DS256 SSDS - Project Report

Roopkatha Banerjee
roopkathab@iisc.ac.in
SR.No. 19488
PhD
Department of Computational Data Sciences
IISc, Bangalore

Shreeparna Dey
shreeparnad@iisc.ac.in
SR.No. 19632
MTech
Department of Computational Data Sciences
IISc, Bangalore

I. Introduction

Deep Neural Networks(DNNs) and Deep Learning has been the driving force behind many recent advances in many software technologies. It has applications in image recognition, self driving cars, natural language processing and recently, also recently, in scientific computing. However as data sets grow larger and larger and the models grow more complex, training DNNs on a single machines take longer and longer, to the point that it may take weeks or even months to train a model. Hence, the recent years, many frameworks for distributed training of the DNNs on GPUs have been proposed and implemented due to the massively parallel computing capabilities. However, since deep learning requires a massive amount of data, which cannot be stored on the limited GPU memory, computations on GPUs also have limitations. Hence many frameworks have been proposed and developed for distributed training on both CPUs and GPUs.

The paper by Mayer et al. [9] studies in detail the three most popular techniques for parallelization, which are Data, Model, and Pipeline parallelism. In Data Parallelism, the data is split into chunks and then sent for training to different workers who have their own independent copy of ML/DL models. Although this scales nicely for computation-expensive models with fewer parameters, parameter synchronization needs to be done at the end each iteration of the training, which can be a bottleneck for the system. Next is Model Parallelism in which the DL model is split among different worker nodes, with only the worker node holding the input layer being given the training data. During a forward pass, communication happens between workers in consecutive instead of laterally. Similarly in backward pass, gradients are passed back in the opposite direction. Here we can see that as the model is split, the memory needed by each worker is less, but the tricky part is to split the model in an optimized way. In pipelined parallelism, both the methods are combined.

Now in data parallelism, we have seen the primary problem is with the parameter synchronization. The paper by Mayer et al. [9] have discussed three ways to approach it - centralized, decentralized, and federated. In a centralized structure, we have a set of parameter servers whom workers periodically report back their calculated parameters. In a decentralized structure here, as a parameter server doesn't exist, the workers share their update parameters among themselves through an all-reduce operation, a common substitute to which is the ring-all reduce. In federated learning, the data which is used to train is locally kept on the user's side and alongside a global model exists which is trained on the updates computed locally on the user's side. Another bottleneck is the algorithm applied to optimize the parameters of the model. The most preferred algorithm is Stochastic Gradient Descent(SGD) for training on large data-sets. However, this requires synchronization between processes at each level. There have been many alternative asynchronous and approximate algorithms that have been proposed, like Hogwild method [10], Elastic Averaging SGD [12] etc.

There have been multiple frameworks proposed and developed that manage and, in some cases exploit, the above mentioned semantics to try an achieve the most efficient distributed training in CPUs and GPUs. In this project we have looked at three such lightweight Python frameworks - Horovod, Ray Train (formally RaySGD), distributed PyTorch(which has assimilated Caffe2). We use Horovod and Ray Train with PyTorch to train three deep-learning models and three data-sets.

In section(II) we review some of the works which have done a similar kind of benchmarking for distributed deep learning libraries and go on to explain our methodology in section(III). We give details of our experimental setup and the relevant hardware specifications in section(IV) and the experimental results in section(V).

II. Related Work

Zhang et al (2018) [13] presents a great look at the large scale distributed deep learning systems of the time and categorises them according to the methods that each employ. Further, they discuss the fault tolerance semantics offered by each of them as well. Kurth et al (2018) [8] does a performance analysis of TensorFlow deployed on a Cray XC40 supercomputer with the use of Horovod and goes into great detail of the workflow of the library. They benchmark the Horovod implementation of the MPI library using two models: HEP-CNN and CosmoGAN and conclude that it gives good weak scaling for small and medium sized models.

