DLoBD: A Comprehensive Study of Deep Learning over Big Data Stacks on HPC Clusters

Xiaoyi Lu, *Member, IEEE*, Haiyang Shi, Rajarshi Biswas,
M. Haseeb Javed, and Dhabaleswar K. Panda, *Fellow, IEEE*

Abstract—Deep Learning over Big Data (DLoBD) is an emerging paradigm to mine value from the massive amount of gathered data. Many Deep Learning frameworks, like Caffe, TensorFlow, etc., start running over Big Data stacks, such as Apache Hadoop and Spark. Even though a lot of activities are happening in the field, there is a lack of comprehensive studies on analyzing the impact of RDMA-capable networks and CPUs/GPUs on DLoBD stacks. To fill this gap, we propose a systematical characterization methodology and conduct extensive performance evaluations on four representative DLoBD stacks (i.e., CaffeOnSpark, TensorFlowOnSpark, MMLSpark/CNTKOnSpark, and BigDL) to expose the interesting trends regarding performance, scalability, accuracy, and resource utilization. Our observations show that RDMA-based design for DLoBD stacks can achieve up to 2.7x speedup compared to the IPoIB-based scheme. The RDMA scheme also scales better and utilizes resources more efficiently than IPoIB. For most cases, GPU-based schemes can outperform CPU-based designs, but we see that for LeNet on MNIST, CPU + MKL can achieve better performance than GPU and GPU + cuDNN on 16 nodes. Through our evaluation and an in-depth analysis on TensorFlowOnSpark, we find that there are large rooms to improve the designs of current-generation DLoBD stacks.

Index Terms—DLoBD, deep learning, big data, CaffeOnSpark, TensorFlowOnSpark, MMLSpark (CNTKOnSpark), BigDL, RDMA

1 Introduction

As the explosive growth of 'Big Data' continues, there is an increasing demand for getting Big Value out of Big Data to drive the revenue continuously growing. To mine more value from the massive amount of gathered data, in these days, Deep Learning over Big Data (DLoBD) is becoming one of the most efficient analyzing paradigms. With this emerging paradigm, more and more Deep Learning tools or libraries start being run over Big Data stacks, such as the most popular representatives—Apache Hadoop and Spark. By combining the advanced capabilities from Deep Learning libraries (e.g., Caffe [1], TensorFlow [2], and Microsoft Cognitive Toolkit (CNTK) [3]) and Big Data stacks (e.g., Spark and Hadoop), the DLoBD approach can enable powerful distributed Deep Learning on Big Data analytics clusters with at least following three major benefits. 1) From the data analytics workflow perspective, if we run Deep Learning jobs on Big Data stacks, we can easily integrate Deep Learning components with other Big Data processing components in the whole workflow. 2) From the data locality perspective, since the large amount of gathered data in companies typically is already stored or being processed in Big Data stacks (e.g., stored in HDFS), Deep Learning jobs on Big Data stacks can easily access the data without moving it back and forth. 3) From the infrastructure management perspective, we do not need to set up new dedicated Deep Learning clusters if we can run Deep Learning jobs directly on existing Big Data analytics clusters. This could significantly reduce the costs of device purchasing and infrastructure management.

With the benefits of integrating Deep Learning capabilities with Big Data stacks, we see a lot of activities in the community to build DLoBD stacks, such as CaffeOnSpark,[1] SparkNet [4], TensorFlowOnSpark,[2] DL4J [5], BigDL [6], and Microsoft Machine Learning for Apache Spark (MMLSpark or CNTKOnSpark) [7]. Many of these DLoBD stacks are also being deployed and used on Cloud Computing platforms, such as Microsoft Azure. For DLoBD stacks, one of the typical concerns is about their 'sub-optimal' performance. As shown in Fig. 1, with the convergence of HPC, Big Data, and Deep Learning, these emerging DLoBD stacks are being designed to leverage Remote Direct Memory Access (RDMA) capable high-performance interconnects and multi-/many-core based CPUs/GPUs. These powerful devices give a lot of opportunities to speed up the DLoBD stacks.

1.1 Motivation

Fig. 1 shows the main components that a typical Deep Learning job running over DLoDB stacks involves. As we can see, there are at least five major layers: Deep Learning model or application layer, Deep Learning library layer, Big Data analytics framework layer, resource scheduler layer,

- *The authors are with the Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43202.*
  E-mail: {lu.932, shi.876, biswas.91, javed.19, panda.2}@osu.edu.

Manuscript received 16 Jan. 2018; revised 1 May 2018; accepted 23 May 2018. Date of publication 11 June 2018; date of current version 29 Jan. 2019.
(Corresponding author: Xiaoyi Lu.)
Recommended for acceptance by R. Grant.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TMSCS.2018.2845886

[1] https://github.com/yahoo/CaffeOnSpark
[2] https://github.com/yahoo/TensorFlowOnSpark

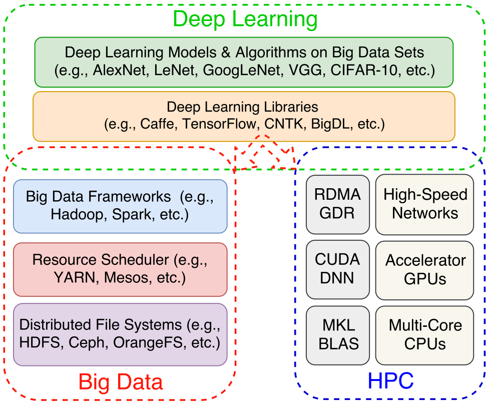

Deep Learning
Deep Learning Models & Algorithms on Big Data Sets (e.g., AlexNet, LeNet, GoogLeNet, VGG, CIFAR-10, etc.)
Deep Learning Libraries (e.g., Caffe, TensorFlow, CNTK, BigDL, etc.)
Big Data Frameworks (e.g., Hadoop, Spark, etc.)
Resource Scheduler (e.g., YARN, Mesos, etc.)
Distributed File Systems (e.g., HDFS, Ceph, OrangeFS, etc.)
Big Data
RDMA GDR
High-Speed Networks
CUDA DNN
Accelerator GPUs
MKL BLAS
Multi-Core CPUs
HPC

Fig. 1. Convergence of deep learning, big data, and HPC; Overview of the corresponding characterization scope of DLoBD stacks.

and distributed file system layer. There are a lot of efforts in the field to improve the performance of each of these layers. For example, the default Caffe, TensorFlow, and CNTK can leverage the high-performance GPU accelerators with the co-designed efficient cuDNN [8] library, while BigDL can efficiently run on Intel CPUs or Xeon Phi devices by utilizing the highly optimized Intel MKL [9] library or BLAS libraries. Yahoo! researchers have proposed RDMA-based communication in CaffeOnSpark and TensorFlowOnSpark. Our earlier work [10], [11], [12] have proposed RDMA-based designs for Spark and Hadoop. Even though these work have been proposed and well studied with their targeted workloads and environments, there is a lack of systematic studies on analyzing the impact of RDMA-capable networks and CPUs/GPUs on DLoBD stacks with different Deep Learning models and datasets. We lack understanding the impact of these advanced hardware and the associated efficient building blocks (e.g., RDMA, GPUDirect RDMA, cuDNN, and MKL) on various Deep Learning aspects, including performance, accuracy, scalability, and resource utilization. These lead to the following broad challenges:

- How are current generation DLoBD stacks being designed? Why do they need high-performance communication subsystems?
- Can RDMA-based designs in DLoBD stacks improve performance, scalability, and resource utilization on high-performance interconnects, GPUs, and multi-core CPUs?
- What are the performance characteristics of representative DLoBD stacks when they run typical Deep Learning workloads on RDMA-capable high-speed networks?
- What kind of trends and insights can we observe in our evaluations for performance and accuracy, which are the two most important factors for Deep Learning workloads?
- How much performance overhead is brought due to the heavy layers of DLoBD stacks? What kind of potential bottlenecks are there in the typical DLoBD stacks?

1.2 Contribution

To address all of these challenges, this paper first selects four representative DLoBD stacks (i.e., CaffeOnSpark, TensorFlowOnSpark, MMLSpark, and BigDL) based on their popularity and designs. We overview their architecture differences and similarities in Section 2, which help us to design our characterization methodology. Then, we further propose a systematical characterization methodology in Section 3 to cover a broad range of evaluation dimensions, such as comparing different networking protocols (i.e., IPoIB versus RDMA), comparing different ways of integration with Big Data stacks (i.e., in-band communication versus out-of-band communication), and comparing solutions using different computing devices (i.e., CPU versus GPU). Our characterization will focus on four different perspectives, including performance, accuracy, scalability, and resource utilization.

Section 4 presents our detailed evaluation, which shows that RDMA-based DLoBD stacks can achieve up to 2.7x speedup compared to the IPoIB based scheme. RDMA-based designs can also scale better and utilize resources more efficiently than the IPoIB scheme. For most cases, we see GPU-based Deep Learning can outperform CPU-based designs, but not always. We see that for LeNet on MNIST, CPU + MKL can achieve better performance than GPU and GPU + cuDNN on 16 nodes.

