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

Joint Job Offloading and Resource Allocation for Distributed Deep Learning in Edge Computing

Haichuan Wang, Xi Chen, Hongli Xu, Jianchun Liu, Liusheng Huang
School of Computer Science and Technology
University of Science and Technology of China
Hefei, China
Email: {none, vcx15, lyl617}@mail.ustc.edu.cn, {xuhongli, lshuang}@ustc.edu.cn

***Abstract*—Under the paradigm of Edge Computing, the enormous data generated at the network edge can be processed locally. Machine learning methods are often adopted to make full utilization of these data, among which the deep learning is a promising one. When considering the inherent distributed feature of these data, it is appeal to conduct distributed deep learning tasks directly at the edge. We focus on an edge computing system that conduct distributed deep learning tasks using gradient-descent based approaches. To ensure the system's performance, there are at least two major challenges to cope with: how to offload the training jobs with multiple data source nodes and how to allocate the limited resources on each edge server among training jobs. In this paper, we jointly consider the two challenges, aiming to maximize the system throughput while ensuring the system's quality of service (QoS). We formulate the joint problem as an integer non-linear program and propose an efficient approximation algorithm based on reformulation and randomized rounding technique. Simulation results prove that the proposed algorithm can improve 56% of the system throughput and 53% of resource utilization when compared to the conventional baseline algorithms.**

***Index Terms*—Edge Computing, Distributed Deep Learning, Job Offloading, Resource Allocation**

I. Introduction

With the advent of paradigms like the Internet of Things (IoT), social networking and smart city, data will be abundantly available [1] at the network edge. Cisco, for example, estimates that the IoT alone will generate over 400ZB of data annually by 2020 [2]. To extract useful information from the generated data, machine learning methods are often adopted. For example, Google Cloud Speech is powered by the machine learning framework, TensorFlow [3]. Among these machine learning methods, deep learning is a promising one as it shows satisfying performance for many problems with large data set. Since model training is resource-intensive, it is naturally required to send the data to the remote cloud with sufficient resources to be processed [4], [5]. However, sending the collected data to the remote cloud may introduce heavy burden to the backbone network, resulting in undesired latency. Even more, it may lead to privacy leakage, which contradicts with the increasing concern about users' data privacy. Therefore, sending the raw data to the remote cloud is often considered unrealistic and unnecessary.

Edge Computing [6], as a new computing paradigm, has emerged to move data and service functions from the remote cloud to the logical edge of the network, enabling local processing of the enormous user data. Compared with cloud computing, edge computing has two significant advantages. First, since the raw data is only transmitted to the nearby edge servers, i.e., without transmitting to the remote cloud, the latency as well as the network overhead are thus reduced considerably. Second, as the raw data no longer needs to be transmitted to the remote cloud, user privacy can be abundantly preserved [7]. Therefore, with the respect to latency and privacy concerns, it is an attractive way to train the model locally.

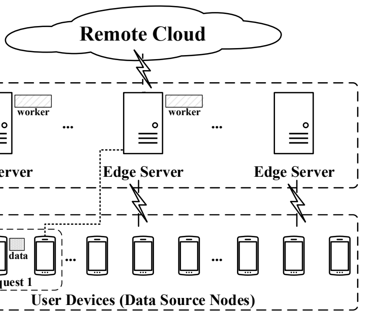

Fig. 1. Illustration of the system architecture, where training tasks are generated from the data source nodes and workers are deployed on edge servers to conduct these tasks.

There are typically two kinds of ways to conduct model training at the edge. In the first way, models are trained separately on different edge devices using their own data even though the models are of the same kind. Previous works [8], [9] mainly focus on how to train the model on a single mobile device or how to offload the training task of the mobile device to the edge server. However, the amount of data on a single edge device is often too small to train a model with desired accuracy for many applications. In the second way, data from different devices are used to train a mutual model, and the training is often done in a distributed way. Edge devices can either join the distributed training and process their own data locally or upload the data to the edge servers to be processed. Previous works [10], [11] mainly focus on the distributed training of a single task, assuming that the data can always be processed locally or the data are always ready on edge servers.

While in a practical edge computing system, there are often multiple training tasks and most data source nodes such as sensors have to upload their data.

We consider the scenario where we deploy workers at the edge servers to collect the data and train the model in a distributed way, as illustrated in Fig. 1. We focus on training models using asynchronous stochastic descent gradient (SGD) based approaches and data parallelism under the parameter server [12] framework. When conducting asynchronous SGD at the edge, the workers, deployed at the edge servers, firstly collect the data from the data nodes that are associated to each individual job request. Each worker then divides the collected data into mini-batches and computes the gradients one by one. After processing several mini-batches, the worker sends the gradients to the parameter servers. Upon receiving the gradients from one worker, the parameter servers perform global parameter updates, and send the updated parameters back to the worker [11]. When the worker receives the updated parameters, it re-computes the gradients using the following mini-batches. After processing the entire collected data, the worker starts another epoch to compute gradients until the required epochs have been achieved. The training workflow is illustrated in Fig. 2.

