Task Scheduling with Optimized Transmission Time in Collaborative Cloud-Edge Learning

Yutao Huang[1], Yifei Zhu[1], Xiaoyi Fan[1], Xiaoqiang Ma[2], Fangxin Wang[1], Jiangchuan Liu[1]
, Ziyi Wang[3], Yong Cui[3]
[1]School of Computing Science, Simon Fraser University, Canada
[2] School of Electronic Information and Communications, Huazhong University of Science and Technology, China
[3] Department of Computer Science and Technology, Tsinghua University, Beijing, China

***Abstract*—Deep learning has been applied in many recent advanced applications in the field of transportation, finance and medicine. These applications require significant computation resources and large-scale training samples. Cloud becomes a natural choice for conducting these learning tasks due to its abundant resources. However, deeper penetration of deep learning techniques in mission critical applications, like driverless car, calls for stricter time requirement to guarantee its interaction and larger amount of dataset for training to guarantee its accuracy, which cannot be easily satisfied by the cloud and makes the network transmission become the bottleneck. Edge learning emerges to be a promising direction to reduce data transmission time by processing and compressing the raw data at the edge of the network, while brings the concern of accuracy reduction at the meantime. To balance this tradeoff under cloud-edge architecture, we study a task scheduling problem for reducing weighted transmission time which takes learning accuracy into consideration. We also propose efficient scheduling algorithms which are able to achieve up to 50% reduction in makespan with extensive trace-driven simulations.**

I. Introduction

With the growing applications of learning based technology in image recognition, speech recognition, and other back-end systems like advertising, artificial intelligence (AI) has finally started to move from research labs to real business. Tech companies spent $20 billion to $30 billion on AI in 2016; external investment growth on AI has also increased 3 folds since 2013 [1]. This technology is expected to benefit a wide spectrum of industry sectors, covering finance, health, transportation and more. In particular, deep learning has demonstrated great success in achieving state-of-the-art performances on various tasks. This result is made possible by the substantial amount of data, the mature of corresponding learning techniques, and hardware with stronger computation power. Given the learning based applications are extremely data intensive, computation intensive, and hardware-dependent, cloud becomes a natural choice for learning task deployment. Major cloud providers like Amazon and Google, both provide their corresponding learning services [2] [3].

Deeper penetration of deep learning techniques into mission critical or personal contexts calls for stricter requirements in latency and privacy, which is hard to be easily satisfied by the cloud. To be specific, transferring large volume of dataset into cloud incurs a significant amount of communication costs to the network. Absorbing data from all sources into one location may also raise the privacy concern. In addition, the possible long distance from the cloud to the users may also introduce high data transmission time which is detrimental to those mission critical tasks. In fact, with the emergence of IoT applications, more devices participating in cloud learning, more pressure the network will take. These concerns in latency and privacy, all present new challenges to the current cloud-oriented learning paradigm[1]. Ideally, running these tasks locally on each end devices could greatly mitigate these concerns. However, current end devices mostly still lack the computation resource to execute complicated deep learning tasks. Some end devices may be capable of executing some simple machine learning models, like Support Vector Machine (SVM) [4]. Not surprisingly, this leads to apparent reduction in learning accuracy and usually will not be considered as an option [5].

The emerging concept of edge learning opens a new opportunity towards the delay-sensitive and cost-efficient learning, as a complement of cloud learning. Fig.1 presents the structure of edge learning. The core idea is to push applications, data, and computing tasks away from centralized cluster to the edge of a network. Through offloading a substantial amount of storage and computation to the edge servers that are close to the users, the traditional cloud computing paradigm will be extended to the network edge. This new generation of paradigm has shown promising advantages in reducing communication latency and traffic transferred to the cloud, comparing to conventional cloud computing [6]. Furthermore, running learning tasks locally also makes it easier for users to accept from privacy's perspective.

Different from scheduling common computation tasks, deep learning applications also desire high learning accuracy which requires large size of training data to support. The high volume of training data needs to be transmitted from end devices to the cloud and produces huge traffic in the core network, which pushes the pressure to the network bandwidth. Under edge learning structure, edge server will split the data or perform dimensionality reduction algorithms on data. Though it leads to the reduction in network traffic size and data transmission time, sacrifice in learning accuracy is inevitable at the meantime [7]. These targets need to be balanced properly without either compromising too much learning performance

[1]We refer to this cloud oriented paradigm as cloud learning in this paper.