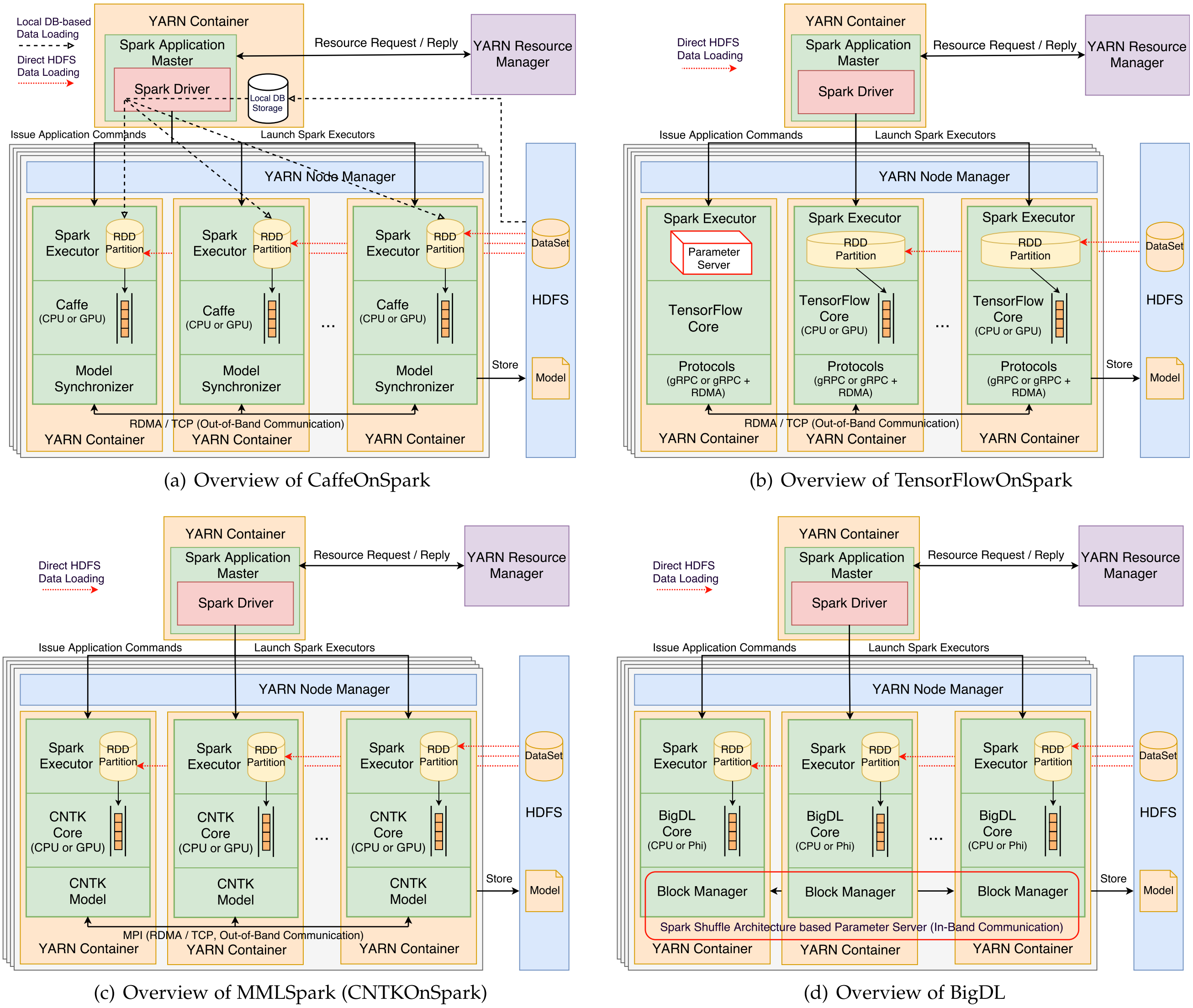
In addition to benchmarking these DLoBD stacks in a black-box manner, we further provide an in-depth analysis of TensorFlowOnSpark in Section 5. The in-depth analysis with TensorFlowOnSpark chooses a vertical approach to breakdown the Deep Learning workload performance across DLoBD layers. From the analysis, we find that up to 15.5 percent time could be spent in the Apache Hadoop YARN scheduler layer, while up to 18.1 percent execution time could be consumed by the Spark job execution layer. Compared to native TensorFlow, TensorFlowOnSpark can get the benefit of automatically scaling out Deep Learning applications across nodes and accessing data easily from HDFS. But in the meantime, our studies show that the community may need to spend more effort to reduce the overhead of heavy DLoBD stacks, even though such kind of overhead may be negligible in long-running Deep Learning jobs.

From the communication perspective, the analysis in Section 5 shows that the RDMA-based communication channel in TensorFlow or TensorFlowOnSpark has the potential to benefit large or complex Deep Learning models. Our evaluation shows that training a ResNet50 model on TensorFlow can get around 21 percent performance benefit with RDMA compared to using the IPoIB protocol.

Through our evaluation and analysis, we see that there are still large rooms to improve the designs of current generation DLoBD stacks. More insights are shared in this paper to guide designing next-generation DLoBD stacks. Section 6 discusses related work. We conclude the paper with observed insights and future work in Section 7.

2 Overview of DLoBD Stacks

There are broadly two mechanisms for parallelizing a Deep Learning algorithm: *Model Parallelism* and *Data Parallelism*. Model Parallelism is when the different processing elements use the same data, but the model is distributed among them. In the Data Parallelism, the same model is used for every processing element, but different parts of the data are read and processed by all processing elements in parallel. This



(a) Overview of CaffeOnSpark

(b) Overview of TensorFlowOnSpark

(c) Overview of MMLSpark (CNTKOnSpark)

(d) Overview of BigDL

Fig. 2. Architecture overview of DLoBD stacks.

paper focuses on data parallelism, which is more related to system-level studies. This paper chooses four popular and representative DLoBD stacks (i.e., CaffeOnSpark, TensorFlowOnSpark, MMLSpark or CNTKOnSpark, and BigDL) which support data parallelism to conduct the detailed analysis. We first compare the architecture of these four systems in the following sections.

2.1 CaffeOnSpark Overview

CaffeOnSpark is a Deep Learning package designed by Yahoo! based upon Apache Spark and Caffe. It inherits features from Caffe like computing on CPU, GPU, and GPU with accelerating components (e.g., cuDNN). CaffeOnSpark enables Deep Learning training and testing with Caffe to be embedded inside Spark applications. Such an approach eliminates unnecessary data movement and benefits Deep Learning from the high performance and scalability of Hadoop and Spark clusters. For example, Flickr team improved image recognition accuracy significantly with CaffeOnSpark by training with the Yahoo Flickr Creative Commons 100M[3] dataset.

The system architecture of CaffeOnSpark (YARN cluster mode) is illustrated in Fig. 2a. CaffeOnSpark applications are launched by standard Spark commands, and then Hadoop YARN launches a number of containers for running Spark executors. After Spark executors are running, there are two approaches to manage training and testing data. One is the local DB-based approach, in which the Spark driver reads database file from HDFS, loads it into local database instance (e.g., LMDB[4]), and then transforms the data inside local database into RDD. The other approach is HDFS-based, which means that the Spark executors fetch training and testing data directly from HDFS. However, in the HDFS-based approach, the raw data needs to be converted into sequence files or DataFrame format. After Spark executors are running and all data are ready, Caffe engines on GPUs or CPUs are setup within Spark executors. The Caffe engine is then being fed with a partition of training data (i.e., data parallelism). After back-propagation of a batch of training examples, the Model Synchronizer will exchange the gradients of model parameters via allreduce style interface over either RDMA or TCP. At the end of each

[3] https://webscope.sandbox.yahoo.com/catalog.php

[4] https://symas.com/lmdb/technical/

CaffeOnSpark application, the final model will be stored on the HDFS.

As we can see, CaffeOnSpark can integrate different components from Deep Learning, Big Data, and HPC community to work together for solving artificial intelligence problems. Default Caffe could not scale out efficiently, but with the help from Hadoop YARN and Spark frameworks, the scaling-out issue can be solved properly. In the meantime, by leveraging HDFS, the data sharing, locality-aware data access, and fault tolerance can be handled automatically as well. All of these are the benefits coming from the DLoBD approach.

Another observation needs to be pointed out is that in CaffeOnSpark, the model synchronizer is designed in a way that it fully bypasses the default Spark data communication or shuffle architecture. This means the parameter exchanging phase is implemented in an out-of-band fashion and there are dedicated communication channels (either RDMA or TCP/IP based) being initialized in the model synchronizers. This approach needs extra effort from the community to maintain these dedicated channels separately and it can not take advantage of the optimizations for default Spark in the Spark community.

2.2 TensorFlowOnSpark Overview

TensorFlow has been seen as one of the most popular Deep Learning frameworks for both academia and industry. Vanilla TensorFlow does not provide support for training over Big Data stacks. SparkNet [4] and TensorFrame [13] are some of the initial efforts in the direction but still leave a lot to be desired regarding the features provided. Thus, Yahoo! researchers take their experience from developing CaffeOnSpark to come up with TensorFlowOnSpark, a framework that enables execution of Deep Learning jobs using TensorFlow on an existing Big Data cluster using Spark and Hadoop YARN to distribute the training and includes support for RDMA over high-speed networks.

TensorFlowOnSpark seamlessly integrates along with other Spark components such as SparkSQL, MLlib, etc. in the overall Spark ecosystem, requiring minimal changes to default TensorFlow code. Fig. 2b presents the architecture overview of TensorFlowOnSpark. TensorFlowOnSpark allows Spark Executors acting as containers used to run TensorFlow code. It provides two different modes to ingesting data; *QueueRunners* are used to read data directly from HDFS using built-in TensorFlow modules whereas *Spark Feeding* provides the data from Spark RDDs to Spark executors, which in turn feed it to the TensorFlow core.

Similar to CaffeOnSpark, TensorFlowOnSpark also bypasses the Spark architecture for communication (i.e., out-of-band communication) therefore achieving similar scalability as standalone TensorFlow jobs. The default TensorFlow officially can support three different channels for data communication, including gRPC, gRPC+Verbs (RDMA), and gRPC+MPI. When utilizing RDMA, tensors are written directly to the memory of remote processes bypassing kernel space. This design provides considerable performance boost compared to the default gRPC design. One different design compared to the architecture of CaffeOnSpark is that TensorFlowOnSpark is based on the Parameter Server approach. The parameter server(s) will be setup embedded inside one or some Spark executor(s) and talk to other tensors over gRPC, gPRC with RDMA, or gRPC with MPI.

As we can see here, TensorFlowOnSpark can also integrate different components from Deep Learning, Big Data, and HPC community, even though there are some differences in the architecture compared to CaffeOnSpark. More in-depth analysis about TensorFlow will be discussed in Section 5.

2.3 MMLSpark Overview

MMLSpark (or CNTKOnSpark), proposed by Microsoft, is a powerful toolkit for Apache Spark in accelerating Deep Learning and data science. It turns parallelizable algorithms from external libraries (e.g., Microsoft Cognitive Toolkit and OpenCV) into Spark Machine Learning pipelines without data transfer overhead, therefore enables one to quickly create powerful, high-scalable predictive and analytical models for large image and text datasets.

