CHRONICA: A Data-Imbalance-Aware Scheduler for Distributed Deep Learning

Sanha Maeng, Gordon Euhyun Moon and Sungyong Park*
Sogang University, Seoul, Republic of Korea
{msh0117, ehmoon, parksy}@sogang.ac.kr

***Abstract*—One of the major challenges in distributed deep learning is attenuating straggler problem. The straggler increases synchronization latency and significantly inhibits the convergence of deep learning model. We empirically observe that the imbalanced data samples worsen the straggler problem and make the convergence of the deep learning model slower. However, existing approaches such as BOA and EP4DDL have not addressed data imbalance issues while solving the straggler problem. To overcome the straggler and data imbalance problems, we propose CHRONICA, a new data-imbalance-aware scheduler. Based on the size of the data samples and the configuration of each worker, CHRONICA elaborately predicts the training time required for each worker. CHRONICA then provides equivalent training time to each of the workers, alleviating both step- and epoch-level straggler problems. Furthermore, CHRONICA suggests a new parameter synchronization scheme to achieve fast convergence based on the weighted average of the training workload on each worker. Our extensive evaluation using four deep learning models on 32 Amazon EC2 GPU instances showed that the new CHRONICA achieves up to 3.19 times speedup over the state-of-the-art systems.**

***Index Terms*—Distributed deep learning, Straggler, Scheduler, Data imbalance**

I. INTRODUCTION

Over the past few years, the scale of data and the size of deep learning models have increased exponentially. It is obvious that as the size of the dataset increases, the training time of the deep learning model also increases. For example, training GPT-3 [1] with 175 billion parameters and 499 billion tokens requires approximately 288 years using a single NVIDIA Tesla V100 GPU [2]. Hence, it is infeasible to train a large-scale deep learning model with very large datasets on a single machine, and therefore distributed deep learning is becoming increasingly critical. The primary strategy for parallelizing deep learning is data parallelism that distributes a batch of datasets across multiple workers [3]. When a deep learning model is trained in a synchronous data-parallel fashion, the weight parameters must be synchronized across workers at every step via parameter servers. However, the synchronous data-parallel training scheme has an inevitable limitation that it may increase synchronization latency while waiting for a *straggler*, i.e., the slowest worker, which is caused by the node heterogeneity of cluster, resource contention and system failure, etc.

In order to address the straggler problem, several approaches such as utilization of extra backup workers [4], dynamic relaxation of synchronization barrier [5], [6] and node-heterogeneity-aware load balance techniques [7], [8] have been proposed. Most of these studies assume that each data sample has the same size, and the straggler stems from the difference in the performance of the node and the heterogeneity of the network.

*Corresponding author.

Table I: Data imbalance factor (DIF) of widely used video datasets. DIF is defined as the standard deviation of the size of data samples: $\sqrt{\frac{1}{|L|-1}\sum_{\lambda \in L}|\lambda - \mu(L)|^2}$ where L and λ denote the size of data samples and size of each data sample, respectively.

Dataset	#Videos	DIF
UCF101 [9]	13,320	3.73
Charades [10]	9,848	9.44
DAVIS [11]	50	18.36
HACS [12]	504,716	64.13
COIN [13]	11,827	70.02
YouTube-BoundingBoxes [14]	285,412	212.64
Sports-1M [15]	1,133,158	539.75

However, there are innumerable data in which the size of data samples are different from each other. For example, there may be three videos with each duration of 10 seconds, 10 minutes, or one hour. After these videos are converted into tensors, each data sample has a different shape and size depending on the duration of the video. Unfortunately, it is not desirable to simply fix the size of data samples equal. In the above example, if the videos are cropped to the shortest one, i.e., 10 seconds, 99% of data are lost. On the other hand, if the videos are zero-padded to the longest one, i.e., one hour, 61% of the data are dummy and the training time increases to the maximum. Therefore, the size of data samples may not be arbitrarily modified, and previous approaches are hard to mitigate the straggler problem when the data is imbalanced. Surprisingly, we found in our measurements that the degree of data imbalance in real-world datasets reaches up to 500 and tends to increase in large datasets, as shown in Table I.

Without considering the difference between the size of data samples, imbalanced data imposes three main problems. First of all, the straggler problem is intensified. That is, each worker may process a different amount of operations because the sizes of data samples are different. This causes the worker training with the largest data samples to become a straggler. Second, the difference in the size of the data samples increases the peak device memory usage and thus decreases the maximum batch size, which limits the throughput. Finally, the convergence is degraded. In general, the optimizer reduces the gradients computed in each worker by averaging them. Since the amount of data trained in each worker is different, the reduced gradient

979-8-3503-0119-9/23/$31.00 ©2023 IEEE
DOI 10.1109/CCGrid57682.2023.00033

is different from that of a single worker, and the deep learning model converges slower. Due to the aforementioned problems, the training time of deep learning model significantly increases as the degree of data imbalance grows.

In this paper, we propose CHRONICA, a data-imbalance-aware scheduler that significantly minimizes the side effects of imbalanced data. The main idea of CHRONICA is to consider the relative size of each data sample. CHRONICA elaborately predicts the training time required for each data sample according to its size. Using the estimated training time, CHRONICA equalizes the training time of each worker for all steps and epochs. Therefore, CHRONICA effectively mitigates both step- and epoch-level straggler problems. Furthermore, CHRONICA reduces the total number of steps for a deep learning model to converge by adjusting the learning rate for each worker based on the weighted average of the amount of data distributed to each worker.

We developed CHRONICA on top of TensorFlow [16] and evaluated it using four popular deep learning models over 32 Amazon EC2 GPU instances by varying the degree of data imbalance. Our evaluation showed that CHRONICA outperforms the state-of-the-art data-parallel implementations with up to 3.19 times faster runtime while ensuring the convergence of deep learning model. To summarize, the key contributions of our work are as follows:

Data-imbalance-aware data scheduling. CHRONICA mitigates the step-level straggler problem by equalizing the training time of all workers for each step.

Data-imbalance-aware data shuffling. CHRONICA mitigates the epoch-level straggler problem by making the training time for all workers equal for each training epoch.

Learning rate compensation. CHRONICA accelerates convergence by adjusting the learning rate of each worker based on the amount of data trained by each worker.

Heterogeneity independence. CHRONICA adaptively schedules data using the feedback of each worker and thus accelerates training regardless of the node and network heterogeneity of cluster.

II. BACKGROUND AND RELATED WORK

In this section, we provide an overview of data parallelism for training a deep learning model and present related previous studies that address the straggler problem.

