On Realizing Distributed Deep Neural Networks: An Astrophysics Case Study

Maria Aspri†⋆, Grigorios Tsagkatakis⋆, Athanasia Panousopoulou⋆ and Panagiotis Tsakalides†⋆
⋆ *Institute of Computer Science, Foundation for Research and Technology-Hellas (FORTH)*
† *Department of Computer Science, University of Crete, Greece*
Heraklion, 70013, Greece
{aspri, greg, tsakalid}@ics.forth.gr, n.panousopoulou@gmail.com

***Abstract*—Deep Learning architectures are extensively adopted as the core machine learning framework in both industry and academia. With large amounts of data at their disposal, these architectures can autonomously extract highly descriptive features for any type of input signals. However, the extensive volume of data combined with the demand for high computational resources, are introducing new challenges in terms of computing platforms. The work herein presented explores the performance of Deep Learning in the field of astrophysics, when conducted on a distributed environment. To set up such an environment, we capitalize on TensorFlowOnSpark, which combines both TensorFlow's dataflow graphs and Spark's cluster management. We report on the performance of a CPU cluster, considering both the number of training nodes and data distribution, while quantifying their effects via the metrics of training accuracy and training loss. Our results indicate that distribution has a positive impact on Deep Learning, since it accelerates our network's convergence for a given number of epochs. However, network traffic adds a significant amount of overhead, rendering it suitable for mostly very deep models or in big Data Analytics.**

***Index Terms*—Distributed Deep Learning, Convolutional Neural Networks, Spectroscopic Redshift Estimation**

I. Introduction

Deep Neural Network (DNN) architectures [1] are becoming a pervasive tool in modern data analysis, with applications in a plethora of domains, from generic image classification [2], to remote sensing [3] and speech recognition [4]. Furthermore, their potential is actively investigated in scientific data processing problems, including the analysis of astronomical observations [5]. To handle the massive quantities of computations needed for training DNNs, Graphical Processing Units (GPUs) have been extensively employed as a commodity hardware due to their parallel computation capabilities and large memory bandwidth [6]. Despite significant advances in terms of hardware capabilities, the fact remains that single machine setups do not posses sufficient computational resources to efficiently training complex DNNs.

To address this limitation, different philosophies regarding the distributed training of DNNs have been considered, including model parallelization [7] and data parallelization [8], with the latter being the preferable approach, since it provides better fault-tolerance and resource utilization. In data parallelization paradigms, the most prominent examples involve synchronous [9] and asynchronous training [10], while coordination is facilitated by a Parameter Server (PS) [11].

Nevertheless, very little has been reported on how data distribution affects the performance of a distributed DNN. In this work, we are trying to fill this gap by studying the effects of data distribution. The proposed approach capitalizes on TensorFlowOnSpark (TFoS), a platform that allows distributed training and inference on a cluster of machines based on the prolific framework of TensorFlow.

To quantify the performance, we train a Convolutional Neural Network (CNN) for estimating galaxy redshift from simulated spectroscopic measurements which adhere to the publicly available specifications of the European Space Agency Euclid platform [12], [13]. Our study contributes the following:

- Evaluation on whether or not different data on each worker can lead to further improvement of the performance of a DNN.
- Discussion and evaluation of the tradeoffs between distributed and traditional DNNs.
- Quantification the impact of the cluster's size on the performance of Distributed DNNs.

II. Distributed DNN framework

The distributed DNN architecture considered in this work is based on TensorFlowOnSpak[1] [14], for managing TensorFlow-based distributed DNN workflows on Apache Spark clusters [15]. Apache Spark is the mainstream technology for in-memory analytics [16] over commodity hardware. Spark extends the MapReduce model [17] by the use of an elastic persistence model, which provides the flexibility to persist these data records, either in memory, on disk, or both. Therefore, Spark favors iterative processes met in machine learning and optimization algorithms. Briefly, Spark organizes the underlying infrastructure into a hierarchical cluster of computing elements, comprised of a master and a set of N workers. The master is responsible for the configuration of the cluster, while the workers perform the learning tasks submitted to them through a driver program, along with an initial dataset. The partitioning of the dataset relies on the concept of the Resilient Distributed Dataset (RDD), which is defined as a read-only collection of data records. Each learning task is split

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

into sequential stages, performed by each worker on different data blocks. Upon finishing, the results return to the driver which assigns another stage of the learning task to the worker, until all stages have been completed. The results can either stay on the driver or be exported to a storage system.