With the emerging of edge intelligence, the edge computing system will face more and more distributed training jobs. In this paper, we expect to maximize the system throughput by performing job offloading and resource allocation in an optimal way and focus on the two critical challenges: how to offload the job requests with multiple data source nodes to maximize the system throughput and how to allocate the limited resources(e.g., computation and communication resources) among parallel training tasks on each edge server to ensure the system's QoS. The main contributions of this paper are summarized as follows:

- We look into the compelling problem of job offloading with multiple data source nodes in edge computing, which has rarely been studied in related previous works as far as we know.
- We jointly consider the job offloading and resource allocation problems and comprehensively formalize the joint problem to maximize the system throughput while ensuring the system's QoS.
- We develop an efficient algorithm by randomized rounding to make optimal job offloading and resource allocation decisions. We further prove the algorithm's performance guarantee based on central limit theorem.
- We conduct extensive experiments to validate the efficiency of the proposed algorithm. The results validate that the algorithm can improve 56% of the system throughput and 53% of resource utilization when compared to the baseline algorithms.

The remainder of this paper is organized as follows: In Section II, we discuss the related works. In Section III, we give the formal definitions of the problem. In Section IV, we propose our algorithm and analyze its performance. We validate our

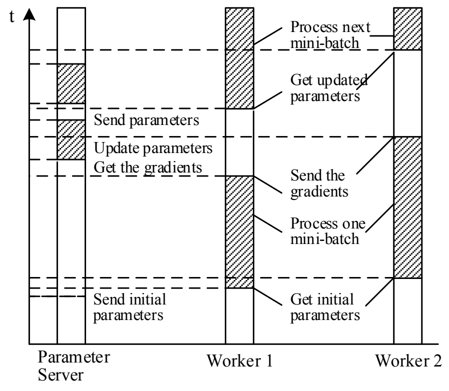

Fig. 2. Illustration of asynchronous SGD based distributed model training workflow when adopting the *Parameter Server* framework and data parallelism.

algorithm's efficiency and optimum by simulations in Section V and conclude in Section VI .

II. Related Works

Job offloading is the central topic of many previous works about edge computing. There have been various works that study job offloading from different points of view. To reduce the task duration, the authors in [13] designed a hierarchical edge computing architecture, based on which they proposed an optimal offloading scheme. In order to keep the delay minimized and save the battery life of user's equipment, works in [14] transformed the job offloading problem into two sub-problems and proposed the respective solutions. In [15], the authors considered the job offloading problem from the perspective of energy saving, a novel user-centric energy-aware mobility management scheme is developed. Works in [16], [17] considered the job offloading from the collaboration view. These works only consider the situation in which each job has only one end device, the offloading of multi-node job has rarely been studied.

Resource allocation is another important research point in edge computing. The authors in [18] proposed efficient algorithms to choose the data centers, meanwhile, by making use of structured network in data center, they developed optimal algorithm for rack and server selection, they also developed heuristic for partitioning the requested resources amongst the selected data centers and racks. In [19], 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 is developed in [20] 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.

There have been a bunch of prior works about conducting machine learning tasks at the edge. A concept termed federated learning is proposed in [10], in which mobile phones and IoT devices can be used to learn a shared model in a decentralized approach, which is a variant of distributed gradient descent. The authors in [11] analyzed the convergence rate of distributed gradient descent from a theoretical point of view and proposed a control algorithm, which determines the best trade-off between local update and global parameter aggregation. In [21], the authors proposed distributed deep neural network over hierarchies consisting of the cloud, the edge and end devices, which significantly reduces the communication cost. The authors in [22] introduced deep learning for IoTs into the edge computing environment and designed novel offloading strategy to optimize the performance. While these previous works focus on conducting training tasks at the edge, they ignored the joint optimization of job offloading and resource allocation for these training tasks, which is of vital importance for the resource limited edge computing environment.

III. Problem Formulation

A. System Model

Assume we are given a set $N = \{1, \cdots, n\}$ of data nodes, and a set $M = \{1, \cdots, m\}$ of edge servers. With every data node $i \in N$ there are associated with some data of amount d_i, which is assumed to be enough for single node model training. Further, for each edge server $j \in M$, the amount of its available type-γ resource is r_j^γ, for example, the amount of storage resource is denoted as r_j^s. The data from the data nodes serve a set $P = \{1, \cdots, p\}$ of training job requests. In addition, for every job request $k \in P$, its job completion time is associated with an upper bound τ_k, indicating that it should be completed within τ_k when accepted.

