One Does Not Simply Cache: A Study on Distributed Caches for AI Training in HPC

Workshop Paper

André Ferreira, Gonçalo Sousa, Janki Bhimani*, Raju Rangaswami*, Amit Ruhela+, John Cazes+, Ricardo Macedo, Cláudia Brito, João Paulo

INESC TEC & University of Minho **Florida International University* *+TACC & UTAustin*

***Abstract*—This paper presents the first study on cache solutions for Distributed Deep Learning training in HPC centers. We categorize these according to their intrusiveness, design, and data placement strategies. Further, through an empirical evaluation on real supercomputers (Deucalion and Vista), we measure the impact of caching and checkpointing data into different storage sources (*i.e.*, parallel file system, local storage, and remote nodes).**

Results show that choosing the optimal caching design is dependent on the targeted infrastructure, the storage resources used for caching data, and the network protocols. Based on these, we outline design trade-offs and propose future directions for co-designing cache systems within HPC infrastructures to more effectively support Distributed Deep Learning workloads.

***Index Terms*—Distributed Deep Learning, Distributed Caches, Storage I/O, High-Performance Computing, Parallel File System**

I. Introduction

Machine Learning (ML) is a powerful tool to solve crucial problems in distinct areas such as image classification, natural language processing, and speech recognition. Its wide adoption, along with the demand for more accurate and generalizable models, has driven the growth of both model and dataset sizes, especially in Deep Learning (DL) and Large Language Models (LLMs) [1].

With this increase in scale and complexity, Distributed Deep Learning (DDL) has emerged as an alternative solution to better leverage multiple compute nodes, parallelizing the computation and I/O required to train large models. In parallel, given the high demand for specialized hardware (*e.g.*, GPUs), a large number of nodes, and storage capacity, High-Performance Computing (HPC) centers have been the preferred choice for running these training workloads [2].

However, HPC centers typically follow a shared storage architecture, meaning multiple user workloads store and fetch data from a shared Parallel File System (PFS). In the specific case of DDL training workflows, multiple ML instances, deployed on different compute nodes, simultaneously read the dataset and save (*i.e.*, checkpointing) their model states to the PFS. These concurrent accesses create significant contention at the PFS, degrading its performance and potentially rendering the file system and the supercomputer unavailable [3].

Challenges and Related Work. Previous research work has been addressing the PFS contention generated by large-scale DDL training workloads, by leveraging both volatile and non-volatile (*i.e.*, local disk devices) memory available at each compute node [4, 5]. These resources are used for caching parts of the ML datasets, reducing the number of accesses made to the shared file system when training AI models.

However, existing solutions differ in multiple aspects, namely: *i)* on their intrusiveness to user workflows, *ii)* on the choice of which data to cache; *iii)* how data is distributed across local resources of each node; and *iv)* even on the cache eviction policies. These differences, which are not trivial to reason about, have not been explored in previous literature, which is mostly centered around surveying parallel I/O workloads and different I/O optimizations for ML workloads in HPC [1, 6]. Further, they depend on assumptions made regarding DDL workloads and the infrastructure they utilize.

Thus, the main contribution of this paper is a consolidation of the literature, complemented with empirical experiments, on the key characteristics of caching solutions for DDL workloads running at HPC supercomputers. Our paper is the first to:

i) make a systematization of key characteristics found in current research work, including intrusiveness, data placement, distribution and granularity, and eviction policies.

ii) complement the literature review with empirical results quantifying how the characteristics of real HPC infrastructures, namely the Deucalion and Vista supercomputers, impact the design of caching for DDL.

II. Background

Modern HPC centers are composed of multiple compute nodes, which are organized within racks and different subsystems, depending on their characteristics (i.e., CPU nodes or GPU nodes). For instance, Deucalion [7] is currently composed of 2132 CPU nodes and 33 GPU nodes, while systems like Vista [8] have 256 CPU and 608 GPU nodes. Other systems, focused on efficiently supporting AI training workloads, like ABCI [9], are equipped with 766 GPU nodes.

