Received 22 March 2020; revised 29 April 2020; accepted 11 May 2020. Date of publication 15 May 2020; date of current version 2 June 2020.

Digital Object Identifier 10.1109/OJCOMS.2020.2994737

HierTrain: Fast Hierarchical Edge AI Learning With Hybrid Parallelism in Mobile-Edge-Cloud Computing

DEYIN LIU, XU CHEN, ZHI ZHOU, AND QING LING

School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China

CORRESPONDING AUTHOR: X. CHEN (e-mail: chenxu35@mail.sysu.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB1001703, in part by the National Science Foundation of China under Grant U1711265, Grant 61802449, Grant 61972432, and Grant 61973324, in part by the Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant 2017ZT07X355, in part by the Guangdong Natural Science Funds under Grant 2018A030313032, and in part by the Program for Basic and Applied Basic Research Fund of Guangdong under Grant 2019A1515010030.

ABSTRACT Nowadays, deep neural networks (DNNs) are the core enablers for many emerging edge AI applications. Conventional approaches for training DNNs are generally implemented at central servers or cloud centers for centralized learning, which is typically time-consuming and resource-demanding due to the transmission of a large number of data samples from the edge device to the remote cloud. To overcome these disadvantages, we consider accelerating the learning process of DNNs on the Mobile-Edge-Cloud Computing (MECC) paradigm. In this paper, we propose HierTrain, a hierarchical edge AI learning framework, which efficiently deploys the DNN training task over the hierarchical MECC architecture. We develop a novel *hybrid parallelism* method, which is the key to HierTrain, to adaptively assign the DNN model layers and the data samples across the three levels of the edge device, edge server and cloud center. We then formulate the problem of scheduling the DNN training tasks at both layer-granularity and sample-granularity. Solving this optimization problem enables us to achieve the minimum training time. We further implement a hardware prototype consisting of an edge device, an edge server and a cloud server, and conduct extensive experiments on it. Experimental results demonstrate that HierTrain can achieve up to 6.9× speedups compared to the cloud-based hierarchical training approach.

INDEX TERMS Edge AI, deep learning, fast model training, mobile-edge-cloud computing.

I. INTRODUCTION

IN RECENT years, deep learning has become a popular research topic and been integrated into a large number of applications, including image recognition [1], natural language processing [2], recommendation systems [3], to name a few. Moreover, empowered by edge computing, many real-time deep learning based edge AI applications are emerging in various domains such as smart healthcare, smart robots, and industrial IoT [4].

As a data-driven approach, deep learning based edge AI typically requires to have adequate data samples, from which deep neural networks (DNNs) are trained to extract features or attributes. These data samples are often generated by mobile and IoT devices at the network edge that have limited communication and computation capabilities, such as mobile phones, smart watches, smart robots, etc. Therefore, how to efficiently utilize the communication and computation capabilities of edge devices to train DNNs with the generated data samples will be a vital issue for many emerging edge AI applications.

One solution to this problem is cloud computing [5], [6], which allows edge devices to offload their data samples to a cloud center. Then, the resource-intensive task of training a DNN is conducted in the cloud center, often implemented in parallel on multiple computing units. Despite cloud computing provides almost unlimited computation resources, the major concern comes from the high data transmission latency and overhead over the Internet, which slows down the training process and hinders the real-time model update. Another solution is to train the DNN in a fully decentralized peer-to-peer manner [7], [8]. This approach avoids the communication overhead between the edge devices and the cloud

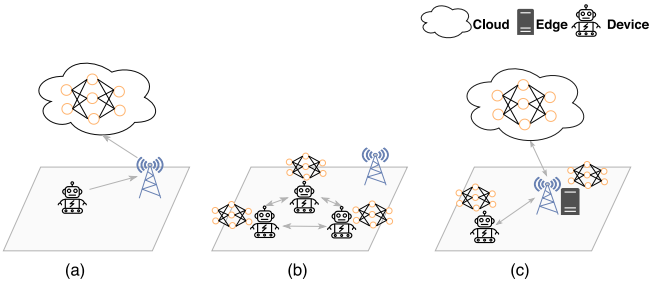


FIGURE 1. Training a DNN: (a) on the cloud center; (b) on the edge devices in a fully decentralized peer-to-peer manner; (c) on the mobile-edge-cloud hierarchical architecture. Here (a) and (b) belong to *horizontal training*, while (c) is *hierarchical training*.

center. Nevertheless, when the computation resources of the edge devices are limited, solely relying on them to train the DNN is impractical, or may cause significant computation delay. We classify these two approaches as *horizontal training*, as the computation tasks are executed over multiple workers at the same system level (either the computing units in the cloud center, or the edge devices in the fully decentralized peer-to-peer network).

There are also *hierarchical training* approaches to efficient training of DNNs. JointDNN is proposed in [9], which trains some layers of a DNN on an edge device and the other layers on the cloud center. However, the latency between the edge device and the cloud center is still the major factor to limit the training speed. The emerging edge computing paradigm provides another option, in which the edge servers are in between of the edge devices and the cloud center, and can fulfill computation tasks as close as possible to the data sources. Comparing to the communication latency between the cloud center and the edge devices, the latency between the edge servers and the edge devices is much lower. These excellent properties motivate the emerging edge learning scheme of jointly training a DNN with edge devices and an edge server [10]. Edge learning focuses on training a DNN model at the network edge near the data sources. The common paradigm of edge learning in the literature (e.g., [11], [12]) is built upon the idea of federated learning (FL) such that each edge device trains a model based on local data, and then these model updates are aggregated at an edge server. The primary objective is to achieve privacy-preserving knowledge sharing among the devices via a joint model learning process. Different from this, this paper considers the fast learning with respect to a specific edge device and leverages a multitude of mobile-edge-cloud resources for training acceleration.

Fig. 1 illustrates the differences between the *horizontal training* and *hierarchical training* paradigms. Observing that the works in [9] and [10] only consider two levels in the mobile-edge-cloud hierarchical architecture – the device and cloud levels in [9] and the mobile and edge levels in [10], the drawback of them is that they did not fully utilize the communication and computation resources of all the three levels. As communication latency between mobile and edge levels is generally low and the computation resource at the cloud level is abundant, a holistic framework that fully exploits the communication and computation resources of all the three levels can definitely leash the great potentials of mobile-edge-cloud computing for accelerating edge AI learning.

Motivated by this, we propose a hierarchical training framework, abbreviated as HierTrain, which efficiently deploys the DNN training tasks over the mobile-edge-cloud levels and achieves minimum training time for fast edge AI learning. In this paper, our contributions are summarized as follows.

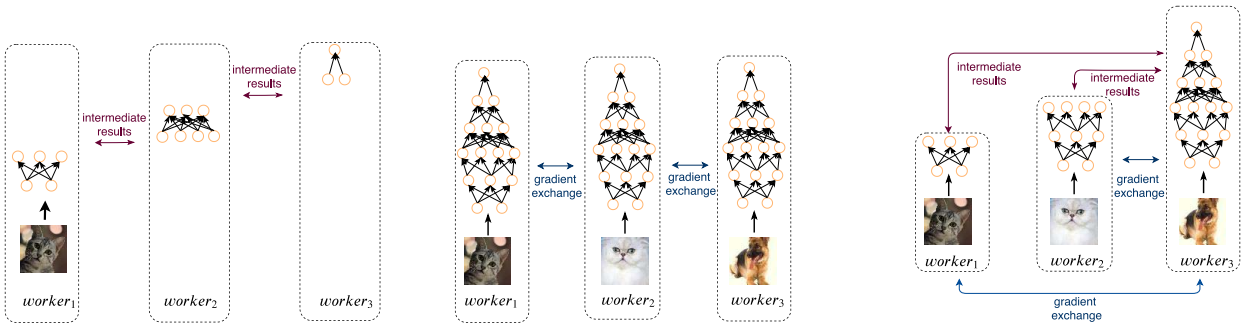
1) We develop a novel *hybrid parallelism* method, which is the key to HierTrain, to adaptively assign the DNN model layers and the data samples to the three levels by taking into account the communication and computation resource heterogeneity therein.
2) We formulate the problem of scheduling the DNN training tasks at both layer-granularity and sample-granularity. Solving this minimization problem enables us to achieve the minimum training time.
3) We implement and deploy a hardware prototype over an edge device, an edge server and a cloud server, and extensive experimental results demonstrate that HierTrain achieves superior performance, e.g., achieving up to 6.9× speedup compared to the cloud-based hierarchical training approach.

