Joint Data Collection and Resource Allocation for Distributed Machine Learning at the Edge

Min Chen, Haichuan Wang, Zeyu Meng, Hongli Xu, *Member, IEEE,* Yang Xu, Jianchun Liu, He Huang, *Member, IEEE*

Abstract—Under the paradigm of edge computing, the enormous data generated at the network edge can be processed locally. To make full utilization of these widely distributed data, we focus on an edge computing system that conducts distributed machine learning using gradient-descent based approaches. To ensure the system's performance, there are two major challenges: how to collect data from multiple data source nodes for training jobs and how to allocate the limited resources on each edge server among these jobs. In this paper, we jointly consider the two challenges for distributed training (without service requirement), aiming to maximize the system throughput while ensuring the system's quality of service (QoS). Specifically, we formulate the joint problem as a mixed-integer non-linear program, which is NP-hard, and propose an efficient approximation algorithm. Furthermore, we take service placement into consideration for diverse training jobs and propose an approximation algorithm. We also analyze that our proposed algorithm can achieve the constant bipartite approximation under many practical situations. We build a test-bed to evaluate the effectiveness of our proposed algorithm in a practical scenario. Extensive simulation results and testing results show that the proposed algorithms can improve the system throughput 56%-69% compared with the conventional algorithms.

Index Terms—Edge computing, model training, job assignment, resource allocation, service placement

1 Introduction

With the advent of paradigms like the Internet of Things (IoT), social networking, smart city and industry 4.0, data will be abundantly available [2]. Cisco, for example, estimates that the Internet of Things (IoT) alone will generate over 400ZB data annually by 2020 [3]. The majority of these data is generated at the network edge, where the working world of devices, sensors, actuators, and other IoT devices are coming into online every day.

To extract useful information from the collected data of various applications, machine learning (ML) methods are often adopted. For example, Google Cloud Speech is powered by the ML framework, TensorFlow [4], with the public data from its users. Since training a machine learning model is resource-intensive, it is naturally required to send the data to the remote cloud with sufficient resources for model training [5] [6]. However, this training scheme has two main disadvantages. On one hand, due to the limited bandwidth of the wide area network (WAN), it will introduce heavy burden to the network if large amounts of data are transmitted through WAN, resulting in undesired latency. On the other hand, transmitting data through WAN may lead to privacy leakage, which contradicts with the increasing concern about users' data privacy. Therefore, it is impractical and often unnecessary to send the raw data to the remote cloud for model training.

Edge computing [7], as a new computing paradigm, has emerged to move data and service functions from the remote cloud to the network edge. As a result, data can be processed locally on edge servers, such as base stations or home gateways. Compared to cloud computing, edge computing has two significant advantages. First, since the raw data are only transmitted to the nearby edge servers, *i.e.*, without transmitting to the remote cloud, the latency, as well as the network overhead (*e.g.*, bandwidth consumption) is reduced. Second, as the raw data no longer need to be transmitted over WAN, user privacy can be abundantly preserved [8].

Thus, it is an attractive way to train the model locally under the edge computing paradigm. Some previous works [9] [10] mainly focus on how to train the machine learning models only with the local data. However, the amount of data on a single device is often too small to make a model reach desired accuracy. Thus, data from different devices are used to train a mutual model, and model training is often performed in a distributed manner. In a practical edge computing system, there are often multiple training tasks and most data source nodes such as cameras are not equipped with sufficient computation capacity. Even though the computation capacity of some devices (*e.g.*, mobile phones) is sufficient to

- *Some preliminary results of this paper were published in the Proceedings of IEEE HPCC 2019 [1]*

- *M. Chen, H. Wang, H. Xu and Y. Xu are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China, 230027, and also with Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou, Jiangsu, China, 215123. E-mails: mchen330@mail.ustc.edu.cn; none@mail.ustc.edu.cn; xuhongli@ustc.edu.cn; xuyangcs@ustc.edu.cn.*
- *Z. Meng is with the School of Cyberspace Security, University of Science and Technology of China, Hefei, Anhui, China, 230027. E-mails: sa516212@mail.ustc.edu.cn*
- *J. Liu is with the School of Data Science, University of Science and Technology of China, Hefei, Anhui, China, 230027. E-mails: jsen617@mail.ustc.edu.cn*
- *H. Huang is with the School of Computer Science and Technology, Soochow University, Suzhou, Jiangsu, China, 215006. Email: huangh@suda.edu.cn.*

perform the distributed training tasks [11], they might still expect their data to be processed on the edge servers for various concerns such as battery life. Therefore, it is often required to collect data from the geographically distributed data source nodes (*e.g.*, surveillance cameras) to edge servers for a training task. We consider the scenario where workers, deployed on the edge servers, collect the data and train the model in a distributed scheme.

We take the gait recognition application as an example. Basically, a gait recognition application consists of two tasks: collecting data from multiple sensors (*e.g.*, cameras, ultrasonic sensors) to edge servers and training the personal gait models. In practice, different training jobs will be posted by users. For example, the face or gait recognition applications usually update models using fresh data from data nodes, so that they can provide services with high accuracy to adapt to the new environment. The face recognition model for face recognition for newcomers' indoor position tracking needs to be retrained periodically using recently collected data. However, due to limited resources (*e.g.*, memory, CPU, bandwidth) in edge computing, it may be impossible to serve all the training jobs simultaneously. Otherwise, the training delay will be large or even unacceptable.

We expect to maximize system throughput by training more machine learning models. There are two critical challenges for job admission decisions. On the one hand, due to the limited bandwidth resources of edge servers, it may be difficult to collect the raw data of all training tasks effectively. Thus, the first challenge is how to admit more training requests with bandwidth constraints on edge servers. On the other hand, when we try to admit a model training job, we will deploy workers on some edge servers. Since some training tasks may be performed on an edge server in parallel, the limited resources (*e.g.*, CPU and outbound bandwidth) of an edge server have to be shared among several tasks. Inappropriate resource allocation may lead to a long latency of some jobs, decreasing the system's QoS. For example, before tracking the newcomers, we need to allocate resources to retrain or fine-tune to recognize newcomers with a certain accuracy. Otherwise, failure to complete model training on time will result in loss position tracking.

We further consider a more practical scenario, in which training a job requires some specific services. For example, when fine-tuning the face recognition model for indoor position tracking on a certain edge server, the pre-trained model will be updated by the collected data under the training environment. Thus, the pre-trained model and training environments need to be placed in this edge server. Otherwise, the fine-tuning process will not be executed.

In this paper, we study how to efficiently implement the distributed model training by joint data collection and resource allocation in edge computing. The main contributions of this paper are summarized as follows:

- We jointly consider the data collection and resource allocation problem and comprehensively formalize this problem to maximize the system throughput while ensuring the system's QoS.
- We develop an efficient algorithm based on reformulation and randomized rounding, and prove the approximation performance based on the central limit theorem.
- We further take the service placement into our problem, and propose an efficient algorithm based on filter and rounding method to maximize the system throughput. Our algorithm guarantees an approximation ratio of $\frac{3\log n}{\alpha}+3$ for system throughput, where n is the number of devices in the system, α is a value depending on system capacity and the minimum resource requirement, with $\alpha > 1$ under most practical situations.
- We implement our proposed algorithms on a test-bed platform. Extensive simulations are conducted to validate high efficiency of the proposed algorithms. The simulation results show that our proposed algorithms can improve the system throughput 56%-69% compared with the existing algorithms.

2 Preliminaries

2.1 Distributed Machine Learning (DML)

Machine learning has been successfully adopted in many applications such as image recognition [12], natural language processing [13], and recommendation systems [14]. However, with more and more data, the size of training data set has grown, which may lead to unacceptable training time by traditional solutions. Moreover, a growing number of latency-sensitive applications require training models in a fast manner. Due to the limited bandwidth, centralized model training cannot meet the latency requirements of these applications. As a result, training models in a distributed scheme is proposed [15].

In edge computing, enormous data are being generated by a large number of IoT devices and social networking applications. Obviously, it is unrealistic to store all the data on a single edge server. Thus, we adopt data parallelism for model training in this paper. Moreover, we do not focus on how to train the machine learning model. For ease of description, we take advantages of the widely adopted stochastic gradient descent (SGD) method and *Parameter Server* (PS) framework [16]. We should note that the proposed algorithms can be easily integrated into other training methods and frameworks.

We assume that one or several parameter servers are responsible for the global model update/maintenance and worker coordination [16]. When there are multiple parameter servers, the model parameters are usually divided evenly among them. Parameter servers can be implemented either at the edge or on the cloud. During the training, the workers compute the local gradients and send them to the parameter servers in each global aggregation iteration. The parameter servers update the global parameters using the gathered local gradients, and send the updated parameters back to the workers.

2.2 System Model

For ease of expression, we list some important notations in Table 1. As illustrated in Fig. 1, there are a set of data nodes $\mathcal{U} = \{u_1, u_2, \cdots, u_n\}$ with $n = |\mathcal{U}|$, and a set of edge servers $\mathcal{V} = \{v_1, v_2, \cdots, v_m\}$ with $m = |\mathcal{V}|$. Every data node $u_i \in \mathcal{U}$ generates and uploads data at a fixed rate u_i^d. To provide

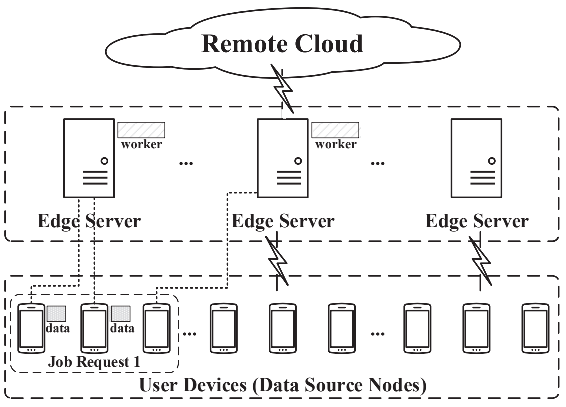


Fig. 1. The System Model

a sufficient amount of data for training a local model, data node $u_i \in \mathcal{U}$ will generate data for a duration T_i continuously. We consider four type of resources, inbound bandwidth, outbound bandwidth, computation capacity, and memory space, on an edge server. For each edge server $v_j \in \mathcal{V}$, the amount of its available type-γ resource is r_j^γ. For example, the amount of inbound bandwidth, computation capacity and outbound bandwidth of edge server v_j is denoted as r_j^d, r_j^c and r_j^b respectively. There is a set of job requests $\Phi = \{\phi_1, \phi_2, \cdots, \phi_q\}$, with $q = |\Phi|$. Before knowledge inferences, the model should be trained to achieve a certain accuracy. We hope that each job request ϕ_k should be completed within an upper bound τ_k. That is, the completion time of job ϕ_k should not exceed τ_k for real-time requirement [17]. In practice, the data nodes usually are sensors (e.g., cameras, ultra-sonic sensors). As a result, each data node only generates one kind of data for a specific job [18]. For example, cameras only provide video streams. We assume that each data node can only serve one job request, and use C_k to denote the data nodes serving job request ϕ_k. Furthermore, we discuss the solution for each data node serving multiple jobs in Section 3.5.

To facilitate these job requests, edge servers will collect the data and train the model (via virtual machines or containers). Due to some factors such as distance, each data node u_i has its own feasible edge servers. Specifically, we denote the feasible edge servers of data node u_i as $\mathcal{V}_i$. During data collection, each data node will only forward its data to one of its feasible edge servers for model training. We call that each data node will be associated with an edge server. The subset of candidate data nodes that can be associated with v_j is denoted as $\mathcal{U}_j$. We define the degree $B(u_i)$ of data node nodes u_i as the number of its feasible edge servers $|\mathcal{V}_i|$. Similarly, the degree $B(v_j)$ of edge server v_j is the number of its candidate data nodes $|\mathcal{U}_j|$.

We collect all job requests in the system and make a unified decision. Some of the job requests will be admitted by the system and allocated a certain amount of resources. The remaining job requests will be dropped temporarily due to the lack of resources. After all the admitted job requests are completely executed, we will make a decision for the remaining and the newly-arrived job requests.

2.3 Data Collection

We use binary variable y_k to represent the admission decision of job request ϕ_k. That is, if $y_k = 1$, it will be admitted and served by the system. Otherwise it is rejected. Rejected job requests will be resubmitted or queued. We use $x_i^j \in \{0, 1\}$ to model the data node association. For data node $u_i \in \mathcal{U}$, if $x_i^j = 1$, a worker will be deployed on edge server v_j to collect its data and conduct distributed model training. Additionally, each data node should associate with only one edge server due to the limited storage resource on edge servers and the feature of data nodes. Even if only partial data are forwarded to an edge server, we still need to place the entire service on this edge server. Redundant placement of the same service will make limited resources more scarce. Each data node, e.g., tiny sensors, generates a specific type of data that usually can be processed together. Then, the amount of collected data on edge server v_j for job request ϕ_k is

$$D_j^k = \sum_{u_i \in C_k} x_i^j \cdot u_i^d \cdot T_i \tag{1}$$

2.4 Resource Allocation

During the training process, the resources on each edge server have to be shared among several jobs. We denote the amount of type-γ resource on edge server v_j allocated for job request ϕ_k as $r_{j,k}^\gamma$. For example, we allocate computation resource of $r_{j,k}^c$ and the outbound bandwidth resource of $r_{j,k}^b$ on edge server v_j for job ϕ_k. We also denote the time for training the model of job ϕ_k on edge server v_j as $t_{j,k}$, which consists of the computing time $t_{j,k}^{cp}$ and the communication time $t_{j,k}^{cm}$.