Opposed to its counterparts (e.g., Distributed TensorFlow [18]) TFoS exploits Spark native mechanisms for the automated configuration of the cluster, while the direct tensor communication favors scalability, simply by adding machines into the cluster. This aspect is also highlighted in in Section III, indicating that the use of TFoS can yield a scalable solution on the problem of galaxy velocity parameter estimation.

In a typical TFoS cluster, each node is coordinated by Spark and acts as a container that locally executes the operations of TensorFlow graphs. A node is randomly selected to act as the PS and is responsible for providing a global average of network parameters, while the rest of the nodes are responsible for training and operating on the RDDs defined by the underlying Spark architecture. TFoS bypasses the communication philosophy of Spark, thereby allowing direct tensor communication between TF processes.

TFoS implements the concept of Parameter Server (PS) for enabling the smooth integration of TensorFlow code over Spark clusters, with minimum changes on the original TensorFlow code. We opted to perform asynchronous training for updating the PS, therefore, an issue known as the stale gradient problem emerged [19]. The stale gradient problem states that when a worker has send its parameters to the PS, the global parameters stored in the PS may have already been updated a number of times by other workers. In order to overcome this issue, instead of updating global parameters immediately, the PS was forced to wait to collect some number s of updates W_j from any of the M training workers, such as $1 \leq s \leq M$. The parameters are then updated according to the following equation:

$$W_{i+1} = W_i - \frac{1}{s} \sum_{j=1}^{s} \lambda(\Delta W_j) \Delta W_j \tag{1}$$

where $\lambda(\Delta W_j)$ is a scalar staleness-dependent scaling factor [19]. Once the variables are updated they are redistributed across the machines, so they can begin the next epoch. Although this update routine mitigates the issue of the stale gradients, it adds overhead to the network since it slows down the overall training process.

III. USE CASE: SPECTROSCOPIC REDSHIFT ESTIMATION

Redshift (z) is parameter encoding the shift of the spectrum of astronomic objects like galaxies towards longer (redder) wavelengths due to the accelerated expansion of the universe. As a result, redshift estimation plays a fundamental role in astronomical signal processing, as it can be used to accurately estimate galaxies' radial distances and their position.

An effective method for accurately predicting redshift estimation is through Deep Learning classification over large amounts of data obtained from spectroscopic observations converted into signals. To classify these signals, we consider a 1D CNN consisting of three layers of convolutions with non-linear activation functions and two fully connected layers, the last of which produces the final classification output. The convolutions employ kernels of length 8 while the Rectified Linear Unit activation function is employed [13].

For our experiments, we produce simulated data, modeled after signals received from real satellites. In order to generate a more realistic dataset, each signal encodes spectroscopic content in the range of 1.1 to 2.0μm which is mapped to 1800 distinct spectral bands. In our setup, we treat the problem of redshift estimation as a multi-class classification problem by dividing the redshift range z = [1 - 1.8) to 800 classes.

In this work, we study the aforementioned problem through a distributed framework. As stated before, our scheme focuses on data parallelization and employs asynchronous training. The simulated dataset, along with their respective labels is given as an input in the form of an RDD bundle, compressed by using the $zipWithIndex$ transformation. The RDD is then split across different machines. Then, the training process is performed locally, by using a replica of the whole CNN. At the end of each epoch, the PS collects the resulting global variables and updates them.

IV. EVALUATION STUDIES

A. Cluster Setup

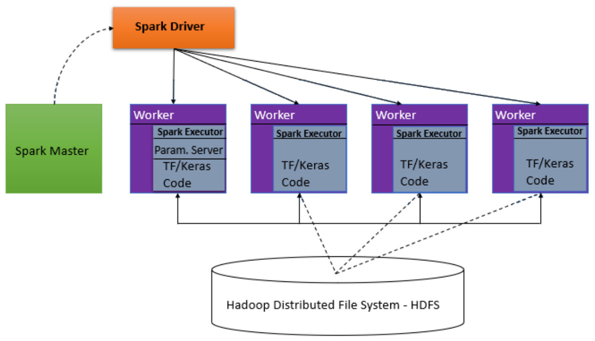


Fig. 1: TFoS Cluster Setup.