A. Background

Deep Learning Model. The deep learning model is a parameterized model based on a deep neural network that takes input data x with a fixed number of parameters w to compute the final output. In the supervised learning fashion, the predicted output from the model $f(x; w)$ and its ground truth y are used for computing the loss value using the loss function $\mathcal{L}$. The goal of deep learning is to find the optimal weight parameters w that minimize the loss value across the training data $x = \{x_1, ..., x_N\}$ (Eq. 1).

$$\underset{w}{\operatorname{argmin}} \mathcal{L}(f(x; w), y) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x_i; w), y_i) \quad (1)$$

The weight parameters can be iteratively optimized by mini-batch stochastic gradient descent (SGD) [17]. For example, w_{t+1} used to compute forward propagation in the $t+1$-th mini-batch is obtained by updating w_t using the gradients g_t from the t-th mini-batch (Eq. 2).

$$\begin{aligned} g_t &\leftarrow \frac{1}{m} \sum_{i=1}^{m} \nabla_{w_t} \mathcal{L}(f(x_i; w_t), y_i) \\ w_{t+1} &\leftarrow w_t - \eta g_t \end{aligned} \quad (2)$$

where m is the size of each mini-batch.

Distributed Deep Learning. Both mini-batch SGD and synchronous SGD [18] can be applied to distributed training using multiple M workers. For each step based on a mini-batch of data samples, M workers simultaneously compute their local gradients $g_{t,j}$ and send the local gradients to the parameter server. After all M gradients are delivered to the parameter server, the weight parameters are updated based on the average of the M gradients (Eq. 3).

$$w_{t+1} \leftarrow w_t - \eta \frac{1}{M} \sum_{j=1}^{M} g_{t,j} \quad (3)$$

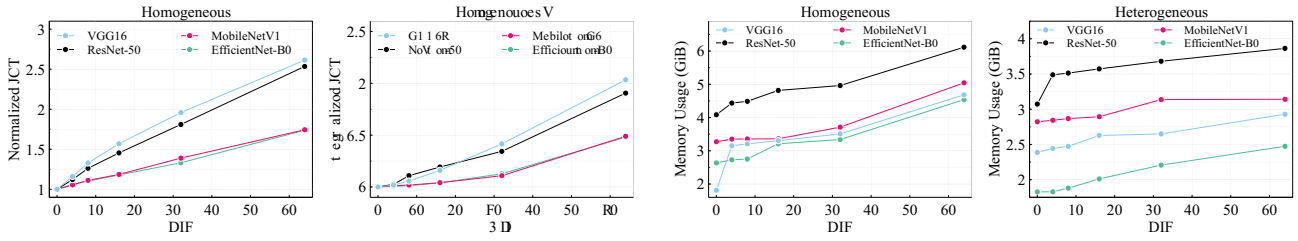
Imbalanced Data. The imbalanced data is a dataset consisting of multiple data samples that are hard to divide and different-sized from each other (e.g., the duration of each video can be varied). After converting data samples into tensors for training, each tensor can maintain a different size of data structure. This indicates that the number of operations required for each tensor can be different from each other. Therefore, considering only the number of data samples while distributing the data samples to multiple workers can cause a huge data imbalance problem. As the training workload of each worker can be different from each other, the straggler problem may lead to higher synchronization latency and slower training time of deep learning models.

B. Related Work

As the straggler problem significantly increases the execution time of distributed deep learning, several algorithms have recently been proposed to solve the straggler problem.

Extra Backup Workers. Chen et al. [4] first adopted extra backup workers to reduce the impact of a straggler. In addition, Xiong et al. [19] introduced another backup worker strategy for dynamically adopting backup workers to further alleviate the straggler problem.

Stale Synchronous Parallel Model. There were other strategies proposed to relax the strict synchronization rule of distributed deep learning. For example, it is possible to balance the synchronous and asynchronous strategies of distributed deep learning. FlexRR proposed by Harlap et al. [5] and the RNA proposed by Yang et al. [6] adopted the stale synchronous parallel approach by dynamically relaxing the synchronization barrier. On the other hand, Ferdinand et al. [20] fixed the duration of the training epoch to restrict the synchronization latency due to the straggler.



(a) Job Completion Time

(b) Peak Device Memory Usage

Figure 1: Normalized JCT and device memory usage with varying the degree of data imbalance (DIF).

Heterogeneity-aware Load Balance Techniques. As the node and network heterogeneity of clusters are known to have a significant impact on the straggler problem, BOA [7] and EP4DDL [8] proposed a load balance technique to dynamically adjust the training workload for each worker in a heterogeneous cluster. BOA is a distributed deep learning framework designed for straggler mitigation on a heterogeneous GPU cluster. Based on its performance model that describes the correlation between local batch size and training time, BOA mitigates the straggler problem by adjusting the number of data samples to be assigned to each worker. On the other hand, EP4DDL adopts a machine learning approach to predict the performance of each worker using GRU networks [21]. Based on the predicted performance, it adjusts the parallelism of each worker to minimize the performance variance between workers and mitigate the straggler problem. Finally, DARL [22] introduced an actor model to fully utilize the resources of dynamic clusters using a deep reinforcement learning-based load balance technique combined with the stale synchronous parallel approach.

Limitations of Prior Work. Most of the existing systems were designed with the assumption that the data is completely balanced. They tried to solve the straggler problem caused by the difference in the performance of node and network. Therefore, their approaches are not appropriate to alleviate the straggler problem given the imbalanced data. Unlike the straggler problem caused by the node and network heterogeneity, which can be significantly mitigated by changing hardware, the straggler problem introduced by imbalanced data can only be resolved by changing the data scheduling policy. This motivates us to design a new data-imbalance-aware scheduler to solve the data imbalance problem.

III. Motivation

In this section, we introduce the problems of imbalanced data. When a deep learning model is trained in a data-parallel fashion with imbalanced data, the following three problems can significantly increase the runtime of deep learning model.

Problem 1: Intensified Straggler. When the size of data samples are different from each other, each worker performs a different number of operations in CUDA kernels. In this case, the difference in training time between workers increases and the straggler problem increases. To analyze the impact of data imbalance on the straggler problem, we trained VGG16 [23], ResNet-50 [24], MobileNetV1 [25] and EfficientNet-B0 [26] by exponentially increasing the DIF of the UCF101 dataset from 0 to 64. In order to evaluate the impact of node heterogeneity on the straggler problem, we conducted experiments on both homogeneous and heterogeneous clusters. The detailed experimental settings are specified in Section VI-A.

Figure 1a shows the increase in job completion time (JCT) according to the degree of data imbalance compared to a baseline that only contains a fixed size of data samples. These results demonstrate that the JCT increases linearly according to the degree of data imbalance, regardless of the node heterogeneity in all workloads. Since both clusters show the same pattern for the increase in JCT, it can be seen that the degree of data imbalance determines the impact of the straggler problem, regardless of the heterogeneity of the node. Furthermore, as shown in Table I, as the degrees of data imbalance in real-world datasets are much higher, the synchronization latency affected by imbalanced data would be much longer.

