Distributed Deep Learning Systems – A Comparative Study onDifferent DDL Frameworks

[1]Dr.R.Anusha, [2] Hari Priya N , [3] Mahek R
anuramsri80@gmail.com,haripriyanarahari28@gmail.com,mahek.ramesh@gmail.com
[1]Asst.Prof, Department of Information Technology,
M.O.P. Vaishnav College for Women (Autonomous), Chennai, India
[2,3]Students, M.Sc(Information Technology),
M.O.P. Vaishnav College for Women (Autonomous), Chennai, India

Abstract:
Deep Learning, a subcategory of machine learning has gained immense success because of the recent advancements in distributed deep learning systems. The concept of distributed deep learning scales well with the available computational methods and this in turn helps to increase the overall productivity. Using this methodology, productivity can be fueled by distributing networks and tasks over a number of devices. This methodology is useful when alarge data model cannot be stored on a single machine. In such cases, DDL techniques like data parallelism and model parallelism are proven to be successful for distributing data acrossseveral machines.

In this paper, we will be discussing the different distributed deep learning frameworks available. TensorFlow, PyTorch, Singa and Caffe will be the main frameworks that will be analysed . The paper highlights the current scenario in the distributed deep learning domainand unfolds areas where there can be progress.

Keywords: Caffe, Distributed deep learning, Data Parallelism, Model Parallelism, PyTorch, Singa, TensorFlow

Introduction:

Deep neural networks consist of a large number of stages. Therefore, to analyze the behaviour of this network, an optimized set of parameters should be defined. In the early years, networks consisted of a lower number of layers and such networks are still being used. But when it comes to complex problems, deep neural networks are essential to deal with a large data set and derive accurate results[1].

Accuracy in a deep learning model can be achieved only by defining a large number of training examples and parameters. Training a neural network on a single machine with different modules is expensive and time consuming even if the machine supports multithreading. Although there are machines with modern GPUs, the need for distributionstill prevails.

The solution for this is to make use of different machines with one or more GPUs. To implement distribution, several useful distributed deep learning frameworks have been developed.These frameworks also consist of some pre-trained frameworks that make computation easy. This paper presents four most widely used distributed deep learning frameworks. All these frameworks function based on the major strategies of distributed deep learning: data parallelism and model parallelism. Furthermore, the architecture, pre-trained models and design principles of each of the frameworks have also been highlighted.

Review of literature

Need of Distributed Systems for Deep Learning

Any deep learning problem involves large amounts of variables and data. [1] explains Parallel and Distributed Deep Learning state that the processing can take up to several days and the data cannot be stored in a single workstation. This leads to the need of a distributed architecture to speed up this process. Several methods have been proposed and adopted for various frameworks based on usage, performance and storage.
Two main strategies by which training partition is done:
Data Parallelism and Model Parallelism

Data Parallelism

In this approach parallelism is achieved by partitioning the data into parts equal to the number of nodes in the group. Then each node works on the subset of data allocated. The model training is achieved by parameter averaging and stochastic gradient descent.

Model Parallelism

This approach is used when the model is too big to fit in a single system. Here the model is divided and each GPU is assigned a subset of the parameters available for training. Then all nodes collaborate together to get the final result.

TensorFlow

TensorFlow was created by the Google Brain Team in the year 2015. It is a free, open -source distributed deep learning framework. According to the paper TensorFlow: A System for Large-Scale Machine Learning, the framework represents the computation and state of the algorithm together using a unified graph. This helps users to work with different parallelisation schemes. The paper also states that the platform functions as a distributed system by using many GPU enabled servers for training a model and gets inference by executing it on large distributed clusters.

PyTorch

PyTorch developed by Facebook's AI Research lab (FAIR) is a free open source library based on Torch. The paper PyTorch Distributed: Experiences on Accelerating Data Parallel Training states that it offers tools like DataParallel and DistributedDataParallel for multi-process data parallel training in distributed systems and RPC distributed model parallel training. This framework uses Parameter averaging that scales out training of models by directly computing the average of all the models (PyTorch Distributed: Experiences on Accelerating Data Parallel Training).