We should emphasize that, different from many existing works focusing on edge AI inference [13], in this study we promote HireTrain for addressing the important issue of edge AI training acceleration. This is due to the emerging demand that many edge AI applications (e.g., smart robots and industrial IoT) require both real-time performance and continuous learning capability of fast model updating with fresh sensing/input data samples and being adaptive to complex dynamic application environments. On the other hand, HierTrain is along the emerging line of promoting in-network model training such as edge learning for intelligent B5G networking [14] for mitigating the significant overhead and latency of transferring the data of massive size to the cloud for remote model training.

It should be noted that our framework can be directly applied to a DNN which can be represented as an ordered sequence of layers, such as VGG [15], YOLO [16], MobileNets [17], etc. Generally speaking, the key idea of hybrid parallelism of HierTrain can be also useful for RNN. However, due to the complicated structure of RNN, splitting the RNN learning task over multiple workers is much more challenging. We will consider extending our approach to support RNN in the future work.

II. BACKGROUND & MOTIVATION

In general, there are three computing workers/nodes for DNN training in the mobile-edge-cloud hierarchical system: edge device, edge server and cloud center, which have diverse



(a) Model Parallelism (b) Data Parallelism (c) Hybrid Parallelism

FIGURE 2. **Illustration of the three parallelism methods. Each row of circles represents a layer in the trained DNN.**

communication and computation capacities. To jointly train a DNN, we need to determine how to split the training data samples and the trained DNN across the three workers. Below, we introduce two traditional methods, *model parallelism* and *data parallelism*, as well as our proposed *hybrid parallelism* method. The three parallelism methods are illustrated in Fig. 2.

1) *Model Parallelism:* Because a DNN is typically stacked by a sequence of distinct layers, it is natural to assign the layers to the workers; see Fig. 2(a). In the *model parallelism* method, each worker holds multiple layers and is in charge of updating the corresponding model parameters. Therefore, when training the DNN with the back-propagation rule in the stochastic gradient descent (SGD) algorithm [18], the workers need to communicate to exchange the intermediate results. The works of JointDNN [9] and JALAD [19] demonstrate the effectiveness of the *model parallelism* method. However, since the layers of the DNN are trained sequentially, when one worker is computing the others must stay idle. Thus, the computation resources are not fully utilized in the *model parallelism* method.

2) *Data Parallelism:* The *data parallelism* method splits the data samples to the workers, trains one local copy of DNN in every worker, and forces the local DNNs to reach a consensus along the optimization process. To implement SGD, the workers need to exchange either the local stochastic gradients or the local model parameters from time to time, as depicted in Fig. 2(b). The works of [20] and [21] show that the *data parallelism* method is able to accelerate the DNN training when the data are collected and split to multiple computing units within the cloud center. Nevertheless, the requirement of transmitting the local stochastic gradients or the local model parameters, whose dimensions are the same, leads to heavy communication overhead when the size of DNN is large. Therefore, the *data parallelism* method is not communication-efficient in the mobile-edge-cloud architecture.

3) *Hybrid Parallelism:* Observe that the backend layers in most DNNs, such as convolutional neural networks (CNNs), are fully connected layers and contain the majority of parameters. This fact motivates us to improve the *model parallelism* method through letting all the backend layers be trained by one worker while the frontend layers be trained by multiple workers. Therefore, the workers just need to exchange a small fraction of the local stochastic gradients or the local model parameters to train the frontend layers, as well as transmit the intermediate results to train the backend layers, thus the communication latency between workers is largely reduced. As shown in Fig. 2(c), the backend layers are only trained by $worker_3$. Some frontend layers are trained by $worker_2$ and $worker_3$, while some are jointly trained by all the workers. Meanwhile, similar to the data parallelism method, training data samples are split and assigned to all the workers according to their computing resource heterogeneity, to further balance the workloads across the device, edge and cloud.

In order to apply the *hybrid parallelism* method to accelerate the training of DNNs over the mobile-edge-cloud architecture, we need to optimize the assignments of the DNN layers and the data samples to the three workers. To this end, we propose HierTrain, a hierarchical training framework, as follows.

III. HIERTRAIN FRAMEWORK

In this section, we present the HierTrain framework, which jointly selects the best partition points of the given DNN model and determines the appropriate number of data samples delegated to different workers in a mobile-edge-cloud hierarchy. Fig. 3 presents the system overview of the HierTrain framework, which consists of three stages: profiling, optimization, and hierarchical training.

At the **profiling stage**, HierTrain performs two initialization steps: (i) profiling the average execution time of different model layers in the device, edge, and cloud workers, respectively; (ii) profiling the size of output for each layer in the model. Specifically, we perform one training iteration on each computing node of the mobile-edge-cloud, and then record the execution times and output sizes of different DNN layers. We repeat this process dozens of times and then take the averages to get stable mean values. It should be noted that the output size of each layer in the model is fixed and just

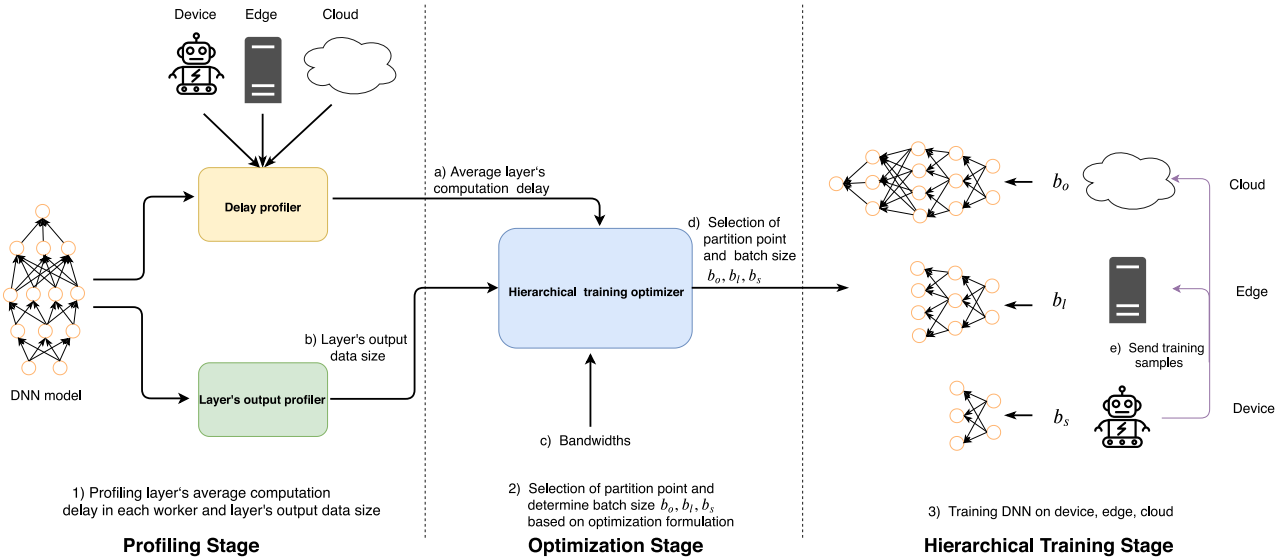


FIGURE 3. System overview.

needs to record only once. Note that since many DNN models have fixed and known structures, we can conduct such profiling steps beforehand in an offline manner to reduce the time overhead. In some challenging scenarios with system dynamics, we can leverage more sophisticated approaches for profiling such as regression-based modeling [22] and machine learning-based prediction [23] by utilizing the collected measurement data together with the dynamic impact factors (e.g., varying computing resources).