In addition to the step-level straggler, an epoch-level straggler can also be possible. The size of a large-scale dataset often exceeds the storage capacity of a single machine. For example, the maximum storage capacity of an Amazon EC2 GPU instance (e.g., p4d.24xlarge) is limited to 8TB while the raw size of YouTube-8M [27] is about 360TB. In this case, a worker has to hold only a subset of the dataset required for each training epoch. Even if the step-level straggler problem is resolved, there still exists an epoch-level straggler to process the remaining data samples at the end of each training epoch. Therefore, the epoch-level straggler should also be considered to completely solve the straggler problem.

Problem 2: Limitations on Batch Size. The device memory of GPU is basically not virtualized. Thus, when the data is imbalanced, the amount of device memory usage varies for each step (mini-batch), and the peak device memory usage generally increases. To inspect the increase in peak device memory usage according to the degree of data imbalance, we profiled the device memory usage using a profiler plugin of TensorBoard [28], which provides a callback to trace CUDA API using CUPTI [29].

Figure 1b shows the increase in peak device memory usage according to the degree of data imbalance. The peak device memory usage also increased according to the degree of data imbalance in all workloads, regardless of the node heterogeneity. This leads to limitations on the maximum batch size allowed, hindering the utilization of CUDA and Tensor

cores in GPUs. When the batch size becomes smaller, the number of steps in a training epoch increases. This brings more frequent occurrences of data transmission across workers and data copies between host and device memory. Therefore, it is crucial to make the size of allocated data samples not biased to achieve higher throughput and prevent GPUs from running out of memory.

Problem 3: Convergence Degradation. When the data is imbalanced, each worker may calculate the local gradients with a different size of data samples. This makes the result of Eq. 3 different from that of Eq. 2 and the convergence is degraded. For example, when a deep learning model is trained in a data-parallel fashion using two videos with 10 frames and 600 frames, the gradient for each frame in the former has a weight of 1/10, while the latter has a weight of 1/600. When these gradients are gathered, the weight of the former becomes 1/20 ($1/10 \times 1/2 = 1/20$) and the weight of the latter becomes 1/1200 ($1/600 \times 1/2 = 1/1200$). Therefore, the larger data samples are trained slower and convergence is degraded when the data is imbalanced. To analyze the degradation of convergence according to the degree of data imbalance, we profiled the convergence of ResNet-50.

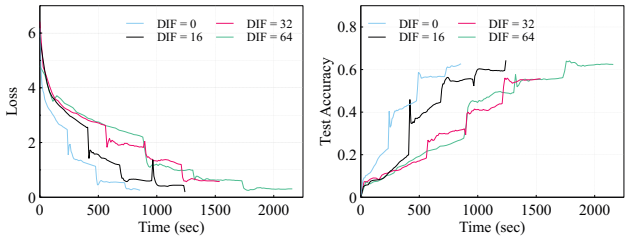


Figure 2: Convergence degradation against the degree of data imbalance.

Figure 2 shows the convergence patterns of ResNet-50 over different degrees of data imbalance. The result demonstrates that convergence is severely hindered as the degree of data imbalance increases. Since the convergence rate decreases, the number of required steps to complete the training increases. Therefore, it is necessary to correct Eq. 3 when the data is imbalanced so that the deep learning model can converge equally as when the data is balanced.

IV. Problem Definition

In this section, we investigate the correlation between the degree of data imbalance and the straggler problem. Within a stable network bandwidth environment, the data transmission cost can be omitted because the size of gradients sent from each worker and the weight parameters are fixed.

Formulation for Straggler Problem. Suppose $T_{i,j}$ is a training time for the j-th worker in the i-th step given a cluster consisting of M workers. Then the straggler problem in the i-th step SP_i can be represented as the difference between the training time of the straggler and the fastest one (Eq. 4).

$$SP_i = max(T_{i,1}, ..., T_{i,M}) - min(T_{i,1}, ..., T_{i,M}) \quad (4)$$

As the straggler stems from the difference between the training time of each worker, it is obvious that the goal of minimizing the straggler problem is to make equal training time for all workers.

Formulation for Training Time. Based on the observation in Section III, $T_{i,j}$ can be described as a linear equation for DIF and appropriate constants, $a_{i,j}$ and $b_{i,j}$ (Eq. 5).

$$T_{i,j} = a_{i,j} DIF + b_{i,j} \quad (5)$$

Since DIF is defined as the standard deviation of the size of data samples $L = \{\lambda_1, ..., \lambda_N\}$ and L is normally distributed, DIF is a generalized linear model [30] of L. Then DIF is approximately linear for $\lambda_{i,j}$, which is the size of data assigned to j-th worker in i-th step. Thus Eq. 5 can be rewritten as a linear equation for $\lambda_{i,j}$ (Eq. 6).

$$T_{i,j} = a'_{i,j} \lambda_{i,j} + b'_{i,j} \quad (6)$$

From Eqs. 4 and 6, the straggler problem is determined by the performance indicators representing the heterogeneity of the node and the network, and the size of the data assigned to each worker (i.e., $a_{i,j}$, $b_{i,j}$ and $\lambda_{i,j}$). In other words, given $T_{i,j}$ and $\lambda_{i,j}$, it is possible to find $a_{i,j}$ and $b_{i,j}$.

V. Design and Implementation

In order to effectively solve the data imbalance problem, there are four major considerations. First of all, the training time for each data sample must be accurately predicted. Second, it is necessary to equalize the training time for each step of all workers to deal with the step-level straggler problem. Third, the training time for each training epoch should be equal for all workers to prevent the epoch-level straggler problem. Finally, updated weights with the imbalanced data calculated by Eq. 3 should be consistent with updated weights using the balanced data to ensure convergence while training a deep learning model.

Our Chronica includes four key components: *runtime prophet*, *data scheduler*, *data shuffler* and *learning rate compensator*. Figure 3 shows the overall workflow of Chronica. (1) The data scheduler solves the step-level straggler problem by using the estimated training time (ETT) of each data sample and assigning the mini-batch for the next step to each worker to make all the workers have the same amount of training time. As shown in Figure 3, in $B-1$-th step, the size of mini-batch assigned by data scheduler to worker 1 ($129 + 105 = 234$) and worker 2 ($156 + 88 = 244$) are similar to each other. (2) Thereafter, all workers compute their local gradients in parallel. When the gradients are reduced, the learning rate is adjusted in the parameter server based on the amount of data trained by each worker. (3) After finishing a step, each worker sends feedback to the runtime prophet. (4) Using the feedback from workers, runtime prophet estimates the training time of each data sample for all workers. Runtime prophet then recalculates the ETT for each data sample. This routine of scheduling-training-feedback-regression is repeated within each training epoch. After completing a training epoch, data