SINGA

SINGA is a distributed deep learning framework created by National University of Singapore in 2014. It is an open source project under Apache.SINGA works by using hybrid parallelism. The model to be trained is divided and executed in different commodity servers that are available. It also offers several partitions to divide the model. SINGA has been deployed in several healthcare centres in Singapore.

Caffe

Caffe (Convolutional Architecture for Fast Feature Embedding) was developed by Berkeley AI Research. It was developed primarily for images. Caffe2 was developed by Facebook and it included all features of Caffe. It uses im2colgpu for unrolling the image onto the kernel (Parallel and Distributed Deep Learning). This framework also allows flexibility and allows training on multiple GPUs on distributed systems and supports mobile operating systems (Deep Learning in Mobile and Wireless Networking: A Survey).

Proposed Work:

Objectives:

In this paper we try to bring out the different ways in which frameworks like TensorFlow, PyTorch, Caffe, and SINGA achieve a distributed architecture. We explore the different features, design principles and user interface of the distributed deep learning systems. The paper also brings out the different parallelism techniques employed by these frameworks

Findings:

TenserFlow:

When it comes to Artificial Intelligence and Machine Learning, TensorFlow is a very famous DDL framework provided by Google. With this framework, several deep learning concepts and algorithms can be implemented in an effortless manner. TensorFlow consists of various machine learning libraries and algorithms that are appropriate for training deep neural networks. In addition to this, TenserFlow is also useful for development of sequence models, image recognition, etc.

Architecture:

The working of a TensorFlow architecture is based on the following parts:

- Initializing the data
- Constructing the model
- Training the model and estimating results

To understand the architecture of TensorFlow, the following components are important:

1. **Servables:** Servables are considered as the central units of computation. This component is generally used by clients to execute computations. Servables can be of any interface type. Servables can be easily extensible and therefore futureenhancements can be easily made.
2. **Servable versions:** In TensorFlow, a single servable instance can accommodate several versions of servables. This facilitates clients to load various algorithms with different configurations concurrently. Based on the model under consideration clients can request for any version of servable to be loaded.
3. **Servable streams:** A servable stream is a collection of different versions of a servable. The servables are arranged based on an increasing order of their versionnumbers.
4. **Models:** A model is represented with the help of one or more servables. A model can be either represented as a single servable or multiple servables not dependent on eachother.
5. **Loaders:** The life cycle of a servable is managed by loaders. These are equipped withan API

that helps in loading and unloading a specific servable.

6. **Sources:** Sources are considered to be additional modules in the architecture that are responsible for searching a specific servable. For every servable stream, the source allocates one loader for each version of the servable.
7. **Managers:** The managers are responsible for monitoring the life cycle of the servables. Some of the major tasks include loading, serving and unloading the servables. Managers are also responsible to fulfill requests from the source.
8. **Core:** The core is responsible for managing the lifecycle and metrics of the servables. Loaders and servables are considered as opaque objects by the TenserFlow servingcore.
9. **Batcher:** Batching is a technique that helps to reduce cost. In batching, multiple requests are converted into a single request in the presence of a GPU. Therefore, TenseFlow is equipped with a special batch widget through which clients can batchtheir requests and those requests can also be processed efficiently.

Data Parallelism:

To distribute training models across several machines, TenserFlow makes use of an API called tf.distribute.Strategy. This API is extremely useful for machine learning engineers and researchers as it is flexible and facilitates ease of use.

There are two types of techniques for distributed training with data parallelism:

- Asynchronous training
- Synchronous training

To understand these techniques, the terms workers and accelerators should be defined. Worker is an individual machine consisting of a CPU and many GPUs. Accelerator is a standalone GPU.