The time for training the mini-batches adds up to the computing time. The mini-batch size of job request ϕ_k is denoted as f_k, and the number of float operations for processing one mini-batch of job request ϕ_k is denoted as g_k. In practice, machine learning models are usually non-convex, and the required number of training epochs for convergence can not be estimated. However, different from experimental models, production models have been tested many times under different situations and the hyper-parameters have

TABLE 1
Important Notations

$\mathcal{U} = \{u_1, u_2, \cdots, u_n\}$	the set of data nodes
$\mathcal{V} = \{v_1, v_2, \cdots, v_m\}$	the set of edge servers
$\Phi = \{\phi_1, \phi_2, \cdots, \phi_q\}$	the set of job requests
$\mathcal{V}_i \subset \mathcal{V}$	the feasible edge servers of data node u_i
$\mathcal{U}_j \subset \mathcal{U}$	the data nodes that can be associsated to edge server v_j
$C_k \subset \mathcal{U}$	the data nodes that serve job request ϕ_k
r_j^d	input data bandwidth of edge server v_j
r_j^c	computation capacity of edge server v_j
r_j^b	communication capacity of edge server v_j
u_i^d	uploading rate of data node u_i
τ_k	upper bound completion time of ϕ_k
f_k	mini-batch size of job request ϕ_k
g_k	the number of float operations for processing one mini-batch of job request ϕ_k
h_k	parameter size (or the gradient size) of job request ϕ_k

been tuned well during the experimental phase [19]. In the edge computing scenario, model training jobs usually are fine-tuning or incremental learning processions which can achieve the ideal training effect in a few epochs, denoted by χ_k. The number of iterations during one epoch can be derived as $\frac{D_j^k}{f_k}$ for job request ϕ_k on edge server v_j. By our survey and experimental results, the computing time has linear relationship with the number of float-point operations [20] and is inversely proportional to computation resource. We calculate the computing time of job ϕ_k on edge server v_j as

$$t_{j,k}^{cp} = \frac{D_j^k \cdot g_k \cdot \chi_k}{f_k \cdot r_{j,k}^c} \tag{2}$$

Moreover, we discuss another method to estimate the computation resource requirement in Section 3.5.

We use h_k to denote the parameters' size (or the gradients' size) of job request ϕ_k. Suppose we perform global update every E_k iterations for job request ϕ_k. The number of communication times during one epoch of job ϕ_k is $\frac{D_j^k}{E_k \cdot f_k}$ on edge server v_j. We calculate the communication time of job request ϕ_k on edge server v_j as

$$t_{j,k}^{cm} = \frac{D_j^k \cdot \chi_k \cdot h_k}{E_k \cdot f_k \cdot r_{j,k}^b} \tag{3}$$

Thus, the total training time of job request ϕ_k on edge server v_j is

$$t_{j,k} = t_{j,k}^{cp} + t_{j,k}^{cm} \tag{4}$$

As a result, the completion time of job request ϕ_k can be expressed as $t_k = \max_{v_j \in \mathcal{V}} t_{j,k}$.

3 Algorithm Description

In this section, we define the Data collection and Resource allocation Problem (DRP) for distributed machine learning at the edge. Then, we design an efficient algorithm for this problem and analyze the approximation performance of our proposed algorithm.

3.1 Problem Formulation

To maximize the system throughput and ensure the system's QoS, we formulate the DRP problem as follows:

$$\max \sum_{\phi_k \in \Phi} y_k \tag{5a}$$

$$\textbf{s.t.} \quad \sum_{v_j \in \mathcal{V}_i} x_i^j = y_k, \quad \forall \phi_k \in \Phi, u_i \in C_k \tag{5b}$$

$$\sum_{u_i \in \mathcal{U}_j} x_i^j u_i^d \le r_j^d, \quad \forall v_j \in \mathcal{V} \tag{5c}$$

$$\sum_{\phi_k \in \Phi} r_{j,k}^c \le r_j^c, \quad \forall v_j \in \mathcal{V} \tag{5d}$$

$$\sum_{\phi_k \in \Phi} r_{j,k}^b \le r_j^b, \quad \forall v_j \in \mathcal{V} \tag{5e}$$

$$t_{j,k} \le \tau_k, \quad \forall \phi_k \in \Phi, v_j \in \mathcal{V} \tag{5f}$$

$$y_k \in \{0,1\}, \quad \forall \phi_k \in \Phi \tag{5g}$$

$$x_i^j \in \{0,1\}, \quad \forall u_i \in \mathcal{U}, v_j \in \mathcal{V} \tag{5h}$$

$$r_{j,k}^c, r_{j,k}^b \ge 0, \quad \forall v_j \in \mathcal{V}, \phi_k \in \Phi \tag{5i}$$

Eq. (5b) ensures the dependence between the data nodes association and job assignment. A job will be assigned if and only if all of the corresponding data nodes are associated. Moreover, due to the binary decisions of job assignments, the sum terms in Eq. (5b) indicate that each data node will be associated to only one edge server. Eq. (5c) indicates that the total collection rate on each edge server should not exceed its inbound bandwidth. Eqs. (5d) and (5e) ensure that the allocated resources for the admitted job requests should not exceed the resource capacity on each edge server. Eq. (5f) states that the completion time of each job should not exceed its upper bound. Our objective is aim to maximize the system throughput in Eq. (5a).

Theorem 1. The DRP problem is NP-hard.

The DRP problem has been formulated as a mixed integer non-linear program in Eq. (5). Intuitively, if the data nodes and edge servers are regarded as items and knapsacks, the DRP problem is similar to multiple knapsack problem (MKP). However, there are basically two different points between these two problems even if we ignore the resource allocation in DRP. On one hand, each data node can be associated with only one subset of edge servers in DRP, while each item can be put into any knapsack in MKP. On the other hand, for all the data nodes of a job request, their association is not independent. That is, either all of them are associated or none of them is associated. We consider a special case of DRP, in which we ignore the upper bound of completion time for each job request ϕ_k by setting τ_k as infinite. We assume each job request has only one data node, which can be associated to all the edge servers. As a result, the simplified version of DRP becomes MKP, which is NP-hard. Since MKP is a special case of DRP, the DRP problem is NP-hard.

Compared to the optimization problem in cloud computing, network topology and the effect of limited resources make our problem difficult. Specifically, due to the locally connected wireless network and limited computation resource, we need to make a decision of data node association rather than collecting all data to the cloud.

3.2 Relaxation and Reformulation

To circumvent the NP-hardness of the DRP problem, we relax the integer constraints Eqs. (5g) and (5h) as

$$y_k \in [0,1], \forall \phi_k \in \Phi \tag{6}$$

$$x_i^j \in [0,1], \forall u_i \in \mathcal{U}, v_j \in \mathcal{V} \tag{7}$$

In a practical DML system, the training time consists of communication time and computation time. We introduce the system-wide configurable coefficient $\epsilon \in (0,1)$, which indicates the ratio of communication time to the total completion time of a job. To facilitate the non-linear constraints in Eq. (5f), we transform them into two sets of linear sub-constraints with the help of ϵ as follows:

$$t_{j,k}^{cm} \le \epsilon \tau_k, \forall \phi_k \in \Phi, v_j \in \mathcal{V} \tag{8}$$

$$t_{j,k}^{cp} \le (1-\epsilon)\tau_k, \forall \phi_k \in \Phi, v_j \in \mathcal{V} \tag{9}$$

Combining Eqs. (2) and (3), the above two equations can be transformed into

$$\frac{D_j^k \chi_k h_k}{E_k f_k r_{j,k}^b} \le \epsilon \tau_k, \forall \phi_k \in \Phi, v_j \in \mathcal{V} \quad (10)$$

$$\frac{D_j^k g_k \chi_k}{f_k r_{j,k}^c} \le (1-\epsilon)\tau_k, \forall \phi_k \in \Phi, v_j \in \mathcal{V} \quad (11)$$

They are equivalent to

$$D_j^k \chi_k h_k - \epsilon \tau_k E_k f_k r_{j,k}^b \le 0, \forall \phi_k \in \Phi, v_j \in \mathcal{V} \quad (12)$$

$$D_j^k g_k \chi_k - (1-\epsilon)\tau_k f_k r_{j,k}^c \le 0, \forall \phi_k \in \Phi, v_j \in \mathcal{V} \quad (13)$$

After relaxation and reformulation, we get the linear program with respect to $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{r}^c$, $\boldsymbol{r}^b$.

3.3 Approximation Algorithm

Given the problem instance I, we develop the rounding-based algorithm (R-DR) in an iterative scheme for the DRP problem. The proposed R-DR algorithm is formally described in Alg. 1.

At the beginning of each iteration, R-DR performs the pruning operation on the instance I (Line 2) to immediately reject those trivial job requests. We regard that job request ϕ_k is non-trivial if all the data nodes in C_k meet the conditions below. For each data node $u_i \in C_k$, at least one edge server $v_j \in \mathcal{V}_i$ satisfies the bandwidth requirement, *i.e.*, $r_j^d \ge u_i^d$. We remove a trivial job request and its data nodes from I. The pruning operations are performed as follows (Lines 22-31). For data node $u_i \in \mathcal{U}$, we remove edge server v_j from $\mathcal{V}_i$ and remove u_i from $\mathcal{U}_j$ if the edge server v_j doesn't meet its bandwidth requirement u_i^d, *i.e.*, $r_j^d < u_i^d$.

We define the degree of data node u_i as the number of feasible edge servers $|\mathcal{U}_i|$, denoted as $B(u_i)$. For job request ϕ_k, let $\mathcal{U}_j^k$ denote the subset of data nodes that are associated with edge server v_j. For the data nodes in $\mathcal{U}_j^k, k \in \{1, 2, \cdots, |\Phi|\}$, we order them in the non-decreasing order of their degree. Let u'^k_j be the data node with the maximum degree in $\mathcal{U}_j^k$. It follows

$$\sum_{u_i \in \mathcal{U}_j^k, B(u_i) \le B(u'^k_j)} u_i^d \le r_j^d \quad (14)$$

$$\chi_k h_k \sum_{u_i \in \mathcal{U}_j^k, B(u_i) \le B(u'^k_j)} x_i^j u_i^d T_i - \epsilon \tau_k E_k f_k r_{j,k}^b \le 0 \quad (15)$$

$$g_k \chi_k \sum_{u_i \in \mathcal{U}_j^k, B(u_i) \le B(u'^k_j)} x_i^j u_i^d T_i - (1-\epsilon)\tau_k f_k r_{j,k}^c \le 0 \quad (16)$$

We remove the data nodes in $\{u_i | u_i \in \mathcal{U}_j^k, B(u_i) > B(u'^k_j)\}$ from $\mathcal{U}_j$ and remove v_j from their feasible edge server sets.

Next, we make a preliminary association for each data node of non-trivial job requests (Lines 7-14). We construct a linear program $LP1_I$ and obtain an optimal solution $\{\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{r}}^c, \hat{\boldsymbol{r}}^b\}$. The optimal (or fractional) solution may be infeasible for the integer constraints of the original problem. We implement the association of data nodes with edge servers by randomized rounding technique on the solution of $LP1_I$. For the data nodes in $C_k, \phi_k \in \Phi$, we regard $\hat{y}_k$ as their priority in the current iteration. In addition, we interpret the fractional values $\hat{x}_i^j$ as probabilities that data node u_i will be associated with edge server v_j. For data node $u_i \in \mathcal{U}$, we introduce the independent discrete random variable $e_i \in \{1, 2, \cdots, m\}$ and probability distribution given by $\hat{x}_i^j$, *i.e.*, $\mathbf{Pr}\{e_i = j\} = \hat{x}_i^j, j = 1, 2, \cdots, m$; and $\mathbf{Pr}\{e_i = 0\} = 1 - \sum_{v_j \in \mathcal{V}} \hat{x}_i^j$. Therefore, we obtain the preliminary data node association based on $\{e_i\}$.

