Characterizing Deep Learning over Big Data (DLoBD) Stacks on RDMA-capable Networks

Xiaoyi Lu, Haiyang Shi, M. Haseeb Javed, Rajarshi Biswas, and Dhabaleswar K. (DK) Panda
Department of Computer Science and Engineering, The Ohio State University
{lu.932, shi.876, javed.19, biswas.91, panda.2}@osu.edu

***Abstract*—Deep Learning over Big Data (DLoBD) is becoming one of the most important research paradigms to mine value from the massive amount of gathered data. Many emerging deep learning frameworks start running over Big Data stacks, such as Hadoop and Spark. With the convergence of HPC, Big Data, and Deep Learning, these DLoBD stacks are taking advantage of RDMA and multi-/many-core based CPUs/GPUs. Even though a lot of activities are happening in the field, there is a lack of systematic studies on analyzing the impact of RDMA-capable networks and CPU/GPU on DLoBD stacks. To fill this gap, we propose a systematical characterization methodology and conduct extensive performance evaluations on three representative DLoBD stacks (i.e., CaffeOnSpark, TensorFlowOnSpark, 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 can also scale better and utilize resources more efficiently than the IPoIB scheme over InfiniBand clusters. For most cases, 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. Through our evaluation, we see that there are large rooms to improve the designs of current generation DLoBD stacks further.**

I. 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 [21], TensorFlow [9]) 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 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 those data without moving them back and forth. 3) From 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 [2], SparkNet [28], TensorFlowOnSpark [11], DL4J [6], and BigDL [1]. For DLoBD stacks, many people have the concern about their 'sub-optimal' performance. 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.

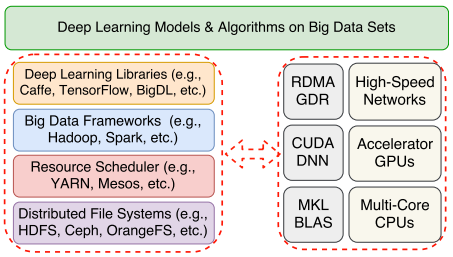


Fig. 1: Characterization Scope of DLoBD Stacks

Figure 1 shows the main components that a typical deep learning job running over DLoDB stacks involves. As we can see, there are at least four major layers: deep learning library layer, big data analytics framework layer, resource scheduler layer, 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 and TensorFlow can leverage the high-performance GPU accelerators with the co-designed efficient cuDNN [18] library, while BigDL can efficiently run on Intel CPUs or Xeon Phi devices by utilizing the highly optimized Intel MKL [33] library or BLAS libraries. Yahoo! researchers have proposed RDMA-based communication in CaffeOnSpark and TensorFlowOnSpark. Our earlier work [20, 24, 26, 34] have proposed RDMA-based designs for Spark and Hadoop. Even though these work have been proposed and well

This research is supported in part by 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 National Science Foundation grant number OCI-1053575.

2332-5569/17 $31.00 © 2017 IEEE
DOI 10.1109/HOTI.2017.24

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:
(1) How are current generation DLoBD stacks and their communication subsystems designed?
(2) Can RDMA-based designs in DLoBD stacks improve performance, scalability, and resource utilization on high-performance interconnects, GPUs, and multi-core CPUs?
(3) What are the performance characteristics of representative DLoBD stacks when they run typical deep learning workloads on RDMA-capable networks?
(4) 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?

To address all of these challenges, this paper first selects three representative DLoBD stacks (i.e., CaffeOnSpark, TensorFlowOnSpark, and BigDL) based on their popularity and designs. We overview their architecture differences and similarities in Section II, which help us to design our characterization methodology. Then, we further propose a systematical characterization methodology in Section III to cover a broad range of evaluation dimensions, such as comparing different networking protocols (i.e., IPoIB vs. RDMA), comparing different ways of integration with big data stacks (i.e., in-band communication vs. out-of-band communication), and comparing solutions using different computing devices (i.e., CPU vs. GPU). Our characterization will focus on four different perspectives, including performance, accuracy, scalability, and resource utilization. Section IV 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. Through our evaluation, we also 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 V discusses related work. We conclude the paper with observed insights and future work in Section VI.

