Performance Analysis of Distributed Deep Learning Frameworks in a Multi-GPU Environment

Tulasi Kavarakuntla, Liangxiu Han, Huw Lloyd, MIEEE, Annabel Latham, SMIEEE, Samson B. Akintoye
Dept. of Computing and Mathematics, Manchester Metropolitan University, Manchester, UK
Tulasi.Kavarakuntla@stu.mmu.ac.uk, {L.Han, Huw.Lloyd, A.Latham, S.Akintoye}@mmu.ac.uk

***Abstract*—Deep Learning frameworks, such as TensorFlow, MXNet, Chainer, provide many basic building blocks for designing effective neural network models for various applications (e.g. computer vision, speech recognition, natural language processing). However, run-time performance of these deep learning frameworks varies significantly even when training identical deep network models on the same GPUs. This study presents an experimental analysis and performance model for assessing deep learning models (Convolutional Neural Networks (CNNs), Multilayer Perceptrons (MLP), Autoencoder) on three frameworks: TensorFlow, MXNet, and Chainer, in a multi-GPU environment. We analyse factors that influence these frameworks' performance by computing the running time of each framework in our proposed model, taking load imbalance factor into account. The evaluation results highlight significiant differences in the scalability of the frameworks, and the importance of load balance in parallel distributed deep learning.**

***Index Terms*—Deep Learning; GPUs; SGD and synchronous SGD; Deep Learning Frameworks, Load imbalance factor.**

I. Introduction

With the available computational power such as GPU, Deep learning (DL) [1], as a subset of machine learning based on artificial neural networks, has attracted much attention due to its nature in discovering correlation structure in data in an unsupervised fashion, which has led to its popularity among the many domains such as image classification, speech recognition, computer vision and natural language processing. However, training a deep learning model is a challenging task due to many constraints such as large data instances and high dimensionality, model complexity and inference time, and model selections. For instance, there are a million parameters defining a deep learning model, which requires large amounts of data to learn from it and is a computationally intensive process. Especially, when the data size and the deep learning models become larger and more complicated, training a model within a considerate period usually demands more hardware memory and computing power such as parallel and distributed computing [2] [3] [4] including data parallelism [5], model parallelism [6], pipeline parallelism [7] and hybrid parallelism [8]. Recently, various distributed deep learning frameworks such as Caffe-MPI [9], TensorFlow [10], MXNet [11], Chainer [12], CNTK [13]) have been proposed, which provide basic building blocks for designing effective neural network models for targeted applications. However, run-time performance of these deep learning frameworks varies significantly even when training identical deep network models on the same GPUs. Existing works have investigated deep learning performance modelling on distributed systems [14], asynchronous stochastic gradient descent performance prediction [15], and analytical models for estimating the optimum utilisation of GPU resources [16] for deep learning, and performance evaluation and benchmarking of deep learning frameworks on GPUs [17] [18]. In this study, we extend the work presented in [18] to analyse the performance of three distributed deep learning frameworks(TensorFlow, MXNet and Chainer) with Convolutional Neural Networks (CNNs), Multilayer perceptrons (MLP) and Autoencoder within a multi-GPU environment. Our contributions include:

- Different from the existing works, by taking account of load imbalance factor and mini-batch time (time taken to divide mini-batches), we build a performance model based on synchronous stochastic gradient descent (S-SGD) to analyse the execution time performance of deep learning frameworks in a multi-GPU environment, and evaluate the model using three deep learning models (Convolutional Neural Networks, Autoencoder and Multilayer Perceptron), each implemented in three frameworks (MXNet, Chainer and Tensorflow) respectively.
- Using our experimental data, we analyze the effect of load imbalance on the scalability of deep learning models, concluding that it is in important contribution to parallel inefficiency.

The remainder of the paper is organised as follows. Section II reviews relevant related work. In section III we develop our performance model based on S-SGD. Section IV presents experiments and analysis on a range of DNN frameworks and models. In Section V, we summarize our conclusions and discuss future work.

II. Related Work

This section presents an overview of the existing performance models used in distributed systems. A performance model [19] [20] provides insight into an implementation's behaviour in future execution contexts and is used to evaluate development-stage design and infrastructure investment decisions.

Yan *et al.* [14] developed performance modelling for exploring the design space and to identify effective system configurations that reduces elapsed time between iterations on the training data. The results shown that error rates of less than

25% enable them to define and differentiate between desirable and undesirable system parameter combinations.

Oyama et al. [15] developed a performance model for SPRINT, an Asychronous SGD based deep learning system based on mini-batch SGD, running on GPU. The model included the key parameters in asynchronous SGD training are mini-batch size and gradient staleness. There were no other parallelism types in the performance model, as weights were directly synchronised between GPUs. The findings showed that the ASGD deep learning system SPRINT's performance model effectively predicted sweeping time, mini-batch size, staleness, and probability distributions of the fundamental parameters on two GPU-based supercomputers. The prediction model was then used to assess deep learning's scalability for future hardware architectures.

