Moneo: Non-intrusive Fine-grained Monitor for AI Infrastructure

1st Yuting Jiang
Nanjing University
Nanjing, China
mf1933044@smail.nju.edu.cn

2nd Yifan Xiong
Microsoft Research
Beijing, China
yifan.xiong@microsoft.com

3rd Lei Qu
Microsoft Research
Beijing, China
lei.qu@microsoft.com

4th Cheng Luo
Microsoft Research
Beijing, China
luocheng@microsoft.com

5th Chen Tian
Nanjing University
Nanjing, China
tianchen@nju.edu.cn

6th Peng Cheng
Microsoft Research
Beijing, China
pengc@microsoft.com

7th Yongqiang Xiong
Microsoft Research
Beijing, China
yongqiang.xiong@microsoft.com

***Abstract*—Cloud-based AI infrastructure is increasingly important, especially on large-scale distributed training. To improve its efficiency and serviceability, real-time monitoring of the infrastructure and profiling the workload are proved to be the effective approach empirically. However, cloud environment poses great challenges as service providers cannot interfere with their tenants' workloads or touch user data, thus previous instrumentation-based monitoring approach cannot be applied, nor does the workload trace collection.**

We propose Moneo, a non-intrusive cloud-friendly monitoring system for AI infrastructure. Moneo is capable of intelligently collecting the key architecture-level metrics at finer granularity in real-time without instrumenting or tracing the workloads, which has been deployed in real production cloud, Azure. We analyze the results reported by Moneo for typical large-scale distributed AI workloads from real deployment. Results demonstrate that Moneo can effectively help service providers understand the real resource usage patterns of various AI workloads and real networking requirements, so as to get valuable findings help improve the efficiency of cloud infrastructure and optimize the software stack with the consideration of the characteristic resource usage requirements for different AI workloads.

***Index Terms*—AI infrastructure, monitor, cloud, distributed training.**

I. Introduction

Training deep neural network (DNN) models usually requires a long time on a single arithmetic computing device, resulting in distributed training using multiple devices are preferred to reduce training time. However, most individual users can hardly afford to purchase many computing devices like GPUs themselves. Consequently, cloud-based AI infrastructure has been popular for conducting distributed training such as Azure [1], Amazon AWS [2] for flexibility and economy.

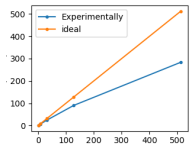
However, for cloud-based AI infrastructure, training efficiency is critical, particularly for large-scale distributed training. Extending training time and cost are the primary concerns, as model parameters have increased dramatically in recent years. It takes OpenAI just over a year to double the capacity of GPT models from 1.5 B to 175 B. The latest GPT-3 model would require 355 years and 4.6 M dollars to train even with the single Tesla V100 cloud instance [3].

Even with the most advanced and expensive AI hardware, we observe a significant decrease in training efficiency as the number of GPUs increases, as illustrated in Fig.1(a). There are two fundamental reasons, one is that the software stack is not optimized sufficiently, and the other is that the hardware is over-provisioning resulting in unnecessary and expensive costs. To optimize the efficiency, they first need to know where the bottleneck is and what is being underutilized. Fortunately, real-time monitoring and profiling are proved to be the effective approach empirically to help pinpoint inefficiencies [4].

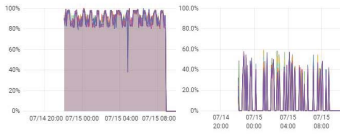
However, the cloud environment imposes significant challenges on real-time profiling because service providers can not interfere with tenants' workload execution and collect highly-related information with the users. Thus, the existing tools can not meet the requirements.

First, instrumentation-based approaches can not be applied since they will interfere with tenants' workloads. For instance, memory usage can be obtained by adding hooks to the code [5] [6]. NVIDIA CUPTI API [7] requires instrumenting the kernel in order to collect events and metrics that are used by a variety of tools including NVProf [8], Nsight [9] and others [10] [11]. Second, the majority of fine-grained profiling tools require the collection of workload traces. Popular frameworks provide TensorBoard [10] for tracing the op-level execution timeline on the GPU. NVIDIA provides performance measurement tools ranging from the kernel to the application level. [8] [9] [12]. These tools will replay kernels, APIs, and gather workload traces. We discovered that Nsight [9] can increase the step time for ResNet50 by more than 50% through experimentation. Not only do these tools incur significant overhead, but they also expose users to privacy risks. Third, some existing AI workload monitors are overly coarse-grained. NVIDIA-smi [13], nvtop [14] can easily monitor GPU utilization, which is the percentage of time a kernel is using GPU over the previous sample period. However, they cannot obtain finer-grained information contained within GPUs. A GPU contains dozens of streaming multiprocessor (SM), and GPU utilization cannot distinguish between only one SM or all SMs in use,

