A Quantitative Study of Deep Learning Training on Heterogeneous Supercomputers

Jingoo Han*, Luna Xu†, M. Mustafa Rafique§, Ali R. Butt*, Seung-Hwan Lim‡,
*Virginia Tech, †IBM Research, §Rochester Institute of Technology, ‡Oak Ridge National Laboratory
*{jingoo, butta}@cs.vt.edu, †xuluna@ibm.com, §mrafique@cs.rit.edu, ‡lims1@ornl.gov

***Abstract*—Deep learning (DL) has become a key technique for solving complex problems in scientific research and discovery. DL training for science is substantially challenging because it has to deal with massive quantities of multi-dimensional data. High-performance computing (HPC) supercomputers are increasingly being employed for meeting the exponentially growing demand for DL. Multiple GPUs and high-speed interconnect network are needed for supporting DL on HPC systems. However, the excessive use of GPUs without considering effective benefits leads to inefficient resource utilization of these expensive setups. In this paper, we conduct a quantitative analysis to gauge the efficacy of DL workloads on the latest HPC system and identify viability of next-generation DL-optimized heterogeneous supercomputers for enabling researchers to develop more efficient resource management and distributed DL middleware. We evaluate well-known DL models with large-scale datasets using the popular TensorFlow framework, and provide a thorough evaluation including scalability, accuracy, variability, storage resource, GPU-GPU/GPU-CPU data transfer, and GPU utilization. Our analysis reveals that the latest heterogeneous supercomputing cluster shows varying performance trend as compared to the existing literature for single- and multi-node training. To the best of our knowledge, this is the first work to conduct such a quantitative and comprehensive study of DL training on a supercomputing system with multiple GPUs.**

***Index Terms*—Deep learning, TensorFlow, GPU Cluster, High Performance Computing, Heterogeneous Supercomputers**

I. Introduction

Deep learning (DL) [1] enabled artificial intelligence (AI) technology is widely used in industry due to its ability to recognize images, process natural languages, play games better than humans, and drive cars. We also have started witnessing the impact of DL in scientific discoveries [2], [3] such as classifying galaxies types [4], analyzing medical images [5], and predicting protein structure properties [6]. A notable trend in designing supercomputers is the adoption of extreme parallelism and heterogeneity [7], [8]. Such a trend makes high-performance computing (HPC) systems favorable to DL workloads. For example, Summit [9], the number one in the Top500 list [10], is equipped with $27,648$ NVIDIA V100 GPUs [11] (six GPUs on each node), and each is linked to IBM PowerPC9 with dual high speed NVLink connections [12]. This configuration enables Summit to be DL ready and makes it "one of the most AI-capable machines ever constructed" [13]. Similarly, Sierra [14] at Lawrence Livermore National Laboratory and AI Bridging Cloud Infrastructure [15] at National Institute of Advanced Industrial Science and Technology are also built for supporting DL workloads. In fact, five of the top ten supercomputers in 2018 have GPU-enabled heterogeneous environment as compared to two out of the top ten supercomputers in 2017. It shows an increasing trend of migration from traditional homogeneous supercomputers to GPU-based heterogeneous supercomputers. We expect this trend to continue with the high computational demands of emerging DL workloads.

Modern supercomputers are DL friendly due to system configuration to reduce I/O movement costs, which further enhances computational capabilities of hardware accelerators. In turn, they are able to meet the requirements of DL workloads entailing transferring enormous amount of data between compute nodes in the cluster to train models at the desired level of accuracy. For example, in the training of deep neural network (DNN) models over a climate data, a single GPU training of a DL model can consume 189 MB/s, which means the 6 GPUs on a Summit node require an aggregated 1.14 GB/s I/O bandwidth. The training of such a DNN model on the full Summit system therefore will require 5.23 TB/s [16]. To deliver the desired I/O bandwidth, HPC systems employ Non-Volatile Memory Express (NVMe) SSDs to reduce data I/O latency [17], [18] between the compute nodes and the storage servers with high throughput parallel file systems. Besides, compute clusters in modern HPC systems provision large memory (e.g., 1.6 TB per node in Summit) and high throughput interconnect, e.g., NVLink [12] that provides 25 GB/s bandwidth between a GPU and a CPU, along with a fast communication interconnect between compute nodes, such as 100 Gbps InfiniBand (IB) [19].

Despite significant efforts to efficiently run AI workloads in HPC environments [20], the effectiveness of a given HPC system for general DL workloads is highly desirable. A number of performance studies [21]–[23] have explored cloud environments to facilitate optimizations of industrial platforms and enable applying best practices to improve efficiency. However, since each supercomputing facility is uniquely built, the outcomes of these studies cannot be applied directly to HPC setups. Deploying an HPC system requires significant human resources and capital cost. Therefore, it is critical to clearly understand the behavior and performance of DL applications in current HPC systems to provide system design principles for future generations of supercomputers, and to guide researchers in better utilizing the available system resources for reducing the training span.

Although the top two supercomputers are equipped with IBM POWER processors [24] and multiple GPUs, most of the existing DL research is conducted on Intel Xeon-based HPC systems. Intel Xeon-based HPC systems use PCI bus between CPU and GPU, while the latest POWER-based HPC systems

978-1-7281-4734-5/19/$31.00 ©2019 IEEE

use NVLink between CPU and GPU, providing 5 times faster connectivity than PCIe [25]. Therefore, it is a critical factor impacting the performance compared to the previous studies as faster NVLink between CPU and GPU significantly reduces communication bottleneck. Although, the existing studies [21], [26]–[31] show that multiple GPUs and large-batch size can improve training throughput, they do not provide insights into the factors impacting the performance. To this end, this paper aims at providing valuable insights into critical performance factors, and show that the general understanding of running DL workloads on conventional data centers cannot be applied directly to supercomputing facilities.

We provide a quantitative study of DL training in modern POWER-based HPC environments, i.e., Summitdev [32], and evaluate representative DL training models with large datasets. We use the popular TensorFlow [27] framework and train DL networks using publicly available datasets, such as ImageNet dataset [33]. Furthermore, we measured the training throughput on a single node and multiple machines, while varying different parameters, such as the number of GPUs, batch size, file systems, etc., to study the impact of these parameters on DL training performance. This paper provides a thorough evaluation using comprehensive metrics such as training scalability, accuracy, storage performance, GPU-CPU data transfer, and GPU utilization. To the best of our knowledge, this is the first work to quantitatively study DL training on IBM POWER-based large-scale HPC systems. Specifically, this paper makes the following contributions:

- We evaluate TensorFlow DL applications on a POWER8-based heterogeneous supercomputer with high-end GPUs, storage, and network devices to identify the performance bottleneck in DL training;
- We compare the effect of Lustre [34] and NVMe-based XFS file systems [35] on I/O throughput and variability during DL training;
- We analyze CPU-to-GPU and GPU-to-CPU communication patterns by scaling our evaluation from 1 GPU to 128 GPUs (comprising $458,752$ CUDA [36] cores); and
- We analyze the impact of different DL framework parameters such as, batch size, and the number of workers and parameter servers, on DL training throughput, and provide key observations for improving DL training performance on HPC systems.

II. Background

In this section, we first give an overview of the architecture of a modern HPC supercomputer. Next, we briefly introduce a representative DL approach, convolutional neural network (CNN), and the basics of parallel training.

A. HPC Supercomputer Architecture

DL has influenced the architecture of modern HPC systems. The world's fastest HPC systems use multiple GPUs for DL computation workloads. Unlike the traditional homogeneous systems, GPU-based heterogeneous HPC systems are more suitable for DL workloads because GPUs provide strong parallel computing capabilities with higher bandwidth between parallel threads. To alleviate I/O overhead of multi-GPU, GPUs and CPUs are connected through high throughput, e.g., 75 GB/s NVLink interconnect. As these supercomputers have thousands of compute nodes, high performance networks, e.g., 100 Gbps IB, are deployed to improve data movement between the nodes. The latest HPC architectures have started using NVM [37] devices as burst buffers to reduce I/O time [38]. Parallel file systems, e.g., Lustre [34], and General Parallel File System (GPFS) [39], are used to improve I/O throughput. Lustre has been deployed on a large number of supercomputers as shared storage resource for efficiently handling I/O-intensive workloads [40], [41]. Recently, Summit has adopted IBM's Spectrum Scale GPFS that provides more than 2 TB/s of bandwidth [13]. Figure 1 shows high-level architecture of Summitdev [32] supercomputer that we use in this paper.