Vanilla MMLSpark is designed for Microsoft Azure Cluster and can be installed on Azure HDInsight Spark Cluster conveniently with user-friendly documents. However, we take some efforts to make it work on top of HDFS instead of Windows Azure Storage Blob (WASB) and compatible with the HPC cluster used for evaluation. Similar as CaffeOnSpark and TensorFlowOnSpark, MMLSpark can also run over Big Data Stacks, like Hadoop YARN, Spark, and HDFS.

The architecture of MMLSpark is depicted in Fig. 2c. The feeding data for CNTK Core (e.g., images or texts) can be directly read from HDFS by Spark Executors. The CNTK Model loads pre-trained model and distributes the model to multiple workers for parallel evaluation. Similar to CaffeOnSpark and TensorFlowOnSpark, MMLSpark employs out-of-band communication (e.g., bypassing the Spark architecture) approach in exchanging model parameters among multiple workers for parallel model evaluation. The out-of-band communication is implemented with MPI library, so that users can specify the employed communication channel (e.g., TCP/IP or RDMA) in compiling stage or runtime depending on which MPI library is chosen to run with. In this paper, MMLSpark is compiled with OpenMPI library since the MMLSpark package has been tightly coupled with OpenMPI. The communication channel is switched between IPoIB and RDMA through configuring the Byte Transfer Layer (btl) in OpenMPI runtime.

2.4 BigDL Overview

BigDL is proposed by Intel to provide a high-performance and distributed Deep Learning runtime which makes efficient use of Intel processors and co-processors (such as Intel Xeon Phi). BigDL uses Spark to scale out to multiple processes. It allows users to import models already trained using Caffe and Torch into the Spark framework, which is then used to pipe the models to the BigDL runtime. BigDL is written using Intel's Math Kernel Library (MKL) which provides optimized support for vector primitives frequently used in Deep Learning applications. Therefore, it significantly outperforms most Deep Learning frameworks out-of-the-box on a single node Intel Xeon Phi processor.

Fig. 2d shows the architecture of BigDL is also based on Parameter Server that is organically designed with Spark Block Manager component. The Spark Block Manager is

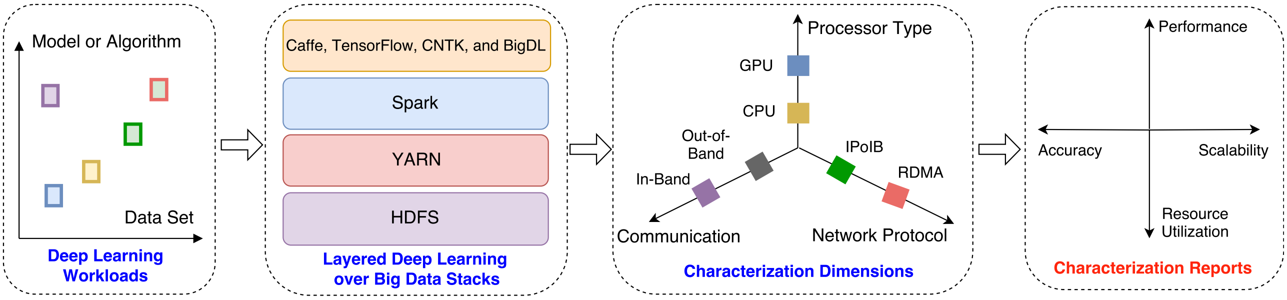


Fig. 3. Evaluation methodology.

heavily involved in the Spark Shuffle architecture. The feeding data to BigDL core is ingesting by Spark Executor which can directly load data from HDFS. Parameter updates of the training model (i.e., major communication phase) are exchanged among BigDL cores through parameter server which is based on Spark shuffle architecture. We call this kind of parameter exchanging approach as in-band communication, which is different than the out-of-band approach (i.e., bypass Spark shuffle) applied in CaffeOnSpark, TensorFlowOnSpark, and MMLSpark. In other words, the in-band communication approach directly utilizes the Spark shuffle engine to synchronize the model. By default, BigDL on Spark does not support RDMA-based model synchronization because the default Spark does not support RDMA-based shuffle. However, our recent work [10] can support native verbs-based high-performance RDMA shuffle in Spark, which can be used to fill this gap for BigDL. This can also be seen as the benefit of choosing in-band communication approach. With in-band communication approach, the BigDL framework can automatically utilize all kinds of optimizations in Spark core (like the RDMA-based shuffle engine), which are available in the Spark community. On the other hand, with in-band communication, the BigDL researchers and developers do not need to maintain their separate communication channels or subsystems.

2.5 Summary

To summarize, these four DLoBD stacks are designed differently, and all of them can take advantage of modern HPC technologies (e.g., multi-core CPUs, GPUs, RDMA, etc.) in varied ways to boost Deep Learning performance. In the meantime, all of them can run on top of the same Big Data stacks (i.e., Spark, Hadoop). These commonalities, as well as their differences, make us choose them to represent a broad range of DLoBD stacks to be investigated in this paper.

3 Characterization Methodology

This section describes our proposed characterization methodology on evaluating DLoBD stacks.

3.1 Methodology Overview

To systematically characterize DLoBD stacks, we propose a holistic evaluation methodology as shown in Fig. 3. The characterization methodology comprises four main aspects. First of all, we conduct an extensive survey on selecting the typical Deep Learning workloads, including popular Deep Learning models and open datasets. We need to make sure the selected models have varied sizes to cover big and small models. Similarly, for datasets, we need to choose both small and large ones. Thus, we could cover different kinds of combinations, such as training varied-size models on both small and large datasets, which could expose more different characterization trends. Since the importance of workload selection, we will give detailed descriptions on selected benchmarks and datasets in Section 3.2.

Second, as discussed in Section 2, we choose to run the Deep Learning workloads on four DLoBD stacks (i.e., CaffeOnSpark, TensorFlowOnSpark, MMLSpark, and BigDL). These four stacks can run on the same underlying environment, including Spark engine, YARN scheduler, and HDFS file system, which are the most popular components for Big Data processing. The evaluations on these stacks will give bigger impact to more researchers and developers in this community.

Third, to organize the experiments properly, we fully take into account three major evaluation dimensions, such as processor type, network protocol, and communication approach in different stacks. For processor type, we first verify the effect of powerful computing accelerators (such as NVIDIA GPU) and multi-core CPUs (such as Intel Broadwell and Haswell) on DLoBD stacks. We want to find out with the highly optimized libraries on GPUs (e.g., cuDNN) and CPUs (e.g., Intel MKL), how will DLoBD stacks perform with typical Deep Learning workloads? For network protocol, we focus on investigating the impact of IPoIB and RDMA on Deep Learning workloads. CaffeOnSpark and TensorFlowOnSpark have RDMA support by default, MMLSpark can leverage RDMA-based MPI library, while BigDL can run with our designed RDMA-Spark [10] to support RDMA-based Deep Learning. Thus, through evaluating these four stacks, we can assess the benefits of RDMA with different RDMA-based communication engine designs for Deep Learning workloads. For the communication approach, we characterize both out-of-band (i.e., CaffeOnSpark, TensorFlowOnSpark, and MMLSpark) and in-band (i.e., BigDL) based communication subsystem designs in DLoBD stacks.

Last but not least, we need to show evaluation reports and analysis in detail based on all the evaluations. Even though the performance is the most important metric in our evaluation, we also care about other metrics, such as accuracy, scalability, and resource utilization. For performance, we will explore three major factors: 1) *end-to-end* model training and testing time (i.e., benchmark job execution time), 2) consumed time to reach a certain *accuracy*, and 3) *epoch-level* execution time. We believe all these factors and metrics are the major aspects for characterizing Deep Learning workloads. From our detail reports, we also want to

TABLE 1
Image Classification Datasets

	MNIST	CIFAR-10	ImageNet
Category	Digit Classification	Object Classification	Object Classification
Resolution	28 × 28 B&W	32 × 32 Color	256 × 256 Color
Classes	10	10	1,000
Training Images	60 K	50 K	1.2 M
Tesing Images	10 K	10 K	100 K

excavate more observations to guide designing efficient next-generation DLoBD stacks.

3.2 Benchmarks and Data Sets

To characterize DLoBD stacks, we have chosen three popular datasets: MNIST,[5] CIFAR-10,[6] and ImageNet,[7] which have different categories, resolutions, classes or scales as shown in Table 1.

MNIST consists of 70K black and white handwritten digit images, which have been size-normalized and centered in a fixed-size 28 × 28. Even though the focus of research has moved on to other much more challenging image recognition problems, the fast speed of training on the MNIST dataset means that it is still a proper problem for evaluation purpose.

CIFAR-10 has 50 K training images and 10K test images, which are 32 × 32 RGB images in ten classes. The ten classes include airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Nearly all Deep Learning frameworks use the CIFAR-10 dataset as one example, and there are many accuracy results reported publicly on it. Hence, the CIFAR-10 dataset is one of the most popular choices to evaluate object recognition algorithms.

ImageNet refers to the dataset for ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012. As in 2012, the ILSVRC competition involved a training set of 1.2 million 256 × 256 color images, in 1,000 categories. The ImageNet problem is one of the most challenging object recognition problems for modern computer vision research and Deep Learning research. Because of the long lasting training time of complex models on the ImageNet dataset, evaluating Deep Learning frameworks on it becomes one of the best choices.

Based on the selection of datasets, we use seven well-known trained models: LeNet [14], SoftMax Regression [15], CIFAR-10 Quick,[8] VGG [16], AlexNet [17], GoogLeNet [18] and ResNet [19]. These models, as depicted in Table 2, differing in general architecture and dataset, may offer different insights in the evaluation of DLoBD stacks. The combinations of models and datasets are illustrated in Table 2, which can generally be grouped into three categories: 1) Simple and "shallow" model with small dataset, such as LeNet with MNIST dataset and CIFAR-10 Quick with CIFAR-10 dataset, 2) Complex and "deep" model with small dataset, like VGG with CIFAR-10, and 3) Complex and "deep" model with large dataset, e.g., AlexNet and GoogLeNet with ImageNet dataset. Among all these models, only ResNet is trained on the synthetic data generated by the TensorFlow CNN benchmark.[9] The script generates synthetic data similar to the data expected by ResNet for ImageNet dataset.

