Moneo: Monitoring Fine-grained Metrics Nonintrusively in AI Infrastructure

Yuting Jiang*, Yifan Xiong†, Lei Qu†, Cheng Luo†,
Chen Tian*, Peng Cheng†, and Yongqiang Xiong†

*Nanjing University †Microsoft Research

Abstract

Cloud-based AI infrastructure is becoming increasingly important, especially on large-scale distributed training. To improve its efficiency and serviceability, real-time monitoring of the infrastructure and workload profiling 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.

In this paper, 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.

This is a revised version of the symposium paper [23] presented in IEEE ICC 2022 originally.

1 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 [17], Amazon AWS [15] for flexibility and economy.

On 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 took OpenAI just over a year to double the capacity of GPT models from 1.5B to 175B [19]. The latest GPT-3 model would require 355 years and 4.6M dollars to train even with the single Tesla V100 cloud instance [25].

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 Figure 1(a). So how to improve the efficiency for AI infrastructure? There may be two main directions, improving performance and reducing cost. If the software stack is not optimized sufficiently, we can optimize and then improve the performance. If the hardware is over-provisioning resulting in unnecessary and expensive costs, we can use other cheaper hardware to reduce the cost. But firstly, it's necessary to know where the bottleneck is and what is underutilized. Fortunately, real-time monitoring and profiling are proved to be the effective approach empirically to help pinpoint inefficiencies [33].

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

First, instrumentation-based approaches can not be applied since they will interfere with tenants' workloads. Through experiment, we discovered that Nsight Compute [6] can increase the step time for ResNet50 [22] by more than 50% in experiment [7, 21, 35].

Second, the majority of fine-grained profiling tools require the collection of workload traces [6, 13, 14, 36]. Not only do those tools incur significant overhead, they also expose users to privacy risks.

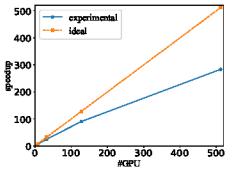
Third, some existing monitors for AI workloads are overly coarse-grained [8, 11]. Most tools can easily monitor GPU utilization, which is the percentage of time when a kernel is using GPU over the previous sampled period. However, they cannot obtain finer-grained information contained within GPUs. A GPU contains dozens of streaming multiprocessors

(SM), and GPU utilization cannot distinguish between only one SM or all SMs in use.

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 usages from hardware counters which introduce little overhead and do not interfere with tenants' workloads.

Besides, we collect metrics at a finer granularity to more accurately and effectively reflect resource usages. 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 Figure 1(b), the average GPU utilization for GPT-3 based mixture-of-experts (MoE) [16] is around 90%, while the average SM utilization is approximately 20%. Thus it can reflect the actual use of computing resources more effectively.

Moreover, a real-time monitor can provide additional benefits. Distributed DNN workloads are more vulnerable to failures from unreliable hardware and software. For instance, if there're network failures, the job may hang in long time. 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 task. A real-time monitor can enable users to check GPU status and stragglers in real-time, assist in online diagnosis for DNN workloads.



(a) Ideal vs. real GPT-2 FP32 speedup with #GPU increasing

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

Figure 1: Experiments of GPT Models

Nevertheless, real-time monitoring fine-grained metrics in AI Infrastructure 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 for AI workloads. Moneo demonstrates its effectiveness in production by its data from three representative distributed workloads, including data parallelism, model parallelism, and MoE [29]. 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.

We summarize our contributions as follows:

1. We design and implement a real-time non-intrusive fine-grained monitor system, Moneo, which has very little overhead, not touch user's data nor affect the performance of tenants' workloads. It is effective for cloud-based AI infrastructure.
2. We identify several key metrics for AI workloads, which make Moneo can effectively reveal computation and communication resource usage patterns and requirements of diverse AI workloads, hejp pinpoint inefficiencies and guide the design of future-proof hardware.
3. Our analysis on the data reported by Moneo shows shat Moneo can assist in get valuable insights help improve the efficiency of cloud infrastructure design and optimize the full software stack.

2 System Design

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 as a service. Moneo introduces limited overhead on target machines, does not instrument or interfere with AI workloads, or collect user-relevant data.

2.1 Architecture Overview

Figure 2 illustrates the system architecture of Moneo including three main components: metrics exporter, data collector, and analyzer.

Metrics Exporter The metrics exporters are running on each compute node and will query hardware or OS metrics exposed by hardware drivers or Linux kernel in short periods.