Compute nodes are typically interconnected by fast network fabrics like Infiniband [7–9]. The same type of fabric is usually used to remotely store and retrieve data from the PFS. The latter is composed of multiple storage and metadata nodes and optimized for large and sequential I/O data access patterns. Storage-wise, many modern HPC centers equip their compute nodes with local disk space private to each node.

A. ML Data Flow

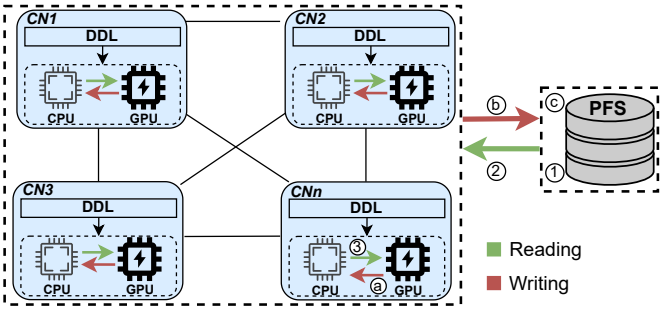


Fig. 1: DDL training data flow in HPC.

In the context of DDL training, the data flows mainly in two distinct directions: when reading the dataset and when writing model checkpoints, as shown in Figure 1. In order to train accurate models, large amounts of training data (*i.e.*, labeled data samples stored across multiple files) are read and processed by the model, leading to read-intensive workloads. Briefly, the dataset is stored initially on the PFS (Fig. 1-①), with each training epoch accessing all dataset samples (*i.e.*, reading all files from the PFS) into each node's host memory (Fig. 1-②) and then moving it to the computation device's memory (*e.g.*, GPU) for processing (Fig. 1-③).

To improve model accuracy, the entire dataset is processed exactly once per epoch in a randomized order, known as dataset shuffling. This results in a random file access pattern, leading to multiple accesses to the PFS metadata nodes, generating I/O contention and degrading overall performance for all user jobs accessing the shared file system [3].

At the end of each epoch, to prevent loss of training progress due to the failure of one or multiple nodes, models' states are checkpointed to persistent storage, usually the PFS. Specifically, the model state is copied from the computation device's memory into the host's memory, *i.e., GPU to CPU memory* (Fig. 1-ⓐ), then it is sent to the PFS (Fig. 1-ⓑ) and stored in it (Fig. 1-ⓒ). Notably, for complex models with a large number of parameters, such as LLMs, model checkpointing results in periodically transferring large amounts of data to the PFS. Previous studies show that checkpointing complex models can saturate up to 40% of the PFS bandwidth, leading again to severe contention for other workloads [10].

III. Systematization of Caches for DDL

Fig. 2 presents a generic overview of the basic workflow of current distributed caching solutions for DDL training. In a DDL workflow, a full copy of the dataset samples resides at the PFS. To minimize PFS access, a distributed cache, composed of the volatile memory and/or local storage resources of nodes participating in the training workload, is used to cache the entire set of dataset samples, or a subset of these if the available aggregated space is insufficient.

When a node asks for a given sample (Fig. 2-①), the local cache instance of that specific node is checked (Fig. 2-②). If

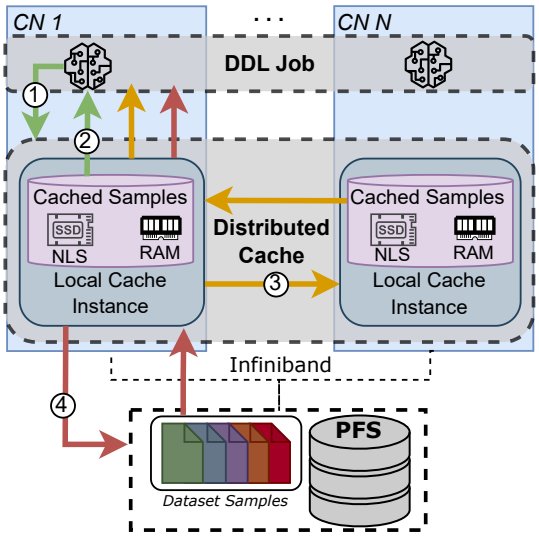


Fig. 2: Data flow of cache solutions for DDL.

the sample is not cached locally, then it is requested from a neighbor node participating in the same DLL workload (Fig. 2-③). If the sample is not cached at any node, it is requested from the PFS (Fig. 2-④).