Algorithm 1 R-DR: Rounding-based algorithm for DRP

Input: Initialize a DRP instance I

```
1:  while $true$ do
2:      PRUNE($I$);
3:      if $\Phi = \emptyset$ then
4:          The algorithm ends;
5:      else
6:          Construct $LP1_I$ as Relaxed DRP
7:          Solve $LP1_I$ and get $\{\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{r}}^c, \hat{\boldsymbol{r}}^b\}$;
8:          Set the priority of $u_i \in C_k$ as $\hat{y}_k$;
9:          Obtain $\tilde{\boldsymbol{x}}$ based on $\hat{\boldsymbol{x}}$ by randomized rounding;
10:         for $v_j \in \mathcal{V}$ do
11:             Set $\mathcal{U}'_j \leftarrow \{u_i | u_i \in \mathcal{U}_j, \tilde{x}_i^j = 1\}$;
12:             Order $\mathcal{U}'_j$ in non-increasing priority order;
13:             Find $u'_j \in \mathcal{U}'_j$ with minimum degree such that
                Eq. (17) is satisfied;
14:             Set $x_i^j \leftarrow 1, u_i \in \mathcal{U}'_j, u_i \ge u'_j$;
15:         for $\phi_k \in \Phi$ do
16:             Set $y_k \leftarrow 1$ if $\sum_{v_j \in \mathcal{V}} x_i^j = 1, \forall u_i \in C_k$;
17:             if $y_k = 1$ then
18:                 Update $r_{j,k}^c, r_{j,k}^b, v_j \in \mathcal{V}$ proportionally to
                    satisfy Eq. (12) and Eq. (13);
19:                 Set $\Phi \leftarrow \Phi - \{\phi_k\}$;
20:                 Set $\mathcal{U} \leftarrow \mathcal{U} - C_k$;
21:         Update $I$;
22: function PRUNE($I$)
23:     Set $\mathcal{V}_i \leftarrow \mathcal{V}_i - \{v_j | v_j \in \mathcal{V}_i, r_j^d < u_i^d\}$;
24:     Set $\mathcal{U}_j \leftarrow \mathcal{U}_j - \{u_i | u_i \in \mathcal{U}_j, r_j^d < u_i^d\}$;
25:     for Each job request $\phi_k \in \Phi$ do
26:         Set $\mathcal{U}_j^k \leftarrow \{u_i | u_i \in \mathcal{U}_j, u_i \in C_k\}$;
27:         Order $\mathcal{U}_j^k$ in non-decreasing degree order;
28:         Find $u'^k_j \in \mathcal{U}_j^k$ with maximum degree such that Eqs.
            (14), (15) and (16) are satisfied;
29:         Set $\hat{U}_j^k \leftarrow \{u_i | u_i \in \mathcal{U}_j^k, u_i > u'^k_j\}$;
30:         Set $\mathcal{U}_j \leftarrow \mathcal{U}_j - \hat{\mathcal{U}}_j^k$, $\mathcal{V}_i \leftarrow \mathcal{V}_i - \{v_j\}, \forall u_i \in \hat{\mathcal{U}}_j^k$;
31:         Set $\Phi \leftarrow \Phi - \{\phi_k\}, \mathcal{U} \leftarrow \mathcal{U} - C_k$ if $\phi_k$ is trivial;
```

Output: $\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{r^c}, \boldsymbol{r^b}$

At last, we adjust the association (Lines 15-20), as the bandwidth constraint or/and the completion time constraint may be violated. In order to derive a feasible association, we denote the set of data nodes associated with edge server v_j as B'_j in the preliminary scheme and rank them in the non-increasing order of their priorities. Let u'_j be the data node with the minimum degree in $\mathcal{U}'_j$ such that

$$\sum_{u_i \in \mathcal{U}'_j, B(u_i) \ge B(u'_j)} u_i^d \le r_j^d \quad (17)$$

Then, we only associate the data nodes in $\{u_i | u_i \in \mathcal{U}'_j, B(u_i) \ge B(u'_j)\}$ with edge server v_j. For a job request, if all of its data nodes are associated, it is admitted. Furthermore, to satisfy the completion time constraint, we update resource allocation for those admitted jobs. Specifically, for each job, we allocate the minimum amount of resources which can meet the completion time constraints. Finally, we remove the admitted job requests with the corresponding data nodes from I and update the available resources on each edge server.

Proposition 1. R-DR terminates in q iterations at most .

Proof. At the beginning of each iteration, the pruning operation on I ensures that edge server $v_j \in \mathcal{V}$ can cope with the extreme situation in which all the candidate data nodes from job request $\phi_k \in \Phi$ are associated. Based on the solution of $LP1_I$, the data nodes of a job request with the highest priority will always be associated in each preliminary scheme. As a result, at least one job request will be admitted in each iteration. Since there are totally q job requests, the R-DR algorithm will terminate in at most q iterations.

3.4 Theoretical Analysis

The iterative feature of Alg. 1 makes it difficult to derive the approximation performance of system throughput directly. We note that the job admission decision is made based on the data node association, which enlightens us on performance analysis through data node association. We first give the Chernoff bound for probability analysis.

Theorem 2. (Chernoff Bound): Given n independent variables: $x_1, x_2, \ldots, x_n$, where $\forall x_i \in [0,1]$. Let $\mu = \mathbb{E}[\sum_{i=1}^{n} x_i]$. Then,

1) *Upper Tail:* $\mathbf{Pr}[\sum_{i=1}^{n} x_i \geq (1+\epsilon)\mu] \leq e^{\frac{-\epsilon^2\mu}{2+\epsilon}}, \forall \epsilon > 0$
2) *Lower Tail:* $\mathbf{Pr}[\sum_{i=1}^{n} x_i \leq (1-\epsilon)\mu] \leq e^{\frac{-\epsilon^2\mu}{2}}, \forall 0 < \epsilon < 1$

We denote the total data upload rate to edge server $v_j \in \mathcal{V}$ by the preliminary scheme as the random variable

$$D_j = \sum_{u_i \in \mathcal{U}_j} u_i^d \cdot \varphi\{e_i = j\} \tag{18}$$

Here $\varphi\{e_i = j\}$ is 1 if $e_i = j$ and 0 otherwise. The probability that data node u_i is associated with v_j is denoted as $\hat{x}_i^j$. We define $\mu_j = E(D_j)$ and $\sigma_j^2 = Var(D_j)$. According to the Chernoff Bound Lower Tail, we obtain the probabilistic tail estimation:

$$\mathbf{Pr}\{D_j < (1-\delta)\mu_j\} < e^{-\frac{\mu_j \delta^2}{2}} \tag{19}$$

where δ is an arbitrary value with $0 < \delta < 1$. Eq. (19) gives an upper bound on the probability that the data amount on edge server v_j does not exceed a certain fraction of its expectation under the preliminary scheme.

We analyze the lower bound of system throughput. Assume that the number of data nodes $n = |\mathcal{U}|$ is large and the data upload rate u_i^d of u_i is uniformly bounded. We suppose that there exists a constant ε such that $x_i^j \in [\varepsilon, 1-\varepsilon]$ is satisfied for data nodes, which ensures the correctness of the central limit theorem [21]. For example, if $\varepsilon = 0$, $x_i^j \in [\varepsilon, 1-\varepsilon]$ can be satisfied for all data nodes obviously. We conclude that $(D_j - \mu_j)/\sigma_j^2 \sim N(0,1)$.

Theorem 3. Our proposed algorithm can achieve the approximation performance of

$$\left(\frac{1}{2} + \frac{\eta}{2n} - \frac{1}{\sqrt{2\pi\eta}}\right) \tag{20}$$

The proof of Theorem 3 is showed in Appendix A

According to some previous work, there is no fully polynomial time approximation scheme (FPTAS) even for MKP with two knapsacks, denoted as 2-KP, unless $\mathbf{P} = \mathbf{NP}$ [22]. The authors [23] have proposed a polynomial-time algorithm with approximation ratio of $\frac{3}{4}$ as the latest heuristics algorithm. Our R-DR algorithm can achieve the approximation factor of $\left(\frac{1}{2} + \frac{\eta}{2n} - \frac{1}{\sqrt{2\pi\eta}}\right)$, where n denotes the number of data nodes and η denotes the maximum number of data nodes that any edge server can serve.

3.5 Discussion

In this section, we give some discussion to enhance the practicability and generality of our proposed algorithm.

In our DRP problem, each data node can serve only one job request. In fact, if data nodes are enabled to serve multiple jobs at the same time, our algorithm can still work with little modification. Specifically, we use Φ_i to denote the job set which data node u_i can serve. Before Line 20 of the R-DR algorithm, we need to remove the admitted job ϕ_k from Φ_i and judge whether Φ_i is empty or not. The modified part of algorithm is shown as follows.

```
1: for $u_i \in C_k$ do
2:     Set $\Phi_i \leftarrow \Phi_i - \{\phi_k\}$;
3:     if $\Phi_i = \emptyset$ then
4:         Set $\mathcal{U} \leftarrow \mathcal{U} - \{u_i\}$;
```

That is, until a data node is no longer required by any job request, we remove it from the unassociated data nodes set.

In the problem definition, we estimate the computation resource requirement based on the given number of epochs for model training. To be more general, we introduce a relaxed solution for resource consumption estimation. The recent work [24] gives a quantitative relationship among computation budget, desired loss, model size and dataset size. This relationship is only suitable for the natural language processing model, but not fit for the general training model. We introduce a new solution based on [24]. Given the desired loss L_k for job request ϕ_k, the computation resource requirement can be simplified as $\frac{C_k^{min}}{\log_{\alpha_c^k} L}$, where C_k^{min} and α_c^k are two experience hyper-parameters related to the dataset size and model size of ϕ_k. In our problem, the computation resource requirement is expressed by $\frac{D_k g_k \chi_k}{f_k}$, which can be replaced by $\frac{C_k^{min}}{\log_{\alpha_c^k} L_k}$. We note that D_k, g_k, χ_k and f_k respectively denote the amount of collected data, the number of float operations for processing one mini-batch, the number of training epochs, and the mini-batch size of job request ϕ_k.

The expression of computing time of job request ϕ_k on edge server v_j in Eq. (2) can be modified accordingly as

$$t_{j,k}^{cp} = \frac{C_k^{min}}{\log_{\alpha_c^k} \cdot L_k} \cdot \frac{1}{r_{j,k}^c} \tag{21}$$

We notice that such replacement will not affect our algorithm design and performance analysis.

4 The Case with Service Placement

In many practical applications, job's execution requires some services (*e.g.*, virtual environment, training models) on edge servers. The lack of required services will cause that some jobs can't be properly executed on edge servers. To fill this gap, we jointly consider service placement and DRP for system throughput maximization.

4.1 Problem Formulation

We consider the data collection, service placement and resource allocation (DSRP) for distributed machine learning. Let $\mathcal{S} = \{s_1, s_2, \cdots, s_p\}$, with $p = |\mathcal{S}|$, denote the set of required services in the system. Specifically, each job ϕ_k requires a service $s_{\phi_k} \in S$, and each service s_l occupies a memory space s_l^o. Due to the limited memory space, an edge server can not cache all services at the same time, and has to judiciously decide which services will be placed. Particularly, a binary variable $z_j^l \in \{0,1\}$ denotes whether service s_l is placed on edge server v_j or not. r_j^o denotes thw memory size for service placement on edge server v_j. We give the problem formulation as follows:

$$\max \sum_{\phi_k \in \Phi} y_k \tag{22a}$$

$$\text{s.t.} \quad \sum_{v_j \in \mathcal{V}_i} x_i^j = y_k, \qquad \forall \phi_k \in \Phi, u_i \in C_k \tag{22b}$$

$$z_j^l \geq x_i^j, \qquad \forall s_l = s_{\phi_k}, u_i \in C_k \tag{22c}$$

$$\sum_{s_l \in \mathcal{S}} z_j^l s_l^o \leq r_j^o, \qquad \forall v_j \in \mathcal{V} \tag{22d}$$

$$\sum_{u_i \in \mathcal{U}_j} x_i^j u_i^d \leq r_j^d, \qquad \forall v_j \in \mathcal{V} \tag{22e}$$

$$\sum_{u_i \in \mathcal{U}_j} r_{i,j}^c \leq r_j^c, \qquad \forall v_j \in \mathcal{V} \tag{22f}$$

$$\sum_{u_i \in \mathcal{U}_j} r_{i,j}^b \leq r_j^b, \qquad \forall v_j \in \mathcal{V} \tag{22g}$$

$$t_{j,k} \leq \tau_k, \qquad \forall \phi_k \in \Phi, v_j \in \mathcal{V} \tag{22h}$$

$$y_k \in \{0,1\}, \qquad \forall \phi_k \in \Phi \tag{22i}$$

$$x_i^j \in \{0,1\}, \qquad \forall u_i \in \mathcal{U}, v_j \in \mathcal{V} \tag{22j}$$

$$z_j^l \in \{0,1\}, \qquad \forall s_l \in \mathcal{S}, v_j \in \mathcal{V} \tag{22k}$$

$$r_{i,j}^c, r_{i,j}^b \geq 0, \qquad \forall v_j \in \mathcal{V}, \phi_k \in \Phi \tag{22l}$$

We note that some constraints in Eq. (22) are same as those in Eq. (5). The main differences are as follows. Eq. (22c) denotes that edge server v_j should cache service s_{ϕ_k} if a data node of job request ϕ_k is associated with edge server v_j. Eq. (22d) ensures that the required storage for service caching should not exceed the memory size of each edge server.

Theorem 4. The DSRP problem is NP-hard.

Proof. We consider a simplified version of DSRP, in which there is no constraint on the memory size of each edge server. Then, the simplified DSRP problem becomes the DRP problem, which is NP-hard by Theorem 1. As a result, DSRP is NP-hard too.

4.2 Algorithm Description

Due to the NP-hardness of DSRP, we propose a rounding-based algorithm R-DSR for this problem. Our algorithm first constructs a linear program as a relaxation of the DSRP problem. By the optimal solution of the relaxed DSRP problem, our algorithm can derive the integral solution for job admission. We should note that since there are three dependent variables in problem formulation, the standard rounding method can not directly solve our problem. To this end, we adopt the Filtering-and-Rounding method [25] to jointly decide the data node association and service placement. The R-DSR algorithm is described in Alg. 2.