978-1-5386-5156-8/18/$31.00 ©2018 IEEE

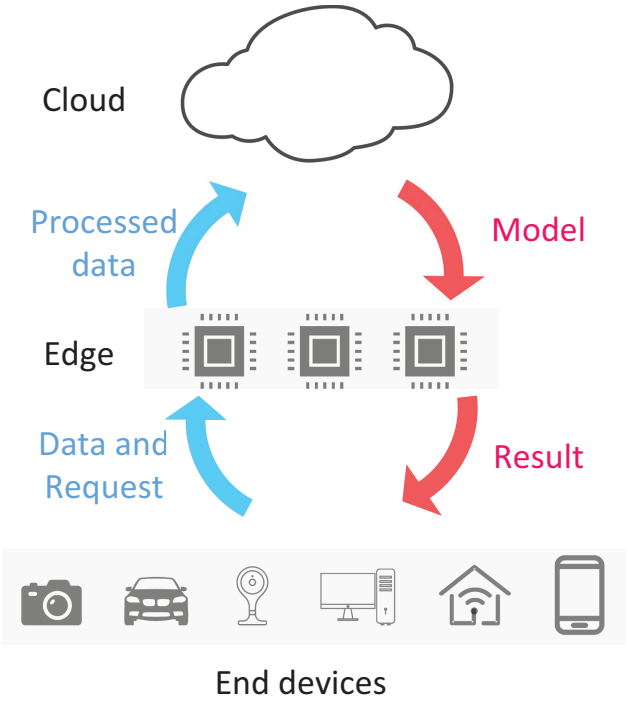


Fig. 1: The overview of edge learning

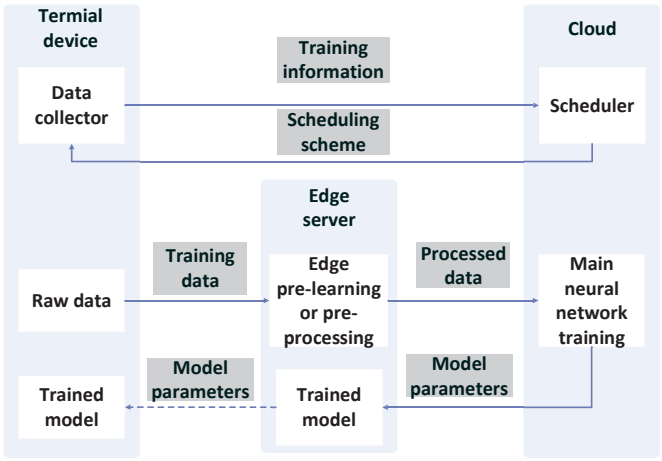


Fig. 2: Edge learning workflow

or wasting too much time for data transmission.

In this paper, we study a task scheduling problem for optimizing weighted data transmission time which takes learning accuracy into consideration. To be specific, we first formulate our scheduling problems as a mixed integer problem. Our problem is then transformed into an integer linear programming problem based on our discovery of the tradeoff between learning accuracy and traffic reduction. We present two algorithms which can solve our problem accordingly. With trace-driven simulations, our algorithms show great performance with reduction up to 50% in data transmission time.

The remainder of this paper is organized as follows. In Section II, we discuss the background for edge learning and describe the edge learning framework. In Section III we propose an offline edge learning task scheduling problem. We also model the tradeoff between learning accuracy and traffic reduction and use the tradeoff model to reduce the scheduling problem to an Integer Linear Programming problem. Section IV presents two algorithms for the job scheduling problem, and present their performance with simulation results in Section V. Section VI summarizes the related work and we conclude our work in Section VII.

II. Background and Edge Learning Framework

A. Deep Learning in the Network

Artificial neural network is one of the key techniques in the field of machine learning, which consists of three layer: input layer, hidden layer and output layer respectively. In particular, the hidden layers contain multiple perceptrons, and the information are fed forward from one layer to the next. Recent years, deep neural network (DNN) has been proposed with multiple hidden layers between the input and output layers. DNNs have achieved great success in numerous applications, such as image recognition, natural language processing, and speech recognition.

However, DNN training is a computation and data intensive task and the state-of-the-art CPU processors, e.g., Intel Xeon Processor E7-8890 v4 with 24 cores and 60M cache, cannot handle it efficiently. Recently, powerful and efficient GPU computing has attracted high interest in academia and industry for deep learning processing. With thousands of computational cores, GPUs have superior performance in running matrix multiplications which can be parallelized and are massively involved in deep learning applications. The strong computation power is typically offered by machine clusters, or more general, modern data centers. Thus, cloud-based deep learning has become a promising solution. For example, Amazon EC2 as a popular cloud platform, offers Elastic GPUs for deep learning computation which can train data at the level of terabyte and even petabyte scales.

B. Benefits and Challenges of Edge Learning

As a supplement to the concept of "cloud", "edge" is defined as the network topology at close proximity to end devices, which can be accessed by Radio Access Networks (RAN), WLAN, ethernet and other network connections. Edge servers are deployed at the network edge, which bring the benefits of low-latency connection between the edge server and the end device.

For deep learning applications, which seek for abundant computation resource and usually bring demands for huge amounts of data, edge learning can be a practical supplement of cloud learning. By pushing partial of learning process to the edge, edge learning has better performance comparing to traditional cloud learning. Recent research has proved

that edge learning can significantly reduce communication cost [4], latency and energy consumption [8] for inference phases. These advantages are critical especially for some deep learning inference applications. For example, face recognition in video streaming requires to recognize the face in just several hundreds of milliseconds, otherwise the face will probably not been catched in the video stream.

Different from inference, model training doesn't require millisecond level latency and usually takes up a long period of time. However, since the high training quality is based on large size of training data, it is impractical to send the whole raw data directly to the cloud due to the bandwidth limitation. Also, privacy concerns sometimes will restrict raw data being transferred in the Internet without special processing. Edge then becomes an intermediate structure to offload partial of training tasks and makes model training on the edge an optimal choice. Recent research showed that model training on the edge can perform near to the optimum with various of machine learning models [9].

Real-world deployment, however, brings challenges for current edge learning applications. Due to the limited computation resources on edge servers, cloud still plays an important role for edge learning applications since only part of the computation can be offloaded to the edge, while the other part still needs to be executed in the cloud data center [6]. In addition, the state-of-the-art deep learning application is designed for cloud-centric paradigm, which is not suitable for the end-edge-cloud architecture. An end user can switch to different edge network by changing the network connection to any of end devices' available access network. Thus, it's a challenge to arrange the connection for each end user for getting the best Quality of Service (QoS).