Jain et al. (2019) [6] presents a through comparison of the performance of TensorFlow and PyTorch with the use of Horovod to perform distributed training on multiple different GPUs. Further, they document the platform dependence of the training time and accuracy as well as the effect of batch sizes on them. They provide an excellent set of Single Node Single Process, Single Node Multi-Process, Multi-Node Multi-Process as well as CPU-GPU comparisons. Finally, they present an extensive profiling of the communication costs that Horovod incurs. [7] does a similar comparison between RaySGD and Horovod. They conclude that on CPUs, with a fixed number of workers working with larger batch sizes, RaySGD outperforms Horovod. They use Horovod to train different models distributedly using Tensorflow and compares the performance of a few asynchronous and approximate alternatives to the serial synchronous stochastic gradient descent algorithm that is used to train the DNNs.

III. Problem statement and Contributions

Through this project, we are trying to understand and compare various distributed deep/machine learning frameworks that have been proposed in literature and are currently in popular use. As such, we have picked two of the most popular libraries in use: Horovod and Ray Train. We have bench-marked the libraries on the hardware specified in the subsection(IV-A) using a combination of three models and three data sets, as described in subsections(IV-B, **??**). One of the models is a custom made DNN which is trained to do object classification on data from the Sloan Digital Sky Survey(SDSS). Since the GPUs employed are of different architectures, we also note the single GPU performance for each model.

Both Ray and Horovod are lightweight libraries that implement multiple optimal communication semantics for PyTorch.

A. Horovod

Horovod originated from Michelangelo's deep learning toolkit, which is Uber's ML-as-a-service platform. It is an open-source framework used for communication by TensorFlow, Keras, PyTorch, and Apache MXNet [1]. It uses MPI(Message Passing Interface) or Gloo for communication among the distributed processes. It uses the ring-all reduce approach, an optimal bandwidth method, in which the worker nodes average the gradients without the need for an explicit parameter server [11]. In the ring-all reduce algorithm, each node communicates with 2 of its peers 2*(N-1) times. They are currently using NCCL's(NVIDIA's collective communication library) highly optimized implementation for ring-all reduce. The library is a popular for distributed training of DNNs.

Horovod provides APIs so that users can leverage Horovod on a Spark as well as Ray cluster. Using the package horovod.spark, one can do data processing, model training, evaluation all in spark [1]. Similarly, using a RayExecutorAPI, a Ray and Horovod integration can be provided. This integration presently supports a Gloo Backend exclusively [1].

B. Ray Train

Ray Train provides a wrapper around PyTorch and TensorFlow modules for training the data parallely [2]. It uses gRPC(Google Remote Procedure Call) for communication among the distributed processes. The library implements a parameter server which employs multi-worker mirrored strategy to accumulate losses and broadcast the gradients calculated [7].

IV. Experimental Setup

We benchmark Horovod and Ray Train using a combination of a pair of very well known models, ResNet50 and MobileNet with two sets of data MNIST and FashionMNIST. The details are given in subsections(IV-B) and subsection(IV-C) for the models and the data respectively. Furthermore, we have made a custom DNN to classify data from SDSS. The hardware is specified in subsection(IV-A).

A. Hardware Specifications

All the programs have been run on three different environments:

- Node 5 of the IOE cluster which houses two Titan RTXs with a total of 48GB GPU memory and two AMD EPYC processors with a total 32 threads and 256GB RAM.
- Node 9 of the IOE cluster which house four V100 with a total of 128GB memory and an Intel Xeon processor with 32 threads and 192GB RAM. These cores have Non-volatile memory and have 2 $\times$ 128GB Intel Optane memory installed.

Each of the nodes on the IOE cluster is running CentOS7, gcc version 9.3 and CUDA version 11.0. The cluster uses PBS job scheduler.

B. Models Trained

1) Resnet50(Residual Neural Net): It is a DNN (Deep Neural Network) which is 50 layers deep where there is 48 Convolution layers and 1 Max-Pooling and 1 AveragePooling layer [3].

2) MobileNetV2: MobileNet-v2 by Google is a convolutional neural network that is 53 layers deep. It is a light-weight feature detector, which is suited to devices with low computational power.