Generally speaking, there are not so many parameters involved in simple and "shallow" models. For example, the LeNet model has total 431 K weights. On the other hand, complex and "deep" models will generate tons of parameters during training time. GoogLeNet, a 22-layer model with complicated Inception modules performing different sizes of convolutions, outputs 7 million weights in all layers. In DLoBD stacks, those model parameters need to be exchanged among all workers. Thus, the model complexity influences the performance of communication subsystem in DLoBD stacks significantly. With the purpose of evaluating DLoBD stacks, we finally select these models, and their detailed descriptions can be found in Table 2. In this table, we also indicate which dataset is used for each model in this paper and which framework has the corresponding model implementation in their official distributions. With our survey, we believe that this paper has covered a large range of available various models and datasets in the Deep Learning community.

4 Performance Evaluation

This section presents detailed characterization results.

4.1 Experimental Setup

(1) *OSU RI2 Cluster (Cluster A)*: The RI2 cluster at The Ohio State University comprises 20 nodes connected via Mellanox single port InfiniBand EDR (100 Gbps) HCA. Each node is equipped with two Intel Broadwell (E5-2680-V4) 14-core processors, 128 GB RAM, 120 GB local HDD, and NVIDIA Tesla K80 GPU.

(2) *SDSC Comet Cluster* [20] *(Cluster B)*: The Comet supercomputing system at San Diego Supercomputer Center (SDSC) has 1,984 nodes. We use up to 17 nodes in the evaluation. Each node is provisioned with Intel Haswell (E5-2680-v3) dual twelve-core processors, 128 GB RAM, 320 GB local SSD. The network topology of Comet is FDR (56 Gbps) InfiniBand with rack-level full bisection bandwidth and 4:1 oversubscription cross-rack bandwidth.

Table 3 describes all used software for four different stacks and which cluster is used for the evaluation. For experiments in this section, if not specified, the number of nodes and batch size have such a relation: #node × batch size $= 128$.

4.2 Evaluation on CPU versus GPU

To characterize the performance of CPU and GPU based Deep Learning solutions on DLoBD stacks, we conduct four kinds of experiments on Cluster A with the CIFAR-10 Quick model on the CIFAR-10 dataset and the LeNet model on the MNIST dataset: 1) CPU + OpenBLAS, 2) CPU + MKL, 3) GPU, and 4) GPU + cuDNN. For these experiments, we first run them with IPoIB protocol to expose possible

[5] http://yann.lecun.com/exdb/mnist/
[6] https://www.cs.toronto.edu/~kriz/cifar.html
[7] http://www.image-net.org
[8] https://github.com/yahoo/CaffeOnSpark/blob/master/data/cifar10_quick_train_testprototxt
[9] https://github.com/tensorflow/benchmarks

TABLE 2
Selected Deep Learning Models and Algorithms

Model	Layers (Convolutional/ Full-connected)	Dataset	Description	Framework
LeNet	2 / 2	MNIST	A CNN designed for handwritten and machine-printed character recognition	CaffeOnSpark, TensorFlowOnSpark
SoftMax Regression	NA / NA	MNIST	A logistic function that compresses a vector to another vector of real values in the range (0, 1) that add up to 1	TensorFlowOnSpark
CIFAR-10 Quick	3 / 1	CIFAR-10	A model reproduced from Alex Krizhevsky's cuda-convnet	CaffeOnSpark, TensorFlowOnSpark, MMLSpark
VGG-16	13 /3	CIFAR-10	A deep convolutional network for object recognition	BigDL
AlexNet	5 / 3	ImageNet	A CNN architecture designed to deal with complex object classification task, won ILSVRC 2012	CaffeOnSpark
GoogLeNet	22 /0	ImageNet	A CNN architecture with an Inception module, won ILSVRC 2014	CaffeOnSpark
ResNet	53 /1	Synthetic	A deep convolutional network based on residual learning framework	TensorFlow

communication bottlenecks. As shown in Fig. 4, the scale of experiment cluster is up to 16 nodes, with one device (CPU or GPU) used per node in training and testing models. The End-to-End time consumed by experiments, as represented by the y-axis in the figure, includes training time and testing time.

The results of CIFAR-10 Quick experiments with CaffeOnSpark are shown in Fig. 4a. The leftmost bars depict the performance of these solutions on one node, which means no communication overhead is involved in. We observe that the solution of CPU + MKL has a 63 percent performance improvement compared with that of CPU + OpenBLAS. This makes sense as the Intel MKL library takes advantage of special features provided by Intel CPUs like Intel AVX2 that greatly speed up matrix calculations. While GPU without cuDNN is 323.6 percent faster than CPU + MKL, and 1723.5 percent faster if cuDNN is deployed. However, once we scale the cluster larger than one node, which indicates that more communication is introduced into, the situation becomes complicated. The slower solutions, such as CPU + OpenBLAS, CPU + MKL, and GPU, will benefit from the scalability of DLoBD stacks, and finally, reach the bottleneck of the network. From the quantitative perspective, compared with the performance on one node, CPU + OpenBLAS on 16 nodes has a 78.3 percent performance improvement, while 41.5 percent for CPU + MKL, and 15.9 percent for GPU. On the other hand, the performance of the fastest solution, e.g., GPU + cuDNN, is degraded while the DLoBD stack is run at a larger scale, which means for the case with fast Deep Learning library like cuDNN, we need to exploit the powerful local computation capability on GPU as much as possible rather than blindly scaling it out.

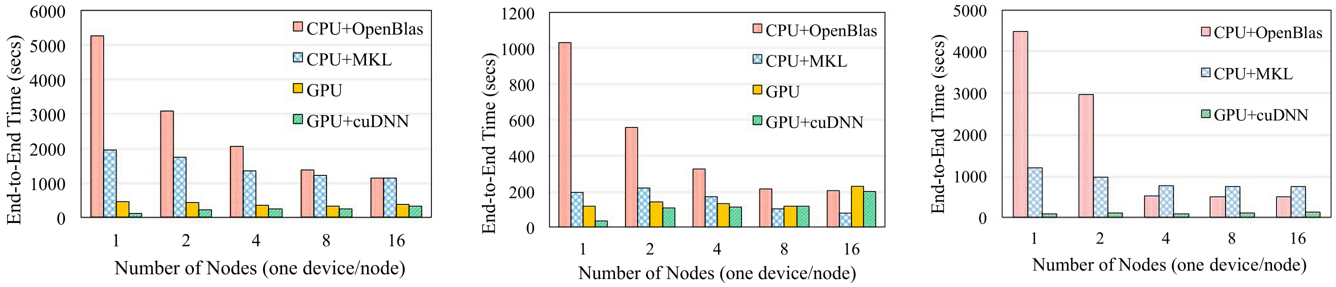
For LeNet experiments with CaffeOnSpark, Fig. 4b shows a different observation that CPU + MKL performs better than GPU and GPU + cuDNN on 8 and 16 nodes. These results indicate two insights. First, the CPU + MKL based solution can perform well on Intel processors for Deep Learning models. Second, the design of the communication library (for TCP/IP-based communication, i.e., IPoIB) used in synchronizing model parameters among CPUs has less overhead, at least for training LeNet model, than the one used among GPUs. More specifically, for training LeNet model on one node, CPU + MKL improves 81.3 percent than CPU + OpenBLAS and is worse than GPU by 1.68x and GPU + cuDNN by 5.8x. But if the same job is running on 16 nodes, the overall time is reduced by 5.02x and 2.43x for CPU + OpenBLAS and CPU + MKL, respectively. For GPU and GPU + cuDNN, however, the overall time is increased by 2x and 5.92x, respectively. This is also similar to the observed trend of CIFAR-10 results.

While in CIFAR-10 experiments with MMLSpark, there are no results for the basic GPU case in Fig. 4c, since CNTK must be compiled with cuDNN if CUDA is enabled. For single node experiments, similar to CIFAR-10 Quick experiment with CaffeOnSpark, the solution of GPU + cuDNN performs best, 55x faster than CPU + OpenBLAS, and 15x faster than CPU + MKL. As aforementioned, once the scale of the cluster becomes larger, there will be extra communication occurring among multiple nodes. The same insight as with CaffeOnSpark is observed in the experiments with MMLSpark that slower solutions (e.g., CPU + OpenBLAS, CPU + MKL) benefit from the scalability of DLoBD stacks,

TABLE 3
Used Software and Clusters

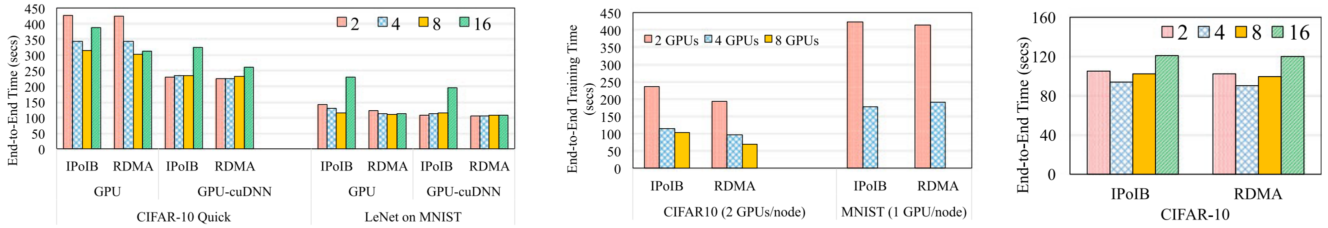
Stack	Software	Cluster
CaffeOnSpark (master branch†)	Java-7 Python-2.7 Spark-1.6 Hadoop-2.6.5	A
TensorFlowOnSpark (1.0)	Java-8 Python-2.7 Spark-2.1 Hadoop-2.7.3	A
MMLSpark (0.10)	Java-8 Python-2.7 Spark-2.2.0 Hadoop-2.7.3	A
BigDL (master branch‡)	Java-8 Scala-2.11 Spark-2.1 Hadoop-2.7.3	B

† *Commit hash: 19df500abe3f0d09511b6434a0ea0bb52a6e8124.*
‡ *Commit hash: a1f3a88517b1b41a9d3554b8715987c66edccfb7.*



(a) CIFAR-10 Quick with CaffeOnSpark (b) LeNet on MNIST with CaffeOnSpark (c) CIFAR-10 with MMLSpark

Fig. 4. Performance comparison for CPU-/GPU-based deep learning with CaffeOnSpark and MMLSpark (Cluster A).



