Studying Storage Systems for ML workloads

Preetham Kikkeri, Ranjitha Kosgi, Varun Kaundinya

Problem Statement

With increasing compute power of GPUs, dataset loading and pre-processing is going to be a significant issue during ML training. Feeding training data fast enough to effectively keep the accelerator utilization high is difficult. Further, different ML workloads have different storage requirements and access patterns. NLP models store texts, image models need images, embedding based models need numerical arrays etc. A storage system which might work well for a particular workload might perform poorly with a different workload. In this project, we try to benchmark the performance of 2 storage systems - **Minlo** and **HDF5**, for different ML models. Minlo is a high performance object storage system, while HDF5 is an open source file format that supports, large, complex, heterogeneous data.

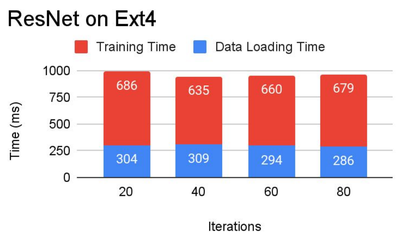
Related work

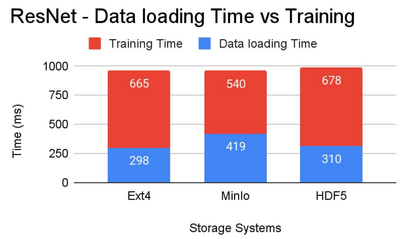
- ***Storage Benchmarking with Deep Learning Workloads*** **-** explores the impact of different parameters, including storage location, storage disaggregation granularity, access pattern, and data format. However, the optimal storage system is workload specific.
- Our project benchmarks the performances of storage systems **Minlo** and **HDF5** for chosen ML models - ResNet-50 (Image), BERT (Text) and possibility DLRM (.npz) using Pytorch Dataloader.
- ***Image Storage systems for ML*** - Efforts have been made on I/O profiling and optimization for ML workloads. Here image storage systems is in HDF5 format for training deep neural networks.
- ***Comparative Performance Analysis*** - *This* study focused on studying and tuning dataloaders for different dataloaders.

Performance Evaluation

Experimental Setup: Architecture: x86_64; GPU(s): NVIDIA T4; CPU(s): 4; Thread(s) per core: 2; Core(s) per socket: 4; Socket(s): 1
Dataloader parameters tuned: Prefetch - disabled. Different number of workers, threads & shuffling.

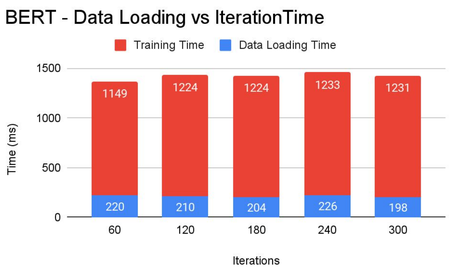
ResNet - Trained ResNet for 80 iterations using all three storage backends - Ext4, Minlo, HDF5 with batch size as 8 and loading images sequentially.

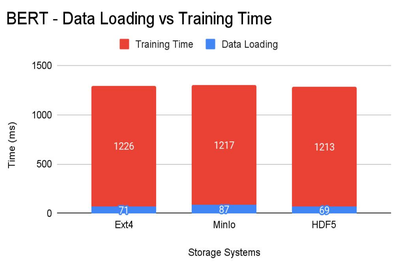


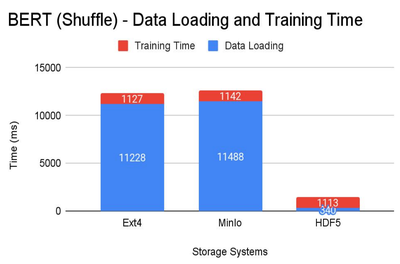


File format	Size (Bytes)
Original Image	255515
Minio object	255515
HDF5	**3329600**
HDF5 - gzip compression (4)	247014
HDF5 - gzip compression (9)	227569

BERT - Trained BERT for 300 iterations using all three storage backends - Ext4, Minlo, HDF5 with batch size as 16 and experimented loading input with & without shuffling.





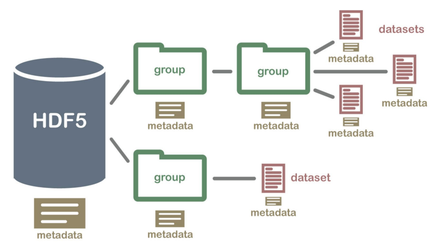


Observations

- ResNet - Ext4 and HDF5 have a similar performance while Minio has a higher load time.
- The data loading time consumes a significant portion of the iteration time.
- BERT (without shuffling) - All storage systems have a similar performance
- BERT (with shuffling) - Ext4 and Minio perform poorly while HDF5 performs considerably better as we can selectively read lines without loading the entire file.
- Increasing the number of workers to 2 decreased the data loading time from 300ms to 3ms across models. This is due to overlapping of fetch and compute.
- For BERT, increasing the batch size beyond 20 lead to GPU going out of memory and similarly for a batch size beyond 16 for ResNet. In BERT instead of storing lines, we store the embedding objects.`
- Memory footprint - An image stored as a HDF5 file takes more than 10X space than the original image, unless we store the images in binary format or compress them.

Future Work

- We will try out different ways to structure the data within a HDF5 file. For example storing each line as a dataset vs storing multiple lines in a dataset.



- Evaluating the performance for DLRM model.
- Fetching a range of images at a time
- Experimenting with compression techniques in HDF5

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON