2)$. Before each task is scheduled or scheduled for service, the ESPT of the task in the service ready list must be calculated. The maximum number of tasks in the service ready list is $(J - 2)$, and the service time of ESPT will not exceed $O(J * (J - 2) * P) = O(P * J^2)$. It takes time $O(J)$ to calculate the estimated path length of each task service in the service ready state list on all computing nodes, and it takes time $O(1)$ to calculate the average path length of each task service. Therefore, the time complexity of task service selection is $O(P * J^2 * P * 1) = O(P^2 * J^2)$. Since the ESPL of the task service has been calculated in the last task service scheduling process, it takes time $O(1)$ to determine the time for the computing node to provide services for each task; The time it takes to select computing nodes for all M task scheduling services is $O(J)$. It takes time $O(J^2)$ to update the tasks in the service ready state list, so the total time complexity of algorithm DSPLS is $O(P * J) * O(P * J^2) + O(P^2 * J^2) + O(P) + O(P^2) = O(P^2 * J^2)$.

5 Performance evaluation

In this section, we evaluate the performance of the proposed algorithm DSPLS by comparing four scheduling algorithms HEFT [41], ACLDS [42], SJF [37] and FCFS [38]. For the DAG examples shown in Fig. 3 and Table 2, the results of the five scheduling algorithms are shown in Fig. 4. In this example, we can see that the DSPLS algorithm is better than the other four algorithms in terms of scheduling completion time. This shows that, based on the constraints of the DAG graph, the algorithm DSPLS can better complete the scheduling faster in a heterogeneous environment.

Table 2 Example of service time matrix

	P_1	P_2	P_3
m_1	55	52	90
m_2	65	42	60
m_3	72	67	87
m_4	17	25	10
m_5	80	67	107
m_6	55	45	45
m_7	37	52	20
m_8	72	57	90
m_9	35	62	75
m_{10}	32	40	82

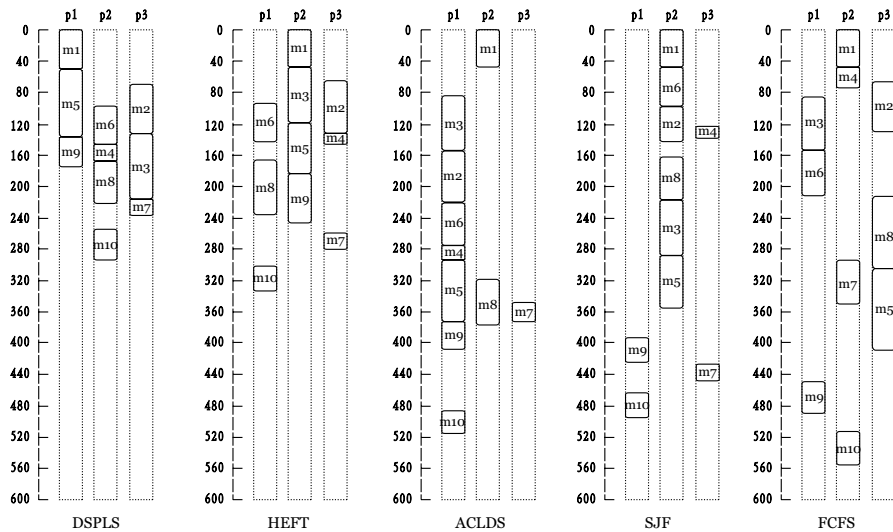


Fig. 4 Scheduling process for different algorithms for example DAG graphs

We further evaluated the performance of the five algorithms in terms of scheduling length ratio (SLR) and winning rate. The scheduling length ratio (SLR) is to obtain the ratio of the scheduling length to the minimum scheduling length by ignoring the defined communication time, as shown in Eq. 23.

$$SLR = \frac{workspan}{\sum_{m_i \in CP_{MIN}} \min [w(p_n, m_i)]}, p_n \in P. \tag{23}$$

CP_{MIN} is the minimum length of the critical path in DAG-based operations after ignoring the communication time between tasks.

5.1 Simulation setup

Our simulation experiments assume that each edge computing node is a single-core processor, so that task components can be allocated to different nodes for service requests. We divide our algorithm into two parts for evaluation. The first part compares the scheduling of DAG tasks separately. The second part is to compare performance with other service deployment algorithms in the EUA [46] data environment.

5.1.1 Randomly generate DAGs

We use 8 parameters to generate the DAG graph [47]. The following is a description of the parameters and the settings used in our experiment.