Due to some factors such as distance, each data node $i \in N$ has its own assignable edge servers, we denote the assignable edge servers of data node i as subset $A_i \subset M$. During job offloading, each data node can only be assigned to one of its assignable edge servers. Meanwhile, we denote the subset of candidate data nodes that can be assigned to edge server $j \in M$ as $B_j \subset N$ and the subset of data nodes that serve job request $k \in P$ as $C_k \subset N$. These sets define a bipartite graph G with color classes M and N, we define the degree of data node i as $|A_i|$ and the degree of edge server j as $|B_j|$. Throughout this paper, we assume that each data node serves only one job, i.e., $C_{k1} \cap C_{k2} = \emptyset, k1, k2 \in P, k1 \neq k2$.

B. Job Offloading

We use binary variable y_k to express the admission decision of job request $k \in P$. That is, if $y_k = 1$, it will be accepted and served by the system, otherwise it is rejected. Rejected job requests can be resubmitted or queued. For an accepted job request, we use binary variables x_i^j to model the data node assignment scheme. For data node $i \in N$, if $x_i^j = 1, j \in M$, a worker will be deployed on edge server j (via containers or virtual machines) to collect its data and conduct distributed model training with the coordination of the parameter servers.

Therefore, the data collected on edge server j for job request k can be derived as

$$D_j^k = \sum_{i \in C_k} x_i^j d_i \tag{1}$$

C. Resource Allocation

We denote the amount of type-γ resource allocated to job request k on edge server j as $r_{j,k}^\gamma$. Specifically, we allocate computation resource to job k on edge server j with the amount $r_{j,k}^c$, and with the amount $r_{j,k}^b$ of communication resource. Further, we denote the time used to train the network model for job request k on edge server j as $t_{j,k}$, which consists of the computing time $t_{j,k}^{comp}$ and the communication time $t_{j,k}^{comm}$.

The time used to train the mini-batches adds up to the computing time. We assume that the mini-batch size of job request k is fixed and denote it as f_k. Therefore, the float operations needed to process one mini-batch of job request k is fixed, we denote it as g_k. Further, the number of iterations during one epoch is fixed and can be derived as $\frac{D_j^k}{f_k}$ for job request k on edge server j. As our survey and experiment result show, the computing time has linear relationship with float point operations and is inversely proportional to computation resource. We calculate the computing time of job request k in one epoch on edge server j as follows:

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

We use h_k to denote the parameters' size of job request k, with gradients' size same as parameters'. Suppose we perform global update every λ_k iterations for job request k, thus the number of communication times during one epoch is $\frac{1}{\lambda_k} \cdot \frac{D_j^k}{f_k}$ on edge server j. Therefore, we calculate the communication time of job request k on edge server j during one epoch as

$$t_{j,k}^{comm} = \frac{1}{\lambda_k} \cdot \frac{D_j^k}{f_k} \cdot \frac{h_k}{r_{j,k}^b} \tag{3}$$

In practical, deep learning models are usually non-convex, we can not expect the training epochs required for the model convergence. However, different from experimental models, production models often have been tested many times in different situations and the hyper-parameters have been tuned well during the experimental phase, thus they are mature enough and can typically converge to the optimum very well in desired training epochs. We suppose the training process of job request k includes T_k epochs, thus the total training time of job request k on edge server j is

$$t_{j,k} = T_k \cdot (t_{j,k}^{comp} + t_{j,k}^{comm}) \tag{4}$$

D. Joint Optimization

To maximize the system throughput and ensure the system's QoS, we jointly consider the Job offloading and Resource allocation Problem (JRP) for distributed deep learning at the edge and formulate it as follows:

$$\max \sum_{k=1}^{p} y_k \tag{5}$$

$$\begin{aligned}
S.t. \sum_{j\in A_i} x_i^j &= y_k, & \forall k \in P, \forall i \in C_k \quad (6)\\
\sum_{i\in B_j} x_i^j d_i &\le r_j^s, & \forall j \in M \quad (7)\\
\sum_{k=1}^{p} r_{j,k}^c &\le r_j^c, & \forall j \in M \quad (8)\\
\sum_{k=1}^{p} r_{j,k}^b &\le r_j^b, & \forall j \in M \quad (9)\\
t_{j,k} &\le \tau_k, & \forall k \in P, \forall j \in M \quad (10)\\
y_k &\in \{0,1\}, & \forall k \in P \quad (11)\\
x_i^j &\in \{0,1\}, & \forall i \in N, \forall j \in M \quad (12)\\
r_{j,k}^c, r_{j,k}^b &\ge 0, & \forall j \in M, \forall k \in P \quad (13)
\end{aligned}$$

In this system of constraints, Eq. (6) represents the assignment for the data nodes of each job request. For a data node, if its associated job request is accepted by the system, one and only one worker will be deployed on a certain edge server to collect its data and train the respective model. Otherwise, it will not be assigned to any edge server. Eq. (7) indicates that the amount of the collected data on each edge server should not exceed its storage capacity. Eq. (8) and Eq. (9) ensure that the resources allocated to the accepted job requests will not exceed the capacity on each edge server. Eq. (10) states that the completion time of each job should not exceed the respective upper bound. Our objective is to maximize the system throughput, as Eq. (5) indicates.