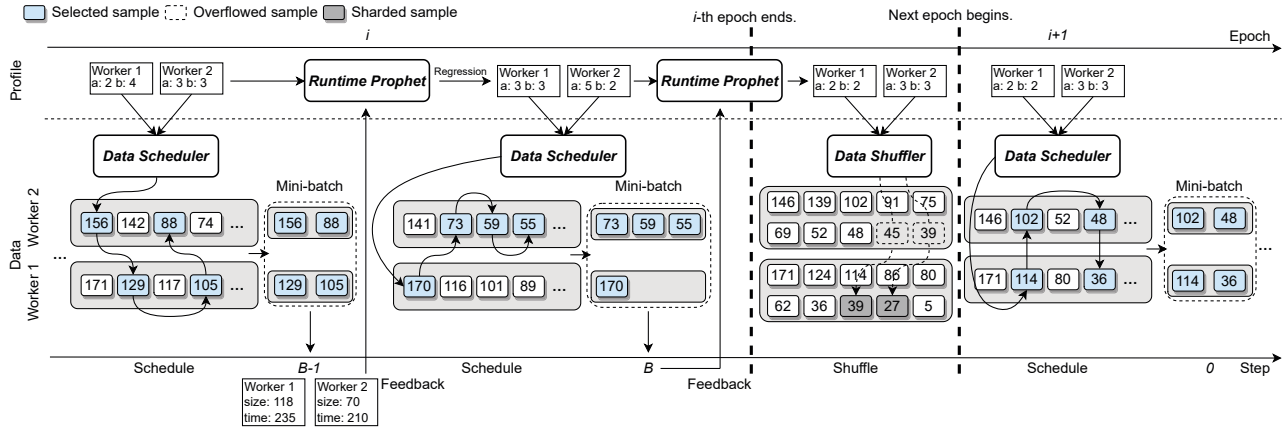


Figure 3: Overall workflow of CHRONICA. The number on each data sample denotes the estimated training time (ETT). B denotes the total number of steps (mini-batches) within an epoch. The *size* and *time* included in each feedback indicate the size of trained data and the actual training time, respectively.

shuffler redistributes the data samples and solves the epoch-level straggler problem by assigning the data samples that require the same amount of training time to each worker. These overall procedures are processed iteratively until the deep learning model converges.

A. Runtime Prophet

Algorithm 1: Performance profiling for worker j

Input: a set of data samples in worker j $items_j$, feedbacks T_j, Λ_j
Output: performance indicators a_j, b_j

$a_j, b_j \leftarrow linear\_regression(T_j, \Lambda_j)$
// Recalculate ETT.
for $items_{i,j} \in items_j$ **do**
  $items_{i,j}.ETT \leftarrow a_j items_{i,j}.size + b_j$
// Sort data samples by ETT.
$sort(items_j)$

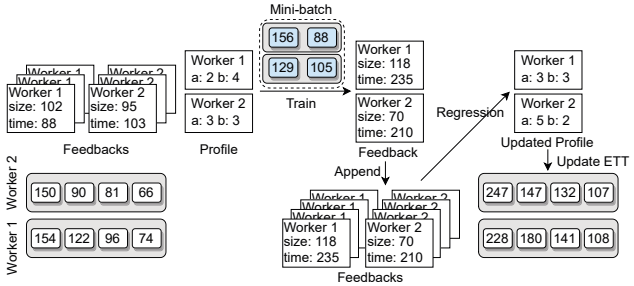


Figure 4: Detailed mechanism of runtime prophet when two workers are participating in the training.

Algorithm 1 shows pseudocode for runtime prophet. As shown in Figure 4, each worker sends feedback, such as the training time and the size of the trained data samples, to the runtime prophet after processing each step. Thereafter, runtime prophet estimates the performance indicators (i.e., $a'_{i,j}$ and $b'_{i,j}$) of each worker. That is, the runtime prophet recalculates $a'_{i,j}$ and $b'_{i,j}$ using linear regression with feedback consisting of $\lambda_{i,j}$ and $T_{i,j}$ in Eq. 6. Then all ETT are recalculated by runtime prophet because $T_{i,j}$ can be calculated when $a'_{i,j}$, $b'_{i,j}$ and $\lambda_{i,j}$ are given. For example, worker 1 sends the feedback that the duration of training time and the size of mini-batch processed in worker 1 were 235 seconds and 118, respectively. Then the performance indicators of worker 1 are updated from (2, 4) to (3, 3) using linear regression. Subsequently, the ETT of the first data sample belonging to worker 2 is updated from 150 seconds to 247 seconds because the size of the sample is 49 ($3 \times 49 + 3 = 150$) and the performance indicators of worker 2 become 5 and 2, respectively ($5 \times 49 + 2 = 247$). After all ETT are updated, data scheduler assigns the next mini-batch using the updated ETT.

B. Data Scheduler

Algorithm 2: CHRONICA scheduling algorithm

Input: a set of all data samples $items$, world size ω, batch size β
Output: scheduled data samples $pivots$

// Initialize $pivots$ to ω empty sets.
$pivots \leftarrow \{\emptyset, ...\}$
for $step \in \{1, ..., \beta\}$ **do**
  if $step = 1$ **then**
    // Initially, select a random pivot.
    $rank, index \leftarrow random(items)$
  else
    $rank, index \leftarrow FFD(items, pivots)$
  $pivot \leftarrow items_{rank,index}$
  $pivots_{rank} \leftarrow pivots_{rank} \cup \{pivot\}$
  $items_{rank} \leftarrow items_{rank} \setminus \{pivot\}$

Algorithm 2 shows pseudocode for data scheduler. Before starting each step, data scheduler assigns the mini-batches to

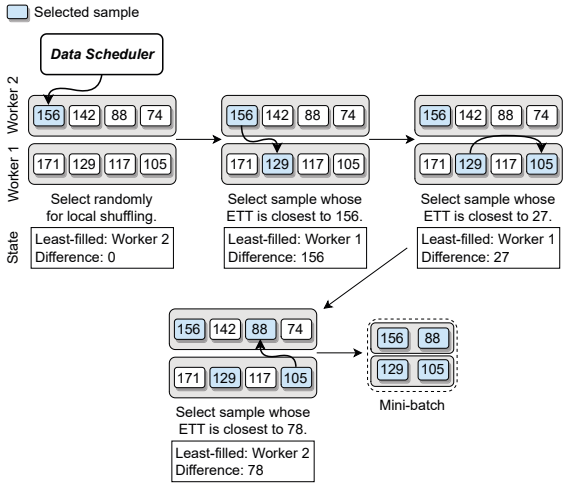


Figure 5: Detailed mechanism of data scheduling sequence when the cluster size is two and the batch size is four.

workers so that all workers have the same amount of ETT. Initially, a data sample from an arbitrary worker is randomly chosen to randomize the training sequence and prevent the overfitting problem. As shown in Figure 5, data scheduler randomly selects a data sample from worker 2, whose ETT is 156 seconds. Subsequently, the data samples are adaptively selected to minimize the difference in ETT between workers. For example, data scheduler selects the second data sample whose ETT is 129 seconds from worker 1 because worker 1 has the least amount of ETT and the data sample closed to 156 seconds is 129 seconds. This procedure is conducted by iterating over the size of mini-batch. Since it is NP-Hard to find a combination of data samples that equalizes the training times of all workers, we adopted first-fit-decreasing bin packing [31] to find a near-optimal solution in polynomial time. Note that if data scheduler controls only the number of data samples to be assigned to each worker, the worker with the largest ETT can be overloaded. For example, assume that the performance of worker 1 is three times faster than that of worker 2 and data scheduler only considers the local batch size. In this case, the first three data samples assigned to worker 1 and only the first data sample assigned to worker 2 will be used. As worker 1 requires 417 seconds ($171 + 129 + 117 = 417$) and worker 2 requires 156 seconds, the required training time is 417 seconds. However, if the data scheduler considers ETT between workers while forming a mini-batch, the training time can be greatly reduced to 244 seconds ($156 + 88 = 244$). Therefore, the data scheduler should consider the size of each data sample as well as the performance of each worker to balance the training workload. After scheduling is completed, the data scheduler sends the indices of selected data samples to each worker's data loader. Each data loader loads the selected data samples into memory using the indices before the next step begins. Since the data scheduler selects the data samples for each step, only the data samples required for each step are constantly loaded within each training epoch.