Lee *et al.* [21] used CNN models to perform image recognition, implementing AlexNet on five different frameworks, which include CNTK [13], Caffe-MPI [9], Theano [22], Torch [23], and TensorFlow [10], and assessed the GPU performance characteristics. Each framework includes a variety of convolution algorithms. They performed a comparison based on the performance of some convolution algorithms such as the Winograd method, GEMM, FFT and direct convolution. Scaling DNNs in a single node with multiple GPUs is essential. As a result, they examined the factors that contributed to their overhead when parallelizing the data. The results indicated that by simply altering the framework's options, the training speed could be increased by a factor of two without modifying any source code.

Qi *et al.* [24] proposed a performance model known as Paleo, which combine parallelization strategies, communication schemes, and network architecture to forecast the deep neural networks training performance. In training the AlexNet model, the results showed that hybrid parallelism outperformed data parallelism. Paleo has been compared in several communication schemes, including OneToAll, Tree AllReduce, and Butterfly AllReduce.

Yufei *et al.* [25] established a performance model for estimating resource consumption and performance efficiency on FPGAs, that was applied to the design phase to find and explore optimal design options. The authors mainly focused on DRAM efficiency, response time, and PE utilization. The evaluation results showed that the model's predictions are quite closely match (within a factor of three) the actual test results obtained on field programmable gate arrays.

Andre Viebke [26] investigated performance prediction accuracy using three alternative CNN models on an Intel Xeon Phi Processor. These two parameterized performance models estimated training convolutional neural networks' execution time. The first performance model used minimal parameter estimate approaches. The second model estimated sequential work by measuring forward and backward propagation. The results showed that the first model's average performance prediction accuracy was 4% higher than the second model.

Shi *et al.* [18] created performance models to assess the performance of a variety of distributed deep learning frameworks (such as CNTK or MXnet) with Alexnet, GoogleNet and ResNet models on GPU computing platforms. They developed models for SGD in single-GPU, multi-GPU, and distributed cluster systems. Through experimental analysis, identified overheads and limitations that could be further optimized in terms of system configuration.

TABLE I: Notation used in this paper (after [18])

Symbol	Description
N_g	Number of total GPUs
t_{iter}	An Iteration time
t_{io}	I/O time of an iteration
t_{h2d}	Communication time between CPU and GPU of an Iteration
t_{md}	Time for dividing batches into mini-batches
t_f	Forward operation time of an iteration
t_b	Backward operation time of an iteration
$t_{f_i}^{(l)}$	Time taken by i^{th} GPU for l^{th} layer in forward operation
$t_{b_i}^{(l)}$	Time taken by i^{th} GPU for l^{th} layer in backward operation
t_{c_i}	Time taken by i^{th} GPU for computing gradients aggregation
t_u	Model update time of an iteration
t_c	Gradients aggregation time of an iteration

In this work, we develop a performance model based on that of [18], and evaluate it in the context of a single node, multi-GPU system. Different from the existing works, we refine some parts of the model by further dividing the timings for stages of the training, and also consider the effect of load imbalance on the performance. We analyse the running performance of Convolutional Neural Network, Multilayer Perceptron and Autoencoder models on three different frameworks respectively.

III. The Proposed Performance Model

A. Preliminaries

For convenience and easy reference, the notations used here follow the notations in [18].

1) Mini-batch SGD: Let consider an L-layered DNN model, which is trained iteratively on a GPU using mini-batch SGD. Each iteration consists of five steps: 1) Fetch a training data mini batch from either internal or external disk; 2)Transfer the training data from CPU memory to GPU memory through PCIe ; 3) Perform feed-forward calculations layer by layer by using GPU kernels; 4) Use backward propagation for gradients computation from Layer L to Layer 1; 5) Calculate average gradients and update the model.

An iteration time can be expressed as:

$$t_{iter} = t_{io} + t_{h2d} + t_f + t_b + t_u = t_{io} + t_{h2d} + \sum_{i=1}^{l} t_f^l + \sum_{i=1}^{l} t_b^l + t_u \quad (1)$$

2) S-SGD using multiple GPUs: In comparison with the SGD, S-SGD consists of six steps. The 1st - 4th steps are similar to the SGD. The 5th step is gradient aggregation, and

the sixth step is updating the model. The iteration time of the S-SGD implementation can be represented as:

$$t_{iter} = t_{io} + t_{h2d} + \sum_{i=1}^{l} t_f^l + \sum_{i=1}^{l} t_b^l + \sum_{i=1}^{l} t_c^l + t_u \quad (2)$$

In the single GPU environment, $\sum_{i=1}^{l} t_c^l = 0$.

