Studying Storage System Backends for ML workloads

Benita Kathleen Britto, Hemalkumar Patel, Reetuparna Mukherjee, Group 13

1 Introduction

With ML powering a plethora of applications across a wide range of domains, there is a massive need for high performance systems capable of processing large datasets, ranging up to petabytes in magnitude. Over the years, with the massive improvement in compute power with GPUs and ASICs, and newer hardware like TPU and GraphCore, the storage layer has become a bottleneck when it comes to ML workloads. These jobs need to feed input training data fast enough to fully utilize the power of their compute units. Another challenge stems from the ubiquitous nature of ML systems and the corresponding variety of workloads - text, images, video, graph, etc. each of which have different compute and disk storage bandwidth requirements.

We focus on the storage aspect of ML workloads, particularly, the storage backend and the dataloaders to support the storage backend. TileDB [11] and TensorStore [10] have emerged as efficient storage solutions for multi-dimensional arrays. Moreover, RocksDB [8] is a key-value store that provides faster IO due to the underlying LSM trees. It is good for inserts but not the best for random reads. TileDB and TensorStore boast about the functionalities around ML workloads, while RocksDB is a generic key-value store. We don't see comparative benchmarking results for these three datastores and hence, explore these storage solutions in our project. With respect to dataloaders, previous work such as Ofeidis et al. [21] provide a comparative study of PyTorch [7], Torchdata [12], Hub [4], FFCV [3], Webdatasets [14], Squirrel [9], and Deep Lake [17]. They use Image Classification and Object Detection type of ML workload. FFCV [3] compares PyTorch [7], Webdataset [14] and DALI [1] for ImageNet. However, they do not include other popular workload types such as graph and NLP. We think that including them could potentially show different trends.

In our project, we plan to assess existing storage systems, namely TileDB, RocksDB and TensorStore. We intend to benchmark these storage systems for different workloads. We would be using text, image and graph datasets with the objective to analyze read throughput for different kinds of data. Existing literature shows that dataloaders seem to have a significant impact on improving latency [21] [20] [18]. We aim to evaluate these systems against each of these workloads and derive inferences as to why a certain system may or may not be best suited for a particular workload. Another area of focus for the study is to understand and evaluate how these different storage mechanisms scale.

Through our experimentation, we want to determine if there are other storage systems that are better suited for ML workloads. We plan to study the effect of preprocessing in the databases (push-down operation) and compare it with the ext4 file system as our baseline. We plan to use a single dataloader across the three storage systems to be consistent while benchmarking. We expect to see different trends for different ML workloads across the storage sytems.

2 Related Work

Existing literature explores various arenas for improving performance of ML processing systems and frameworks ranging across caching capabilities and IO optimizations.

Caching: Quiver [18] achieves cache reuse across multiple jobs securely. As a result, even when only a small part of data fits in cache, Quiver improves performance by using substitutable hits and co-operative miss handling to co-ordinate I/O access across multiple jobs. It uses a benefit-aware strategy in its placement thereby using the cache frugally and prioritizing jobs that benefit the most. Distributed caching in the cluster context has been explored more broadly in the analytics community. To extract the best performance from a query, Pac-Man [15] explored co-ordinated caching of data across different workers of an analytics job. Quartet [16] explores intelligent scheduling of big data jobs to maximize cross-job sharing of cached data.

I/O optimization: DeepIO [23] primarily features an I/O pipeline that utilizes several novel optimizations (RDMA -assisted in-situ shuffling, input pipelining, and entropy-aware opportunistic ordering) and additionally provides a portable storage interface to support efficient I/O on any underlying storage system. Crail [22] is a fast multi-tiered distributed storage system for high-performance network and storage hardware to deliver user-level I/O.

Data Stall [20] analyzes how data stalls affect the data pipeline and DNN training. Their analysis suggests that

data stalls take away the benefits of faster GPUs, even on ML optimized servers like the DGX-2. The modern ML frameworks like PyTorch and TensorFlow utilizes the state-of-the-art dataloaders like Dali, which reduce prep stalls using GPU-accelerated data pre-processing. Even then they are inefficient in their use of memory and CPU resources.