In the case of synchronous training, pieces of data are sent to the worker or the accelerator. Each of the devices consist of a local copy of the model and training is done only on a subset of data. A forward pass is initiated in all the devices and several outputs are computed. While the devices are communicating, all the outputs from the workers or accelerators are combined with the help of an all-reduce algorithm. This algorithm is designed to combine all data and parameters from the various workers and accelerators. After aggregation, the gradients are allocated back to the devices. The devices update the new value of the weights and a backward pass is initiated only after all the variables are updated. Therefore, all devices have the same weights at each time although they are trained on different data. This is the reason why this technique is said to be synchronous.

To reduce the idle time of the workers, the asynchronous training approach is appropriate. Asynchronous training is a good choice if there are fewer devices with restricted potential. Therefore, to accomplish asynchronous training, parameter server strategy can be used.

Parameter server strategy is a technique where a group of workers are divided into two types:

- **Parameter servers:** These are responsible to store and update the different variablesof the model

- **Training workers:** These are responsible for running the training process andproducing gradients.

A copy of the model is created and stored in each of the devices. The training worker collects the variables from the parameter servers. The training worker then performs the training and the gradients are sent back to the servers. The servers update the weights accordingly. Therefore, here the training is carried out independently

Model parallelism:
In the case of large models that do not fit into memory, the technique of model parallelism isa helpful approach. The large model is divided into different parts and these parts are distributed across several machines. Each part is trained separately using the same data. This approach requires overhead and is difficult to maintain.

Pre-trained models:
Pre-trained models are helpful as these are already programmed with the required data to solve a problem. Some noteworthy pre-trained models of TensorFlow are as follows:

Pose: This is an API for monitoring and detecting human poses.BlazeFace: This model is

built for face detection.

DeepLab v3: This is built for semantic segmentation.BodyPix: This is useful for body parts segmentation
Coco SSD: This model is meant for detecting several objects in a single image.

Design principles:
On the whole, TensorFlow consists of useful features and functionalities ideal for various machine learning and deep deep learning models. It consists of the following features:

Optimization:
Mathematical expressions can be solved and calculated with multi-dimensional arrays called tensors. This facilitates optimization and ease of use.

Highly Scalable:
This framework consists of various data sets and is ideal for computing almost any kind of expression or data.

High level of Abstraction:
Development time can be reduced to a considerable amount with TensorFlow. Users canfocus more on the logical development of their model.

Good Platform Support:
Another special feature of TensorFlow is that it can be used on various platforms rangingfrom Android, IoS, cloud to various other embedded systems.

SINGA

For training large deep learning models, there is a requirement of a distributed platform thatis highly scalable and flexible. SINGA is a generalized distributed deep learning frameworkthat supports training of large models that consist of a large dataset. It is designed based on the intuitive programming approach which means that it is easy to use and compute results.

Architecture:
SINGA is based on the stochastic gradient descent (SGD) algorithms for training the models. Based on this algorithm, the workload is distributed among several worker and server units.

Logical Architecture:
Being a highly flexible framework, SINGA scales well with both synchronous and asynchronous strategies of training. In addition to these common strategies, hybrid
parallelism is also supported. There are three different kinds of neural net partitioningschemes namely batch partitioning, feature partitioning and hybrid partitioning. Generally, the goal of training a model is to define certain parameters that are appropriate forderiving the desired results in any specified task. Therefore, with the help of the SGD
algorithm, parameters are initialized and updated randomly.

The architecture of the SINGA framework consists of two major components:

1. **Server Group:** This group consists of multiple servers that are responsible for storinga copy of the parameters. In addition to this, the servers are also responsible for handling and updating requests from the worker group.
2. **Worker Group:** The worker group also consists of a local copy of the model and performs training based on a particular dataset. The parameter gradients of the training are also computed. Every worker group can communicate to only one single server group. Inside a worker group, parameter gradients are computed
synchronously. However, the communication between worker groups and server groups is asynchronous.

