A Framework for Distributed Deep Neural Network Training with Heterogeneous Computing Platforms

Bontak Gu[1], Joonho Kong[1], Arslan Munir[2], Young Geun Kim[3]

[1]School of Electronics Engineering, Kyungpook National University, Daegu, South Korea, {gubontak, joonho.kong}@knu.ac.kr
[2]Dept. of Computer Science, Kansas State University, Manhattan, KS, amunir@ksu.edu
[3]Dept. of Computer Science, Korea University, Seoul, South Korea, carrotyone@korea.ac.kr

***Abstract*— Deep neural network (DNN) training is generally performed by cloud computing platforms. However, cloud-based training has several problems such as network bottleneck, server management cost, and privacy. To overcome these problems, one of the most promising solutions is distributed DNN model training which trains the model with not only high-performance servers but also low-end power-efficient mobile edge or user devices. However, due to the lack of a framework which can provide an optimal cluster configuration (i.e., determining which computing devices participate in DNN training tasks), it is difficult to perform efficient DNN model training considering DNN service providers' preferences such as training time or energy efficiency. In this paper, we introduce a novel framework for distributed DNN training that determines the best training cluster configuration with available heterogeneous computing resources. Our proposed framework utilizes pre-training with a small number of training steps and estimates training time, power, energy, and energy-delay product (EDP) for each possible training cluster configuration. Based on the estimated metrics, our framework performs DNN training for the remaining steps with the chosen best cluster configurations depending on DNN service providers' preferences. Our framework is implemented in TensorFlow and evaluated with three heterogeneous computing platforms and five widely used DNN models. According to our experimental results, in 76.67% of the cases, our framework chooses the best cluster configuration depending on DNN service providers' preferences with only a small training time overhead.**

Keywords*— Distributed processing; Deep neural network; Training time; Energy efficiency; Edge Computing

I. INTRODUCTION

Deep neural networks (DNNs) are being adopted in an increasing number of application domains including speech recognition, machine translation, medical image analysis, autonomous driving, and games. As DNNs permeate the mobile application market, it is imperative to develop efficient DNN frameworks for both training and inference. Particularly, since battery-powered mobile devices typically have tight energy and power budgets, appropriate management of power and energy consumption when training or inferencing the DNNs is crucial.

While DNN inference tasks are often executed in mobile edge or end-user devices, the most widely used method for DNN training is cloud-based processing that offloads the DNN training tasks to powerful cloud computing devices or platforms. However, training DNN models on cloud for an increasing number of mobile and Internet of things (IoT) devices would not only result in excessive data communication burden to the network and data processing load for the cloud platforms but would also incur a huge cost for the DNN service provider. The problems arising due to training DNN models on cloud can be summarized as follows. First, there could be serious network bottleneck and data processing burden in cloud as a huge amount of the data must be uploaded to the cloud servers. This network bottleneck may also result in large latency in DNN training. Second, the cloud-only DNN training may incur an excessive cost as total cost of ownership (TCO) for maintaining a cloud-based server system is huge. Instead of deploying one's own customized cloud, one may choose cloud computing services such as Amazon EC2 [2]; however, the cost of using commercial cloud computing platforms is not trivial. Third, the use of commercial cloud can create privacy issues as user/training data, which may contain sensitive information (e.g., medical images), needs to be sent to the cloud servers. Fourth and finally, to support on-device incremental learning [30], it is crucial to have DNN training capability in edge nodes or devices.

To overcome the above mentioned limitations in cloud-based DNN training, many studies related to edge- or user-side device processing have been carried out [12][21][25][31] as the computing capability of those devices is increasing. The core enablers for the edge or user-side device processing technology are high-performance application processors or system-on-chips which are generally outfitted with artificial-intelligence (AI) processing units or powerful graphics processing units (GPUs). These edge/user devices can help cloud servers by enabling distributed DNN training [10][18][29][32], which performs the training with both high-performance servers and edge/user-side devices.

Recent studies on distributed DNN training mainly focus on reducing network communication latency overheads in multi-GPU or multi-processor systems that leads to performance improvement. Even though performance improvement is still crucial for efficient distributed DNN training, it is also very important to consider costs (e.g., TCO) and energy consumption of the overall distributed cluster systems [13][28]. As several different design metrics exist in configuring the distributed DNN training cluster, there could be different preferences depending on the DNN server maintainers or service providers. Hence, it is imperative to consider those preferences together when performing distributed DNN training.

In this paper, we propose a novel framework for distributed DNN training with heterogeneous computing platforms. The proposed framework takes into account the characteristics of heterogeneous computing platforms/devices. For example, heterogeneous platforms typically have different CPUs (e.g., high performance server CPUs or low-power embedded CPUs), GPUs, memory capacity, storage capacity, etc., which manifest different performance and energy efficiency when

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

running DNN training. With our proposed framework, DNN service providers choose one among four metrics (training time, power, energy consumption, and energy-delay product). Our framework then composes a training cluster that is estimated (by our framework) as the best training cluster for the chosen metric. The proposed framework is evaluated for five prominent DNN models such as AlexNet [20], CIFAR10 (model provided by TensorFlow[1], not a dataset), Network in Network (NiN) [24], ResNet [14], and SqueezeNet [17] with a mix of high-/mid-/low-end devices for distributed processing.