The cache can be populated when the DDL workload begins or when a miss occurs and samples are fetched from the PFS. The choice of which compute nodes cache a given sample is based on a distribution strategy (*e.g.*, consistent hashing, static partitioning) [4, 11]. The choice of what samples to evict (Fig. 2-④) when the cache is full and a new item needs to be promoted is based on an eviction policy.

Some HPC centers, such as ABCI [9], enable users to deploy general-purpose distributed file systems (*e.g.*, BeeGFS) on-demand over the local storage of multiple compute nodes [12]. While these solutions spare I/O requests to the PFS they have several drawbacks. First, the dataset must be loaded from the PFS to the on-demand file system and must fully fit on the latter. Second, these are generic systems agnostic to DDL's I/O patterns (*e.g.*, random accesses to large sets of small files), suffering from performance contention due to metadata operations at scale, especially for I/O bound models [13].

To overcome the aforementioned drawbacks, distinct specialized cache solutions have emerged over the past few years. Table I organizes these according to key characteristics.

Intrusiveness. Diesel [14] offers a specialized file system, mounted over the local disks of compute nodes and built specifically for the small-file random I/O patterns of AI workloads. As it exports a generic file system interface, it is compatible with commonly used AI frameworks such as PyTorch [20] or TensorFlow [21].

HDF5 Cache Vol [15] improves on the HDF5 I/O library data format [22] to support caching. While this solution does

System	Intrusiveness	Cache Resources	Eviction Policy	Samples Placement	Samples Retrieval
DIESEL+ [14]	File System API	NLS	Eviction	Static	Prefetch
HDF5 Cache Vol [15]	Code Changes	In-Memory	Static	Best Effort	File
SHADE [16]	Code Changes	In-Memory	Data Substitution, Eviction	Best Effort	Prefetch
DeepFetch [17]	Code Changes	NLS	Eviction	Static	Prefetch
Quiver [18]	Code Changes	NLS	Data Substitution	Static	File
NoPFS [19]	Code Changes	Multi-Tier	Eviction	Static	Prefetch
DistMonarch [11]	LD_PRELOAD	NLS	Static	Best Effort	File
FanStore [5]	LD_PRELOAD	NLS	Static	Static	File
HVAC [4]	LD_PRELOAD	NLS	Eviction	Static	File Data

TABLE I: Analysis of state-of-the-art distributed cache systems regarding: *intrusiveness, cache resources, eviction policies, samples placement and data granularity.*

not require changing AI frameworks, it requires using the specific file format supported by the library. Other works, such as SHADE [16], DeepFetch [17], Quiver [18], and NoPFS [19], need direct changes over the frameworks' code, *e.g.,* in their data loaders, to know how samples are read or even to intercept and redirect these requests to other cached samples.

Alternatively, FanStore [5], DistMonarch [11], and HVAC [4] transparently intercept and manipulate I/O requests done by AI frameworks by using `LD_PRELOAD`. This approach requires no direct changes to the framework's code.

Cache resources. Choosing which local compute node resources are used to cache samples typically changes according to the details of the HPC infrastructure. Some solutions build their caches exclusively with volatile RAM memory from compute nodes, as these may not include local disks [15, 16].

Others take advantage of the node's local storage (NLS) to cache data, which has the benefit of freeing volatile memory for other processing needs of the ML workflow (*e.g.*, copying data from CPU to GPU, checkpointing) [4, 5, 11, 14, 17, 18]. NoPFS [19] considers a range of storage tiers, with different transfer speeds and storage capacities, combining both volatile memory and local storage at each node.