Algorithm 2 R-DSR: Rounding-based algorithm for DSRP

```
1: Step 1: Solving the Relaxed DSRP Problem
2: Construct $LP2_I$ as Relaxed DSRP;
3: Obtain the optimal solution $\{\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{r}}^c, \hat{\boldsymbol{r}}^b, \hat{\boldsymbol{z}}\}$;
4: Step 2: Determining Job Request Admission Using Randomized Rounding
5: Derive an integer solution $\tilde{\boldsymbol{y}}$ by randomized rounding ;
6: Step 3: Determining Data Node Association and Service Placement Using Filtering and Rounding
7: for each job $\phi_k$ in $\{\phi_k | \phi_k \in \Phi, \tilde{y}_k = 1\}$ do
8:     Construct a new solution $\{\boldsymbol{x}', \boldsymbol{z}'\}$ by Eq. (26);
9:     Step 3.1: Filtering the fractional solution
10:    Construct filtered feasible edge server set $F_i$ for data node $u_i \in C_k$;
11:    Construct $\{\boldsymbol{x}'', \boldsymbol{z}''\}$ by Eq. (29);
12:    Step 3.2: Rounding the filtered solution
13:    while $C_k \neq \emptyset$ do
14:        Find edge node $u_i \in C_k$ with the minimum association cost $\Theta_i$ ;
15:        Find the edge server $v_{j(i)} \in F_i$ with the smallest placement cost $s_{j(i)}^o$ ;
16:        Set $\tilde{z}_{j(i)}^l \leftarrow 1$, $\tilde{x}_i^{j(i)} \leftarrow 1$;
17:        Set $C_k \leftarrow C_k - u_i$;
18:        for $u_{i'} \in C_k$ do
19:            if $F_i \cap F_{i'} \neq \emptyset$ then
20:                Set $\tilde{x}_{i'}^{j(i)} \leftarrow 1$;
21:                Set $C_k \leftarrow C_k - u_{i'}$;
```

Our R-DSR algorithm first constructs a linear program as a relaxation of the DSRP problem. By relaxing these assumptions in Eqs. (22i)-(22k), data collection of each data node is permitted to be splittable. We relax the integer constraints of the DSRP problem as follows:

$$y_k \in [0,1], \forall \phi_k \in \Phi \tag{23}$$

$$x_i^j \in [0,1], \forall u_i \in \mathcal{U}, v_j \in \mathcal{V} \tag{24}$$

$$z_j^l \in [0,1], \forall s_l \in \mathcal{S}, v_j \in \mathcal{V} \tag{25}$$

After relaxation and reformulation, we construct the linear program $LP2_I$. We can derive the optimal solution of $LP2_I$ as $\{\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{r}}^c, \hat{\boldsymbol{r}}^b, \hat{\boldsymbol{z}}\}$ and the optimal result is $\sum_{\phi_k \in \Phi} \hat{y}_k$. As $LP2_I$ is a relaxation of DSRP, $\sum_{\phi_k \in \Phi} \hat{y}_k$ is the upper-bound result for DSRP.

In the second step, we derive the integral solution $\tilde{\boldsymbol{y}}$, from $\hat{\boldsymbol{y}}$, using the randomized rounding method. For example, the fractional solution of $\hat{\boldsymbol{y}}_{\boldsymbol{k}}$ is 0.3. We randomly choose a value from 0 to 1. Assume that the random value is 0.6, which is larger than 0.3, the job will be admitted. If the value is 0.2, the job will be rejected.

Then, we use the Filtering-and-Rounding method [25] to decide both the data node association and service placement (Lines 6-21). Using this method, we derive the integral solution, denoted as $\{\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{z}}\}$.

For each job $\phi_k \in \Phi$, the cost of service placement is denoted as $P(\boldsymbol{z}) = \sum_{v_j \in \mathcal{V}} \beta_j^o s_l^o z_j^l$ and the cost of data node association is $Q(\boldsymbol{x}, \boldsymbol{z}) = \sum_{u_i \in C_k} \sum_{v_j \in \mathcal{V}_i} (\beta_j^d u_i^d + \beta_j^c r_{i,j}^c + \beta_j^b r_{i,j}^b) x_i^j$, where the factors β_j^o, β_j^d, β_j^c, and β_j^b denote the cost on edge server v_j of unit memory resource, unit inbound bandwidth resource, unit computation resource and unit outbound bandwidth resource, respectively. For example, we determine β_j^d as the ratio of the optimal inbound bandwidth resource cost to the edge server's inbound bandwidth, that is, $\beta_j^d = \frac{\sum_{\phi_k \in \Phi} \sum_{u_i \in C_k} u_i^d \hat{x}_i^j}{r_j^d}$.

Additionally, we interpret the fractional values $\hat{x}_i^j$ as the probability that data node u_i will be associated with edge server v_j. Similarly, $\hat{z}_j^l$ denotes the probability that service s_l will be placed on the edge server v_j. After we have determined the admission of each job request, we re-calculate the probabilities x'^j_i and z'^l_j as follows

$$x'^j_i = \frac{\hat{x}_i^j}{\hat{y}_k}, z'^l_j = \frac{\hat{z}_j^l}{\hat{y}_k} \tag{26}$$

The new fractional solution is denoted as $\{\boldsymbol{x}', \boldsymbol{z}'\}$.

Filtering Step: In this step, we construct a feasible solution by filtering [26]. Based on the fractional solution $\{\boldsymbol{x}', \boldsymbol{z}'\}$, we define the fractional association cost for data node u_i as:

$$\Theta_i = \sum_{v_j \in \mathcal{V}_i} (\beta_j^d u_i^d + \beta_j^c r_{i,j}^c + \beta_j^b r_{i,j}^b) x'^j_i \tag{27}$$

To filter out a set of candidate edge servers for data node u_i, we fix a parameter $\lambda > 0$. Based on the association cost of different edge servers, we construct a filtered feasible edge server set $F_i \subseteq \mathcal{V}_i$ for data node u_i as follows:

$$F_i = \{v_j | x'^j_i > 0, \beta_j^d u_i^d + \beta_j^c r_{i,j}^c + \beta_j^b r_{i,j}^b \leq (1+\lambda)\Theta_i\} \tag{28}$$

We derive another feasible fractional solution $\{\boldsymbol{x}'', \boldsymbol{z}''\}$ as

$$\begin{cases} x''^j_i &= \begin{cases} 0, & v_j \notin F_i \\ \frac{x'^j_i}{\sum_{v_j \in F_i} x'^j_i}, & v_j \in F_i \end{cases} \\ z''^l_j &= \min\{1, \frac{1+\lambda}{\lambda} z'^l_j\} \end{cases} \tag{29}$$

Rounding Step: This step will round $\{\boldsymbol{x}'', \boldsymbol{z}''\}$ to the feasible integer solutions. For each admitted job ϕ_k, *i.e.*, $\phi_k \in \{\phi_k | \phi_k \in \Phi, \tilde{y}_k = 1\}$, we find an disassociated data node $u_i \in C_k$ with the smallest association cost. To place services for this data node u_i, we find the edge server $v_{j(i)} \in F_i$ with the smallest placement cost. We then associate data node u_i with edge server $v_{j(i)}$ and place service s_{ϕ_k} on $v_{j(i)}$. If there exists a data node $u_{i'}$ which satisfies $F_i \cap F_{i'} \neq \emptyset$, we associate $u_{i'}$ with edge server $v_{j(i)}$. After that, we repeat the rounding process until all data nodes in C_k are checked.

4.3 Approximation Performance Analysis

We analyze the approximation performance of the proposed R-DSR algorithm. In Alg. 1, the R-DR algorithm adopts the randomized rounding method to obtain the solution for data node association $\boldsymbol{x}$ and the request admission decision $\boldsymbol{y}$ is derived from $\boldsymbol{x}$. In Alg. 2, we use the randomized rounding method to obtain the solution $\boldsymbol{y}$ for request admission, and then determine the domain of $\boldsymbol{x}$. Finally, we get a feasible solution $\{\boldsymbol{x}, \boldsymbol{z}\}$ for data node association and service placement. Thus, due to the difference between the order of we derive the feasible solution for variables, approximation performance analysis of the R-DR algorithm is not suitable for that of the R-DSR algorithm.

For sake of brevity, Eqs. (12) and (13) can be reformulated as

$$D_j^k \frac{\chi_k h_k}{\epsilon \tau_k E_k f_k} \leq r_{j,k}^b$$
$$D_j^k \frac{g_k \chi_k}{(1-\epsilon)\tau_k f_k} \leq r_{j,k}^c$$

Combining Eqs. (1), (22f) and (22g), it follows

$$\sum_{\phi_k \in \Phi} \left(\sum_{u_i \in C_k} x_i^j u_i^d T_i \frac{\chi_k h_k}{\epsilon \tau_k E_k f_k} \right) \leq r_j^b, v_j \in \mathcal{V}$$
$$\sum_{\phi_k \in \Phi} \left(\sum_{u_i \in C_k} x_i^j u_i^d T_i \frac{g_k \chi_k}{(1-\epsilon)\tau_k f_k} \right) \leq r_j^c, v_j \in \mathcal{V}$$

The above equations can be expressed as

$$\sum_{u_i \in \mathcal{U}_j} x_i^j w_i^b \leq r_j^b, v_j \in \mathcal{V}$$
$$\sum_{u_i \in \mathcal{U}_j} x_i^j w_i^c \leq r_j^c, v_j \in \mathcal{V}$$

where w_i^b and w_i^c denote these system settings.

$$w_i^b = \frac{\chi_k h_k u_i^d T_i}{\epsilon \tau_k E_k f_k}, u_i \in C_k \tag{30}$$

$$w_i^c = \frac{g_k \chi_k u_i^d T_i}{(1-\epsilon)\tau_k f_k}, u_i \in C_k \tag{31}$$

Assume that the minimum inbound bandwidth of all the edge servers is denoted by r_{min}^d. We define r_{min}^b as the minimum outbound bandwidth and r_{min}^c as the minimum computation capacity of all the edge servers. We define a constant α as follows:

$$\alpha = \min \left\{ \begin{array}{l} \min\{\frac{r_{min}^d}{u_i^d}, u_i \in B_j\}, \\ \min\{\frac{r_{min}^b}{w_i^b}, u_i \in \mathcal{U}\}, \\ \min\{\frac{r_{min}^c}{w_i^c}, u_i \in \mathcal{U}\} \end{array} \right\} \tag{32}$$

To meet the conditions of the Chernoff Bound, we use α to make the ϕ_k term be within [0,1] while maintaining resource constraints. In general, the resource demand of a job request will not exceed the edge server capacity. For example, the data upload rate (*e.g.*, 300Mbps [27]) does not exceed the edge server's inbound bandwidth (*e.g.*, 10Gbps). Thus, we regard $\alpha \geq 1$ under many practical scenarios.

System Throughput: A fractional solution $\hat{y}_k$ can be derived by solving the relaxed problem $LP2_I$. The result of $LP2_I$, denoted as $\hat{Y}$, is the upper-bound result of the DSRP problem.

We define a random variable φ_k to denote the admission decision of job request ϕ_k as follow:

$$\varphi_k = \begin{cases} 1, & \textit{with probability } \hat{y}_k \\ 0, & \textit{with probability } 1 - \hat{y}_k \end{cases}$$

The expected number of admitted job requests is:

$$\mathbb{E}[\sum_{\phi_k \in \Phi} \varphi_k] = \sum_{\phi_k \in \Phi} \mathbb{E}[\varphi_k] = \sum_{\phi_k \in \Phi} \hat{y}_k \le \hat{Y} \tag{33}$$

Combining Eq. (33) and the definition of φ, it follows

$$\begin{cases} \frac{\varphi_k \cdot \alpha}{\hat{Y}} \in [0, 1] \\ \mathbb{E}[\sum_{\phi_k \in \Phi} \frac{\varphi_k \cdot \alpha}{\hat{Y}}] \le \alpha \end{cases} \tag{34}$$

By applying the Chernoff Bound Upper Tail, assume that ρ is an arbitrary positive value. We have:

$$\mathbf{Pr}[\sum_{\phi_k \in \Phi} \frac{\varphi_k \cdot \alpha}{\hat{Y}} \ge (1+\rho)\alpha] \le e^{\frac{-\rho^2 \alpha}{2+\rho}} \tag{35}$$

We assume that

$$\mathbf{Pr}[\sum_{\phi_k \in \Phi} \frac{\varphi_k \cdot \alpha}{\hat{Y}}] \ge (1+\rho)\alpha] \le e^{\frac{-\rho^2 \alpha}{2+\rho}} \le \frac{\mathcal{H}}{n} \tag{36}$$

where $\mathcal{H}$ is a function of system-related variables (such as the number of edge servers n). We get the result:

$$\rho \ge \frac{\log \frac{n}{\mathcal{H}} + \sqrt{\log^2 \frac{n}{\mathcal{H}} + 8\alpha \log \frac{n}{\mathcal{H}}}}{2\alpha}, n \ge 2 \tag{37}$$

We set $\mathcal{H} = \frac{1}{n^2}$. Apparently $\mathcal{H} \to 0$ as $n \to \infty$. With respect to Eq. (37), we set

$$\begin{aligned} \rho &= \frac{\log \frac{n}{\mathcal{H}} + \log \frac{n}{\mathcal{H}} + 4\alpha}{2\alpha} \\ &= \frac{3 \log n}{\alpha} + 2 \end{aligned} \tag{38}$$

We use Y_{OPT} and Y_{R-DSR} to denote the optimal system throughput and the result of the R-DSR algorithm. According to the problem relaxation, we know that $Y_{OPT} \le \hat{Y}$. By Eq. (38), there exists

$$Y_{OPT} \le \hat{Y} \le Y_{R-DSR} \cdot O(\frac{\log n}{\alpha} + 1) \tag{39}$$

Thus, the R-DSR algorithm can achieve a system throughput at least $O\left(1/(\frac{\log n}{\alpha} + 1)\right)$ of the optimum.