At the **optimization stage**, the hierarchical training optimizer selects the best DNN model partition points and determines the number of training samples for the workers of edge device, edge server, and cloud center, respectively. This scheduling policy is generated by the optimization algorithm introduced in Section V. The optimization algorithm minimizes the DNN training time with respect to five decision variables m_s, m_l, b_o, b_s, b_l (m_s, m_l represent partition points, b_o, b_s, b_l represent the number of samples processed on each worker, which will be defined in Section IV). It depends on the following inputs: (i) the profiled average execution time of different model layers in the three workers; (ii) the profiled size of output for each layer in the model; (iii) the available bandwidth between the edge device and the edge server, and that between the edge server and the cloud center.

At the **hierarchical training stage**, the edge device first sends the delegated data samples to the edge server and the cloud center according to the scheduling policy given in the optimization stage. Once having the needed data samples at hand, the edge device, the edge server and the cloud center start their scheduled training tasks (i.e., the assigned model training modules) immediately, and perform collaborative model training in a hierarchical manner.

Note that the hierarchical training stage depicted in Fig. 3 only shows one possible scheduling policy, in which the cloud center trains the full model while the edge server and the edge device only train parts of the model. This scheduling policy is suitable for the scenario that the bandwidth between edge device and cloud center is in a good condition. However, when the network bandwidth becomes the bottleneck, the scheduling policy may choose the edge server or the edge device to train the full model. In the next section, we will elaborate on how the data samples and the model layers are partitioned.

IV. PROBLEM STATEMENT OF HIERTRAIN TASK SCHEDULING

A. TRAINING TASKS IN HIERTRAIN

We consider that a DNN is stacked by a sequence of distinct layers, and the output of one layer feeds into the input of the next layer. Our goal is to reduce the overall training time in the mobile-edge-cloud environment. Towards this end, we first define three types of training tasks, depicted in Fig. 4 and explained as follows.

`TASK O (Original Task)`: Training the full DNN with b_o data samples.

`TASK S (Short Task)`: Training m_s consecutive layers from layer 1 to layer m_s with b_s data samples.

`TASK L (Long Task)`: Training m_l consecutive layers from layer 1 to layer m_l with b_l data samples.

Here m_s and m_l are positive integers, and we assume $m_s \leq m_l \leq N$ (N is the total number of layers in the DNN model).

The key motivations of defining the three task types above are as follows. On one hand, only `TASK O` contains the most backend layers (e.g., fully connected layers in many DNNs) that typically have the majority of the parameters, and this helps to reduce the communication overheads for parameter exchange across different tasks. On the other hand, `TASK O`, `L` and `S` all contain the frontend layers (e.g., convolution

TABLE 1. List of notations.

Parameter	Description
$L^f_{j,i}$	forward time to handle 1 sample for layer i on $worker_j$
$L^b_{j,i}$	backward time to handle 1 sample for layer i on $worker_j$
$L^u_{j,i}$	weight update time for layer i on $worker_j$
MP_i	number of parameters in layer i
MO_i	output size of layer i to handle 1 sample in forward phase

layers in many DNNs) that are often computationally intensive, and this also helps to exploit the computing resources of different workers in parallel to accelerate the DNN training. Furthermore, we have the flexibility to optimize the computing workloads of different tasks by varying their input data sample sizes.

In the following, we denote the workers that execute `TASK O`, `TASK S` and `TASK L` as $worker_o$, $worker_s$ and $worker_l$, respectively. We also denote the profiling values $L^f_{j,i}$, $L^b_{j,i}$, $L^u_{j,i}$ and MP_i, MO_i. Their meanings are shown in Table 1.

By defining the three task structures, we have rich flexibility in optimizing the training workloads across the edge device, the edge server and the cloud center by tuning the sizes of their data samples and assigned model layers, tailored to their computation resources and network conditions.

B. TRAINING PROCEDURE IN HIERTRAIN

Based on the above-defined three tasks, we elaborate on the training procedure in HierTrain as follows. First, the scheduling policy determines how to assign the model layers and the data samples to the three workers. Second, the edge device initiates the training procedure and sends the partitioned data samples to the edge server and the cloud center. Last, the following three phases are executed iteratively.

1) FORWARD

$worker_s$ executes the forward phase (i.e., inference through the DNN model to obtain the current model loss) over the assigned layers, using a mini-batch b_s of data samples. Once completing the forward phase over the assigned layers, $worker_s$ sends the output to $worker_o$. Then, $worker_o$ proceeds to execute the forward phase over the rest of the layers. $worker_l$ acts the same as $worker_s$, using a mini-batch b_l of data samples. $worker_o$ also executes the forward phase, but it is over all the layers and using a mini-batch b_o of data samples. When the forward phase ends, $worker_o$ collects the model losses from $B = b_s + b_l + b_o$ data samples.

2) BACKWARD

For each data sample, $worker_o$ starts the backward phase (i.e., back-propagation using the loss to obtain the stochastic gradients) from the last layer of the DNN. If the data sample belongs to $worker_o$, then $worker_o$ executes the full backward phase. If the data sample belongs to $worker_l$, then $worker_o$ sends the intermediate results to $worker_l$ upon reaching layer $m_l + 1$, and $worker_l$ proceeds to execute the backward phase over the rest of the layers. The same rule applies to $worker_s$, except that $worker_o$ sends the intermediate results to $worker_s$ upon reaching layer $m_s + 1$. When the backward phase ends, every worker obtains the stochastic gradients of the assigned layers.

3) WEIGHT UPDATE

$worker_l$ and $worker_s$ send the computed stochastic gradients to $worker_o$. Then $worker_o$ averages the stochastic gradients layer-wise, and sends the averaged stochastic gradients to $worker_l$ and $worker_s$ according to the layers assigned to them. With these stochastic gradients, the three workers update the weights of their assigned layers independently.

C. FORMULATION OF MINIMIZING TRAINING TIME

The core of HierTrain is a scheduling policy that determines how the model layers and the data samples are assigned to the three workers. The goal is to minimize the training time, which is determined by the computation and communication latencies. To analyze these two quantities, we assume that the DNN has N layers and the size of each data sample is Q bits.

