A Decomposed Deep Training Solution for Fog Computing Platforms

Jia Qian*
jiaq@dtu.dk
DTU Compute, Technical University of Denmark
Kongens Lyngby, Copenhagen, Denmark

Mohammadreza Barzegaran
mohba@dtu.dk
DTU Compute, Technical University of Denmark
Kongens Lyngby, Denmark

Abstract

Legacy machine learning solutions collect user data from data sources and place computation tasks in the Cloud. Such solutions eat communication capacity and compromise privacy with possible sensitive user data leakage. These concerns are resolved by Fog computing that integrates computation and communication in Fog nodes at the edge of the network enabling and pushing intelligence closer to the machines and devices. However, pushing computational tasks to the edge of the network requires high-end Fog nodes with powerful computation resources. This paper proposes a method whose computation tasks are decomposed and distributed among all the available resources. The more resource-demanding computation is placed in the Cloud, and the remainder is mapped to the Fog nodes using migration mechanisms in Fog computing platforms. Our presented method makes use of all available resources in a Fog computing platform while protecting user privacy. Furthermore, the proposed method optimizes the network traffic such that the high-critical applications running on the Fog nodes are not negatively impacted. We have implemented the (deep) neural networks - using our proposed method and evaluated the method on MNIST and CIFAR100 as the data source for the test cases. The results show advantages of our proposed method comparing to other methods, i.e., Cloud computing and Federated Learning, with better data protection and resource utilization.

ACM Reference Format:
Jia Qian and Mohammadreza Barzegaran. 2021. A Decomposed Deep Training Solution for Fog Computing Platforms. In *The Sixth ACM/IEEE Symposium on Edge Computing (SEC '21), December 14–17, 2021, San Jose, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3453142.3493509

SEC '21, December 14–17, 2021, San Jose, CA, USA

ACM ISBN 978-1-4503-8390-5/21/12…$15.00
https://doi.org/10.1145/3453142.3493509

1 Introduction

Big data has been expanded in all areas affecting business and research models [3, 18, 33]. These new models range from data-oriented economies that supply and facilitate data to the areas where data analytics help with improving productivity and reliability [37]. A challenge to realize this vision is the need of collecting a huge amount of data into data center facilities that are equipped with different communication solutions. However, the power of Big Data will be revealed with the use of Machine Learning (ML) [36] which requires powerful computation resources to perform [51]. Such computation and communication capabilities are integrated in Cloud Computing (CC) which has gained significant popularity and success in the past decade [25].

With CC providing cost-efficient, powerful computation and storage capabilities [25], Big data and ML have been spread in different application areas [25]. Companies such as Google, Microsoft, and Amazon have been providing different solutions for collecting data from producers, e.g., sensors, to their Cloud facilities and for data processing to analyze the data. Collecting all raw data from producers to data centers compromises data privacy, is more vulnerable to security attacks [6], and is forbidden according to General Data Protection Regulation (GDPR) in Europe. Data leakage has been agreed as one of the major challenges in the Cloud [34]. It is also massive and often repetitive that eats network bandwidth. Nevertheless, CC having non-deterministic behavior does not fit the applications that are latency-sensitive. An example of such area is the industrial applications that are high-critical, i.e., they have stringent timing requirements.

An architectural means to solve such issues is the Fog Computing (FC) that is defined as a "system-level architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things" [9]. An interchangeable term is Edge Computing (EC) [41] with the difference that FC emphasizes more about the infrastructures and EC focuses more on the computation part. A fog computing platform (FCP) brings computation and storage resources closer to the edge of the network and it is composed of several interconnected fog nodes (FNs). The vicinity property of FC outcasts CC by lower latency, local computation and high bandwidth for clients [50]. As shown in Figure 1, FNs are placed at the edge of the network and in the proximity of data sources.

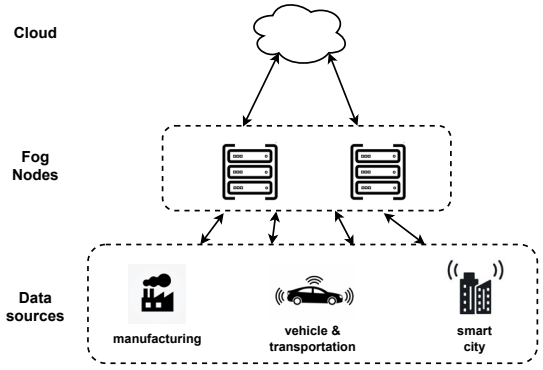

Figure 1. Overview of Fog Computing Platform

FC is literally pushing computation, data storage, data analytics, and service away from centralized Cloud to FNs. FC has been envisioned to implement different application areas such as industrial Internet of Things [28], self-driving vehicles [7], and smart healthcare [32] and so on [22] in the near future. Several types of FNs from powerful high-end FNs to low-end FNs with limited resources have been proposed by researchers [5, 29] and have been developed by companies [26, 47].