- (1) The number of tasks in the DAG graph J .
 - $J \in \{9, 10, 11, 13, 15, 27\}$
- (2) *regularity*: The number of tasks in each layer in the DAG graph.
 - $regularity \in \{0.2, 0.8\}$
- (3) *jump*: In the generated DAG graph, the edges can span the maximum span of the layer.
 - $jump \in \{1, 2, 4\}$
- (4) *density*: This parameter indicates the number of edges between the layer and the layer in the DAG graph.
 - $density \in \{0.2, 0.8\}$
- (5) *fat*: Represents the aspect ratio in the DAG graph.
 - $fat \in \{0.1, 0.4, 0.8\}$
- (6) *CCR*: This parameter represents the ratio of the average communication time between tasks in the DAG graph to the average calculation time of tasks.
 - $CCR \in \{0.1, 0.5, 0.8, 1.0, 1.3, 1.5\}$
- (7) N : Number of service nodes.
 - $N \in \{3\}$
- (8) α : This parameter indicates the difference between all the relevant tasks of computing time.

- $\alpha \in \{0.2, 0.3, 0.5, 0.6, 1.0\}$

In our simulation, we use the following parameters to randomly generate a DAG graph to test the scheduling performance of our algorithm.

5.1.2 Real-world environment

We use the EUA [46] data sets to perform simulation experiments on service placement. This repository maintains a set of EUA [46] data sets which we collected from real-world data sources. The data sets are publicly released to facilitate research in Edge Computing. The data in this data set are all from Australia. The data set includes the latitude and longitude information of the edge server and the user. As shown in Fig. 5, examples of edge service node and their coverage and end users.

5.2 Evaluate algorithm scheduling performance by randomly generating DAG graphs

Figure 6 shows the time it takes for different algorithms to complete scheduling under different number of tasks. In general, the scheduling completion time will increase as the number of tasks increases. We can see that the DSPLS algorithm achieves the shortest scheduling completion time in the scheduling completion time of different tasks. When assigning service nodes to tasks, the HEFT [41] algorithm considers the earliest completion time of the current task, and assigns tasks to the

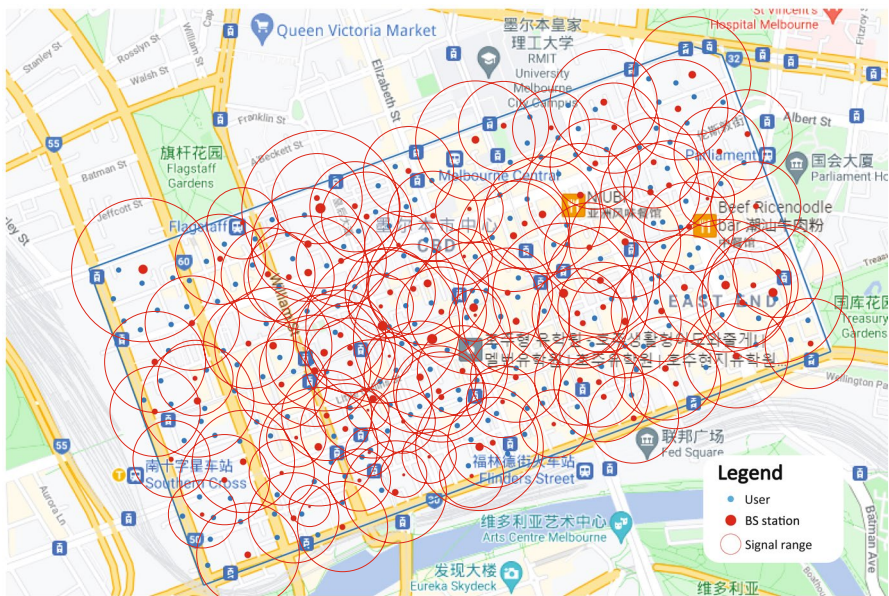


Fig. 5 Distribution of users and edge service nodes in the central business district