We summarize our contributions as follows:

- We propose a novel DNN training framework that comprehensively considers the characteristics of the heterogeneous devices, communication overheads, available network bandwidths, and DNN service providers' preferences.
- We propose an analytical model for training time estimation in the distributed processing environment with heterogeneous devices and a regression-based power model for accurate power estimation for DNN training.
- We implement our proposed framework based on TensorFlow which is one of the most widely used DNN frameworks.
- We show that the proposed framework selects the best configurations in 76.67% of the total feasible device cluster configurations, which means our framework leads to better distributed DNN training considering DNN service providers' preference.
- The training cluster chosen by our proposed framework actually results in reduced training time, power, energy, and EDP by 45.5%, 4.5%, 31.6%, and 27.6%, respectively, compared to the second-best cluster configurations, on average.

The rest of the paper is organized as follows. Section II discusses recent related work. Section III explains our proposed framework in detail. Section IV demonstrates our experimental results and lastly, Section V concludes this paper.

II. Related Work

A. Mobile Device-Based DNN Processing

For efficient DNN training and inference, many studies regarding the mobile device-based processing have been carried out. Guo [12] evaluated and compared performance of user device-based and cloud-based inference. According to their analysis, cloud-based inference shows reasonable response time and energy consumption, whereas user device-based inference is feasible in a certain scenario because of the computing capability constraints in mobile devices. In [21], for mobile sensor-based inference, Lane et al. demonstrated a low-power inference engine prototype in a mobile device by using CPU and DSP. When applying deep neural networks to a various artificial intelligence applications, their proposed framework shows a feasibility of running inference tasks in mobile domains under the resource constraints and a robustness in inference tasks. Deepmon framework [16] appropriately modifies DNN models for continuous image or vision data processing while extracting the video frame characteristics to reduce the amount of the data required for DNN processing. The studies or proposals introduced above demonstrate a feasibility or solution for mobile device-based inferences. Compared to these works, our work mainly focuses on the distributed DNN training tasks considering the heterogeneity of the diverse computing devices or platforms from high-performance devices to low-end mobile user or edge devices.

B. Task Offloading for Distributed DNN Processing

Task offloading can be a good alternative for efficient DNN processing that leads to fast response time and low energy consumption. DeepX [22] divides a DNN model to unit-blocks and assigns them to heterogeneous processors in a local device (e.g., CPUs, GPUs, DSPs, etc.) considering the resource usages while performing model reduction when there is a resource constraint. Xu et al. [33] proposed a technique to reduce a DNN model for mobile inference tasks and assigns tasks to mobile and wearable devices considering the user preference and resource status in the devices. These works [22][33] focus on efficient DNN inference in mobile devices while our work focuses on the efficient DNN training with heterogeneous devices. Deep^3 [27] proposes a unified framework for DNN training and inference with heterogeneous devices. Deep^3 framework aims to determine the best neural network size by considering hardware resource constraints of computing platforms. In addition, it includes a new DNN graph traversal method that efficiently exploits parallelisms at various levels (e.g., hardware, neural networks, etc.). However, their work [27] did not consider network communication overheads while our framework considers impacts of the available network bandwidth on distributed DNN training.

C. Communication-Aware Distributed DNN Processing

Since communication overhead is a huge factor in distributed DNN training performance and energy efficiency, many studies which consider communication overheads in distributed DNN processing have also been performed. Keuper et al. [18] analyzed the impact of communication overheads, parallelism in matrix computations, and data distributions in distributed DNNs by using data-parallelized stochastic gradient descent (SGD). According to their analysis, scaling more than 16 nodes in distributed DNN processing is not efficient due to the communication overheads (i.e., communication-bound due to a large number of nodes). Sergeev and Del Balso [29] demonstrated that communication overheads in distributed DNN training are a huge factor for DNN training performance. They showed that the effective hardware utilization is only around 50% when training Inception V3 and ResNet-101 models with 128 Nvidia Pascal GPUs due to the communication overheads. They also proposed *Horovod* which applies *Baidu's ring-allreduce* algorithm [3] to improve inter-GPU communication

[1] Available at: https://www.tensorflow.org/tutorials/images/deep_cnn#cifar-10_model
In this paper, we call this DNN model as 'CIFAR10' as they call it as 'CIFAR10' model in the above URL. Please also note that 'CIFAR10' in this paper does not indicate a dataset [4].

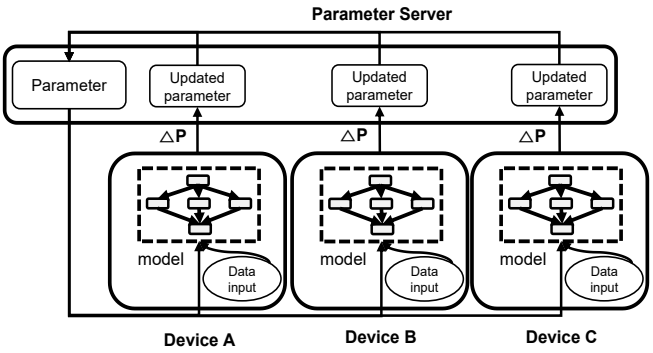


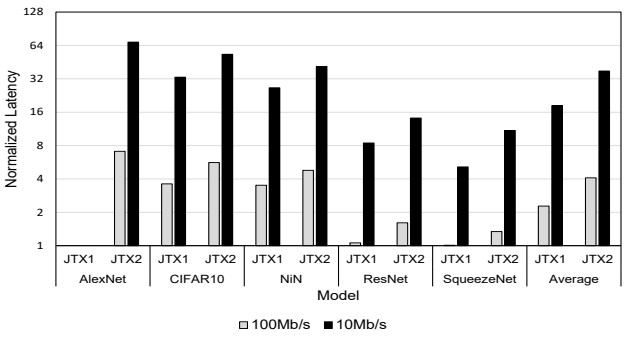
Fig. 1. A conceptual block diagram of data parallelism with asynchronous update.