FC enables performing intelligence at the edge of the network using the integrated computation resources and data storage of FNs. It also avoids the raw data transmission between FNs and the Cloud, avoiding privacy leakage and bandwidth overuse. To this end, a favorable computation framework - Federated Learning (FL) [16] is proposed by Google where they suggest exchange gradients between FNs and the Cloud to jointly train a global model that may capture the information of the whole environment. However, FL may not be suitable in applications where FNs have less computation budget.

Unlike CC that places all the computation on Cloud and FL that leaves all the computation on the FNs, we would like to propose a new approach in which the computation is split among FNs and Cloud. We consider Deep Learning (DL) as the model, and the most costly computational task is model training and divides it into two steps: forward and backward. We feed inputs to the model during forward step and use the chain rule to compute gradients in backward step. As pointed out by [13], in most cases, backward computation is much more expensive than forward computation. Therefore, we assign a part of forward computation to the FN, and the remainder together with the backward computation to the Cloud.

Our main contributions are as follows. We motivate the demand of addressing the computation decomposition between FCP and server, for applications that require more computational resources that could not be met by edge devices. We propose an approach that decomposes the training process and tests the proposed approach on neural network models since it often requires more computational expense than other ML models [10]. We compare our proposed methods with other CC and plain FL and evaluate the performance of our proposed methods on several test cases. We compare the model performance, communication cost and also simulate it on OMNET++.

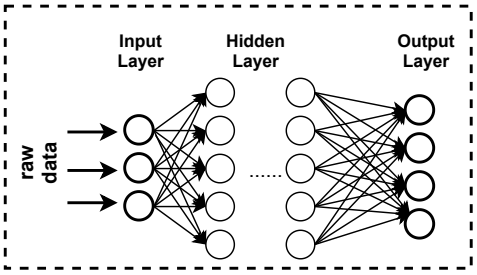

Figure 2. Multilayer Perception

The remainder of this paper is structured as follows. We present CC-based and FL in Section 2. Afterward, Section 3 presents the details of our proposed method. We evaluate our proposed approach on test cases MNIST and CIFAR100. The related work is presented in Section 5 and Section 6 concludes the paper.

2 Background and Related Concepts

Deep Learning (DL) [20] has gained significant success and beat the state-of-the-art in many fields such as object detection and recognition, speech recognition, medical diagnosis [46], and many other domains, e.g., genomics [30]. The presence of a neural network avoids the non-trivial feature extraction work, which was required in conventional ML methods. Instead, the model takes raw data as the input, and the model automatically carries out the feature extraction work. There are several types of neural network architectures such as Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Long short-term memory (LSTM).

MLP [35] was first invented and intended to model how the human brain processed visual data and learned to recognize objects. It is a set of layers stacking together with both linear and non-linear transformations in an interleaving way, sees Figure 2. CNN [21] is probably the most widely used network since it largely reduced the number of model parameters comparing with MLP. Instead of one unit connecting with all units in the previous layer, it only connects to the receptive field where convolution kernel is applied, see Figure 3. The convolution kernel is shared in the convolution layer, which made CNN shift-invariant. Namely, certain features can be detected regardless of their location in the input image. Moreover, RNN [38] and LSTM [12] are often used for speech recognition or any sequential data. In this work we use CNN and MLP in our proposed implementation method.

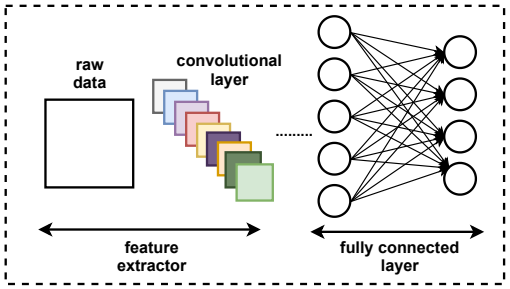

Figure 3. Convolutional Neural Network

2.1 Cloud-based Deep Learning

In CC-based DL, the FNs collect data from the sensors connected to them and transfer the raw data to the Could where DL model is implemented as computation tasks for training. Figure 4 shows the demonstration of the CC-based DL which consists of p FNs.

Each FN receives the input data produced by the connected sensors. We denote the input data as $x \in \mathbb{R}^d$. The FN integrates B data instances to form one batch, thus each batch transmits $B \times d$. Typically the batch size is much smaller than the input data dimension $d \gg B$. Since all the p FNs transmit data to the Cloud, the size of total data transmitted in each iteration is $p \times B \times d$. We only consider one iteration of data transmission for the comparison with all the other methods. The communication cost is fixed in CC regardless of the ML models since only training data is considered. However, in some applications, the data storage is not permanent on Cloud, and the same data is transmitted multiple times, which is another downside of CC. For instance, using GPU in Colab [4] requires transmitting data whenever it reconnects.