C. Data Sets Used

We have used three sets of data: MNIST, FM-NIST and Data from the Sloan Digital Sky Survey. Each of them is described in the subsections below:

1) MNIST Dataset: It is a image dataset of handwritten digits from 0 to 9 and this dataset can be used as a multiclass-classification problem with 10 class labels. It has 60000 training and 10000 testing data points. The dimension of one image which is in grayscale is 28x28.

2) Fashion MNIST Dataset: It is a image dataset of Zalando's article images and this dataset can be used as a multiclass-classification problem with 10 class labels. It has 60000 training and 10000 testing data points. The dimension of one image which is in grayscale is 28x28.

3) Sloan-Digital-Sky-Survey Dataset: The Sloan-Digital-Sky-Survey(SDSS) is an imaging and spectroscopic red-shift survey of the northern and southern hemispheres. We will primarily use data from SDSS Phase-IV imaging camera, which is an optical survey that takes high-resolution images in spectroscopic bands U(3551Å),G(4686Å),R(6166Å),I(7480Å) and Z(8932Å). With this along with the red-shift data from the survey, we will classify the objects present in the images as a star, galaxy or quasar, the three primary artifacts visible in the night sky at optical wavelengths. We will be using data from Data Release 14 [5] and Data Release 17 [4] for this purpose. The data is retrieved from the SDSS Data Repository called SciServer using the CasJobs query tool. The follwowing is the qurey that we used to download the data.

```
SELECT TOP 50000
  p.objid,
  p.ra,
  p.dec,
  p.u,
  p.g,
  p.r,
  p.i,
  p.z,
  p.petroMag_u,
  p.petroMag_g,
  p.petroMag_r,
  p.petroMag_i,
  p.petroMag_z,
  p.petroRad_u,
  p.petroRad_g,
  p.petroRad_r,
  p.petroRad_i,
  p.petroRad_z,
  s.specobjid,
  s.class,
  s.z as redshift into mydb.dr14
  from PhotoObj AS p
JOIN SpecObj AS s ON s.bestobjid =
p.objid
WHERE
p.u BETWEEN 0 AND 19.6
AND g BETWEEN 0 AND 20
```

The columns we are training for are the optical bands and the corresponding Petrosian Radii and the Petrosian Flux for each of the bands.

V. Experimental Results

We report the runtimes for the above mentioned models and data sets in table(**??**) for Horovod and table(II) for Ray Train. Larger models like VGG16 could not be run since the GPU was out of memory and also more number of GPU cores could not be allocated. The error is shown in shown in figure(1).

In most of the observations we see that single machines' accuracy is comparable, and even better in some cases, to the accuracy for 2 GPU. The Ray Train Framework has shown good speed up in case of increasing GPUs core for data parallel training. In case of Horovod, the observations of training time were quite erratic. We don't see the proper speed up that should have been reflected while increasing GPU cores for training. This is possible because it is specialized to communicate the gradient inter-node while these tests were run on GPUs connected to the node. While training for models like ResNet-50 the Horovod Framework's cache was spilling and was raising a warning.

VI. Conclusions and Discussions

Through our project we explored different Distributed Deep Learning Frameworks which do distributed training on GPU clusters. We also explored various Deep Neural Networks and learnt about them. We faced a lot of challenges through the project and also we hope to explore more in the domain by including distributed learning on data clusters in spark and ray in detail.

VII. Acknowledgement

We would like to expresses our gratitude to Mr. Bhanu TejaBattu, for his indispensible help in setting up libraries and dependencies on the I.O.E Cluster.

References

[1] Horovod Documentation. https://horovod.readthedocs.io/en/stable/summary_include.html.

[2] RaySGD Documentation. https://docs.ray.io/en/latest/raysgd/raysgd.html.

[3] ResNet Model. https://github.com/keras-team/keras-applications/blob/master/keras_applications/resnet50.py.

[4] Abdurro'uf et al. The seventeenth data release of the sloan digital sky surveys: Complete release of manga, mastar and apogee-2 data. 2021.

