Proof-of-Concept of a Flexible and High-Fidelity Approach to Distributed DNN Training Emulation

Banruo Liu[*]
Tsinghua University

Mubarak Adetunji Ojewale
KAUST

Yuhan Ding
Tsinghua University

Marco Canini
KAUST

ABSTRACT

We propose NeuronaBox, a flexible, user-friendly, and high-fidelity approach to emulate DNN training workloads. We argue that to accurately observe performance, it is possible to execute the training workload on a subset of real nodes and emulate the networked execution environment along with the collective communication operations. Initial results from a proof-of-concept implementation show that NeuronaBox replicates the behavior of actual systems with high accuracy, with an error margin of less than 1% between the emulated measurements and the real system.

CCS CONCEPTS

• **Networks → Network experimentation**; • **Computing methodologies → Machine learning**.

KEYWORDS

Distributed Deep Learning Training, Machine Learning Systems, DNN Training Emulation

ACM Reference Format:
Banruo Liu, Mubarak Adetunji Ojewale, Yuhan Ding, and Marco Canini. 2024. Proof-of-Concept of a Flexible and High-Fidelity Approach to Distributed DNN Training Emulation. In *SIGCOMM Workshop on Networks for AI Computing (NAIC '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3672198.3673793

1 INTRODUCTION

Modern DNN training clusters are remarkable engineering feats that more closely resemble high-performance specialized computing environments – and the large costs that these entail – than their mainstream counterparts in commodity cloud computing datacenters. Optimizing resource utilization and overall efficiency is paramount to maximizing the performance of training workloads and minimizing associated costs.

Conducting in-depth "what if" analyses is essential to making informed decisions and beneficial for a variety of scenarios. For instance, a ML engineer may want to explore for a given model the impact of a particular parallelization strategy on training time and resource utilization. But it is not practical to profile the training workload on thousands of HW accelerators (GPUs, TPUs, etc.) for each possible strategy and different configurations. Recent work has shown the potential of simulation and analytical methods to gain insights about DNN training behavior [1, 2, 11–13, 17]. However, these approaches suffer from at least one of three limitations: 1) they require significant effort to transform the actual workloads into an input model for the simulator, 2) they require explicit models of parallelization strategies and incorporating new ones entails non-trivial development of new simulation models, and, 3) the fidelity of their results is limited by how faithful the underlying analytical models of computation and communication are, which are notoriously difficult to get right at scale [9].

This work pioneers and advocates the use of emulation to aid in the analysis and experimentation of distributed DNN training workloads. In a nutshell, we propose to isolate a node subset (denoted as $\mathcal{N}$) of a distributed training job and emulate the networked execution environment (denoted as $\mathcal{E}$) from the perspective of the nodes in $\mathcal{N}$. We elect to view the network as a natural boundary between the real and emulated environments since communication between nodes in distributed training jobs typically occurs through a collective communication library (e.g., NCCL [10]) that both isolates the training scripts from dealing with all the unnecessary details of the underlying network and demarcates clear points for inter-process synchronization. We refer to our approach as NeuronaBox.

Notably, in this approach, the nodes in $\mathcal{N}$ run unmodified training scripts, DNN frameworks and libraries. In particular, the communication is handled by the actual collective communication library over the network fabric. Meanwhile, the emulation environment $\mathcal{E}$ executes on dedicated hardware resources. The requirements for the emulation environment are modest: it can run on a single CPU-based node, and it only requires network bandwidth to match the available aggregate bandwidth of nodes in $\mathcal{N}$.

The key benefit of this approach is that it allows us to faithfully execute on real hardware a portion of the training workload, which executes without overheads from instrumentation (since there is none) nor profiling $\mathcal{N}$ in controlled conditions. Therefore, we can observe the actual behavior of the training job, including the HW utilization metrics and collective communication patterns that are critical in analyzing the performance of distributed training workloads. We wish to stress that our objective is to enable performance analysis and optimization of distributed training workloads. Implications on model quality are out of scope. Thus, in this work, we initiate the study of these core research questions: 1) *What aspects of the workload must $\mathcal{E}$ emulate?* 2) *How can this approach maintain high fidelity while retaining wide applicability?*

[*]Work done primarily while author was interning at KAUST.