(a) CaffeOnSpark (1 GPU/Node) (b) TensorFlowOnSpark (c) MMLSpark (1 GPU/Node)

Fig. 5. Performance comparison for IPoIB and RDMA with CaffeOnSpark, TensorFlowOnSpark and MMLSpark (Cluster A).

and yet the performance of faster solutions (e.g., GPU + cuDNN) is degraded by the overheads introduced by communication. Quantitatively, the performance of GPU + cuDNN degrades by up to 49.7 percent at a scale of 16 nodes, while the performances of CPU + OpenBLAS and CPU + MKL are improved by 88.8 and 37.9 percent, respectively. However, we observe that CPU + OpenBLAS outperforms CPU + MKL with more than two nodes, which is not demonstrated in experiments with CaffeOnSpark. This could be because CNTK core may use OpenBLAS in a more efficient manner than CaffeOnSpark.

As we can see, Deep Learning frameworks can benefit from the high performance of the DLoBD stacks, even though the overall performance will reach the network bottleneck at some point if we use the sub-optimal IPoIB network protocol. For some cases, solutions with CPU + MKL could outperform GPU-based solutions depending not only on the model, but also on the scale of the system.

4.3 Evaluation on IPoIB versus RDMA

For evaluating DLoBD stacks with IPoIB and RDMA, we conduct three experiments on Cluster A for CaffeOnSpark, TensorFlowOnSpark, and MMLSpark, respectively.

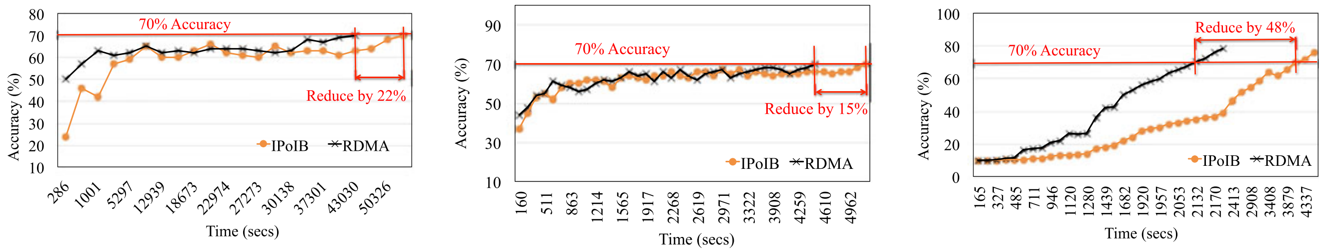
The results of experiments on CaffeOnSpark, presented in Fig. 5a, show that CaffeOnSpark indeed has communication overhead at the scale of 16 nodes for training CIFAR-10 Quick model over both IPoIB and RDMA, and for training LeNet model over IPoIB. Our observation, however, indicates that CaffeOnSpark benefits from the high performance of RMDA compared to IPoIB once communication overhead becomes significant. To be quantitative, the overall performance of 16 nodes is improved by 14.2 and 13.3 percent with employing RDMA instead of IPoIB in training CIFAR-10 Quick model with GPU and GPU + cuDNN, respectively. The performance is also improved by 51.2 and 45.6 percent for training LeNet model with GPU and GPU + cuDNN over RDMA, respectively.

The results of experiments on TensorFlowOnSpark are presented in Fig. 5b. For CIFAR10 example, we try to scale the single-node, multi-GPU example provided to multi-node, multi-GPU cluster. For this benchmark, RDMA outperforms IPoIB by a significant margin (53.8 percent for 8 GPUs). We can do so for up to 8 GPUs, but beyond that, it could not scale and seems to be running into a race condition (as identified by one of the TensorFlowOnSpark developers[10]) which causes it to crash. The MNIST example provided by TensorFlowOnSpark uses SoftMax Regression model. We observe that for the smaller number of nodes, RDMA outperforms IPoIB by about 4.9 percent, but as we scale to more number of nodes, the performance of RDMA is slightly worse than IPoIB. Moreover, scaling it beyond four nodes causes the job hangs indefinitely. Because of this, further performance numbers could not be taken. These observations suggest that the RDMA design in TensorFlowOnSpark is not fully optimized yet.

From these tests, it appears that TensorFlowOnSpark is a recent step in the right direction. Architecturally it seems to be better designed compared with CaffeOnSpark. However, the implementation for TensorFlowOnSpark is not stable yet, and the examples provided so far have not been designed to scale to multi-node multi-GPU clusters. We believe with the continued effort in this project, TensorFlowOnSpark can become a major player in the DLoBD community. On this front, we provide an in-depth analysis in Section 5 to further understand the internals of TensorFlowOnSpark and the performance bottlenecks.

For MMLSpark, as discussed in Section 2.3, its communication performance heavily depends on the underlying MPI library. We choose OpenMPI 1.10.3 for our experiments as it is the default dependent library for MMLSpark. Fig. 5c depicts the results of CIFAR-10 experiments with MMLSpark

[10] https://github.com/yahoo/TensorFlowOnSpark/issues/81#issu



(a) AlexNet on ImageNet with CaffeOnSpark (b) GoogLeNet on ImageNet with CaffeOnSpark (c) VGG on CIFAR-10 with BigDL

Fig. 6. Performance and accuracy comparison of CaffeOnSpark (Cluster A) and BigDL (Cluster B) with IPoIB and RDMA.

at various scales, which indict that the performance of training over IPoIB is comparable with that of training over RDMA. During the experiments, we monitor the network usage, which helps us verify the communication channel utilized by MMLSpark. We do observe that packages go through IPoIB and RDMA respectively for different experiments, even though the times consumed by training stages of IPoIB and RDMA are nearly the same. The possible reason is that the latency and bandwidth of IPoIB in Cluster A are sufficient for exchanging such an amount of parameters among multiple CNTK instances during training CIFAR-10 model in parallel. However, we cannot verify this reason by training different models as we do not find other available benchmarks to evaluate MMLSpark. To propose useful benchmarks for MMLSpark or other DLoBD stacks could be a potential research direction for the community.

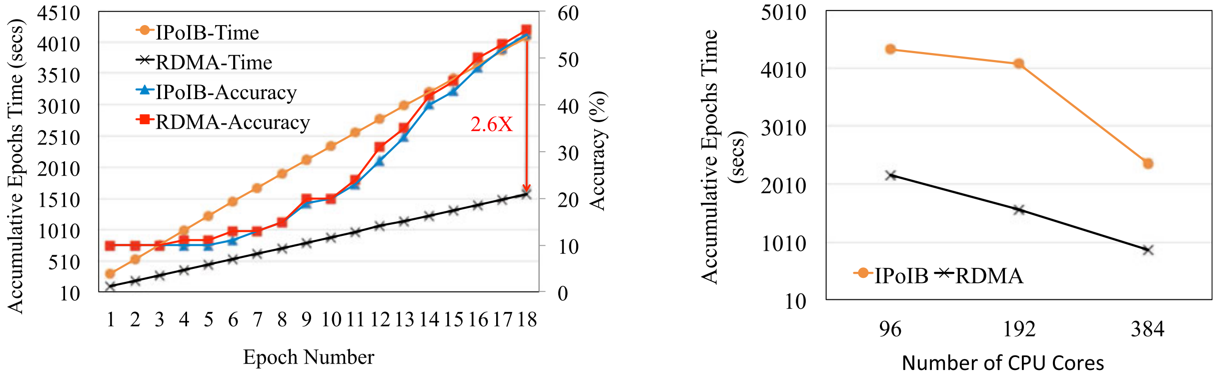
4.4 Evaluation on Performance and Accuracy

To characterize the performance and accuracy of DLoBD stacks with IPoIB and RDMA, three well-known and trained Deep Learning models, such as AlexNet, GoogLeNet and VGG are chosen. Because of the model and dataset combination strategy as mentioned in Section 3.2, three kinds of experiments are designed: 1) AlexNet + ImageNet, 2) GoogLeNet + ImageNet, and 3) VGG + CIFAR-10. The ImageNet referred in this section is a subset consisting of the first ten classes of ILSVRC12 training and validation dataset. Such a choice is because of the physical hardware limitations on Cluster A. In the three experiments, training time to achieve a 70 percent accuracy is the only factor to evaluate the performance of CaffeOnSpark and BigDL. For CaffeOnSpark, Figs. 6a and 6b depict that replacing IPoIB with RDMA reduces the overall time cost by 22 and 15 percent in training AlexNet on ImageNet and training GoogLeNet on ImageNet, respectively.

Fig. 6c shows the performance and accuracy comparison of training VGG model using BigDL on default Spark with IPoIB and our Spark with RDMA. These tests are run on Cluster B. While running BigDL on our RDMA Spark, we observe that the model reaches an accuracy of 70 percent in 2,132 seconds. On the other hand, when the default Spark on IPoIB is used with BigDL, VGG model achieves the same accuracy in 4,337 seconds. Therefore, with the help of RDMA, we can reach the same accuracy in 48 percent less time than IPoIB. As the nature of training Deep Learning models like VGG is communication intensive, RDMA provides a superior solution to train a model when compared to IPoIB.