2.2 Federated Deep Learning

Federated Learning (FL) was first proposed by Google as a solution for large-scale mobile network training [16]. It suggests distributing the computational workload among nodes connected via the network, whereas the server in the network maintains globally the shared parameters of

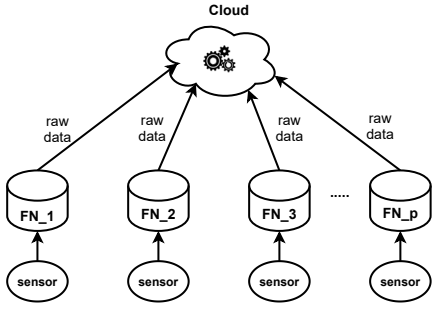

Figure 4. Cloud Computing-based ML method

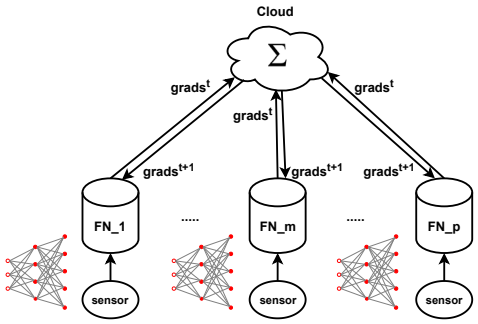

Figure 5. Federated Deep Learning at round t.

DL, which are in the form of dense or sparse vectors and matrices. There is a similar work in the literature named as "parameter server" [23], which focuses on the system design part and FL caught more attention in ML community.

In FL diagram, the FNs or data sensor perform most of the computation, and the Cloud aggregates and shares the parameters, shown in Figure 5. Each FN implements local training using its own data (from its connected sensor) and sends the gradient information to the Cloud. On the other hand, the Cloud aggregates all the gradient information from FNs and sends the aggregated gradient information back to them for the next round of local training. The overall view of the FL method is introduced in Algorithm 1.

At each round, every FN receives B instances of raw data (each instance has size d) from sensors connected to it and takes the raw data as the input to perform the local computation. Presumably, the data denoted as (X, Y) is generated from the empirical data distribution $\hat{\mathcal{D}}$. ℓ_j is defined as the local loss function in FN j, v_j^t is the Jacobian matrix (containing all the partial derivatives) of FN j at round t, and η is the learning rate. Cloud instead performs the simple arithmetic calculations that aggregate the model parameters from all the active FNs.

Algorithm 1 Federated Learning

Input: w^0, X, Y
for t=1,...T **do**
  =>**workers:**
  for j=1,2,..,p **do** p devices (in Parallel)
    1) $v_j^t = \nabla \hat{\ell}_j(f(X_j^t; w^t), Y_j^t)$
    with $((X_j^t = \{x_{jk}^t\}_{k=1,..B}, Y_j^t = \{y_{jk}^t\}_{k=1,...,B}) \sim \hat{\mathcal{D}}$
    2) share v_j^t with server
  end
  =>**server:**
  $\overline{v}^t = \frac{1}{p} \sum_{j=1}^{p} v_j^t$
  $w^{t+1} = w^t - \eta \times \overline{v}^t$
  share w^{t+1} with devices for next round
end
Return w^T

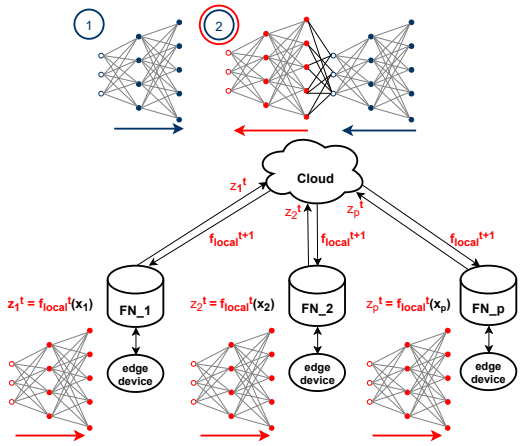

Figure 6. Fog Computing-based Decomposed Deep Training.

Leaving most computation on the local devices constrains the range of applications that FL may be applicable. For instance, it is not suitable for the scenario that the computational resources on device are insufficient to carry out all the tasks. On the other hand, the available assets on Cloud are not fully utilized neither, e.g., computation power and storage.

Regarding to the communication cost, say we have a neural network f with m+n layers and $|v_j^t|$ represents the number of elements in Jacobian matrix v_j^t. At each round t, FNs and the Cloud transmit data with total size of $2 \times p \times \sum_j^{m+n} |v_j^t|$ or $2 \times p \times \sum_j^{m+n} |w_j^t|$ for the two-way transmission (from devices to server and from server to devices).

3 Solution

In this section, we present our method that decomposes the computational tasks among the distributed FNs and the Cloud. More specifically, the overall model is divided into two parts: local model and remote model. The local model is stored on FNs and Cloud, whereas the remote model is only stored on Cloud. Note the Cloud also owns a copy of the local model for the error propagation step. On the FNs side, first, they collect data from sensors connected to them, and the data is used as the input to the local models (named as *local computation*). *Local computation* is one part of the overall foreward step. Second, all FNs send the intermediate results (output of local model) to the Cloud. On the Cloud side, it first finishes the rest of the forward task on the remote model and then implements the overall backward computation (error propagation) for training using the combo of the local and remote models. The proposed method is illustrated in Figure 6 where the local models and the remote model are shown in red and navy blue circles, respectively.