C. Data Shuffler

Algorithm 3: CHRONICA shuffling algorithm

Input: a set of all data samples $items$, world size ω, a set of performance indicators A, B
Output: a set of redistributed data samples $items$

```
// Initialize Σ to ω zeros.
Σ ← {0, ...}
// Calculate sum of ETT.
for rank ∈ {1, ..., ω} do
    Σ_rank ← sum(items_rank)
μ ← mean(Σ)
// Move overflowed items.
storage ← ∅
for rank ∈ {1, ..., ω} do
    if μ < Σ_rank then
        while μ < Σ_rank do
            index ← search(items_rank, Σ_rank − μ)
            ψ ← items_{rank,index}.ETT
            storage ← storage ∪ {items_{rank,index}}
            items_rank \ {items_{rank,index}}
            Σ_rank ← Σ_rank − ψ
// Redistribute overflowed items.
while 0 < |storage| do
    pivot, rank ← FFD(items, storage)
    items_rank ← items_rank ∪ {pivot}
    storage ← storage \ {pivot}
    // Recalculate ETT.
    pivot.ETT ← A_rank pivot.size + B_rank
```

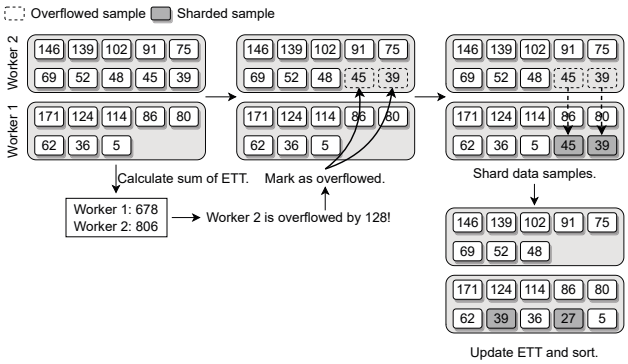


Figure 6: Detailed mechanism of the data shuffling sequence when the data samples are partitioned into two workers.

Algorithm 3 shows pseudocode for data shuffler. After processing each epoch, the data shuffler redistributes the data samples to enable all workers to execute the same amount of training time in the next epoch. As shown in Figure 6, data shuffler first detects the overflowed data samples. Thereafter data shuffler shards the overflowed data samples to workers with the data samples, which have relatively less ETT. Since the performance indicators of workers are different from each

other, the ETT of the sharded data samples is recalculated. For example, data shuffler marks two data samples from worker 2 as overflowed, whose ETT are 45 seconds and 39 seconds, because the difference in ETT between worker 1 and worker 2 is 128 ($806 - 678 = 128$). Then data shuffler shards the two overflowed data samples from worker 2 to worker 1 and ETT of sharded data samples are recalculated from 45 and 39 to 39 and 27, respectively. As a result, the minimum amount of data samples can be transmitted during global shuffling. On the other hand, the data scheduler randomly selects the first pivot to satisfy the local shuffling rather than shuffling the entire data.

D. Learning Rate Compensator

Algorithm 4: CHRONICA gradient reduction algorithm

Input: a set of gradients G, a set of feedbacks Λ, learning rate η
Output: Decrement of weight parameters Δ

```
// Initialize Δ to a zero tensor.
Δ ← g_1 − g_1
Σ ← sum(Λ)
for g, λ ∈ G, Λ do
    Δ ← Δ + (η/Σ) g λ
```

$$w_{t+1} \leftarrow w_t - \eta \frac{1}{\sum_{j=1}^{M} \Lambda_{t,j}} \sum_{j=1}^{M} g_{t,j} \Lambda_{t,j} \tag{7}$$

Algorithm 4 shows pseudocode for learning rate compensator. After the gradients computed from every worker are gathered in the parameter server, the optimizer updates the weight parameters. Instead of using the same learning rate across all workers, CHRONICA compensates the learning rate to consider the size of trained data (Eq. 7). For instance, in the example of Section III, the weight imbalance problem can be resolved using learning rate compensation that makes all the weights 1/610 ($10/610 \times 1/10 = 1/610$, $600/610 \times 1/600 = 1/610$).

E. Implementation Details

We implemented the prototype of CHRONICA based on TensorFlow 2.6.2 using approximately 1K LoC in Python 3.6.9. CHRONICA is available to use as an open source at online[1]. To reduce the communication overhead, we parallelized the feedback phase and combined it with the scheduling phase as a round trip. That is, each worker simultaneously calculates its own performance indicators and the feedback and scheduling results are exchanged in one communication. For the linear regression in runtime prophet, we used `linear_model` module provided by Scikit-learn [32]. Our learning rate compensator described in Section V-D is implemented in `Reduction` class of TensorFlow. Furthermore, we extended `Sequence` class of Keras [33] to provide the users with an interface to the relative size of each data sample.

[1]https://github.com/9rum/chronica

The Python code shown below demonstrates the interface to represent the relative size of each data sample in the extended `Sequence` class.

```
class Sequence(object):
  ...

  @abstractmethod
  def __sizeof__(self, index):
    # Gets relative size of item
    # at position `index`.
    raise NotImplementedError
```

VI. EVALUATION

This section evaluates the performance of CHRONICA by comparing the makespan, convergence, and scalability with various state-of-the-art frameworks such as BOA and EP4DDL that address the mitigation of straggler problem. We trained four different deep learning models with six different imbalanced datasets over homogeneous and heterogeneous clusters.