C. General Edge Learning Framework Overview

We now describe a general edge learning framework based on the end-edge-cloud architecture. Here, we focus on the model training. Suppose there are multiple deep learning users, and their raw data for training is stored in their own end devices. For each end device, there is one or several edge servers can be accessed. The data needs to be offloaded to one from the accessible edge servers for pre-learning or preprocessing before being uploaded to the cloud for main neural network training. Besides the edge learning process, our cloud-edge learning system will provide an appropriate scheduling scheme to arrange end device to an edge server, so that resource on edge servers will be efficiently utilized.

Our workflow is shown in Fig.2, and it mainly consists of two steps. The first step is the scheduling process. In this step, end devices send their information about training data size and chosen deep learning model to the cloud. Then cloud gathers all the information and runs the scheduler to generate a connection scheme and finally sends back the edge server connection information to each end device.

The second step is the deep learning process in our cloud-edge learning system. The training data is produced by end devices, e.g. mobile phones, intelligent cameras, and all kinds of IoT devices. The data then is sent to the arranged edge server for edge pre-learning or preprocessing. Here, the jobs for the edge are basically to extract the features, perform the dimensionality reduction algorithm, or even separate the whole deep neural network and move partial of the workload from the cloud to the near-end edge server. Introducing edge for cloud deep learning applications can significantly reduce the data traffic volume and the cloud's workload. However, at the same time, it will influence the training accuracy since raw data has been lossily compressed on the edge server. Finally, the processed data will be transferred to the cloud which is equipped with powerful and scalable GPU resources for the remaining training.

III. Scheduling for edge learning: System model and Problem Formulation

In this section, we will focus on the offline deep learning task scheduling problem and formulate it with mathematical expressions. By analyzing the relationship between learning accuracy and traffic reduction, we claim there exists a best space remaining ratio for each job in our formulated problem. Thus, we can further transform our problem formulation as an Integer Linear Programming (ILP) problem.

A. System Model

Suppose there are learning tasks denoted as $J = \{J_1, J_2, ..., J_n\}$, totally n tasks on different end devices. And there are edge servers denoted as $M = \{M_1, M_2, ..., M_m\}$ with total number of m. We define $D = \{D_1, D_2, ..., D_n\}$ as raw dataset size, and $R = \{R_1, R_2, ..., R_n\}$ as space remaining ratio which equals to the ratio of compressed dataset size to raw dataset size. Then for the task J_i, the data size transferred to the cloud equals to $D_i * R_i$. For each task J_i, there is a set $E_i = \{M_1, M_2, ..., M_{k_i}\}$ representing the set of edge servers which are available to connect. The available connection information for all tasks is denoted as $E = \{E_1, E_2, ..., E_n\}$.

Also we define $B = \{B_1, B_2, ..., B_m\}$ as the bandwidth constraints for each edge. Then we can calculate the data transmission time $P_{i,j}$ as

$$P_{i,j} = \frac{D_i * R_i}{B_j} \tag{1}$$

where i represents the index of tasks, and j represents the index of edge servers.

We define $A = \{A_1, A_2, ..., A_n\}$ as the relative learning accuracy for each learning task.

B. Problem Formulation

We declare $x_{i,j} = 1$ if we assign task J_i to edge server M_j; otherwise $x_{i,j} = 0$. To reflect the impact of learning accuracy reduction, we introduce weighted data transmission time $W = \{W_1, W_2, ..., W_n\}$ to replace real uploading time P. The weighted data transmission time equals to the ratio of uploading time to relative accuracy, can be denoted as

$$W_i = \frac{\sum_{j=0}^{m} x_{i,j} * P_{i,j}}{A_i} \tag{2}$$

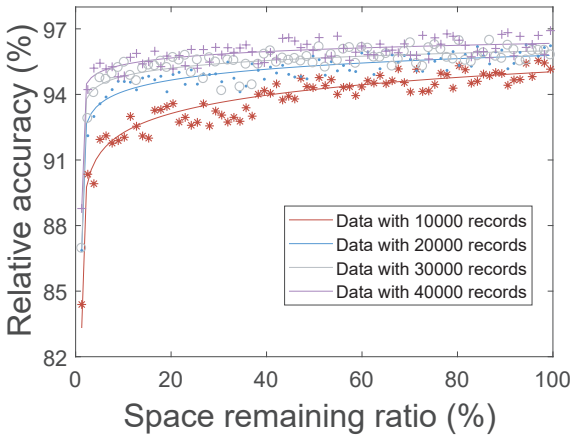


Fig. 3: Accuracy curve

Our goal is to minimize the maximum data transmission time on each edge server (makespan), so as to obtain

$$\min_{1\leq j\leq m} \max \sum_{x_{i,j}=1} W_i \tag{3}$$

s.t.

$$x_{i,j} \in \{0,1\}, \forall i,j \tag{4}$$

$$\sum_{M_j \in E_i} x_{i,j} = 1, \forall i \tag{5}$$

$$A_i \geq LowA_i, \forall i \tag{6}$$

Constraints (4) and (5) confine that for each job, it can be only assigned to exactly one edge server among those accessible servers. Constraint (6) is the relative learning accuracy bound constraint. Since learning accuracy will be influenced in the cloud-edge system, to guarantee the learning quality, a lower bound $0 \leq LowA_i \leq 1$ of each job's relative accuracy should be set.

C. Tradeoff Modelling: Accuracy vs Traffic Reduction

In this subsection, we design a specific scenario in our cloud-edge system to model the tradeoff between accuracy and traffic reduction. We choose to perform autoencoder compression algorithm on the edge server to preprocess the learning data gathered from end devices, and then send the processed data to the more powerful cloud for further training.