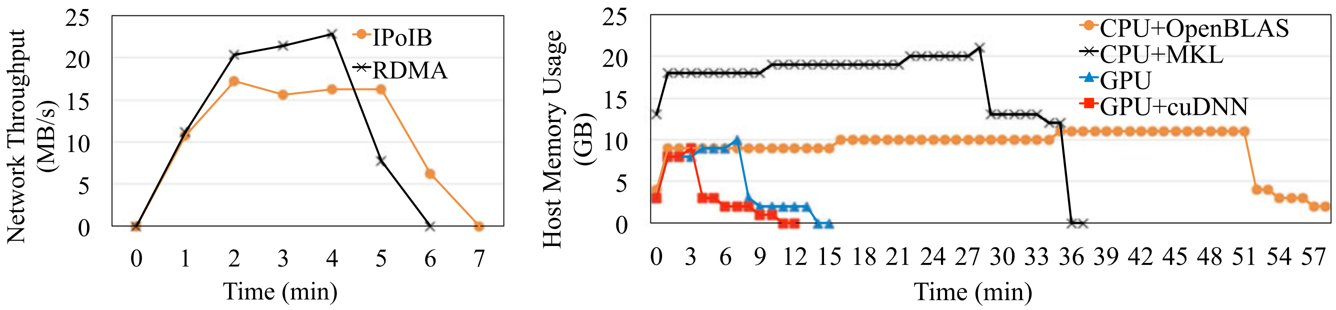
4.5 Epoch-Level Evaluation

In neural network terminology, an epoch can be described as one pass of all the training examples. Fig. 7a shows the epoch level evaluation of training VGG model, on Cluster B, using BigDL on default Spark with IPoIB and our Spark with RDMA. For each epoch, it includes computation (model local training) time and communication time (model synchronization). For these experiments, the total number of CPU cores used is 192, and the batch size is 768. The epoch level evaluation gives a clear picture of the performance comparison of training Deep Learning model with IPoIB and RDMA from the systems perspective. As we can see from Fig. 7a, to finish every epoch, RDMA version takes constantly less time than the IPoIB version. For example, RDMA can reach the end of epoch 18 in 2.6x time faster than IPoIB. This is because with RDMA, the DLoBD stack can perform much faster communication for model



(a) Epoch-Level Evaluation (b) Scalability Evaluation

Fig. 7. Epoch-level and scalability evaluation with BigDL (Cluster B).



(a) Network Utilization

(b) Host Memory Utilization

Fig. 8. Resource utilization comparison (Cluster A).

synchronization than the IPoIB scheme. Thus, RDMA shows much faster progress to reach a certain accuracy.

Interestingly, compared to the time saving (i.e., up to 48 percent) of reaching a certain accuracy, we see higher (i.e., 2.6x) performance improvement with RDMA for epoch-level evaluation. From these numbers, we see that it is important to understand the way how the benefits are reported in Deep Learning related studies, since using different metrics may show quite different numbers.

4.6 Scalability Evaluation

Fig. 7b shows the scalability evaluation of training VGG model on Cluster B by using BigDL on default Spark with IPoIB and on our Spark with RDMA. The figure shows the accumulative time taken to finish the 18th epoch when different numbers of CPU cores are used. We run these tests on (up to) 17 nodes (16 worker nodes and one master node). We observe that when our RDMA spark is used with BigDL to train VGG model, the system scales better than the case when default IPoIB Spark is used with BigDL. Besides, from Fig. 7b, we see with 384 CPU cores and same batch size, RDMA can finish epoch 18 in around 870 seconds. On the other hand, IPoIB takes 2,372 seconds to finish the same number of epochs with the same configuration. Therefore, with RDMA, we can achieve up to 2.7x speedup for the epoch-level training time.

4.7 Evaluation on Resource Utilization

In this section, we first compare two kinds of resource utilization based on the monitoring results in training CIFAR-10 Quick model on the CIFAR-10 dataset with CaffeOnSpark on Cluster A: 1) Network Utilization, 2) Host Memory Utilization. Both utilization results are generated with the average resource consumption per time window of 60 seconds. As shown in Fig. 8a, the RDMA-based design utilizes the network resource more efficiently than the IPoIB-based communication in CaffeOnSpark. The communication library inside CaffeOnSpark benefits from the lower latency and higher achieved throughput of RDMA. It, however, still does not fully utilize the high throughput characteristic of RDMA based on Fig. 8a, which should be more beneficial to DLoBD stacks.

Fig. 8b presents the host memory utilization during training. The two GPU-based solutions consume less host memory than the two CPU-based solutions because they mostly utilize GPU memory. The CPU + MKL solution uses host memory more efficiently and effectively, so it has better performance than CPU + OpenBLAS. In our experiments, we observe that 1.5 GB and 10.9 GB GPU memory are consumed in training with CaffeOnSpark and TensorFlowOnSpark, respectively. The GPU memory utilization is monitored by the command *nvidia-smi* with a time window of 20 seconds. From the results represented in Fig. 8b, we can see that so far, none of these frameworks can utilize both CPU and GPU memory fully and efficiently, which means there is huge performance improvement potential for the community to explore.

5 In-Depth Analysis of TensorFlowOnSpark

Section 4 presents performance characterization of DLoBD stacks almost in a black-box manner. In this section, we further provide an in-depth analysis of TensorFlowOnSpark. We choose TensorFlowOnSpark mainly because TensorFlow has been widely used in many scenarios and we also want to understand the internals of TensorFlowOnSpark. In this section, we focus on two broad analysis aspects. 1) How much performance overhead is brought due to the heavy layers of DLoBD stacks? 2) How the existing communication subsystem being designed in TensorFlow or TensorFlowOnSpark? Are there any potential bottlenecks?

5.1 Understanding Performance Overhead of TensorFlowOnSpark

As depicted in Fig. 2b, TensorFlowOnSpark does not involve Spark drivers in tensor communication. Thus it enables direct tensor communication among different TensorFlow processes such as workers and parameter servers. TensorFlowOnSpark can easily scale by leveraging this Process-to-Process communication. In this way, TensorFlowOnSpark could achieve similar scalability and performance as stand-alone TensorFlow. However, in our performance comparison between TensorFlowOnSpark and native (stand-alone) TensorFlow, we find that there indeed has some overhead involved in different layers of DLoBD stacks.

Fig. 9 shows the time spent in different phases of training in TensorFlowOnSpark and Native TensorFlow, respectively. For this experiment we run the SoftMax Regression model, over MNIST dataset, on a four-node parameter server based cluster, including one parameter server and three workers. Current TensorFlowOnSpark is not matured yet and it runs into some issues if we try to run a larger training job. We keep the batch size 128 and use CPU for the actual training. As we can see from this figure, TensorFlowOnSpark

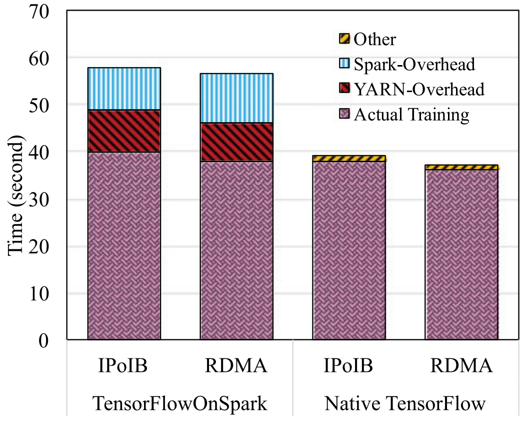


Fig. 9. Performance analysis of TensorFlowOnSpark and native TensorFlow (Lower Better).

spends up to 15.5 percent time in Apache Hadoop YARN scheduler layer, and up to 18.1 percent execution time in Spark job execution layer. We breakdown the times spend in different stages through careful logging and log analysis in different components of TensorFlowOnSpark. We verify these numbers in multiple rounds of tests. From these numbers, we see some performance overhead in both YARN scheduler and Spark execution layer. Since the data size is small, we do not count the time spent on accessing HDFS layer. On the other hand, the Native TensorFlow does not suffer from this extra overhead. These results also indicate that the community may need more effort to reduce this kind of overhead across different layers of DLoBD stacks.

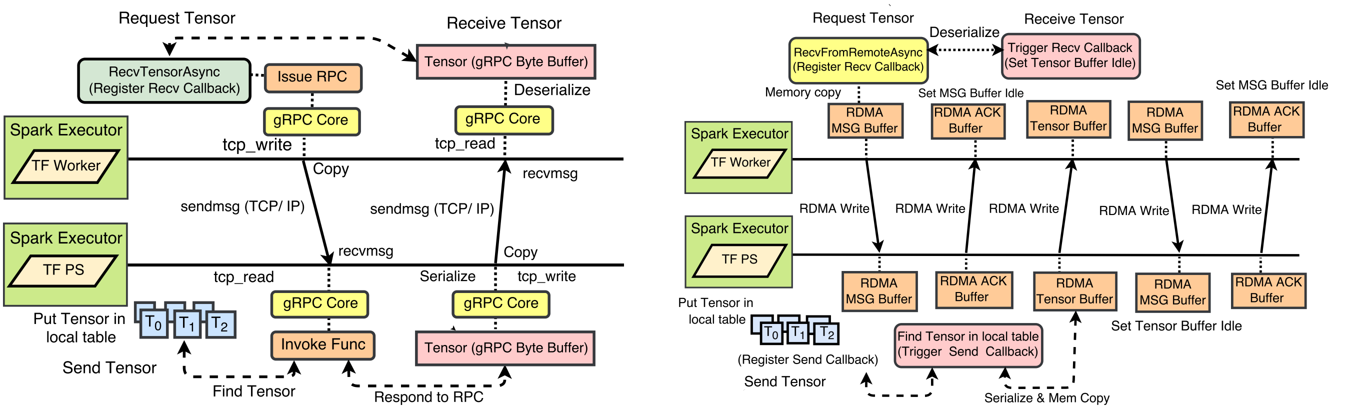
However, note that this overhead can be amortized in long-running Deep Learning jobs, which are the typical cases for many Deep Learning applications. For instance, some Deep Learning models may be trained for many hours or even several days. In these cases, this overhead could be negligible. Not surprisingly, we observe that the actual training time remains similar for both TensorFlowOnSpark and native TensorFlow. This is because TensorFlowOnSpark does not involve Spark core components in tensor communication and it is still mainly relying on the available communication mechanisms in native TensorFlow.

Moreover, from Fig. 9, we notice that RDMA does not improve the training time (about 4 percent) much for this case (i.e., SoftMax Regression model over MNIST dataset). The primary reason is that SoftMax Regression model is simple and thus does not involve too many data-intensive parameter updates. To further understand the difference between the IPoIB-based communication scheme with gRPC and the RDMA-based communication channels (i.e., with Verbs and MPI) in TensorFlow, we further perform more internal analysis in TensorFlow codebase and experiments with native TensorFlow, as described in the following section.