performance. Compared to [29], our work assumes that we use heterogeneous devices whose performance and available resources are diverse (from low-end to high-end devices). It means our framework can be more broadly applied to real-world systems where many types of the heterogeneous devices can exist in the cluster. Hsieh et al. [15] demonstrated that the geological location of the computing server devices significantly affects distributed DNN training performance. For example, if a distributed DNN training task requires to communicate between Singapore and Sao Paolo through WAN, performance is worsened by 13.7X~26.8X. Thus, they proposed to reduce communication between the devices by updating the data only when they have sufficiently meaningful training results (i.e., more than a certain threshold) from the local devices. Similarly, Wang et al. [32] proposed a control algorithm for the best trade-off between the local update and global parameter aggregation while reducing an adverse impact on the convergence rate of DNNs. Their proposed algorithm tries to reduce communication overheads arisen from the synchronization of the parameters which hardly affect the convergence rate of the DNNs. However, their algorithm does not consider available network bandwidth which significantly affects distributed DNN training performance and energy-efficiency.

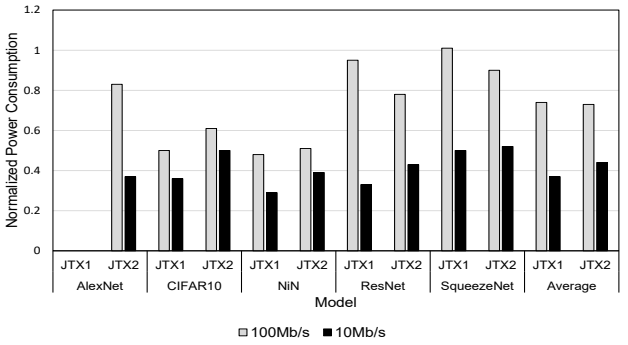
III. Our Proposed Framework

In this section, we introduce our proposed deep neural network (DNN) training framework with heterogeneous computing platforms or devices. Our framework considers available device resource, communication overhead, network bandwidth, and DNN service provider's preference to derive the best distributed DNN training cluster configurations with a given set of available computing devices. For criteria of DNN service providers' preferences, we consider training time, power, energy, and energy-delay product (EDP).

Fig. 1 depicts our baseline distributed DNN training architecture with data parallelism, asynchronous update, and star topology (parameter server architecture [23] is one of the representative realizations of the star topology). In each device, the entire DNN model is loaded while the datasets for DNN model training in each device are different. Each device asynchronously sends and updates the training outputs to the parameter server in order to maintain the up-to-date global model state.



(a) Execution time per step (y-axis: $\log_2$ scale)



(b) Power consumption

Fig. 2. Training time and power impacts due to network bandwidth normalized to the case of 1Gbps network bandwidth. In the case of AlexNet, JTX1 cannot participate in the training due to hardware limitation of the JTX1.

A. Key Challenges to be Addressed in Distributed DNN Training with Heterogeneous Devices

1) Parameter communication overhead and limited network bandwidth

In the case of distributed DNN training, communication overhead (latency and power) hugely affects training time and energy efficiency of the DNN training cluster. In [15], [18], and [29], they already observed that the network status and communication overhead are key factors for distributed DNN training performance and energy efficiency. Since DNN has many different network architectures such as convolutional neural network (CNN), recurrent neural network (RNN), and generative adversarial network (GAN), etc., the available DNN models have diverse sizes (e.g., layers, number of parameters, etc.). Depending on these characteristics of the DNN models, communication intensity between the nodes also varies. According to our evaluation and analysis on distributed DNN training with heterogeneous devices, as we have smaller DNN models, parameter communication overhead more hugely affects training time and energy efficiency of the DNN training. The reason is that the portion of the parameter communication latency among the entire execution time will be larger than that of the computation latency. In addition, if we configure the distributed DNN training cluster with high-performance devices, the parameter communication overhead will also hugely affect performance and energy efficiency during the DNN training because relative data communication latency and energy will also be

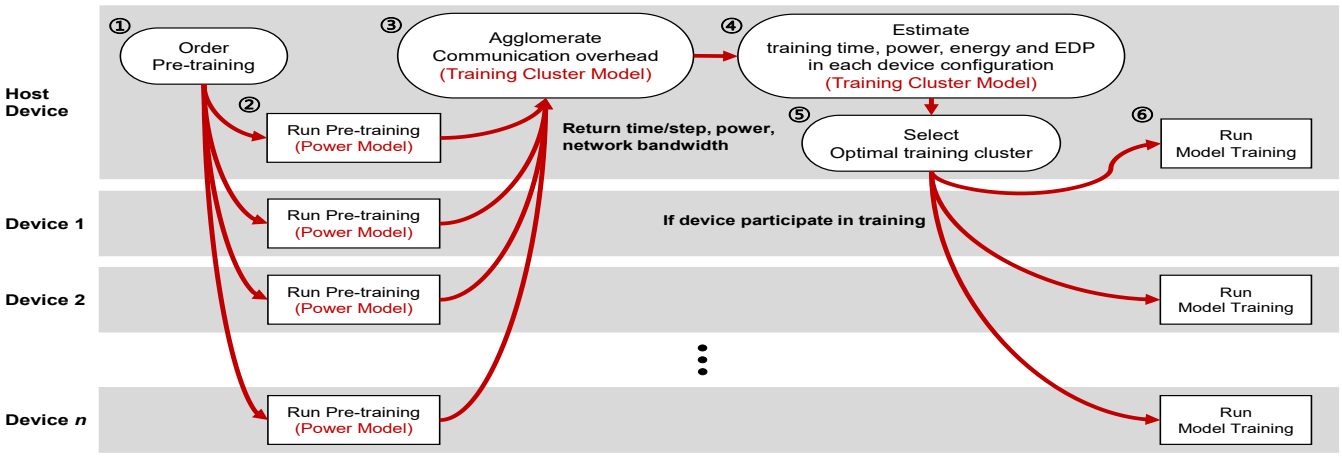


Fig. 3. Overview of the proposed DNN training framework. In this example, Device 1 is excluded from the cluster as our framework decides that participating Device 1 in the cluster is not efficient for meeting the DNN service providers' preferences.