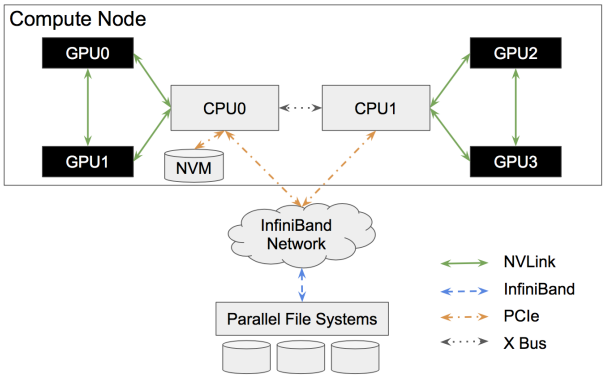


Fig. 1: Architecture of the studied HPC system.

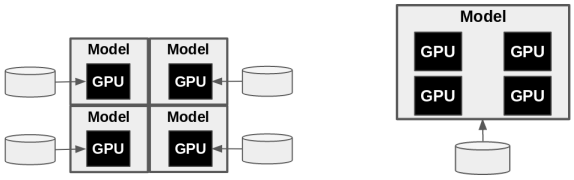
B. Convolutional Neural Network (CNN)

A CNN is a type of DL model that contains multiple convolutional layers. The layers extract features from spatial information of the input data, without manual feature extraction [1], [42]. A convolution layer comprises multiple feature maps to extract different features. A single feature map shares the same weights and the same bias to detect the same feature from the input images. However, other feature maps in the same layer use their own sets of weight and bias to extract different types of features. Each layer performs convolution operation on the feature maps, and then forwards these results to the next layer. Convolution requires the sum of the element-wise multiplication for each pixel. Therefore, training a CNN involves convolution operation with a large number of parameters at each layer, which requires very large number of image data and high computational capacity. Overall, DL training involves a series of complex processes, each adjusting parameters and processing data.

Table I shows popular CNN models. LeNet, originally developed in 1998, has significantly less number of parameters than other models [43]. AlexNet and ResNet-50 contain more complex layers and parameters than LeNet, and have 61 million and 25 million parameters, respectively [44]. Although ResNet-50 contains much deeper layers than AlexNet, its number of parameters are much less than that of AlexNet as

TABLE I: CNN model information.

Network	Convolution Layers	Fully Connected Layers	Parameter Size
LeNet	2	2	60K
AlexNet	5	3	61M
ResNet-50	49	1	25M
ResNet-101	100	1	45M



(a) Data parallelism **(b)** Model parallelism

Fig. 2: Data and model parallelism for DNN training on multiple GPUs.

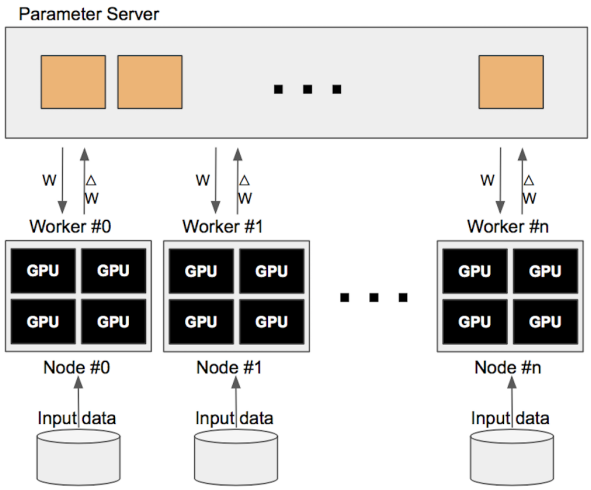


Fig. 3: Parameter server architecture for distributed deep learning.

ResNet deploys residual blocks that allow very deep networks with a smaller number of parameters than other CNN models.

1) Parallel Training: Training a large scale deep neural network (DNN) on a single server, even with high-end resources, would take many hours [45], [46]. Therefore, it is common practice to train such models in parallel. Figure 2 shows the two usual ways of parallelizing a DNN training, i.e., data parallel and model parallel. Data parallel approach partitions input data into batches, e.g., multiple images per batch, and replicates the network model on each GPU in a single node or multiple nodes. It essentially performs training of the same model with different batches of data. The weights are updated across all GPUs after each iteration. Increasing batch size reduces weight synchronization steps because the weight synchronization happens per batch, while very big batch sizes have a negative effect on training convergence [26]. Conversely, model parallel approach partitions a DNN model and assigns different portion of the DNN model's computation to multiple GPUs. For example, filters of a layer is split across GPUs, where each GPU performs convolutions with different parts of the filters [47]. Each GPU requires the output from other GPUs to synchronize results for a layer [26]. If the size of a model is large, then it would be challenging to upload the entire DNN model into GPU memory. In such a case, model-parallel approach is preferred. Previous studies have shown that the model parallel training causes more frequent synchronization overhead than the data parallel training [45], and data parallel approach scales better for convolutional layers than model parallel approach [26]. Therefore, the model parallel approach is currently not widely used. In our evaluation, we tested AlexNet and ResNet-50, where the number of convolution layers is larger than the number of fully connected layers. These DNN models fit into GPU memory on our HPC systems. Hence, we selected data parallel training for our evaluation.

2) Parallel Training in HPC: As shown in Figure 1, each node in our evaluation environment is equipped with multiple GPUs connected to two CPU sockets via NVLink, where all GPU-to-GPU and CPU-to-GPU connections go through NVLink. We evaluate our target system for data parallel approach at intra-node and inter-node levels. For intra-node level, each node fetches parameters into main memory, which are then copied into each GPU memory through NVLink. It enables each GPU to have its own copy of the parameters that it uses to process different part of a batch. Whenever one iteration is finished, a master GPU, referred to as GPU0, gathers the gradients from other GPUs through NVLink. GPU0 aggregates all gradients, and then sends the aggregate to the CPU on the worker. Finally, the CPU pushes the single aggregated gradient into the parameters.

For inter-node level, we adopt the widely applied parameter server architecture [48] for distributed parameter aggregation as shown in Figure 3. Under such a scheme, two types of nodes are deployed, i.e., parameter server and workers. Parameter server maintains globally shared parameters that can be shared by multiple workers. Input data and computational workload are distributed to multiple worker nodes. Each worker node retrieves the latest parameters from the parameter server, reads input data from the storage systems, processes training, and sends updates back to the parameter server directly.

Summary: In distributed training, the total number of iterations for training a single network model are distributed between workers. Before performing the computation, the model parameter W is pulled from the parameter server to all workers. The training data subset, called mini-batch, is loaded from the storage devices to the main memory. After decoding and resizing the data, the data is then divided across the GPUs. Multiple GPUs are deployed in each node for achieve further intra-node level data parallelism, i.e., if a mini-batch has 128 images and there are four GPUs in a single compute node, then 32 images can be trained simultaneously during one iteration by each GPU at each node. GPU0 gathers gradients from other GPUs, and sends an aggregated gradient to the CPU. Then, each worker pushes the updated delta W to the parameter server to maintain the latest model parameters.

III. Methodology

Performance of DL training has been well studied on commodity servers [28], [31] and cloud environments [27]. However, DL training has not been thoroughly studied on the latest HPC systems (such as IBM-built heterogeneous supercomputers), in spite of the effort and revenue invested in building DL-ready HPC systems. Simply applying the lessons from commodity servers and clouds to a HPC systems is not recommended given the significant differences in the hardware and software architecture, configurations, and scale. Thus, it is crucial to understand how DL training performs atop state-of-the-art DL-ready HPC systems, and to evaluate cost–benefit trade-offs, diagnose bottlenecks, and improve hardware and software designs to efficiently support DL training.

Existing studies reveal that DL training is compute-intensive [49], less sensitive to data I/O [50], and suffer from network communication overhead when scaling-out [29], [50]. In fact, modern HPC systems are being designed with these assumptions about the DL training behavior. In our study, we quantitatively validate these conclusions in top HPC systems with high-end GPUs, storage systems, and network devices. Our study aims to answer the following research questions:

- Do traditional performance understanding and assumptions of DL training still hold in DL capable HPC systems? For example, do large-batch training and multi-GPU training provide scalability?
- How much performance benefit can be achieved from the upgrade of GPUs, modern storage and network devices for DL applications?
- Do DL applications scale with the increasing number of parameter nodes, worker nodes, and GPU devices? What is the communication overhead for scaling-up and scaling-out?
- How do workload characteristics have impact on the performance indications?