5.2 Understanding Communication Performance in TensorFlowOnSpark

TensorFlow can use gRPC, gRPC+Verbs, and gRPC+MPI based channels for Process-to-Process tensor communication. Fig. 10a shows tensor transfer over gRPC (over TCP/IP or IPoIB) channel. TensorFlow uses a rendezvous protocol for tensor transmission. As shown in this figure, the TF (TensorFlow) worker and PS (Parameter Server) resides on different Spark Executors. The Sender (TF PS in this scenario) always puts the tensors in the local table, whereas the receiver (TF worker) actively requests for the tensor only when it is needed. The default gRPC core uses `sendmsg` and `recvmsg` primitives for sending and receiving payloads. These primitives are useful for sending or receiving from one or more buffers in a single function call. TensorFlowOnSpark directly uses gRPC ByteBuffer to generate the tensor response to avoid extra protocol buffer serialization overhead. From Fig. 10a, we see that the communication path is pretty clean and only one round trip gets involved, which is a good design for TCP/IP protocol. However, for RDMA-capable high-speed networks (like InfiniBand), this design may incur a lot of performance overhead due to the internal buffer copies and context switches, which have been discussed in other related studies [10], [21].

For efficient tensor communication on high-performance networks such as InfiniBand and RoCE, among different processes, TensorFlowOnSpark can use a Verbs-based channel. Fig. 10b represents the Verbs-based tensor communication in TensorFlowOnSpark, from TF Worker to TF PS. The Verbs-based channel transfers all the payloads by employing an RDMA Write operation. TF Worker and PS maintain a set of several pre-pinned RDMA buffers for a Process-to-Process communication. These buffers include two message buffers, two ACK buffers, and many tensor buffers.



(a) Communication Flow in gRPC Channel

(b) Communication Flow in RDMA Channel

Fig. 10. Analysis of tensor transfer over gRPC channel and verbs channel in TensorFlowOnSpark and native TensorFlow.

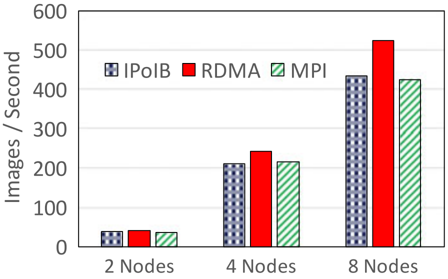


Fig. 11. Performance of native TensorFlow (Higher Better).

Tensor buffers are allocated once at the beginning, and then reused across all training steps of a TensorFlow job to minimize tensor buffer creation. However, when the tensor size increases, the current tensor buffer is discarded and a new buffer of larger size is created and pinned. Upon requesting a tensor, TF Worker sends a message to notify the TF PS. TF PS first sends an ACK so that TF Worker can set the message buffer idle. Then TF PS finds the tensor locally and places at corresponding RDMA tensor buffer for transmission.

From Fig. 10b, we see that the Verbs-based RDMA communication channel could use the RDMA-capable networks in a much more efficient manner than the IPoIB protocol. But it seems like too many communication round trips being involved in the default RDMA design in TensorFlow, which may incur some performance overhead, especially for small message transfers in small model based training. This could be another reason why we do not see too much performance benefit with RDMA in the experiments of Fig. 9. Moreover, TensorFlow has support for MPI-based channel that can leverage RDMA-capable networks for tensor transfers. The communication flow for the MPI-based channel is similar as the flow shown in Fig. 10b. Instead of directly using Verbs-based RDMA operations, the MPI-based design in TensorFlow uses `MPI_Isend`, `MPI_Improbe`, `MPI_MRecv` etc. MPI calls to implement the communication flow. To avoid repetition, we do not show the corresponding communication flow figure for the MPI channel in this paper.

To further understand the impact of RDMA in TensorFlow, we train a more complex deep neural network. We choose ResNet50 [19] for this experiment available in TensorFlow CNN Benchmark. We keep the batch size 64 and use GPU for training. We use 2, 4 and 8 nodes (2 GPUs per node) TensorFlow cluster deployed in parameter server (1 PS and rest workers) mode. The training data is generated by the script in the CNN benchmark suite and the performance is measured in terms of images processed per second. Fig. 11 shows the results of these experiments. From this figure, we see that TensorFlow processes around 8 percent (2 nodes), 15 percent (4 nodes), and 21 percent (8 nodes) more images when RDMA channel is used compared to the scheme of using IPoIB. However, in our experiments we see the MPI-based (tried with both MVAPICH2-2.3b and Intel-MPI-2018) channel performance is similar to the IPoIB channel. The main reason behind these numbers is that the current design of the MPI-based channel in TensorFlow could not use MPI in the best manner. There is a big room for further improvements in the MPI-based channel.

We could not run the same benchmark with TensorFlowOnSpark in Section 5.1, as this CNN benchmark is not available for TensorFlowOnSpark yet.

From the above experiments, we have the following key observations: 1) For complex deep neural network models, we observe RDMA contributes clear benefit for TensorFlow performance. 2) As we increase the number of worker nodes (like from 2 to 8), RDMA can deliver more performance benefit than IPoIB. 3) As observed for both Verbs-based and MPI-based channel performances, designing the communication substrate for TensorFlow in such a way that fully leverages RDMA capabilities also plays an important role. These experimental results indicate that for complex deep neural network models deployed in a larger scale, where the Process-to-Process communication is more frequent during the training time, RDMA can improve TensorFlow performance.

6 Related Work

We summarize and discuss the related work along the following four different categories.

Deep Learning over Big Data Stacks. TensorFlowOnSpark, CaffeOnSpark, MMLSpark, and BigDL have already been discussed in Section 2. However, there have been other efforts in this direction as well. DL4J [5] is an effort to bring Deep Learning to the enterprise, which already has a lot of resources dedicated to a JVM based setup. SparkNet [4] enables users to run TensorFlow jobs in a distributed manner on Spark Executors. TensorFrame [13] integrates Spark DataFrame API with TensorFlow. Our choice of frameworks for this study not only depends on their popularity but also because the selected frameworks support the evaluative characteristics upon which we want this study to be based on. They provide support for RDMA over high-performance interconnects and also support training using GPUs and CPUs on Big Data stacks.

Optimizing Big Data Stacks over High-Performance Networks. High-performance networking technologies such as InfiniBand and RDMA have improved network I/O latency an order of magnitude compared to their Ethernet counterpart. Recently, RDMA-enhanced versions of Hadoop [11], [12], [21], Spark [10] show that Big Data technologies can also exhibit vast performance improvements by utilizing the features provided by these fast networks.

Optimizing DL Frameworks with High-Performance Communication Techniques. To scale out Deep Learning frameworks, HPC capabilities are brought to the Deep Learning arena these days. The Microsoft Cognitive Toolkit [3] is a unified deep-learning toolkit, which implements stochastic gradient descent (SGD, error backpropagation) learning with automatic differentiation and parallelization across multiple GPUs and servers. Ammar et al. propose S-Caffe [22], an MPI-based Caffe design for modern multi-GPU clusters. Zhang et al. propose [23], a high performance plugin for DL frameworks which reduces the communication overhead by optimizing the messages sent to synchronize gradient vectors across nodes. Abhinav et al. [24] extend Google TensorFlow for execution on large-scale clusters using MPI. Moreover, several other reduction-tree based or Allreduce-like approaches [25], [26] have been proposed for TensorFlow as well.

Related Studies on Deep Learning over Big Data. The literature contains a few studies examining the intersection of Deep Learning and Big Data. The authors in [27] survey Deep Neural Networks that have been successfully trained on the Big Data level. The authors in [28] explore Deep Learning algorithms which have been executed on Big Data stacks and identify key areas of research that would help the community better utilize Big Data stacks for Deep Learning workloads. Chi et al. implement a data-intensive machine learning framework called Harp-DAAL [29] intended to enhance Hadoop-based machine learning applications running on HPC platforms. It does this by providing an interface to natively use Intels Data Analytics Acceleration Library (DAAL)[11] to accelerate JVM based data-intensive machine learning applications.

Different than the above-mentioned studies, this paper aims to characterize DLoBD stacks regarding performance, scalability, accuracy, and resource utilization on RDMA-capable high-speed networks and multi-core CPUs/GPUs. This paper extensively extends its earlier version [30] with the following aspects: 1) Add architecture discussions and performance characteristics for MMLSpark; 2) Provide an in-depth performance analysis for TensorFlow and TensorFlowOnSpark, which identify the potential bottlenecks inside the TensorFlowOnSpark stack; 3) More technical discussions and findings are added in this version. All of these differences significantly improve this paper to summarize the characteristics of current-generation DLoBD stacks and provide more interesting future research avenues.

7 Conclusions

This paper first presents a detailed architectural overview of four representative DLoBD stacks (i.e., CaffeOnSpark, TensorFlowOnSpark, MMLSpark, and BigDL) over RDMA-capable high-speed networks. Then, we conduct a comprehensive evaluation of these four stacks to characterize their performance, scalability, accuracy, and resource utilization with typical Deep Learning models and datasets over CPU, GPU, and InfiniBand. Our evaluation reports show the following insights and guidance:

- No matter our RDMA-Spark design or other RDMA-based designs in the community, we see the RDMA scheme can benefit Deep Learning workloads. This paper shows up to 2.7x performance speedup with RDMA compared to the IPoIB scheme for Deep Learning workloads. The RDMA scheme can also scale better and utilize resources more efficiently than the IPoIB scheme over InfiniBand clusters.
- Both GPU and CPU can compute Deep Learning workloads faster with their co-designed efficient Deep Learning oriented libraries, such as cuDNN and Intel MKL. For most cases, GPU-based Deep Learning designs can outperform CPU-based designs, but not always. We see that for LeNet on MNIST, CPU + MKL can achieve better performance than GPU and GPU + cuDNN on 8 and 16 nodes.
- For the same design, we can report performance benefits from two perspectives: time to reach a certain accuracy and consumed time for epoch-level. High-performance schemes (e.g., RDMA) can benefit Deep Learning workloads from both perspectives, but we see higher improvement with RDMA for epoch-level evaluation.
- The current generation DLoBD stacks, like the ones we have evaluated in this paper, still can not utilize all the available cluster resources efficiently. There are still large rooms for them to be further improved. Furthermore, due to lack of standard benchmarks that can run on all these DLoBD stacks, some meaningful performance comparisons among these DLoBD frameworks can not be done easily. These will be the interesting and important future research avenues.