The JRP is formulated as a mixed integer non-linear program above. Intuitively, when we regard the data nodes and edge servers as the items and knapsacks in multiple knapsack problem (MKP), the JRP is similar to MKP. However, there are basically two main differences between JRP and MKP even if we ignore the resource allocation process. On one hand, in JRP, each data node can be assigned to only a subset of edge servers. On the other hand, for the data nodes of the same job request, their assignments are not independent, either all of them are assigned or none of them is assigned. Nevertheless, we consider a special case of JRP, in which the upper bound of completion time τ_k is infinite for job request k so that we can ignore those constraints related to the resource allocation. Further, we assume each job request has only one data node, and each data node can be assigned to all the edge servers. As a result, our problem becomes MKP, thus MKP is a special case of JRP. Since the MKP is NP-hard, JRP is thus NP-hard in the strong sense.

IV. Algorithm Description

A. Relaxation and Reformulation

To circumvent the NP-hardness of the JRP, we relax the integer constraints Eq. (11) and Eq. (12) to

$$y_k \in [0,1], \forall k \in P \tag{14}$$

$$x_i^j \in [0,1], \forall i \in N, \forall j \in M \tag{15}$$

Meanwhile, we notice that in a practical distributed deep learning system, in order to ensure the system's overall performance, the communication time of a job is often restricted. Therefore, we introduce the system-wide configurable coefficient $\epsilon \in (0,1)$, which indicates the ratio of communication time to the total time of the accepted job requests. In order to facilitate the non-linear constraints in Eq. (10), we transform them into two sets of linear sub constraints by making use of ϵ as follows:

$$T_k h_k D_j^k - \epsilon \tau_k \eta_k f_k r_{j,k}^b \le 0, \forall k \in P, \forall j \in M \tag{16}$$

$$T_k g_k D_j^k - (1-\epsilon)\tau_k f_k r_{j,k}^c \le 0, \forall k \in P, \forall j \in M \tag{17}$$

After the relaxation and reformulation, we get the linear program with respect to $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{r}^c$, $\boldsymbol{r}^b$. Given an instance I of JRP, we denote this linear program by LP_I. We can solve LP_I efficiently and get the optimal vertex solution, which may not be feasible for the integer constraints of the original problem. Based on the solution to LP_I, we propose our approximation algorithm by adopting randomized rounding technique.

B. Approximation Algorithm

In the optimization for JRP, we consider the case in which the set of job requests P is given. This is feasible in a typical parallel distributed system, where we have to schedule a set of queued job requests from time to time. For the sake of brevity, we say job $k \in P$ is non-trivial if $\forall i \in C_k$ there exist at least one edge server $j \in M_i$ that satisfies $r_j^s \ge d_i$.

Given the instance I, we develop the algorithm in an iterative scheme. In each iteration, we first perform pruning on I. The pruning is performed as follows, for data node $i \in N$, we remove edge server j from A_i and remove i from B_j if $r_j^s < d_i$. For edge server $j \in M$, we first group the remaining data nodes in B_j by the job request to which they belong as $B_j^1, B_j^2, \cdots, B_j^p$. For the data nodes in group $B_j^k, k \in \{1,2,\cdots,p\}$, we order them in non-decreasing degree order, let b_j^k be the maximum data node in ordered B_j^k such that

$$\sum_{i\in B_j^k | i \le b_j^k} d_i \le r_j^s \tag{18}$$

$$T_k h_k \sum_{i\in B_j^k | i \le b_j^k} x_i^j d_i - \epsilon \tau_k \eta_k f_k r_j^b \le 0 \tag{19}$$

$$T_k g_k \sum_{i\in B_j^k | i \le b_j^k} x_i^j d_i - (1-\epsilon)\tau_k f_k r_j^c \le 0 \tag{20}$$

we remove the data nodes in $\{i \in B_j^k | i > b_j^k\}$ from B_j and remove j from their assignable edge server set.