In the following, we will analyze the approximation performance of resource cost.

Theorem 5. Let $\Psi(\boldsymbol{x}, \boldsymbol{z}) = P(\boldsymbol{z}) + Q(\boldsymbol{x}, \boldsymbol{z})$ denote the total resource cost, including the cost of both data node association and service placement. The fractional solution constructed by Eq. (29) is a feasible solution and satisfies $\Psi(\boldsymbol{x}'', \boldsymbol{z}'') \le \frac{1+\lambda}{\lambda} \Psi(\boldsymbol{x}', \boldsymbol{z}')$.

The proof of Theorem 5 is showed in Appendix B.

Then, we consider the service placement for job $\phi_k \in \Phi$.

Proposition 2. Given the candidate set F_i of data node $u_i \in C_k$, at most one edge server $v_j \in F_i$ will be chosen to place s_{ϕ_k} according to the R-DSR algorithm.

The proof is given in Appendix C.

Based on Proposition 2, we analyze the approximation ratio of the filtering-and-rounding method in Appendix D. The result is shown as follows.

Theorem 6. The filtering-and-rounding method can achieve the approximation ratio of $\max\{3(1+\lambda), (1+\frac{1}{\lambda})\}$. We obtain an approximation ratio of 4 by setting $\lambda = \frac{1}{3}$.

According to the above analysis, the R-DSR algorithm can guarantee that the storage resource cost, the computation resource cost, the communication resource cost, and the bandwidth cost will hardly be violated by a factor of 4. Moreover, R-DSR can achieve the system throughput at least $O\left(1/(\frac{\log n}{\alpha} + 1)\right)$ of the optimum.

4.4 Discussion

In DSRP, each job request associates a set of data nodes and a kind of service. When we consider the scheme of only one job request, the data collection and service placement problem becomes the incapacitated facility location (UFL) problem [28]. However, we need to decide the job admission for multiple job requests and consider the limited resource in DSRP. Sviridenko et. al. [29] show that there is no ρ-approximation for the UFL problem if $\rho > 0.684$ unless $\mathbf{P} = \mathbf{NP}$. Our R-DSR algorithm can guarantee that the memory resource cost, the computation resource cost and the bandwidth cost will hardly be violated by a factor of 4. Moreover, R-DSR can achieve the system throughput at least $O\left(1/(\frac{\log n}{\alpha} + 1)\right)$ of the optimum, with $\alpha \ge 1$ under many practical scenarios.

5 Performance Simulation

In this section, we evaluate the performance of our algorithms through extensive simulations, which are conducted on a personal computer with CPU (Intel i7-8550U) and 16GB memory.

5.1 Simulation Settings

The simulations are performed over two typical network topologies, the hexagon cells topology [30] and the grid network topology [31]. In the first topology, we simulate an area which represents the communication with base stations. This area is divided into 19 hexagonal cells, as depicted in Fig. 8(a). Within each hexagonal cell, there exists one edge server (typically in a small cluster), on which we deploy workers for processing the jobs. Moreover, we assume that a data node can be associated with the edge servers located in the local and adjacent edge cells. In the second topology, we simulate an area and randomly deploy our edge servers on 19 of the 20 center points, as depicted in Fig. 8(b). Edge servers are regularly deployed on a grid network. Each edge server covers a circular region and connects to all the data nodes in this region. We use this topology to simulate ratio access network (RAN) [32]. Mobile devices, such as mobile phones, access the network through the base station. Data nodes are distributed randomly. These two network topologies reflect the main features of edge network topologies in our problem formulation, such as local connection, limited resources, and wireless links.

We deploy a set of identical edge servers in both topologies. For each edge server, we set its floating point operation

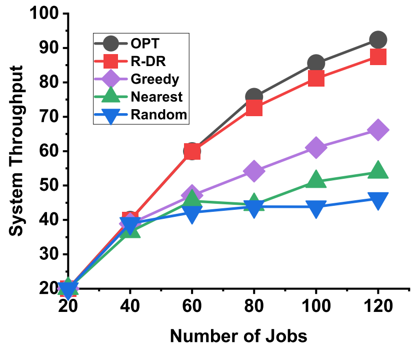


Fig. 2. System throughput under uniform distribution in hexagon cells.

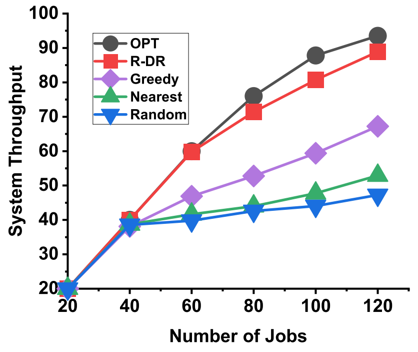


Fig. 3. System throughput under normal distribution in hexagon cells.

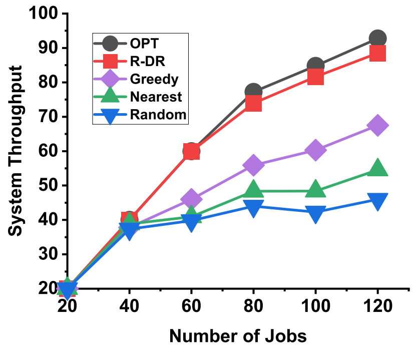


Fig. 4. System throughput under Pareto distribution in hexagon cells.

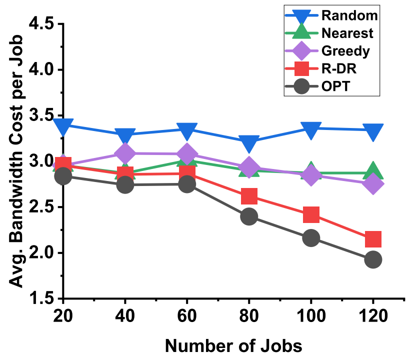


Fig. 5. Average bandwidth cost per job under uniform distribution in hexagon cells.

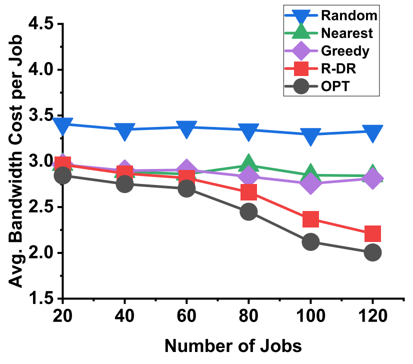


Fig. 6. Average bandwidth cost per job under normal distribution in hexagon cells.

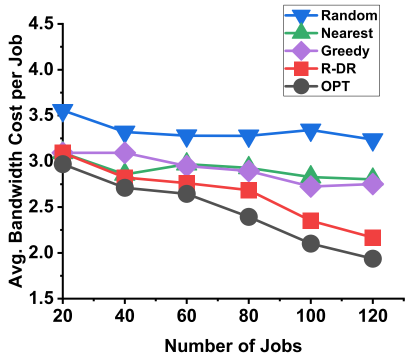


Fig. 7. Average bandwidth cost per job under Pareto distribution in hexagon cells.

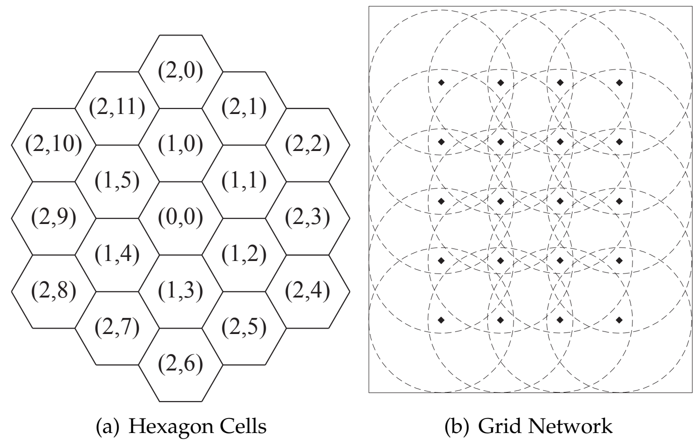


Fig. 8. Illustration of two typical network topologies

capability as 150GFLOPS and set the storage capacity between 100GB to 200GB for caching services. Furthermore, we set the available communication bandwidth from each edge server to parameter servers as 10Gbps [16]. Moreover, we set the communication bandwidth from data nodes to each edge server from 8Gbps to 12Gbps [33].

We assume that each job request needs 15 data nodes to fulfill a training job. We set the number of data nodes required by a job request based on a practical scenario. For example, an indoor position tracking system often contains 10-20 sensors, such as cameras or Wi-Fi sensors [34]. Meanwhile, our experiments are performed over three different distributions of data amount among data nodes, *i.e.*, uniform distribution, normal distribution and Pareto distribution [35]. In the *Uniform distribution*, the minimum value and the maximum value of the distribution are 2GB and 8GB, respectively. In the *Normal distribution*, we set the mean value of the distribution as 5GB, and the standard deviation as 1GB. In the *Pareto distribution*, the minimum value of the distribution is 2GB, and the shape value is 2GB. Additionally, the data nodes provide a data stream for model training and the data generation rate of each data node ranges from 300Mbps to 360Mbps [27]. We set the storage space requirement of services from 20GB to 40GB [36]. Furthermore, we assume that each job requires a different service due to the difference among the trained models of these jobs.

According to the productive models [37], we set the size of parameters (gradients as well) from 30MB to 575MB for different training jobs. Under the data parallelism scheme, the mini-batch size is 6MB, depending on different training data size. We set the number of floating point operations during one iteration based on the respective parameter size and mini-batch size according to the statistics from [38]. For each job, we set 3-8 iterations between global update. Moreover, the job completion time ranges from 1 hour to 2 hours.

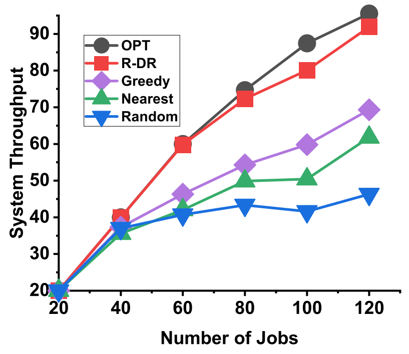


Fig. 9. System throughput under uniform distribution in grid network.

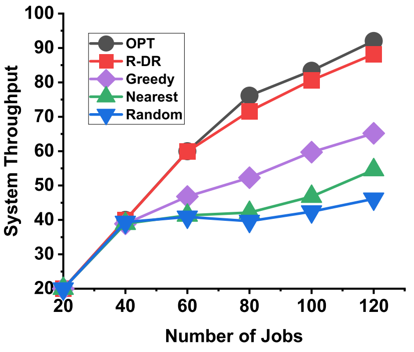


Fig. 10. System throughput under normal distribution in grid network.

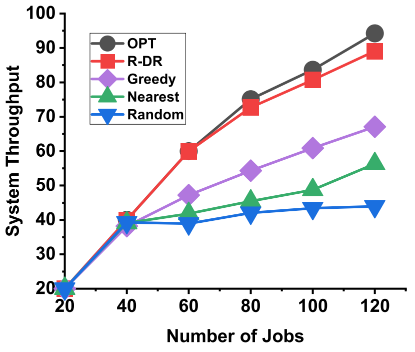


Fig. 11. System throughput under Pareto distribution in grid network.

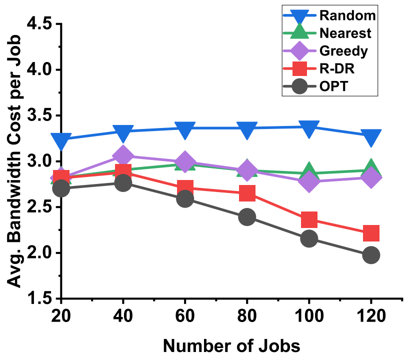


Fig. 12. Average bandwidth cost per job under uniform distribution in grid network.

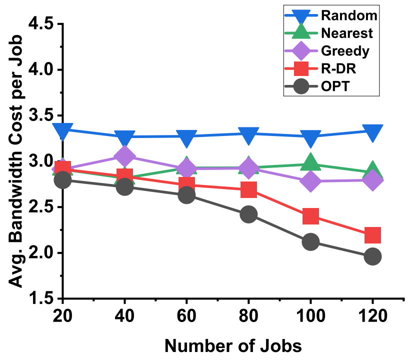


Fig. 13. Average bandwidth cost per job under normal distribution in grid network.

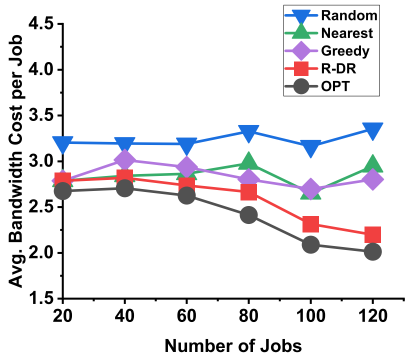


Fig. 14. Average bandwidth cost per job under Pareto distribution in grid network.