Dataloaders: Ofeidis et al. [21] go over the existing libraries and the dataloaders they offer and compares their performance against three datasets CIFAR-10 (can fit in memory), CoCo, RANDOM (created by the authors by augmenting CIFAR-10). Using frameworks like PyTorch, Torchdata, Hub, FFCV, WebDatasets, Squirrel, and Deep Lake, they show that the dataloaders provided by a certain framework are not optimized for all workloads when run on limited resources. Metrics collated over the training data, like number of processed data points (images) per second, total running time and data loader initialization, and further tuning the dataloader on hyperparameters like batch size, sampling approach, collating/padding to link batches, prefetching, number of workers, number of GPUs show the performance of dataloaders varies based on the workload. They also test when data is stored remotely and gather that remote storage does not hamper performance by a lot. Storing data in a remote storage environment is beneficial as the data setup becomes more streamlined. LBANN, on the other hand, uses node-local storage devices to store dataset, but not all clusters feature the expensive fast speed node-local storage devices [19].

To the best to our knowledge, the related work focuses on improving the data pipelines and stalls, and do not explore the impact of the underlying storage systems. Therefore, we plan to understand how underlying storage layer can impact the performance of ML tasks under different types of workloads.

3 Timeline and Evaluation Plan

We plan to experiment with the following datasets:

- ImageNet [5] for images
- Twitter [13] for text (tweet sentiment detection)
- FB15K [2] for graph
- MovieLens 25M Sparse dataset [6] for recommendation system

We will be using CloudLab with a single GPU as our infrastructure setup.

Overall our timeline is:

- 31st Oct - 7th Nov: Initial code setup
- 7th Nov - 21st Nov: Integration to datastores
- 21st Nov - 28th Nov: Benchmarking Initial
- 28th Nov - 5th Dec: Benchmarking Final
- 5th Dec - 15th Dec: Wrap-up (Poster Presentation, Report)

References

[1] Dali, https://docs.nvidia.com/deeplearning/dali/.

[2] FB15K Dataset, https://paperswithcode.com/dataset/fb15k.

[3] FFCV: Train machine learning models fast, https://ffcv.io/.

[4] Hub: A dataset format for ai. a simple api for creating, storing, collaborating on ai datasets of any size streaming them to ml frameworks at scale.

[5] ImageNet dataset, https://www.image-net.org/.

[6] MovieLens 25M Dataset, https://www.kaggle.com/datasets/garymk/movielens-25m-dataset.

[7] Pytorch: An open source machine learning framework that accelerates the path from research prototyping to production deployment.

[8] RocksDB: A Key-Value store, http://rocksdb.org/.

[9] Squirrel: A python library that enables ml teams to share, load, and transform data in a collaborative, flexible, and efficient way.

[10] Tensorstore is a library for efficiently reading and writing large multi-dimensional arrays.

[11] TileDb: Securely manage all data assets and supercharge your analytics, data science and machine learning with a universal database, https://github.com/tiledb-inc/tiledb-py.

[12] Torchdata: A prototype library of common modular data loading primitives for easily constructing flexible and performant data pipelines.

[13] Twitter Dataset, https://www.kaggle.com/datasets/kazanova/sentiment140.

[14] Webdataset: Webdataset is a pytorch dataset (iterabledataset) implementation providing efficient access to datasets stored in posix tar archives and uses only sequential/streaming data access.

[15] G. Ananthanarayanan, A. Ghodsi, A. Warfield, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica. {PACMan}: Coordinated memory caching for parallel jobs. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 267–280, 2012.

[16] F. Deslauriers, P. McCormick, G. Amvrosiadis, A. Goel, and A. D. Brown. Quartet: Harmonizing task scheduling and caching for cluster computing. In *8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16)*, 2016.

[17] T. A. G. L. R. F. T. H. I. D. H. M. H. T. S. I. Hambardzumyan, S. and Buniatyan. Deep Lake: a lakehouse for deep learning.

[18] A. V. Kumar and M. Sivathanu. Quiver: An informed storage cache for deep learning. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 283–296, 2020.

[19] N. Maruyama. Scalable distributed training of large neural networks with lbann. 2019.

[20] J. Mohan, A. Phanishayee, A. Raniwala, and V. Chidambaram. Analyzing and mitigating data stalls in dnn training. *arXiv preprint arXiv:2007.06775*, 2020.

[21] I. Ofeidis, D. Kiedanski, and L. Tassiulas. An overview of the data-loader landscape: Comparative performance analysis. *arXiv preprint arXiv:2209.13705*, 2022.

[22] P. Stuedi, A. Trivedi, J. Pfefferle, R. Stoica, B. Metzler, N. Ioannou, and I. Koltsidas. Crail: A high-performance i/o architecture for distributed data processing. *IEEE Data Eng. Bull.*, 40(1):38–49, 2017.

[23] Y. Zhu, F. Chowdhury, H. Fu, A. Moody, K. Mohror, K. Sato, and W. Yu. Multi-client deepio for large-scale deep learning on hpc systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC 2018)*, 2018.