larger. On the other hand, if we compose a distributed DNN training cluster with a mix of high-end and low-end computing devices, the main performance bottleneck would be not low performance of the low-end computing devices but decreased throughput of high-end devices due to the parameter communication overhead particularly when training a large DNN model. Consequently, we should carefully consider the communication overheads attributed to the parameter communication, limited network bandwidth, computing capability of the devices, and DNN model size, which in turn affect training time, power, energy, and EDP of the distributed DNN training cluster.

Along with the parameter communication overhead, limited network bandwidths in the cluster are also a key challenge. Fig. 2 depicts our preliminary evaluation results that show the training time per step and power consumption when varying the network bandwidth (1Gbps, 100Mbps, and 10Mbps) with heterogeneous computing devices. We perform the experiments with the Desktop PC as a parameter server and Nvidia Jetson TX1 (JTX1) or Jetson TX2 (JTX2) as a worker (please refer to Section IV-A for more detailed hardware specifications and our experimental setup). As the network bandwidth decreases, the time per step significantly increases. Compared to the case of the 1Gbps bandwidth, when using JTX2 as a worker, time per step increases by ~4 times and >32 times in the cases of the 100Mbps and 10Mbps network bandwidths, respectively. Power consumption tends to be reduced when the network bandwidth decreases. This is mainly because the idle time increases in the computing devices due to higher waiting delay in the nodes (i.e., waiting for data from other nodes). In the case of the energy consumption, due to the increased time per step, energy consumption will significantly increase despite of reduced power consumption. As demonstrated in Fig. 2, if the available network bandwidths are not carefully considered, training time and energy efficiency of DNN training would also be seriously hurt.

2) Power and energy constraints

Though improving DNN training performance is a key issue for deploying DNN in many real-world applications, reducing TCO is also one of the important challenges in the datacenters or large-scale server systems. To overcome this problem, some of the training tasks can be taken over from the cloud servers to the mid-end or low-end computing devices such as mobile end-user (i.e., local) or edge devices [26] as the computing capability of the local or edge devices increases. However, mobile edge or local computing devices generally have a limited power and energy budget. Thus, improving power and energy efficiency is also a very important challenge in distributed DNN training not only for high-performance server devices but also for mid-/low-end (mobile) edge or end-user devices.

Previous works for power and energy management in distributed DNNs aim to: 1) remove or reduce redundant operations by modifying the DNN training algorithms [9], and 2) distribute subtasks to mobile devices and cloud servers by dividing the DNN model by layers [11] in order to manage battery power in mobile devices and computational loads in cloud servers. As those works already demonstrated, we need a holistic framework for distributed DNN training which takes into account power and energy efficiency as well as training time.

B. Framework Overview

Considering DNN service providers' preference, we propose a distributed DNN framework that takes into consideration different characteristics of heterogeneous computing devices. Our framework comprehensively considers the two key challenges in distributed DNN training: communication overhead and power/energy constraints. Fig. 3 depicts an overview of our proposed framework. Our framework has six steps for distributed DNN training. Each step works as follows:

① Before the actual DNN training, in order to find the best distributed DNN training cluster configuration (i.e., which devices should participate in or not), our framework orders a DNN pre-training in each device for a small number of the steps.

② Each device measures available network bandwidth to consider communication overheads and executes DNN training for a small number of the steps. In this phase, each device measures average training time per step and estimates power consumption by

using the power estimation model (for more details, see Section III-C2).

③. With the measured training time per step and estimated power in ②, our framework reflects communication overhead (i.e., parameter communication and network bandwidth) in order to precisely estimate distributed DNN training time and power consumption under a certain cluster configuration (for more details, see Equation (2) and (3) in Section III-D).

④. With the results from ② and ③, by using our training cluster model (see Section III-D), our framework estimates training time, power, energy, and EDP for each cluster configuration candidate.

⑤. Depending on DNN service providers' preference[2] (i.e., in terms of training time, power, energy, or EDP), our framework chooses the best cluster configuration and orders the DNN training to the selected devices.

⑥. The selected devices start the DNN training for the remaining training steps. When performing distributed training with multiple heterogeneous devices, the remaining steps are dynamically distributed in runtime (i.e., upon finishing a certain training step, the next remaining step is immediately assigned to the available devices).

In the following subsections, we explain pre-training and training cluster model in more details.

C. Pre-Training

1) Training time per step measurements

In stochastic gradient descent (SGD) methods, which are generally used for DNN training, throughput (i.e., amount of computation) typically tends to be steadily maintained across training steps because computation in each step is iteratively performed with sub-datasets. Consequently, by only sampling a small number of training steps, one can easily and accurately estimate the entire training time and power consumption. In the pre-training phase of our framework, DNN training is performed in each device for 100 steps. Upon finishing the pre-training, our framework collects the training time results from each device and calculates *training time per step* (denoted as T_i where i corresponds to a certain device) for each device.

