SurrogateTrain: Drastically Improving Performance of Data Loading for Training Scientific Surrogate Models

Baixi Sun
Indiana University
Bloomington, IN, USA
sunbaix@iu.edu

Xiaodong Yu
Argonne National Laboratory
Lemont, IL, USA
xyu@anl.gov

Kamil A. Iskra
Argonne National Laboratory
Lemont, IL, USA
iskra@mcs.anl.gov

Dingwen Tao
Indiana University
Bloomington, IN, USA
ditao@iu.edu

1 MOTIVATION

Specially designed deep learning surrogate models draw more and more attention in scientific applications. For instance, PtychoNN [1] is a surrogate model for X-ray ptychography image reconstruction, which substitutes for high-computation-cost phase retrieval algorithms. CosmoFlow [3] is also provided as a surrogate model to perform cosmological parameter predictions, which substitutes for physical simulations to abstract the dark matter's features. Such surrogate models rely on the enormous volume of training data to lower the approximation variance and thus improve the generality of the model. To train surrogate models with large datasets, the data-parallel mechanism is utilized on high-performance computing (HPC) clusters to distribute the workload on multiple devices such as graphics processing units (GPUs). More specifically, each node holds a disjoint subset of the shuffled dataset and has a full replicate of the model to train [4].

Technically, due to the problem size, the training process requires fetching the large-size dataset stored on the parallel filesystem (PFS). Additionally, due to the cold-start data-loading policy—the permission to access a node-local SSD is granted only on job allocation—fetching from PFS is required on a training start. In principle, data loading from PFS to compute-ready GPU memory could be expensive as it involves network activity and multi-layer data loading (e.g., PFS to node-local SSD, SSD to host memory, host memory to GPU memory). Therefore, data loading is the performance bottleneck in training, and the optimization of data loading can pragmatically benefit the overall distributed training performance. Based on the benchmark, we observe that data I/O dominates the training epoch time at a ratio of 81%. Our result is consistent with previous work [2, 6]—data I/O takes up 90% of training time.

2 LIMITATIONS OF THE STATE OF THE ART

Pytorch DataLoader [5] provides the prefetch option to overlap the I/O time with the computation time. This improved the end-to-end training performance to some extent. However, when the data loading time dominates the iteration time the GPUs are idling, waiting for the batch of data to be loaded. Yang et al. [7] proposed a locality-aware loading strategy that utilizes an in-memory buffer to store the data loaded at the first iteration on each node, then schedule the inter-node communication to exchange the data samples. However, it introduces communication overhead to reduce data loading time for each iteration. Zhu et al. [8] proposed DeepIO that provided a shuffle strategy based on the RDMA (Remote Direct Memory Access). However, their work tradeoff training accuracy to reduce the data I/O overhead. Dryden et al. [2] proposed prefetching and buffering strategy that utilizes the multi-layer architecture of the HPC cluster (i.e. Parallel Filesystem, Node Local SSD, and Memory). However, scheduling which data sample to be buffered by predicting the distribution of data samples on each node is not always accurate.

3 KEY INSIGHTS

We hereby present an efficient data loading methodology SurrogateTrain for distributed training (data-parallel based scheme) of surrogate models. Specifically, the design of SurrogateTrain is based on four key observations.

Data Reuse and Epoch Order. 1). The data reuse rate varies when re-arranging the order of epochs, thus we can schedule the order of the shuffle list to maximize the data reuse. (2). The shuffled index list can be generated ahead of training instead of during runtime. Here, the epoch order is defined as the order of using the shuffled index lists, and the shuffled index lists are a set of dataset indices lists generated from random shuffling operations. Each shuffled index list indicates the data samples' access order for an epoch.

Data Reuse and Data Locality. We are inspired by the fact that changing the access order within a global batch does not change the training result after synchronization [7]. Thus, we can actually adjust the node-to-sample mapping to achieve locality for data reuse.

Load Balance and Computation Balance. In our experiment we observe that load balance dominates the training performance when the computation is fast. Thus, we can trade computation balance for load balance to reduce the overall training time in distributed training.

Read Small Files and Read in Chunks. We observe that reading small files at a time from an HDF5 dataset is slow compared to reading in chunks. Thus, we design the aggregated chunked loading in SurrogateTrain to load the data more efficiently compared to PyTorch DataLoader.

4 KEY RESULTS AND CONTRIBUTIONS

Our key contributions are summarized as follows: (1) We perform extensive benchmarking on a typical scientific surrogate model and identify the major bottleneck of the training process. (2) We propose an offline scheduling optimization that rearranges the access order of the data samples to increase the data reuse. (3) We propose runtime strategies that determine the time to evict in-buffer data and schedule aggregated chunked loading. (4) We evaluate SurrogateTrain on the scientific surrogate model and demonstrate that SurrogateTrain reduces the amount of data loaded by 6.7× and achieves up to 4.7× speedup in data loading.

REFERENCES

[1] Mathew J Cherukara, Tao Zhou, Youssef Nashed, Pablo Enfedaque, Alex Hexemer, Ross J Harder, and Martin V Holt. 2020. AI-enabled high-resolution scanning

coherent diffraction imaging. *Applied Physics Letters* 117, 4 (2020), 044103.

[2] Nikoli Dryden, Roman Böhringer, Tal Ben-Nun, and Torsten Hoefler. 2021. Clairvoyant prefetching for distributed machine learning I/O. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.

[3] Amrita Mathuriya, Deborah Bard, Peter Mendygral, Lawrence Meadows, James Arnemann, Lei Shao, Siyu He, Tuomas Kärnä, Diana Moise, Simon J Pennycook, et al. 2018. CosmoFlow: Using deep learning to learn the universe at scale. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 819–829.

[4] Shuo Ouyang, Dezun Dong, Yemao Xu, and Liquan Xiao. 2021. Communication optimization strategies for distributed deep neural network training: A survey. *J. Parallel and Distrib. Comput.* 149 (2021), 52–65.

[5] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[6] Sarunya Pumma, Min Si, Wu-Chun Feng, and Pavan Balaji. 2019. Scalable deep learning via I/O analysis and optimization. *ACM Transactions on Parallel Computing (TOPC)* 6, 2 (2019), 1–34.

[7] Chih-Chieh Yang and Guojing Cong. 2019. Accelerating data loading in deep neural network training. In *2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. IEEE, 235–245.

[8] Yue Zhu, Fahim Chowdhury, Huansong Fu, Adam Moody, Kathryn Mohror, Kento Sato, and Weikuan Yu. 2018. Entropy-aware I/O pipelining for large-scale deep learning on HPC systems. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 145–156.