5.2 Performance Metrics and Benchmarks

For performance comparison, we adopt the following metrics:

- *System throughput*. The system throughput is the number of admitted jobs, and is computed as $\sum_{\phi_k \in \Phi} y_k$.
- *Average bandwidth cost per job*. To fully enlarge the system throughput under the limited resources on edge servers, the smaller average bandwidth cost means better throughput. Thus, the average bandwidth cost of each job is of great importance. We take the average bandwidth cost per job as another metric, which can be modeled as $\frac{\text{Total bandwidth cost}}{\text{System Throughput}}$.

We compare the proposed R-DR algorithm with three baseline algorithms and choose one baseline algorithm for the proposed R-DSR algorithm in terms of the two metrics. These baseline algorithms initially follow the same philosophy to decide which job request will be admitted. Intuitively, to maximize the system throughput, these algorithms iteratively choose the job request with the minimum data amount and allocate the resources based on the completion time constraints until no job request can be accommodated. The difference lies in how to assign the data nodes of admitted job requests. For the R-DR algorithm, we choose three benchmarks as follows:

Randomized Algorithm: we randomly choose feasible edge servers for data nodes of the admitted job requests.

Greedy Algorithm: We greedily associate data nodes of the admitted job requests with edge severs [39]. Specifically, for a data node, the algorithm orders its feasible edge servers in non-decreasing order of their storage usage and chooses the first edge server that can satisfy the time constraint for the respective job request.

Nearest Algorithm: We associate the data nodes with the nearest feasible edge server [40]. Then, we remove the partly admitted jobs and remove the jobs in descending order of bandwidth requirements until the constraints are satisfied.

The baseline algorithm for the R-DSR algorithm is R-DR with storage constraint, which chooses edge server for the data nodes by the R-DR algorithm regardless the service storage constraint and revokes the association which violates the service storage constraint at end. Since the R-DR algorithm is based on randomized rounding, we regard the solution of $LP1_I$ in the first iteration of Alg. 1 as the optimal solution. Similarly, we regard the solution of $LP2_I$ as the optimal solution of the DSRP problem.

5.3 Numerical Results for R-DR

The first set of simulations compares the system throughput and average bandwidth cost per job for R-DR with three benchmarks under two network topologies. The simulation results are shown in Figs. 2-14.

Figs. 2-4 present the system throughput of different algorithms with different data amount distributions under the hexagon cells topology. When the edge system can supply the redundant resource for job requests, these algorithms

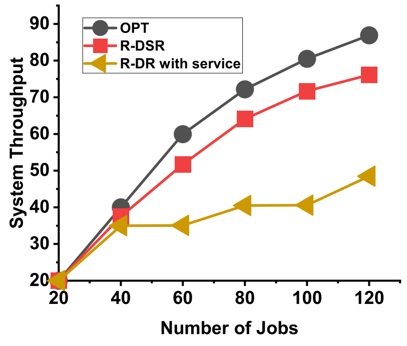


Fig. 15. System throughput under uniform distribution in hexagon cells.

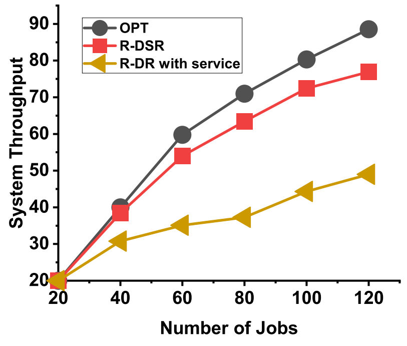


Fig. 16. System throughput under normal distribution in hexagon cells.

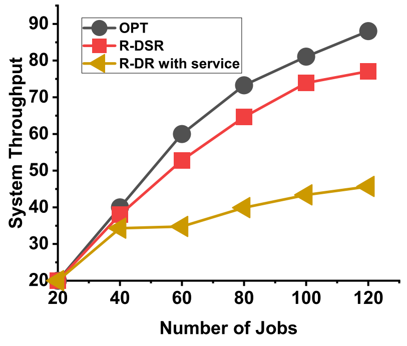


Fig. 17. System throughput under Pareto distribution in hexagon cells.

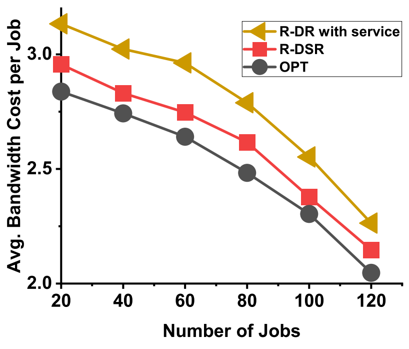


Fig. 18. Average bandwidth cost per job under uniform distribution in hexagon cells.

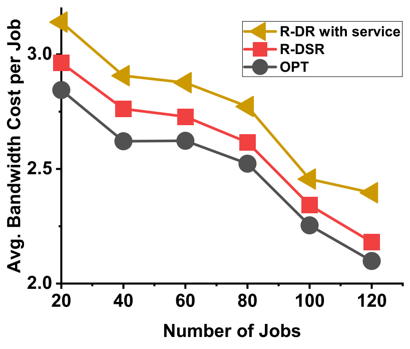


Fig. 19. Average bandwidth cost per job under normal distribution in hexagon cells.

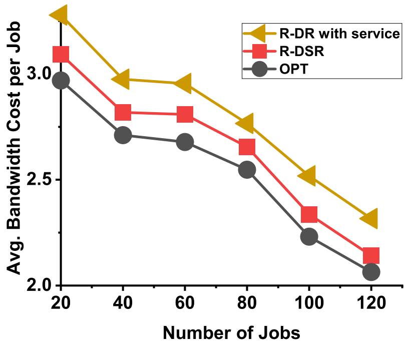


Fig. 20. Average bandwidth cost per job under Pareto distribution in hexagon cells.

exhibit relatively close performance of system throughput. As the number of job requests grows, our R-DR algorithm can accommodate more jobs than the baseline algorithms. In Fig. 2, when the data amount follows the uniform distribution, the proposed algorithm can improve the system throughput by 33%, 51% and 69% compared with the greedy algorithm, the nearest algorithm, and the random algorithm, respectively. In Fig. 3, when the data amount follows the normal distribution, the proposed algorithm can improve the system throughput by 34%, 56% and 69% compared with the other three algorithms. Similar results can be found in Fig. 4. Meanwhile, the proposed algorithm can achieve at least 92% of the optimal performance in terms of system throughput.

Figs. 5-7 illustrate the average bandwidth cost per job of different algorithms. Under the same job requests and system resources, our proposed algorithm can make better admission choices that helps improve system throughput. With the increasing number of job requests, the bandwidth cost per job of our algorithm shows a sharp downward trend. However, the trend of the baseline algorithms is not obvious. In Fig. 5, the proposed algorithm can reduce the bandwidth cost per job by 15%, 16% and 27% compared with the greedy, nearest and random algorithms. Similar results can be founded in Figs. 6 and 7. Moreover, the proposed algorithm only increases the bandwidth cost per job 12% compared with the optimal solution.

Figs. 9-11 present the system throughput of different algorithms with different data amount distributions under the grid network topology. The simulation results show the same trend as those in the hexagon cells topology, which can reflect the excellent performance of our algorithm. In Fig. 9, when the data amount among data nodes follows the uniform distribution, R-DR increases the system throughput by 33%, 50% and 79% compared with the greedy algorithm, the nearest algorithm, and the random algorithm, respectively. Under the normal distribution, the proposed R-DR algorithm can improve the system throughput by 35%, 57% and 77% compared with the other three algorithms by Fig. 10. Similar results are illustrated in Fig. 11. Meanwhile, the proposed algorithm can achieve at least 94% of the optimal performance in terms of system throughput.

Figs. 12-14 illustrate the algorithm performance in terms of the average bandwidth cost per job. Similar to the hexagon cells topology, it reflects the great differences between our algorithm and the baseline algorithms. In Fig. 12, the proposed algorithm can reduce the bandwidth cost per job by 14%, 17% and 27% compared with the greedy algorithm, the nearest algorithm, and the random algorithm, respectively. Similar results can also be derived in Fig. 13 and Fig. 14. Moreover, compared to the optimal solution, the proposed R-DR algorithm only raises the bandwidth cost per job by 11% at most.

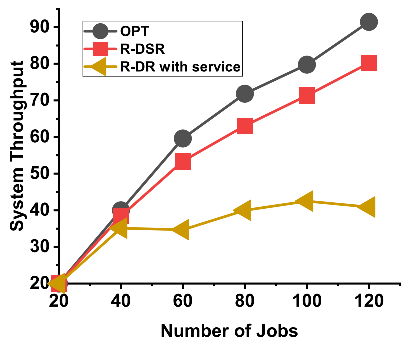


Fig. 21. System throughput under uniform distribution in grid network.

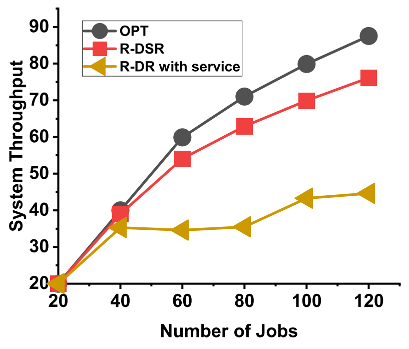


Fig. 22. System throughput under normal distribution in grid network.

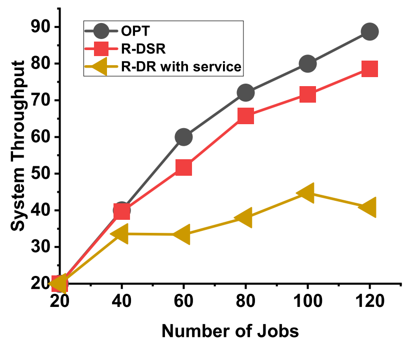


Fig. 23. System throughput under Pareto distribution in grid network.

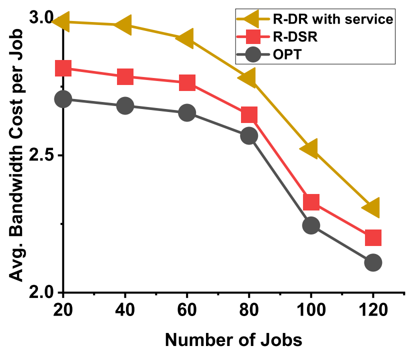


Fig. 24. Average bandwidth cost per job under uniform distribution in grid network.

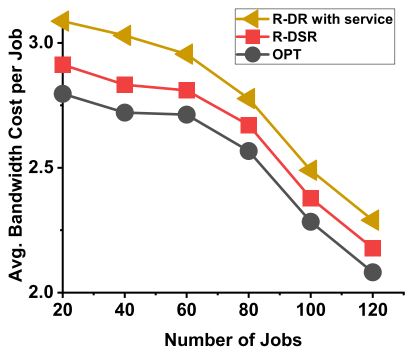


Fig. 25. Average bandwidth cost per job under normal distribution in grid network.

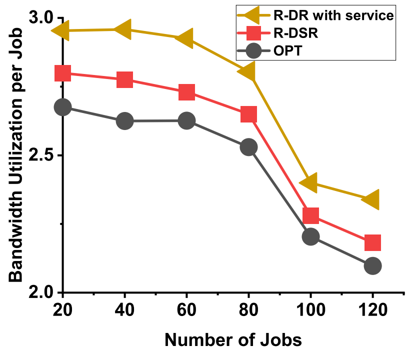


Fig. 26. Average bandwidth cost per job under Pareto distribution in grid network.

5.4 Numerical Results for R-DSR

The second set of simulations compares the system throughput and average bandwidth cost per job for R-DSR with one benchmark under two network topologies.

Figs. 15-17 present the system throughput of different algorithms with different data amount distributions under the hexagon cells topology. We observe that when the system can supply the redundant resource for job requests, these algorithms exhibit relatively close system throughput for all these algorithms. As the number of job requests grows, the gap between the system throughput of our algorithm and that of baseline algorithms is gradually widening. Specifically, in Fig. 15, when the data amount among data nodes follows the uniform distribution, the R-DSR algorithm can improve the system throughput by 63% compared with the R-DR algorithm with service storage constraint. In Fig. 16, when the data amount follows the normal distribution, the system throughput of the proposed algorithm is 69% more than the R-DR algorithm. Similar results are given in Fig. 17. Meanwhile, the proposed algorithm can achieve at least 89% of the optimal performance in terms of system throughput.

Figs. 18-20 illustrate the average bandwidth cost per job. Under the same job requests and system resources, our algorithm can make better choices and make an association that helps improve system throughput. With the increasing number of job requests, the average bandwidth cost per job drops dramatically and maintains a stable gap between our algorithm and the baseline algorithm. In Fig. 18, the R-DSR algorithm can reduce the bandwidth cost per job by 9% compared with the baseline algorithm. Similar results can also be derived in Fig. 19 and Fig. 20. Moreover, compared with the optimal solution, the proposed algorithm only raise 4% bandwidth cost per job.

Figs. 21-23 present the system throughput of different algorithms with different data amount distributions under the grid network topology. It shows that our algorithm still performs well in this network topology. In Fig. 21, when the data amount among data nodes follows the uniform distribution, the system throughput of the proposed algorithm increase by 67% compared with that of the R-DR algorithm with service storage constraint. In Fig. 22, the proposed algorithm can improve the system throughput 69% compared with the baseline algorithm under the normal distribution. Similar results are given in Fig. 23. Meanwhile, the proposed algorithm can achieve at least 88% of the optimal performance in terms of system throughput.