so it cannot effectively reflect the actual use of computing resources. In comparison, we demonstrate finer-grained metric, SM utilization, highlighting the proportion of SMs that are used. A kernel that utilizes all SMs and runs for the duration of the time interval will have 100% activity. As illustrated in Fig.1(b), the average GPU utilization for mixture of experts (MoE) GPT-3 is approximately 90%, while the average SM utilization is approximately 20%.



(a) ideal vs. real GPT-2 FP32 speedup with increase in GPUs



(b) GPU utilization vs. SM utilization of MoE GPT-3

Fig. 1. Experiments of GPT models

To address the aforementioned limitations, we propose Moneo, a non-intrusive fine-grained real-time monitor that is effective for cloud-based AI infrastructure. To avoid instrumenting individual applications, we collect information at the architecture level. Rather than collecting traces, we track communication and computation resource usage. Besides, we collect metrics at a finer granularity to more accurately and effectively reflect resource usage. Moreover, a real-time monitor can provide additional benefits. For instance, the majority of distributed DNN workloads employ the synchronization method, but this results in the degradation of performance on one device being propagated to all. Thus, from an end-to-end perspective, all devices execute at the same speed, and we cannot differentiate between normal and straggler devices. However, the straggler consumes resources abnormally, whereas others perform similarly in the same job, as illustrated in Fig.2. If we can detect it in real-time, we can perform better GPU resource scheduling to reduce job completion time and avoid wasting healthy resources, such as replacing the issued GPU with a new one to continue and repairing the issued GPU.

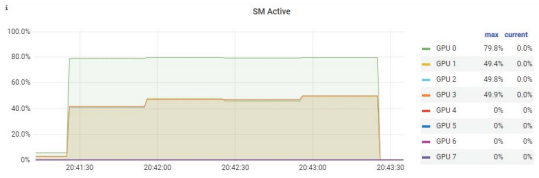


Fig. 2. Real-time abnormal resource usage

Nevertheless, real-time monitoring still has some challenges. The GPU contains hundreds of hardware counters and metrics, which significantly increase the cost of data querying and transmission. Additionally, too fast collection will impose significant overhead and stress on the system. To address these issues, we use data-driven statistical methods to identify several key metrics about communication and computation and dynamically adjust the collection frequency to significantly reduce overhead.

Moneo has been deployed in Azure to monitor fine-grained resource usage metrics in real-time. Moneo's data can reveal the resource usage patterns and assist in understanding the resource requirements of AI workloads. The resource usage patterns can help identify underutilized resources, such as GPU, memory, and network utilization, indicating the direction for full stack optimization, including arithmetic operators [15], compilation [16], and communication strategy [17]. Understanding the resource requirements of diverse workloads can guide the design of efficient architecture and future-proof hardware. Moreover, Moneo provides real-time, graphical analyzers enable users to monitor the health of system and tasks.

Moneo demonstrates its effectiveness in production by its data from three representative distributed workloads, including data parallelism, model parallelism, and MoE [18]. Analyzing these data sheds light on the performance optimization potential for the application software stack and hardware architecture design. Computing resources and GPU interconnection are underutilized to a great extent during model training. For instance, the peak Tensor Core usage is less than 35% and the maximum NVLink bandwidth is only approximately 10% of the specification. Additionally, we discover the divergent networking requirements of distributed workloads, which may aid in the design of efficient and flexible cloud-based AI system architectures. For example, model parallelism can be accomplished using existing network resources, whereas MoE may require a more powerful and dedicated network.

II. Moneo Overview

We propose an end-to-end pipeline for collecting fine-grained counters for AI workloads from AI hardware, scraping and storing the data in a time-series database, and presenting them in real-time to end-users. We identify a set of fine-grained key architecture-level metrics and collect them via hardware counters with dynamic collection frequency using a service. Moneo imposes little overhead on target machines, does not instrument or interfere with AI workloads, and does not collect user-relevant data.