In the future, we plan to investigate more components in DLoBD stacks and propose advanced designs to further improve their performance. In the meantime, we also plan to design and develop more DLoBD benchmarks to help the DLoBD community to perform more types of comparisons such as to show which DLoBD stack should be used in a single-machine or multi-machine scenario.

Acknowledgments

The authors gratefully acknowledge support of this project by US National Science Foundation grants #CNS-1419123, #IIS-1447804, #ACI-1450440, #CNS-1513120, and #IIS-1636846. It used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by US National Science Foundation grant number OCI-1053575.

References

[1] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” in *Proc. 22nd ACM Int. Conf. Multimedia*, 2014, pp. 675–678.

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., “TensorFlow: A system for large-scale machine learning,” in *Proc. 12th USENIX Symp. Operating Syst. Des. Implement.*, 2016, vol. 16, pp. 265–283.

[3] F. Seide and A. Agarwal, “CNTK: Microsoft’s open-source deep-learning toolkit,” in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 2135–2135.

[4] P. Moritz, R. Nishihara, I. Stoica, and M. I. Jordan, “SparkNet: Training deep networks in spark,” *arXiv preprint arXiv:1511.06051*, Nov. 2015, http://adsabs.harvard.edu/abs/2015arXiv151106051M

[5] “Deeplearning4j: Open-Source Distributed Deep Learning for the JVM,” 2018, https://deeplearning4j.org

[6] J. Dai, Y. Wang, X. Qiu, D. Ding, Y. Zhang, Y. Wang, X. Jia, C. Zhang, Y. Wan, Z. Li, J. Wang, S. Huang, et al., “BigDL: A distributed deep learning framework for big data,” *arXiv preprint arXiv:1804.05839*, Apr. 2018, http://adsabs.harvard.edu/abs/2018arXiv180405839D

[7] M. Hamilton, S. Raghunathan, A. Annavajhala, D. Kirsanov, E. de Leon, E. Barzilay, I. Matiach, J. Davison, M. Busch, M. Oprescu, et al., “Flexible and scalable deep learning with MMLSpark,” *arXiv preprint arXiv:1804.04031*, Apr. 2018, http://adsabs.harvard.edu/abs/2018arXiv180404031H

[8] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, “CuDNN: Efficient primitives for deep learning,” *arXiv preprint arXiv:1410.0759*, Oct. 2014, http://adsabs.harvard.edu/abs/2014arXiv1410.0759C

[9] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang, “Intel Math Kernel Library,” in *High-Performance Computing on the Intel® Xeon Phi^{TM}*. Berlin, Germany: Springer, 2014, pp. 167–188.

[10] X. Lu, D. Shankar, S. Gugnani, and D. K. D. K. Panda, “High-performance design of apache spark with RDMA and its benefits on various workloads,” in *Proc. IEEE Int. Conf. Big Data*, Dec. 2016, pp. 253–262.

[11] M. Wasi-ur Rahman, X. Lu, N. S. Islam, R. Rajachandrasekar, and D. K. Panda, “High-performance design of YARN MapReduce on modern HPC clusters with lustre and RDMA,” in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2015, pp. 291–300.

[12] N. S. Islam, X. Lu, M. Wasi-ur Rahman, D. Shankar, and D. K. Panda, “Triple-H: A hybrid approach to accelerate HDFS on HPC clusters with heterogeneous storage architecture,” in *Proc. 15th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2015, pp. 101–110.

[11] https://software.intel.com/en-us/intel-daal

[13] Databricks, “Tensorframes: Tensorflow wrapper for Dataframes on Apache Spark,” 2016, https://github.com/databricks/tensorframes

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[15] W. Chong, D. Blei, and F.-F. Li, “Simultaneous image classification and annotation,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 1903–1910.

[16] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, Sep. 2014, http://adsabs.harvard.edu/abs/2014arXiv1409.1556S

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.

[19] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[20] R. L. Moore, C. Baru, D. Baxter, G. C. Fox, A. Majumdar, P. Papadopoulos, W. Pfeiffer, R. S. Sinkovits, S. Strande, M. Tatineni, et al., “Gateways to discovery: Cyberinfrastructure for the long tail of science,” in *Proc. Annu. Conf. Extreme Sci. Eng. Discovery Environ.*, 2014, Art. no. 39.

[21] X. Lu, N. S. Islam, M. W. Rahman, J. Jose, H. Subramoni, H. Wang, and D. K. Panda, “High-performance design of Hadoop RPC with RDMA over InfiniBand,” in *Proc. IEEE 42nd Int. Conf. Parallel Process.*, Oct. 2013, pp. 641–650.

[22] A. Awan, K. Hamidouche, J. Hashmi, and D. K. Panda, “S-Caffe: Co-designing MPI runtimes and caffe for scalable deep learning on modern GPU clusters,” in *Proc. 22nd ACM SIGPLAN Symp. Principles Practice Parallel Program.*, Feb. 2017, pp. 193–205.

[23] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing, “Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters,” *Usenix Annu. Tech. Conf.*, pp. 181–193, 2017.

[24] A. Vishnu, C. Siegel, and J. Daily, “Distributed TensorFlow with MPI,” *arXiv preprint arXiv:1603.02339*, Mar. 2016, http://adsabs.harvard.edu/abs/2016arXiv160302339V

[25] “Bringing HPC techniques to deep learning.” (2017). [Online]. Available: http://research.baidu.com/bringing-hpc-techniques-deep-learning/

[26] A. Sergeev and M. Del Balso, “Horovod: Fast and easy distributed deep learning in TensorFlow,” *arXiv preprint arXiv:1802.05799*, Feb. 2018, http://adsabs.harvard.edu/abs/2018arXiv180205799S

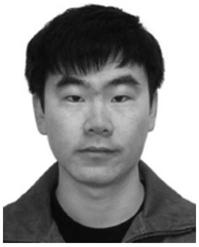
[27] X.-W. Chen and X. Lin, “Big data deep learning: Challenges and perspectives,” *IEEE Access*, vol. 2, pp. 514–525, 2014.

[28] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, “Deep learning applications and challenges in big data analytics,” *J. Big Data*, vol. 2, no. 1, pp. 1–21, 2015.

[29] L. Chen, B. Peng, B. Zhang, T. Liu, Y. Zou, L. Jiang, R. Henschel, C. Stewart, Z. Zhang, E. Mccallum, et al., “Benchmarking Harp-DAAL: High performance Hadoop on KNL clusters,” in *Proc. IEEE 10th Int. Conf. Cloud Comput.*, 2017, pp. 82–89.

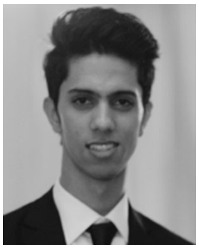
[30] X. Lu, H. Shi, M. H. Javed, R. Biswas, and D. K. Panda, “Characterizing deep learning over big data (DLoBD) stacks on RDMA-capable networks,” in *Proc. IEEE 25th Annu. Symp. High-Perform. Interconnects*, Aug. 2017, pp. 87–94.

Xiaoyi Lu is a research scientist with the Department of Computer Science and Engineering, Ohio State University, USA. His current research interests include high performance interconnects and protocols, big data, Hadoop/Spark/Memcached Ecosystem, parallel computing models (MPI/PGAS), virtualization, cloud computing, and deep learning. He is currently leading the design and development for the High-Performance Big Data (HiBD) project (http://hibd.cse.ohio-state.edu). The HiBD packages are currently being used by more than 285 organizations in 34 countries. More than 26,100 downloads of these libraries have taken place. He has published more than 90 papers in major journals and international conferences related to these research areas and is actively involved in various professional activities in academic journals and conferences. He is a member of the IEEE and ACM. More details are available at http://web.cse.ohio-state.edu/∼lu.932.



Haiyang Shi received the bachelor of engineering degree in computer science and technology from Tianjin University, China. He is working toward the PhD degree in the Department of Computer Science and Engineering, Ohio State University, USA. He is also a graduate research assistant in NOWLAB, led by Dr. D. K. Panda. His research interests include big data, distributed file system, and erasure code. Before joining OSU, he worked as a big data engineer at MiningLamp and Weibo, China.

Rajarshi Biswas received the bachelor of engineering degree in information technology from Jadavpur University, India. He is working toward the master’s degree in Computer Science and Engineering, The Ohio State University. He is also a graduate research assistant in NOWLAB, led by Dr. D. K. Panda. His research interests include distributed computing, deep learning, and big data. Before joining OSU, he worked as a senior software development engineer at Citrix Research and Development, India.



M. Haseeb Javed received the undergraduate degree in software engineering from the National University of Science and Technology (NUST), Pakistan. He is working toward the graduate degree at The Ohio State University and a research assistant at NOWLAB, supervised by Dr. D. K. Panda. His current focus is on systems research cross-cutting the domain of big-data and high-performance computing.

Dhabaleswar K. Panda is a professor of computer science and engineering with Ohio State University. He has published more than 400 papers in major journals and international conferences. He and his research group members have been doing extensive research on modern networking technologies including InfiniBand, High-Speed Ethernet and RDMA over Converged Enhanced Ethernet (RoCE). The MVAPICH2 (High Performance MPI over InfiniBand, iWARP and RoCE) and MVAPICH2-X software libraries, developed by his research group (http://mvapich.cse.ohio-state.edu), are currently being used by more than 2,900 organizations worldwide (in 86 countries). This software has enabled several InfiniBand clusters to get into the latest TOP500 ranking during the last decade. More than 465,000 downloads of this software have taken place from the project’s website alone. The RDMA packages for Apache Spark, Apache Hadoop, and Memcached together with OSU HiBD benchmarks from his group (http://hibd.cse.ohio-state.edu) are also publicly available. These libraries are currently being used by more than 285 organizations in 34 countries. More than 26,100 downloads of these libraries have taken place. His research has been supported by funding from US National Science Foundation, US Department of Energy, US Department of Defense, and several industry including Intel, Cisco, Cray, SUN, Mellanox, QLogic, NVIDIA, Microsoft, and NetApp. He is an IEEE fellow and a member of the ACM. More details are available at http://web.cse.ohio-state.edu/∼panda.2/.