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

A. CaffeOnSpark Overview

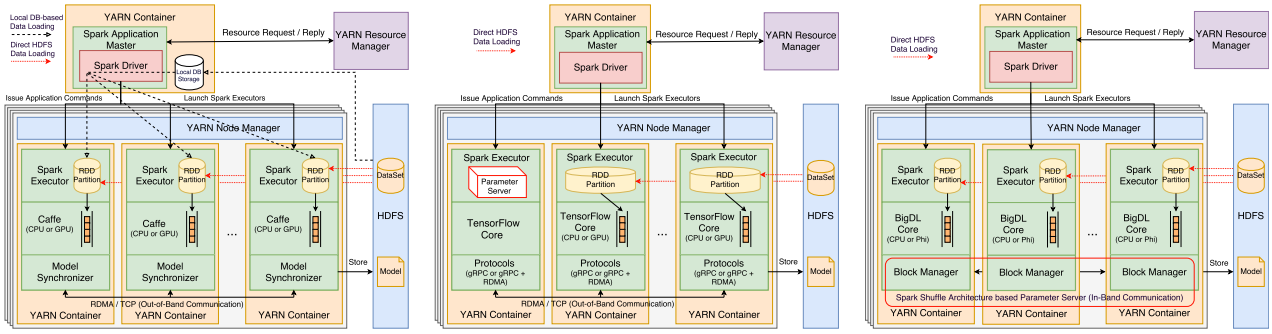
CaffeOnSpark is a Spark deep learning package designed by Yahoo! based upon Apache Spark and Caffe. It inherits features from Caffe like computing on CPU, GPU, and GPU with accelerating components (e.g. cuDNN). CaffeOnSpark enables deep learning training and testing with Caffe to be embedded inside Spark 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 [14] dataset.

The system architecture of CaffeOnSpark (YARN cluster mode) is illustrated in Figure 2(a). CaffeOnSpark applications are launched by standard Spark commands, and then Spark on Yarn launches a number of 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 [8]), 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 CaffeOnSpark application, the final model will be stored on the HDFS.

B. TensorFlowOnSpark Overview

Vanilla TensorFlow does not provide support for training over Big Data stacks. SparkNet [28] and TensorFrame [10] are some of the initial efforts in the direction but left a lot to be desired regarding the features provided. Thus, Yahoo! took 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 to distribute the training and includes support for RDMA over capable 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. Figure 2(b) 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;



(a) Overview of CaffeOnSpark (b) Overview of TensorFlowOnSpark (c) Overview of BigDL

Fig. 2: Architecture Overview of DLoBD Stacks

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. One different design compared to the architecture of CaffeOnSpark is that TensorFlowOnSpark is ParameterServer-based approach. The parameter server will be setup embedded inside one Spark executor and talk to other tensors over gRPC or gPRC with RDMA.

C. 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. 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.

Figure 2(c) shows the architecture of BigDL is also based on Parameter Server that is tightly designed with Spark Block Manager component, which is heavily involved during Spark Shuffle. The feeding data to BigDL core are 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. This in-band communication approach is different from the out-of-band approach (i.e., bypass Spark shuffle) applied in CaffeOnSpark and TensorFlowOnSpark. Moreover, default Spark does not support RDMA-based shuffle, which implies BigDL cannot utilize high-performance networks in the most efficient manner by default. However, our recent studies [24, 26] can support native verbs-based high-performance RDMA shuffle in Spark, which can be used to fill this gap for BigDL.

D. Summary

To summarize, these three 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 speed up 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.

III. Characterization Methodology

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

A. Methodology Overview

To systematically characterize DLoBD stacks, we propose a holistic evaluation methodology as shown in Figure 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 III-B.

Secondly, as discussed in Section II, we choose to run the deep learning workloads on three DLoBD stacks (i.e., CaffeOnSpark, TensorFlowOnSpark, and BigDL). These three 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 on this community.

Thirdly, to organize the experiments properly, we fully take into account the 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 some 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, while BigDL can run with our designed RDMA-Spark [24, 26] to support RDMA-based deep learning. Thus, through evaluating these three stacks, we can assess the benefits of RDMA with different RDMA-based communication engine designs for deep learning workloads. For communication approach, we characterize both out-of-band (i.e., CaffeOnSpark, TensorFlowOnSpark) and in-band (i.e., BigDL) based communication 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 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 excavate more observations to guide designing efficient next-generation DLoBD stacks.

B. Benchmarks and Data Sets

To characterize DLoBD stacks, we have chosen three popular datasets: MNIST [13], CIFAR-10 [12], and ImageNet [7], which have different categories, resolutions, classes or scales as shown in Table I.

MNIST consists of 70K black and white handwritten digit images, which have been size-normalized and centered in a fixed-size 28 $\times$ 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 50K training images and 10K test images, which are 32 $\times$ 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 $\times$ 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 six well-known trained models: LeNet [23], SoftMax Regression [19], CIFAR-10 Quick [3], VGG [30], AlexNet [22] and GoogLeNet [31]. These models, as depicted in Table III, 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 III, which can be generally 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.