A. System Architecture

Fig.3 illustrates the system architecture of Moneo including three main components, including metrics exporter, data collector, and analyzer.

a) Metrics Exporter: The metrics exporters are running on each target compute node and will query hardware or OS metrics exposed by hardware drivers or Linux kernel in short periods. They are responsible for exporting these metrics in real-time as a service. Each node may contain multiple metric exporters, each of which queries for unique hardware. There is, for instance, a GPU counter exporter, a network exporter, and a node exporter for basic operating system metrics such as CPU and disk. Metric exporters consume very little CPU resources as background services and do not interfere with running workloads.

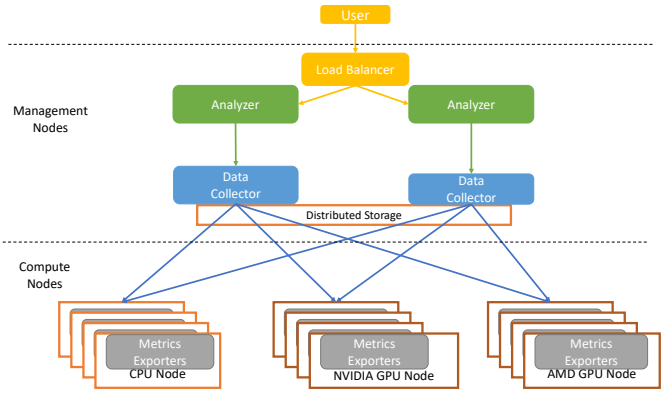


Fig. 3. Moneo System Architecture

b) Data Collector: The data collectors, which are run on additional management nodes, are in charge of grabbing metrics via HTTP from the computing node's metric exporter and storing them in the time series database (TSDB). When the AI cluster grows in size, the data collector can also be distributed across multiple nodes to ensure high availability and fault tolerance.

c) Analyzer: We define distinct dashboards for various metric and analysis categories to enable users to easily observe insights generated by analyzers.

B. Data-driven Metric Selection

We focus on the metrics for NVIDIA/AMD GPU and NVIDIA Mellanox InfiniBand, mainly used for computation and communication during AI workloads. We initially collected all counters available through the System Management Interface and GPU manager tools, and discovered that there are several hundred. The system will be burdened by high data query and transmission costs due to real-time monitoring of many metrics. However, not knowing which metrics are indicative, we employ data-driven statistical methods to identify a representative subset.

For further analysis, we collect time series of all metrics across diverse workloads, as shown in Fig.4. First, some metrics are almost constant whatever the workloads, so we can't learn much from them. In this case, if the time series of a metric's variance is below the threshold, it will be filtered out first. In addition, we find that the times series of some metrics are very similar so that we can keep just one. Initially, we tried to calculate the correlation coefficients between different metrics. In practice, however, metrics are highly correlated with workload execution, so using correlation coefficients to compare them may lead to unexpected results. For instance, the correlation coefficients of even the less similar metric like metric1 and metric2 in Fig.4 exceed 0.9. Then we use Euclidean distance to compare similarity of time series with no obvious shifts. However, the value range of different metrics may be very different, leading to inaccurate results. Therefore,

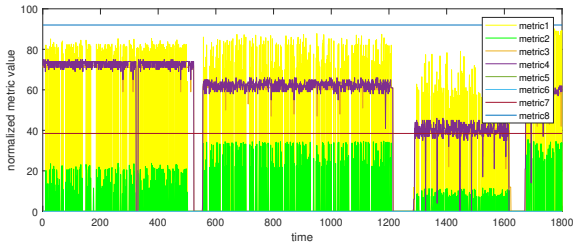


Fig. 4. Examples of metric patterns

we normalize the distance by dividing the maximum value of the metric.

$$\left(\sum_{i=1}^{n}\left|\frac{x_i - y_i}{\max\{x_1, y_1, \cdots, x_n, y_n\}}\right|^2\right)^{\frac{1}{2}} \quad (1)$$

If the distance is below the threshold, we keep only one metric.