Data Parallelism:
In data parallelism, the worker group is responsible for computing all the parameters againsta subset of data.

Model Parallelism:
In model parallelism, worker groups are partitioned and parameters are computed based on the partitioned data.

Hybrid Parallelism:
With the help of hybrid parallelism, layers can be partitioned with batch partitioning orfeature partitioning.

Pre-trained models:
SINGA consists of some useful pre-trained models. Some of them are listed below[3].

Sandblaster: Used by Google Brain, this is a synchronous training model where data is divided into multiple nodes.

AllReduce: In this synchronous framework, servers are not involved. Worker groups are responsible for computation.

Downpour: In this asynchronous framework, there are several groups executing simultaneously and each group is not aware of the other group's existence.

Hogwild: In this asynchronous framework, each node maintains a copy of the parameters and updates them locally.

Design Principles:
SINGA is one of the most convenient distributed deep learning frameworks that is perfectly suitable for training complex models. Some of the design features of SINGA are listed below

Scalability: A major goal of SINGA is to reduce the time that is consumed for training complex models.

Easy to use: As SINGA is designed based on the intuitive programming approach, all tasks can be completed with ease without any hassle.

Py Torch

Architecture:
The front end of PyTorch is Python. This makes it simple to use and work with.
PyTorch is mainly written in C++. C++ is known to be fastest in terms of performance and speed. From the system's view, C++ is closer to the system in levels of abstraction. This helps to achieve high performance.
The DataParallel class is the main core class that enables the data to be executed on several GPUs. The DistributedDataParallel API makes sure to communicate the process and system number where the data is being trained. This information needs to be communicated as the framework is a distributed environment. Later when the data is modelled the parameters are averaged by calling the AllReduce. This is also done by the DistributedDataParallel. It also guarantees return of the parameter values back to the exact system.

Data Parallelism:
All data in PyTorch is stored as tensors. A tensor is defined as the object that represents the vector relationships of data. This makes working with matrices and vectors easier.
PyTorch allows distributing data across several threads and several systems[5]. Distribution of data across several systems is achieved by partitioning the data into several parts across several GPUs[9]. GPUs enable high speed processing of large amounts of data thus making calculations faster. This is done by the DataParallel class. Synchronisation of data is achieved by Parameter averaging. AllReduce operation is used for parameter averaging. The main requirement to call AllReduce is that the data must be in equal sizes in the form of tensors.
Many communication libraries are used for achieving synchronisation.

Pre-trained models:
Pre-trained models are models in which data has been curated and already processed to find results. These are useful and efficient as they save time and computation.

Pytorch offers a range of pre-trained models. Some of them are:

3D ResNet - Used for video ClassificationNTSnet – Used for birds Classification

PyTorch-Transformers –Used for NLP implementation EfficientNet – Used for Image classification

GoogLeNet – Used for Convolutional Neural Network

Model parallelism:
Model Parallelism comes into picture when we have to train a model that is too big to be trained in a single CPU. In such cases the model is usually split into several GPUs. PyTorch achieves this by using the forward() function[3]. The forward() when invoked divides the layersof the model into several GPUs. Then the results are copied back from the GPUs after processing.

Design principles
Everything related to python spells simplicity, ease of use, functional and performance oriented. PyTorch has also been designed on similar grounds.

Simplicity:
PyTorch uses Python which is known for it's easy, English-like syntax. Thus this enables users to concentrate on their task rather than worry and spend time learning how to go about using the framework[7].

Performance oriented:
Trade-off between performance and simplicity is always difficult. If the framework gets too simple the performance degrades and vice-versa. PyTorch strikes the balance by prioritising performance, thus it agrees to slight increase in simplicity to achieve high performance.

Ease of use:
PyTorch abstracts the implementation of various computations to help users carry out their implementation with ease. This also enables not so good programmers to achieve their goals using the framework.