B. The Proposed Performance Model based on S-SGD

In this work, different from the existing works [18], we build a performance model of S-SGD by inclusion of two new parameters: time taken to divide the batch into mini-batches and maximum time taken by GPU, taking load imbalance factor into account.

Assume that a machine contains k GPUs. Given the model to be trained, each GPU will individually keep a complete set of model parameters, although parameter values are identical and synchronised across GPUs. For an example, Figure 1 describes the workflow of the performance model when $k = 4$. In general, the model works as discussed in section III-A2 using multiple GPUs. Thus, we develop our proposed performance model of training DNNs with S-SGD in the TensorFlow, MXNet and Chainer frameworks.

Here, S-SGD executes feed-forward and backward propagation simultaneously on each GPU with the same model and distinct training datasets. We consider the time taken for dividing each batch into mini-batches and we also consider the maximum time taken by each GPU in forward processing. By substituting these two parameters in our modelling function, the iteration time t_{iter} for the S-SGD implementation can be represented as follows:

$$t_{iter} = t_{io} + t_{h2d} + t_{md} + max_{i\epsilon(1,n)}(\sum_{i=1}^{l} t_f^l + \sum_{i=1}^{l} t_b^l + \sum_{i=1}^{l} t_c^l) + t_u \quad (3)$$

In the single GPU environment, $\sum_{i=1}^{l} t_{c_i}^l = 0$. The time of an iteration can be written as:

$$t_{iter} = t_{io} + t_{h2d} + t_{md} + \sum_{i=1}^{l} t_f^l + \sum_{i=L}^{1} t_b^l + t_u \quad (4)$$

We now consider the effects of optimization strategies, which make use of task parallelism, which are found in the existing deep learning frameworks. We can notice two possible optimization opportunities. Initially, we can parallelize data reading tasks with the computing tasks, which effectively hide the time cost of disk I/O. Secondly, gradient communication tasks with the back propagation computing tasks can be parallelized. In the case of overlapping I/O with computation, the first step is frequently processed with multiple threads, allowing the I/O time of a new iteration to overlap with the computing time of the preceding iteration. In such a manner, computing in the following iteration can begin immediately after the model is completed. Thus, the average iteration time of pipelined SGD is calculated as;

$$t_{iter} = \max(t_f + t_b + t_u, t_{io} + t_{h2d} + t_{md}) \quad (5)$$

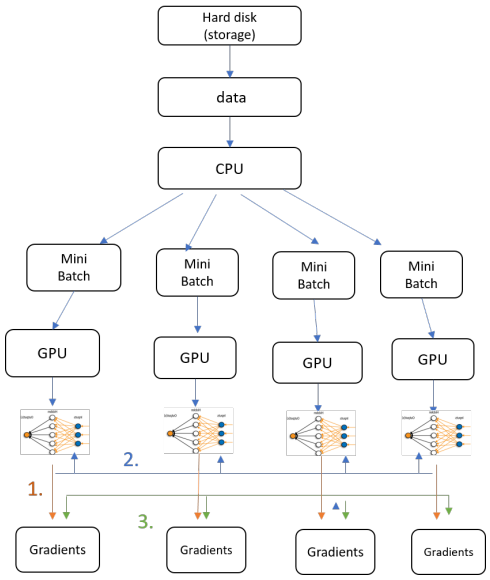


Fig. 1: Workflow of the model: (1) loss and gradient computation, (2) gradient aggregation, (3) parameter update

In a scenario where the gradient communication overlaps with the computation, the gradient communication could be re-programmed to run concurrently with the backpropagation steps. Therefore, the overheads of I/O and gradient communications need to be reduced to achieve good performance and scalability, Let t'_{iter} and t'_{io} represent the iteration time and I/O times respectively on N_g GPUs. The speedup of using N_g GPUs is the given by

$$S = N_g \frac{t_{iter}}{t'_{iter}} \quad (6)$$

Accounting for the optimizations described above, we can now write this as:

$$S = N_g \frac{\max\{t_{io} + t_{h2d}, t_f + t_b\}}{\max\{t'_{io} + t_{h2d}, t_f + t_b + t_c\}} \quad (7)$$

IV. Experiments

In this section, we describe our experimental environment and present the results of experiments to investigate the running time performance of DNN models and frameworks, and how communication tasks affect the scalability of S-SGD.

A. Experimental Setup

Initially, we define the hardware specification conducted in our experiments. We used a single node with three GPUs. GPU@ GEFORCE RTX 2080, CPU@ 2.60 GHZ 2.81GHZ and Memory (RAM)- 16.0GB. Software used for the experimentation are TensorFlow version-2.1.0, MXnet version -1.6.0, Chainer version-7.4.0, python version-3.6.9, CUDA version-10.2. and operating system- Linux. We used Nsight profiler [27] to find the running time performance of GPU activity.