A. Experimental Setup

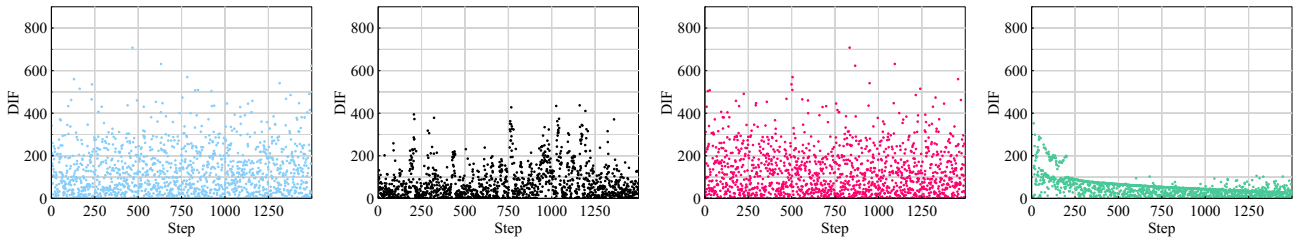
Testbed. We used eight Amazon EC2 GPU instances for our evaluation. To show that our CHRONICA is node and network heterogeneity-independent, we evaluated CHRONICA on both homogeneous and heterogeneous clusters. For the homogeneous cluster, we used eight g4dn.xlarge instances, each equipped with a NVIDIA Tesla T4 GPU. For the heterogeneous cluster, we used four g4dn.xlarge instances with four g3s.xlarge instances, each equipped with 1 NVIDIA Tesla M60 GPU. When evaluating the scalability and overhead of CHRONICA, we used 32 instances for large-scale training. That is, 32 g4dn.xlarge instances were used for the homogeneous cluster and 16 g4dn.xlarge instances with 16 g3s.xlarge instances were used for the heterogeneous cluster. The specifications for the experimental environments are described in Table II.

Table II: Amazon EC2 instances used to evaluate CHRONICA.

	Instance type	
	g4dn.xlarge	g3s.xlarge
CPU	Xeon P-8259L	Xeon E5-2686
Memory (GiB)	16	30.5
GPU	Tesla T4	Tesla M60
Device memory (GiB)	16	8
Network (Gbps)	Up to 25	Up to 10

Workloads. We used four deep learning models consisting of different numbers of weight parameters and operations. The detailed characteristics of each deep learning model are specified in Table III. We trained the deep learning models by increasing the DIF from 0 to 64 with the UCF101 dataset. The size of data samples are normally distributed and the average durations of datasets are equal to 45 frames. As deep learning models are used to solve a classification task, we used categorical cross-entropy loss [34] and Adam optimizer [35] to adjust the parameters. For collective communication, we used NCCL [36] to synchronize the weight parameters.

Systems. We compared CHRONICA with three distributed deep learning systems such as BOA [7], EP4DDL [8] and



(a) TensorFlow (b) BOA (c) EP4DDL (d) CHRONICA

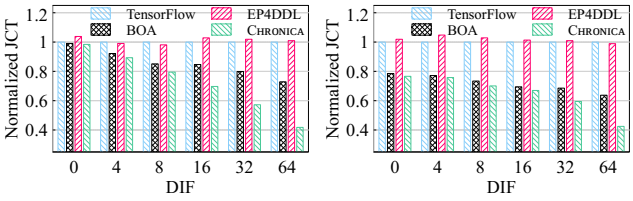
Figure 7: Comparison of the degree of data imbalance over steps.

Table III: Models used to evaluate CHRONICA.

Model	#Params	#FLOPs	Top-1 Accuracy
VGG16	138.4M	15.3B	71.3%
ResNet-50	25.6M	3.78B	74.9%
MobileNetV1	4.3M	0.56B	70.4%
EfficientNet-B0	5.3M	0.4B	77.1%

TensorFlow [16]. Among the previous related works, BOA and EP4DDL were chosen to compare our data-imbalance-aware scheduling approach with the conventional node and network heterogeneity-aware scheduling. BOA considers the runtime minimization problem for each step as a min-max integer programming problem and solves it with an integer linear programming solver, which uses the branch and bound methods [37]. EP4DDL also uses branch and bound methods to find an optimal combination of parallelism when minimizing the performance variance between workers. As CHRONICA is implemented on TensorFlow, the key differences from TensorFlow are the four system designs presented in Sections V-A to V-D. To show the effectiveness of our approach, we also compared CHRONICA with TensorFlow, which randomly selects data samples.

B. Robustness against Data Imbalance



(a) Homogeneous (b) Heterogeneous

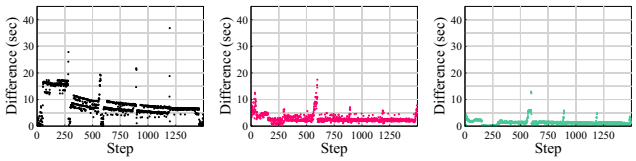
Figure 8: Comparison of JCT over degrees of data imbalance on homogeneous and heterogeneous clusters.

Makespan. Figure 8 and Table IV show the normalized JCT and average runtime for each training epoch according to the degree of data imbalance, respectively. To validate the robustness of CHRONICA against the data imbalance for makespan, we evaluated the JCT of CHRONICA according to the degree of data imbalance using ResNet-50. As the degree of data imbalance increased, the JCT of CHRONICA was significantly reduced compared to other systems in both clusters by up to 59%.

To verify the reason for the performance improvement of CHRONICA, we analyzed it in two aspects: predictive accuracy and degree of data imbalance. To evaluate the prediction ability of the runtime prophet, we measured the difference between the estimated training time and the actual training time for each step. As shown in Figure 9, the differences between the estimated and actual training time of CHRONICA are much less than those of BOA and EP4DDL. Although EP4DDL is unaware of data imbalance, it also showed a relatively small difference between the estimated and actual training time. However, EP4DDL consistently showed JCT similar to TensorFlow because it only adjusts the number of CPU threads even though our environment consists of multiple GPUs.

Table IV: Comparison of training time per epoch with different degrees of data imbalance in a homogeneous cluster (in seconds).

DIF	System			
	TensorFlow	BOA	EP4DDL	CHRONICA
0	234	232	235	231
4	263	242	260	235
8	295	251	289	236
16	334	283	338	237
32	423	343	431	243
64	578	426	584	247

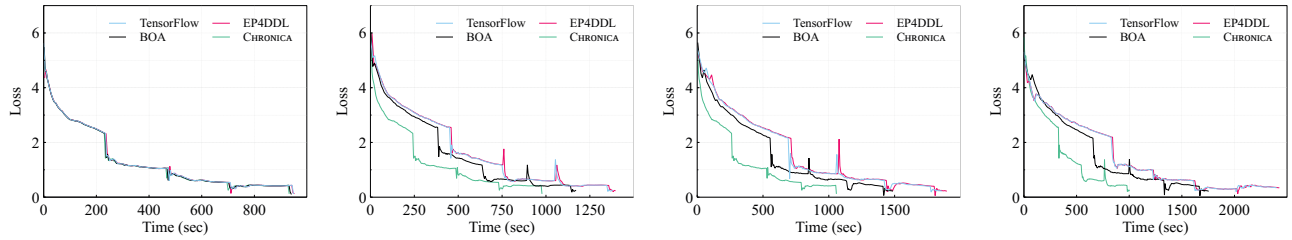


(a) BOA (b) EP4DDL (c) CHRONICA

Figure 9: Difference in seconds between the estimated training time and the actual training time over steps.