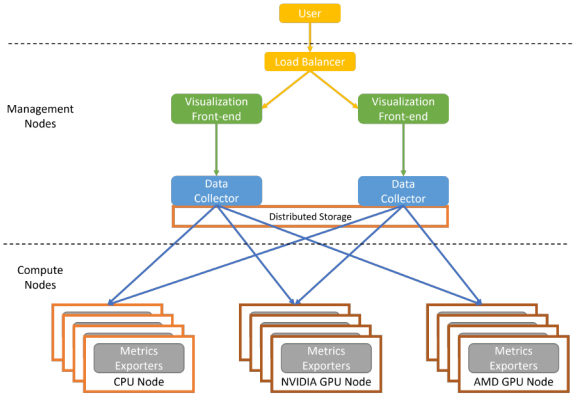


Figure 2: Moneo System Architecture

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 are, for instance, a GPU exporter, a network exporter, and a node exporter for basic operating system metrics such as CPU and disk related. Metric exporters consume very little CPU resources as background services and do not interfere with running workloads.

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.

Analyzer We define distinct Grafana dashboards [3] for various metric and analysis categories to enable users to easily observe insights generated by analyzers.

2.2 Data-driven Metric Selection

We focus on the metrics for NVIDIA / AMD GPU and NVIDIA Mellanox InfiniBand, mainly used for computation and communication during large-scale distributed AI workloads. We initially collected all counters available through the `nvidia-smi` [8] and GPU manager tools [6], and discovered that there are several hundreds. 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 Figure 3. First, some metrics are almost constant whatever the workloads are, so we cannot learn much from them. In this case, if the time series of a metric's variance is less than 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 Figure 3 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, for each two metrics, we normalize the distance of them by dividing the maximum value in the time series of these two metrics. So the distance of each two metrics is

$$\left(\sum_{t=1}^{T} \left| \frac{x_t - y_t}{\max_{t=1}^{T} \{x_t, y_t\}} \right|^2 \right)^{\frac{1}{2}} \quad (1)$$

where x_t and y_t are the metrics at timestamp t over the whole duration T. We only keep one of them in the group if the distance is less than threshold.

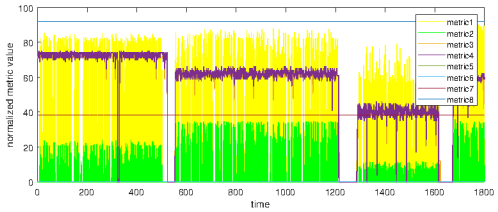


Figure 3: Examples of Metric Patterns

2.3 Adaptive Monitoring Frequency

One of the characteristics of AI workload is periodicity because it repeats the same operations in each training step. That indicates metric ranges 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 Figure 4(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 Figure 4(b), the peak performance will change over time. The sudden changes in metrics are unpredictable. So we dynamically adjust data collection frequency on the fly. 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

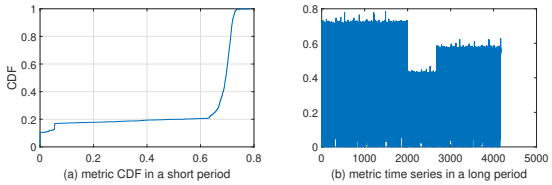


Figure 4: Metric Change in Short Time and Long Time

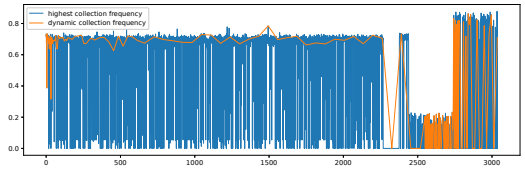


Figure 5: Adaptive and Highest Collection Frequency

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 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 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% 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 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 Figure 5, 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 highest frequency.

3 Implementation

3.1 Monitor Metrics

The collected metrics for GPU and InfiniBand are divided into six categories, as shown in Table 1. The first three are related to GPU computation, and the last three are used for inter-GPU, GPU-CPU, or inter-node communications. Those metrics are collected through monitoring APIs provided by NVIDIA or AMD GPU management tools [1, 12], and Linux sysfs interface. Other hardware like disk and CPU will be used for data I/O and processing during AI workloads training. We leverage existing metrics from Prometheus node exporter [2] to monitor them.