Autoencoder is an artificial neural network used for unsupervised learning of efficient codings. The neural network of autoencoder can be divided as three parts: the input layer, the hidden layers and the output layer. The learning process of the autoencoder can be explained as to learn an approximation to the identity function, where the input and the output are both the raw data it self. The neural network structure from the input layer to the hidden layers is called the encoder, and the structure from the hidden layers to the output layer is called the decoder. By extracting the information from hidden layers, most representative features from the raw data can be achieved. Thus compression can be realized by the encoder.

Autoencoders allows users to set the dimensionality of extracted features in the middle hidden layers. Also, other than different dimensionality reduction algorithms, such as Principle Component Analysis (PCA), which are restricted to a linear map, autoencoder can have nonlinear encoder/decoder. By setting appropriate dimensionality and sparsity constraints, autoencoders can produce more representative data projections. This is the main reason we choose autoencoder as the edge preprocessing algorithm.

Since autoencoder is a lossy compression method, the deep neural network training quality will be affected when we use the compressed data by autoencoder as the input data. To fully understand how learning accuracy is impacted by the traffic reduction, we build a real-world testbed. We realize the autoencoder with 3 fully connected hidden layers, and choose the MS-Celeb-1M dataset as the data source. The experiment result shows that with certain raw dataset size, the learning accuracy will increase but with a descending increasing rate along with the increase in space remaining ratio. Here, the space remaining ratio is defined as the degree of reduction in traffic size. To precisely describe the relationship between learning accuracy and space remaining ratio, we manage to model it as a logarithmic function as below.

$$A = a * \ln(R + b) + c \tag{7}$$

Here, A is the learning accuracy ratio between our edge learning and direct cloud learning, R is the space remaining ratio, a, b and c are the three parameters being calculated by curve fitting.

Fig. 3 presents the details of this regression. The real learning accuracy data are marked as points, while the logarithmic regression functions are plotted as the curves. Because the number of input data records are different, there are slight differences between these curves. However, the differences are becoming smaller when the number of data records increases. So that we can assume that the accuracy curve can be regarded as the same one if the number of training records is large enough. Now with this fitting function, we are able to predict the decreased learning accuracy under different space remaining ratio.

D. Formulation Transformation

In traditional task scheduling problem, each task usually costs fixed time to complete. In our system, W_i is however a variable and it is decided by the task arrangement information matrix $x_i = \{x_{i,1}, x_{i,2}, ..., x_{i,m}\}$, edge server bandwidth B_j, space remaining ratio R_i, dataset size D_i and learning accuracy A_i.

We inspect W_i with its corresponding parameters. In our formulated problem, the only difference between edge servers is their available upstream network bandwidth, so that A_i is not affected by x_i. Also from the accuracy vs traffic reduction analysis in section III. B, A_i can be represented as a function of R_i and D_i. In addition, B_j and D_i are fixed values.

Therefore, W_i can be actually described as the function with only two variables R_i and x_i which are irrelative. This means the R_i we choose for each task doesn't affect our scheduling. In other words, to pursue our goal of minimizing the overall makespan, there should exist a fixed value as the best space remaining ratio for each job no matter what the scheduling scheme is.

Here, we continue to use the specific edge learning scenario where the edge servers perform autoencoder as an example, to illustrate our observation with mathematical expressions. By combining equation (1)(2)(7), we can formulate W_i by a function of R_i and x_i as below.

$$W_i(R_i, x_i) = \frac{R_i}{a * \ln(R_i + b) + c} * \sum_{j=0}^{m} \frac{x_{i,j}}{B_j} * D_i \quad (8)$$

In expression (8), it is easy to find that R_i and $x_{i,j}$ are the only variables and they are irrelative which justifies our previous inference. Hence, whatever the generated task arrangement scheme is, it is feasible to get the best space remaining ratio $BestR_i$ for task J_i by finding the minimum point of (8). The common way to find the minimum point is to calculate the value R_i where the function's derivative equals to zero, and then exclude the maximum point solution. Also, it is important to notice that $BestR_i$ should be ranged from $(0, 1]$ and satisfies the constraint (6). So that the range of R_i can be represented as below.

$$\max(0, \exp(\frac{LowA_i - c}{a}) - b) \leq R_i \leq 1 \quad (9)$$

We use $BestR = \{BestR_1, BestR_2, ..., BestR_n\}$ to describe the set with all best space remaining ratios. As the best space remaining ratio can be calculated, our problem can be transformed as an Integer Linear Programming (ILP) problem:

$$\min \quad W^* \quad (10)$$

s.t.

$$\sum_{i=1}^{n} x_{i,j} * V_{i,j} \leq W^*, \forall j \quad (11)$$

$$\sum_{j=1}^{m} x_{i,j} = 1, \forall i \quad (12)$$

$$x_{i,j} \in \{0, 1\}, \forall i, j \quad (13)$$

$$V_{i,j} = \begin{cases} \frac{D_i * BestR_i}{B_j}, & M_j \in E_i \\ +\infty, & M_j \notin E_i \end{cases} \quad (14)$$

where W^* is the makespan, $BestR_i$ equals the best space remaining ratio for task J_i and $V_{i,j}$ equals to the weighted data transmission time for task J_i arranged on machine M_i with the best space remaining ratio $BestR_i$.