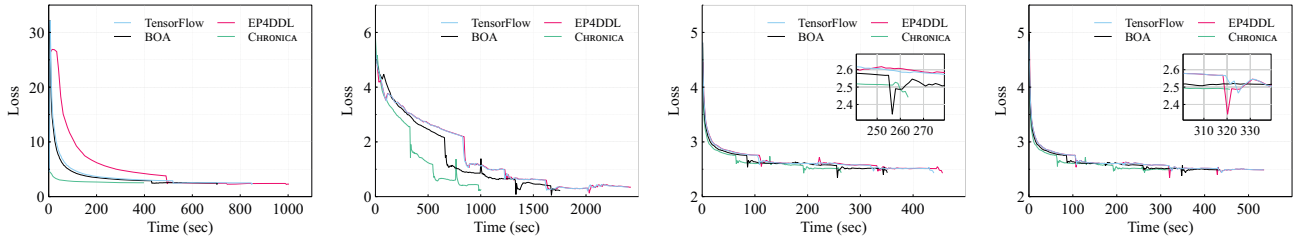
The degree of data imbalance of each system in the homogeneous cluster is shown in Figure 7. In order to verify how well CHRONICA mitigates the data imbalance problem, we measured the degree of data imbalance for each step. The degree of data imbalance for each step was calculated as the standard deviation of the size of mini-batch allocated to each worker. It is obvious that the degree of data imbalance in CHRONICA is definitely low compared to those of other systems. This result validates that CHRONICA effectively controls the imbalanced data and provides a balanced training workload for each worker. Although BOA did not consider the data imbalance, it showed a relatively low degree of data imbalance. Therefore, BOA produces a better JCT than EP4DDL.

Convergence. Figure 10 shows the convergence rates accord-



(a) DIF = 0 (b) DIF = 16 (c) DIF = 32 (d) DIF = 64

Figure 10: Comparison of convergence over training time with different degrees of data imbalance.

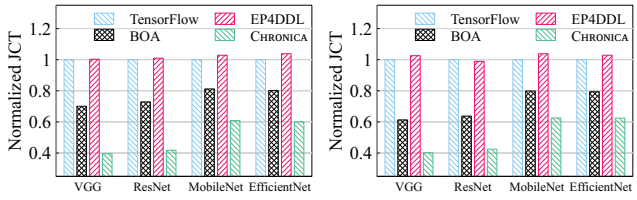


(a) VGG16 (b) ResNet-50 (c) MobileNetV1 (d) EfficientNet-B0

Figure 11: Comparison of convergence over training time on four different workloads.

ing to the degree of data imbalance. For a fair evaluation, we saved randomly initialized model parameters using TensorFlow's model checkpoint function [38] and loaded the checkpoint before the training of each system begins. This ensures that all compared systems were trained under the same conditions. As the degree of data imbalance grew, the convergences of other systems were severely degraded while CHRONICA consistently maintained a similar convergence rate. This validates that CHRONICA not only reduces the training time for each step, but also ensures the convergence of the deep learning model.

C. Robustness against Workload



(a) Homogeneous (b) Heterogeneous

Figure 12: Comparison of JCT over workloads on homogeneous and heterogeneous clusters.

Table V: Comparison of training time per epoch over four different workloads in a homogeneous cluster (in seconds).

Model	System			
	TensorFlow	BOA	EP4DDL	CHRONICA
VGG16	628	440	630	249
ResNet-50	578	426	584	247
MobileNetV1	139	113	143	85
EfficientNet-B0	143	115	148	86

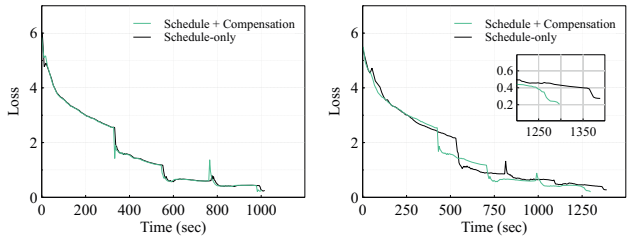
Makespan. Figure 12 and Table V show the normalized JCT and average runtime for each training epoch by workload, respectively. In order to evaluate the generality of our approach, we evaluated CHRONICA using four deep learning models specified in Table III. When measuring the makespan, we fixed DIF to 64 and used the average JCT over five different executions for the indicator of makespan. Even if the workload changes, CHRONICA consistently showed the fastest JCT. The larger the increase in JCT in Section III, the greater the effect of the acceleration of training. Especially for VGG16, which showed the largest increase in JCT, the training time was accelerated by 61%.

Convergence. The convergence rates for each workload are shown in Figure 11. We evaluated the workload-robustness of CHRONICA by measuring the convergence rate for each workload. When evaluating the convergence rate against the workload, DIF was fixed to 64 and the same techniques described in Section VI-B were applied for a fair evaluation. Although the degree of acceleration for convergence was different depending on the workload, CHRONICA consistently showed the fastest convergence for all workloads.

D. Effectiveness of Learning Rate Compensation

To validate the effectiveness of learning rate compensation in CHRONICA, we compared the convergence rate of ResNet-50 with and without learning rate compensation when DIF was set to 64. As shown in Figure 13, in a homogeneous cluster, data scheduler removes most of the degree of data imbalance and thus the effect of learning rate compensation is negligible (approximately 1%). On the other hand, in a heterogeneous cluster, the convergence was additionally accelerated by 7%. The degree of data imbalance in a heterogeneous cluster is generally higher than that in a homogeneous cluster because

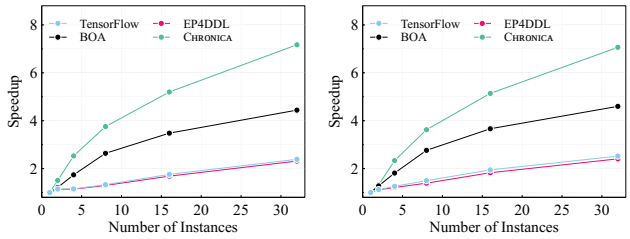
the difference in performance between workers is larger. This makes learning rate compensation more effective in a heterogeneous cluster.



(a) Homogeneous (b) Heterogeneous

Figure 13: Comparison of convergence over the presence of learning rate compensation.

E. Scalability



(a) Homogeneous (b) Heterogeneous

Figure 14: Comparison of scalability in homogeneous and heterogeneous clusters.

Figure 14 shows the scalability of the systems for ResNet-50, where we set DIF to 64 and increased the cluster size from 1 to 32. The local batch size for each worker was set to four and the global batch size was set to the product of the number of workers and the local batch size, i.e., $4\times$cluster size. As the cluster size grew, CHRONICA achieved better scalability than all the other three systems. Especially when the cluster size was set to 32, CHRONICA showed $3.19\times$ faster runtime than EP4DDL in the heterogeneous cluster. When the synchronization latency increases due to straggler, the scalability is significantly constrained. For example, if there are two workers and four data samples where each of the two workers contains two data samples with ETT of 1 second and 100 seconds, then it takes 101 seconds to complete the work. However, when these four data samples are distributed to four workers, it takes 100 seconds to complete, and there will be only $1.01\times$ speedup compared to running with two workers, indicating that it obtains very poor scalability. As a result, systems with a lower degree of data imbalance achieved better scalability.