Eviction Policies. Typically, when a requested sample is not present in the cache, the sample is fetched from the PFS and then placed into the cache to avoid further cache misses in the next training epochs. However, when the cache is full, the eviction strategy to be adopted varies.

Some works use a static cache, not evicting or promoting any samples once the cache is full [5, 11, 15]. The rationale is justified by the random access pattern of DDL workloads across all dataset samples under each training epoch. Evicting samples to promote others that, in the next training epoch, may be requested later than the evicted ones will lead to multiple misses and to extra accesses to the PFS. Contrarily, other works use standard eviction policies, such as Random Sample Eviction, without introducing any specific policies geared towards DDL [4, 14, 16, 17, 19].

There are also solutions that, instead of fetching missed samples from the PFS, serve the DDL with other cached samples [16, 18]. This substitution promotes cache hits and avoids accessing the PFS, but it changes the unbiased randomness required by the training model for its accuracy guarantees. SHADE [16] combines sample eviction with substitution to reduce the accuracy penalty of the latter. It caches the samples that are more important for the training process in each epoch and uses them when there are cache misses.

Samples placement. In a distributed caching solution, each node is expected to store a portion of the full dataset samples. The distribution of samples across nodes can be done statically, for instance, by calculating a hash digest of the sample's file name or content [4, 5, 14, 17, 18]. NoPFS follows a static pattern, based on the access pattern of files, which is known in advance by instrumenting the AI framework's code [19].

Other solutions opt instead for a best-effort approach, where each node will cache missed samples that are not in its local cache nor in the cache of its neighboring nodes [11, 15, 16]. When a given node caches a new sample, it notifies its neighbors about this. We call this a best-effort approach since there is a time window where multiple nodes may end up caching the same sample. Given the random-access I/O nature of DDL workloads, this is, however, a rare event.

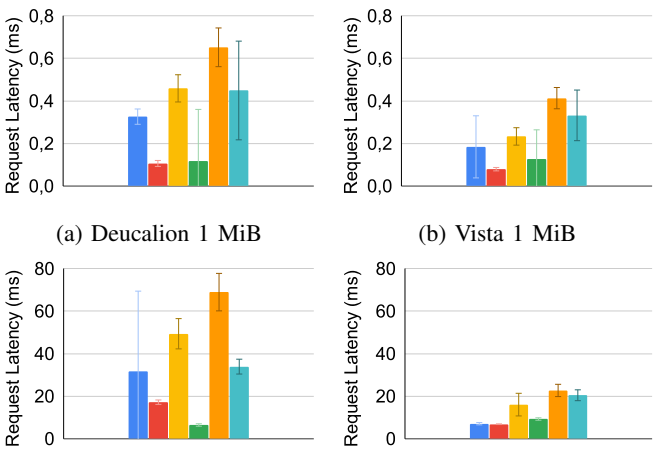
Samples retrieval. When a given sample is requested from neighboring nodes, solutions like HVAC explicitly request only that single sample [4]. Other approaches leverage larger file formats that store multiple samples (*e.g.*, TFRecords). This provides an opportunity to request the full file's content, optimizing network bandwidth, as the samples contained in it are expected to be accessed in the near future [5, 11, 15, 18]. To further optimize the use of the high-performance network bandwidth in HPC centers, some solutions prefetch more than one file per request [14, 16, 17, 19].

IV. Experimental Study

Our systematization of existing solutions shows that these take distinct approaches regarding the resources used for caching data, the policies for cache eviction, how samples are placed across nodes, and how these are exchanged across neighboring nodes. Next, we complement this information with a set of experiments, done using real supercomputers, to better understand the trade-offs of some of these decisions while pointing out open research directions.