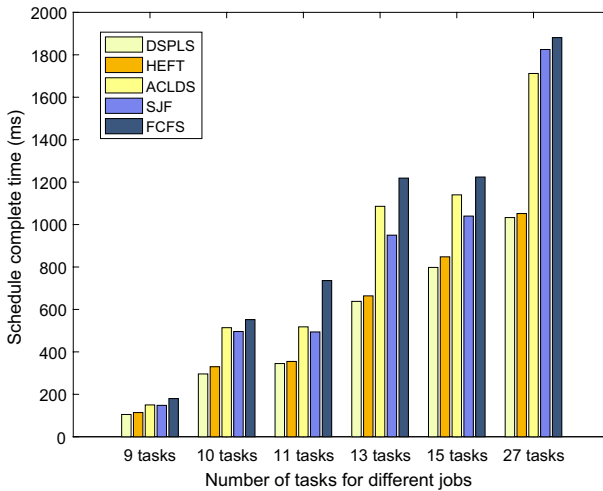


Fig. 6 Completion time of different scheduling algorithms

service nodes through the strategy of assigning priority to all tasks. However, the algorithm does not consider the impact of the current task assignment on subsequent tasks, and a certain communication delay may be lost in the process of subsequent task scheduling, resulting in an increase in the overall scheduling time. The ACLDS [42] algorithm is an efficient and low-latency scheduling algorithm based on ant colony. The advantage of this algorithm is that the decision-making speed is fast, and the operating parameters are dynamically adjusted according to the node state. First, task scheduling is prioritized by average execution time and subsequent maximum task communication and execution time. Then, by the ratio of the earliest start time of the task to the earliest completion time, it is determined which settlement node the task should be executed by. However, in the initial stage of scheduling, the algorithm tends to assign tasks to the nodes with the least execution time, ignoring the influence of communication delay on the total delay, and this algorithm is not conducive to short task scheduling.

The SJF [37] scheduling algorithm only ensures that the shortest task can be scheduled at the earliest. For long tasks, the scheduling completion time cannot be guaranteed. It can be seen from our simulation experiments that short jobs are preferentially scheduled in jobs with a small number of tasks, and the algorithm is slightly ahead of the ACLDS algorithm. However, when the number of tasks is large, the performance of ACLDS [42] algorithm is better than that of SJF. The FCFS [38] algorithm only considers the earliest time when the task arrives at the service node. This algorithm is relatively fair for task allocation, but if the task that requires a long service time is scheduled first, it will cause the following task with a short service time to take a lot of time to wait. The algorithm DSPLS creates a remaining service schedule by jointly considering the current task and all subsequent tasks. In the process of table building, according to Eq. 12 and 16, the DSPLS of all tasks on each service node takes into account the number of service nodes and

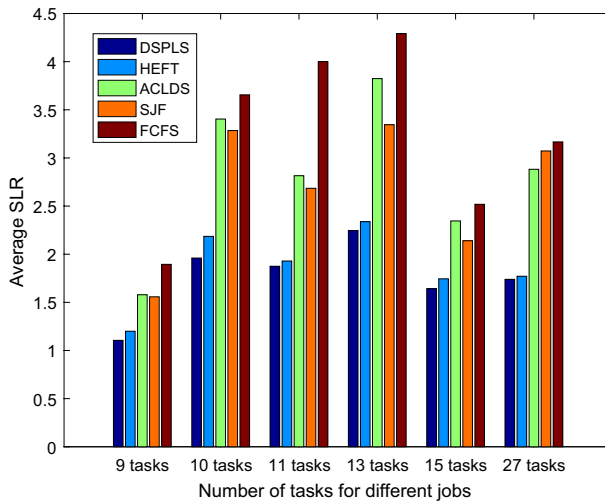


Fig. 7 Average SLR under different scheduling algorithms

the number of parallel tasks. Therefore, the DSPLS algorithm performance scheduling completion time of the algorithm is earlier than that of the HEFT [41], ACLDS [42], SJF [37] and FCFS [38] algorithms.

Figure 7 shows the average SLR state of different algorithms under different number of tasks. In the case of the same number of service nodes but different tasks, the average SLR value of the DSPLS algorithm is always the lowest. This situation is because the DSPLS algorithm improves the scheduling performance by comprehensively considering the relationship between the current task, the status of the service node, and the subsequent tasks. The remaining four algorithms do not consider the status of subsequent tasks and the communication delay during scheduling, which leads to higher average SLR. Similarly, we can also see from SLR that the scheduling performance of SJF [37] algorithm is better than ACLDS [42] when the number of tasks is small. Once, the number of tasks increases, the performance of ACLDS [42] algorithm is better than that of SJF [37]. From the experimental results, this situation is in line with the respective characteristics of the two algorithms.