We design our experiments to answer the above research questions. First, we conduct our experiments on a supercomputer, i.e., Summitdev, that provides an early access development platform for the top ranked supercomputer Summit with latest configurations. Our system architecture is shown in Figure 1. Specifically, our test bed has 54 compute nodes where each compute node has four NVIDIA Tesla P100 GPUs and two IBM POWER8 CPUs as shown in Table II. Each P100 GPU has 3584 CUDA cores and 16 GB memory (maximum memory clock frequency is 715 MHz, and the peak memory bandwidth is 732 GB/s).

We select TensorFlow deep learning framework for our experiments because of its wide adoption in academia and industry [51], [52]. We choose TensorFlow 1.8 with CUDA 9.0 [53] and cuDNN 7.0.3 [54]. We use the default GPU memory option, where TensorFlow allocates GPU memory dynamically when necessary. We focus on CNN training as it has been well-studied with a lot of variable models, and is proven to be easy to scale with data-parallel schemes. We select ImageNet dataset [33] in this paper for training the model as it is widely used for DL evaluation [55], [56], and therefore, offers a representative benchmark. There are about 1.2 million images of $1,000$ categories for training and $50,000$ images for validation. For training, we use TensorFlow-Slim [57] suite, a library that provides various image categorization models.

TABLE II: Specification of the evaluation environment.

	Summitdev Supercomputer
Compute nodes	54
GPUs	4 NVIDIA Tesla P100
CPUs	2 IBM POWER8
Cores per CPU	10
CPU core clock	2.0 GHz
Main memory	256 GB
NVLink	NVLink 1.0 (40 GB/s)
Interconnection	Mellanox IB EDR (100 Gbps)
File systems	Lustre, NVMe-based XFS
NVMe	800 GB NVMe per compute node

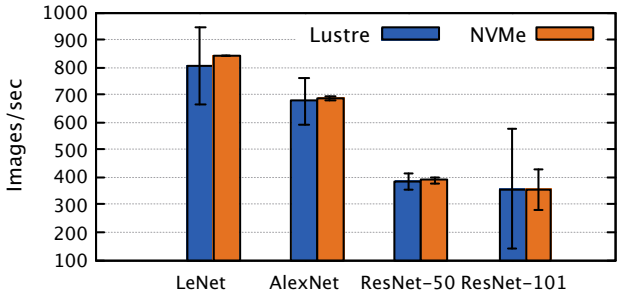
We measure the training throughput (images per second) as our performance metric, since it is a de facto metric used in the DL community. We conduct experiments with data both from locally-attached NVMe SSDs and Lustre file system to identify the performance benefit of fast local storage. Moreover, we record the performance variance of both scenarios as we expect Lustre to suffer from network congestion when I/O load is high, while local SSDs will not be affected by other users. We conduct experiments to analyze scaling characteristics of the studied DL workloads. We scale our workloads from 1 GPU to 128 GPUs. For distributed training, we conduct experiments with increasing number of workers with the fixed number of parameter server, and increasing number of parameter servers with fixed number of workers to study the impact for different scaling configurations. We quantitatively study the communication overhead of scaling-up with data transfer measurements between CPU and GPU, as well as between multiple GPUs. Finally, to answer the last question, i.e., effects of workload characteristics, we analyze the performance impact of model parameters, such as batch size and data store formats, in addition to selecting models with different characteristics, in the evaluation of the DL models studied in this paper.

IV. Evaluation Results

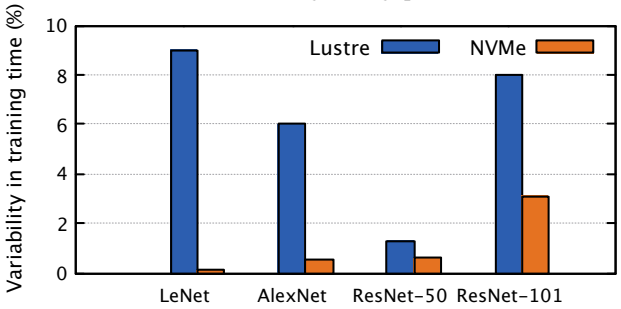
We setup our experiments according to the methodology as described in Section III. We built TensorFlow to support the IBM POWER architecture in our platform, and configure TensorFlow to use IP over IB for distributed training in parameter server mode. We also made necessary modifications to TensorFlow-Slim package to support distributed training with parameter server mode. We run each experiment five times and report averages with 95% confidence intervals, except for cases where negligible variation is observed between runs.

A. Overall Performance

We first give an overview of performance of the studied workloads (Table I) in our experimental setup. As shown in Figure 4a with the results of Lustre, LeNet [42] performance is faster than other neural networks. This is because LeNet has less number of layers and parameters as compared to the other



(a) Training throughput.



(b) Variability in training time.

Fig. 4: Training with Lustre and NVMe.

DL models. Similarly, ResNet-101 has the worst performance among the models as it has the biggest number of layers as shown in Table I. The evaluation of different models on HPC shows a similar trend to the existing studies [58]. In our following sections, we will breakdown the obtained performance to investigate in detail the impact of each component in our HPC setting on the training of the studied DL models.

The biggest highlight of DL-ready supercomputers is the upgrade of GPUs. To test how much DL can benefit from a top class GPU, we also tested the GPU utilization using the studied workloads. Figure 5 shows the results. Our evaluation shows that the GPU utilization increases with the increase in the number of layers of DL models. LeNet shows the lowest GPU utilization with only 2.3% whereas the utilization goes up to 69.9% for ResNet-101. This result confirms that the computation overhead of ResNet-101 is higher than other CNNs, which slows down the overall training throughput. However, we can see that the number of layers almost doubles from ResNet-50 to ResNet-101 model, but the performance only decreases by 7%. This is due to the high utilization of GPU using ResNet-101 model, which significantly benefits the training throughput. Our results show that GPU upgrade is useful for compute-intensive models, such as ResNet, instead of simple DL models, such as LeNet. Moreover, even with ResNet-101, the GPU utilization is only less than 70%. It is essential to design algorithms for models, resource managers, workload schedulers that are able to fully utilize available high performance GPUs. We further discuss the communication overheads with the upgrade of numbers of GPUs in a single node in Section IV-C.

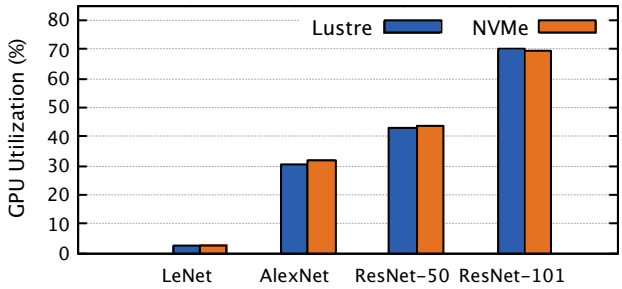


Fig. 5: GPU utilization using different storage configurations.

B. Impact of Storage

As described in Section III, each node in our experimental platform is supplied with a 800 GB NVMe SSD, and a parallel file system (Lustre) is shared by all cluster nodes for serving I/O requests. We first measure the I/O performance of both storage configurations. We run the default read tests with a 2.5 GB file using the UNIX `dd` [59] command. We observe an I/O read throughput of 7.2 GB/s for NVMe SSD, and 4.2 GB/s for Lustre file system. We also observe that the performance variance of NVMe SSD is more stable than Lustre, i.e., $7.0 \sim 7.3$ GB/s for NVMe SSD, $2.0 \sim 5.2$ GB/s for Lustre. This gives a performance variation of 0.3 GB/s for NVMe SSD as compared to a performance variation of 3.2 GB/s for Lustre. Note that performance of Lustre file system is dependent on the I/O requests from all users in the system. With this raw performance difference in mind, we now examine how high throughput I/O devices, such as NVMe SSD, can impact the performance of DL using different training models.