IV. Task Scheduling for Weighted Data Transmission Time Optimization

In this section, we propose two optimization algorithms to solve our task scheduling problem. The first algorithm takes a Linear Programming(LP)-relaxation on our formulated ILP problem, and give out a scheduling scheme based on the extreme point solution to the LP problem. Since solving the LP problem will take up a long period of time, we then propose the second algorithm which introduces simulated annealing, a heuristic strategy.

A. An Extreme Point Linear Programming Solution Based Rounding Algorithm

Our first algorithm is based on the extreme point solution to the LP problem. We first transform the ILP problem (10) - (14) into an LP problem by relaxing the constraint (13) to

$$x_{i,j} \geq 0 \quad (15)$$

We can use LP solver to calculate the optimal makespan for our LP problem, and to generate the extreme point solution as well. However, since $x_{i,j}$ can be a fractional value, it can be not be directly used as a scheduling scheme. We made modifications to this scheduling by assigning each task J_i to one machine M_j only when $x_{i,j}$ has the largest value among $1 \leq j \leq m$, so that all the values in x will be integral. We summarize this algorithm in Algorithm 1.

Our algorithm gives out a 2-approximation scheduling. For the proof, we first claim two properties of an extreme point solution to a LP problem.

Lemma 1. Any extreme point solution has at most $n + m$ nonzero variables.

Proof: For our formulated LP problem, there are $n * m$ variables and $n * m + n + m$ equalities and inequalities constraints. The $n * m + n + m$ constraints can be divided by three types: n of form $\sum_{j=1}^{m} x_{i,j} = 1$, m of form $\sum_{i=1}^{n} x_{i,j} * V_{i,j} \leq W^*$, and $n * m$ of form $x_{i,j} \geq 0$. Since extreme point solution will set $n * m$ independent constraints to equalities, thus there should be at least $n * m - n - m$ pairs of the third form inequalities $x_{i,j} \geq 0$ are set to be equality, so that at most $n + m$ variables are nonzero. ■

Lemma 2. Any extreme point solution has an upper bound m of tasks which are set fractionally.

Proof: We use α and β to represent the number of integrally set tasks and fractionally set tasks respectively. Due to constraint (14), we can get $\alpha + 2 * \beta \leq n + m$ since there must be at least two $x_{i,j}$ satisfying $x_{i,j} > 0$ for fractionally set task J_i. Because $\alpha + \beta = n$ and $\alpha + 2 * \beta \leq n + m$, we can get $\beta \leq m$. ■

We now show that our algorithm gives a 2-approximation scheduling comparing to the optimal solution.

Theorem 1. Algorithm 1 gives a 2-approximation scheduling.

Proof: Since there are at most m tasks are set fractionally, then we can arrange each machine only process at most one of

Algorithm 1 Extreme Point LP Solution Based Algorithm

Input: n, m, V, specified value ε
Output: x

Use LP solver to get the feasible solution for the minimum makespan W^*
$x \leftarrow$ an extreme point solution for the LP problem
for i from 1 to n **do**
  for j from 1 to m **do**
    if $x_{i,j}$ is the largest among $x_{i,k(1\leq k\leq m)}$ **then**
      $x_{i,j} \leftarrow 1$
    else
      $x_{i,j} \leftarrow 0$
    end if
  end for
end for
return x

these fractionally set tasks. Each fractional task's processing time is bounded by W^*, the resulting makespan is at most $2 \times W^*$. Because the LP optimal solution is better than ILP optimal solution, it's clear that $W^* \leq OPT$ where OPT is the optimal makespan. Thus our resulting makespan is at most $2 * W^* \leq 2 * OPT$. ■

B. A Simulated Annealing Rearranging Algorithm

The extreme point LP solution based algorithm costs long execution time due to the large-scale computation with LP solver. Thus we propose a heuristic algorithm which introduces the simulated annealing strategy.

This algorithm has two steps. The first step is to find a scheduling scheme with the greedy search. We first sort the tasks in a non-increasing order by its compressed data size $BestR_i * D_i$ under the best space remaining ratio, then greedily arrange each task to one specific edge server which makes current makespan smallest.

The second step is to rearrange the task. Here, we chose simulated annealing as the rearrangement strategy. Simulated annealing is a heuristic search algorithm. It uses probabilistic technique to achieve the approximation result which is close to the global optimum of a given function. The core idea of this algorithm is that it allows the state move which achieve better global return of the given function, while accepting the state move which gets worse global return under the dynamic probability at the same time. Each time the system accepts the worse state move, the probability of accepting the next worse state move will decrease. For our rearrangement process, each task will have a chance to be reassigned to one of the edge server. If this task is rearranged to the other edge server which obtains smaller global makespan, then we state it as a better state move and accept this move. Otherwise, we state it as a worse state move and probably accept this move under the dynamic probability. This rearrangement process will only last for a certain number of rounds concerning the running time. The whole algorithm's pseudo code is shown as Algorithm 2.