On the other hand, picking up the sample of 100 steps immediately after starting the DNN model training may lead to inaccurate training time estimation. According to our empirical results, T_i severely fluctuates before finishing around 300th step while it is almost stable for the rest of the training steps (i.e., after around 300th step). The main reason for this phenomenon would be due to warming up of the hardware and software in the early-steps of the DNN training. Thus, we measure T_i with the 100 steps after 300th step (i.e., the results from 301~400th training step). Please note that we run pre-training in each device for 100 steps (301~400th step). Though we could reduce the training time overhead by running the different training steps in each device, supporting the training of different steps for each device is out of scope of this paper. Along with the T_i, we also measure model build time (denoted as T^B_i where i corresponds to a certain device) in each device as a large DNN model generally requires a huge model build time before starting the execution of training steps.

Our training time estimation model shows only small error rates: 0.98%, 3.34%, 16.01%, 1.13%, and 3.88% for AlexNet, CIFAR10, NiN, ResNet, and SqueezeNet, respectively. In the case of NiN, a training time estimation error seems to be a little higher than the other models. This is because it takes longer time to enter a stabilized phase than the other models. In this case, we may be able to perform pre-training later than 400th step. However, late pre-training also incurs training time and energy overhead because we should use all available devices for training time estimation until finishing the pre-training. Considering the trade-off between the training time estimation accuracy and pre-training overheads, we decided to use 301~400th training steps for training time estimation. Even with the training time estimation error in NiN model, it is still tolerable for selecting the best cluster configuration (see our experimental results).

2) Power estimation model

To accurately estimate power consumption in each device, we propose a linear regression-based power estimation model similar to [19]. In this model, various performance counters and device-specific parameters are collected during 401~500th training step. The following equation is used for the linear regression:

$$\begin{aligned} P_i = y &+ x_1 \times F_{cpu} + x_2 \times F_{gpu} + x_3 \times F_{emc} + x_4 \times F_{cp} + \\ & x_5 \times U_{cpu} + x_6 \times U_{gpu} + x_7 \times U_{cp} + x_8 \times U_{mem} + \\ & x_9 \times M_{gpu} + x_{10} \times M_{swap} + x_{11} \times C_{cp} \end{aligned} \quad (1)$$

For accurate power estimation, 11 performance counter values are collected from each device while x_1~x_{11} and y values are determined through a linear regression method with the actual power measurement results, which can be done in offline. Please note that F, U, M, and C mean the clock frequency, utilization rate, memory usage and the number of active cores, respectively. The subscripts in Equation (1) also corresponds to the hardware components in devices (*emc*: external memory controller, *cp*: co-processor, *swap*: swap memory). The estimated power consumption for each device will be used to calculate energy and EDP with estimated training time.

Our power estimation model accurately estimates power consumption of different computing devices. For Desktop PC, JTX1, and JTX2, average estimation errors are 2.2%, 1.73%, and 1.17%, respectively, when running DNN training with TensorFlow and MNIST dataset.

D. Training Cluster Model

After the pre-training, our framework obtains the T_i, T^B_i, and P_i, which correspond to average training time per step for device i, model build time for device i, and estimated average power consumption for device i, respectively. By aggregating the estimated results, our framework estimates

[2] In this paper, our proposed framework can only select one metric among four. However, we could also give weights to multiple metrics (e.g., 30% for training time and 70% for energy) to select the best configuration though it is out-of-the-scope of this paper. We leave it as our future work.

the total training time (T^T_j), total power (P^T_j), total energy (E^T_j), and total EDP (EDP^T_j) in a certain cluster configuration *j*.

To integrate communication overheads into our model, we derive T^C_i (average training time per step of device *i* including communication overheads) and P^C_i (average power of device *i* including communication overheads) from T_i and P_i, respectively, by using the following Equations.

$$T^C_i = T_i \times \alpha^T_i \times \beta^T_{i,k} \quad (2)$$

$$P^C_i = P_i \times \alpha^P_i \times \beta^P_{i,k} \quad (3)$$

α^T_i , $\beta^T_{i,k}$, α^P_i, and $\beta^P_{i,k}$ values are adjusting factors to reflect the impacts of parameter communication overheads and network bandwidths. α^T_i (α^P_i) means a weight value for training time (power) to reflect impacts of parameter communication on device *i*. $\beta^T_{i,k}$ ($\beta^P_{i,k}$) means a weight value for training time (power) to reflect impacts of network bandwidth on device *i* under network bandwidth *k*. Please note that α^T_i, $\beta^T_{i,k}$, α^P_i, and $\beta^P_{i,k}$ can be empirically determined through multiple measurements with varying the network bandwidths and thoroughly calibrated. We could develop the systematic method instead of relying on empirical measurements to determine these values though we also leave it as our future work.

Assuming that we have *N* available computing devices in a certain device cluster configuration, we can derive the total estimated training time for a certain cluster configuration by using the following Equation:

$$S = \sum \frac{T^T_j - T^B_i}{T^C_i} \ (i = 1 \dots N) \quad (4)$$

where T^T_j and *S* means the total estimated training time for a certain cluster configuration *j* and the number of the training steps, respectively. In this work, we use 100,000 for *S* value.

For power, energy, and EDP, we can obtain each result by using Equations (5), (6), and (7), respectively.

$$P^T_j = \sum P^C_i \ (i = 1 \dots N) \quad (5)$$

$$E^T_j = P^T_j \times T^T_j \quad (6)$$

$$EDP^T_j = T^T_j \times E^T_j \quad (7)$$

For all possible cluster configurations, T^T_j, P^T_j, E^T_j, and EDP^T_j are calculated. After that, our framework sorts the possible cluster configurations according to DNN service providers' preference. Finally, our framework picks the best (estimated by our model) cluster configuration and actually performs DNN model training for the remaining steps with the selected device cluster configuration.