We measure the training throughput with input data in Lustre and XFS (local file system on NVMe SSD) storage systems respectively to identify the storage bottlenecks of these storage systems. In our evaluation platform, Lustre is distributed across multiple storage nodes and might create a network bottleneck with increasing number of I/O requests from distributed cluster nodes, whereas NVMe is mounted using XFS file system where each node has its own NVM that serves as a burst buffer. Therefore, storage bottleneck can cause a performance degradation in DL training over Lustre storage system. Figure 4a shows the results. We can observe from the figure that training on the same dataset has a similar performance trend for both back-end storage systems. The performances of LeNet and AlexNet DL models using NVMe are faster than the corresponding performances on Lustre, i.e., 4.63% and 1.75%, respectively, while the performances of ResNet-50 and ResNet-101 models are almost identical for both configurations. The result shows that NVMe can enhance the training throughput of DL models where computational overhead is not significant.

The I/O performance of Lustre file system is heavily dependant on the overall I/O in the system, whereas the performance of NVMe is dependent on the I/O load at the corresponding node. Figure 4b shows the difference between the shortest and the longest training times using the two

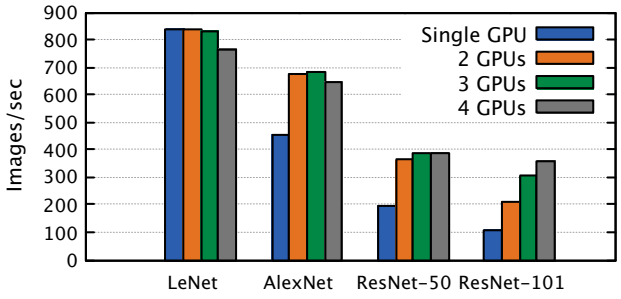


Fig. 6: Training throughput using multiple GPUs.

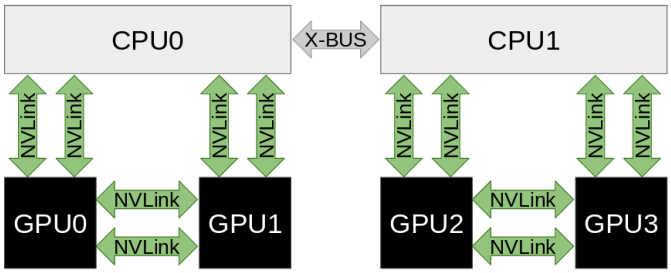


Fig. 7: NVLink topology used in our evaluation setup.

configurations. While the training times on NVMe is very stable and shows a negligible performance difference between 0.1% and 3.1%, the training time on Lustre shows a variability between $1.3\% \sim 9.0\%$. This result shows that an unstable storage system can affect DL performance because of the variability in the training time. We expect the variance in the case of Lustre file system to further increase when it is being used by a large number of users with a diverse set of workloads, thus creating more noise for the file system.

C. Impact of Scale

In the next set of experiments, we explore how DL training scales in HPC systems with increasing number of hardware resources. We analyze scale-up in a single node with a maximum of 4 GPUs per node, and scale-out to the maximum 32 compute nodes in our HPC cluster. Overall, we scale DL training to 128 distributed GPUs in our experimental setup.

1) Scale-up with Multiple GPUs on a Single node: We evaluate the impact of varying the number of GPUs on a single compute node on the training performance of DL models, and show the result in Figure 6. The performance of AlexNet, ResNet-50, and ResNet-101 is improved with the increasing number of GPUs as computational load is distributed between available GPUs. However, a performance degradation is observed for LeNet when two or more GPUs are used. LeNet does not need to distribute computation loads to multiple GPUs because its computation workload is less intensive than other models. As shown in Figure 5, LeNet has a very low GPU utilization and adding more GPUs incurs more coordination overhead than the achieved parallelism. The training performance of AlexNet and ResNet-50 substantially increases when two GPUs are used as compared to using a single GPU. However, further increase in the number of GPUs for training does not provide any performance improvement because of the communication overhead between the participating GPUs. ResNet-101 shows better scalability due to higher computation overhead than other models. It achieves non-linear speedup, i.e., $3.26\times$, with four GPUs. This result shows a different performance trend as compared to a recent research [49], where stronger scalability is achieved with four V100 GPUs on a single node.

To further investigate scaling-up bottlenecks, we take a deeper look into the communication overhead of our setup. Figure 7 shows the NVLink connection topology between the CPUs and GPUs in our evaluation platform. We have two CPUs and four GPUs connected together through NVLink with different CPU affinities. Two GPUs, i.e., GPU0 and GPU1, are connected to the CPU0, while the other two GPUs, i.e., GPU2 and GPU3, are connected to the CPU1 through NVLink. NVLink provides peak unidirectional theoretical bandwidth of 40 GB/s [60]. However, in our measurements, we found that NVLink is only able to achieve 79% (31.77 GB/s) and 87% (35.12 GB/s) of the theoretical peak bandwidth for GPU-to-CPU and GPU-to-GPU communication, respectively.

To identify the communication load, we measure the size of the transferred data through NVLink during one epoch both for CPU-to-GPU and GPU-to-GPU communications. As Table III shows, large amount of data is being transferred through NVLink during DL training. The total data transfer size is about 24 TB and 1.8 TB for AlexNet and LeNet, respectively. The data transfer size for ResNet-50 (15 TB) is smaller than AlexNet even though the former has much deeper layers in its training model. Therefore, the GPU-to-GPU/GPU-to-CPU workload is related to the parameter size of the model. We also observed that GPU-to-CPU communication is higher than GPU-to-GPU communication, which means that multi-GPU environment needs to focus on GPU-to-CPU workloads for achieving faster DL training. The ratio of GPU-CPU to GPU-GPU communications is related to NVLink topology (shown in Figure 7). There is a direct connection between GPU0 and GPU1, while there is no direct connection between GPU0 and GPU2, and between GPU0 and GPU3.

In TensorFlow, GPU0 gathers gradients from other GPUs in each iteration and then sends the aggregated gradients to the CPU. This scheme is effective for Intel-based systems as they connect CPU and GPU0 using PCIe bus while GPU1/2/3 are connected directly with GPU0 via NVLink. However, in POWER-based systems, this scheme is ineffective because GPU2 and GPU3 cannot send data directly to GPU0. Hence, GPU2 and GPU3 send their gradients through additional paths e.g., through NVLink between GPU2 and CPU1, through X-BUS between CPU1 and CPU0, and through NVLink between CPU0 and GPU0, which requires additional hops. According to our measurements, bandwidth between GPU0 and GPU1 is 35.12 GB/s, while bandwidth between GPU0 and GPU2 is 21.47 GB/s. Roughly 40% bandwidth degradation occurs due to the indirect connection between two GPUs, i.e., GPU0 and GPU2, and GPU1 and GPU3. When only two GPUs (e.g., GPU0 and GPU1) are used, there is no such additional transfer

TABLE III: Total data transfer (GB) through NVLink using 4 GPUs.

	GPU-to-CPU	GPU-to-GPU	Total
LeNet	1520	326	1846
AlexNet	20355	3832	24187
ResNet-50	13136	1948	15084

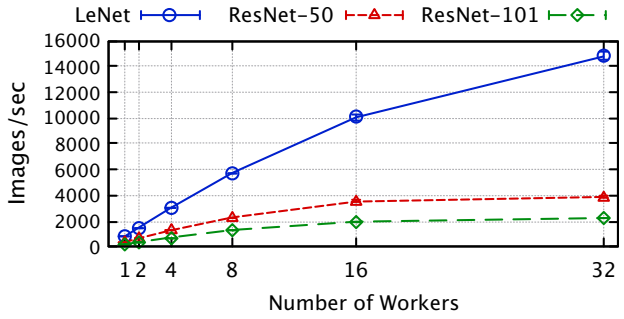
bottleneck. However, when four GPUs are used, GPU0 has to wait for a longer period of time to receive gradients from GPU2 and GPU3. Thus, the current multi-GPU environment of our target HPC system does not provide scalability for DL training with TensorFlow framework. This observation advocates developing new workflow schemes that incorporate GPU connection topology in making task assignments decisions.

2) Scale-out with Multiple nodes: One of the common approaches for distributed training is to use the standard distributed TensorFlow which consists of multiple parameter servers and workers. One or more parameter servers aggregate gradients and broadcast corresponding updates to the worker nodes. Each worker then executes computational parts, such as convolution operation, of the DL model. The native distributed TensorFlow framework uses protocol buffer [61] based Google's remote procedure call (gRPC) [62] for efficient high performance communication between the compute nodes.