1) COMPUTATION LATENCY

Recall that the DNN training procedure is divided into three phases: forward, backward and weight update. In the forward and backward phases, the amount of computation is proportional to the number of processed data samples [24]. We denote $T_{j,i,b,forward}$ and $T_{j,i,b,backward}$ as the computation latencies of executing layer i on $worker_j$ with b input data samples in the forward and backward phases, respectively. Here $j \in \{o, s, l\}$, $i \in \{1, 2, \ldots, N\}$, and $b \in \{b_o, b_o + b_l, b_o + b_l + b_s\}$. Then we have

$$T_{j,i,b,forward} = bL^f_{j,i}, \tag{1}$$

$$T_{j,i,b,backward} = bL^b_{j,i}. \tag{2}$$

The computation latency $T_{j,update}$ of the weight update phase on a worker $j \in \{o, s, l\}$ is the summation of the execution time over the involved layers, given by

$$T_{j,update} = \sum_{i=1}^{m_j} L^u_{j,i}. \tag{3}$$

2) COMMUNICATION LATENCY

The workers are bidirectionally connected with each other. For example, the edge device and the edge server are connected with the high-speed wireless local-area-network (WLAN) link, while the edge server and the cloud center are connected with the bandwidth-limited wide-area-network (WAN) link. Let $B_{o,s}$ denote the bandwidth between $worker_o$ and $worker_s$, $B_{o,l}$ the bandwidth between $worker_o$ and $worker_l$, $B_{s,l}$ the bandwidth between $worker_s$ and $worker_l$. The communication latency is the ratio of the transferred data size and the bandwidth between two workers, as

$$T_{communication} = \frac{DataSize}{Bandwidth}. \tag{4}$$

3) TRAINING TIME

As depicted in Fig. 4, $worker_o$, $worker_s$, $worker_l$ use b_o, b_s, b_l data samples as inputs, respectively. Layer 1 to layer m_s are executed in parallel over the three workers, layer m_s+1 to layer m_l are executed in parallel over $worker_o$ and $worker_l$, and the rest of layers are executed on $worker_o$. Below we calculate the training time, beginning with those in the forward and backward phases.

Denote $T^1_{forward}$ and $T^1_{backward}$ as the latencies of executing layers between 1 and m_s over the three workers in the forward and backward phases, respectively, given by

$$T^1_{forward} = \max\Bigg\{T_{o,input} + \sum_{i=1}^{m_s} T_{o,i,b_o,forward}, T_{s,input} + \sum_{i=1}^{m_s} T_{s,i,b_s,forward} + T_{s,output}, T_{l,input} + \sum_{i=1}^{m_s} T_{l,i,b_l,forward}\Bigg\}, \tag{5}$$

$$T^1_{backward} = \max\Bigg\{\sum_{i=1}^{m_s} T_{o,i,b_o,backward}, \sum_{i=1}^{m_s} T_{s,i,b_s,backward} + T_{s,grad}, \sum_{i=1}^{m_s} T_{l,i,b_l,backward}\Bigg\}. \tag{6}$$

Here $T_{j,input}$ is the communication latency of $worker_j$ to receive b_j data samples, $j \in \{o,s,l\}$. We use (4) to calculate $T_{j,input}$, where $DataSize = b_j \times Q$ and $Bandwidth$ is the bandwidth between the edge device and $worker_j$. $T_{s,output}$ represents the communication latency of $worker_s$ to transmit its forward output to $worker_o$. Recall that MO_{m_s} is the output size of layer m_s in the forward phase for one data sample, b_s is the number of data samples of $worker_s$, and $B_{o,s}$ is the bandwidth between $worker_o$ and $worker_s$. Then according to (4), $T_{s,output} = \frac{b_s \times MO_{m_s}}{B_{o,s}}$. $T_{s,grad}$ represents the communication latency of $worker_o$ to send the intermediate results to $worker_s$ in the backward phase. The size of the intermediate results is equal to the output data of layer m_s in forward phase. Thus, $T_{s,grad} = T_{s,output}$.

Denote $T^2_{forward}$ and $T^2_{backward}$ as the latencies of executing layers between m_s+1 and m_l over $worker_o$ and $worker_l$ in the forward and backward phases, respectively, given by

$$T^2_{forward} = \max\Bigg\{\sum_{i=m_s+1}^{m_l} T_{o,i,b_o+b_s,forward}, \sum_{i=m_s+1}^{m_l} T_{l,i,b_l,forward} + T_{l,output}\Bigg\}, \tag{7}$$

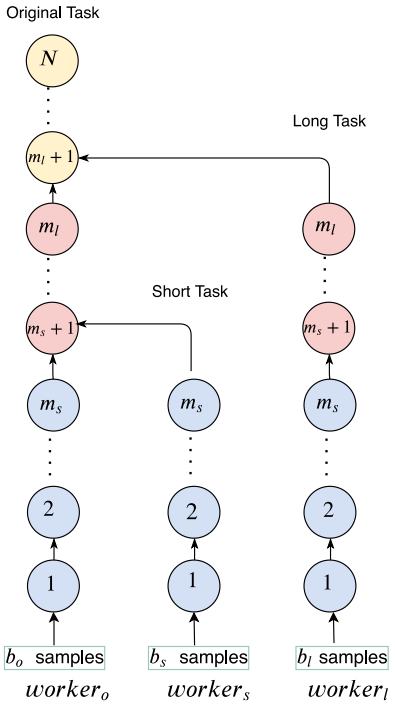


FIGURE 4. $worker_o$, $worker_s$, $worker_l$ use b_o, b_s, b_l data samples as inputs, respectively. Layer 1 to layer m_s are executed in parallel over the three workers, layer m_s+1 to layer m_l are executed in parallel over $worker_o$ and $worker_l$, and the rest of layers are executed on $worker_o$.

$$T^2_{backward} = \max\Bigg\{\sum_{i=m_s+1}^{m_l} T_{o,i,b_o+b_s,backward}, \sum_{i=m_s+1}^{m_l} T_{l,i,b_l,backward} + T_{l,grad}\Bigg\}. \tag{8}$$

Here $T_{l,output}$ is the communication latency of $worker_l$ to transmit its forward output to $worker_o$, given by $T_{l,output} = \frac{b_l \times MO_{m_l}}{B_{o,l}}$. $T_{l,grad}$ represents the communication latency of $worker_o$ to send the intermediate results to $worker_l$ in the backward phase, it is equal to $T_{l,output}$.

Denote $T^3_{forward}$ and $T^3_{backward}$ as the latencies of executing layers between m_l+1 and N over $worker_o$ in the forward and backward phases, respectively, given by

$$T^3_{forward} = \sum_{i=m_l+1}^{N} T_{o,i,b_o+b_s+b_l,forward}, \tag{9}$$

$$T^3_{backward} = \sum_{i=m_l+1}^{N} T_{o,i,b_o+b_s+b_l,backward}. \tag{10}$$

Now we consider the training time in the weight update phase. After the backward phase finishes, $worker_l$ and $worker_s$ send the stochastic gradients to $worker_o$. Then $worker_o$ sends the averaged stochastic gradients to $worker_l$ and $worker_s$ according to the layers assigned to them, and the three workers update the weights of their assigned layers. The total time cost in the weight update phase is denoted

as T_{update}, given by

$$T_{update} = \max\{T_{o,update}, T_{s,update}, T_{l,update}\} + \max\{T_{s,weightgrad}, T_{l,weightgrad}\}. \quad (11)$$

Here $T_{j,update}$ is the computation latency of the weight update phase on $worker_j$, $j \in \{o, s, l\}$, as defined in (3). $T_{s,weightgrad}$ and $T_{l,weightgrad}$ represent the communication latencies of $worker_s$ and $worker_l$ to send the stochastic gradients to and receive the updated weights from $worker_o$, respectively. For layer i, the sizes of the stochastic gradients and the updated weights are both MP_i. Therefore, we have $T_{s,weightgrad} = \frac{2}{B_{o,s}} \sum_{i=1}^{m_s} MP_i$ and $T_{l,weightgrad} = \frac{2}{B_{o,l}} \sum_{i=1}^{m_l} MP_i$.

