A High-Performance I/O Framework for Accelerating DNN Model Updates Within Deep Learning Workflow

Jie Ye*
jye20@hawk.iit.edu
Illinois Institute of Technology
Chicago, IL, USA

Jaime Cernuda*
jcernudagarcia@hawk.iit.edu
Illinois Institute of Technology
Chicago, IL, USA

Bogdan Nicolae†
bnicolae@anl.gov
Argonne National Laboratory
Lemont, IL, USA

Anthony Kougkas*
akougkas@iit.edu
Illinois Institute of Technology
Chicago, IL, USA

Xian-He Sun*
sun@iit.edu
Illinois Institute of Technology
Chicago, IL, USA

1 EXTENDED ABSTRACT

Deep Learning (DL) has been widely used at a very large scale on supercomputing infrastructures in many scientific areas, such as climate modeling, physical simulations, virtual drug response prediction, cancer research, etc. The integration of Deep Learning and traditional HPC simulations has shown significant promise in accelerating scientific discoveries.

A DL workflow typically involves two phases: *training* and *inference*. In the training phase, a Deep Neural Network (DNN) model is trained by feeding a snapshot of data into various deep learning frameworks (e.g., TensorFlow and PyTorch). Once trained, the model is saved for subsequent use. In the inference phase, the trained model will be deployed and queried by various end-user applications. In order to offer real-time predictions for end-users, a wide range of inference serving systems, such as TensorFlow Serving [4], NVIDIA Triton [3] and Clipper [1], has been developed in recent years. Traditional DL workflow prefers using offline learning to learn the DNN model, in which the DNN model is only trained and deployed once and all subsequent inference requests are served by the same model instance. Yet, in practice, modern scientific AI applications often operate in a dynamic environment where data are continually changing over time. They require the DNN models to be constantly updated with fresh data. In such scenarios, Offline learning is infeasible because it needs to retrain the DNN model from scratch when enough new data are available to keep the model fresh. Retraining a large and complex model from the beginning is slow due to the constant accumulation of training data. To accommodate shifts and new data patterns, a method known as Continuous Learning has been proposed [2]. It allows a DNN model to be continuously (re)-trained on a data stream using various online techniques (e.g., merging rehearsal of past training data with new data). Continuous Learning also implies continuously updating the deployed model residing in the Inference Serving System, increasing the frequency of model updates between scientific AI applications (referred to as producers) and Inference Serving Systems (referred to as consumers).

2 MOTIVATION AND CHALLENGES

2.1 Motivation

However, producers and consumers are often linked via a model repository, a storage system used for sharing DNN models, including PFS, DFS, Database, Amazon S3, and Google Cloud(Figure 1).

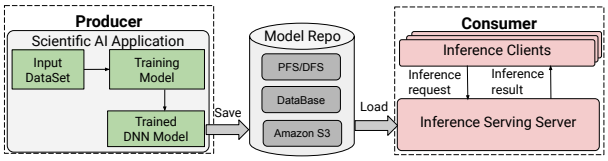


Figure 1: Coupling of training and inference through PFS

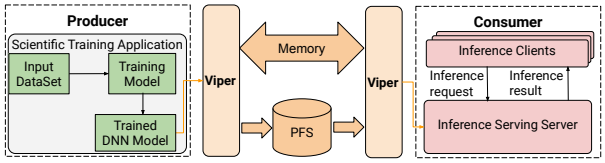


Figure 2: Coupling of training and inference through both Memory and PFS

This coupling method has two primary limitations. First, it results in significant latency for model updates due to the model repository bottleneck. PFS has limited I/O bandwidth and may experience severe performance degradation due to highly concurrent random accesses to small files — a typical access pattern in deep learning. Second, consumers may not promptly discover the new/updated model, further exacerbating the end-to-end latency. Existing Inference Serving Systems usually employ a static fixed-interval polling method to monitor changes in the model repository. Infrequent polling might not promptly detect changes, whereas polling at high frequencies can significantly burden the PFS since each poll may require accessing the file's metadata. Thus, to avoid the PFS bottleneck and speed up the model update latency, there is a growing need of building a direct communication channel for model updates between producers and consumers.(Figure 2).

2.2 Challenges

Although the support of the direct communication channel opens new opportunities, it also increases the complexity of the system and adds new challenges. One challenge lies in deciding the location of acquiring the DNN model for consumers. A trained DNN model could be cached in multiple alternative places on the producer when performing checkpointing, including GPU, DRAM, local storage device, or PFS. Thus, a consumer can retrieve the required model from either the remote GPU/DRAM of another node or the PFS. However,

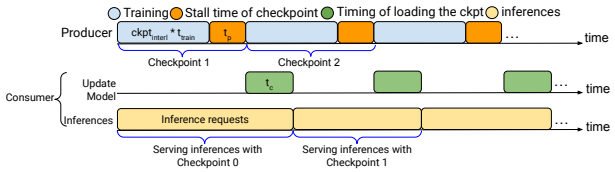


Figure 3: Impact of model update frequency on training and inference

the workloads of different data transfer strategies may be different at the same point. Therefore, it is essential and challenging for a consumer to intelligently select an optimal data transfer strategy for delivering the model state between producers and consumers. Another challenge is deciding the model update frequency between producers and consumers. Model update frequency also refers to the model checkpoint interval, which is defined as the number of training iterations between two model checkpoints. Figure 3 illustrates that the frequency of model updates may affect both the training and inference performance. Frequently model updates increase the training time due to the frequent interruption caused by saving a model checkpoint on the producer side. Conversely, infrequent model updates may lead to inference serving delay or inaccurate inference results since the Inference Serving System is serving the inferences with an outdated version model. Thus, it is challenging to balance the trade-off between training performance and inference performance because it may require dynamically adjusting the model update frequency during runtime.