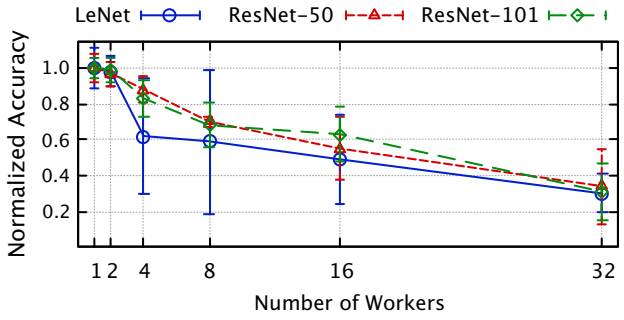
We employ the standard distributed TensorFlow framework with multiple parameter servers and workers and study the impact of cluster configurations on distributed DL throughput. We varied the number of parameter servers or workers in this set of experiments, while other options, such as back-end storage, are fixed. Each scenario was executed on parallel file system with a fixed batch size of 128 images per node. We do not include AlexNet in our scalability study due to the occurrence of a well-known NaN (Not a Number) [63] error with TensorFlow.

a) Scalability with Multiple Workers: Figure 8a shows the training throughput when the number of compute nodes increases 32 nodes using a separate parameter server. LeNet shows a scalable performance improvement up to 1211%, and ResNet-50 also shows performance improvement of 94% $\sim$ 1094%, as compared to the single-worker training. ResNet-101 shows a similar performance trend, i.e., 93% $\sim$ 1012% improvement, as compared to the single-worker training. The ResNet models show less performance improvement with 32 nodes as compared to LeNet. This is because LeNet has lower checkpointing overhead with less parameters as compared to ResNet models. Although the training is scalable, the checkpointing mechanism is not scalable with multi-node. In our experiment, checkpointing with Tensorflow takes about 5 seconds in LeNet, but it takes about 70 seconds in ResNet-101. This result shows that although high speed network can provide scale-out performance, it is critical to develop scalable checkpointing techniques for models that have a large number of parameters to deliver high scalability.

Accuracy: Existing studies [64], [65] show that training with large batches using multiple nodes results in slow convergence. Our experiments also confirm this trend as shown in figure 8b. However, distributed training is still promising and



(a) Training throughput.



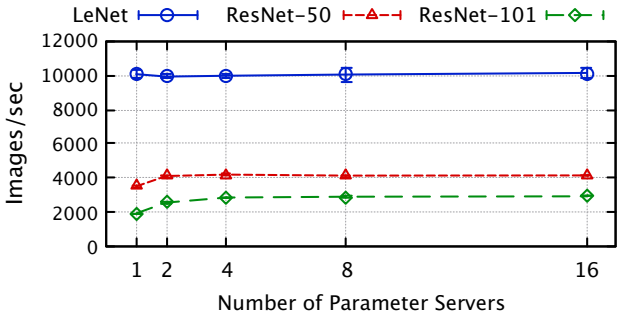
(b) Accuracy.

Fig. 8: Distributed training with multiple workers.

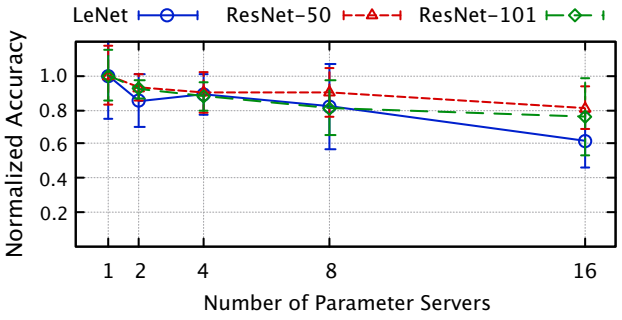
several efforts [16], [50], [66], [67] are exploring methods to reduce accuracy degradation. Learning rate algorithms, such as Layer-wise Adaptive Rate Scaling (LARS) [50], can be effectively used to mitigate the impact of accuracy loss. Recent efforts [16], [66], [67] have demonstrated performance improvement for large-batch training without sacrificing accuracy by using LARS-based methods.

b) Scalability with Multiple Parameter Servers: Our previous experiments use only one parameter server. In this experiment, we study parameter server scalability by increasing the number of parameter servers from 1 to 16 on a cluster with 16 worker nodes and show the results in Figure 9a.

Counterintuitively, increasing the number of parameter servers is less effective for all DL models, which shows different trend compared to the past research [30]. For LeNet, using more parameter servers is ineffective, whereas, for ResNet-50 and ResNet-101, there is a slight improvement initially, but no performance gain is observed going beyond more than 4 parameter servers. Combined with the previous observations of scaling with multiple workers, the result of this experiment shows that when scaling-out a DL model, it is more efficient to increase the number of workers instead of the number of parameter servers. Network bandwidth is known [46] to have an impact on parameter server's performance as network congestion increases when serving multiple workers. However, high network bandwidth, i.e. 100 Gbps, equipped in our setup reduces the performance impact of such network congestion and results in the negligible end-to-end performance improvement. On the contrary, older HPC systems or cloud computing systems deploy network with lower bandwidth (e.g., NERSC



(a) Training throughput.



(b) Accuracy.

Fig. 9: Distributed training with multiple parameter servers.

Cori 66.4 Gbps, Amazon EC2 10 Gbps) [68] can benefit more from scaling parameter servers.

Accuracy: The existing study [23] shows that multiple parameter servers results in slow convergence performance. Our result, shown in Figure 9b, shows a similar pattern. The accuracy reduces with the increase in the number of parameter servers. Asynchronous updates of multiple parameter servers results in unsynchronized partitions between parameter server nodes, which causes slow-down of convergence [48], [69]. This result underscores the conclusions of the existing studies [48], [69]–[71].

D. Impact of Workload Configurations

In this set of experiments, we explore the performance impact of application configurations. The goal of this evaluation is to provide insights and guidance on how to configure DL training on supercomputers to best utilize the capability of the resources and realize best performance without loosing training accuracy.

1) Data Format: A recent study shows that the data format can impact training time while achieving same quality [72]. Specifically, key-value (KV) stores such as LMDB [73] can provide 17× speedup [72] as compared to using separate image files (e.g., PNG format) as input dataset. KV stores have also been used as HPC storage back-ends because of achieving high performance and high scalability with simple design and easy to use interfaces. Thus, we investigate how using KV stores in our HPC platform can impact the performance of DL applications. To this end, we measure the training performance with two different storage options, namely, TFRecord-formatted [74] files on Lustre file system, and LMDB [73]

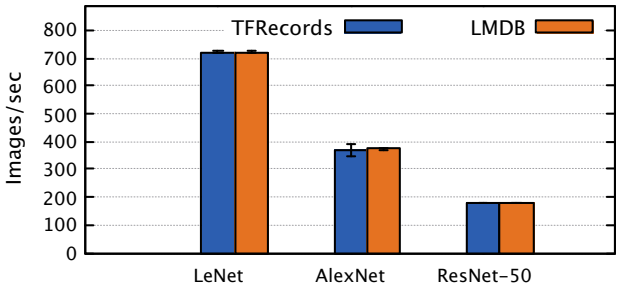


Fig. 10: Training throughput with TFRecords and LMDB.

database files on the Lustre file systems. TFRecord file format is a TensorFlow-specific binary format whereas LMDB is a light and simple-to-deploy B+ tree-based KV store.

The default implementation of TensorFlow does not provide mechanism to create LMDB data files from the input datasets. Therefore, we implemented a C++ based parser that creates a LMDB database file from the provided ImageNet datasets. In our evaluation, TFRecords-formatted files and LMDB database files are located on the Lustre file systems. Figure 10 shows that the performance difference between TFRecords and LMDB is negligible, i.e., under 1% as compared to each other. We note a different performance trend from an existing analysis [72] of using image files in the storage systems for training DNN models. File systems cannot provide superior indexing and caching mechanism to locate image files because it caches blocks from the same image file [72], thus providing poor performance when using separate image files. Instead of using separate image files, TFRecords-formatted files for ImageNet dataset consist of 1024 files for about 1.2 million images. Thus, TFRecords-formatted files can provide comparable throughput when compared to LMDB KV store.