4) MINIMIZATION OF TRAINING TIME

Therefore, the time of training the DNN for one iteration, including both computation and computation, is given by

$$T_{total} = \sum_{k=1}^{3} \left(T^k_{forward} + T^k_{backward}\right) + T_{update}, \quad (12)$$

in which the number of used data samples is

$$B = b_o + b_s + b_l. \quad (13)$$

Here B is the predefined batch size, while b_o, b_s and b_l are decision variables.

The number of layers m_s and m_l for `TASK S` and `TASK L` are also decision variables. It is possible in some scenarios that m_s or m_l can equal to 0, meaning that $worker_s$ or $worker_l$ will not participate in the DNN training procedure. For these scenarios, we do not assign any data samples to $worker_s$ or $worker_l$, such that $b_s = 0$ or $b_l = 0$. To characterize these connections, we introduce constraints

$$0 \le b_s \le m_s B, \quad (14)$$

$$0 \le b_l \le m_l B. \quad (15)$$

When $m_s = 0$ or $m_l = 0$, (14) or (15) ensures that $b_s = 0$ or $b_l = 0$. Otherwise, if m_s or m_l is any positive integer, (14) or (15) automatically satisfies due to (13).

In summary, when $worker_s$, $worker_l$ and $worker_o$ have been fixed, to minimize the training time, HierTrain solves the following optimization problem

$$\mathcal{P}_1: \underset{\{b_o, b_s, b_l, m_s, m_l\}}{minimize} \quad T_{total}, \quad (16)$$

$$s.t. \quad b_o + b_s + b_l = B, \quad (17)$$

$$0 \le b_s \le m_s B, \quad (18)$$

$$0 \le b_l \le m_l B, \quad (19)$$

where the decision variables b_o, b_s, b_l, m_s, m_l are all nonnegative integers. Since there are 6 possible mappings between $worker_s$, $worker_l$, $worker_o$ and the edge device, the edge server, the cloud center, we can enumerate all the mappings, calculate the optimal scheduling policy $\{b_o, b_s, b_l, m_s, m_l\}$ for each mapping, and then find the global optimal scheduling policy. The next section gives details of the proposed algorithm.

Algorithm 1 HierTrain Algorithm

Input:
1) $L^f_{k,i}, L^b_{k,i}, L^u_{k,i}, k \in \{d, e, c\}$: profiling values of device, edge, cloud
2) BW_{de}, BW_{ec}: bandwidth of device-edge and edge-cloud
3) MP_i: layer i parameters data size
4) MO_i: layer i output data size

Output: optimal solution $\{m^*_s, m^*_l, b^*_o, b^*_s, b^*_l\}$

Initialization: $T_{total,minimum}$ = MAX ▷ MAX is an infinite number
for map {Device, Edge, Cloud} to $\{worker_o, worker_s, worker_l\}$ **do**
  for $m_s = 0 \to N$ **do**
    for $m_l = m_s \to N$ **do**
      Solve the relaxed problem of $\mathcal{P}_1$ with m_s and m_l to get $\{b_o, b_s, b_l\}$
      $\{b_o, b_s, b_l\} \leftarrow Round(b_o, b_s, b_l)$ ▷ rounding b_o, b_s, b_l to integers
      Calculate T_{total} according to (12)
      if $T_{total} < T_{total,minimum}$ **then**
        $\{m^*_s, m^*_l, b^*_o, b^*_s, b^*_l\} = \{m_s, m_l, b_o, b_s, b_l\}$
        $T_{total,minimum} = T_{total}$
      end if
    end for
  end for
end for
Return: $\{m^*_s, m^*_l, b^*_o, b^*_s, b^*_l\}$

V. OPTIMIZATION OF SCHEDULING POLICY

Note that even when $worker_s$, $worker_l$ and $worker_o$ have been fixed, solving $\mathcal{P}_1$ is still challenging because: (i) in the objective T_{total}, the terms of T_{update}, $T^k_{forward}$ and $T^k_{backward}$, where $k = 1, 2, 3$, all contain summations with the numbers of summands determined by m_s and m_l; (ii) the decision variables b_o, b_s, b_l, m_s, m_l are all integers.

To address the first challenge, we observe that when m_s and m_l are fixed, $\mathcal{P}_1$ will become a standard integer linear programming (ILP) problem and is relatively easier to solve. Motivated by this observation, we enumerate the values of m_s and m_l, solve the resulting ILP problems, and then find the best one among the ILP solutions. This enumeration is feasible because the numbers of layers m_s and m_l are often modest in practice (such as AlexNet: 8 layers, VGG-16: 16 layers, GoogleNet: 22 layers, MobileNet: 28 layers).

To address the second challenge, for each ILP problem, we relax the integer variables to real ones, solve the relaxed linear programming (LP) problem, and then round the solution to integers. To be specific, the relaxed LP problem can be efficiently solved with CPLEX, Gurobi or CVXPY. Although these optimization solvers can solve ILP problem directly, we choose to convert the ILP problem to LP problem the reason is that these solvers solve LP problem are much faster than solve ILP problem. Further, the **rounding operation** works as follows. Given a real solution (b_o, b_s, b_l) of the relaxed LP problem, we divide them into integer parts $int(b_j)$ and fraction parts $frac(b_j)$, $j \in \{o, s, l\}$, and then sort the fraction parts in a descending order. For b_j with the largest fraction part, we let $b^*_j = int(b_j) + 1$, while for the other

TABLE 2. Algorithm running time.

LeNet	AlexNet	VGG-16	VGG-19	googLeNet	ResNet-34
0.52s	1.48s	3s	4s	5.3s	12s

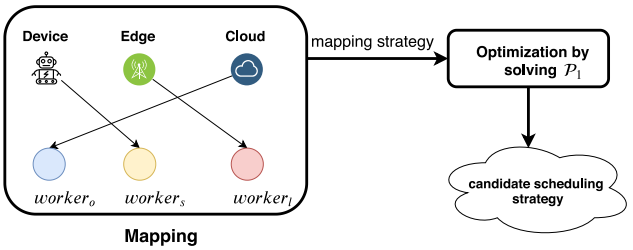


FIGURE 5. Each mapping strategy corresponds to a candidate optimal scheduling policy.

two b_j, $b_j^* = int(b_j)$. If $b_o^* + b_s^* + b_l^* = B$ is satisfied, then the rounding operation ends. Otherwise, for the two b_j with the largest fraction parts, we let $b_j^* = int(b_j) + 1$, while for the other b_j, $b_j^* = int(b_j)$. The constraint $b_o + b_s + b_l = B$ can be satisfied after at most two steps.

So far, we have solved $\mathcal{P}_1$ given that $worker_s$, $worker_l$ and $worker_o$ have been fixed. In order to deploy the DNN training task over the mobile-edge-cloud environment, we still need to find the best mapping strategy between the device, edge and cloud workers and $worker_o$, $worker_s$ and $worker_l$. As illustrated in Fig. 5, since the overall number of mappings is only 6, we can enumerate all the mapping, find a candidate optimal scheduling policy for each mapping, and then choose the best mapping strategy with the minimum training time. The algorithm is outlined in Algorithm 1.