An overview of our proposed method FCDDT is presented in Algorithm 2 where two main functions FogCom and CloudCom are introduced for all the computational tasks on

Algorithm 2 Fog Computing-based Decomposed Deep Training (FCDDT)

Input: X, Y, $[w^1_{\textbf{local}}, w^1_{\textbf{remote}}]$, η
for t=1,...,T **do**
  for j=1,...,p **do** p devices (in Parallel)
    $z_j^t = \textbf{FogCom}(X_j^t; w^t_{\text{local}})$ ▷ Computation on fog nodes (FNs)
    with $((X_j^t = \{x_{jk}^t\}_{k=1,..B}, Y_j^t = \{y_{jk}^t\}_{k=1,...,B}) \sim \hat{\mathcal{D}}$ ▷ j indicates data on worker j
  end
  $z^t = [z_1^t, z_2^t, .., z_p^t]$ ▷ Cloud collects the outputs from FNs
  $y^t = [y_1^t, y_2^t, ..., y_p^t]$ ▷ Cloud collects the labels from FNs
  $\hat{y}^t = \textbf{CloudCom}(z^t, y^t, \eta)$ ▷ Computation on Cloud
end
Return $[w^T_{\textbf{local}}, w^T_{\textbf{remote}}]$

FNs and Cloud. At round t, FogCom takes X_j^t as the input and returns the intermediate result z_j^t where X_j^t represents the training data of FN j at round t. After the active FNs shared the intermediate output and labels with the server, the server implements CloudCom, which outputs the updated parameters of the local model w^{t+1}_{local}. This procedure repeats T times and finally it returns the complete model parameterized $[w^T_{\text{local}}, w^T_{\text{remote}}]$. The unbalanced computational resources on FNs might delay the aggregation on Cloud. However, the synchronization is not necessarily required, in particular, when the number of FNs or batch size is big, see Section 4 for the related observation.

We present the details of computation on Cloud (CloudCom) in Algorithm 3. CloudCom takes all the outputs from local computation and the labels at the round t, and the learning rate η as the inputs. CloudCom first completes the forward step computation using the remote model f_{remote}, and then implements the backward step computation and shares the updated parameters of the local model with FNs.

Implementation: We assume that the remote process (CloudCom) is implemented as an application on the Cloud that is connected to the FCP. However, implementing the local process (FogComp) requires a mechanism for application deployment. Thus, we assume that FogComp is implemented as dedicated applications for each FN, and they are submitted to the FCP via the Cloud. Once submitted, the *Fog Controller* FN, which is determined at runtime using mechanisms such as [15], receives the submission request. Since the Fog controller has knowledge of the available resources on the FNs using the resource discovery algorithms such as [8, 15, 27] at runtime, it can decide the placement of applications on the FNs using a decentralized resource allocation technique [2, 43]. In the case of an FN leaving the FCP, Fog controller is able to migrate application from the leaving FN to a next available FN.

Algorithm 3 CloudCom

1: **Input:** z^t, η, y^t ▷ z^t is the combination of all outputs from FNs, η is the learning rate and y^t is the labels at round t
2: $\hat{y}^t = f_{\text{remote}}(z^t; w_{\text{remote}})$ ▷ f_{remote} is the remote model on server, parameterized by w^t_{remote}
3: $w^{t+1} = w^t - \eta \times \nabla_w \hat{\ell}(\hat{y}^t, y^t)$ ▷ $\nabla_w \hat{\ell}(\hat{y}^t, y^t)$ is gradients w.r.t. all the parameters
4: **Return** $w^{t+1}_{\textbf{local}}$ ▷ w^{t+1}_{local} is one part of w^{t+1}

Communication Expense: The network traffic in FCDDT consists of exchanging messages that enable the decomposed training. The message exchanging can be split into two parts: (1) the messages uploading intermediate results and labels to the Cloud at each round and (2) the messages downloading updated model parameters from the Cloud to the FNs. Considering the p number of FNs in the FCP that collects B number of data instances to a data batch, and l_m nodes sitting in layer m with the assumption that it is a fully-connected layer. Thus, the total size of uploading messages is equal to $p \times B \times l_m + B$. Since $p \times B \times l_m \gg B$, we can easily omit B. Similarly, the size of downloading messages is equal to $p \times \sum_i^m |w_i|$.

4 Experiments and the Results

The structure of this section is as follows. We first describe our test setup and the test cases we used for evaluation in Section 4.1. We compare our proposed FCDDT method with the related work in Section 4.2. Afterward, we evaluate the accuracy of our proposed method on the test cases in Section 4.3. Section 4.4 presents the OMNET++ simulation and evaluation results on the traffic latency.