IV. Experimental Results

A. Experimental Setup

1) Deep learning framework

For our experiments, we use GPU-mode TensorFlow [5] which is one of the most widely used distributed deep learning frameworks. For large-scale distributed training, our TensorFlow adopts parameter-server architecture [23]. For parameter server selection from multiple devices, we select the CPU of the device which shows the highest throughput (i.e., performance) among the available computing devices in a certain cluster configuration. GPUs in each device run as workers. In the case of single-device cluster, device GPU works as both the parameter server and worker.

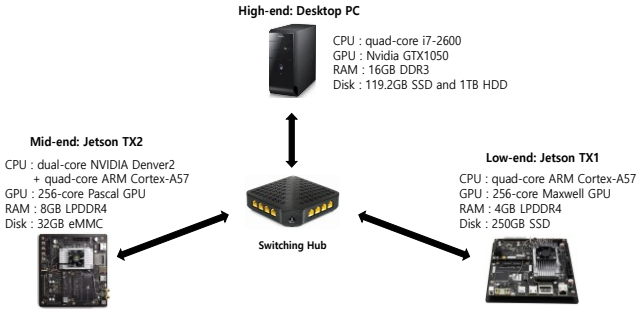


Fig. 4. Experimental environment and device specifications.

TensorFlow provides two different methods to implement data parallelism: In-Graph Replication and Between-Graph Replication. In the case of In-Graph Replication, the TensorFlow only generates one model graph which is divided and assigned to each worker and the final output is aggregated. On the contrary, Between-Graph Replication generates similar graphs in each worker, reducing the communication overheads. In general, Between-Graph Replication is more widely used than In-Graph Replication method. Thus, we use Between-Graph Replication for enabling the data parallelism.

2) Deep neural network models and datasets

For our experiments, we use AlexNet [20], CIFAR10 (not a dataset [4], please refer to footnote 1), Network in Network [24], ResNet [14], and SqueezeNet [17] models, which are very widely used in real-world image classification tasks. We use a dataset available in [4], which has 60,000 (50,000 for training and 10,000 for testing) 32X32 color images. For model training, we use a batch size of 32.

3) Computing devices for distributed DNN training

We use three heterogeneous computing devices: high-end, mid-end, and low-end devices. For high-end device, we use a high-performance desktop PC that equips Intel i7-2600 quad-core CPU and Nvidia GTX1050 GPU. For mid-end and low-end devices, we use Nvidia Jetson TX2 (JTX2) and TX1 (JTX1), respectively. Fig. 4 demonstrates our available hardware devices, their specifications, and connection architecture. Three devices are connected via a switching hub to compose a cluster connected via local Ethernet connection.

4) Miscellaneous

For our framework implementation, we use paramiko [8] providing both client and server functionalities by a python implementation of the SSHv2 protocol. In addition, since we need to measure network bandwidth in runtime, we use iperf [7]. Ethtool [6] is also used to forcibly restrict the available network bandwidths for our experiments. To actually measure power consumption from the devices (for linear regression and comparison of our estimated results with the actual measured results), we use HPM-300A power meter [1] and on-board power monitoring circuit for Desktop PC and JTX1/JTX2, respectively. To capture network bandwidth impacts, we carry out our experiments with three different network bandwidths (1Gbps, 100Mbps, and 10Mbps) by

using Ethtool.

B. Training Cluster Model Prediction Results

In this subsection, we evaluate whether or not our framework finds the actual best device cluster configuration. We compare the cluster configuration estimated by our framework and the configuration which actually leads to the best results. Depending on the DNN service providers' preferences, our model can select different configurations. Since we use three computing devices for cluster composition, possible cluster configurations are:

- **Single device training**
  - ***Config 1***: JTX2 GPU
  - ***Config 2***: JTX1 GPU
  - ***Config 3***: Desktop PC GPU
- **Distributed training with 2 devices**
  - ***Config 4***: Parameter Server (Desktop PC CPU) ↔ 2 workers (Desktop PC GPU and JTX2 GPU)
  - ***Config 5***: Parameter Server (Desktop PC CPU) ↔ 2 workers (Desktop PC GPU and JTX1 GPU)
  - ***Config 6***: Parameter Server (JTX2 CPU) ↔ 2 workers (JTX2 GPU and JTX1 GPU)
- **Distributed training with 3 devices**
  - ***Config 7***: Parameter Server (Desktop PC CPU) ↔ 3 workers (Desktop PC GPU, JTX2 GPU, and JTX1 GPU).

For α^{T_i} , $\beta^{T_{i,k}}$, α^{P_i}, and $\beta^{P_{i,k}}$ values used in our experiments, we use the parameters empirically obtained. The α^{T_i} , $\beta^{T_{i,k}}$, α^{P_i}, and $\beta^{P_{i,k}}$ values of 1.0 mean there is no training time or power impact from the parameter communication or network bandwidths. The larger adjusting factor values we have, the larger training time or power impact we exhibit from the parameter configuration or network bandwidths.

1) Framework accuracy

In this subsection, we show the results of our framework accuracy. The accuracy is a ratio between the number of the cases our framework selected the actual best configuration among seven configurations and the total number of the cases. In our experiments, the total number of the cases are 240 (5 models × 4 repetitions × 4 preferences × 3 network bandwidths). To smooth out any discrepancies and fluctuations, we repeat the experiments four times (i.e., four repetitions) for each combination of DNN service provider's preference, network bandwidth, and DNN model. In order to figure the framework accuracy out, we actually measured training time, power, energy, and EDP for all 105 cases (5 models × 3 network bandwidths × 7 configurations). We then compare whether or not the selection from our framework is matched to the configuration that leads to the actual best result.