To benchmark the proposed framework, the experiments were run on a cluster of 5 PCs, featuring TensorFlow Core version 1.2.1, Apache Spark version 2.1.1, Apache Hadoop version 2.6, and TensorFlowOnSpark version 1.0.0. Master is configured with an Intel Core i7-6700 3.40GHz CPU, and has allocated 8GB memory and 450 GB disk space. Meanwhile, workers are configured with Intel Core i5-3470 3.20GHz CPUs, have allocated 2GB memory and 450 GB disk space.

Figure 1 presents our architecture. It depicts Spark nodes connected via Ethernet. The Master configures the cluster through the driver, and workers communicate through TFoS tensor communication. Hadoop's Distributed File System (HDFS) [20]is configured for hosting the dataset, which is then distributed across all Workers of the cluster distributes RDDs across the nodes responsible for training the CNN.

B. Experimental Results

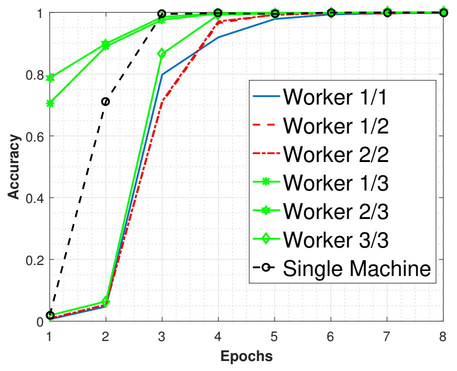
A series of experiments were conducted using two sets simulated noisy data from section III, consisted of 5K and 15K training samples respectively. Our goal is to study the impact of the following parameters: (i) the number of distributed computing nodes, and (ii) the distribution of data among the nodes. Regarding the data distribution, we explore two cases, where in the first case the same data is presented to all working nodes, while in the second case, disjoint sets of examples are utilized by each node. The experiments presented below, were also conducted on an identical non-parallelized CNN, which serves as our baseline.

Figure 2 presents the performance of the CNN as a function of training accuracy. From top to bottom, Figures 2(a),(b) present the results of the 5K samples set for the cases of same and different data, respectively, while Figures 2(c),(d) provide the results for the 15K samples set. We notice that that in most cases, using multiple working nodes has a positive impact in terms of accuracy, for a given number of training epochs. However, adding more machines raises the cluster's total overhead, due to increased network traffic. For the case of two workers, we observe that when same data are used, the performance between 1 and 2 working nodes is similar, while when different sets of examples are considered, there is a significant difference in terms of accuracy by each of the distributed nodes. For the case of three computing nodes, there is a clear benefit in both cases of using the same or different training examples.

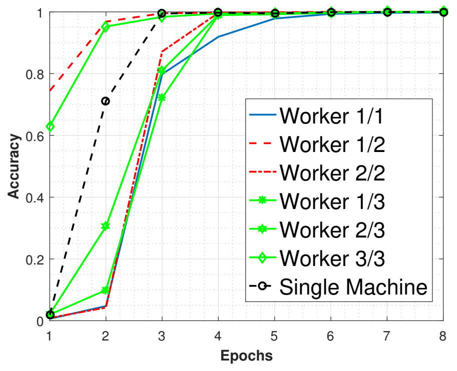
Specifically for the 15K samples set, we observe that the larger amount of training samples has a positive effect on the system's learning capacity and the performance variation across different machines is less prominent. The sequential CNN attains over 75% accuracy faster for this set, at three epochs, but the cluster gains similar behavior as we add more machines. Specifically, in the one-node cluster, the CNN needs about 8 epochs to reach the same accuracy as the sequential case, while for the 3-node cluster the amount of epochs required is reduced to 4.

Figure 3 presents the number of epochs required to reach a stable performance, which is introduced through an early stopping criterion in terms of differences in accuracy between successive training epochs. These results clearly support the argument that distribution processing platforms can have a significant advantage in terms of processing time, leading to a 25% reduction in the epochs needed to achieve stable performance, for both training datasets. Furthermore, we also observe that for larger number of computing nodes, the impact of data distribution becomes less important. A natural question to ask is how training loss behaves in our distributed framework. Figure 4 depicts the number of epochs required to reach a stable performance, using identical criteria as before but in terms of loss differences between epochs instead of accuracy, while Figure 5 presents the values of the achieved by the loss function, both on the 5K dataset.