Furthermore, we measure the time duration of an iteration for processing a mini-batch of input data to evaluate the

execution performance. Here we choose three Neural Network models i.e., the Multilayer Perceptron model (MLP), Convolutional Neural Network model (CNN) and Autoencoder model. The models are trained on the MNIST dataset on three frameworks i.e., TensorFlow [10], MXNet [11] and Chainer [12] by applying distributed and parallel training. The MNIST dataset contains 70,000 images of ten handwritten digits and is divided into training and test datasets. The training dataset has 60,000 images, while the test dataset contains 10,000 images. All the two datasets have 10 classes, the 10 numerical digits. In our experiment, we run two epochs and discard the result of the first epoch, since this will include some setup which is not representative of the average training load over a long run. We recorded each iteration time and average these over the second epoch, calculating the mean and standard deviation of each time.

B. Performance Metrics

The speedup and load imbalance factor are selected as performance metrics for run time evaluation on three different frameworks. The speedup is defined in Equation 7. The load imbalance factor for a set of parallel process which execute in times $t_1 \dots t_N$ is defined as

$$LIF = \frac{\max_{i\in[1\dots N]} t_i}{(1/N)\sum_{i=1}^{N} t_i}. \quad (8)$$

$LIF = 1$ corresponds to a perfectly balanced load, whereas for imbalanced loads, $LIF > 1$.

The experimental evaluation is focused on two goals below.

- The first experimental goal is to investigate running time performance of each model using different frameworks in a multi-GPU environment.
- The second experimental goal is to investigate how load imbalance factor of each model under different computing nodes/GPU affects the computing efficiency.

C. Results and Analysis

This section illustrates the running performance followed with analysis based on the performance modelling of TensorFlow, MXNet and Chainer in training CNN, MLP and Autoencoder models in a multiple GPU environment.

1) Single GPU: Initially, we describe the performance results obtained on a single GPU. The average time taken by a framework to complete one iteration during training is used to evaluate the framework's performance. As a result, we can compare the time spent on each step of SGD. The timings are given in Table II and shown graphically in Figure 2. The results of each phase will be discussed in the following sections.

In the initial phase of the performance model, all three frameworks have multiple threads to read data from the CPU memory to the GPU. By observing the results in Table II we see that for all frameworks the I/O time is small. In the second phase, after the reading of data from disk to memory, the data should be transmitted to the GPU for training purpose. Our tested environment uses PCIe to connect the CPU and GPU, which provides a total bandwidth of 11 GB/sec. From the results in Table II, we see that Chainer typically has higher memory copy time than both TensorFlow and MXNet.

TABLE II: Measured time of SGD phases on single GPU. All times are given in seconds, as the mean and standard deviations over all iterations in a single epoch of training.

CNN	Chainer	MXNet	TensorFlow
t_{io}	0.0004±0.00002	0.0002±0.00005	0.0006±0.00008
t_{h2d}	0.0383±0.0054	0.0201±0.0027	0.0212±0.0023
t_{md}	0.0006±0.00003	0.0003±0.00001	0.0005±0.00002
$\sum_{i=1}^{l} t_{f_i}^l$	0.0663±0.0031	0.0307±0.0073	0.3489±0.0729
$\sum_{i=1}^{l} t_{b_i}^l$	0.0594±0.0030	0.1347±0.0040	0.1151±0.0170
t_u	0.2365±0.0194	0.1564± 0.0514	0.2636±0.0469
t_{iter}	0.4009±0.0240	0.3421±0.0354	0.7494 ±0.1391

MLP	Chainer	MXNet	TensorFlow
t_{io}	0.0001±0.000018	0.0005±0.00008	0.0003±0.000025
t_{h2d}	0.0331±0.0062	0.0182±0.00078	0.0199±0.0035
t_{md}	0.0006±0.00001	0.0003±0.00003	0.0005±0.00008
$\sum_{i=1}^{l} t_{f_i}^l$	0.0523±0.0067	0.1034 ±0.0082	0.0576±0.0045
$\sum_{i=1}^{l} t_{b_i}^l$	0.0481±0.0280	0.1754±0.0187	0.1680 ±0.0134
t_u	0.4533±0.0095	0.2054±0.00099	0.5985±0.0089
t_{iter}	0.5869±0.0575	0.3992±0.0591	1.4371±0.0597

AN	Chainer	MXNet	Tensorflow
t_{io}	0.0004±0.00005	0.0001±0.00003	0.0005±0.00008
t_{h2d}	0.0316±0.0026	0.0185±0.0090	0.0215±0.0030
t_{md}	0.0006±0.00003	0.0003±0.00001	0.0005±0.00008
$\sum_{i=1}^{l} t_{f_i}^l$	0.1388±0.0045	0.1322±0.0064	0.1595±0.0072
$\sum_{i=1}^{l} t_{b_i}^l$	0.1421± 0.0056	0.2265± 0.0076	0.4274±0.0103
t_u	0.3675±0.0201	0.3287±0.0307	0.3765 ±0.0215
t_{iter}	0.6804±0.0328	0.706±0.0258	0.9854±0.0320