After the pruning, we reject those trivial job requests. When a job request is rejected, we remove it and the corresponding data nodes from I. If the current job request set P is empty, the algorithm finishes, otherwise, we construct LP_I and find an optimal vertex solution $\{\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{r}}^c, \hat{\boldsymbol{r}}^b\}$ to it. For the data nodes in C_k, 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. For data node $i \in N$, we introduce the independent discrete random variable e_i, with sample space $\{1, 2, \cdots, m\}$ and probability distribution given by $\hat{x}_i^j$, i.e, $Pr(e_i = j) = \hat{x}_i^j, (j = 1, 2, \cdots, m)$ and $Pr(e_i = 0) = 1 - \sum_{j=1}^m \hat{x}_i^j$. Therefore, based on e_i we obtain the preliminary integer data node assignment.

However, this preliminary assignment may not be feasible, as the storage constraint or completion time constraint may be violated. We denote the set of data nodes that are assigned to edge server j as B_j' in the preliminary assignment and order them in non-increasing priority order. Let b_j be the minimum data node in ordered B_j' such that

$$\sum_{i \in B_j' | i \geq b_j} d_i \leq r_j^s \tag{21}$$

Then, in order to get a feasible assignment, we only assign the data nodes in $\{i \in B_j' | i \geq b_j\}$ to edge server j. For a job request, if all of its data nodes are assigned, it is accepted. Further, to maintain the job completion time achieved previously, we update resources allocated to the accepted jobs on each server proportionally based on the solution to LP_I. Finally, we remove the accepted job requests together with the corresponding data nodes from I and update the available resources on each edge server. After updating I, the algorithm goes on another iteration. The algorithm is formally described in Alg. 1.

Proposition 1: Alg. 1 finishes in at most p iterations.

Proof: At the beginning of each iteration, the pruning on I ensures that edge server $j \in M$ can cope with the extreme situation in which all the candidate data nodes from job request $k \in P$ are assigned to it. Thus in each iteration's preliminary assignment, based on the solution to LP_I, the data nodes of the job request with the highest priority can always be assigned. As a result, there is at least one job request accepted in each iteration or the algorithm finishes. Since the size of the initial job requests set is p, Alg. 1 finishes in at most p iterations.

Algorithm 1 Approximation algorithm for JRP

Input: Initial JRP instance I

```
1: while $true$ do
2:     PRUNE($I$);
3:     if $P = \emptyset$ then
4:         The algorithm ends;
5:     else
6:         Solve $LP_I$ and get $\{\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{r}}^c, \hat{\boldsymbol{r}}^b\}$;
7:         Set the priority of $i \in C_k$ as $\hat{y}_k$;
8:         Obtain $\tilde{\boldsymbol{x}}$ based on $\hat{\boldsymbol{x}}$ by randomized rounding;
9:         for $j \in M$ do
10:            Set $B_j' \leftarrow \{i \in B_j | \tilde{x}_i^j = 1\}$;
11:            Order $B_j'$ in non-increasing priority order;
12:            Find minimum $b_j \in B_j'$ such that Eq. (21) is satisfied;
13:            Set $x_i^j \leftarrow 1, i \in B_j', i \geq b_j$;
14:        end for
15:        for $k \in P$ do
16:            Set $y_k \leftarrow 1$ if $\sum_{j \in M} x_i^j = 1, \forall i \in C_k$;
17:            if $y_k = 1$ then
18:                Update $r_{j,k}^c, r_{j,k}^b, j \in M$ proportionally to satisfy Eq. (16) and Eq. (17);
19:                Set $P \leftarrow P - \{k\}, N \leftarrow N - C_k$;
20:            end if
21:        end for
22:        Update $I$;
23:    end if
24: end while
25: function PRUNE($I$)
26:    Set $A_i \leftarrow A_i - \{j \in A_i | r_s^j < d_i\}$;
27:    Set $B_j \leftarrow B_j - \{i \in B_j | r_s^j < d_i\}$;
28:    for $k \in P$ do
29:        Set $B_j^k \leftarrow \{i \in B_j | i \in C_k\}$;
30:        Order $B_j^k$ in non-decreasing degree order;
31:        Find maximum $b_j^k \in B_j^k$ such that Eq. (18), Eq. (19) and Eq. (20) are satisfied;
32:        Set $\hat{B}_j^k \leftarrow \{i \in B_j^k | i > b_j^k\}$;
33:        Set $B_j \leftarrow B_j - \hat{B}_j^k$, $A_i \leftarrow A_i - \{j\}, \forall i \in \hat{B}_j^k$;
34:        Set $P \leftarrow P - \{k\}, N \leftarrow N - C_k$ if $k$ is trivial;
35:    end for
36: end function
```