In our algorithm, there are 6 possible mappings. For each mapping, the number of enumerations for m_s and m_l are $\frac{N^2+3N+2}{2}$. This is because the feasible values of m_s are from 0 to N, and we need to enumerate the values of m_l from m_s to N for each m_s. Therefore, we could get that the total number of enumerations in our algorithm are $3(N^2 + 3N + 2)$. Note that for the case of HierTrain, the number of model layers are typically small. Hence, enumerating strategy is an amenable solution for our problem. For other large-scale cases, we may consider to utilize efficient heuristic optimization approaches such as simulated annealing and evolutionary optimization which can be efficient for large-scale combinatorial optimization problems.

As shown in the Table 2, in order to verify the efficiency of our algorithm, we list the algorithm running time based on some common deep neural networks configuration. All results are obtained on a desktop computer equipped with an Intel Core i7-6700 3.4 GHz with 8 GB RAM running Linux. We use python as programming language and CPLEX as optimization problem solver. From Table 2, we see that the proposed algorithm runs very fast, and in practice its running time can be ignored compared with the long DNN training time.

VI. EVALUATION

A. DATASET AND MODELS

We evaluate HierTrain by training two well-known CNNs for image classification tasks. The first CNN is LeNet-5 [25], and we train it with the CIFAR-10 dataset [26]. CIFAR-10 contains 50,000 training images and 10,000 testing images, each of which has 10 labels. The second CNN is AlexNet [27], which is more complicated than LeNet-5. We train AlexNet on the tiny ImageNet dataset. The tiny ImageNet dataset has 200 classes, while each class has 500 training images, 50 validation images, and 50 testing images.

B. EXPERIMENTAL SETUP

We use a Raspberry Pi 3 tiny computer to act as an edge device. The Raspberry Pi 3 has a quad-core ARM processor at 1.2 GHz with 1 GB of RAM. We use an Intel NUC, a small but powerful mini PC which is equipped with a four Intel Cores i3-7100U with 8 GB of RAM, to emulate the edge server. Unless specifically indicated, we only use one core of the edge server in our experiments. This is to simulate the application scenarios where the edge server has to serve multiple edge devices and each edge device cannot occupy all the computation resource of the edge server. The cloud center is a Dell Precision T5820 Tower workstation with 16 Intel Xeon processor at 3.7 GHz and with 30 GB of RAM, and equipped with NVIDIA GPU GeForce GTX 1080 Ti. The computation capability of the cloud center is one order magnitude higher than those of the edge device and the edge server. All the three workers run the Ubuntu system, and we use Linux Traffic Control on them to emulate constrained network bandwidths.

There are many existing open-source platforms for training CNNs, such as TensorFlow [28], Theano [29], MXNet [30], PyTorch [31], and Chainer [32]. Among them we choose Chainer because it is flexible and able to leverage dynamic computation graphs, which facilitates the application of the proposed *hybrid parallelism* method.

C. BASELINES

To elucidate the performance of the proposed HierTrain framework, we consider the following baselines in the experimental evaluation.

1) ALL-EDGE

The edge device transmits all the training data samples to the edge server, and the edge server completes the DNN training.

2) ALL-CLOUD

The edge device transmits all the training data samples to the cloud center, and the cloud center completes the DNN training.

3) JOINTDNN [9]

The edge device and the cloud center jointly train the DNNs.

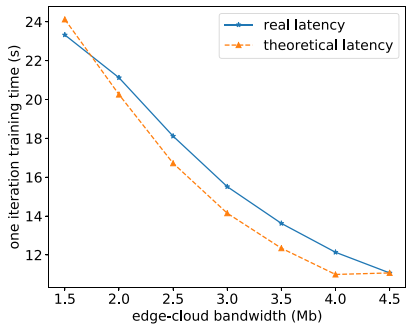


FIGURE 6. Comparison of real and theoretical latencies of training AlexNet.

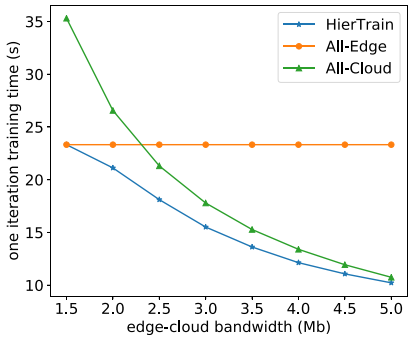


FIGURE 7. Per-iteration training time of AlexNet for HierTrain, All-Edge and All-Cloud under different bandwidths.

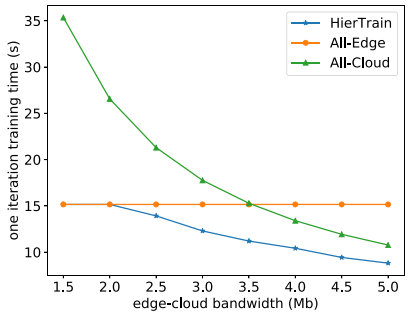


FIGURE 8. Per-iteration training time of LeNet-5 for HierTrain, All-Edge and All-Cloud under different bandwidths.

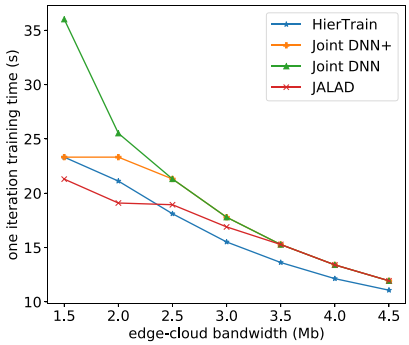


FIGURE 9. Per-iteration training time of AlexNet for HierTrain, JointDNN, JointDNN+, and JALAD under different bandwidths.

4) JOINTDNN+

We extend JointDNN to train the DNNs in the mobile-edge-cloud architecture. Following the design of JointDNN, the scheduling in JointDNN+ is by solving a shortest path problem over a graphic model.

5) JALAD [19]

The edge server and the cloud center jointly train the DNNs. A data compression strategy is applied to reduce the edge-cloud transmission latency. In our experiments we set the number of bits c used in data compression as 8.

D. RESULTS

1) MODEL VALIDITY

We first validate the formulated model that captures the execution delay of one iteration in training a DNN. Using the same scheduling policy, we obtain the real latency measured from the experiment and the theoretical latency, both in training AlexNet. As is shown in Fig. 6, the real and theoretical latencies highly match.

2) COMPARISON WITH ALL-EDGE AND ALL-CLOUD

Next we compare HeirTrain with the two baselines, All-Edge and All-Cloud, by fixing the mobile-edge bandwidth to 5 Mbps and varying the edge-cloud bandwidth from 1.5 Mbps to 5 Mbps. Fig. 7 shows the average per-iteration time to train AlexNet. The time cost of All-Cloud decreases as the edge-cloud bandwidth increases, while that of All-Edge remains unchanged. HeirTrain outperforms, and achieves up to 2.3× and 4.5× speedup comparing to All-Edge and All-Cloud, respectively. Similar observations can be found in training LeNet-5, as depicted in Fig. 8. HierTrain is the best among the three schemes, achieves up to 1.7× and 6.9× speedup comparing to All-Edge and All-Cloud, respectively.

3) COMPARISON WITH JOINTDNN, JOINTDNN+ AND JALAD

Now we conduct experiments to compare HierTrain with the three baselines: two state-of-the-art methods JointDNN and JALAD, as well as JointDNN+ that extends JointDNN to the mobile-edge-cloud architecture. The results on training AlexNet and LeNet-5 are demonstrated in Fig. 9 and Fig. 10, respectively. Observe that HierTrain outperforms both JointDNN and JointDNN+. Among these two baselines, JointDNN+ is better than JointDNN because it can utilize the edge server when the edge-cloud bandwidth is as low as 1.5 Mbps or 2 Mbps. When the edge-cloud bandwidth becomes larger, both JointDNN and JointDNN+ choose to run the training tasks in the cloud center.