3.2 Deployment

Testbed Moneo has been deployed on 30 Azure Standard_ND96asr_v4 nodes, each of them is equipped with 96 AMD Epyc CPU cores, 900 GiB memory, 8x NVIDIA A100 SXM 40GB GPUs connected by 3rd-generation NVLink, and 8x 200 Gbps Mellanox HDR InfiniBand [18].

Workloads There are three typical workloads we profiled for large-scale distributed training:

1. *Data parallelism:* GPT-2 [28], BERT-Large [20], ResNet50 [22], and VGG11 [32] model training in single precision using PyTorch, on single node or multiple nodes.
2. *Model parallelism:* 175B GPT-3 [19] model training using Megatron-LM [31], on 196 GPUs. Tensor parallelism is applied to all GPUs inside one node, and pipeline parallelism is used to scale up across nodes.
3. *Mixture-of-Experts (MoE):* GPT-3 based MoE training using Fairseq [27], on 128 GPUs.

4 Evaluation

We first evaluate monitoring overhead, then show some valuable findings and insights from observed data from Moneo.

4.1 Monitoring Overhead

On 30 different nodes, we run distributed different training workloads with and without Moneo monitoring. The results indicate that the variance in training throughput with and without Moneo ranges from -1.28% to 1.8%. We also measure the CPU and memory overhead of Moneo. In the most of time, Moneo consumes less than 900 MB memory and around 3 CPU cores, which are less than 0.1% of the total memory and 3.5% of CPU on each Azure Standard_ND96asr_v4 node [18]. To conclude, Moneo introduces limited overhead on monitored nodes and does not interfere with the original workloads.

4.2 Insights from Observed Data

4.2.1 Underutilized Computation Resources

We study the computation resource utilization of various workloads 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.

Table 1: Metrics List and Descriptions

Category / Metric	Description
GPU Basics	Common health monitoring for a single GPU, including 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 fraction of cycles one SM was active, averaged over all SMs.
Tensor Active (%)	The fraction of cycles Tensor pipe was active, indicating utilization of Tensor Core.
FP64 / FP32 / FP16 Active(%)	The fraction of cycles FP64 / FP32 / FP16 pipe was active.
GPU Memory	GPU global memory, can be used for storing models and optimizers.
Mem Active (%)	The ratio of cycles when GPU memory interface is active sending or receiving data.
NVLink / xGMI	Used for high-speed inter-GPU communication inside one node.
NVLink TX/RX (GB/s)	The rate of transmitted or received data over NVLink for each GPU, in GB/s.
PCIe	Used for memory copy between GPU devices and host, including DtoH and HtoD.
PCIe TX/RX (GB/s)	The rate of transmitted or received data over PCIe for each GPU, in GB/s.
InfiniBand	Used for inter-node communication with very high throughput and very low latency.
InfiniBand TX/RX (GB/s)	The rate of transmitted or received data throughput InfiniBand, in GB/s.

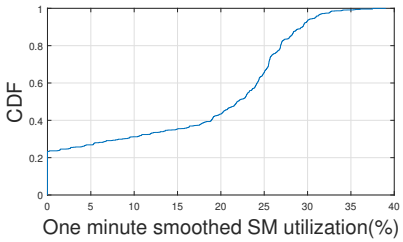


Figure 6: Smoothed SM utilization Per Minute CDF

Then, we compare the computation resource usage patterns for various data parallelism models on single node, as shown in Table 2. It's worth noting that Tensor Core usage in model training is relatively low. However, single precision Tensor Core is a critical improvement for NVIDIA A100 GPU, which makes it 10x faster than V100 on single precision [24]. We compare model training performance to an ideal GPU kernel micro-benchmark. According to Figure 7, the ideal GPU kernel micro-benchmark uses up to 90% Tensor Core, while model training uses up to 35%. Thus Tensor Core is idle during most of the time in model training, and there is much space for more arithmetic operator optimizations to be adapted by leveraging Tensor core's capabilities.