In Fig. 8, we randomly generated a series of DAG graphs to simulate the total time spent by different algorithms when scheduling a batch of tasks. When scheduling tasks in batches, the total scheduling time of the DSPLS algorithm is always less than that of other scheduling algorithms. In a batch environment, the scheduling differences between algorithms are magnified to facilitate comparison of the scheduling performance of each algorithm. The DSPLS algorithm comprehensively considers the current task, the current service node status and subsequent related tasks to reduce the waiting time of the task as much as possible. The other four algorithms are only scheduled for the current task, ignoring subsequent related tasks and service node status. It is worth noting that the total scheduling delay of the ACLDS [42] algorithm in the batch environment is always higher than that of the SJF [37]

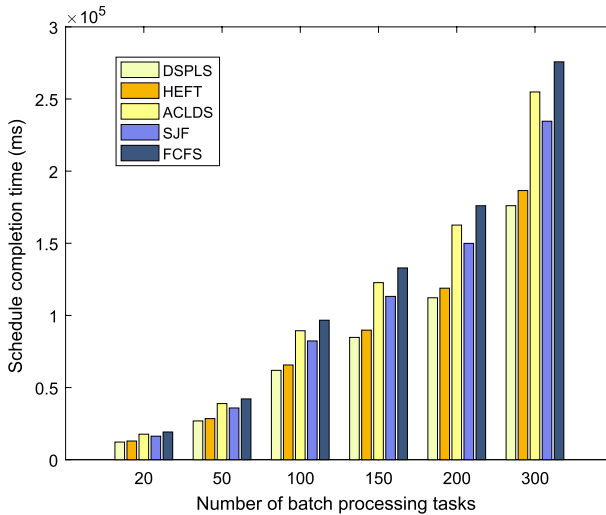


Fig. 8 The time for each scheduling algorithm to complete scheduling in a batch processing environment

algorithm. The reason for this is because of the problem of our experimental environment. There are many small tasks generated in our experimental environment, which leads to the poor performance of the ACLDS [42] algorithm. However, this does not mean that the ACLDS [42] algorithm is not good. In the original text, the experimental environment of the ACLDS [42] algorithm focuses on testing the scheduling performance of hundreds of tasks. The experimental results also show that different scheduling algorithms have their best performance execution scenarios.

5.3 Service placement performance evaluation in the EUA data set environment

At this stage, there are various devices connected to the Internet. We assume that in the future, each device will have multiple ways to access the Internet. In this case, there is a problem of network access point selection. Therefore, we have added network access point selection to the algorithm DSPLS. We propose the network access point selection to reduce the transmission delay of user upload tasks. We calculate the network delay in different algorithms according to time slot statistics. As shown in Fig. 9, we compared several network selection strategies [48–50]. According to Eq. 7 of queuing theory, it can be seen that our network selection strategy is the smallest average delay.

Among them, the worst algorithm is to select the worst transmission channel for each transmission, as the baseline of the network selection performance. Algorithm GRN [49] is to randomly select the network channel. The advantage of this algorithm is that data transmission is performed without considering the channel state, but it may select a channel with a normal or poor signal state for communication, resulting in poor average network performance. The (SR)ARQ [48] algorithm uses a polling method for network channel transmission. The channel selected each time is different from the previous one. This algorithm ensures the balance of the

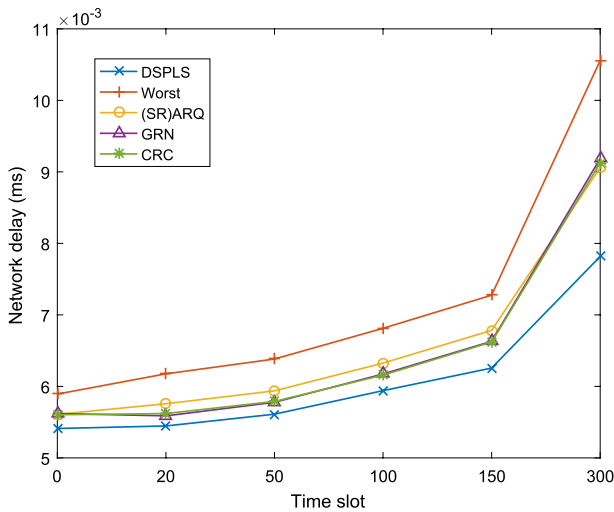


Fig. 9 Network delay under different network selection strategies

transmission line load but cannot guarantee that the best transmission channel can be selected every time. The principle of the CRC [50] algorithm is to select the network transmission channel through the hash value, and its purpose is also to ensure the balance of all channels in the network. The network selection strategy we propose through queuing theory is to select the transmission line with the smallest delay each time based on the transmission line determined by the current channel state. Therefore, our algorithm guarantees the transmission performance of the network.