A. Sample retrieval latency

Current solutions require reading samples from multiple sources, such as the PFS, local node disks, and in-memory or



(a) Deucalion 1 MiB (b) Vista 1 MiB

(c) Deucalion 100 MiBs (d) Vista 100 MiBs

Fig. 3: Read Results; PFS-R; NLS-R; NRM-R (TCP); NRM-R (Verbs); NRS-R (TCP); NRS-R (Verbs)

on-disk caches of neighboring nodes. However, these works do not quantify the impact on I/O latency of using such sources, especially under different hardware configurations.

Workload. We set up a synthetic workload that mimics the I/O read patterns of training the ILSVRC2017 subset of the *ImageNet* dataset [23]. Our stress workload issues 10,000 read requests, each targeting a random file from a set of 1000 files. We vary the size of files being read between 1 MiB and 100 MiBs. The latter is similar to the typical size of a TFRecord (*e.g.*, an average *Imagenet* TFRecord is sized 119 MiBs), while the former allows assessing solutions that do not perform full-file prefetching by requesting a smaller data size.

Experimental setup. Experiments were conducted on the Deucalion Supercomputer [7], on the `normal-a100-40` partition, and the Vista Supercomputer [8], on the `gg` partition. Deucalion is composed of 2165 compute nodes, with a local SATA SSD disk, and shares a PFS backend of size 10 PiB. Vista comprises 256 CPU and 608 accelerator-based compute nodes, each with a local NVMe SSD, and sharing a 1 TiB PFS per user. Both systems use Lustre as their shared file system.

Scenarios. Based on the different data sources, we defined four scenarios: **PFS-R**: reading from the PFS; **NLS-R**: reading from a node's local disk; **NRM-R**: fetching files from a remote node's in-memory cache; **NRS-R**: fetching files from a remote node's local disk. Scenarios *NRM-R* and *NRS-R* were assessed using two different transfer protocol APIs for data transmission between nodes, TCP and Verbs for Infiniband, the latter using the Mercury RPC Framework [24]. These are represented in Fig. 3 as *NRS-R (TCP)* and *NRS-R (Verbs)*, respectively.

In general, reading data from the node's local disk presents the smallest average latency, ranging from 0.08ms to 0.11ms for 1 MiB and 6.8ms to 17.2ms for 100 MiBs, across the two supercomputers. There is an exception, namely with the Vista supercomputer, and for 100 MiBs files, where reading data from the PFS is similar to doing so from the node's local disk (i.e., an average latency of ≈7.1ms and ≈6.9ms respectively).

With TCP over InfiniBand, rather than using the InfiniBand-specific API, I/O network latency between nodes is significantly higher. In the *NRS-R* scenario, the increase is up to 2.2x for Deucalion and 3.2x for Vista, making read accesses to the PFS faster, which uses InfiniBand through more optimized APIs. On Deucalion, the change in the network transfer API allows the *NRM-R* scenario to outperform fetching data from the PFS. Verbs is an API specific to the Infiniband network hardware, taking full advantage of its low latency and high throughput.

Reading data from remote nodes' memory is faster than reading from the PFS, up to 80% for Deucalion and 32% for Vista, and introduces less variability. Reading from a remote node's disk also reduces variability, but has slightly higher average latency. Even if PFSs are optimized for large read payloads, their performance highly depends on other workloads running simultaneously and using shared storage resources. Therefore, for larger payloads and under stable operation, the PFS may be a better option than remote nodes.

> **Takeaway 1.** *A node reading data from its local disk achieves lower latency and variability than when using the PFS (up to 3.1x and 36.6x, respectively). Requesting samples from peer disks is slower than local disks and PFS (up to 4x higher latency), but more stable (up to 10x lower variance) than PFS, particularly under heavy contention. Thus, performance-aware cache designs should stem from the trade-off between variability and latency across all available storage tiers, rather than only focusing on minimizing PFS accesses (as discussed in section II).*

> **Takeaway 2.** *On HPC systems, peer caching with optimized software and network stacks (e.g., Infiniband Verbs) outperforms traditional TCP-based approaches, which can have up to 7.7x higher latency. In some cases, this overhead compromises the cache, leading to worse overall performance than the PFS. Accordingly, general network approaches, widely used on other data centers such as cloud computing, are detrimental, performance-wise, when applied in HPC environments.*

B. Load Balancing

Distributed caches for DDL must implement a strategy to distribute data across multiple nodes. The experiment described next reproduces the behavior of a best effort placement approach used by related work and discussed in Section III [11]. We study how the strategy impacts the number of accesses made by a DDL training workload to each node's local disk, neighboring nodes, and the PFS.