2) Batch Size: Batch size per machine is an important parameter configuration in DL applications that can also impact the training performance [65], [66]. As discussed in Section II, a large dataset is partitioned into smaller batches to achieve data parallel training. Large batch sizes will increase computation workload with decreased gradient aggregation frequency and weight update. Here, we analyze how batch size affects training performance by scaling the batch size from 128 to 8192 images on a single node. Figure 11 shows the results. The results of ResNet-50 are not available after 512 images as going beyond 512 images per batch saturates the GPU memory. Intuitively, bigger batch size with higher compute and less communication overhead is preferred for throughput reasons. However, we observe that increasing batch size on a single node does not lead to performance enhancement on the target supercomputer. When the batch size is increased from 128 images to 256 images, it results in negligible performance enhancement, i.e., 1.0% $\sim$ 3.7% for the studied models. Moreover, performance degradation occurs when very large, i.e., 4K and 8K, batch sizes are used. Large batch training is not effective in our system setup. Previous studies [64], [75] have used large batch sizes to reduce the number of training

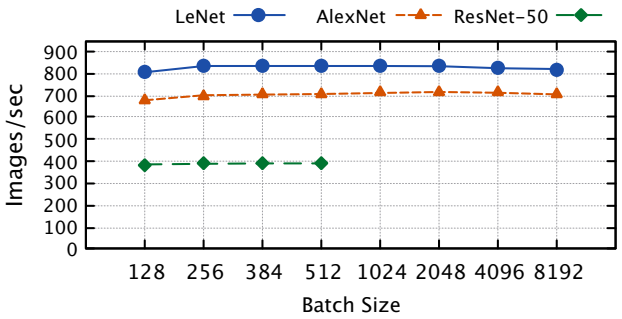


Fig. 11: Training throughput on a single node with increasing batch size.

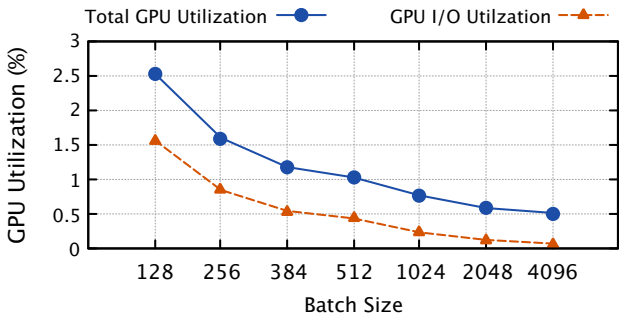
steps, where the performance gains are obtained by reducing the I/O overhead between CPU and GPU. This assumes slower connections between CPU and GPU, and reducing the training steps is effective in reducing the I/O overhead. While the existing Intel-based HPC systems use PCIe bus between CPU and GPU, the latest POWER-based HPC systems have NVLink (5× faster than PCIe) between CPU and GPU. As a result, the percentage of GPU I/O utilization (e.g., GPU-to-GPU copy, CPU-to-GPU copy) is already much low, i.e, the maximum observed is 18.06%, as shown in Figure 12. LeNet and ResNet-50 show lower GPU I/O utilization than AlexNet (3.50% ∼ 0.08%), as the number of parameters in these models are less than AlexNet. In our setup, large batch sizes are not helpful in reducing GPU I/O time.

We further measure how large-batch per node affects distributed training with 16 worker nodes and 1 parameter server as shown in Figure 13. Similarly, large-batch training in distributed setting is also not effective on training throughput, while accuracy drops significantly.

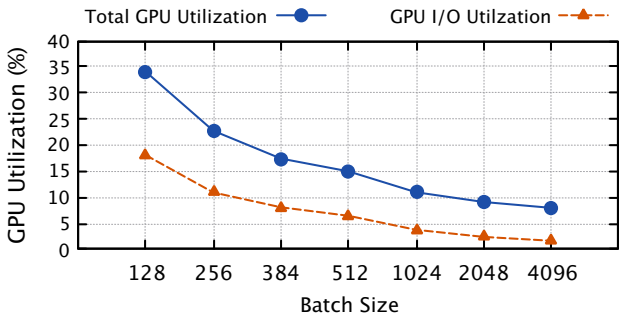
V. Discussion

We made the following key findings based on our evaluation, which show different performance trends from past works on analyzing the performance of DL workloads. These findings help provide insights to system designers and DL scientists.

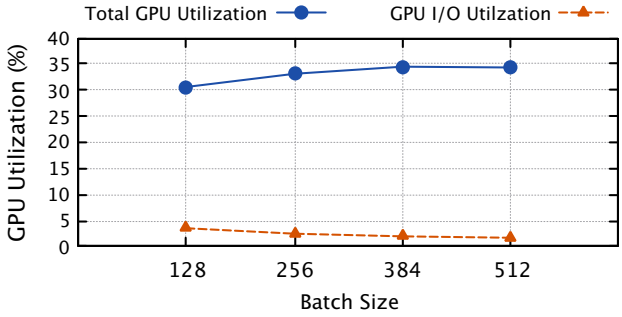
- Employing multiple GPUs on a single node for training DL workloads does not help in improving the model performance. While using two GPUs may help improve model performance by sharing the load, using additional GPUs diminishes the performance benefits because of the coordination overhead. This is in contrast to the existing practices [49], [58], where using a larger number of GPUs is recommended for scaling DL training performance.
- GPU utilization should be increased by means of designing efficient resource managers or schedulers [76] to obtain scalable training in a multi-GPU system. Although NVLink is used to connect GPUs on a single node, GPUs are not fully utilized.
- Distributed training with multiple workers enhances throughput as compared to single node training, but the performance does not scalable with the increasing number of parameter servers.



(a) LeNet



(b) AlexNet

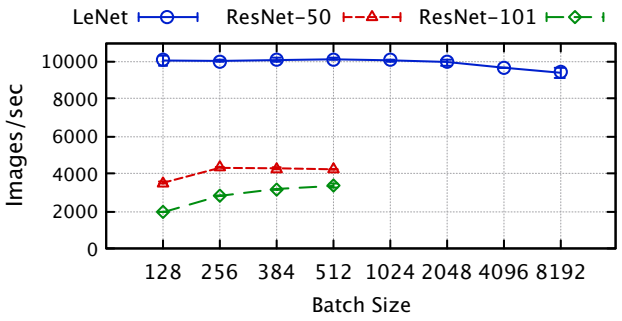


(c) ResNet-50

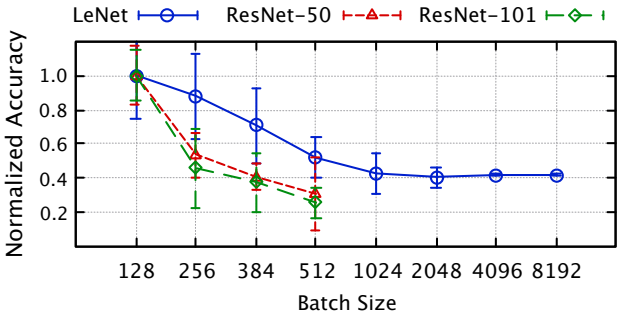
Fig. 12: GPU utilization on a single node with increasing batch size.

- Large batch size does not improve training throughput both on a single node and in distributed training. Normally, increasing batch size provides performance enhancements, however, our evaluation reveals only a slight performance improvement, i.e., 2.7% on average, when using a batch size of 256. Also, large-batch training has very limited performance gain for distributed training on multiple nodes. Thus, large batching size may not yield substantial benefits for DL training as commonly expected in other use cases.
- Upgrade of storage system gives limited performance improvement to DL training. Considering the data copying overhead (from parallel file system to locally attached NVMe SSD), it might not be worth using the high performance SSDs for faster data loading.

The lesson learned from the study, i.e., simply adding high-throughput multi-GPUs in HPC might be ineffective for improving DL training performance, would also apply to other accelerators, such as, TPUs, FPGAs, and ASICs, as



(a) Training throughput.



(b) Accuracy.

Fig. 13: Distributed training with increasing batch size.

the main underlying causes exposed by our work are the performance differences between CPUs and accelerators, and the I/O limitations between the available processing devices. These causes will exacerbate with the use of specialized accelerators. If these accelerators have comparable compute power as a powerful GPUs and use fast connectivity, such as NVLink, then they will provide a similar performance pattern as observed in this work.