In the EUA [46] data set environment, we compared LGG [33] algorithm, DDF [34] algorithm, EWMP [35] algorithm and RRSP [36] algorithm. We use randomly generated tasks and perform performance comparisons through different service placement strategies. Our main task from the total service time to evaluate the performance of the algorithm. The service time includes the transmission time of the user's transmission task to the edge service node and the service time of the service node.

Figure 10 shows the total service time of different algorithms in batch task processing in the EUA [46] environment. Generally speaking, as the number of tasks increases, the overall service time for the tasks will also increase. The LGG [33] algorithm will select the data transmission network before transmitting the task, and the data transmission delay is relatively low. However, the LGG [33] algorithm selects the service node closest to the user to provide the service. This causes a single service node to be in a high load state for a long time, and the waiting time for subsequent tasks is too long before being served. In the DDF [34] algorithm, no network selection is performed before the data is transmitted, and the link is randomly selected for data transmission, and the transmission delay is relatively large. The algorithm uses randomly selected multiple nodes to deploy services at the same time in the service placement strategy to improve the parallelism of services. The EWMP [35] algorithm is consistent with the DDF [34] algorithm in network selection.

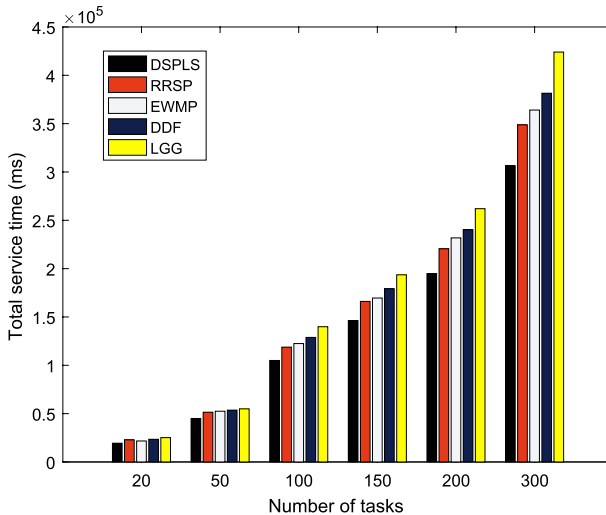


Fig. 10 Comparison of service placement algorithms in a batch processing environment

However, in the selection of service nodes, the EWMP [35] algorithm will first evaluate and select appropriate nodes for service placement. The RRSP [36] algorithm is mainly for service placement decisions in a specific environment. The RRSP [36] algorithm is mainly to ensure the high availability of the service, and to place the service by evaluating the reliability of the nodes. Through reliability evaluation, the algorithm tends to provide more services at reliable nodes. Normally, reliable nodes have more resources and can handle more task requests. Because it is in a special environment, there may be no conditions for selecting a network link. Therefore, a part of the communication performance is lost in our experimental environment. We can see from Fig. 10 that the performance of the RRSP [36] algorithm is slightly lower than that of the EWMP [35] algorithm in the environment of 20 tasks. As the number of tasks increases, the RRSP [36] algorithm exhibits the performance due to the EWMP [35] algorithm. The DSPLS algorithm we proposed not only includes network selection, but also fully considers the DAG dependency of the task in the process of service placement, and selects multiple service nodes for service placement. This not only reduces the time delay of data transmission, but also reduces the service delay required to complete the task.

6 Conclusion

In this paper, we researched the issue of service placement in edge computing. We improve system performance from two aspects: reducing transmission delay and reducing service scheduling time. We first select the network link before transmitting data to ensure the efficiency of data transmission. Then, in view of the heterogeneity of edge service nodes, we select the appropriate node to place the corresponding service. We combined network access point selection and DAG related

dependencies, and proposed a service placement algorithm. We conducted simulation experiments on the two parts of the algorithm, network selection and DAG scheduling, to verify the performance of our algorithm.

Many scenarios in this article are idealized modeling based on assumptions. In future research, we will reduce assumptions and make the model closer to real life. For example, when the task component is being served, the state of the edge computing node changes, how to adjust the service placement strategy. The splitting and distribution of task components should be more intelligent. Not necessarily all tasks need to be split to different nodes for execution, and multi-core devices can also complete tasks in parallel. With the development of the Internet of Things and edge computing, our solutions provide a basis for related research in the future.