Algorithm 2 Simulated Annealing Rearranging Algorithm

Input:
n, m, D, $BestR$, V, specified limitation for rearrangement rounds $RoundLimit$, starting temperature Tp, temperature decreasing ratio Ra
Output: x

Sort the tasks according to the corresponding value of $D_i * BestR_i$ in a non-decreasing order where $1 \leq i \leq n$.
$Makespan \leftarrow 0$
for i from 1 to n **do**
  $j \leftarrow 1$
  $Makespan \leftarrow$ the makespan after $Connect(x, i, j)$
  $Temp \leftarrow 1$
  for j from 2 to m **do**
    $NewMakespan \leftarrow$ the makespan after $Connect(x, i, j)$
    if $NewMakespan < Makespan$ **then**
      $Temp \leftarrow j$
    end if
  end for
  $Connect(x, i, Temp)$
  $Makespan \leftarrow$ the makespan of x
end for
for r from 1 to $RoundLimit$ **do**
  for k from 1 to n **do**
    $i \leftarrow$ randomly pick one task which has not been reassigned in this round
    for j from 1 to m **do**
      $NewMakespan \leftarrow$ the makespan after $Connect(x, i, j)$
      $\delta \leftarrow NewMakespan - Makespan$
      if $\delta < 0$ **then**
        $Connect(x, i, j)$
        $Makespan \leftarrow NewMakespan$
      else
        if $e^{Tp} > random(0, 1)$ **then**
          $Connect(x, i, j)$
          $Makespan \leftarrow NewMakespan$
          $Tp \leftarrow Ra * Tp$
        end if
      end if
    end for
  end for
end for
return x

Our Algorithm 2 is running far more efficiently than Algorithm 1. The time complexity for Algorithm 2 is $O(n * m) + O(RoundLimit * n * m) = O(n * m)$, while solving the LP problem in Algorithm 1 has the time complexity at the exponential level for the worst case [10]. Since for real-world cases, there should be over thousands of edge servers, which will cost too much time when we use LP solver to generate a scheduling. So it is more realistic to choose Algorithm 2 as

the scheduling algorithm for real-world cases.

V. Performance Evaluation

In this section, we will introduce our testbed environment, and present the performance of our algorithms.

A. Simulation Implementation

We evaluate our algorithms with simulations. In the simulation environment, there are totally 10 edge servers and up to at most 600 learning tasks. For edge servers, we reference the available bandwidth across geo-distributed datacenters [11], and define each edge server has a random uploading bandwidth between 100 Mbps to 1,000 Mbps. For learning tasks, and the size of each data record is 15KB which is the same as the size of one picture from MS-Celeb-1M dataset. To calculate the best space remaining ratio for each learning task, we set the number of data records for one task is a random integer between 20,000 to 80,000, so that we can use the same regression curve. We choose the curve of data with 40,000 records in Fig.3, where the parameters in equation (2) are $a = 0.004032$, $b = -0.012755095$ and $c = 0.9634233$.

We design three typical stages for connections between end devices and edge servers: one-to-one connectable, one-to-more connectable, and fully connectable. The first stage one-to-one connectable controls that, for most end devices (80%) only has one edge server connectable, while only few end devices can access two edge servers. This stage fits for the early edge learning scenarios while there are only a few edge servers and most end devices can connect to only one edge server. For the second stage, each end device is connectable with one or several edge servers. This stage is designed for general edge learning scenarios. The third stage is designed for future edge learning scenarios where all edge servers in one specific area can communicate with each other with low communication cost. Thus the workload from one edge server can be easily transferred to another edge server, all edge servers can realize the full connection.

We choose the random choosing algorithm as the baseline algorithm, which will arrange the scheduling in a random way. This is the method for edge server connection arrangement without centralized scheduling from the cloud. Also, we select Ameer's multi-phase heuristic algorithm [12] which is a state-of-the-art algorithm for solving unrelated machines scheduling problem. Our algorithms proposed in Section V are comparing the performance with these two strategies for analysis.

B. Performance Evaluation

Firstly, we compare the overall maximum weighted data transmission time (makespan). For stage 1 one-to-one connectable, see Fig. 4(a), our simulated annealing rearranging algorithm and extreme point LP solution based algorithm both can get lower makespan with about 8% - 24% reduction comparing to the baseline algorithm. Also our algorithms returns the similar result as the multi-phase heuristic algorithm. The difference is not apparent since for each end device under this stage, there is almost no choice for it to choose a connection with more than one edge server. For stage 2, see Fig. 4(b), our algorithms start to show up with the more apparent reduction in makespan. It will reduce at least 20% and up to 30% in makespan comparing to the baseline and even achieve about 10% reduction comparing to the multi-phase heuristic algorithm. For stage 3, see Fig. 4(c). Comparing to the baseline, our algorithms can reduce about 30% - 50% of makespan which is similar to the multi-phase heuristic algorithm and further proves our solutions' efficiency. We can infer that when the network topology becomes more complicated, our algorithms are able to generate more effective scheduling schemes.

We also compare the standard deviation of each edge server's weighted data transmission time. See Fig. 5, Algorithm 1 and Algorithm 2 both produce scheduling with a less standard deviation of weighted data comparing with the baseline. And with the increase in the number of learning tasks, the difference with baseline becomes more obvious. Our algorithms can reduce at most about 2/3 times of the standard deviation of the baseline algorithm.

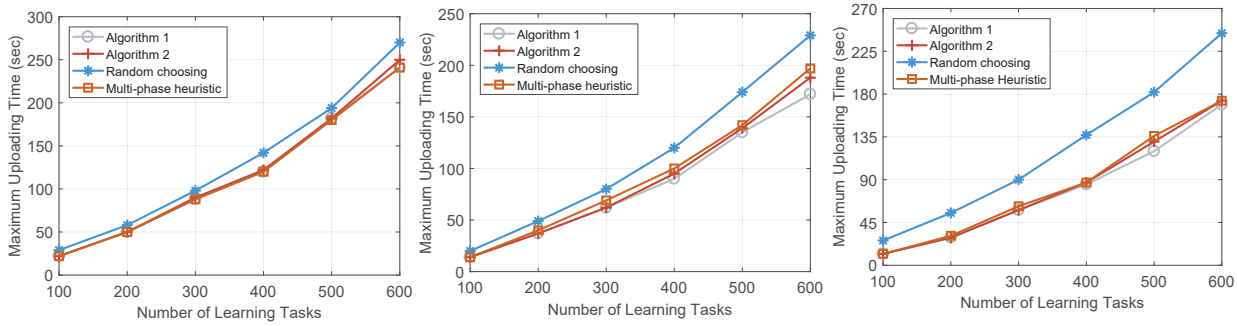
We also count the number of jobs on the most loaded edge server. Besides the network bandwidth bottleneck, each edge server also has limitations on computation resources. Too many tasks to be assigned to one edge server may overload the server and our algorithms should avoid this situation. We can see from these three graphs in Fig. 6, the scheduling generated by our algorithms has the least number of jobs on the most loaded edge server. So that our algorithm will less likely to overload the edge server.