Generally speaking, there are not so many parameters involved in simple and "shallow" models. For example, the LeNet model has total 431K 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 III. 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.

IV. Performance Evaluation

This section presents detailed characterization results.

A. 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, and NVIDIA Tesla K80 GPU.

(2) SDSC Comet Cluster [27] (Cluster B): The Comet supercomputing system at 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 SDD. The network topology of Comet is FDR (56 Gbps) InfiniBand with rack-level full bisection bandwidth and 4:1 oversubscription cross-rack bandwidth.

Table II describes all used software for three 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 $\times$ batch size $= 128$.

B. Evaluation on CPU vs. 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 communication bottlenecks. As shown in Figure 4, the scale of experiment cluster is up to 16 nodes, with one device (CPU or

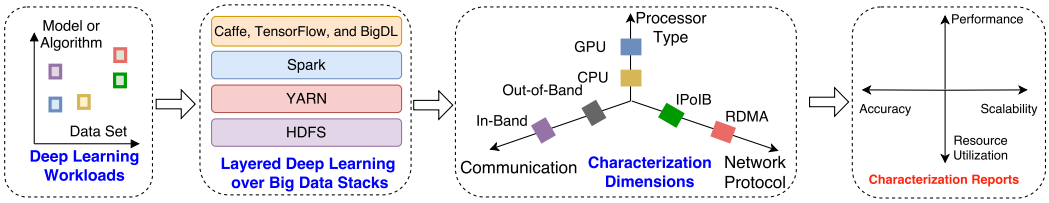


Fig. 3: Evaluation Methodology

TABLE I: 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	1000
Training Images	60 K	50 K	1.2 M
Tesing Images	10 K	10 K	100 K

TABLE II: Used Software and Clusters

Stack	Software	Cluster
CaffeOnSpark (latest 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
BigDL (latest master branch)	Java-8 Scala-2.11 Spark-2.1 Hadoop-2.7.3	B

TABLE III: 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 [5]	CaffeOnSpark, TensorFlowOnSpark
VGG	16 / 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	21 / 1	ImageNet	A CNN architecture with an Inception module, won ILSVRC 2014	CaffeOnSpark

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.

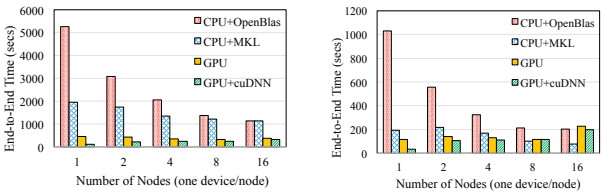
The results of CIFAR-10 Quick experiments are shown in Figure 4(a). The leftmost bars depict the performance of these solutions on one node, which means no communication overhead is involved in. We observe that the solution of CPU + MKL has a 63% performance improvement compared with that of CPU + OpenBLAS. While GPU without cuDNN is 323.6% faster than CPU + MKL, and 1723.5% 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% performance improvement, while 41.5% for CPU + MKL, and 15.9% for GPU. On the other hand, the performance of the fastest solution, e.g. GPU + cuDNN, is degraded while the DLoBD stack is at a large scale.

For LeNet experiments, Figure 4(b) shows a different observation that CPU + MKL performs better than GPU and GPU + cuDNN on 8 and 16 nodes. This result indicates 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% 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.

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 models, solutions with CPU + MKL may outperform GPU-based solutions.

C. Evaluation on IPoIB vs. RDMA

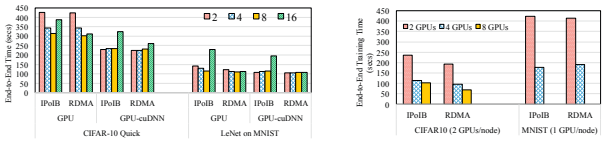
For evaluating DLoBD stacks with IPoIB and RDMA, we conduct two experiments on Cluster A for CaffeOnSpark and TensorFlowOnSpark, respectively. The results of experiments on CaffeOnSpark, presented in Figure 5(a), show that CaffeOnSpark indeed has communication overhead at the scale of 16 nodes for training CIFAR-10 Quick model over both IPoIB



(a) CIFAR-10 Quick (b) LeNet on MNIST

Fig. 4: Performance Comparison for CPU-/GPU-based Deep Learning with CaffeOnSpark (Cluster A)

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% 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% for training LeNet model with GPU and GPU + cuDNN over RDMA, respectively.