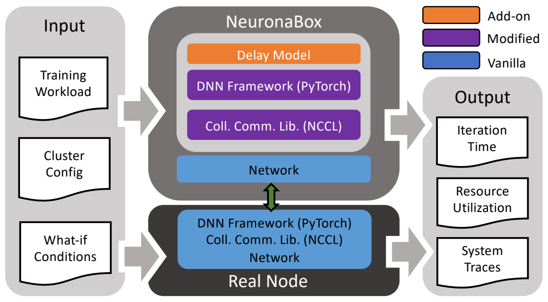

Figure 1: Overall workflow and architecture of NeuronaBox.

2 CONTRIBUTION AND PRELIMINARY RESULTS

Our goal is to enable any subset $\mathcal{N}$ of nodes in a distributed DNN training job to execute the workload as if it were running on the entire set of nodes and resources. We propose to achieve this goal by emulating the interactions between $\mathcal{N}$ and its networked environment $\mathcal{E}$, which in a sense can be viewed as a virtualization of the remaining job's nodes. We argue that by observing the performance of $\mathcal{N}$, we can analyze and extrapolate the behavior for an entire job with high fidelity. In our design, we envision that NeuronaBox 1) *should be used without any modification to their existing user code; and* 2) *should be flexible to seamlessly adapt to changes in parallelization strategies, including new ones that may emerge in the future.*

Workflow and architecture. Fig. 1 depicts an overview of our approach. The high-level workflow of NeuronaBox is as follows: First, the user provides the training script, the job configuration (e.g., world size, nodes in $\mathcal{N}$, HW resources, etc.), and optionally a set of what-if conditions for experimentation (an example is given later). Second, NeuronaBox initializes the emulation environment by synthesizing the network topology and instantiating a communication model that calculates delay times for collective operations within the emulated environment. Third, the training script is launched (e.g., via torchrun). Meanwhile, desired performance metrics like iteration time and resource utilization are gathered in $\mathcal{N}$. Traces of collective communication (e.g., NCCL traces) can also be collected.

Assumptions. We assume that nodes have uniform hardware and network configuration. In practice, it is common to execute distributed training jobs on homogeneous clusters [7, 8, 15, 16, 18]. We assume that the model fits entirely within $\mathcal{N}$. This assumption is not restrictive, as it is common to use model or tensor parallelism within a node or a shard [6, 14]. These assumptions imply a sort of symmetry in the workload distribution across the nodes, which allows us to treat the nodes in $\mathcal{N}$ as a representative sample of the entire nodes. Further, we assume that the collective communication layer is the only point of interaction between $\mathcal{N}$ and $\mathcal{E}$. This assumption is reasonable, as the collective communication layer is the primary interface between the computation and the network stack in distributed training jobs. Finally, note that we are free to modify the DNN framework and collective communication libraries within the emulator. That is how we are able to implement NeuronaBox!

Scalability. Our key insight is that we are only interested in the interaction between $\mathcal{N}$ and the outside world. And so, the actual communication between the emulated nodes can be skipped. Instead, only the delay resulting from these communication operations needs to be incorporated into the emulation. As a result, the number of connections as well as the amount of data transfer for NeuronaBox are the same as that of $\mathcal{N}$. This observation allows NeuronaBox to potentially scale to a large number of nodes. We plan to further explore scalability in future work.

Model	Time-E	Time-B	CPU-E	CPU-B
BERT	629 ± 3.0	628±1.1	12.93%	14.25%
ResNet152	1061±19.8	1063±16.3	12.68%	12.95%
DeepLight	727±15.0	726± 13.8	7.52%	7.75%

Table 1: End-to-end workload comparison. 'E' and 'B' stand for emulator-enabled (NeuronaBox) and the baseline, respectively. 'Time' stands for the training times in milliseconds; and 'CPU' stands for the percentage of CPU usage in a node.

Proof-of-concept implementation. Our proof-of-concept implementation entails the development of an end-to-end system using the PyTorch DNN framework and NCCL as the collective communication library, chosen because of their popularity. Our implementation is able to run two-node training using a distributed data-parallel strategy. We plan to release NeuronaBox as open source.

Experimental setup. To evaluate NeuronaBox's ability to accurately emulate end-to-end DNN training, we conducted experiments using three real-world DNN models; BERT [4], ResNet152 [5], and DeepLight [3].