Acknowledgements This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grants no.62172142, no. 61602155, and in part by the Scientific and Technological Innovation Team of Colleges and Universities in Henan Province under Grants No. 20IRTSTHN018, and in part by the Key Technologies R & D Program of Henan Province under Grants No. 202102210169 and No. 212102210088, and in part by the Leading Talents of Zhongyuan Science and Technology Innovation under Grants No. 214200510012.

References

1. Wu D, Huang X, Xie X, Nie X, Bao L, Qin Z (2021) Ledge: leveraging edge computing for resilient access management of mobile iot. *IEEE Trans Mob Comput* 20(3):1110–1125. <https://doi.org/10.1109/TMC.2019.2954872>
2. Alhajja HA, Mustikovele SK, Mescheder L, Geiger A, Rother C (2017) Augmented reality meets computer vision: efficient data generation for urban driving scenes. *Int J Comput Vis* 2:1–12
3. Lai Z, Hu YC, Cui Y, Sun L, Dai N, Lee H-S (2020) Furion: engineering high-quality immersive virtual reality on today's mobile devices. *IEEE Trans Mob Comput* 19(7):1586–1602. <https://doi.org/10.1109/TMC.2019.2913364>
4. Xu W, Song H, Hou L, Zheng H, Zhang X, Zhang C, Hu W, Wang Y, Liu B (2021) Soda: Similar 3d object detection accelerator at network edge for autonomous driving. In: *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pp. 1–10. <https://doi.org/10.1109/INFOCOM42981.2021.9488833>
5. Colak I, Bayindir R, Sagioglu S (2020) The effects of the smart grid system on the national grids. In: *2020 8th International Conference on Smart Grid (icSmartGrid)*, pp. 122–126. <https://doi.org/10.1109/icSmartGrid49881.2020.9144891>
6. Paul PV, Saraswathi R (2017) The internet of things a comprehensive survey. In: *2017 International Conference on Computation of Power, Energy Information and Commuication (ICCPEIC)*, pp. 421–426. <https://doi.org/10.1109/ICCPEIC.2017.8290405>
7. Alfonso Velosa HLJFHSRM (2015) Earl Perkins: Predicts 2015: The Internet of Things. <https://www.gartner.com/en/documents/2952822>
8. Vestberg H (2018) CEO to shareholders: 50 billion connections 2020. Preprint at <https://www.ericsson.com/en/press-releases/2010/4/ceo-to-shareholders-50-billion-connections-2020>
9. Zeng J, Banerjee I, Gensheimer M, Rubin D (2020) Cancer treatment classification with electronic medical health records (student abstract). *Proc AAAI Conf Artif Intell* 34(10):13981–13982. <https://doi.org/10.1609/aaai.v34i10.7263>
10. Wan Y, Xu K, Xue G, Wang F (2020) Iotargos: A multi-layer security monitoring system for internet-of-things in smart homes. In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pp. 874–883. <https://doi.org/10.1109/INFOCOM41043.2020.9155424>