(a) CaffeOnSpark (b) TensorFlowOnSpark

Fig. 5: Performance Comparison for IPoIB and RDMA with CaffeOnSpark and TensorFlowOnSpark (Cluster A)

The results of experiments on TensorFlowOnSpark are presented in Figure 5(b). 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% 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 TFoS developers [4]) 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%, 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.

D. 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 III-B, three kinds of experiment are designed: 1) AlexNet + ImageNet, 2) GoogLeNet + ImageNet, and 3) VGG + CIFAR-10. The ImageNet referred in this subsection 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% accuracy is the only factor to evaluate the performance of CaffeOnSpark and BigDL. For CaffeOnSpark, Figure 6(a) and 6(b) depict that replacing IPoIB with RDMA reduces the overall time cost by 22% and 15% in training AlexNet on ImageNet and training GoogLeNet on ImageNet, respectively.

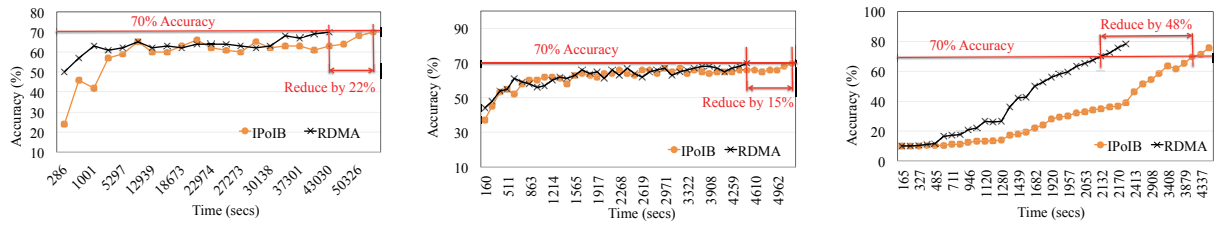
Figure 6(c) shows the performance and accuracy comparison of training VGG model using BigDL on default Spark with IPoIB and our Spark with RDMA. These tests are run on Cluster B. While running BigDL on our RDMA Spark, we observe that the model reaches an accuracy of 70% in 2,132 seconds. On the other 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% 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.

E. Epoch-Level Evaluation

In neural network terminology, an epoch can be described as one pass of all the training examples. Figure 7(a) 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 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 Figure 7(a), 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. Interestingly, compared to the time saving (i.e., up to 48%) of reaching a certain accuracy, we see higher (i.e., 2.6x) performance improvement with RDMA for epoch-level evaluation.

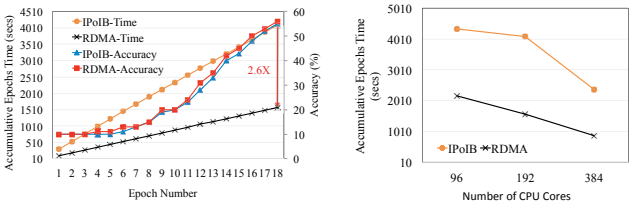
F. Scalability Evaluation

Figure 7(b) 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 observe that when our RDMA spark is used with BigDL to train VGG model, the system scales better than when default IPoIB Spark is used with BigDL. Besides, from Figure 7(b), 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



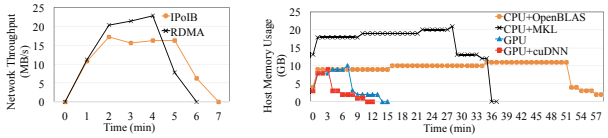
(a) AlexNet on ImageNet with CaffeOnSpark (b) GoogLeNet on ImageNet with CaffeOnSpark (c) VGG on CIFAR-10 with BigDL

Fig. 6: Performance and Accuracy Comparison of CaffeOnSpark (Cluster A) and BigDL (Cluster B) with IPoIB and RDMA



(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)

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.

G. Evaluation on Resource Utilization

In this subsection, 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 by the command **sar**, which outputs the average consumption per time window of 60 seconds. As shown in Figure 8(a), 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 Figure 8(a), which should be more beneficial to DLoBD stacks.

Figure 8(b) 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 Figure 8(b), we can see that so far, none of these frameworks can utilize both CPU and GPU memory fully and efficiently, which means there are huge performance improvement potential for the community to explore.

V. Related Work

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

Deep Learning over Big Data Stacks: TensorFlowOnSpark, CaffeOnSpark, and BigDL have already been discussed in Section II. However, there have been other efforts in this direction as well. DL4J [6] is an effort to bring Deep Learning to the enterprise, which already has a lot of resources dedicated to a JVM based setup. SparkNet [28] enables users to run TensorFlow jobs in a distributed manner on Spark Executors. TensorFrame [10] 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 [20, 25, 34], Spark [24, 26] show that big data technologies can also exhibit vast performance improvements by utilizing the features provided by these fast networks.