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

C. Theoretical Analysis

According to Eq. (6), the system throughput of the proposed algorithm is $\sum_{k=1}^p \sum_{j \in A_i} x_i^j, i \in C_k$. The iterative feature of Alg. 1 makes it difficult to derive the approximation performance for the system throughput directly. We notice that the job admission decision is made based on the data node assignment on each edge server, which enlightens us on analyzing the performance of the proposed algorithm through data node assignment on the edge server. We denote the total data assigned to edge server $j \in M$ by the preliminary assignment as the random variable

$$D_j = \sum_{i \in B_j} d_i \cdot \varphi(e_i = j) \tag{22}$$

here $\varphi(e_i = j)$ is 1 if $e_i = j$ and 0 otherwise. Eq. (22) indicates that D_j is a sum of independent random variables. Thus we consider the data node assignment process for edge server j as $|B_j|$ *Poisson* tests, in which data node i is assigned to j with probability $\hat{x}_i^j$. We define $\mu_j = E(D_j)$ and $\sigma_j^2 = Var(D_j)$. According to the Chernoff Bound, we get the following probabilistic tail estimate:

$$Pr\left(D_j < (1-\delta)\mu_j\right) < e^{-\frac{\mu_j \delta^2}{2}} \tag{23}$$

for every δ with $0 < \delta < 1$. Eq. (23) gives an upper bound on the probability that the data amount on edge server j does not exceed a certain fraction of its expectation in the preliminary assignment.

We analyze the lower bound of the proposed algorithm's performance based on the central limit theorem [23]. Firstly, we assume n is large and the data amounts $d_i, i \in N$ are uniformly bounded. Suppose there is an ε such that $x_i^j \in [\varepsilon, 1-\varepsilon]$ is satisfied for most data nodes, which ensures that the central limit theorem can hold. Therefore, according to the central limit theorem, we may conclude that $(D_j - \mu_j)/\sigma_j^2 \sim N(0,1)$. Now we formally prove the performance guarantee of our algorithm.

Theorem 1: Let OPT_I denote the optimal value of LP_I and η denote the maximum number of data nodes that any edge server can serve. Assume that the central limit theorem can hold for $j \in M$. The approximate performance lower bound of the proposed algorithm is

$$\left(1 - \frac{1}{2\eta} - \frac{1}{\sqrt{2\eta\pi}}\right) \cdot OPT_I$$

Proof: Let D_j' be the final data amount assigned to edge server j by the proposed algorithm in each iteration. Therefore, if $D_j \le r_s^j$, D_j' is equal to D_j, otherwise we can only expect $D_j' \ge \mu_j - d_{max}$, where $d_{max} = max_k \sum_{i \in B_j^k} d_i$. However, when the storage constraint is violated on edge server j, we remove those data nodes with lower priority until the storage constraint is satisfied. After removing those data nodes, edge server j is assigned at least $1 - \frac{1}{\eta}$ of the total capacity, i.e, $D_j' \ge (1 - \frac{1}{\eta}) r_s^j$. Therefore

$$E(D_j') \ge \int_0^{r_s^j} z Pr(z) dz + (1 - \frac{1}{\eta}) r_s^j Pr(D_j > r_s^j) \tag{24}$$

Meanwhile, we may assume that $\mu_j = r_s^j$ without generality, since the central theorem can hold and $(D_j - \mu_j)/\sigma_j^2 \sim N(0,1)$, we can derive that

$$E(D_j') \ge (1 - \frac{1}{2\eta})\mu_j - \frac{\sigma_j}{\sqrt{2\pi}} \tag{25}$$

$$\sigma_j^2 = \sum_{i \in B_j} d_i^2 x_i^j (1 - x_i^j) \le d_{max}\mu_j \le \frac{\mu_j^2}{\eta} \tag{26}$$

Therefore,

$$\begin{aligned} E(D_j') &\ge (1 - \frac{1}{2\eta})\mu_j - \frac{1}{\sqrt{2\pi}} \cdot \frac{\mu_j}{\sqrt{\eta}} \\ &= (1 - \frac{1}{2\eta} - \frac{1}{\sqrt{2\pi\eta}})\mu_j \end{aligned} \tag{27}$$

The theorem thus follows.

V. Performance Evaluation

A. Performance Metrics and Benchmarks

For performance comparison, we adopt the following metrics:

- *System throughput*. The system throughput is the major objective, thus we take the system throughput, i.e. number of accepted job requests as a metric. We formally denote the system throughput as $L = \sum_{k \in P} y_k$.
- *Utilization rate*. To fully utilize the limited resources on the edge servers, the storage utilization rate is of great importance. Therefore, we take the average storage utilization rate as another metric, which can be modeled as $\kappa = \frac{1}{m}\sum_{j=1}^{m} \frac{\sum_{i=1}^{n} x_i^j d_i}{r_j^s}$.