C. Adaptive monitor frequency

One of the characteristics of DNN workload is repetitive because it repeats almost the same operations in each step. This means that metric changes may be relatively stable or have a regular pattern during the execution of a task, so that we may not always need to collect with the fastest frequency, so as to reduce the overhead on storage, memory, and network of the system. As shown in Fig.5(a), we observed most metrics' peak performance is fairly stable over a short period. 80% of the values fall between 0.65 and 0.75 around the peak, and the main reason for the fluctuation is a small amount of scattered minimum values. In this case, we can ignore the small number of minimum and focus on the metric peak performance envelope. Thus if the peak performance is stable over some time, an appropriate increase in the data collection interval can also indicate the peak performance during this period, allowing us to reduce the overhead.

However, as shown in Fig.5(b), the peak performance will change over time. The sudden changes in metrics are unpredictable. So we dynamically adjust data collection frequency online. If the current metric's peak value is stable, we will double the collection interval i to reduce system overhead. A small random number will be added to i to reduce the chance of error. If we detect that the peak value is not stable, we reduce the collection interval to the minimum.

There are two cases to check if the peak performance is stable. First, the peak value over the current sampling period is quite different from the historical peak value. The other is that although the peak performance in the current sampling period is similar to the historical peak performance, the values change dramatically in the current sampling period. Therefore, outside of the dynamic collection process, we also regularly sample the metric at the minimum frequency to obtain the ground truth of metric values in a short period. For the first case, we calculate the difference between the real peak value of the current sample and collected peak value since the previous sampling point in history, and check if it is greater than 10%

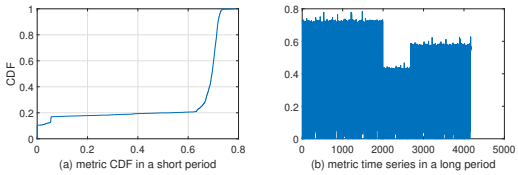


Fig. 5. Metric change characteristics in short time and long time

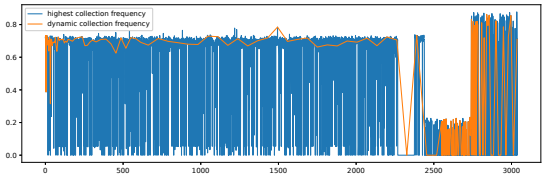


Fig. 6. Adaptive collection frequency and highest collection frequency

of the historical collected peak. For the second case, we count the values' probability density to check if the density around the peak value is greater than the threshold in the current sampling window. The sampling period is proportional to the collection interval. Besides, to avoid errors due to periodicity, we analyzed the cumulative distribution of stable intervals of the metrics from a large amount of historical data in advance and set it to 5 percentile of the stable intervals as the upper limit of the sampling period time.

As shown in Fig.6, when the peak values are relatively stable, we slow down the collection frequency to reduce system overhead. When the peak changes suddenly, we adjust to the fastest frequency.

III. Implementation and Deployment

A. Monitor Metrics

The collected metrics for GPU and InfiniBand are divided into six categories, the first three are related to the computation of GPU, and the last three are used for inter-GPU, intra-node, or inter-node communications, as shown in the Tab.I. They are collected through monitoring APIs provided by NVIDIA, AMD GPU manager tools, and Linux sysfs interface. Other hardware like disk and CPU will be used for dataset I/O and data processing during AI workloads training. We leverage existing metrics from node exporter [19] to monitor them.

B. Deployment and workloads

Moneo has been deployed to monitor the NDv4 instances in Azure which equip with AMD EPYC 7V12 Processor, 96 CPU cores, 900 GB memory, 8 NVIDIA Ampere A100 Tensor Core GPUs and each GPU is provided with its own dedicated, topology-agnostic 200 Gbps NVIDIA Mellanox HDR InfiniBand connection.