VI. Related Works

The performance of DL training has been studied on different platforms. Mojumder et al. [58] analyzed DL workloads on DGX-1 system that uses the fastest V100 GPUs. Shi et al. [49] evaluated distributed DL performance with multiple machines that have multiple GPUs. Chien et al. [77] analyzed effects of the number of threads, burst buffer for checkpoint, and prefetcher on TensorFlow performance with Xeon-based multi-GPU supercomputers. Xu et al. [31] compared performance of V100 and P100 GPUs. However, these studies are not conducted with the latest hardware upgrades, such as NVLink for GPU-CPU communication, which are critical for improving the performance of DL applications. In contrast, our work performs intensive performance analysis including the impact of NVLink on the performance of DL applications.

There are existing studies on DL training analysis conducted on IBM POWER systems. Shams et al. [29] evaluated DL training performance with NVIDIA's NVLink on an IBM POWER-based machine. Gupta et al. [78] investigated tradeoffs between accuracy and DL training time on IBM POWER-based homogeneous supercomputers without GPUs. While these works have used IBM POWER processors, their evaluation environments are different from the emerging IBM POWER-based heterogeneous supercomputers. Shams et al. focused on a single machine only, unlike HPC systems. Similarly, Gupta et al. did not evaluate on multiple GPUs, which is common in top-ranked supercomputers.

DL performance evaluation has also been studied [21], [79] on cloud platform where Intel CPUs and NVIDIA GPUs are supported by Amazon Web Services (AWS). However, cloud environment differs a lot from HPC environment where dedicated high performance resources, such as CPU, GPU, memory and SSDs are provided without virtualization. Thus, our study is unique and different from such works, and unlike them, our results are applicable to HPC systems.

Although the top two supercomputers are IBM POWER-based heterogeneous HPC systems, there is no comprehensive performance research on HPC systems where IBM POWER CPUs coupled with high-end GPUs are used as compute nodes. To the best of our knowledge, ours is the first work that studies DL on representative systems (similar to top-ranked supercomputers) and provides detailed performance breakdown using different hardware and software settings.

VII. Conclusion

The use of DL for solving highly complex problems and deriving scientific discoveries is increasing fast. To meet this growing demand, supercomputers are evolving to support popular DL and machine learning workloads by deploying energy-efficient GPUs (such as NVIDIA Tesla GPUs), faster interconnect (such as NVLink and IB), and faster storage systems (such as NVMe SSDs). In this paper, we presented a quantitative performance study and analysis of deep learning training using popular TensorFlow framework on a heterogeneous supercomputing cluster built on IBM POWER-based machines. To the best of our knowledge, this is the first effort to evaluate DL performance on IBM POWER-based heterogeneous supercomputers with a thorough analysis of the impact of hardware settings, such as storage, interconnect, CPUs and GPUs, as well as software configurations, such as batch size and data format. Our evaluation reveals that system insights are required to improve DL performance. Existing practices, e.g., large-batch training, and multi-GPU training, may not be helpful in improving DL throughput performance. The main factors impacting the DL training throughput are GPU utilization, and the communication link performance between CPU and GPU.

Acknowledgment

This work is sponsored in part by the NSF under the grants: CNS-1405697, CNS-1615411, and CNS-1565314/1838271. This research used resources of the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at the Oak Ridge National Laboratory, which is supported by the Office of Science of the DOE under Contract DE-AC05-00OR22725.

REFERENCES

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[2] L. Xu, S. Lim, M. Li, A. R. Butt, and R. Kannan. Scaling up data-parallel analytics platforms: Linear algebraic operation cases. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 273–282, Dec 2017.

[3] L. Xu, S. Lim, A. R. Butt, S. R. Sukumar, and R. Kannan. Fatman vs. littleboy: Scaling up linear algebraic operations in scale-out data platforms. In *2016 1st Joint International Workshop on Parallel Data Storage and data Intensive Scalable Computing Systems (PDSW-DISCS)*, pages 25–30, Nov 2016.

[4] Sander Dieleman, Kyle W Willett, and Joni Dambre. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly notices of the royal astronomical society*, 450(2):1441–1459, 2015.

[5] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[6] Jian Zhou and Olga G Troyanskaya. Deep supervised and convolutional generative stochastic network for protein secondary structure prediction. *arXiv preprint arXiv:1403.1347*, 2014.

[7] Amogh Katti, Giuseppe Di Fatta, Thomas Naughton, and Christian Engelmann. Epidemic failure detection and consensus for extreme parallelism. *The International Journal of High Performance Computing Applications*, 32(5):729–743, 2018.

[8] Jeffrey S. Vetter, Ron Brightwell, Maya Gokhale, Pat McCormick, Rob Ross, John Shalf, Katie Antypas, David Donofrio, Travis Humble, Catherine Schuman, Brian Van Essen, Shinjae Yoo, Alex Aiken, David Bernholdt, Suren Byna, Kirk Cameron, Frank Cappello, Barbara Chapman, Andrew Chien, Mary Hall, Rebecca Hartman-Baker, Zhiling Lan, Michael Lang, John Leidel, Sherry Li, Robert Lucas, John Mellor-Crummey, Paul Peltz Jr., Thomas Peterka, Michelle Strout, and Jeremiah Wilke. Extreme heterogeneity 2018 - productive computational science in the era of extreme heterogeneity: Report for doe ascr workshop on extreme heterogeneity. 12 2018.

[9] ORNL Launches Summit Supercomputer. *https://www.ornl.gov/news/ornl-launches-summit-supercomputer*, 2018.

[10] Top500. *http://www.top500.org*, 2018.

[11] NVIDIA Tesla V100. *https://www.nvidia.com/en-us/data-center/tesla-v100/*, 2018.

[12] NVLink. *https://www.nvidia.com/en-us/data-center/nvlink/*, 2018.

[13] Jonathan Hines. Stepping up to summit. *Computing in Science & Engineering*, 20(2):78–82, 2018.

[14] Sierra. *https://hpc.llnl.gov/hardware/platforms/sierra*, 2018.

[15] AI Bridging Cloud Infrastructure (ABCI). *https://abci.ai/*, 2018.

[16] Thorsten Kurth, Sean Treichler, Joshua Romero, Mayur Mudigonda, Nathan Luehr, Everett Phillips, Ankur Mahesh, Michael Matheson, Jack Deslippe, Massimiliano Fatica, et al. Exascale deep learning for climate analytics. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, page 51. IEEE Press, 2018.

[17] Ning Liu, Jason Cope, Philip Carns, Christopher Carothers, Robert Ross, Gary Grider, Adam Crume, and Carlos Maltzahn. On the role of burst buffers in leadership-class storage systems. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pages 1–11. IEEE, 2012.

[18] Jeffrey S Vetter and Sparsh Mittal. Opportunities for nonvolatile memory systems in extreme-scale high-performance computing. *Computing in Science & Engineering*, 17(2):73–82, 2015.

[19] Infiniband trade association. *https://www.infinibandta.org/*, 2018.

[20] Thorsten Kurth, Jian Zhang, Nadathur Satish, Evan Racah, Ioannis Mitliagkas, Md Mostofa Ali Patwary, Tareq Malas, Narayanan Sundaram, Wahid Bhimji, Mikhail Smorkalov, et al. Deep learning at 15pf: supervised and semi-supervised classification for scientific data. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 7. ACM, 2017.

[21] Shang-Xuan Zou, Chun-Yen Chen, Jui-Lin Wu, Chun-Nan Chou, Chia-Chin Tsao, Kuan-Chieh Tung, Ting-Wei Lin, Cheng-Lung Sung, and Edward Y Chang. Distributed training large-scale deep architectures. In *International Conference on Advanced Data Mining and Applications*, pages 18–32. Springer, 2017.

[22] Christian Pinto, Yiannis Gkoufas, Andrea Reale, Seetharami Seelam, and Steven Eliuk. Hoard: A distributed data caching system to accelerate deep learning training on the cloud. *arXiv preprint arXiv:1812.00669*, 2018.

[23] Pijika Watcharapichat, Victoria Lopez Morales, Raul Castro Fernandez, and Peter Pietzuch. Ako: Decentralised deep learning with partial gradient exchange. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pages 84–97. ACM, 2016.