Based on these two metrics, we compare the proposed algorithm with several baseline algorithms. These baseline algorithms initially follow the same philosophy to decide which job request to accept. Intuitively, to maximize the system's throughput, these algorithms iteratively choose the job request with minimum data amount and allocate the resources based on the completion time constraints until no more job request can be accommodated by thegt system. The difference lies in how to assign the accepted job request's data nodes. The first baseline algorithm is a randomized algorithm, which randomly chooses a qualified edge server for the data nodes of the accepted job request. The second baseline algorithm is a greedy load balancing algorithm. Specifically, for a data node of the accepted job request, the algorithm orders the data node's assignable edge servers in non-decreasing storage usage order and choose the first edge server that can satisfy the time constraint for the respective job request in the ordered sequence. Since our algorithm is based on randomized rounding, we regard the solution of LP_I in the first iteration of Alg. 1 as the optimal solution.

B. Simulation Setup

In order to mimic a practical edge computing environment, we simulate an area which is divided into 19 hexagonal cells, as depicted in Fig. 3. Within each of these hexagonal cells, there exists one edge server (typically resides in a small cluster), on which we deploy workers for the accepted job requests. For each edge server, we set its floating point operation capability as 150GFLOPS and set the storage capacity between 100GB to 200GB. Further, since the edge servers and

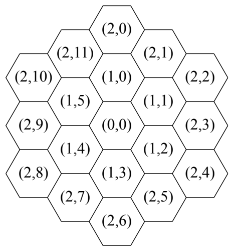

Fig. 3. Illustration of the simulated area.

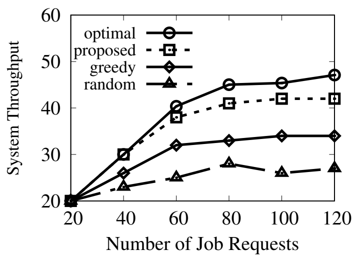

Fig. 4. System throughput for uniform distribution.

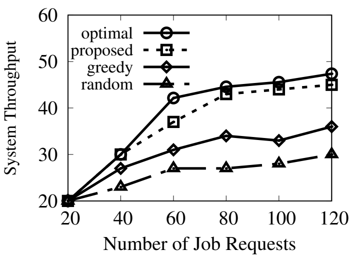

Fig. 5. System throughput for normal distribution.

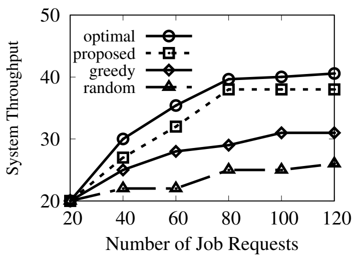

Fig. 6. System throughput for Pareto distribution.

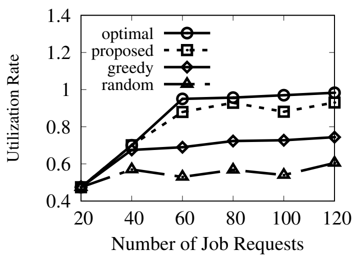

Fig. 7. Utilization rate for uniform distribution.

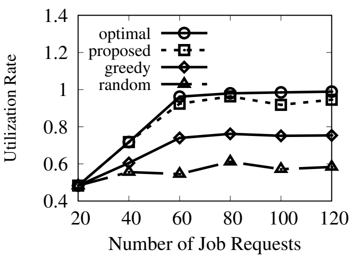

Fig. 8. Utilization rate for normal distribution.

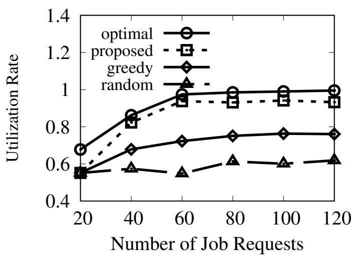

Fig. 9. Utilization rate for Pareto distribution.

parameter servers are usually connected by dedicated fibre networks at the edge, we set the available communication resource of each edge server as 10Gbps.

As for the data nodes of each job request, we perform our experiments over three kinds of data amount distributions, i.e, uniform distribution, normal distribution and Pareto distribution, which correspond to the possible realistic application scenarios. We set the data amount according to [24]. Specifically, in uniform distribution, we set the range of the distribution as 2GB to 8GB. In normal distribution, we set the mean value of the distribution as 5GB, and the standard deviation as 1. In Pareto distribution, we set the minimal value of the distribution as 2GB, and set the shape value as 2. Meanwhile, we assume that the number of the data nodes that serve the same job request is about 15. For each data node, we randomly set the cell where it is located within the simulated area. Moreover, we assume that a data node can be assigned to the edge server in the same cell and those in the neighbor cells.