[5] Bela Abolfathi et al. The fourteenth data release of the sloan digital sky survey: First spectroscopic data from the extended baryon oscillation spectroscopic survey and from the second phase of the apache point observatory galactic evolution experiment. *Astrophysical Journal Supplement Series*, 235:42–42, 2017.

[6] Arpan Jain, Ammar Ahmad Awan, Quentin G. Anthony, Hari Subramoni, and Dhableswar K. D. K. Panda. Performance characterization of dnn training using tensorflow and pytorch on modern clusters. *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–11, 2019.

[7] Shruti Kunde, Amey Pandit, and Rekha Singhal. Benchmarking performance of raysgd and horovod for big data applications. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 2757–2762, 2020.

[8] Thorsten Kurth, Mikhail E. Smorkalov, Peter Mendygral, Srinivas Sridharan, and Amrita Mathuriya. Tensorflow at scale: Performance and productivity analysis of distributed training with horovod, mlsl, and cray pe ml. *Concurrency and Computation: Practice and Experience*, 31, 2019.

[9] Ruben Mayer and Hans-Arno Jacobsen. Scalable deep learning on distributed infrastructures: Challenges, techniques and tools. *arXiv: Distributed, Parallel, and Cluster Computing*, 2019.

[10] Benjamin Recht, Christopher Ré, Stephen J. Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.

[11] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *ArXiv*, abs/1802.05799, 2018.

[12] Sixin Zhang, Anna Choromańska, and Yann LeCun. Deep learning with elastic averaging sgd. In *NIPS*, 2015.

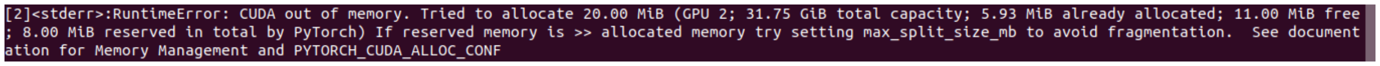
[13] Zhaoning Zhang, Lujia Yin, Yuxing Peng, and Dongsheng Li. A quick survey on large scale distributed deep learning systems. *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 1052–1056, 2018.

			ResNet50+MNIST	MobileNet+FMNIST	DNN+SDSS
Node5	1GPU	Training Time (in secs)	4.132	0.691	0.002
		Testing Accuracy(%)	90.2	72.9	73.8
		Iterations per second	7.56	45.23	171.77
	2GPUs	Training Time (in secs)	3.4474	0.6409	0.00326
		Testing Accuracy(%)	90.4	72	73.9
		Iteration per second	4.53	24	57.05
Node9	1GPU	Training Time (in secs)	3.4474	0.6409	0.00336
		Testing Accuracy(%)	90.3	72.2	74.2
		Iterations per second	9.1	50.84	110.73
	2GPUs	Training Time (in secs)	3.7002	0.936	0.0039
		Testing Accuracy(%)	90.3	71.8	74.1
		Iterations per second	4.23	16.7	47.6

TABLE I: Table of data for experiments with Horovod

			ResNet50+MNIST	MobileNet+FMNIST
Node5	1GPU	Training Time (in secs)	4.0798	0.817
		Testing Accuracy(%)	98.83	69.51
	2GPUs	Training Time (in secs)	2.798	0.654
		Testing Accuracy(%)	99.22	47
Node9	1GPU	Training Time (in secs)	3.353	0.792
		Testing Accuracy(%)	99	58.02
	2GPUs	Training Time (in secs)	1.48	0.808
		Testing Accuracy(%)	99	74.98

TABLE II: Table of data for experiments with Ray Train



```
[2]<stderr>:RuntimeError: CUDA out of memory. Tried to allocate 20.00 MiB (GPU 2; 31.75 GiB total capacity; 5.93 MiB already allocated; 11.00 MiB free
; 8.00 MiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.  See document
ation for Memory Management and PYTORCH_CUDA_ALLOC_CONF
```

Fig. 1: The error that the IOE cluster throws on trying to use larger models or trying to use a larger number of GPUs on Node 9