TABLE III: Gradient aggregation time in the multi-GPU experiments

Network	Framework	t_{comm}	
		2 GPUs	3 GPUs
CNN	Tensorflow	0.3945	0.4017
	MXNet	0.3245	0.3415
	Chainer	0.3106	0.3404
MLP	Tensorflow	0.3024	0.4145
	MXNet	0.3156	0.2569
	Chainer	0.2945	0.2345
Autoencoder	Tensorflow	0.7187	0.7199
	MXNet	0.3565	0.3698
	Chainer	0.4563	0.4583

In the third phase (t_{md}), the three frameworks differ in the data distribution to GPUs. In the Chainer framework, the data batch is divided into multiple batches in the GPU whereas in the MXNet and Tensorflow framework, batches are divided into mini-batches on the CPU and then transferred to the GPUs dynamically. As a result the Chainer framework takes 0.3s higher compared to MXNet and TensorFlow.

In the forward phase, we can see that while the results are comparable in the case of the Autoencoder and MLP models, in the CNN model, Tensorflow is significantly slower than both the MXNet and Chainer frameworks. MXNet's performance is good in the forward phase due to its usage of auto symbolic differentiation and imperative programmimg [11]. In the case of the CNN, both Chainer and MXNet are able to autotune to determine the optimal convolutional algorithms for convolutional layers, but TensorFlow does not allow the

convolution techniques to be customized. TensorFlow uses the Winograd algorithm, which in some situations may be suboptimal. Considering the CNN model, MXNet makes use of GEMM-based convolution, which results in 0.05s less forward phase and up to 0.15s more backward phase. Chainer employs the FFT technique [21], which results in a forward phase that is 0.06s higher and 0.1s less in the backward phase.

Next in the backward phase, MXNet is slower than the TensorFlow and Chainer frameworks. The values t_f and t_b are different in performance due to differing use of the cuDNN API. cuDNN may have different performance depending on the parameters that are used. Some factors that affect performance are: Data Layout, Implicit matrix multiplications, Dimension quantization techniques, Convolution parameters such as Batch size, Height and width filtersize, channels in and out (NHWC, NCWH) and strides. For example, in MXNet and Chainer, the NCHW data layout are used whereas TensorFlow has NHWC layout which acts as a performance factor.

2) Multi-GPU: In a multi-GPU per node testing, we scaled the mini-batch with the number of GPUs. Each GPU has the same dataset. As the number of GPUs increases, data communication overhead increases due to the data aggregation process between devices. Our measurements of this time t_{comm} are give in Table III. Figure 3 shows the results for the speedup when running on two and three GPUs, and the breakdown of the timings in terms of the performance model are shown in Figure 4.

From Figure 3, we see that MXNet achieves linear scaling on one to three GPUs, while Chainer achieves speeds 0.2X less than MXNet. From Figure 4, we see that the data aggregation time t_a in MXNet is less than in the TensorFlow and Chainer frameworks. Here, MXNet parallelizes the gradient aggregation with back propagation i.e., after the gradients of the current layer(l_i) are computed, the preceding layer (l_{i-1}) of backward propagation can be performed without latency. As a result, gradient computation of (l_{i-1}) is parallelized with gradient aggregation of l_i. Thus, following computing layers can hide much of the synchronisation overhead of gradients. As a result, MXNet has less aggregation time and good scalability compared to other frameworks. TensorFlow implements S-SGD differently. It has no parameter server and uses peer-to-peer memory access if it is compatible with the hardware topology. Each GPU receives gradients from other GPUs, averages them, and updates the model when the backward propagation completes, even from the decentralised method. In this process, the model update t_u and backward propagation has no computation overlap, which led to the observed relatively poor scaling performance in TensorFlow.

3) Load Imbalance Factor: Load balancing in a parallel system plays a major role in determining scalability. A load imbalance occurs when work is distributed unevenly among workers. Here we have calculated the *Load Imbalance Factor* for each neural network model in each deep learning framework based on Equation 8.

From the results in Table IV, it is clear that all three frameworks are not well balanced, since in all cases the load imbalance factor is greater than one, and in some cases significantly greater. Qualitatively, we see that the higher values of load imbalance correspond to the lower speedups, and degraded scalability, see Figure 3. For example, in the case of Tensorflow, we see that poor scalability is accompanied by relatively high values of the load imbalance factor.