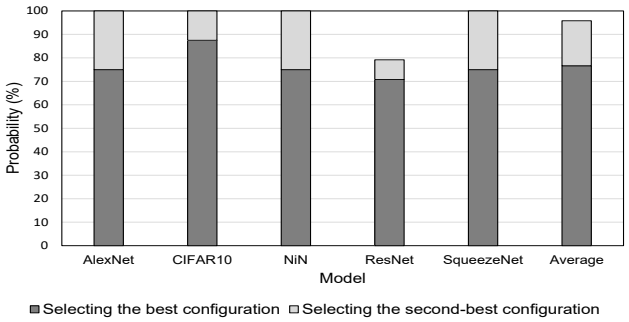


Fig. 5. Cluster selection accuracy of our proposed framework.

As shown in Fig. 5, our training cluster model actually selects the best and second-best configurations by over 95.83% accuracies. Overall, the selection accuracy of our framework for choosing the best device cluster configuration is 76.67% (=184/240). Even in the case where our framework could not choose the best configuration, our framework chooses the second-best configurations for 82.14% (=46/56) of the remaining cases. In the cases of AlexNet, CIFAR10, NiN, ResNet, and SqueezeNet, the accuracies are 75.00%, 87.50%, 75.00%, 70.83%, and 75.00%, respectively. The reason why our framework does not lead to 100% accuracy is mainly because our training time and power estimation method cannot predict the actual training time and power consumption with 100% accuracy. To improve the accuracy, we can perform pre-training with more number of steps (i.e., >100 steps) though it also introduces more training time overhead. We believe there can be an optimal point that balances accuracy and training time overhead though the detailed investigation is our future work.

Fig. 6 summarizes the actual reductions of the training time, power, energy, and EDP when adopting our framework. Our framework leads to training time, power, energy, and EDP reductions by 45.5%, 4.5%, 31.6%, and 27.6%, respectively, compared to the actual second-best configurations (i.e., the second-best configurations derived from the actual measurements, not from our framework), on

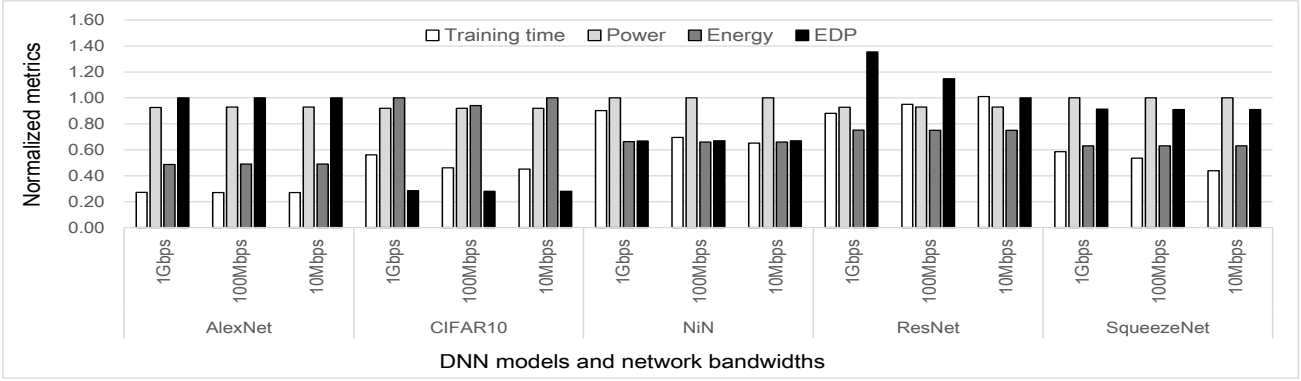


Fig. 6. Normalized training time, power, energy, and EDP results when we perform DNN training with the chosen cluster configuration by our framework compared to the actual (measured, not from our framework) second-best cluster configuration results. The results less than 1.0 mean that our framework picked the actual best configurations. If the results are equal to 1.0, our framework selected the second-best configuration. If the results are higher than 1.0, our framework selected other cluster configuration which is worse than the second-best cluster configuration.

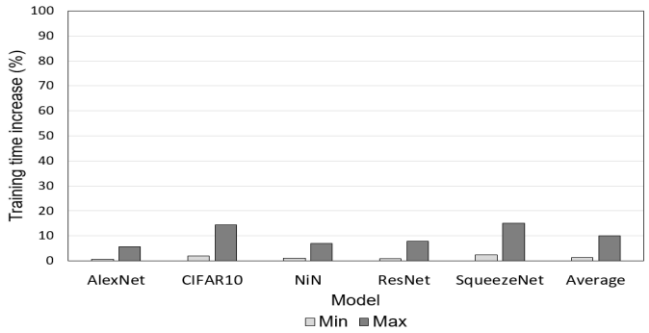


Fig. 7. Training time overhead by our framework.

average. In addition to accurately predicting the best cluster configuration, our framework actually results in better training time, power, energy, or EDP of the distributed DNN training cluster.

2) Training time overhead

Fig. 7 shows the training time overhead incurred by our framework. Our framework actually increases training time by 0.69%~15.09% compared to the case where we run the DNN training with the same configuration without our framework. The main reason for training time increase is that our framework needs to collect the data required for training cluster model and calculate estimated training time, power, energy, and EDP for each possible cluster configuration. In addition, we need to compare and sort T^T_j, P^T_j, E^T_j, and EDP^T_j results to determine which configuration is the best one depending on DNN service providers' preferences. However, our actual training time reduction compared to the case of second-best cluster configuration is more than 45%, sufficiently offsetting the training time overheads.