Table 2: Peak Value of GPU Metrics on Different Models

	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 (%)	60.0	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

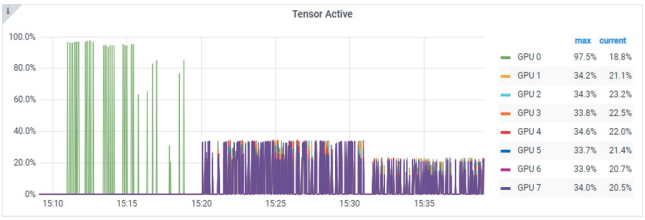


Figure 7: Tensor Active in Kernel Micro-benchmark, BERT-Large and GPT-2 Model Training, Respectively

4.2.2 Underutilized GPU Inter-connection

Figure 8 shows another issue that NVLink bandwidth in distributed training is much lower than its capacity. We run all-reduce micro-benchmark with 8 GiB message size in nccl-tests [4] to validate the accuracy of measured metrics. Moneo's result is similar to the 235 GB/s bus bandwidth reported by nccl-tests. Though the peak bandwidth for NVLink is 300 GB/s per direction [26], it is less than 35 GB/s during model training of GPT-2, BERT-Large, ResNet50, and VGG11 on single node, as shown in Table 2. According to Figure 9, the utilized bandwidth is highly related to message size. Only large sizes can saturate NVLink while model training usually uses dozens MiB for all-reduce. Thus, the GPU inter-

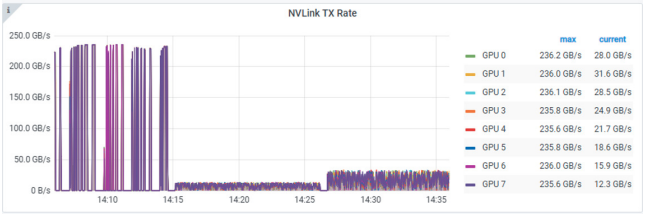


Figure 8: NVLink Bandwidth in Micro-benchmark, BERT-Large and GPT-2 Model Training, Respectively

connection is underutilized during model training, which may provide an opportunity to optimize the collective communication primitives.

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

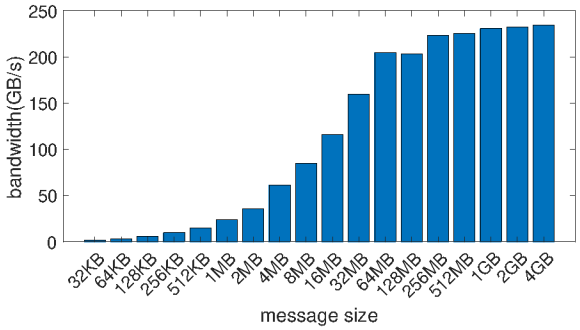


Figure 9: All-reduce Bus Bandwidth on NVLink

4.2.3 Diverse Network Requirements

Apart from single node workloads, we also study the inter-node communication and network bandwidth requirements of various distributed workloads:

1. *Data parallelism:* For GPT-2 model, InfiniBand and NVLink bandwidth are similar and less than 40 GB/s. The all-reduce communication bottleneck is in networking so NVLink bandwidth is limited.
2. *Model parallelism:* For 175B GPT-3 model, intra-node tensor parallelism can utilize around 30 GB/s NVLink bandwidth, while inter-node pipeline parallelism only sends activation across nodes, which requires extreme low bandwidth.
3. *Mixture-of-Experts:* For GPT-3 based MoE model, NVLink bandwidth for each GPU is around 35 GB/s, and total InfiniBand bandwidth measured is greater than 100 GB/s. MoE model uses all-to-all to dispatch data among GPUs, which require much higher networking bandwidth. Figure 11 shows the GPU and networking usage during MoE model training, the computation is partially idle while communication is always busy. Unlike all-reduce, all-to-all communication cannot overlap with computation because of data dependency, which makes peak network bandwidth more important for MoE model training.

Different network bandwidth requirements show that while existing network configurations can support model parallelism, MoE model 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.

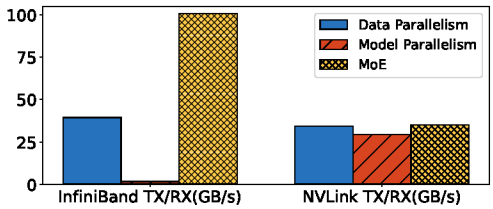


Figure 10: InfiniBand and NVLink Bandwidth of Proposed Distributed Workloads

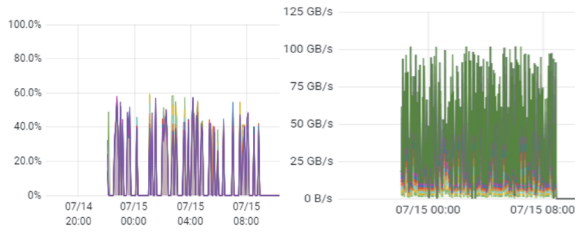


Figure 11: SM Utilization and InfiniBand Bandwidth during MoE Model Training

5 Discussion

Moneo aims to understand AI workloads patterns without interfering with GPU applications from an infrastructure perspective, so as to find out the bottlenecks and improve the overall resource utilization for AI infrastructure. Though Moneo has monitored some fine-grained metrics from existing hardware counters and can get some valuable findings, there still exist some possible improvements for future works in Meneo.