In general, these experiments show that our proposed two algorithms both generate more balanced scheduling schemes with smaller makespan, smaller standard deviation of each edge server's weighted data transmission time and fewer jobs on the most loaded edge server comparing with the baseline random choosing algorithm. And also the generated scheduling's performance is similar to state-of-the-art multi-phase heuristic algorithm which further proves our algorithms are efficiency.

VI. Related Work

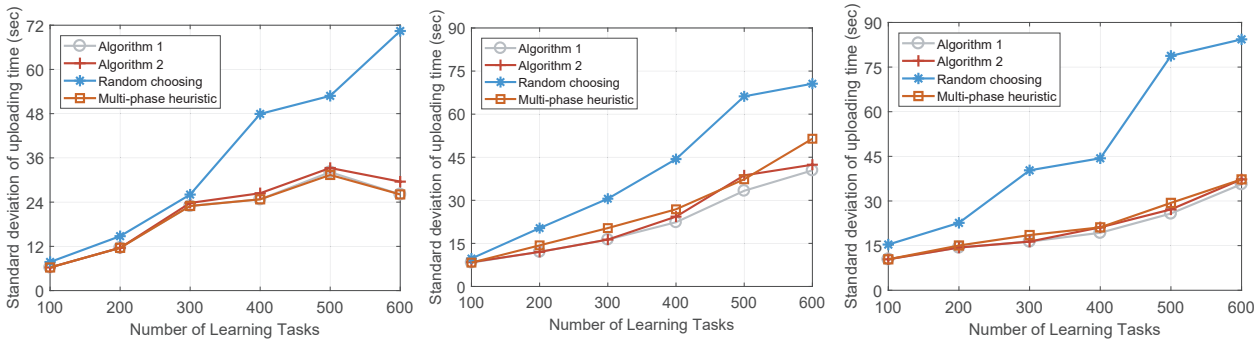
A. Deep Learning

Neural network approaches [13] have attracted significant efforts in recent decades. By overcoming the over-fitting problem, neural networks are evolved to be driven by deep learning and their applications are gaining renewed interest. In particular, Kasun *et al.* [14] introduced a feed-forward neural network with a fast learning speed and good generalization capability. Courbariaux *et al.* [15] proposed Binarized neural network (BNN) which is a recent type of neural network by setting the weights equals to -1 and 1 in linear and convolutional layers. Comparing to a standard floating-point neural network, BNN has emerged to outperform with less memory and reduced computation. McDanel *et al.* [16] proposed embedded binarized neural network (eBNN) which is an extended version of the BNN. It reorders the operation in inference and reduces floating point temporaries, so that embedded devices are more



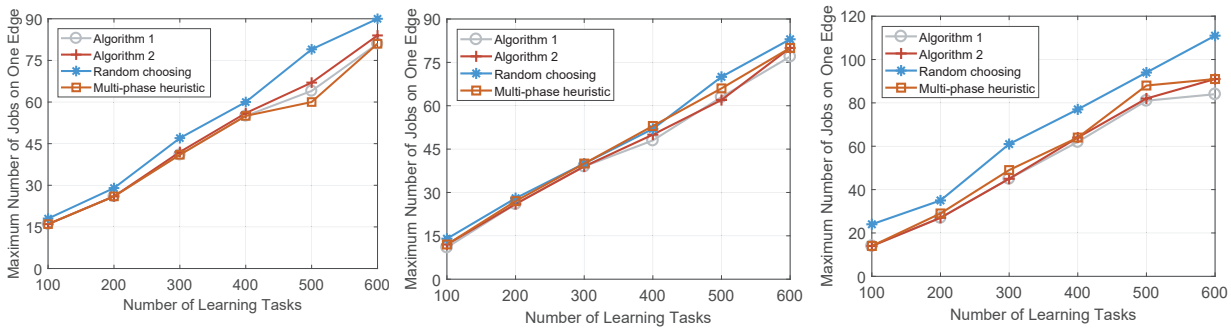
(a) Stage 1: one-to-one (b) Stage 2: one-to-many (c) Stage 3: fully connected

Fig. 4: Comparison of overall weighted data transmission time



(a) Stage 1: one-to-one (b) Stage 2: one-to-many (c) Stage 3: fully connected

Fig. 5: Comparison of standard deviation of weighted data transmission time



(a) Stage 1: one-to-one (b) Stage 2: one-to-many (c) Stage 3: fully connected

Fig. 6: Comparison of number of jobs on the most loaded edge server

easy to fit. Teerapittayanon *et al.* [17] introduced BranchyNet. They proposed a concept named early exit point. The idea is to classify samples at earlier points in a neural network and calculate the classification confidence criteria based on the entropy.