Fig. 9 also compares HierTrain and JALAD in training AlexNet. When the edge-cloud bandwidth ranges from 1.5 Mbps to 2 Mbps, JALAD performs better than HierTrain. The reason is that the data compression strategy of JALAD can largely reduce the amount of transmitted data between the edge server and the cloud center. This makes JALAD

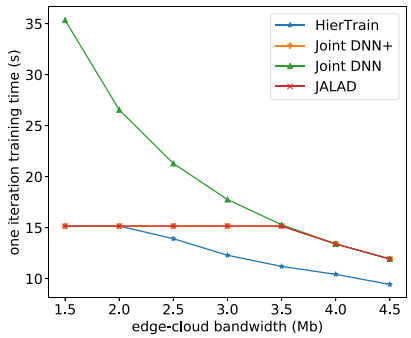


FIGURE 10. Per-iteration training time of LeNet-5 for HierTrain, JointDNN, JointDNN+, and JALAD under different bandwidths.

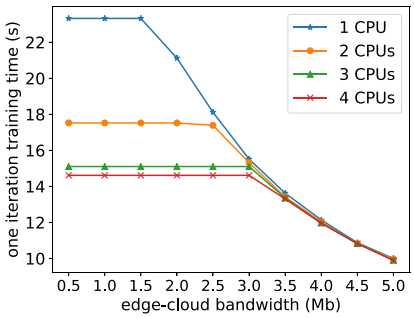


FIGURE 11. Effect of varying computation capability of edge server in HierTrain under different bandwidths.

advantageous in the low bandwidth condition as the communication time cost is the dominating factor in the overall delay. However, when the bandwidth increases, the benefit from reducing communication delay with data compression degrades sharply, and HierTrain outperforms JALAD. In Fig. 10 that shows the experimental results of training LeNet-5, the curves of JALAD and JointDNN+ overlap, because their scheduling policies are the same in the scenario – they are the same as the All-Edge strategy in the low bandwidth condition and the All-Cloud strategy in the high bandwidth condition.

4) EFFECT OF VARYING EDGE SERVER RESOURCES

Finally, we investigate the performance of HierTrain when the computation capability at the edge server changes. We consider training AlexNet, while keep the mobile-edge bandwidth as 5 Mbps and the edge-cloud bandwidth as 3.5 Mbps. We use docker to control the CPU cores used in the training process. As shown in Fig. 11, when the edge-cloud bandwidth is very low ($\leq$ 1.5 Mbps), improving the computation capability of the edge server can speedup the training process. This performance gain shrinks when the computation capability of the edge server keeps increasing. To be specific, varying from 1 CPU to 2 CPUs leads to large speedup, while varying from 3 CPUs to 4 CPUs yields insignificant speedup. When the edge-cloud bandwidth is sufficiently large ($\geq$ 3 Mbps), the computation capability of the edge server does not influence the overall performance. The reason for this phenomenon is that when the edge-cloud bandwidth is sufficiently large, the optimal policy is training on the cloud.

VII. RELATED WORK

Due to the attractive features of elasticity in computing power and flexible collaboration, hierarchically distributed computing structures naturally become a popular choice for executing DNN training or inference. Considering the deployment location for DNN, existing approaches can be divided into three classes.

A. CLOUD-BASED

Conventionally, most DNNs are usually deployed on the powerful cloud datacenters [33]. However, this means that a large amount of original data should be uploaded to the cloud, causing prohibitive communication overhead. In order to improve efficiency, Neurosurgeon [22] proposed a computation offloading idea in DNNs between the edge device and the cloud server at layer-granularity. Neurosurgeon explored one suitable partition point of DNN model and the execution starts with edge device and then switches to the cloud, which performs the rest of the computation. Reference [34] presented an optimal scheduling algorithm for collaboratively computation of feed-forward neural networks to achieve maximum performance and energy efficiency. JointDNN [9] provided optimization formulations at layer-granularity for forward and backward propagation in DNNs, which can get the optimal computation scheduling of processing some layers on the edge device and some layers on the cloud server. The limitation of cloud-based approach is the long WAN latency between device and cloud.

B. EDGE-BASED

An alternative is to deploy DNN at the edge of network. Li *et al.* [35] proposed a collaborative and on-demand DNN co-inference framework which could leverage hybrid computation resources of device and edge so as to achieve on-demand low-latency. Reference [36] exploited the virtual machine technique to let mobile users utilize nearby server called "cloudlet" to speed up service. Gabriel [37], [38] is a system that uses "cloudlet" for speech and face recognition applications. The focus of these works above is DNN inference at the edge. For the edge learning that considers the DNN training, many existing works target at the fast and cost-efficient FL (federated learning) scheme in order to train a commonly-shared model across multiple devices [39]. Along a different line, we consider the fast model learning with respect to a specific edge device and leverage a multitude of mobile-edge-cloud resources to training acceleration.

C. HIERARCHY-BASED

Alternatively, some studies focus on using both central cloud servers and edge servers for the execution. Reference [40] proposed a novel distributed DNN framework over distributed computing hierarchies (consisting of cloud, edge,

devices), which can allow low-latency classification via early exit. Li *et al.* [19] decoupled the DNN to execute on an edge and the cloud. They not only take into account latency measurement and raw data quantity between layers, but also take the compression of in-layer data into account. Huang *et al.* [41] proposed a DeePar framework which can exploit all the available resources from the device, the edge, and the cloud to improve the overall inference performance. Lin *et al.* [42] proposed a cost-driven offloading strategy based on a self-adaptive particle swarm optimization (PSO) algorithm using the genetic algorithm (GA) operators (PSO-GA) to minimize the system cost during offloading DNN layers over the cloud, edge, and devices.

Previous studies above mainly focus on distributed DNN inference. And they follow the scheme of partitioning DNNs into several parts then executing sequentially, which could not fully utilize the computation resources. In our work, we consider accelerating training DNNs in a hierarchical computing paradigm. To this end, we propose the training methodology *hybrid parallelism* which can dynamically adapt the number of parallel execution layers over computing nodes. In addition, different from previous studies we separate computation overhead not only on layer-granularity but also on sample-granularity.

VIII. CONCLUSION

In this paper, we study the problem of accelerating the training procedure of DNNs on the mobile-edge-cloud architecture. To this end, first, we present a novel *hybrid parallelism* method for training DNNs. Secondly, in order get scheduling policy of using *hybrid parallelism* method to train DNNs on the mobile-edge-cloud environment, we formulate the problem of computation scheduling of training DNNs at layer-granularity and sample-granularity as a minimization optimization programming problem, and solve it to get the scheduling policy. In addition, we test HierTrain in the real hardware and the results show that it could obviously outperform the naive policies such as all-edge and all-cloud, and also outperform exist prior works like JointDNN and JALAD.

For the future work, we are going to generalize the HierTrain framework to the application scenarios in multi-device and multi-edge environments, in which the federated learning across multi-devices and the device-to-edge association are interesting and challenging.

REFERENCES

[1] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 818–833.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018. [Online]. Available: arXiv:1810.04805.

[3] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for YouTube recommendations," in *Proc. 10th ACM Conf. Recommender Syst.*, 2016, pp. 191–198.

[4] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[5] M. Kumar, S. Sharma, A. Goel, and S. Singh, "A comprehensive survey for scheduling techniques in cloud computing," *J. Netw. Comput. Appl.*, vol. 143, pp. 1–33, Oct. 2019.