3 DESIGN AND IMPLEMENTATION

To overcome limitations, we design an I/O framework, Viper, to accelerate DNN model delivery and discovery within a DL workflow. Figure 4 illustrates its high-level architecture design, which is centered around the following principles:

- Leveraging **a novel cache-aware model handler** to deliver the model through different data transfer strategies to reduce model update latency.
- Utilizing **a lightweight pub-sub notification mechanism** to reduce model discovery latency on the consumer side.
- Creating **an intelligent inference performance predictor** to dynamically find an optimal model update frequency during running.

4 RESULTS

Testbed and Application: All the tests were conducted on the Polaris [5] Supercomputer at Argonne National Laboratory. Each node is equipped with four NVIDIA A100 GPUs (160GB HBM2 Memory), 512 GB of DDR4 Memory. The supporting storage file system is Lustre. Application CANDLE-NT3, a representative DNN model in the CANDLE project, is used to evaluate the I/O framework.

Analysis: 1) Latency Anatomy Of Each Stage: In this experiment, we evaluated the latency of each stage during the model update through three data transfer strategies: GPU-to-GPU, DRAM-to-DRAM, and PFS, as shown in Figure 5(a). One producer and one consumer are deployed on two separate nodes. The producer and consumer are running in parallel. The model size is about 1700MB. From Figure 5(a), we observed that: for GPU, transferring data takes

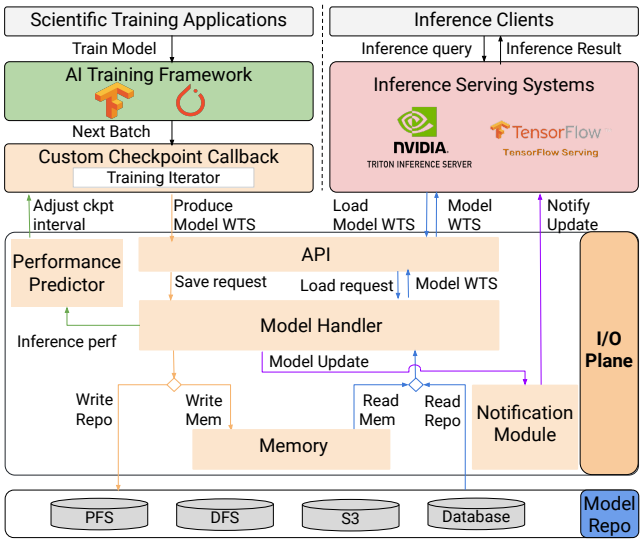
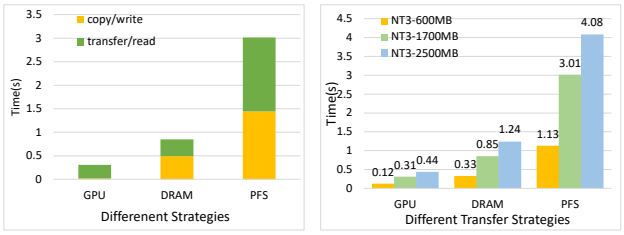


Figure 4: High-Level Design Architecture



(a) Latency anatomy of each stage (b) End-to-end model update latency

Figure 5: Model update latency for different strategies

the most time while the time of copying data is very mall; for DRAM, copying data takes more time because copying model on DRAM involves copying the tensors from GPU to CPU and converting tensors to NumPy array; for PFS, the write and read latency is almost the same.

2) End-to-end Model Update Latency Analysis: This test is to illustrate the variation in end-to-end model update latency when employing different data strategies and to explore how this latency evolves as the model size increases for a specific data transfer strategy. We tested three different sizes of CANDLE-NT3 models: 600MB, 1700MB, and 2500MB. Figure 5(b) shows that GPU-to-GPU has the lowest model update latency among the data transfer strategies. For a given data transfer strategy, the model update latency increases linearly when increasing the model size.

5 CONCLUSIONS

In this study, We designed and implemented an I/O framework to accelerate model delivery and discovery within a deep-learning workflow. The preliminary results indicate that transferring the DNN model via a direct communication channel can reduce the end-to-end latency by 10X for GPU and 3.5X for DRAM when compared to PFS In future work, we need to focus on building and evaluating the performance predictor. In addition, we also need to create an algorithm that can intelligently select a data transfer strategy based

on the current workloads of producers and consumers, which is a significant challenging optimization problem.

ACKNOWLEDGMENT

This work is supported by the National Science Foundation under Grant no. NSF CSSI-2104013.

REFERENCES

[1] Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System.. In *NSDI*, Vol. 17. 613–627.

[2] Jie Liu, Bogdan Nicolae, Dong Li, Justin M Wozniak, Tekin Bicer, Zhengchun Liu, and Ian Foster. 2022. Large Scale Caching and Streaming of Training Data for Online Deep Learning. In *Proceedings of the 12th Workshop on AI and Scientific Computing at Scale using Flexible Computing Infrastructures*. 19–26.

[3] NVIDIA. 2022. *NVIDIA Triton Inference Server.* https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/user_guide/architecture.html

[4] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. 2017. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139* (2017).

[5] Polaris. 2023. *ALCF Polaris Supercomputer.* https://www.alcf.anl.gov/polaris