Workload. We train the ResNet18 image classification model with using Tensorflow v2.8, running on 4 nodes with Data Parallelism, a batch size of 64, and during 4 training epochs [21, 25]. The dataset used is the ILSVRC2017 subset of

	Files in Cache	PFS Access	NLS Access	NLS Access by Peer Request	Request to Peers
Node 1	261	5828	138172	442346	433194
Node 2	262	5901	139227	431572	431271
Node 3	251	5969	134085	409560	437844
Node 4	273	5913	144688	442802	423969
Total	1047	23611	556172	1726280	1726278

TABLE II: Accesses to storage sources under DDL training.

the ImageNet, containing 1024 TFRecords with 1200 samples each and a total size of 220 GiBs [23].

Experimental setup. The experiment is conducted on Deucalion on the `normal-a100-40` partition, using the 4 NVIDIA Ampere A100 40 GB GPUs per node and the SATA SSD disk limited to a size of 100 GiBs. Tensorflow runs distributed across 4 of these nodes.

Table II shows that TFRecord files are nearly equally distributed across the disks of the 4 compute nodes. Given the best effort placement strategy used in the experiment, 2% of files end up replicated on the local disks of multiple nodes.

When the ResNet model is being trained, ≈6k accesses are made by each node to the PFS. Since data is only cached upon a miss, most accesses made in the first training epoch lead to a miss and consequently a request to the PFS. Note that the number of I/O operations is significantly higher than that of TFRecord files because Tensorflow issues multiple operations per file for reading subsets of samples.

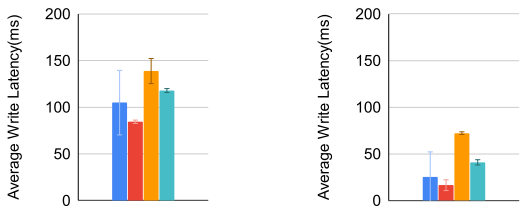
We also observe a balanced number of operations, considering the accesses made by all nodes to their local disks and the number of accesses each node receives from its neighbors.

The number of files each node caches directly impacts the number of times it accesses its own local disk and requests samples from other nodes' disk. In our specific setup, each node caches ≈256 files, occupying less than 40 GiBs, meaning that 60 GiBs of space are wasted. This observation is similar to other solutions using hashing schemes for data placement.

> **Takeaway 3.** *With higher numbers of compute nodes in DDL training workloads, total storage capacity grows, allowing the replication of dataset samples. Even with the best-effort placement strategy, data requests to peer nodes tend to be evenly balanced between all nodes, due to dataset shuffling. This promotes load balancing and minimizes hotspots, supporting uniform replication. Moreover, replicating data locally maximizes node local disk usage, enhancing throughput (Takeaway 1) and reducing network contention during training, as each node can issue up to 3.3x more requests to peers than accesses to its local disk.*

C. Model Checkpointing

DDL workflows must checkpoint the model's state periodically to guarantee quick recovery upon failures of one or multiple nodes. Next, we study the impact of storing checkpoint files over the PFS and across the local disk of compute nodes.



(a) Deucalion (b) Vista

Fig. 4: Write Results; PFS-W; NLS-W; NRS-W (TCP); NRS-W (Verbs);

Workload. We set up an experimental synthetic workload that mimics the I/O write patterns of writing checkpoint files. Namely, our stress workload issues 250 writes, each with a size of 176 MiBs. These parameters reproduce the values found when checkpointing the Bert model trained with 250 epochs.