F. Overhead

Table VI shows the breakdown of elapsed time in seconds for running CHRONICA. To confirm whether the overhead of CHRONICA is reasonable, we measured the time taken for each operation in CHRONICA on 8 and 32 instances using MobileNetV1. In addition to feedback, scheduling, and shuffling operations, there is a *lock* operation included in CHRONICA that occurs while waiting for all workers to complete training and access shared resources. As the size of the cluster increases, the number of required steps within each training epoch decreases linearly and the number of scheduling procedures decreases. The execution of feedback operation is parallelized and the complexities of scheduling and shuffling algorithms are sublinear. Since these operations only transmit a few bytes of data and most of the communication occurs while transmitting the weight parameters and gradients, these operations require an extremely small amount of time. As a result, the overhead of CHRONICA accounts for less than 3% of the total runtime on 8 and 32 instances.

Table VI: Breakdown of elapsed time for running CHRONICA on different cluster size (in seconds).

Operation	Number of Instances	
	8	32
Feedback	1.57 (0.60%)	0.48 (0.34%)
Schedule	0.06 (0.02%)	0.03 (0.02%)
Shuffle	0.02 (0.01%)	0.02 (0.02%)
Lock	6.23 (2.37%)	1.75 (1.25%)
Training	255.41 (97.01%)	137.73 (98.38%)
Total	263.28 (100%)	140.00 (100%)

VII. CONCLUSION

We explored the side effects of data imbalance on distributed training of deep learning model. The imbalanced data significantly exacerbates the straggler problem and degrades the convergence of deep learning model. In order to solve the data imbalance problem, we proposed CHRONICA, a new data-imbalance-aware scheduler. CHRONICA significantly mitigates the straggler problem and convergence degradation given the imbalanced data. Our extensive evaluation over 32 Amazon EC2 GPU instances showed that CHRONICA accelerates training at most 3.19 times faster than the state-of-the-art systems with better scalability while ensuring the convergence of deep learning model.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their insightful comments on this work. This research was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No. NRF-2021R1A2C2014386).

REFERENCES

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[2] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, *et al.*, "Efficient large-scale language model training on gpu clusters using megatron-lm," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2021.

[3] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–43, 2019.

[4] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," *arXiv preprint arXiv:1604.00981*, 2016.

[5] A. Harlap, H. Cui, W. Dai, J. Wei, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing, "Addressing the straggler problem for iterative convergent parallel ml," in *Proceedings of the seventh ACM symposium on cloud computing*, pp. 98–111, 2016.

[6] D. Yang, W. Rang, and D. Cheng, "Mitigating stragglers in the decentralized training on heterogeneous clusters," in *Proceedings of the 21st International Middleware Conference*, pp. 386–399, 2020.

[7] E. Yang, D.-K. Kang, and C.-H. Youn, "Boa: batch orchestration algorithm for straggler mitigation of distributed dl training in heterogeneous gpu cluster," *The Journal of Supercomputing*, vol. 76, no. 1, pp. 47–67, 2020.

[8] Z. Ji, X. Zhang, J. Li, J. Wei, and Z. Wei, "Ep4ddl: addressing straggler problem in heterogeneous distributed deep learning," *The Journal of Supercomputing*, pp. 1–18, 2022.

[9] K. Soomro, A. R. Zamir, and M. Shah, "Ucf101: A dataset of 101 human actions classes from videos in the wild," *arXiv preprint arXiv:1212.0402*, 2012.

[10] G. A. Sigurdsson, G. Varol, X. Wang, A. Farhadi, I. Laptev, and A. Gupta, "Hollywood in homes: Crowdsourcing data collection for activity understanding," in *European Conference on Computer Vision*, pp. 510–526, Springer, 2016.

[11] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. Van Gool, M. Gross, and A. Sorkine-Hornung, "A benchmark dataset and evaluation methodology for video object segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 724–732, 2016.

[12] H. Zhao, A. Torralba, L. Torresani, and Z. Yan, "Hacs: Human action clips and segments dataset for recognition and temporal localization," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 8668–8678, 2019.

[13] Y. Tang, D. Ding, Y. Rao, Y. Zheng, D. Zhang, L. Zhao, J. Lu, and J. Zhou, "Coin: A large-scale dataset for comprehensive instructional video analysis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1207–1216, 2019.

[14] E. Real, J. Shlens, S. Mazzocchi, X. Pan, and V. Vanhoucke, "Youtube-boundingboxes: A large high-precision human-annotated data set for object detection in video," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5296–5305, 2017.

[15] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

[16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: a system for large-scale machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 265–283, 2016.

[17] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[18] M. Langer, Z. He, W. Rahayu, and Y. Xue, "Distributed training of deep learning models: A taxonomic perspective," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 12, pp. 2802–2818, 2020.

[19] G. Xiong, G. Yan, R. Singh, and J. Li, "Straggler-resilient distributed machine learning with dynamic backup workers," *arXiv preprint arXiv:2102.06280*, 2021.

[20] N. Ferdinand, B. Gharachorloo, and S. C. Draper, "Anytime exploitation of stragglers in synchronous stochastic gradient descent," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 141–146, IEEE, 2017.

[21] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[22] H. Lu and K. Wang, "Distributed machine learning based mitigating straggler in big data environment," in *ICC 2021-IEEE International Conference on Communications*, pp. 1–6, IEEE, 2021.

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

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[25] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[26] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, pp. 6105–6114, PMLR, 2019.

[27] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan, "Youtube-8m: A large-scale video classification benchmark," *arXiv preprint arXiv:1609.08675*, 2016.

[28] G. Developers, "Optimize tensorflow performance using the profiler," 2021.

[29] NVIDIA, "Nvidia cuda profiling tools interface (cupti) - cuda toolkit," 2022.

[30] J. A. Nelder and R. W. Wedderburn, "Generalized linear models," *Journal of the Royal Statistical Society: Series A (General)*, vol. 135, no. 3, pp. 370–384, 1972.

[31] D. S. Johnson, *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[33] F. Chollet *et al.*, "Keras: The python deep learning library," *Astrophysics source code library*, pp. ascl–1806, 2018.

[34] Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," *Advances in neural information processing systems*, vol. 31, 2018.

[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[36] NVIDIA, "Nvidia collective communication library," 2020.

[37] S. Boyd and J. Mattingley, "Branch and bound methods," *Notes for EE364b, Stanford University*, vol. 2006, p. 07, 2007.

[38] G. Developers, "Training checkpoints," 2022.