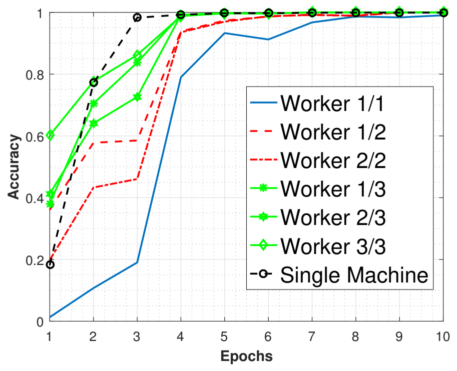
The results of Figure 4 once again demonstrate the positive



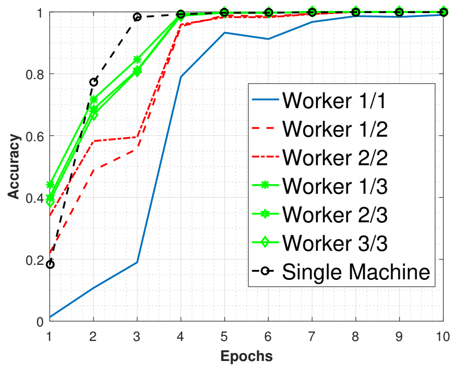
(a) 5K samples set, same data on each worker.



(b) 5K samples set, different data on each worker.



(c) 15K samples set, same data on each worker.



(d) 15K samples set, different data on each worker.

Fig. 2: Training epochs for various cluster sizes.

		Criteria 1 (Patience=1, minimum delta=0.001)		Criteria 2 (Patience=3, minimum delta=0.0001)	
		5K-Same Data	5K-Different Data	5K-Same Data	5K-Different Data
Single Machine		8 epochs	8 epochs	14 epochs	14 epochs
1 Worker		8 epochs	8 epochs	16 epochs	16 epochs
2 Workers	Worker#1	7 epochs	6 epochs	13 epochs	15 epochs
	Worker#2	7 epochs	5 epochs	13 epochs	15 epochs
3 Workers	Worker#1	6 epochs	6 epochs	13 epochs	10 epochs
	Worker#2	6 epochs	6 epochs	13 epochs	10 epochs
	Worker#3	6 epochs	6 epochs	13 epochs	10 epochs

TABLE I: Epochs needed to achieve stable performance for different early stopping parameters.

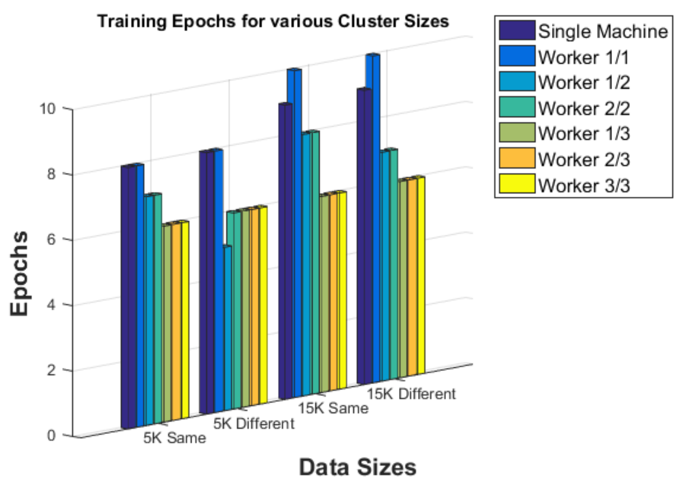
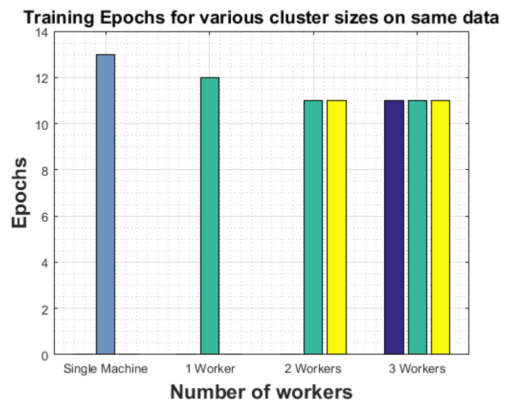


Fig. 3: Number of epochs required to reach a plateau in training accuracy performance.

effect of our platform in terms of distributed processing time, reducing it by 50% when different sets of examples are considered. However, when each worker sees the same data the impact, although visible, is negligible. Meanwhile, Figure 5 suggests that, when the same data is used, the performance between most of the nodes is similar, while when different sets of examples are considered, there is a significant difference in terms of loss in the cases of two- and three-nodes cluster. We also observe that loss behaves similar to accuracy, in that it drops faster on the sequential CNN, however similar performance is attained one epoch later, regardless of its size.