4.1 Test Cases and Setup

We implemented our method in Python and ran it on a Titan X GPU with Architecture Maxwell. The performance of FCDDT is evaluated on test cases that are derived from two sets of images: MNIST and CIFAR10. Each image in MNIST has the size of 28×28 pixels and represents gray-scale handwritten digits. MNIST consists of 50K images for training and 10K images for test that are grouped to 10 classes corresponding to digits from zero to nine. Each image in CIFAR10 has the size of $3 \times 32 \times 32$ pixels (i.e., images are RGB) and represents an object of 10 classes such as airplanes, automobiles, etc. CIFAR10 consists of 50K images for training and 10K images for the test. FCDDT can implement different ML methods, thus, we used a two-layer CNN, two-layer MLP for MNIST, LeNet5, and VGG for CIFAR10. The introduction of CNN and MLP are already presented in Section 2. LeNet5 [20] is a commonly used convolutional neural network, composed of 5 convolutional layers and one dense layer (fully connected). VGG was proposed by [42] and designed for large-scale images classification tasks. It uses the small kernel with size 3×3, and for more details, we refer the readers to [42].

We also evaluated the effect of FCDDT on the network traffic using simulation in OMNET++. The simulation is carried out on 6 test cases with the increasing number of FNs in the FCP. We assumed that the FNs in each test case exchange both critical and ML-related messages. The critical message exchanging between an FN and the Cloud is modeled as a periodic request with the size of 1400 bytes and a periodic reply with the size of 5000 bytes. The periods of the request and reply messages are equal to 400 ms. Each FN exchanges critical messages with two of its neighbors, which are similarly modeled as periodic request and reply messages that have periods are randomly chosen among 200, 300, and 400 ms and sizes equal to 1400 and 4000 bytes, respectively.

Moreover, the ML messages are also modeled as periodic request and reply messages that have equal periodic to 1000 ms. The size of request messages is 1200 bytes and the size of reply messages is equal to the data size in each method with data from MNIST. We simulate the network traffic for a duration of hyperperiod, i.e., the least common divisor of all message periods, and evaluate FCDDT for its effect on the mean bandwidth usage and mean imposed delay on the critical messages.

Table 1. Comparison of FCDDT with the related work

Feature	FCDDT	FL	CC
Compute nodes	FNs & Cloud	FNs	Cloud
Raw data exchange	No	No	Yes
Comm. cost	$p(Bl_m + \sum_i^m \lvert w_i \rvert)$	$2p(\sum_i^{m+n} \lvert w_i \rvert)$	pBd
Implementation	via Cloud	on each FN	via Cloud

4.2 Comparison with the related work

We compare the features of FCDDT with the related work in Table 1. The related work consists of FL [16] and CC [25] solutions. The first feature in the table represents the computation platform where each solution runs. As shown in the table, FCDDT is the only solution that uses all the resources in the FCP and distributes computation among the processing nodes. Note here in FL, the Cloud only carries out easy the arithmetic calculation, we ignore it in Table 1. Regarding privacy, both FL and FCDDT avoid sending raw data and only transmit the model parameters or other intermediate results. However, as shown in the table, FCDDT is more efficient comparing with FL in communication cost since it uses less bandwidth $Bl_m + \sum_i^m |w_i| < 2 \sum_i^{m+n} |w_i|$. Note we can further save bandwidth by allocating a really small local model to FNs (in terms of depth and width of neural network) such that $\sum_i^m |w_i|$ is relatively small. CC requires least bandwidth without considering re-connection problem that we discussed in Section 2.1, but it compromises the sensitive data privacy.

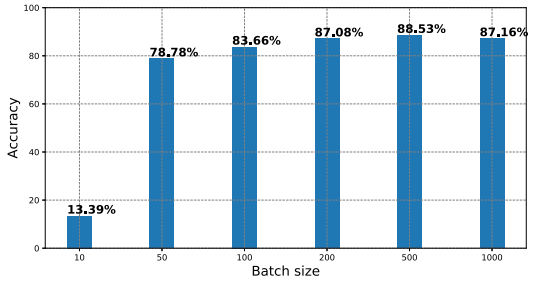

Figure 7. batch size comparison

CC solution is suitable for the implementation on large scale FCPs since the training model applications are only implemented to run on the Cloud, thus it is scalable concerning the network size. Our proposed solution FCDDT uses a similar approach by employing application deployment mechanisms that automatically place applications on the FNs, which makes the solution scalable as well. However, FL requires all nodes to be programmed individually, unless a specific mechanism is introduced for a large scale FCP.

4.3 Performance evaluation

The accuracy evaluations of FCDDT on datasets MNIST and CIFAR10 using different ML methods are shown in Table 2 and Table 3 accordingly. As the results show, our method FCDDT either has identical performance or mildly degraded comparing with CC and FL. Note that we assume local training happened on FNs in FL framework for only one iteration and we repeat the experiments for 5 times to evaluate the standard deviation.