11. Rafique W, Qi L, Yaqoob I, Imran M, Rasool RU, Dou W (2020) Complementing iot services through software defined networking and edge computing: a comprehensive survey. *IEEE Commun Surv Tutor* 22(3):1761–1804. <https://doi.org/10.1109/COMST.2020.2997475>
12. Liang Y, Ge J, Zhang S, Niu C, Song W, Luo B (2020) Efficient service entity chain placement in mobile edge computing. In: 2020 16th International Conference on Mobility, Sensing and Networking (MSN), pp. 182–189. <https://doi.org/10.1109/MSN50589.2020.00042>
13. Zhang Y, He J, Guo S (2018) Energy-efficient dynamic task offloading for energy harvesting mobile cloud computing. In: 2018 IEEE International Conference on Networking, Architecture and Storage (NAS), pp. 1–4. <https://doi.org/10.1109/NAS.2018.8515736>
14. Taleb T, Samdanis K, Mada B, Flinck H, Dutta S, Sabella D (2017) On multi-access edge computing: a survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Commun Surv Tutor* 19(3):1657–1681. <https://doi.org/10.1109/COMST.2017.2705720>
15. Zhao G, Xu H, Zhao Y, Qiao C, Huang L (2020) Offloading dependent tasks in mobile edge computing with service caching. In: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, pp. 1997–2006. <https://doi.org/10.1109/INFOCOM41043.2020.9155396>
16. Phan TK, Rocha M, Griffin D, Rio M (2018) Utilitarian placement of composite services. *IEEE Trans Netw Serv Manag* 15(2):638–649. <https://doi.org/10.1109/TNSM.2018.2798413>
17. Mosa A, Sakellariou R (2019) Dynamic virtual machine placement considering cpu and memory resource requirements. In: 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), pp. 196–198. <https://doi.org/10.1109/CLOUD.2019.00042>
18. Yu Y (2016) Mobile edge computing towards 5g: Vision, recent progress, and open challenges. *China Commun* 13(Supplement2):89–99. <https://doi.org/10.1109/CC.2016.7833463>
19. Poularakis K, Llorca J, Tulino A.M, Taylor I, Tassiulas L (2019) Joint service placement and request routing in multi-cell mobile edge computing networks. In: IEEE INFOCOM 2019—IEEE Conference on Computer Communications, pp. 10–18. <https://doi.org/10.1109/INFOCOM.2019.8737385>
20. Garcia-Saavedra A, Iosifidis G, Costa-Perez X, Leith DJ (2018) Joint optimization of edge computing architectures and radio access networks. *IEEE J Sel Areas Commun* 36(11):2433–2443. <https://doi.org/10.1109/JSAC.2018.2874142>
21. Chaaban A, Maier H, Sezgin A, Mathar R (2016) Three-way channels with multiple unicast sessions: Capacity approximation via network transformation. *IEEE Trans Inf Theory* 62(12):7086–7102. <https://doi.org/10.1109/TIT.2016.2614318>
22. Loghini D, Ramapantulu L, Teo Y.M(2017) On understanding time, energy and cost performance of wimpy heterogeneous systems for edge computing. In: 2017 IEEE International Conference on Edge Computing (EDGE), pp. 1–8. <https://doi.org/10.1109/IEEE.EDGE.2017.10>
23. Horner, L.J.: Edge strategies in industry: Overview and challenges. *IEEE Transactions on Network and Service Management*, 1–1 (2021). <https://doi.org/10.1109/TNSM.2021.3092940>
24. Arabnejad H, Barbosa J (2012) Fairness resource sharing for dynamic workflow scheduling on heterogeneous systems. In: 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications, pp. 633–639. <https://doi.org/10.1109/ISPA.2012.94>
25. Xie G, Zeng G, Li R, Li K (2017) Energy-aware processor merging algorithms for deadline constrained parallel applications in heterogeneous cloud computing. *IEEE Trans Sustain Comput* 2(2):62–75. <https://doi.org/10.1109/TSUSC.2017.2705183>
26. Piao J.T, Yan J (2010) A network-aware virtual machine placement and migration approach in cloud computing. In: 2010 Ninth International Conference on Grid and Cloud Computing, pp. 87–92. <https://doi.org/10.1109/GCC.2010.29>
27. Beraldi R, Mtibaa A, Alnuweiri H (2017) Cooperative load balancing scheme for edge computing resources. In: 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), pp. 94–100. <https://doi.org/10.1109/FMEC.2017.7946414>
28. Alasmari KR, Green RC, Alam M (2018) Mobile edge offloading using markov decision processes. In: Liu S, Tekinerdogan B, Aoyama M, Zhang L-J (eds) *Edge Computing - EDGE 2018*. Springer, Cham, pp 80–90
29. Ouyang T, Li R, Chen X, Zhou Z, Tang X (2019) Adaptive user-managed service placement for mobile edge computing: An online learning approach. In: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, pp. 1468–1476. <https://doi.org/10.1109/INFOCOM.2019.8737560>
30. Fan Y, Tao L, Chen J (2019) Associated task scheduling based on dynamic finish time prediction for cloud computing. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), pp. 2005–2014. <https://doi.org/10.1109/ICDCS.2019.00198>