Figs. 24-26 illustrate the algorithm performance in terms of the average bandwidth cost per job. As shown in Fig. 24, the R-DSR algorithm can reduce the bandwidth cost per job by 10% compared with the baseline algorithm. Similar results can also be derived in Figs. 25 and 26. Moreover, the proposed algorithm only increases the bandwidth cost per job 6% compared with the optimal solution.

From these simulation results in Figs. 2-26, we can make some conclusions. First, by Figs. 2-14, our R-DR algorithm can improve the system throughput about 59% compared with the baseline algorithms, and achieve 92% of the optimal system throughput on both two topologies. Moreover, the R-DR algorithm can reduce average bandwidth cost per job by about 17%. Second, by Figs. 15-26, our R-DSR algorithm can improve the system throughput about 67% compared with the baseline algorithms, and achieve 88% of the optimal system throughput on both two topologies. Moreover, the R-DSR algorithm can reduce average bandwidth cost per job by about 11%. Third, our R-DR and R-DSR algorithms can achieve a stable performance on different topologies. Topology changes have little effect on the performance of our algorithms.

6 Test-bed Evaluation

In this section, we set up a test platform based on embedded AI computing device NVIDIA Jetson Tx2 and perform the experiments. Moreover, we evaluate the performance of our proposed algorithm in the test platform.

6.1 Test-bed Setup

The prototype system, illustrated in Fig. 27, consists of 1 server, 4 edge servers and 4 data nodes, which are interconnected via two wireless routers. We model the system as a three-layer structure.

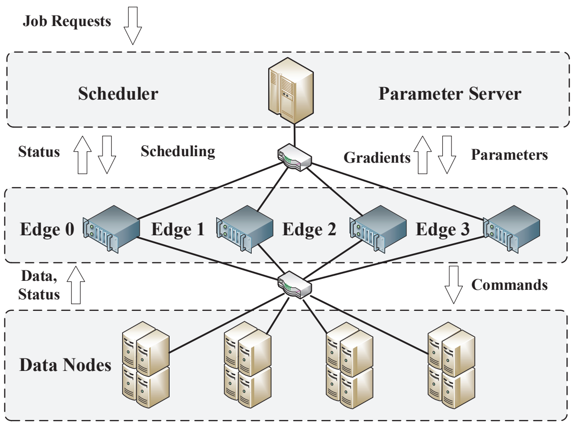


Fig. 27. The network topology of the test-bed

On the bottom layer, all these 4 data nodes store local datasets for model training. With the benefit of data container technique, jobs can be supplied with data individually. The data nodes are responsible to generate and pre-process the data. When a job request is submitted, its data nodes send the status to the edge servers.

On the middle layer, training workers are deployed on edge servers, which provide computing resource and memory resource by a NVIDIA Jetson Tx2. Moreover, the inbound and outbound bandwidths of these edge servers are both 1000Mbps according to the wireless NIC capacity. According to the actual measurement, the wireless network environment of the test-bed is shown in the Table 2. Specifically, the latency refers to the time of transmitting one training batch of data (*e.g.*, 1MB).

TABLE 2
WLAN Environment

Device	Edge 0	Edge 1	Edge 2	Edge 3
RSSI(dBm)	−35	−36	−47	−65
Latency(sec)	0.560	0.564	0.653	0.722

* RSSI stands for Received Signal Strength Indicator

Local model training takes place on the edge servers. Based on the scheduling results from the scheduler, edge servers begin to collect the data for job requests and start the model training. After collecting the data and allocating the currently available resources for job requests, each edge server starts the training processes and synchronizes models between parameter servers. Our demo experiments show that the service placement problem has little effect on the performance (*e.g.*, completion time of job requests) when there are not enough number of edge servers. Thus, we use the R-DR algorithm in this test-bed.

On the top layer, the parameter servers and scheduler are located on the desktop. When we establish the test-bed platform, a scheduler instance and parameter server instances are initially implemented. The scheduler uses the R-DR algorithm to make an association based on the edge server information and job requests. According to the association, one parameter server is deployed for each admitted job, and is responsible for setting up models and updating models for the jobs.

Job Requests: Our job request set is based on four prototype jobs, *i.e.* training of Convolutional Neural Network (CNN) and SVM models on two different datasets. To simulate the data generated by the devices, we choose the MNIST (52MB) and CIFAR-10 (177MB) datasets with the same data amount [41] [42]. Each dataset is divided into 5 parts, with one part on a data node for each training job. In our evaluation, we set the mini-batch size to 64 samples and the number of epochs to 20 for all job requests. In this way, we generate five sets of job requests based on these four prototype jobs, including 14, 16, 18, 20, and 22 jobs, respectively.

Due to certain gap of computing power and the number of edge servers between practical environment and laboratory environment. The settings of simulation and test-bed are different.

Performance Metrics and Benchmarks: For performance comparison, we refine the system throughput metric as two metrics. As the number of job requests increases, we consider two situations. On one hand, edge servers are able to complete all job requests with the abundant system resources. We compare the completion time of all job requests. On the other hand, only partial jobs can be completed due to the resource constraints. We compare the completion ratio during the evaluation. Additionally, we compare the time cost to make a decision. We adopt the following metrics:

- *Computation time of algorithms*. We compare the computation time of algorithms.
- *Completion time of all job requests*. We compare the completion time of all job requests on edge servers.
- *Completion ratio at same timestamp*. We observe the ratio of completed jobs to all jobs at the same timestamp.

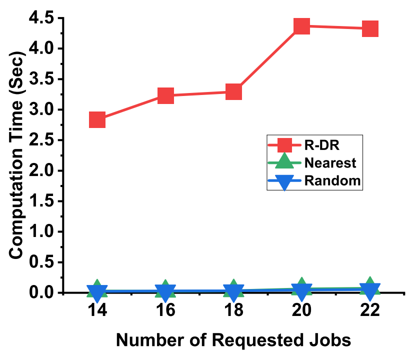


Fig. 28. Computation Time of Algorithms

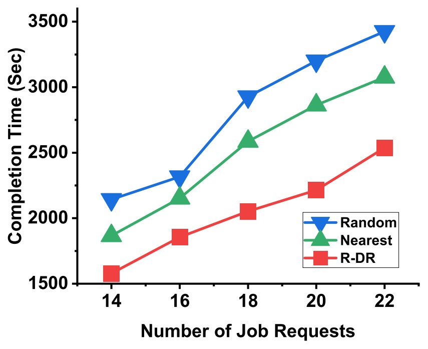


Fig. 29. Completion Time of Job Requests

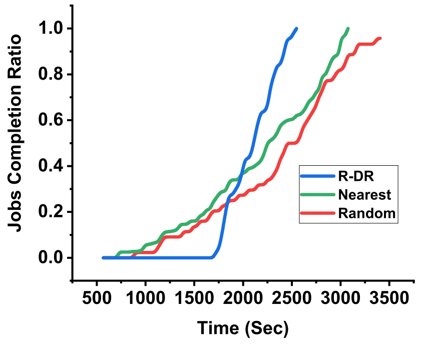


Fig. 30. Jobs Completion Ratio during a Simulation (22 jobs)

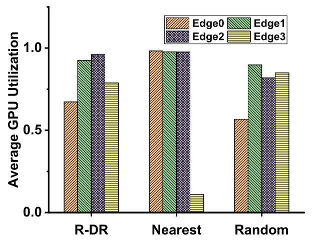


Fig. 31. Average GPU Utilization among Each Edges (16 jobs)

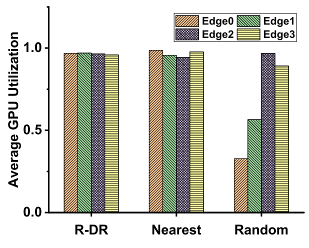


Fig. 32. Average GPU Utilization among Each Edges (22 jobs)

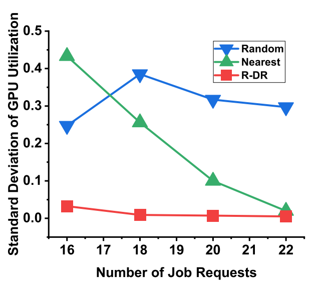


Fig. 33. Standard Deviation of GPU Utilization among Each Edges

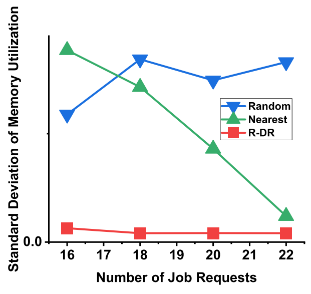


Fig. 34. Standard Deviation of Memory Utilization among Each Edges

The higher ratio represents higher efficiency of our algorithm.

Due to the different system setups of simulations and test-bed, computing resource and memory resource become the performance bottleneck, rather than bandwidth resource. We will analyze the resource performance at the end of this section. In practical scenarios, users' model training requests usually offload to the nearest edge server. To verify the performance of our algorithm in edge computing scenarios, we approximate the practical scenario as a random process and compare it with our algorithm.

6.2 Experimental Results

We run three algorithms on five sets of job requests to demonstrate the efficiency of our proposed algorithm. Specifically, the nearest algorithm make a decision based on the RSSI information.

Computation Time Results: The first evaluation investigates the computation time of algorithms by changing the number of job requests. Compared to the completion time of model training jobs, the computation time of our algorithms only takes a small part. Additionally, our algorithm can help the system achieve better performance in terms of job completion time at a little more decision time cost. We have added experiments to record time for the decision process and training process. The results are shown in Fig. 28. The decision process only takes less than 5 seconds, which can be negligible relative to the about 1-hour total time (Fig. 29). Although more time is spent on making decisions, the total time consumption is reduced compared to the benchmark algorithms.

Completion Time Results: In the second set of evaluations, we investigate the completion time of all jobs by changing the number of job requests. The results are shown in Fig. 29. As the number of job requests increases, all algorithms take more time to complete jobs. However, for each job request, our R-DR algorithm reduces the completion time 26% compared with the random algorithm and 20% compared with the nearest algorithm.

Completion Ratio Results: The third set of evaluation investigates the job completion ratio during the system running. Specifically, we take 22 job requests as an example. On the early stage of training (before about 1800s in Fig. 30), the random association and the nearest association have a better performance than ours. Since the data nodes are randomly associated, some jobs' data nodes are associated with the idle edge servers and these jobs will be completed faster. With system running (after about 1800 s in Fig. 29), our algorithm can complete all jobs faster. We note that, due to multi-type resource constraints, the data nodes associated with the same edge may exceed its capacity and not all jobs can be completed before the deadline by the random algorithm. Thus, our proposed algorithm can complete more jobs in the same time period with resource constraints.

To further explore the reasons for the above experimental results, we record the GPU utilization and the memory utilization during the evaluations. Fig. 33 presents the standard deviation of the GPU utilization among edge nodes. Obviously, the standard deviation of two benchmarks is much larger than that of our algorithm, which implicates better computing load balancing of our algorithm is better. To further reveal the system performance, we record the GPU and memory utilization of edge servers during model training. We observe the GPU utilization as an example, as shown in Figs. 31-32. The GPU utilization of four edge servers by our algorithm is greater than 95%. As for the random method and nearest method, the GPU utilization of some edges is only 30% even 11%. Fig. 34 shows the same feature on memory consumption. This also explains why our algorithm takes less time to complete the same number of jobs and completes more tasks with the limited resources.

7 Related Works

7.1 Data Offloading

Since few works focus on data collection for distributed deep learning in edge computing, we alternatively investigate the problem of job offloading which has a resemblance in assignment relationship. There are various works that study job offloading from different points of view. To reduce the task duration, the authors in [43] designed a hierarchical edge computing architecture, and proposed an optimal offloading scheme. In order to keep the delay minimum and save the battery life of user's equipment, works [44] transformed the job offloading problem into two sub-problems and proposed the respective solutions. In [45], the authors considered the job offloading problem from the perspective of energy saving, a novel user-centric energy-aware mobility management scheme was developed. Several works [46], [47] considered the job offloading from the collaboration view in which each job has only one end device.

Resource allocation is another important research point in edge computing. In [48], through implementing via successive convex approximation, the authors proposed a novel specialized resource allocation approach for such applications as augmented reality. A joint scheduling algorithm that allocates both radio and compute resources coordinately was developed in [49] to avoid wasting resources. These proposed resource allocation schemes mainly focus on generic tasks, however, further performance gain can be obtained if we take the feature of distributed training tasks into account.

7.2 Service Placement

Service placement will also impact the performance of task offloading. Various solutions have been developed to place content popularity [50] [51] or request history [52]. Only a few works have considered multiple types of resources (*e.g.*, storage, computation, communication). In [53], mixed integer linear programs (MILPs) were formulated for placing contents or service functions, and activating storage, computation, and communication resources in a distributed cloud network. Various works have studied the joint optimization of service placement and task offloading in MEC [36] [54]. However, the existing works only consider the situation in which each job only needs to place one or fixed number of services, placement for uncertain number of services is rarely considered.