First, we still want to propose some new metrics which are also crucial for understanding AI workloads patterns and bottlenecks better but there might be no available approaches to collect them non-intrusively yet. These metrics include:

1. *SM and Memory Throughput*: overall compute and memory pipeline throughput in a GPU. SM and memory throughput can be used to understand whether the workload is computation or memory-bound on the current device, which helps in understanding the bottleneck of the workloads. For instance, the workload may be memory-bound if its memory throughput is high, and increasing memory bandwidth may be able to improve its training performance more.
2. *Per SM Activity*: the fraction of cycles one SM was active for each SM in the GPU. Moneo collects the SM Activity per GPU level, while it would be better to get that metric per SM level. SM activity for each SM could tell whether a workload really needs all SMs in one GPU, so that we can do better resource allocation or GPU sharing for workloads to improve the overall GPU utilization.
3. *L1TEX, LTS, and Shared Memory Throughput*: L1 or TEX cache, L2 cache, and shared memory throughput in the GPU. These three metrics can be used to check whether

there're contentions when computation and communication overlap, different types of computation kernels would have different patterns.

4. Besides above GPU metrics, *NVSwitch metrics* could also help us better understand whether there is contention in GPU interconnection which may help improve communication libraries.

In addition, we also plan to deploy Moneo on more models, different hardware, and larger clusters, to strengthen the data analysis results. By exploring the resource utilization patterns of different workloads in various AI systems, we can understand the characteristics of different AI workloads and learn what AI hardware is more suitable for them, which helps with better resource allocation and scheduling. By studying resource utilization in larger clusters, we can gain more valuable insights at the cluster level into more prevalent resource usage bottlenecks and underutilized resources, and discover trends in AI workload development, which can help on decision marking of AI infrastructure improvement and evolution.

6 Related Work

Commonly used lightweight monitor tools like `nvidia-smi` [8] and `rocm-smi` [11] provided by GPU vendors can monitor GPU utilization and memory usage on the GPUs. Other third-party tools like `nvtop` [9] are actually the wrapper of them. But non of them could collect fine-grained metrics.

The majority of fine-grained profiling tools require the collection of workload traces. Popular frameworks provide TensorBoard [13] for tracing the op-level execution timeline on the GPU. NVIDIA provides performance measurement tools ranging from the kernel to the application level including NVProf [10], Nsight Systems [7], and Nsight Compute [6] which will replay kernels, APIs, then gather workload traces. Academic efforts including TAU Performance System [30], HPCToolkit [14] and CUDABlamer [34, 36] also collect several useful performance metrics and scale to large HPC applications. Aforementioned profilers employ NVIDIA CUPTI API [5] which requires instrumenting the kernel in order to collect events and metrics.

Besides, there are some other methods instrumenting the executed codes and software stack. GVProf [35] instruments GPU memory instructions to profile value redundancies. DCUDA [21] tracks memory utilization by intercepting API calls and the function parameters. Both of them will incur significant overhead and expose users to privacy risks.

7 Conclusion

This paper presents 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 or workloads. Moneo can help to identify resource usage patterns and requirements of diverse AI workloads. Analyzing the results of typically distributed training workloads in production demonstrates 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

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] DCGM Library API Reference Guide - Data Center GPU Management Documentation. https://docs.nvidia.com/datacenter/dcgm/dcgm-api/. Accessed Feb 2022.

[2] Exporter for machine metrics. https://github.com/prometheus/node_exporter. Accessed Feb 2022.

[3] Grafana: The open observability platform. https://grafana.com/grafana/dashboards/. Accessed Feb 2022.

[4] NCCL Tests. https://github.com/NVIDIA/nccl-tests. Accessed Feb 2022.

[5] NVIDIA CUDA Profiling Tools Interface (CUPTI) - CUDA Toolkit. https://developer.nvidia.com/cupti. Accessed Feb 2022.

[6] NVIDIA Nsight Compute. https://developer.nvidia.com/nsight-compute. Accessed Feb 2022.