31. Farhadi V, Mehmeti F, He T, Porta T.L, Khamfroush H, Wang S, Chan K.S. (2019) Service placement and request scheduling for data-intensive applications in edge clouds. In: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, pp. 1279–1287. <https://doi.org/10.1109/INFOCOM.2019.8737368>
32. Gao B, Zhou Z, Liu F, Xu F (2019) Winning at the starting line: Joint network selection and service placement for mobile edge computing. In: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, pp. 1459–1467. <https://doi.org/10.1109/INFOCOM.2019.8737543>
33. Borst S, Gupta V, Walid A (2010) Distributed caching algorithms for content distribution networks. In: 2010 Proceedings IEEE INFOCOM, pp. 1–9. <https://doi.org/10.1109/INFOCOM.2010.5461964>
34. Giang N.K, Blackstock M, Lea R, Leung V.C.M (2015) Developing iot applications in the fog: A distributed dataflow approach. In: 2015 5th International Conference on the Internet of Things (IOT), pp. 155–162. <https://doi.org/10.1109/IOT.2015.7356560>
35. Gupta H, Vahid Dastjerdi A, Ghosh S.K, Buyya R (2016) ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience* **47**(1)
36. Huang H, Zhang H, Guo T, Guo J, He C (2019) Reliable redundant services placement in federated micro-clouds. In: 2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS), pp. 446–453. <https://doi.org/10.1109/ICPADS47876.2019.00070>
37. Bo MA, Zahle TU (1977) Scheduling according to job priority with prevention of deadlock and permanent blocking. *Acta Informatica* **8**(2):153–175
38. Peha J.M, Tobagi F.A (1990) Evaluating scheduling algorithms for traffic with heterogeneous performance objectives. Evaluating scheduling algorithms for traffic with heterogeneous performance objectives
39. Kaur R, Singh G (2012) Genetic algorithm solution for scheduling jobs in multiprocessor environment. In: 2012 Annual IEEE India Conference (INDICON), pp. 968–973. <https://doi.org/10.1109/INDCON.2012.6420757>
40. Wang G, Guo H, Wang Y (2015) A novel heterogeneous scheduling algorithm with improved task priority. In: 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, pp. 1826–1831. <https://doi.org/10.1109/HPCC-CSS-ICSS.2015.48>
41. Munir E.U, Mohsin S, Hussain A, Nisar M.W, Ali S (2013) Sdbats: A novel algorithm for task scheduling in heterogeneous computing systems. In: 2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum, pp. 43–53
42. Yang P, Liu J, Chen C, Ding Y, Deng C, Zeng Z (2020) An efficient low delay task scheduling algorithm based on ant colony system in heterogeneous environments. In: 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 519–524. <https://doi.org/10.1109/HPCC-SmartCity-DSS50907.2020.00064>
43. Pujolle G (2020) Virtualization. In: *Software Networks: Virtualization, SDN, 5G, and Security*, pp. 1–12. <https://doi.org/10.1002/9781119694748.ch1>
44. Wu J, Wong E.W.M, Chan Y.-C, Zukerman M (2017) Energy efficiency-qos tradeoff in cellular networks with base-station sleeping. In: GLOBECOM 2017 - 2017 IEEE Global Communications Conference, pp. 1–7. <https://doi.org/10.1109/GLOCOM.2017.8254042>
45. Lau T.L, Tsang E.P.K (1998) The guided genetic algorithm and its application to the generalized assignment problem. In: *Proceedings Tenth IEEE International Conference on Tools with Artificial Intelligence (Cat. No.98CH36294)*, pp. 336–343. <https://doi.org/10.1109/TAI.1998.744862>
46. PhuLai: *EUA Datasets*. <https://github.com/swinedge/eua-dataset> (2020)
47. Cordeiro D, Mouni G, Perarnau S, Trystram D, Vincent J.-M, Wagner F (2010) Random graph generation for scheduling simulations. In: *Proceedings of 3rd International ICST Conference on Simulation Tools and Techniques*
48. Nannicini S, Pecorella T (1998) Performance evaluation of polling protocols for data transmission on wireless communication networks. In: *ICUPC '98. IEEE 1998 International Conference on Universal Personal Communications. Conference Proceedings (Cat. No.98TH8384)*, vol. 2, pp 1241–12452
49. Krapivsky PL, Redner S, Leyvraz F (2000) Connectivity of growing random networks. *Phys Rev Lett* **85**(21):4629–4632

50. Jain R (1992) A comparison of hashing schemes for address lookup in computer networks. *IEEE Trans Commun* 40(10):1570–1573

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Junwei Xu¹ · Ruijuan Zheng¹  · Lei Yang² · Muhua Liu¹ · Jianqiang Song¹ · Mingchuan Zhang¹ · Qingtao Wu¹

Junwei Xu
190319050221@stu.haust.edu.cn

Lei Yang
citicyanglei@163.com

Muhua Liu
liumuhua@haust.edu.cn

Jianqiang Song
9943517@haust.edu.cn

Mingchuan Zhang
zhang_mch@haust.edu.cn

Qingtao Wu
wqt8921@haust.edu.cn

- ¹ School of Information Engineering, Henan University of Science and Technology, 263 Kaiyuan Avenue, Luoyang 471000, Henan, China
- ² Information Technology Management Center and the Big Data Center, CITIC Heavy Industry, 206 Jianshe Road, Luoyang 471023, Henan, China