To further evaluate the distributed framework, we applied another set of early stopping parameters, tested on the 5K samples dataset, and present the results along with the initial ones in Table I. We experimented on various cluster sizes, considering both cases of data distribution. Training accuracy was selected as the quantity to be monitored. The set of parameters we decided to experiment with are: (i) patience, which represents the number of epochs before stopping, once accuracy stops improving and (ii) minimum delta, a threshold to quantify whether accuracy has improved or not at the end of each epoch. As a result, our new criteria (right column) are less strict than the old ones (left column), as indicated by the higher patience and the smaller minimum delta.

These changes lead to a higher number of epochs required for training. Similar behavior can be observed in both cases of early stopping, with the amount of epochs decreasing as more nodes are added into the cluster. However, for our new set of parameters, we observe that after a certain number of nodes the amount of epochs required to achieve a stable performance remains the same, when the same set of examples is taken into consideration. On the other hand, this is not true for the case of different sets of data, as in the 3 nodes case the results show a 30% reduction in processing time. However, TFoS does not perform well compared to non-parallelized CNN when same data are considered, since it needs 13 epochs to reach a a stable performance for the 3-node cluster, while the sequential CNN needs 14. When different datasets are considered, the performance of TFoS caps at 10 epochs for the 3-node cluster, much better than the sequential CNN.



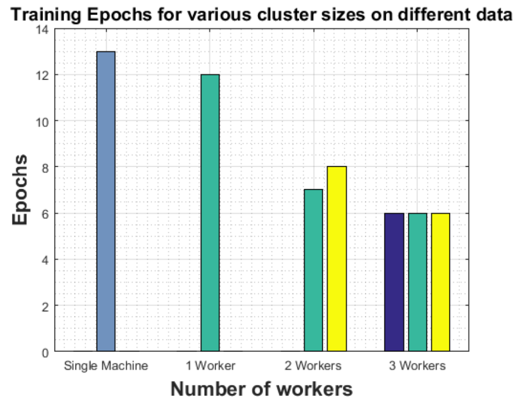
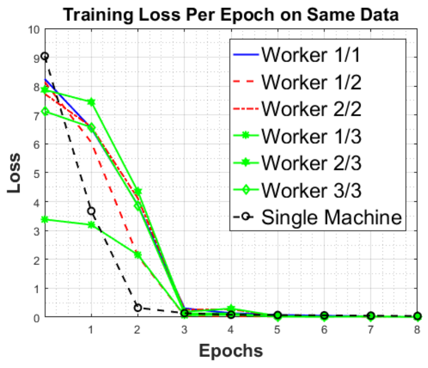


Fig. 4: Epochs required to obtain the minimum loss.



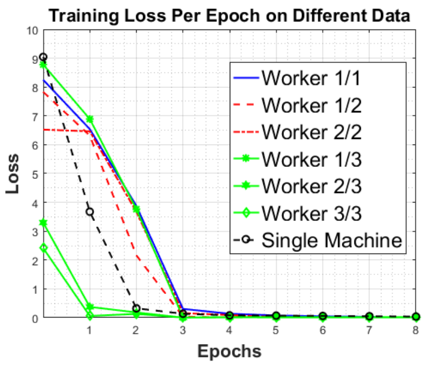


Fig. 5: Loss performance for different cluster sizes.

V. Conclusions

In this work we evaluate the performance of TensorFlowOnSpark, over a 5-node CPU Spark Cluster. Our proposed setup implements data parallelization, in tandem with asynchronous distributed training. We conducted experiments with noisy simulated data, used for redshift estimation, and our results provide the following contributions:

- Distributed CNNs have a similar behavior to traditional setups, in the sense that larger amount of training samples has a positive effect on the system's learning capacity. As a result, it is safe to assume that transporting a DNN to a distributed environment does minimal damage on the network's overall performance.
- Multiple workers have a positive effect on our distributed CNN, since they help our network to converge much faster than the sequential case. However, they also add considerable overhead to the network, due to the higher number of transfers and bottlenecks caused by the PS.
- In terms of accuracy, the impact of data distribution becomes less important the higher the number of training node a cluster has. For fewer number of nodes though, our experiments showed better performance when each node has access to different sets of training data.
- However, when we conduct experiments in terms of loss, data distribution has high impact regardless of cluster size. In this case, for different data per worker the cluster converges faster the more nodes we add, while for same data the differences are minuscule no matter the cluster size.
- Furthermore, we concluded that for distributed training to be worthwhile, the computation benefit of multiple machines, has to outweigh the introduced overheads. This usually happens when training time on a single machine becomes extremely large, either due to network complexity or large amounts of data.