There are 3 typical workloads for large-scale distributed model training in production. The workload of data parallelism is to train GPT-2, BERT-Large, ResNet50, and VGG models with single precision in PyTorch on a single node or multiple nodes using data parallelism. The workload of

TABLE I
METRICS AND THEIR MEANINGS USED IN THIS SECTION

Category/ Metric	Description
GPU Basics	Common health monitoring for a single GPU, such as clock frequency,temperature, power, GPU level utilization, memory error correction code (ECC), etc.
GPU SM/CU	Streaming multiprocessor (SM) in NVIDIA GPU or computing unit (CU) in AMD GPU is used for tensor computation in AI workloads, and each GPU has many SMs or CUs.
SM Active (%)	The ratio of time SMs are active averaged over SMs over a time interval. A kernel using all SMs that runs over the entire time interval is 100%.
Tensor Active (%)	The fraction of cycles Tensor (HMMA/IMMA) pipe was active. Higher values indicate higher utilization of the Tensor Cores.
FP64/32/16 Active(%)	The fraction of cycles FP64/32/16 pipe was active.
GPU Memory	GPU memory is used for storing models and optimizers.
Mem Active (%)	The ratio of cycles the GPU memory interface is active sending or receiving data over the time interval.
NVLink/xGMI	NVIDIA NVLink and AMD xGMI are high-speed, direct GPU to GPU interconnect inside one node.
NVLink TX/RX (GB/s)	The rate of data transmitted/received over NVLink for each GPU in GB/s over a time interval.
PCIe	PCIe is used for memory copy between GPU devices and host.
PCIe TX/RX (GB/s)	The rate of data transmitted/received over PCIe for each GPU in GB/s over a time interval.
InfiniBand	NVIDIA Mellanox InfiniBand is used for inter-node communication with very high throughput and very low latency.
InfiniBand TX/RX (GB/s)	The rate of data transmitted/received all InfiniBand cards in a node in GB/s over a time interval.

model parallelism is to train GPT-3 [20] with 175 B parameters on 192 GPUs leveraging Megatron-LM [21], an open-source implementation of model parallelism. When using model parallelism, tensor parallelism is applied to all GPUs in one node, and then pipeline parallelism is used to scale up across nodes. The workload of MoE is to train the GPT-3 MoE model on 128 GPUs using Fairseq [22], a sequence modeling toolkit that supports MoE.

Note that Moneo will not log any user-related data concerning user privacy.

IV. Evaluation

We first simply evaluate Moneo overhead, then show some valuable findings based on the results reported by Moneo.

A. Moneo Overhead

On 30 different NDv4 machines, we train GPT-2, BERT-Large, VGG, ResNet, and DenseNet models in PyTorch. The results indicate that the variance in training throughput with and without Moneo ranges from -1.28% to 1.8%. And Moneo consumes no more than 0.1 percent of the CPU and 0.1 percent of the memory on each machine. As a result, Moneo imposes minimal overhead on monitored machines and does not affect the normal execution of workloads.

B. Some findings from data reported by Moneo

1) Underutilized computation resources: We study the computation resource utilization of various workloads on a cluster by collecting SM utilization data over one day and smoothing it per minute. Moneo discovers that over 90% of the time, the smoothed SM utilization is less than 30%, indicating a large room for computation resource utilization improvement.

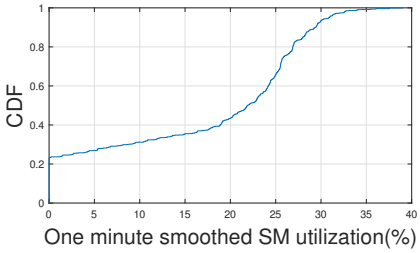


Fig. 7. CDF - One minute smoothed SM utilization

Then, we study the computation resource usage patterns for various models on a single node using data parallelism, as shown in Table.II. It's worth noting that Tensor Core usage in model training is extremely low. However, Tensor Core is a critical improvement for NVIDIA A100 GPU, as it enables it to run 10x faster than the V100 on single-precision [23]. We compared training model performance to an ideal kernel microbenchmark. According to Fig.8, the ideal kernel microbenchmark uses up to 90% Tensor Core, while model training uses up to 35%. Thus Tensor Core is idle during most of the time of the model training, and there is much space for more arithmetic operator optimization to be adapted to the Tensor core's capabilities.