[6] H. Zhang *et al.*, "Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters," in *Proc. USENIX Annu. Techn. Conf. USENIX ATC)*, 2017, pp. 181–193.

[7] V. Mathur and K. Chahal, "Hydra: A peer to peer distributed training & data collection framework," 2018. [Online]. Available: arXiv:1811.09878.

[8] X. Chen, L. Pu, L. Gao, W. Wu, and D. Wu, "Exploiting massive D2D collaboration for energy-efficient mobile edge computing," *IEEE Wireless Commun.*, vol. 24, no. 4, pp. 64–71, Aug. 2017.

[9] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Trans. Mobile Comput.*, early access, Oct. 16, 2019, doi: 10.1109/TMC.2019.2947893.

[10] J. Ren, G. Yu, and G. Ding, "Accelerating dnn training in wireless federated edge learning system," 2019. [Online]. Available: arXiv:1905.09712.

[11] L. Valerio, A. Passarella, and M. Conti, "A communication efficient distributed learning framework for smart environments," *Pervasive Mobile Comput.*, vol. 41, pp. 46–68, Oct. 2017.

[12] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *Proc. IEEE Int. Conf. Commun.*, Shanghai, China, 2019, pp. 1–7.

[13] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 447–457, Jan. 2020.

[14] M. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," 2019. [Online]. Available: arXiv:1908.00080.

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: arXiv:1409.1556.

[16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 779–788.

[17] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017. [Online]. Available: arXiv:1704.04861.

[18] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. Comput. Stat. (COMPSTAT'2010)*, 2010, pp. 177–186.

[19] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "JALAD: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution," in *Proc. IEEE 24th Int. Conf. Parallel Distrib. Syst.*, Singapore, 2018, pp. 671–678.

[20] P. Goyal *et al.*, "Accurate, large minibatch SGD: Training ImageNet in 1 hour," 2017. [Online]. Available: arXiv:1706.02677.

[21] Y. You, I. Gitman, and B. Ginsburg, "Large batch training of convolutional networks," 2017. [Online]. Available: arXiv:1708.03888.

[22] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *ACM SIGARCH Comput. Architect. News*, vol. 45, no. 1, 2017, pp. 615–629.

[23] S. Mustafa, I. Elghandour, and M. A. Ismail, "A machine learning approach for predicting execution time of spark jobs," *Alexandria Eng. J.*, vol. 57, no. 4, pp. 3767–3778, 2018.

[24] A. Devarakonda, M. Naumov, and M. Garland, "AdaBatch: Adaptive batch sizes for training deep neural networks," 2017. [Online]. Available: arXiv:1712.02029.

[25] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[26] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep. TR-2009, 2009.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[28] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Conf. Oper. Syst. Design Implement. (OSDI16)*, 2016, pp. 265–283.

[29] J. Bergstra *et al.*, "Theano: A CPU and GPU math expression compiler," in *Proc. Python Sci. Comput. Conf.*, vol. 4. Austin, TX, USA, 2010, p. 3.
[30] T. Chen *et al.*, "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," 2015. [Online]. Available: arXiv:1512.01274.
[31] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.
[32] S. Tokui, K. Oono, S. Hido, and J. Clayton, "Chainer: A next-generation open source framework for deep learning," in *Proc. Workshop Mach. Learn. Syst. 29th Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, vol. 5, 2015, pp. 1–6.
[33] K. Skala, D. Davidovic, E. Afgan, I. Sovic, and Z. Sojat, "Scalable distributed computing hierarchy: Cloud, fog and dew computing," *Open J. Cloud Comput.*, vol. 2, no. 1, pp. 16–24, 2015.
[34] A. E. Eshratifar and M. Pedram, "Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment," in *Proc. Great Lakes Symp. VLSI*, 2018, pp. 111–116.
[35] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proc. Workshop Mobile Edge Commun.*, 2018, pp. 31–36.
[36] M. Satyanarayanan, V. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct.–Dec. 2009.
[37] K. Ha, P. Pillai, W. Richter, Y. Abe, and M. Satyanarayanan, "Just-in-time provisioning for cyber foraging," in *Proc. 11th Annu. Int. Conf. Mobile Syst. Appl. Services*, 2013, pp. 153–166.
[38] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proc. 12th Annu. Int. Conf. Mobile Syst. Appl. Services*, 2014, pp. 68–81.
[39] W. Y. B. Lim *et al.*, "Federated learning in mobile edge networks: A comprehensive survey," 2019. [Online]. Available: arXiv:1909.11875.
[40] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, Atlanta, GA, USA, 2017, pp. 328–339.
[41] Y. Huang, F. Wang, F. Wang, and J. Liu, "DeePar: A hybrid device-edge-cloud execution framework for mobile deep learning applications," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops*, Paris, France, 2019, pp. 892–897.
[42] B. Lin, Y. Huang, J. Zhang, J. Hu, X. Chen, and J. Li, "Cost-driven offloading for DNN-based applications over cloud, edge and end devices," 2019. [Online]. Available: arXiv:1907.13306.

DEYIN LIU received the B.S. degree in computer science from the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China, in 2017, where he is currently pursuing the master's degree. His research interests include mobile-edge computing, deep learning, and distributed computing.

XU CHEN received the Ph.D. degree in information engineering from the Chinese University of Hong Kong in 2012. He is a Full Professor with Sun Yat-sen University, Guangzhou, China, and the Vice Director of the National and Local Joint Engineering Laboratory of Digital Home Interactive Applications. He was a Postdoctoral Research Associate with Arizona State University, Tempe, AZ, USA, from 2012 to 2014, and a Humboldt Scholar Fellow with the Institute of Computer Science, University of Goettingen, Germany, from 2014 to 2016. He was a recipient of the Prestigious Humboldt Research Fellowship awarded by the Alexander von Humboldt Foundation of Germany, the 2014 Hong Kong Young Scientist Runner-Up Award, the 2016 Thousand Talents Plan Award for Young Professionals of China, the 2017 IEEE Communication Society Asia–Pacific Outstanding Young Researcher Award, the 2017 IEEE ComSoc Young Professional Best Paper Award, the Honorable Mention Award of 2010 IEEE International Conference on Intelligence and Security Informatics, the Best Paper Runner-Up Award of 2014 IEEE International Conference on Computer Communications, and the Best Paper Award of 2017 IEEE International Conference on Communications. He is currently an Area Editor of the IEEE OPEN JOURNAL OF THE COMMUNICATIONS SOCIETY and an Associate Editor of the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, the IEEE INTERNET OF THINGS JOURNAL, and the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS Series on Network Softwarization and Enablers.

ZHI ZHOU received the B.S., M.E., and Ph.D. degrees from the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China, in 2012, 2014, and 2017, respectively. He is currently a Research Associate Fellow with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. In 2016, he was a Visiting Scholar with the University of Gottingen. His research interests include edge computing, cloud computing, and distributed systems.

QING LING received the B.E. degree in automation and the Ph.D. degree in control theory and control engineering from the University of Science and Technology of China, Hefei, China, in 2001 and 2006, respectively. He was a Postdoctoral Research Fellow with the Department of Electrical and Computer Engineering, Michigan Technological University, Houghton, MI, USA, from 2006 to 2009, and an Associate Professor with the Department of Automation, University of Science and Technology of China from 2009 to 2017. He is currently a Professor with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. His current research interest includes decentralized network optimization and its applications. He received the 2017 IEEE Signal Processing Society Young Author Best Paper Award as a Supervisor. He is an Associate Editor of the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT and IEEE SIGNAL PROCESSING LETTERS.