Moreover, we explore the impact of batch size on accuracy using FCDDT. We test it on CIFAR10 in Figure 7 where VGG16 is applied as the model. As we mentioned before, the synchronization between FNs is not necessarily required. Here we introduce the *straggler rate* α to simulate the scenario where the Cloud aggregates the transmission from top $1 - \alpha$ FNs. Figure 8 shows that the different straggler rates have various impacts on the model performance when batch size is different. The four colors represent batch size equal

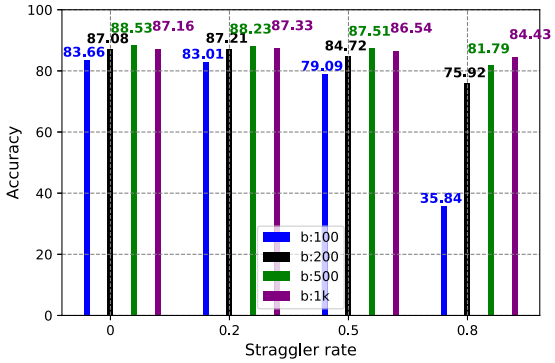

Figure 8. Straggler rate comparison (CIFAR10). Percentage sign is omitted. $b : x$ represents the batch size.

Table 2. MNIST performance

Name	CC	FL	FCDDT
MLP (Two layers)	0.9 ± 0.007	0.9 ± 0.01	0.89 ± 0.003
CNN (Two layers)	0.96 ± 0.0014	0.96 ± 0.003	0.96 ± 0.002

to 100, 200, 500, 1K accordingly. As we can see, the accuracy drops drastically when batch size is 100, and the straggler rate decreases to 0.8. However, it has no significant influence on the performance when the batch is equal to 500 or 1K even with four FNs. We suggest using the big batch size in the environment where computational resources are not evenly distributed among FNs, and the *straggler* phenomenon is likely to happen.

4.4 Network traffic evaluation

We simulate the network traffic for FCDDT and the related work for the 8 test cases in Table 4. It includes the mean end-to-end delay for all messages in μs and the mean bandwidth usage for the traffic. More specifically, the simulated results show that FCDDT is able to reach 16% less bandwidth usage comparing to FL and 19% less than CC. The saved bandwidth usage of FCDDT contributes to decreasing the average end-to-end delay to 59 μs, which is 8 μs and 18 μs less than FL and CC, respectively.

A key observation is that in the case where no ML service is running on the FCP, i.e., ignoring the ML messages, the average bandwidth usage is 20% and the average end-to-end delay is 55 μs. Taking this as the baseline, our method FCDDT introduces only 1% and 4 μs for the average bandwidth usage and the average end-to-end delay in addition, with ML service.

5 Related Work

It's challenging to train the machine learning model on one device with the exponential growth of data and connected devices that generate data. Two types of parallel technologies are often used: *data parallelism* and *model parallelism*. Data parallelism is commonly used where each processor (GPU) owns one complete copy of the model and processes one subset of the data. It is effective in some applications when data is naturally distributed across multiple devices, and processing data in a parallel way saves the cost of data transmission to one centralized machine. However, it requires frequent communications between processors for the synchronization of models trained separately. It was considered as the main drawback of data parallelism [24, 39, 44]. It [45]

Table 3. CIFAR10 performance

Name	CC	FL	FCDDT
LeNet5	0.445 ± 0.01	0.45 ± 0.05	0.45 ± 0.003
VGG16	0.93 ± 0.074	0.92 ± 0.13	0.885 ± 0.297

Table 4. Simulation results on six test cases

#	No. of FNs	FCDDT		FL		CC	
		E2E delay (μs)	bandwidth usage	E2E delay (μs)	bandwidth usage	E2E delay (μs)	bandwidth usage
1	4	49	18%	52	21%	57	22%
2	7	52	19%	56	23%	65	24%
3	10	56	21%	63	25%	75	26%
4	12	62	22%	69	25%	83	27%
5	15	66	24%	77	28%	89	29%
6	18	71	25%	84	29%	96	31%

demonstrates how to translate high-level data parallel to GPU programs. The authors from [40] experimentally study the effect of increasing batch size during training, measured by the number of steps required to reach a preset error, and how does this relationship changes with respect to different training methods, network architecture, and dataset.

Another category is model parallelism. The increasing size of neural networks (layers and parameters) can no surprise result in higher accuracy. However, there is a limit to the maximum model size fit in a single processor (GPU). Thus, the model is partitioned into several parts, and each part is processed on one GPU. Each GPU is responsible for its weight updates accordingly. The main drawback of model parallelism techniques is the dependence between processors, namely, the processor has to wait for the previous processor to finish its work and take the output of the previous processor as the input. They [11] introduce a paralleled computation by pipelining execution across multiple machines, which show 95% for large deep neural networks relative to data-parallel training. Instead of dividing the model into consecutive layers, it partitions the model horizontally [17]. They design a model that is horizontally symmetric, so it can be separated into two parts and train independently on two processors, and combine at the end of each backward.