The *experimental setup* is similar to the one used in IV-A, while the testing *scenarios* are also identical but focused on writing data to multiple storage sources. In detail, we have three scenarios: **PFS-W**: writing checkpoints to the PFS; **NLS-W**: writing to a node's local disk; **NRS-W**: writing checkpoint files to a remote node's disk. The latter two scenarios were assessed using TCP and Verbs for Infiniband network protocol APIs. These are represented in Fig. 4 as *NRS-W (TCP)* and *NRS-W (Verbs)* respectively.

Results show a similar performance pattern to the one observed for read requests. Namely, writing to the local disk exhibits the smaller latency (≈84.4ms for Deucalion and ≈16,4ms for Vista), followed by using the PFS (≈104,8ms for Deucalion and ≈25,4ms for Vista). Using Verbs also reduces latency when compared with the TCP protocol, up to 15% for Deucalion and 43% for Vista. When data is written to the local storage of remote nodes, using the Verbs protocol, it exhibits a slightly higher latency (≈117.8ms for Deucalion and ≈40,9ms for Vista) but less variability than using the PFS.

> **Takeaway 4.** *While node local storage presents the fastest write time, using it to buffer checkpoints may not be beneficial if the fault model considers that the full node may crash. An interesting alternative is using the local disk of remote nodes for replicating checkpoints, which showed higher latency than using the PFS (up to 2.8x) but with lower I/O variance (up to 18.7x).*

V. Conclusion and Open Challenges

Current DLL cache solutions alleviate the I/O contention experienced on the PFSs of modern supercomputers, while improving the use of local disk resources at compute nodes. This paper makes the first systematization of such works, pinpointing their shared and divergent characteristics, and opens up the path for new research challenges.

Co-designing caches for DLL with HPC infrastructures. The latency and bandwidth of I/O operations change when using the PFS, local node disks, or even remote nodes. Further,

these metrics may vary according to the supercomputer being used. Therefore, one should design new cache solutions that consider and adapt to such differences.

Leveraging leftover disk space. For large-scale (*e.g.*, tens to hundreds of nodes) training workloads, one can expect the combined space of compute nodes' local disks to significantly surpass the size of datasets. This opens the opportunity to replicate samples across the local disks of nodes and benefit from faster access speed.

Buffering checkpoints. Caches for DDL do not consider buffering model checkpoints in nodes' local disks. Taking advantage of these local resources can alleviate performance contention at the PFS for large models with many parameters.

Checkpoint file placement Hybrid replication strategies that leverage local and remote storage resources for persisting checkpoints may be worth exploring, along with their impact over training performance and fault-tolerance.

VI. Acknowledgments

We thank the anonymous reviewers for their insightful comments and feedback. This work was co-funded by the European Regional Development Fund (ERDF) through the NORTE 2030 Regional Programme under Portugal 2030, within the scope of the project BCD.S+M, reference 14436 (NORTE2030-FEDER-00584600), by national funds through FCT - Fundação para a Ciência e a Tecnologia, I.P., under the support UID/50014/2023 (https://doi.org/10.54499/UID/50014/2023), and by FCCN within the scope of the project Deucalion, with reference 2024.00014.TEST.DEUCALION. This work was also supported by funding from NSF (grants CNS-1956229, CSR-2402328, CAREER-2338457, and CSR-2323100), as well as generous donations from NetApp and Seagate.

References

[1] N. Lewis, J. L. Bez, and S. Byna, "I/O in Machine Learning Applications on HPC Systems: A 360-degree Survey," *ACM Computing Surveys*, vol. 57, no. 10, pp. 1–41, Oct. 2025.

[2] A. K. Paul, A. M. Karimi, and F. Wang, "Characterizing Machine Learning I/O Workloads on Leadership Scale HPC Systems," in *2021 29th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. Houston, TX, USA: IEEE, Nov. 2021, pp. 1–8.

[3] S. A. Wright and S. A. Jarvis, "Quantifying the effects of contention on parallel file systems," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, 2015, pp. 932–940.