As shown in Fig. 7, smaller models (e.g., SqueezeNet and CIFAR10) tend to show the larger training time overhead. This is because the absolute time our framework additionally adds to the total training time (e.g., power estimation, training cluster model, cluster selection, etc.) is almost same across the models. It means that the relative training time overhead for the large DNN models (i.e., longer training time) tends to be less than the small DNN models. Though the results shown in Fig. 7 are obtained by running 100,000 steps for training, if we run more steps (i.e., > 100,000 steps), training time overhead will be further reduced.

V. Conclusions

In this paper, we propose a novel framework that can determine the best device cluster configuration with heterogeneous computing devices when running DNN training. Through the pre-training phase, depending on DNN service providers' preferences, our framework finds the best cluster configuration from the given hardware devices and DNN model with our training time, power, energy, and EDP models. After determining the cluster configuration for distributed DNN training, our framework actually performs the DNN training tasks with the selected device cluster configuration. From our experimental results, our framework chooses the best configuration for 76.67% of the total cases. For 82.14% of the remaining cases, our framework also chooses the second-best configurations. As our future work, we will 1) evaluate our framework with more diverse heterogeneous devices, 2) consider diverse network bandwidths, and 3) also consider the data synchronization impact.

Acknowledgement

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2018R1D1A3B07045908). We would also like to thank anonymous reviewers for their helpful feedback.

References

[1] ADPower HPM-300A Power Meter, http://adpower21.com
[2] Amazon EC2, https://aws.amazon.com/
[3] Baidu-allreduce, https://github.com/baidu-research/baidu-allreduce, https://github.com/baidu-research/tensorflow-allreduce.
[4] CIFAR-10 Dataset. https://www.cs.toronto.edu/~kriz/cifar.html
[5] Distributed TensorFlow, https://www.tensorflow.org/deploy/distributed
[6] Ethtool, https://www.kernel.org/pub/software/network/ethtool/
[7] Iperf, https://iperf.fr/
[8] Paramiko, http://www.paramiko.org/
[9] W. Chen et al. "Compressing neural networks with the hashing trick." *International Conference on Machine Learning*, 2015.
[10] J. Dean et al. "Large scale distributed deep networks." *Advances in neural information processing systems*, 2012.
[11] A. E. Eshratifar et al. "JointDNN: an efficient training and inference engine for intelligent mobile cloud computing services." *arXiv preprint arXiv:1801.08618*, 2018.
[12] T. Guo. "Towards efficient deep inference for mobile applications." *arXiv preprint arXiv:1707.04610,* 2017.
[13] J. Hauswald et al. "DjiNN and Tonic: DNN as a service and its implications for future warehouse scale computers." *In ISCA 2015*.
[14] K. He et al. "Deep residual learning for image recognition." *In CVPR 2016.*
[15] K. Hsieh et al. "Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds." In *NSDI* 2017.
[16] L. N. Huynh et al. "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications." *In MobiSys 2017*.
[17] F. N. Iandola et al. "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size." *arXiv preprint arXiv:1602.07360,* 2016.
[18] J. Keuper et al. "Distributed training of deep neural networks: Theoretical and practical limits of parallel scalability."*In MLHPC 2016*.
[19] M. Kim et al. "Enhancing online power estimation accuracy for smartphones," *in IEEE Transactions on Consumer Electronics*, vol. 58, no. 2, pp. 333-339, May 2012.
[20] A. Krizhevsky et al. "Imagenet classification with deep convolutional neural networks." In *Advances in neural information processing systems*, 2012..
[21] N. D. Lane et al. "Can deep learning revolutionize mobile sensing?." In *HotMobile 2015*.
[22] N. D. Lane et al. "DeepX: A software accelerator for low-power deep learning inference on mobile devices." *In IPSN 2016.*
[23] M. Li et al. "Parameter server for distributed machine learning." In *Big Learning NIPS Workshop*. Vol. 6. 2013.
[24] M. Lin et al. "Network in network." *arXiv preprint arXiv:1312.4400,* 2013.
[25] M. Motamedi et al. "Cappuccino: Efficient CNN Inference Software Synthesis for Mobile System-on-Chips." *IEEE Embedded Systems Letters,* 2018.
[26] K. Ota et al. "Deep learning for mobile multimedia: A survey." *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 13.3s : 34, 2017.
[27] B. D. Rouhani et al. "Deep^3: Leveraging three levels of parallelism for efficient deep learning." *In DAC 2017.*
[28] F. Schuiki et al. "A Scalable Near-Memory Architecture for Training Deep Neural Networks on Large In-Memory Datasets." *arXiv preprint arXiv:1803.04783,* 2018.
[29] A. Sergeev et al. "Horovod: fast and easy distributed deep learning in TensorFlow." *arXiv preprint arXiv:1802.05799,* 2018.
[30] P. Sundaramoorthy et al. "HARNet: Towards On-Device Incremental Learning using Deep Ensembles on Constrained Devices" *In EMDL 2018*.
[31] J. Wang et al. "Deep Learning towards Mobile Applications." *In ICDCS 2018*.
[32] S. Wang et al. "When edge meets learning: Adaptive control for resource-constrained distributed machine learning." *arXiv preprint arXiv:1804.05271,* 2018.
[33] M. Xu et al. "Enabling Cooperative Inference of Deep Learning on Wearables and Smartphones." *arXiv preprint arXiv:1712.03073,* 2017.