TABLE IV: Load Imbalance Factor

Network	Framework	Load Imbalance Factor	
		2 GPUs	3 GPUs
TensorFlow	CNN	1.15	1.23
	MLP	1.189	1.20
	AN	1.175	1.27
Chainer	CNN	1.025	1.052
	MLP	1.032	1.043
	AN	1.152	1.202
MXNet	CNN	1.013	1.030
	MLP	1.015	1.079
	AN	1.142	1.213

Here we also present further linear regression analysis to understand how load imbalance factor contributes to parallel inefficiency, according to the equation below:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon \tag{9}$$

where x_1 and x_2 represent the number of GPUs and load imbalance factor respectively, y is the total execution time of an epoch, $\beta_0, \beta_1, \beta_2$ are the regression coefficients and ϵ represents a random value indicating the error in each observation of y. The values of β_0,β_1, and β_2 should be chosen to minimise the sum of squared prediction errors.

We find the following values for the coefficients in the nine cases. For CNN model using TensorFlow,

$$y^{'} = 0.00001 - 40.5588\bar{X}_1 + 369.4848\bar{X}_2 \tag{10}$$

For MLP model using TensorFlow,

$$y^{'} = 0.00001 - 16.1671\bar{X}_1 + 245.9177\bar{X}_2 \tag{11}$$

For AN model using TensorFlow,

$$y^{'} = 0.00002 - 49.1037\bar{X}_1 + 398.6701\bar{X}_2 \tag{12}$$

For CNN model using MXNet,

$$y^{'} = 0.000011 - 5.57483\bar{X}_1 + 287.4133\bar{X}_2 \tag{13}$$

For MLP model using MXNet,

$$y^{'} = 0.00002 - 28.9346\bar{X}_1 + 326.7283\bar{X}_2 \tag{14}$$

For AN model using MXNet,

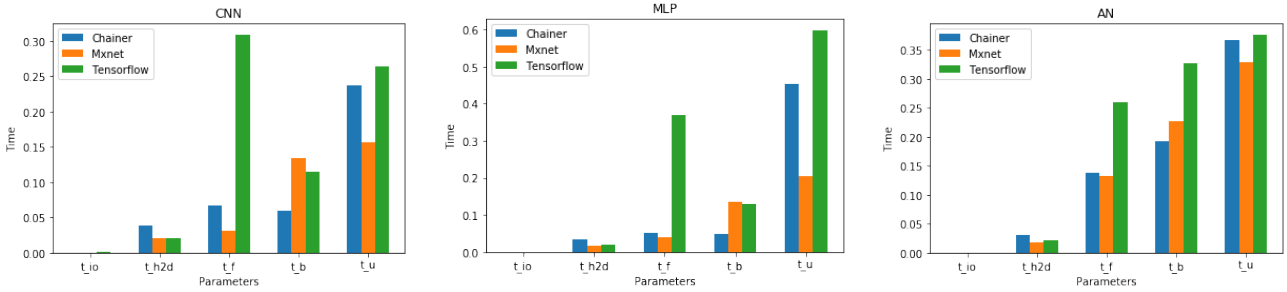
$$y^{'} = 0.00002 - 33.1037\bar{X}_1 + 398.6701\bar{X}_2 \tag{15}$$