7.3 Learning at the edge

There have been a bunch of prior works about conducting machine learning tasks at the edge. A concept termed federated learning is proposed in [55], in which mobile phones and IoT devices can be used to learn a shared model in a decentralized approach. In [56], the authors proposed a distributed deep neural network over hierarchies consisting of the cloud, the edge and end devices, thus significantly reducing the communication cost. The authors in [57] introduced deep learning for IoTs into the edge computing environment and designed a novel offloading strategy to optimize the performance. The authors [58] proposed a control algorithm dynamically adapts the frequency of global aggregation in real time to minimize the learning loss under a fixed resource budget. The authors in [59] provided an algorithm to strike the best error-runtime trade-off in decentralized SGD by carefully tuning the frequency of inter-node communication. While these works focus on conducting training tasks at the edge, they ignore the joint optimization of data collection and resource allocation for these training tasks, which is of vital importance for the resource constrained edge computing environment.

8 Conclusion

In this paper, we have studied the joint data collection and resource allocation for distributed machine learning in edge computing system. The joint problem is first formulated as a mixed integer non-linear program. We propose an efficient algorithm with a provable performance guarantee. Furthermore, we jointly consider the service placement, and design an efficient algorithm with constant bipartite approximation in many practical situations. We set up a test-bed to evaluate the effectiveness of our proposed algorithm. Compared with the existing algorithms, our proposed algorithms can improve the system throughput $56\% - 69\%$ by the simulation results.

Acknowledgment

Hongli Xu and He Huang are corresponding authors. This article was supported in part by the National Science Foundation of China (NSFC) under Grants 61822210, 61936015, and U1709217; and in part by Anhui Initiative in Quantum Information Technologies under Grant AHY150300.

References

[1] 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 *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2019, pp. 734–741.

[2] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, "Big data: The next frontier for innovation, competition, and productivity," *Analytics*, 2011.

[3] C. V. Networking, "Cisco global cloud index: Forecast and methodology, 2015-2020. white paper," *Cisco Public, San Jose*, 2016.

[4] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Savannah, GA, 2016, pp. 265–283.

[5] D. Agrawal, S. Das, and A. El Abbadi, "Big data and cloud computing: Current state and future opportunities," in *Proceedings of the 14th International Conference on Extending Database Technology*, ser. EDBT/ICDT '11, 2011, pp. 530–533.

[6] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, “Distributed graphlab: A framework for machine learning and data mining in the cloud,” *Proc. VLDB Endow.*, vol. 5, no. 8, pp. 716–727, Apr. 2012.

[7] P. Mach and Z. Becvar, “Mobile edge computing: A survey on architecture and computation offloading,” *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, thirdquarter 2017.

[8] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, “A survey on mobile edge computing: The communication perspective,” *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, Fourthquarter 2017.

[9] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, “Deepx: A software accelerator for low-power deep learning inference on mobile devices,” in *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*, ser. IPSN ’16, 2016, pp. 23:1–23:12.

[10] X. Chen, W. Li, S. Lu, Z. Zhou, and X. Fu, “Efficient resource allocation for on-demand mobile-edge cloud computing,” *IEEE Transactions on Vehicular Technology*, vol. 67, no. 9, pp. 8769–8780, Sept 2018.

[11] Z. Lu, S. Rallapalli, K. S. Chan, S. Pu, and T. La Porta, “Augur: Modeling the resource requirements of convnets on mobile devices,” *IEEE Transactions on Mobile Computing*, 2019.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[13] H. Li and Z. Lin, “Accelerated proximal gradient methods for nonconvex programming,” in *Advances in neural information processing systems*, 2015, pp. 379–387.

[14] A. M. Elkahky, Y. Song, and X. He, “A multi-view deep learning approach for cross domain user modeling in recommendation systems,” in *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2015, pp. 278–288.

[15] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, “Large scale distributed deep networks,” in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[16] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, “Scaling distributed machine learning with the parameter server,” in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, Broomfield, CO, 2014, pp. 583–598.

[17] L. Yang, B. Liu, J. Cao, Y. Sahni, and Z. Wang, “Joint computation partitioning and resource allocation for latency sensitive applications in mobile edge clouds,” *IEEE Transactions on Services Computing*, 2019.

[18] L. Chettri and R. Bera, “A comprehensive survey on internet of things (iot) toward 5g wireless systems,” *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 16–32, 2019.

[19] J. Howard and S. Ruder, “Universal language model fine-tuning for text classification,” in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 2018, pp. 328–339.

[20] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, “Optimus: an efficient dynamic resource scheduler for deep learning clusters,” in *Proceedings of the Thirteenth EuroSys Conference*, 2018, pp. 1–14.

[21] N. Guillotin-Plantard and C. Prieur, “Central limit theorem for sampled sums of dependent random variables,” *ESAIM: Probability and Statistics*, vol. 14, p. 299–314, 2010.

[22] H. Kellerer, U. Pferschy, and D. Pisinger, “Multidimensional knapsack problems,” in *Knapsack problems*. Springer, 2004, pp. 235–283.

[23] A. Caprara, H. Kellerer, and U. Pferschy, “A 3/4-approximation algorithm for multiple subset sum,” *Journal of Heuristics*, vol. 9, no. 2, pp. 99–111, 2003.

[24] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, “Scaling laws for neural language models,” *arXiv preprint arXiv:2001.08361*, 2020.

[25] D. B. Shmoys, E. Tardos, and K. Aardal, “Approximation algorithms for facility location problems (extended abstract),” in *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, ser. STOC ’97, 1997, pp. 265–274.

[26] J.-H. Lin and J. S. Vitter, “e-approximations with minimum packing constraint violation (extended abstract),” in *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing*, ser. STOC ’92, 1992, pp. 771–782.

[27] S. Barua, Y. Miyatani, and A. Veeraraghavan, “Direct face detection and video reconstruction from event cameras,” in *2016 IEEE winter conference on applications of computer vision (WACV)*. IEEE, 2016, pp. 1–9.

[28] P. B. Mirchandani and R. L. Francis, *Discrete location theory*, 1990.

[29] M. Sviridenko, “An improved approximation algorithm for the metric uncapacitated facility location problem,” in *International Conference on Integer Programming and Combinatorial Optimization*. Springer, 2002, pp. 240–257.

[30] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, “Dynamic service migration in mobile edge-clouds,” in *2015 IFIP Networking Conference (IFIP Networking)*. IEEE, 2015, pp. 1–9.

[31] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, “Joint service placement and request routing in multi-cell mobile edge computing networks,” in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 10–18.

[32] H. Zhou, H. Wang, X. Li, and V. C. Leung, “A survey on mobile data offloading technologies,” *IEEE Access*, vol. 6, pp. 5101–5111, 2018.

[33] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, “Communication efficient distributed machine learning with the parameter server,” in *Advances in Neural Information Processing Systems*, 2014, pp. 19–27.

[34] B. P. L. Lau, S. H. Marakkalage, Y. Zhou, N. U. Hassan, C. Yuen, M. Zhang, and U.-X. Tan, “A survey of data fusion in smart city applications,” *Information Fusion*, vol. 52, pp. 357–374, 2019.

[35] A. Sinaeepourfard, J. Garcia, X. Masip-Bruin, E. Marn-Tordera, J. Cirera, G. Grau, and F. Casaus, “Estimating smart city sensors data generation,” in *Ad Hoc NETWORKING Workshop*, 2016, pp. 1–8.

[36] J. Xu, L. Chen, and P. Zhou, “Joint service caching and task offloading for mobile edge computing in dense networks,” in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 207–215.

[37] F. N. Iandola, M. W. Moskewicz, K. Ashraf, and K. Keutzer, “Firecaffe: near-linear acceleration of deep neural network training on compute clusters,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2592–2600.

[38] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[39] J. Li, L. Feng, and S. Fang, “An greedy-based job scheduling algorithm in cloud computing.” *JSW*, vol. 9, no. 4, pp. 921–925, 2014.

[40] M. Jia, J. Cao, and W. Liang, “Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks,” *IEEE Transactions on Cloud Computing*, vol. 5, no. 4, pp. 725–737, 2015.

[41] Q. Cao, N. Balasubramanian, and A. Balasubramanian, “Mobirnn: Efficient recurrent neural network execution on mobile gpu.” New York, NY, USA: Association for Computing Machinery, 2017.

[42] S. Sarkar, V. M. Patel, and R. Chellappa, “Deep feature-based face detection on mobile devices,” in *2016 IEEE International Conference on Identity, Security and Behavior Analysis (ISBA)*. IEEE, 2016, pp. 1–8.

[43] L. Tong, Y. Li, and W. Gao, “A hierarchical edge cloud architecture for mobile computing,” in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.

[44] M. Chen and Y. Hao, “Task offloading for mobile edge computing in software defined ultra-dense network,” *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, March 2018.

[45] Y. Sun, S. Zhou, and J. Xu, “Emm: Energy-aware mobility management for mobile edge computing in ultra dense networks,” *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2637–2646, Nov 2017.

[46] S. M. S. Tanzil, O. N. Gharehshiran, and V. Krishnamurthy, “A distributed coalition game approach to femto-cloud formation,” *IEEE Transactions on Cloud Computing*, vol. 7, no. 1, pp. 129–140, Jan 2019.

[47] L. Chen and J. Xu, “Socially trusted collaborative edge computing in ultra dense networks,” in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC ’17, 2017, pp. 9:1–9:11.

[48] A. Al-Shuwaili and O. Simeone, “Energy-efficient resource allocation for mobile edge computing-based augmented reality applications,” *IEEE Wireless Communications Letters*, vol. 6, no. 3, pp. 398–401, June 2017.

[49] Y. Yu, J. Zhang, and K. B. Letaief, “Joint subcarrier and cpu time allocation for mobile edge computing,” in *2016 IEEE Global Communications Conference (GLOBECOM)*, Dec 2016, pp. 1–6.

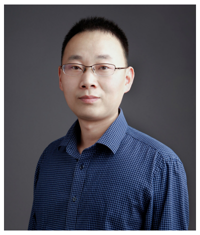
[50] G. Dán and N. Carlsson, “Dynamic content allocation for cloud-assisted service of periodic workloads,” in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 853–861.

[51] S. Shukla, O. Bhardwaj, A. A. Abouzeid, T. Salonidis, and T. He, "Hold'em caching: Proactive retention-aware caching with multi-path routing for wireless edge networks," in *Proceedings of the 18th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM, 2017, p. 24.
[52] I.-H. Hou, T. Zhao, S. Wang, and K. Chan, "Asymptotically optimal algorithm for online reconfiguration of edge-clouds," in *Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2016, pp. 291–300.
[53] M. Barcelo, J. Llorca, A. M. Tulino, and N. Raman, "The cloud service distribution problem in distributed cloud networks," in *2015 IEEE International Conference on Communications (ICC)*. IEEE, 2015, pp. 344–350.
[54] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1468–1476.
[55] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *CoRR*, vol. abs/1602.05629, 2016. [Online]. Available: http://arxiv.org/abs/1602.05629
[56] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 328–339.
[57] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, Jan 2018.
[58] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.
[59] J. Wang, A. K. Sahu, Z. Yang, G. Joshi, and S. Kar, "Matcha: Speeding up decentralized sgd via matching decomposition sampling," *arXiv preprint arXiv:1905.09435*, 2019.

Min Chen received the B.S. degree in software engineering from the Xi'an Jiaotong University, China, in 2018. He is currently pursuing his Ph.D. degree in the School of Computer Science and Technology, University of Science and Technology of China. His research interests include mobile edge computing and distributed machine learning.

Haichuan Wang got his M.S. degree in computer science and technology from University of Science and Technology of China in 2020. He got his B.S. degree in University of Science and Technology of China in 2017. His research interests include Edge Computing and Distributed Systems.

Zeyu Meng received B.S. degree in 2016 from the Sun Yat-Sen University. He is currently a Ph.D. student in the School of Cyberscience, University of Science and Technology of China. His main research interest is software defined networks, edge computing and networking for AI.



Hongli Xu (Member, IEEE) received the B.S. degree in computer science from the University of Science and Technology of China, China, in 2002, and the Ph. D degree in computer software and theory from the University of Science and Technology of China, China, in 2007. He is a professor with the School of Computer Science and Technology, University of Science and Technology of China (USTC), China. He was awarded the Outstanding Youth Science Foundation of NSFC, in 2018. He has won the best paper award or the best paper candidate in several famous conferences. He has published more than 100 papers in famous journals and conferences, including the IEEE/ACM Transactions on Networking, IEEE Transactions on Mobile Computing, IEEE Transactions on Parallel and Distributed Systems, Infocom and ICNP, etc. He has also held more than 30 patents. His main research interest is software defined networks, edge computing and Internet of Thing.

Yang Xu (Member, IEEE) is currently an associate researcher in the School of Computer Science and Technology at University of Science and Technology of China. He got his Ph.D. degree in computer science and technology from University of Science and Technology of China in 2019. He got his B.S. degree in Wuhan University of Technology in 2014. His research interests include Ubiquitous Computing, Deep Learning and Mobile Edge Computing.

Jianchun Liu is currently a Ph.D. candidate in the School of Data Science, University of Science and Technology of China (USTC). His main research interests are software defined networks, network function virtualization, edge computing and federated learning.

He Huang (Member, IEEE) received the Ph.D. degree from the School of Computer Science and Technology, University of Science and Technology of China (USTC), in 2011. He is currently a Professor with the School of Computer Science and Technology, Soochow University, China. His current research interests include traffic measurement, computer networks, and algorithmic game theory. He is a member of the ACM.