Acknowledgment

This work was funded by the DEDALE project contract no. 665044 within the H2020 Framework Program of the European Commission.

References

[1] Y. LeCun, Y. Bengio, and G. E.Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436444, 2015.
[2] S.-J. Lee, T. Chen, L. Yu, and C.-H. Lai, "Image classification based on the boost convolutional neural network," IEEE Access , 2018.
[3] Q. Zou, L. Ni, T. Zhang, and Qian Wang, "Deep learning based feature selection for remote sensing scene classification.," IEEE Geosci. Remote Sensing Lett., vol. 12, no. 11, pp. 23212325, 2015.
[4] M. Chen, X. He, J. Yang, and H. Zhang,"3-d convolutional recurrent neural networks with attention model for speech emotion recognition," IEEE Signal Process. Lett., vol. 25, no. 10, pp. 14401444, 2018.
[5] P. Graff, F. Feroz, M. P Hobson, and A. Lasenby, "Skynet: an efficient and robust neural network training tool for machine learning in astronomy," Monthly Notices of the Royal Astronomical Society, 2014
[6] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of gpu-based convolutional neural networks," in Parallel, Distributed and Network-Based Processing, 2010 18th Euromicro International Conference on. IEEE
[7] S. Lee, J.K. Kim, X. Zheng, et al., "On model parallelization and scheduling strategies for distributed machine learning," in Advances in neural information processing systems, 2014.
[8] M. Zinkevich, M. Weimer, et al., "Parallelized stochastic gradient descent," in Advances in neural information processing systems, 2010
[9] D. Das, S. Avancha, D. Mudigere, et al.,"Distributed deep learning using synchronous stochastic gradient descent," CoRR, 2016.
[10] Y. Oyama, A. Nomura, I. Sato, H. Nishimura, Y. Tamatsu, and S. Matsuoka, "Predicting statistics of asynchronous sgd parameters for a large-scale distributed deep learning system on gpu supercomputers," in Big Data, 2016 IEEE International Conference on. IEEE, 2016
[11] M. Li, D. G Andersen, J.W. Park, A. J Smola, et al., "Scaling distributed machine learning with the parameter server.," in OSDI, 2014
[12] J. Rhodes, R. C Nichol, E. Aubourg, R. Bean, et al., Scientific synergy between lsst and euclid, The Astrophysical Journal Supplement Series, vol. 233, 2017
[13] R. Stivaktakis, G. Tsagkatakis, B. Moraes, F. Abdalla, Jean-Luc Starck, and P. Tsakalides, "Convolutional neural networks for spectroscopic redshift estimation on euclid data," unpublished.
[14] X. Lu, H. Shi, R. Biswas, et al., "Dlobd: A comprehensive study of deep learning over big data stacks on hpc clusters," IEEE Transactions on Multi-Scale Computing Systems, 2018.
[15] M. Zaharia, R. S Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, et al., "Apache spark: a unified engine for big data processing," Communications of the ACM, vol. 59, no. 11, pp. 5665, 2016.
[16] Z. Zhang, K. Barbary, F.A. Nothaft, E. Sparks, O. Zahn, M. J Franklin, et al., "Scientific computing meets big data technology: An astronomy use case," in Big Data (Big Data), 2015 IEEE International Conference on. IEEE, 2015, pp. 918927.
[17] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, 2008.
[18] C. Jia, J. Liu, X. Jin, H. Lin, H. An, W. Han, et al., "Improving the performance of distributed tensorflow with RDMA," International Journal of Parallel Programming, vol. 46, no. 4, 2018.
[19] W. Zhang, S. Gupta, X. Lian, and J. Liu, "Staleness-aware async-sgd for distributed deep learning," InProceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, 2016.
[20] M. Saouabi and A. Ezzati, "A comparative between hadoop mapreduce and apache spark on HDFS," in Proceedings of the 1st International Conference on Internet of Things and Machine Learning, IML 2017, Liverpool, United Kingdom, October 17-18, 2017.