A similar work [49] study the offloaded task from edge devices to server where they divided the framework into three layers: edge devices, edge servers, and server. They only considered the case when local training on edge devices occurred only once before the communication with the edge server. Another work [48] additionally proposed to apply Reinforcement Learning to automatically offload the computational task between server and edge devices. Again, they did not consider multiple epochs on edge devices. They [1] empirically study the deep neural network-powered apps in the Android market and identify some optimization techniques, such as model pruning and quantization. The other works like [14, 19] focus on the system design on offloading the neural network computation from edge devices to the server.

Our method is a mixture of data and model parallelism. Different from the works mentioned before, we focus more on the ML side and particularly propose a decomposition approach under the federated framework. The approach is flexible to applications with the varying placement of computational and storage resources.

6 Conclusion and future work

Fog Computing as an enabler for Industry 4.0 which integrates mixed-criticality applications on a shared computing platform envision to run data analysis at the edge of the network. This capability prevents sending all data to the Cloud, thus prevents the user data leakage and eating communication capacity. Although Fog Computing Platforms employ separation mechanisms to protect critical applications, the resource-demanding data analysis computation may debilitate Fog nodes from providing critical applications sufficient resources.

We propose a decomposed deep training solution that distributes its workload among the Fog nodes and the Cloud. It is generally applicable to any neural network architecture, but to save the bandwidth, we suggest dividing the neural network at the layer producing the output with a small dimension, e.g., a fully-connected layer with few nodes. With such a solution, less resource-demanding computation is assigned to the Fog nodes, and the remainder of computation is assigned to the Cloud, which exchanges messages for performing data analytics. To further enhance the data security, we refer to another work [31], which studies the lower bound of model structure to avoid the possible input reconstruction. The simulated network traffic in OMNET++ shows that our method is more suitable for mixed-critical systems than the literature. In our future work, we will consider the optimal decomposition of the computation that will use the maximum available resources on the Fog nodes.

The limitation of our method is that only one iteration is allowed to implement locally before the communication with the server. We believe that it can be addressed by a hierarchical FL architecture where the sub-network is introduced and composed of a set of devices and a FN. The sub-network may approximate the local Stochastic Gradient Descent (SGD) with the assumption that the devices and FN are connected via an intranet. Or we can explore another way of model decomposition.

References

[1] Mario Almeida, Stefanos Laskaridis, Abhinav Mehrotra, Lukasz Dudziak, Ilias Leontiadis, and Nicholas D Lane. 2021. Smart at what cost? Characterising Mobile Deep Neural Networks in the wild. *arXiv preprint arXiv:2109.13963* (2021).

[2] Cosmin Avasalcai, Christos Tsigkanos, and Schahram Dustdar. 2019. Decentralized resource auctioning for latency-sensitive edge computing. In *IEEE International Conference on Edge Computing*. 72–76.

[3] Andrew L Beam and Isaac S Kohane. 2018. Big data and machine learning in health care. *Jama* 319, 13 (2018), 1317–1318.

[4] Ekaba Bisong. 2019. Google colaboratory. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Springer, 59–64.

[5] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. 13–16.

[6] Deyan Chen and Hong Zhao. 2012. Data security and privacy protection issues in cloud computing. In *2012 International Conference on Computer Science and Electronics Engineering*, Vol. 1. IEEE, 647–651.

[7] Mung Chiang, Bharath Balasubramanian, and Flavio Bonomi. 2017. *Fog for 5G and IoT*. Vol. 288. Wiley Online Library.

[8] European Telecommunications Standards Institute. 2016 (accessed May 7, 2021). Mobile edge computing (MEC) framework and reference architecture (GS MEC 003 V1.1.1). http://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/01.01.01_60/gs_MEC003v010101p.pdf.

[9] Fog Computing and Networking Architecture Framework. 2018 (accessed April 1, 2021). IEEE 1934-2018 - IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing. https://standards.ieee.org/standard/1934-2018.html.

[10] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[11] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. 2018. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377* (2018).

[12] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[13] Zhouyuan Huo, Bin Gu, Heng Huang, et al. 2018. Decoupled parallel backpropagation with convergence guarantee. In *International Conference on Machine Learning*. PMLR, 2098–2106.

[14] Hyuk-Jin Jeong, Hyeon-Jae Lee, Chang Hyun Shin, and Soo-Mook Moon. 2018. IONN: Incremental offloading of neural network computations from mobile devices to edge servers. In *Proceedings of the ACM Symposium on Cloud Computing*. 401–411.

[15] Vasileios Karagiannis and Apostolos Papageorgiou. 2017. Network-integrated edge computing orchestrator for application placement. In *IEEE International Conference on Network and Service Management*. 1–5.

[16] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012), 1097–1105.

[18] Alexandros Labrinidis and Hosagrahar V Jagadish. 2012. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment* 5, 12 (2012), 2032–2033.

[19] Stefanos Laskaridis, Stylianos I Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D Lane. 2020. SPINN: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–15.

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[21] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. 1999. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*. Springer, 319–345.

[22] Kanghyo Lee, Donghyun Kim, Dongsoo Ha, Ubaidullah Rajput, and Heekuck Oh. 2015. On security and privacy issues of fog computing supported Internet of Things environment. In *2015 6th International Conference on the Network of the Future (NOF)*. IEEE, 1–3.