[24] Satish Kumar Sadasivam, Brian W Thompto, Ron Kalla, and William J Starke. Ibm power9 processor architecture. *IEEE Micro*, 37(2):40–51, 2017.

[25] Burak Bastem, Didem Unat, Weiqun Zhang, Ann Almgren, and John Shalf. Overlapping data transfers with computation on gpu with tiles. In *2017 46th International Conference on Parallel Processing (ICPP)*, pages 171–180. IEEE, 2017.

[26] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[27] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[28] Víctor Campos, Francesc Sastre, Maurici Yagües, Jordi Torres, and Xavier Giró-i Nieto. Scaling a convolutional neural network for classification of adjective noun pairs with tensorflow on gpu clusters. In *Cluster, Cloud and Grid Computing (CCGRID), 2017 17th IEEE/ACM International Symposium on*, pages 677–682. IEEE, 2017.

[29] Shayan Shams, Richard Platania, Kisung Lee, and Seung-Jong Park. Evaluation of deep learning frameworks over different hpc architectures. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 1389–1396. IEEE, 2017.

[30] Amrita Mathuriya, Thorsten Kurth, Vivek Rane, Mustafa Mustafa, Lei Shao, Debbie Bard, Victor W Lee, et al. Scaling grpc tensorflow on 512 nodes of cori supercomputer. *arXiv preprint arXiv:1712.09388*, 2017.

[31] Rengan Xu, Frank Han, and Quy Ta. Deep learning at scale on nvidia v100 accelerators. 2018.

[32] Summitdev. *https://www.olcf.ornl.gov/for-users/system-user-guides/summitdev-quickstart-guide/*, 2019.

[33] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[34] Lustre. *http://lustre.org*, 2018.

[35] Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, and Geoff Peck. Scalability in the xfs file system. In *USENIX Annual Technical Conference*, volume 15, 1996.

[36] David Kirk. Nvidia cuda software and gpu parallel computing architecture. In *Proceedings of the 6th International Symposium on Memory Management*, ISMM '07, pages 103–104, New York, NY, USA, 2007. ACM.

[37] Sparsh Mittal and Jeffrey S Vetter. A survey of software techniques for using non-volatile memories for storage and main memory systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(5):1537–1550, 2016.

[38] Sudharshan S Vazhkudai, Bronis R de Supinski, Arthur S Bland, Al Geist, James Sexton, Jim Kahle, Christopher J Zimmer, Scott Atchley, Sarp Oral, Don E Maxwell, et al. The design, deployment, and evaluation of the coral pre-exascale systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, page 52. IEEE Press, 2018.

[39] Frank B Schmuck and Roger L Haskin. Gpfs: A shared-disk file system for large computing clusters. In *FAST*, volume 2, 2002.

[40] Yingjin Qian, Xi Li, Shuichi Ihara, Lingfang Zeng, Jürgen Kaiser, Tim Süß, and André Brinkmann. A configurable rule based classful token bucket filter network request scheduler for the lustre file system. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 6. ACM, 2017.

[41] Weikuan Yu, Jeffrey Vetter, R Shane Canon, and Song Jiang. Exploiting lustre file joining for effective collective io. In *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, pages 267–274. IEEE, 2007.

[42] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[43] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[44] Yunhe Wang, Chang Xu, Jiayan Qiu, Chao Xu, and Dacheng Tao. Towards evolutional compression. *arXiv preprint arXiv:1707.08005*, 2017.

[45] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. Deep learning with cots hpc systems. In *International Conference on Machine Learning*, pages 1337–1345, 2013.

[46] Trishul M Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *OSDI*, volume 14, pages 571–582, 2014.

[47] Omry Yadan, Keith Adams, Yaniv Taigman, and Marc'Aurelio Ranzato. Multi-gpu training of convnets. *arXiv preprint arXiv:1312.5853*, 2013.

[48] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[49] Shaohuai Shi, Qiang Wang, Xiaowen Chu, and Bo Li. Modeling and evaluation of synchronous stochastic gradient descent in distributed deep learning on multiple gpus. *arXiv preprint arXiv:1805.03812*, 2018.

[50] Yang You, Igor Gitman, and Boris Ginsburg. Scaling sgd batch size to 32k for imagenet training. *arXiv preprint arXiv:1708.03888*, 2017.

[51] Bokyung Seo, Myungjae Shin, Yeong Jong Mo, and Joongheon Kim. Top-down parsing for neural network exchange format (nnef) in tensorflow-based deep learning computation. In *Information Networking (ICOIN), 2018 International Conference on*, pages 522–524. IEEE, 2018.

[52] Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. Tensorfi: A configurable fault injector for tensorflow applications. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 313–320. IEEE, 2018.

[53] Compute Unified Device Architecture (CUDA). *https://developer.nvidia.com/cuda-zone*, 2019.

[54] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[55] Paweł Michalski, Bogdan Ruszczak, and Michał Tomaszewski. Convolutional neural networks implementations for computer vision. In *International Scientific Conference BCI 2018 Opole*, pages 98–110. Springer, 2018.

[56] Wei Huang, Jing Zeng, Peng Zhang, Guang Chen, and Huijun Ding. Single-target localization in video sequences using offline deep-ranked metric learning and online learned models updating. *Multimedia Tools and Applications*, 77(21):28539–28565, 2018.

[57] Tensorflow-slim. 2018. https://github.com/tensorflow/models/tree/master/research/slim.

[58] Saiful A Mojumder, Marcia S Louis, Yifan Sun, Amir Kavyan Ziabari, José L Abellán, John Kim, David Kaeli, and Ajay Joshi. Profiling dnn workloads on a volta-based dgx-1 system. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pages 122–133. IEEE, 2018.

[59] Unix dd. *https://en.wikipedia.org/wiki/Dd_(Unix)*, 2019.

[60] Marcelo Amaral, Jordà Polo, David Carrera, Seetharami Seelam, and Malgorzata Steinder. Topology-aware gpu scheduling for learning workloads in cloud environments. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 17. ACM, 2017.

[61] Google Protocol Buffer. *https://developers.google.com/protocol-buffers/*, 2018.

[62] gRPC. *https://grpc.io/*, 2018.

[63] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, 23(1):5–48, 1991.

[64] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.

[65] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[66] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. Imagenet training in minutes. In *Proceedings of the 47th International Conference on Parallel Processing*, ICPP 2018, pages 1:1–1:10, New York, NY, USA, 2018. ACM.

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

[68] Yang You, Aydın Buluç, and James Demmel. Scaling deep learning on gpu and knights landing clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 9. ACM, 2017.

[69] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu. Heterogeneity-aware distributed parameter servers. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 463–478. ACM, 2017.

[70] Mu Li, Li Zhou, Zichao Yang, Aaron Li, Fei Xia, David G Andersen, and Alexander Smola. Parameter server for distributed machine learning. In *Big Learning NIPS Workshop*, volume 6, page 2, 2013.

[71] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 583–598, 2014.

[72] Seung-Hwan Lim, Steven R Young, and Robert M Patton. An analysis of image storage systems for scalable training of deep neural networks. *system*, 5(7):11, 2016.

[73] Lightning Memory-Mapped Database Manager (LMDB). *http://www.lmdb.tech/doc/*, 2018.

[74] TFRecords. *https://www.tensorflow.org/tutorials/load_data/tf_records*, 2018.

[75] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch sgd: training resnet-50 on imagenet in 15 minutes. *arXiv preprint arXiv:1711.04325*, 2017.

[76] Luna Xu. *A Workload-aware Resource Management and Scheduling System for Big Data Analysis*. PhD thesis, Virginia Tech, 2019.

[77] Steven WD Chien, Stefano Markidis, Chaitanya Prasad Sishtla, Luis Santos, Pawel Herman, Sai Narasimhamurthy, and Erwin Laure. Characterizing deep-learning i/o workloads in tensorflow. *arXiv preprint arXiv:1810.03035*, 2018.

[78] Suyog Gupta, Wei Zhang, and Fei Wang. Model accuracy and runtime tradeoff in distributed deep learning: A systematic study. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pages 171–180. IEEE, 2016.

[79] Liang Luo, Jacob Nelson, Luis Ceze, Amar Phanishayee, and Arvind Krishnamurthy. Parameter hub: a rack-scale parameter server for distributed deep neural network training. *arXiv preprint arXiv:1805.07891*, 2018.