TABLE II
GPU PROFILING METRICS OF DIFFERENT MODELS ON SINGLE NODE

peak value	GPT-2	BERT-Large	ResNet50	VGG11
SM active (%)	85.1	88.2	93.3	86.3
Tensor active (%)	23.8	34.7	20.6	24.2
Mem active (%)	60	55.3	20.5	16.6
FP32 active (%)	14.8	23.5	70.7	52.1
NVLink TX/RX (GB/s)	33.1	13.8	3.9	19.4

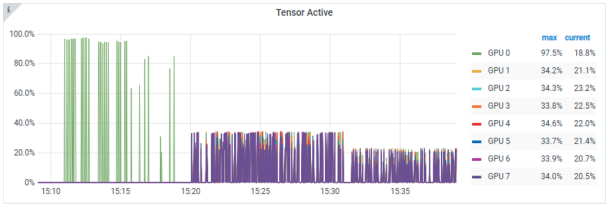


Fig. 8. Tensor active (%) in microbenchmark, BERT-Large and GPT-2

2) Underutilized GPU interconnection: As shown in Fig.9, we find another issue that NVLink bandwidth usage in distributed training is much lower than the specification. Meanwhile, we use a microbenchmark running NCCL all reduce operation with 8GB message size to validate the accuracy of measured results. Moneo's result is similar to the results reported by NCCL as 235GB/s. Though the peak NVLink bandwidth should be 300 GB/s per direction [24], it is no more than 35 GB/s for the model training of GPT-2, BERT-Large, ResNet50, and VGG11 on a single machine as shown in Table.II. According to Fig.10, NCCL performance is related to message size. Thus, the GPU interconnection is significantly underutilized, which may provide an opportunity to optimize the collective communication library.

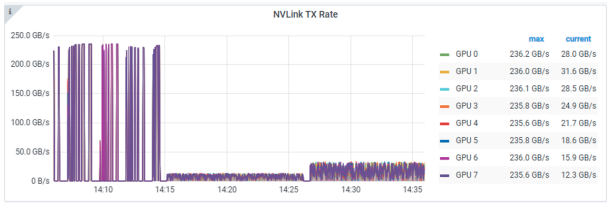


Fig. 9. NVLink bandwidth (GB/s) in microbenchmark, BERT-Large, GPT-2

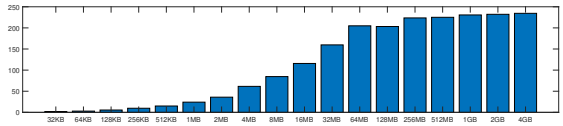


Fig. 10. NVLink bandwidth(GB/s) bandwidth of NCCL with message size from 32KB to 4GB

The results from underutilized GPU-related resource usage indicate the software may not be optimized enough for the workloads to fit hardware and point us in the direction for software stack optimization.

3) Diverse network requirements: We study the communication traffic and network bandwidth requirements of distributed workloads.

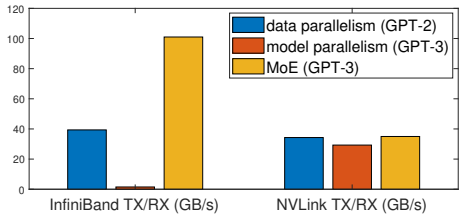


Fig. 11. Network and NVLink bandwidth of different distributed workloads

For data parallelism of GPT-2, InfiniBand bandwidth and NVLink bandwidth are similar and no more than 40 GB/s. This is because it uses ring all reduce to communicate for each GPU, so the bandwidth requirements between nodes and between GPUs inside the node are similar.

For model parallelism of GPT-3, the intra-node of model parallelism using tensor parallelism still needs high NVLink bandwidth of about 30 GB/s. However the InfiniBand bandwidth is less than 2 GB/s. Because pipeline parallelism only

requires sending activation tensor between layers, the inter-node network bandwidth requirement for model parallelism is very low.

For GPT-3 MoE model, NVLink bandwidth is about 35 GB/s for each GPU, and overall InfiniBand bandwidth measured is over 100 GB/s. Since there is all to all communication between experts in each GPU, the InfiniBand bandwidth requirements for 8 GPUs in the node should be 35*8 GB/s which means that MoE needs a very high network bandwidth requirement. We combine SM utilization and InfiniBand bandwidth to analyze. As shown in Fig.12, the calculation is partially idle, but the communication is always dense. Thus computation and communication cannot overlap, indicating peak network bandwidth provided has a significant effect on the MoE training performance.

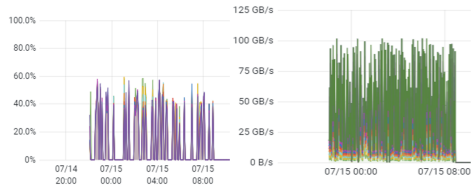


Fig. 12. SM utilization vs. InfiniBand bandwidth of MoE training

Different network bandwidth requirements show that while existing network configurations can support model parallelism, MoE requires higher performance and dedicated network architecture. Therefore, AI infrastructure should consider resource requirements of different workloads when designing system architecture for better efficiency and performance.