For CNN model using chainer,

$$y^{'} = 0.00001 - 9.00791\bar{X}_1 + 286.8447\bar{X}_2 \tag{16}$$

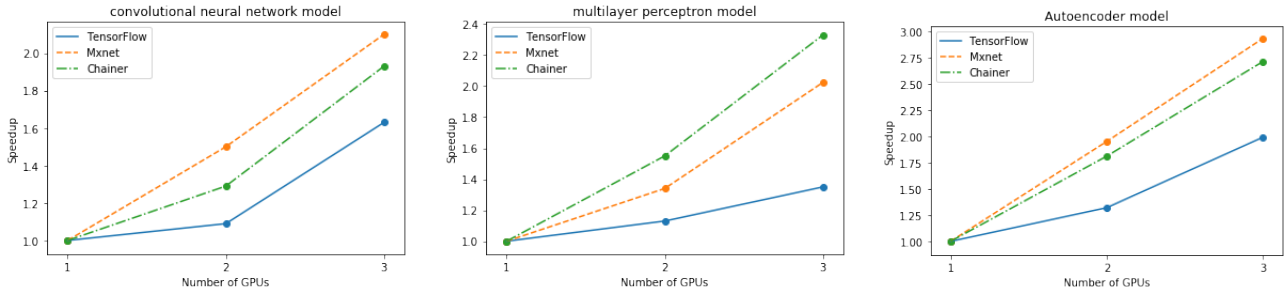
For MLP model using chainer,

$$y^{'} = 0.000016 - 10.1584\bar{X}_1 + 292.1287\bar{X}_2 \tag{17}$$



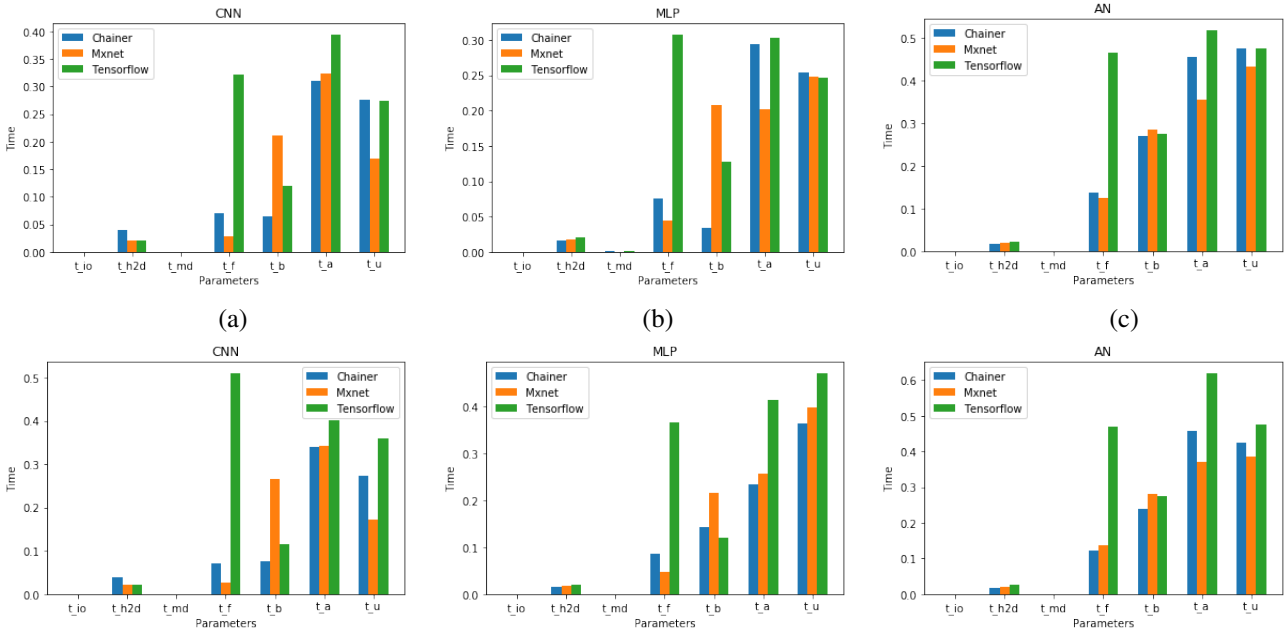
(a) (b) (c)

Fig. 2: Iteration times on a single GPU for (a) CNN (b) MLP and (c) Autoencoder models



(a) (b) (c)

Fig. 3: Measured speedup for the three frameworks on different numbers of GPUs for the three DNN models (a) CNN, (b) MLP and (c) Autoencoder.



(a) (b) (c)

(d) (e) (f)

Fig. 4: Iteration time on multiple GPUs. Results for two GPUs are shown in (a), (b), (c) for CNN, MLP and Autoencoder. Results for three GPUs are in (d), (e) and (f).

For AN model using chainer,

$$y^{'} = 0.000068 - 25.584\bar{X}_1 + 260.263\bar{X}_2 \quad (18)$$

where $y^{'}$ is the computed prediction execution time as a function of the number of GPUs and the load imbalance factor.

We compute their coefficients of determination, R^2, to further investigate the impact of number of GPUs and load imbalance factor on execution time. R^2 represents the proportion of the variance in execution time that is predicted from both number of GPUs and load imbalance factor. It is defined as follows:

$$R^2 = 1 - \frac{SS_{rss}}{SS_{tss}} \tag{19}$$

where SS_{rss} and SS_{tss} are the residual sum of squares and the total sum of squares. They are defined as:

$$SS_{rss} = \sum (y - y^{'})^2 \tag{20}$$

and,

$$SS_{tss} = \sum (y - \bar{y})^2 \tag{21}$$

We find $R^2 values for (CNN, MLP, AN) Tensor Flow of (0.9904, 0.9965, 0.9960), for MXNet of (0.9901, 0.9981, 0.9905) and for Chainer$ increases, the model's fit to the training data becomes more accurate and precise. The results confirm the importance of load balancing to achieve scalability in distributed deep learning.

V. Conclusion and future work

We have evaluated the performance of different deep learning frameworks over different deep learning neural networks in terms of scalability in a multi-GPU environment, taking into account a range of factors affecting performance, including load imbalance. We have further extended an existing performance model [18] based on synchronous-S-SGD with the inclusion of two new parameters: time taken to divide the batch into mini-batches and maximum time taken by GPU. The proposed performance model was built to measure the performance of different deep learning framework implementations which include TensorFlow, MXNet and Chainer frameworks on three models: Convolutional neural network, Multilayer perceptron and Autoencoder models, in a multi-GPU environment. The experimental results have shown that MXNet and Chainer have better scalability compared to TensorFlow for all three models. Moreover, our analysis of the load imbalance factor has shown that load balancing is a contributing factor to scalability in distributed deep learning, and high load imbalance is strongly correlated with poor scalability in our experiments. Future work will probe the reason for the load imbalance in these cases, with the aim of discovering optimal parameters to keep the load balanced.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] Y. Ko, K. Choi, J. Seo, and S.-W. Kim, "An in-depth analysis of distributed training of deep neural networks," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2021, pp. 994–1003.