Inspired by real productive models [25], we set the size of parameters (gradients as well) for different training job requests from 30MB to 575MB. Although data parallelism is adopted among edge servers, model parallelism can be adopted within each edge server to handle models of large size. The mini-batch size is set about 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 [26]. For each job request, we set the number of iterations between global update between 3 and 8. Moreover, we set the job completion time ranges from 1 hour to 2 hours.

C. Numerical Results

Figs. 4-6 present the system throughput of the algorithms in the evaluation instances with different data amount distribution. We observe that when the total resource demand to the system is moderate, these algorithms exhibit relatively close throughput performance. As the number of job requests grows, however, the advantage of our proposed algorithm becomes clear. Specifically, in Fig. 4, when the data amount follows uniform distribution, compared with greedy algorithm and random algorithm, it is shown that the proposed algorithm can achieve more than 24% and 56% the throughput respectively. In Fig. 5, when the data amount follows normal distribution, the throughput of the proposed algorithm is 25% and 50% more than the other two algorithms respectively. Similar results are given in Fig. 6. Meanwhile, the proposed algorithm can achieve at least 89% of the optimal performance in terms of system throughput.

Figs. 7-9 illustrate the algorithm performances in terms of the average storage utilization rate. It is shown that when the system's resources are sufficient, the algorithms achieve similar performance. However, with the growth of the number of job requests, the utilization rate of each algorithm gradually reaches their respective peaks, from which we can observe that the proposed algorithm performs much better than the baseline algorithms. In particular, as Fig. 7 shows, the utilization rate of the proposed algorithm can be 25% and 53% more than the greedy and random algorithm respectively. Similar results can also be derived in Fig. 8 and Fig. 9. Moreover, from the perspective of utilization rate, the proposed algorithm can achieve at least 93% of the optimal performance.

Furthermore, we observe that the utilization rate of the optimal algorithm can always reach the peak near 1, while the peaks of the other algorithms never reach that point. That is because the assignment provided by the solution to the LP_I can contain fractional values, while in the other algorithms, even if the storage capacity is sufficient, the completion time constraint will limit the utilization rate. Meanwhile, it is observed that although the throughput in Pareto distribution is less than that in other distributions, the utilization rate is similar or even higher, as Fig. 6 and Fig. 9 show, which is because in the Pareto distribution, most of the data nodes' data amount is much larger than that in other distributions.

VI. Conclusion

In this paper, we have studied joint job offloading and resource allocation for distributed deep learning oriented edge computing system. The joint problem is first formulated as a mixed integer non-linear program to maximize the system throughput while ensuring the system's QoS. To make the problem tractable, we relaxed and reformulated some constraints to get a linear program. Based on the linear program, we proposed an efficient algorithm with provable performance guarantee. As far as we know, this is the first study of job offloading with multiple data nodes in edge computing.

Acknowledgment

This search is supported by the National Natural Science Foundation of China (NSFC) under Grant 61822210, and Grant U1709217; NSF of Jiangsu in China under No. BK20161257; by Anhui Initiative in Quantum Information Technologies under No. AHY150300. The corresponding author of this paper is Hongli Xu.

References

[1] 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.

[2] G. C. Index, "Cisco global cloud index: Forecast and methodology, 2013 c2018."

[3] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[4] 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. New York, NY, USA: ACM, 2011, pp. 530–533. [Online]. Available: http://doi.acm.org/10.1145/1951365.1951432

[5] 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. [Online]. Available: https://doi.org/10.14778/2212351.2212354

[6] 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.

[7] 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.

[8] 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. Piscataway, NJ, USA: IEEE Press, 2016, pp. 23:1–23:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2959355.2959378

[9] 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.

[10] 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

[11] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," *CoRR*, vol. abs/1804.05271, 2018. [Online]. Available: http://arxiv.org/abs/1804.05271

[12] 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: USENIX Association, 2014, pp. 583–598. [Online]. Available: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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. New York, NY, USA: ACM, 2017, pp. 9:1–9:11. [Online]. Available: http://doi.acm.org/10.1145/3132211.3134451

[18] M. Alicherry and T. V. Lakshman, "Network aware resource allocation in distributed clouds," in *2012 Proceedings IEEE INFOCOM*, March 2012, pp. 963–971.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] N. Guillotin-Plantard and C. Prieur, "Central limit theorem for sampled sums of dependent random variables," *ESAIM: Probability and Statistics*, vol. 14, p. 299314, 2010.

[24] 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.

[25] F. Iandola, M. W. Moskewicz, K. Ashraf, and K. Keutzer, "Firecaffe: Near-linear acceleration of deep neural network training on compute clusters," 06 2016, pp. 2592–2600.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385