Preliminary results. As shown in Table 1, NeuronaBox is quite accurate in a two-node environment training with data parallelism, with error less than 1%. For CPU usage, it actually drops a little bit in all scenario. We attribute that to: (1) the efficient and lightweight implementation of NeuronaBox, which keeps the overhead generally low; (2) the removal of computation in backward pass, which eliminates a lot of memory allocation and data movements. So the net effect is a drop in CPU usage. This is promising in terms of the potential scalability of NeuronaBox.

3 FUTURE WORK

The proof-of-concept implementation only features a 2-node cluster emulation. We intend to extend our implementation to emulate many nodes simultaneously. We intend to achieve this by intercepting the topology detection and memory allocation interfaces of the DNN framework and then modifying them to fit into the workflow of NeuronaBox. Also, the current implementation only emulates the **amount** of data transmitted over the network and not the actual **values** of the data. This is sufficient to observe the time for each training iteration, but not sufficient to observe the training progress per iteration, especially for DNN training speed-up techniques such as lossy quantization and compression. We intend to address this problem in the future. Finally, we encourage further research in this direction, as there are still many questions to be answered.

ACKNOWLEDGMENTS

This publication is based upon work supported by the King Abdullah University of Science and Technology (KAUST) Office of Research Administration (ORA) under Award No. ORA-CRG2020-4382.

REFERENCES

[1] Newsha Ardalani, Saptadeep Pal, and Puneet Gupta. 2024. DeepFlow: A Cross-Stack Pathfinding Framework for Distributed AI Systems. *ACM Trans. Des. Autom. Electron. Syst.* 29, 2 (2024).

[2] Jehyeon Bang, Yujeong Choi, Myeongwoo Kim, Yongdeok Kim, and Minsoo Rhu. 2023. vTrain: A Simulation Framework for Evaluating Cost-effective and Compute-optimal Large Language Model Training. (2023). arXiv:cs.LG/2312.12391

[3] Wei Deng, Junwei Pan, Tian Zhou, Deguang Kong, Aaron Flores, and Guang Lin. 2021. DeepLight: Deep Lightweight Feature Interactions for Accelerating CTR Predictions in Ad Serving. In *WSDM*.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*.

[6] Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, Joe Chau, Peng Cheng, Fan Yang, Mao Yang, and Yongqiang Xiong. 2023. Tutel: Adaptive Mixture-of-Experts at Scale. In *MLSys*.

[7] Fan Lai, Wei Zhang, Rui Liu, William Tsai, Xiaohan Wei, Yuxi Hu, Sabin Devkota, Jianyu Huang, Jongsoo Park, Xing Liu, Zeliang Chen, Ellie Wen, Paul Rivera, Jie You, Chun cheng Jason Chen, and Mosharaf Chowdhury. 2023. AdaEmbed: Adaptive Embedding for Large-Scale Recommendation Models. In *OSDI*.

[8] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. (2019). arXiv:cs.CL/1907.11692

[9] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. 2018. Revisiting Network Support for RDMA. In *SIGCOMM*.

[10] NVIDIA. 2024. Collective Communication Library (NCCL). (2024). https://developer.nvidia.com/nccl.

[11] Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. 2020. ASTRA-SIM: Enabling SW/HW Co-Design Exploration for Distributed DL Training Platforms. In *ISPASS*.

[12] Saeed Rashidi, William Won, Sudarshan Srinivasan, Srinivas Sridharan, and Tushar Krishna. 2022. Themis: A Network Bandwidth-Aware Collective Scheduling Policy for Distributed Training of DL Models. In *ISCA*.

[13] Wilfredo J. Robinson M., Flavio Esposito, and Maria A. Zuluaga. 2022. DTS: A Simulator to Estimate the Training Time of Distributed Deep Neural Networks. In *MASCOTS*.

[14] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. (2020). arXiv:cs.CL/1909.08053

[15] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. (2023). arXiv:cs.CL/2302.13971

[16] Guanhua Wang, Heyang Qin, Sam Ade Jacobs, Xiaoxia Wu, Connor Holmes, Zhewei Yao, Samyam Rajbhandari, Olatunji Ruwase, Feng Yan, Lei Yang, and Yuxiong He. 2024. ZeRO++: Extremely Efficient Collective Communication for Large Model Training. In *ICLR*.

[17] William Won, Taekyung Heo, Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. 2023. ASTRA-sim2.0: Modeling Hierarchical Networks and Disaggregated Systems for Large-model Training at Scale. In *ISPASS*.

[18] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. 2022. Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning. In *OSDI*.