[7] NVIDIA Nsight Systems. https://developer.nvidia.com/nsight-systems. Accessed Feb 2022.

[8] NVIDIA System Management Interface. https://developer.nvidia.com/nvidia-system-management-interface. Accessed Feb 2022.

[9] NVTOP: NVIDIA GPUs htop like monitoring tool. https://github.com/Syllo/nvtop. Accessed Feb 2022.

[10] Profiler - CUDA Toolkit Documentation. https://docs.nvidia.com/cuda/profiler-users-guide/index.html. Accessed Feb 2022.

[11] ROCm Command Line Interface. https://rocmdocs.amd.com/en/latest/ROCm_System_Managment/ROCm-SMI-CLI.html. Accessed Feb 2022.

[12] ROCm Data Center Tool User Guide. https://github.com/RadeonOpenCompute/rdc/blob/roc-5.0.

x/docs/AMD_ROCm_Data_Center_Tool_User_Guide.pdf. Accessed Feb 2022.

[13] TensorBoard: TensorFlow's visualization toolkit. https://www.tensorflow.org/tensorboard. Accessed Feb 2022.

[14] Laksono Adhianto, Sinchan Banerjee, Mike Fagan, Mark Krentel, Gabriel Marin, John Mellor-Crummey, and Nathan R Tallent. Hpctoolkit: Tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience*, 22(6):685–701, 2010.

[15] Amazon. Cloud Services - Amazon Web Services (AWS). https://aws.amazon.com/. Accessed Feb 2022.

[16] Mikel Artetxe, Shruti Bhosale, Naman Goyal, Todor Mihaylov, Myle Ott, Sam Shleifer, Xi Victoria Lin, Jingfei Du, Srinivasan Iyer, Ramakanth Pasunuru, et al. Efficient large scale language modeling with mixtures of experts. *arXiv preprint arXiv:2112.10684*, 2021.

[17] Microsoft Azure. Cloud Computing Services | Microsoft Azure. https://azure.microsoft.com/en-us/. Accessed Feb 2022.

[18] Microsoft Azure. ND A100 v4-series - Azure Virtual Machines. https://docs.microsoft.com/en-us/azure/virtual-machines/nd-v4-series. Accessed Feb 2022.

[19] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[21] Fan Guo, Yongkun Li, John CS Lui, and Yinlong Xu. Dcuda: Dynamic gpu scheduling with live migration support. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 114–125, 2019.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] Yuting Jiang, Yifan Xiong, Lei Qu, Cheng Luo, Chen Tian, Peng Cheng, and Yongqiang Xiong. Moneo: Non-intrusive fine-grained monitor for AI infrastructure. In *2022 IEEE International Conference on Communications (ICC): SAC Cloud Computing, Networking and Storage Track (IEEE ICC'22 - SAC-02 CCNS Track)*, Seoul, Korea (South), May 2022. IEEE.

[24] Ronny Krashinsky et al. NVIDIA Ampere Architecture In-Depth. https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/. Accessed Feb 2022.

[25] Chuan Li. OpenAI's GPT-3 Language Model: A Technical Overview. https://lambdalabs.com/blog/demystifying-gpt-3/. Accessed Feb 2022.

[26] NVIDIA. NVLink & NVSwitch for Advanced Multi-GPU Communication. https://www.nvidia.com/en-us/data-center/nvlink/. Accessed Feb 2022.

[27] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*, 2019.

[28] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[29] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[30] Sameer S Shende and Allen D Malony. The tau parallel performance system. *The International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[31] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[33] Minlan Yu, Albert G Greenberg, David A Maltz, Jennifer Rexford, Lihua Yuan, Srikanth Kandula, and Changhoon Kim. Profiling network performance for multi-tier data center applications. In *NSDI*, volume 11, pages 5–5, 2011.

[34] Hui Zhang and Jeffrey Hollingsworth. Understanding the performance of gpgpu applications from a data-centric view. In *2019 IEEE/ACM International Workshop on Programming and Performance Visualization Tools (ProTools)*, pages 1–8. IEEE, 2019.

[35] Keren Zhou, Yueming Hao, John Mellor-Crummey, Xiaozhu Meng, and Xu Liu. Gvprof: A value profiler for gpu-based clusters. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[36] Keren Zhou, Mark W Krentel, and John Mellor-Crummey. Tools for top-down performance analysis of gpu-accelerated applications. In *Proceedings of the 34th ACM International Conference on Supercomputing*, pages 1–12, 2020.