[4] A. Khan, A. K. Paul, C. Zimmer, S. Oral, S. Dash, S. Atchley, and F. Wang, "Hvac: Removing I/O Bottleneck for Large-Scale Deep Learning Applications," in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*. Heidelberg, Germany: IEEE, Sep. 2022, pp. 324–335.

[5] Z. Zhang, L. Huang, U. Manor, L. Fang, G. Merlo, C. Michoski, J. Cazes, and N. Gaffney, "FanStore: Enabling Efficient and Scalable I/O for Distributed Deep Learning," 2018.

[6] H. Ather, J. L. Bez, C. Wang, H. Childs, A. D. Malony, and S. Byna, "Parallel I/O Characterization and Optimization on Large-Scale HPC Systems: A 360-Degree Survey," 2025.

[7] "Deucalion user guide," https://docs.macc.fccn.pt/, 2025, [Online; accessed 11-07-2025].

[8] "Vista user guide," https://docs.tacc.utexas.edu/hpc/vista/, 2025, [Online; accessed 11-07-2025].

[9] "ABCI user guide," https://docs.abci.ai/en/, 2025, [Online; accessed 11-07-2025].

[10] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient large-scale language model training on GPU clusters using megatron-LM," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, Nov. 2021, pp. 1–15. [Online]. Available: https://dl.acm.org/doi/10.1145/3458817.3476209

[11] M. Moreira, "Otimizações de armazenamento distribuído para aprendizagem profunda," Master's thesis, Universidade do Minho, 2023.

[12] "BeeGFS," https://www.beegfs.io/c/, 2025, [Online; accessed 11-07-2025].

[13] F. Chowdhury, Y. Zhu, T. Heer, S. Paredes, A. Moody, R. Goldstone, K. Mohror, and W. Yu, "I/O Characterization and Performance Evaluation of BeeGFS for Deep Learning," in *Proceedings of the 48th International Conference on Parallel Processing*. Kyoto Japan: ACM, Aug. 2019, pp. 1–10.

[14] L. Wang, Q. Luo, and S. Yan, "DIESEL+: Accelerating Distributed Deep Learning Tasks on Image Datasets," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 5, pp. 1173–1184, May 2022.

[15] H. Zheng, V. Vishwanath, Q. Koziol, H. Tang, J. Ravi, J. Mainzer, and S. Byna, "HDF5 Cache VOL: Efficient and Scalable Parallel I/O through Caching Data on Node-local Storage," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. Taormina, Italy: IEEE, May 2022.

[16] R. I. S. Khan, A. H. Yazdani, Y. Fu, A. K. Paul, B. Ji, X. Jian, Y. Cheng, and A. R. Butt, "{SHADE}: Enable fundamental cacheability for distributed deep learning training," in *21st USENIX Conference on File and Storage Technologies (FAST 23)*, 2023, pp. 135–152.

[17] L. Kong, F. Mei, C. Zhu, W. Cheng, and L. Zeng, "DeepFetch: A Node-Aware Greedy Fetch System for Distributed Cache of Deep Learning Applications," in *2024 International Conference on Networking, Architecture and Storage (NAS)*. Zhuhai, China: IEEE, Nov. 2024, pp. 1–8.

[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)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 283–296.

[19] N. Dryden, R. Böhringer, T. Ben-Nun, and T. Hoefler, "Clairvoyant prefetching for distributed machine learning I/O," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. St. Louis Missouri: ACM, Nov. 2021, pp. 1–15.

[20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[22] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, "An overview of the HDF5 technology suite and its applications," in *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*. Uppsala Sweden: ACM, Mar. 2011, pp. 36–47.

[23] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[24] J. Soumagne, D. Kimpe, J. Zounmevo, M. Chaarawi, Q. Koziol, A. Afsahi, and R. Ross, "Mercury: Enabling remote procedure call for high-performance computing," in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, 2013, pp. 1–8.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385