B. Edge Computing

Zhu *et al.* [18] proposed a content optimization infrastructure with the help of mobile edge computing. The content optimizer can analyze the network information and dynamically perform content optimization combining with the edge stored content optimization information. Karim *et al.* [19] proposed FemtoClouds to enable multiple mobile devices to

be configured into a coordinated cloud computing service, which leverages mobile devices to provide cloud services at the edge. Dawei *et al.* [20] proposed DeepCham, an adaptive mobile object recognition framework, which allows deep learning techniques to be used successfully in mobile environments. DeepCham introduces an edge master server to coordinate with participating mobile users and collaboratively train a domain-aware adaptation model. Chen *et al.* [21] proposed a distributed computational offloading model for mobile edge computing. Drolia *et al.* [22] proposed Cachier that is distributed across edge servers and devices for recognition applications. Huang *et al.* [23] proposed an approximation algorithm for the Connected Facility Location problem, which ensures caching fairness among peer devices in pervasive edge computing environment. Tan *et al.* [24] proposed online algorithms for the job dispatching and scheduling problem in edge-cloud systems. Li *et al.* [25] captured the unique features in both MEC (Mobile Edge Computing) and CRAN (Cloud Radio Access Network) with respect to communication and computation efficiency constraints. Wang *et al.* [26] proposed a novel online algorithm with a carefully designed logarithmic objective by producing feasible solutions for edge cloud resource allocation.

VII. Conclusion

In this paper, we concentrated on the issue of deep learning applications in a cloud-edge learning system. Deep learning with the cloud requires large amounts of training data to support while it brings huge traffic to the network and cost a long time to transmit the data. Edge learning becomes a promising method to solve the transmission issue while brings accuracy reduction. We study a task scheduling problem for reducing the maximum weighted uploading time on the edge server which also takes accuracy into consideration. Since there is a tradeoff between learning accuracy and traffic reduction, we can use the best space remaining ratio for edge learning tasks and reduce the problem to an ILP problem. Then we propose two algorithms and present their efficiency by simulation. Our simulation results show that our algorithms are able to outperform the baseline algorithm.

VIII. Acknowledgements

This research is supported by an NSERC Engage Grant and an NSERC Discovery Grant.

References

[1] McKinsey&Company, "How artificial intelligence can deliver real value to companies," https://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/how-artificial-intelligence-can-deliver-real-value-to-companies.

[2] Amazon, "Amazon web service cloud," https://aws.amazon.com.

[3] Google, "Google cloud," https://cloud.google.com.

[4] S. Teerapittayanon, B. McDanel, and H. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proceedings of 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 328–339.

[5] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[6] A. Ahmed and E. Ahmed, "A survey on mobile edge computing," in *Proceedings of 10th International Conference on Intelligent Systems and Control (ISCO)*. IEEE, 2016.

[7] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*. ACM, 2016, pp. 123–136.

[8] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2017, pp. 615–629.

[9] 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," *arXiv preprint arXiv:1804.05271*, 2018.

[10] G. B. Dantzig and M. N. Thapa, *Linear programming 2: theory and extensions*. Springer Science & Business Media, 2006.

[11] Z. Hu, B. Li, and J. Luo, "Flutter: Scheduling tasks closer to data across geo-distributed datacenters," in *Proceedings of 35th International Conference on Computer Communications (INFOCOM)*. IEEE, 2016, pp. 1–9.

[12] A. Al Salem and R. L. Armacost, "Unrelated machines scheduling with machine eligibility restrictions," 2002.

[13] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[14] L. L. C. Kasun, H. Zhou, G.-B. Huang, and C. M. Vong, "Representational learning with elms for big data," *IEEE Intelligent Systems*, vol. 28.

[15] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proceedings of the 29th Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 3123–3131.

[16] B. McDanel, S. Teerapittayanon, and H. Kung, "Embedded binarized neural networks," *arXiv preprint arXiv:1709.02260*, 2017.

[17] S. Teerapittayanon, B. McDanel, and H. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *Proceeding of the 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.

[18] J. Zhu, D. S. Chan, M. S. Prabhu, P. Natarajan, H. Hu, and F. Bonomi, "Improving web sites performance using edge servers in fog computing architecture," in *Proceedings of 7th International Symposium onService Oriented System Engineering (SOSE)*. IEEE, 2013, pp. 320–323.

[19] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *Proceedings of 8th International Conference on Cloud Computing (CLOUD)*. IEEE, 2015, pp. 9–16.

[20] D. Li, T. Salonidis, N. V. Desai, and M. C. Chuah, "Deepcham: Collaborative edge-mediated adaptive deep learning for mobile object recognition," in *Proceedings of 1st Symposium on Edge Computing (SEC)*. IEEE, 2016, pp. 64–76.

[21] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[22] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *Proceedings of 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 276–286.

[23] Y. Huang, X. Song, F. Ye, Y. Yang, and X. Li, "Fair caching algorithms for peer data sharing in pervasive edge computing environments," in *Proceedings of 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 605–614.

[24] H. Tan, Z. Han, X.-Y. Li, and F. C. Lau, "Online job dispatching and scheduling in edge-clouds," in *Proceedings of 36th International Conference on Computer Communications (INFOCOM)*. IEEE, 2017, pp. 1–9.

[25] T. Li, C. S. Magurawalage, K. Wang, K. Xu, K. Yang, and H. Wang, "On efficient offloading control in cloud radio access network with mobile edge computing," in *Proceedings of 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2258–2263.

[26] L. Wang, L. Jiao, J. Li, and M. Mühlhäuser, "Online resource allocation for arbitrary user mobility in distributed edge clouds," in *Proceedings of 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 1281–1290.