V. Conclusion

This paper introduces Moneo, a non-intrusive monitor for cloud-based AI infrastructure. Moneo intelligently collects several key architecture-level fine-grained resource usage metrics in real-time without touching users' data and workloads. Moneo can help to identify resource usage patterns and requirements of diverse AI workloads. Analyzing the results of typically distributed DNN workloads in production demonstrate that Moneo can effectively obtain valuable insights into the optimization space of software, including arithmetic operators and communication libraries, as well as more efficient hardware architecture design to meet the networking requirements of different workloads.

Acknowledgment

The authors would like to thank anonymous reviewers for their valuable comments. This research is supported by the National Natural Science Foundation of China under Grant Numbers 62072228, the Fundamental Research Funds for the Central Universities, the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Jiangsu Innovation and Entrepreneurship (Shuangchuang) Program.

References

[1] (2021) Cloud Computing Services | Microsoft Azure. [Online]. Available: https://azure.microsoft.com/en-us/

[2] (2021) Cloud Services - Amazon Web Services (AWS). [Online]. Available: https://aws.amazon.com/

[3] C. Li. OpenAI's GPT-3 Language Model: A Technical Overview. (2021). [Online]. Available: https://lambdalabs.com/blog/demystifying-gpt-3/

[4] M. Yu, A. G. Greenberg, D. A. Maltz, J. Rexford, L. Yuan, S. Kandula, and C. Kim, "Profiling network performance for multi-tier data center applications." in *NSDI*, vol. 11, 2011, pp. 5–5.

[5] F. Guo, Y. Li, J. C. Lui, and Y. Xu, "Dcuda: Dynamic gpu scheduling with live migration support," in *Proceedings of the ACM Symposium on Cloud Computing*, 2019, pp. 114–125.

[6] K. Zhou, Y. Hao, J. Mellor-Crummey, X. Meng, and X. Liu, "Gvprof: A value profiler for gpu-based clusters," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16.

[7] N. Developer. (2021) NVIDIA CUDA Profiling Tools Interface (CUPTI) - CUDA Toolkit. [Online]. Available: https://developer.nvidia.com/cupti

[8] (2021) Profiler User's Guide. [Online]. Available: https://docs.nvidia.com/cuda/profiler-users-guide/index.html

[9] (2021) NVIDIA Nsight Systems. [Online]. Available: https://developer.nvidia.com/nsight-systems

[10] Google. (2021) TensorBoard. [Online]. Available: https://tensorflow.google.cn/tensorboard

[11] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent, "Hpctoolkit: Tools for performance analysis of optimized parallel programs," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 685–701, 2010.

[12] N. Developer. (2021) NVIDIA Nsight Compute. [Online]. Available: https://developer.nvidia.com/nsight-compute

[13] ——. (2021) NVIDIA System Management Interface. [Online]. Available: https://developer.nvidia.com/nvidia-system-management-interface

[14] (2021) Syllo/nvtop. [Online]. Available: https://github.com/Syllo/nvtop

[15] G. Lu, W. Zhang, and Z. Wang, "Optimizing depthwise separable convolution operations on gpus," *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[16] G.-J. van den Braak, B. Mesman, and H. Corporaal, "Compile-time gpu memory access optimizations," in *2010 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*. IEEE, 2010, pp. 200–207.

[17] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, and C. Guo, "A generic communication scheduler for distributed dnn training acceleration," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 16–29.

[18] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

[19] (2021) Prometheus - monitoring system & time series database. [Online]. Available: https://prometheus.io/

[20] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[21] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[22] (2021) pytorch/fairseq: Facebook AI Research Sequence-to-Sequence Toolkit written in Python. [Online]. Available: https://github.com/pytorch/fairseq/#requirements-and-installation

[23] N. D. Blog. (2021) NVIDIA Ampere Architecture In-Depth. [Online]. Available: https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/

[24] (2021) NVLink & NVSwitch for Advanced Multi-GPU Communication. [Online]. Available: https://www.nvidia.com/en-us/data-center/nvlink/