[3] S. Shi, Q. Wang, K. Zhao, Z. Tang, Y. Wang, X. Huang, and X. Chu, "A distributed synchronous sgd algorithm with global top-k sparsification for low bandwidth networks," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 2238–2247.

[4] Y. Kim, H. Choi, J. Lee, J.-S. Kim, H. Jei, and H. Roh, "Efficient large-scale deep learning framework for heterogeneous multi-gpu cluster," in *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, 2019, pp. 176–181.

[5] B. Shinde and S. T. Singh, "Data parallelism for distributed streaming applications," in *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*, 2016, pp. 1–4.

[6] M. H. Mofrad, R. Melhem, Y. Ahmad, and M. Hammoud, "Studying the effects of hashing of sparse deep neural networks on data and model parallelisms," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, 2020, pp. 1–7.

[7] N. Takisawa, S. Yazaki, and H. Ishihata, "Distributed deep learning of resnet50 and vgg16 with pipeline parallelism," in *2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW)*, 2020, pp. 130–136.

[8] K.-N. Joo and C.-H. Youn, "Accelerating distributed sgd with group hybrid parallelism," *IEEE Access*, vol. 9, pp. 52 601–52 618, 2021.

[9] A. A. Awan, K. Hamidouche, J. M. Hashmi, and D. K. Panda, "S-caffe: Co-designing mpi runtimes and caffe for scalable deep learning on modern gpu clusters," in *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2017, pp. 193–205.

[10] 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)*, 2016, pp. 265–283.

[11] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[12] S. Tokui, R. Okuta, T. Akiba, Y. Niitani, T. Ogawa, S. Saito, S. Suzuki, K. Uenishi, B. Vogel, and H. Yamazaki Vincent, "Chainer: A deep learning framework for accelerating the research cycle," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 2002–2011.

[13] A. A. Awan, K. Hamidouche, A. Venkatesh, and D. K. Panda, "Efficient large message broadcast using nccl and cuda-aware mpi for deep learning," in *Proceedings of the 23rd European MPI Users' Group Meeting*, 2016, pp. 15–22.

[14] F. Yan, O. Ruwase, Y. He, and T. Chilimbi, "Performance modeling and scalability optimization of distributed deep learning systems," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 1355–1364.

[15] Y. Oyama, A. Nomura, I. Sato, H. Nishimura, Y. Tamatsu, and S. Matsuoka, "Predicting statistics of asynchronous sgd parameters for a large-scale distributed deep learning system on gpu supercomputers," in *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 2016, pp. 66–75.

[16] M. Song, Y. Hu, H. Chen, and T. Li, "Towards pervasive and user satisfactory cnn across gpu microarchitectures," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 1–12.

[17] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," in *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*. IEEE, 2016, pp. 99–104.

[18] S. Shi, Q. Wang, and X. Chu, "Performance modeling and evaluation of distributed deep learning frameworks on gpus," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2018, pp. 949–957.

[19] S. Pllana, S. Benkner, F. Xhafa, and L. Barolli, "Hybrid performance modeling and prediction of large-scale computing systems," in *2008 International Conference on Complex, Intelligent and Software Intensive Systems*. IEEE, 2008, pp. 132–138.

[20] T. Fahringer, S. Pllana, and J. Testori, "Teuta: Tool support for performance modeling of distributed and parallel applications," in *International Conference on Computational Science*. Springer, 2004, pp. 456–463.

[21] H. Kim, H. Nam, W. Jung, and J. Lee, "Performance analysis of cnn frameworks for gpus," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2017, pp. 55–64.

[22] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky *et al.*, "Theano: A python framework for fast computation of mathematical expressions," *arXiv e-prints*, pp. arXiv–1605, 2016.

[23] R. Collobert, S. Bengio, and J. Mariéthoz, "Torch: a modular machine learning software library," Idiap, Tech. Rep., 2002.
[24] H. Qi, E. R. Sparks, and A. Talwalkar, "Paleo: A performance model for deep neural networks." in *ICLR (Poster)*, 2017.
[25] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Performance modeling for cnn inference accelerators on fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 4, pp. 843–856, 2019.
[26] A. Viebke, S. Pllana, S. Memeti, and J. Kolodziej, "Performance modelling of deep learning on intel many integrated core architectures," in *2019 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2019, pp. 724–731.
[27] "Nsight systems," https://developer.nvidia.com/nsight-systems, accessed: 2021.