[23] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*. 583–598.

[24] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. 2017. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887* (2017).

[25] Sean Marston, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang, and Anand Ghalsasi. 2011. Cloud computing—The business perspective. *Decision support systems* 51, 1 (2011), 176–189.

[26] Nebbiolo Technologies, Inc. 2021 (accessed April 1, 2021). Nebbiolo. https://www.nebbiolo.tech/.

[27] OpenStack. 2020 (accessed May 7, 2021). Documentation on Nova Scheduler. https://docs.openstack.org/developer/nova/filter_scheduler.html.

[28] Paul Pop, Bahram Zarrin, Mohammadreza Barzegaran, Stefan Schulte, Sasikumar Punnekkat, Jan Ruh, and Wilfried Steiner. 2021. The FORA Fog Computing Platform for Industrial IoT. *Information Systems* 98 (2021), 101727.

[29] Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito, and Omer Rana. 2019. Fog computing for the Internet of Things: A Survey. *ACM Transactions on Internet Technology (TOIT)* 19, 2 (2019), 1–41.

[30] Jia Qian and Matteo Comin. 2019. MetaCon: unsupervised clustering of metagenomic contigs with probabilistic k-mers statistics and coverage. *BMC bioinformatics* 20, 9 (2019), 1–12.

[31] Jia Qian, Hiba Nassar, and Lars Kai Hansen. 2020. Minimal conditions analysis of gradient-based reconstruction in Federated Learning. *arXiv preprint arXiv:2010.15718* (2020).

[32] Jia Qian, Prayag Tiwari, Sarada Prasad Gochhayat, and Hari Mohan Pandey. 2020. A noble double-dictionary-based ECG compression technique for IoTH. *IEEE Internet of Things Journal* 7, 10 (2020), 10160–10170.

[33] Junfei Qiu, Qihui Wu, Guoru Ding, Yuhua Xu, and Shuo Feng. 2016. A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing* 2016, 1 (2016), 1–16.

[34] R Velumadhava Rao and K Selvamani. 2015. Data security challenges and its solutions in cloud computing. *Procedia Computer Science* 48 (2015), 204–209.

[35] Frank Rosenblatt. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* 65, 6 (1958), 386.

[36] Seref Sagiroglu and Duygu Sinanc. 2013. Big data: A review. In *2013 international conference on collaboration technologies and systems (CTS)*. IEEE, 42–47.

[37] Ralph Schroeder. 2016. Big data business models: Challenges and opportunities. *Cogent Social Sciences* 2, 1 (2016), 1166924.

[38] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing* 45, 11 (1997), 2673–2681.

[39] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns. In *Fifteenth Annual Conference of the International Speech Communication Association*.

[40] Christopher J Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. 2018. Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600* (2018).

[41] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE internet of things journal* 3, 5 (2016), 637–646.

[42] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[43] Olena Skarlat, Vasileios Karagiannis, Thomas Rausch, Kevin Bachmann, and Stefan Schulte. 2018. A framework for optimization, service placement, and runtime operation in the fog. In *IEEE International Conference on Utility and Cloud Computing*. 164–173.

[44] Nikko Strom. 2015. Scalable distributed DNN training using commodity GPU cloud computing. In *Sixteenth Annual Conference of the International Speech Communication Association*.

[45] David Tarditi, Sidd Puri, and Jose Oglesby. 2006. Accelerator: using data parallelism to program GPUs for general-purpose uses. *ACM SIGPLAN Notices* 41, 11 (2006), 325–335.

[46] Prayag Tiwari, Jia Qian, Qiuchi Li, Benyou Wang, Deepak Gupta, Ashish Khanna, Joel JPC Rodrigues, and Victor Hugo C de Albuquerque. 2018. Detection of subtype blood cells using deep learning. *Cognitive Systems Research* 52 (2018), 1036–1044.

[47] TTTech Computertechnik AG. 2019 (accessed October 7, 2019). Nerve. http://tttech.com/products/industrial/industrial-iot/nerve.

[48] Di Wu, Rehmat Ullah, Paul Harvey, Peter Kilpatrick, Ivor Spence, and Blesson Varghese. 2021. FedAdapt: Adaptive Offloading for IoT Devices in Federated Learning. *arXiv preprint arXiv:2107.04271* (2021).

[49] Yunfan Ye, Shen Li, Fang Liu, Yonghao Tang, and Wanting Hu. 2020. EdgeFed: Optimized federated learning based on edge computing. *IEEE Access* 8 (2020), 209191–209198.

[50] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. 2015. Fog computing: Platform and applications. In *2015 Third IEEE workshop on hot topics in web systems and technologies (HotWeb)*. IEEE, 73–78.

[51] Lina Zhou, Shimei Pan, Jianwu Wang, and Athanasios V Vasilakos. 2017. Machine learning on big data: Opportunities and challenges. *Neurocomputing* 237 (2017), 350–361.