HPC-based Deep Learning: To scale out DL frameworks, HPC capabilities are brought to the DL arena these days. The Microsoft Cognitive Toolkit (CNTK) [15] 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 [16], an MPI-based Caffe design for modern multi-GPU clusters. Abhinav et al. [32] extend Google TensorFlow for execution on large-scale clusters using MPI.

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 [17] survey Deep Neural Networks that have been successfully trained on the big data level. The authors in [29] 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. Different than those studies, this paper aims to characterize DLoBD stacks regarding performance, scalability, accuracy, and resource utilization on RDMA-capable networks and multi-core CPUs/GPUs.

VI. Conclusions

This paper first presents a detail architectural overview of three representative DLoBD stacks (i.e., CaffeOnSpark, TensorFlowOnSpark, and BigDL) over RDMA-capable networks. Then, we conduct a comprehensive evaluation on these three 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:

(1) No matter our RDMA-Spark design or the other two RDMA-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.

(2) 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.

(3) 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.

(4) 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.

In the future, we plan to investigate more components in DLoBD stacks and propose new advanced designs.

References

[1] "BigDL," https://github.com/intel-analytics/BigDL.
[2] "CaffeOnSpark," https://github.com/yahoo/CaffeOnSpark.
[3] "CIFAR-10 Quick," https://github.com/yahoo/CaffeOnSpark/blob/master/data/cifar10_quick_train_test.prototxt.
[4] "CIFAR Job With 4 Spark Executors And Above Crashes," https://github.com/yahoo/TensorFlowOnSpark/issues/81#issuecomment-302464649.
[5] "cuda-convnet," https://code.google.com/archive/p/cuda-convnet/.
[6] "Deeplearning4j," http://deeplearning4j.org.
[7] "IMAGENET," http://www.image-net.org/.
[8] "LMDB," https://symas.com/lightning-memory-mapped-database/.
[9] "TensorFlow," http://tensorflow.org/.
[10] "TensorFrames," https://github.com/databricks/tensorframes.
[11] "TesnorFlowOnSpark," https://github.com/yahoo/TensorFlowOnSpark.
[12] "The CIFAR-10 Dataset," https://www.cs.toronto.edu/ kriz/cifar.html.
[13] "THE MNIST DATABASE," http://yann.lecun.com/exdb/mnist/.
[14] "Yahoo Flickr Creative Commons 100M," https://webscope.sandbox.yahoo.com/catalog.php?datatype=i&did=67.
[15] "CNTK," Feb. 2017. [Online]. Available: http://www.cntk.ai/
[16] 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 *22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, February 2017.
[17] X.-W. Chen and X. Lin, "Big Data Deep Learning: Challenges and Perspectives," vol. 2. IEEE, 2014, pp. 514–525.
[18] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient Primitives for Deep Learning," 2014.
[19] W. Chong, D. Blei, and F.-F. Li, "Simultaneous Image Classification and Annotation," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 1903–1910.
[20] 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 *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*. IEEE, 2015, pp. 101–110.
[21] 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 *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.
[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet Classification with Deep Convolutional Neural Networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[24] 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 *2016 IEEE International Conference on Big Data (Big Data)*, Dec 2016, pp. 253–262.
[25] 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 *The Proceedings of IEEE 42nd International Conference on Parallel Processing (ICPP)*, France, October 2013.
[26] X. Lu, M. W. U. Rahman, N. Islam, D. Shankar, and D. K. Panda, "Accelerating Spark with RDMA for Big Data Processing: Early Experiences," in *High-performance interconnects (HOTI), 2014 IEEE 22nd annual symposium on*. IEEE, 2014, pp. 9–16.
[27] 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 *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*. ACM, 2014, p. 39.
[28] P. Moritz, R. Nishihara, I. Stoica, and M. I. Jordan, "Sparknet: Training Deep Networks in Spark," 2015.
[29] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep Learning Applications and Challenges in Big Data Analytics," vol. 2, no. 1. Springer International Publishing, 2015, p. 1.
[30] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-scale Image Recognition," 2014.
[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.
[32] A. Vishnu, C. Siegel, and J. Daily, "Distributed TensorFlow with MPI," *CoRR*, vol. abs/1603.02339, 2016.
[33] 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*. Springer, 2014, pp. 167–188.
[34] 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 *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*. IEEE, 2015, pp. 291–300.