Caffe

Architecture:
Caffe's front end is Python. Thus users can import the Caffe model in Python and use it to train data. The core is primarily written in C. This enables high performance and easy integration between various systems. Caffe provides a CPU and GPU mode. The GPU mode is used for training the model, the CPU mode is used to deploy and get results for the trained model. Switching is usually done using a flag bit by the system.

As Caffe was primarily made for images, it has high speed performance compared to other architectures.

Caffe is highly modular. All components are implemented as different modules.

Data Parallelism:
Data in Caffe is represented as blobs. A blob is a four-dimensional array. Blob holds parameters and images that have to be modelled or trained. Thus Caffe abstracts memory into a single unit. All computations take blob as the input and return a blob output.

Caffe, like any distributed deep learning architecture, achieves data parallelism with the use of GPUs. The data is unwrapped using the im2colgpu onto different GPUs. The results of the GPUs are then passed to the CPU that handles the different experiments on the models and produces results[8].

Pre-trained Models:
Pre-trained models save time and computation. Some of the pre trained models in Caffe are:

OpenPose: Used to identify people

DeepYeast: Images on yeast cellsDQRN: Q-Net using Caffe ParseNet: Used for Parsing

Design Principles:
Modularity:
One of the key features of Caffe has been its modularity. Modularity helps users implement the features they want and it also makes understanding the framework easier. This also makes updates and upgrades easier.

Abstraction:
Caffe was designed such that functionality of components is separated from its implementation. This helps users to easily use the tools required for various purposes and not worry about its implementation or background design. The functioning on CPU and GPU is also taken care by the framework[8]

Python:
To use Caffe, the model can be imported to the Python shell. Python's easy and simple to use syntax enables Caffe to be user-friendly. Thus users who are not so good in coding can also easily use the framework to achieve desired results. Dual mode**:**

One of the striking features of Caffe is its Dual mode feature. Dual mode enables Caffe to switch between CPU and GPU to perform computations and train the data without the needof special hardware.

Conclusion:

Distributed deep learning is a vast area of machine learning that defines various strategies and techniques for training deep learning models. In this paper, four most noteworthy distributed deep learning frameworks namely TensorFlow, SINGA, Pytorch and Caffe have been discussed. The different aspects and functionalities of each of the frameworks have also been highlighted[8]. All the above-mentioned frameworks are highly scalable and easy to use.

Generally, distributed systems are known to bring in major complexities. But with the help of these frameworks, training of deep learning models both simple and complex can be achieved with ease.

Future Work:

We have analysed and compared how different frameworks have achieved a distributed architecture for training deep learning models. Most of them employ CPUs and GPUs to train and test models. As different frameworks have different approaches, a future work might be to combine the best features of these frameworks and to increase the performance capability for very large datasets. Finding ways to achieve more goals of a distributed system in these frameworks willalso be a interesting work for the future.

References

[1] Parallel and Distributed Deep Learning
https://web.stanford.edu/~rezab/classes/cme323/S16/projects_reports/hedge_usmani.pdf
[2]TensorFlow: A System for Large-Scale Machine Learning
https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf
[3] https://arxiv.org/pdf/2006.15704.pdf
SINGA: A Distributed Deep Learning Platform
https://dl.acm.org/doi/pdf/10.1145/2733373.2807410
[4] SINGA: Putting Deep Learning in the Hands of Multimedia Users
https://www.comp.nus.edu.sg/~ooibc/singa-mm15.pdf
[5] Deep Learning in Mobile and Wireless Networking: A Survey
https://arxiv.org/pdf/1803.04311.pdf
[6] https://www.quinbay.com/blog/distributed-training-in-deep-learning-models-2[
[7] PyTorch: An Imperative Style, High-Performance Deep Learning Library
https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf
[8]Comparative Study Of Caffe, Neon, Theano, And Torch For DeepLearning
https://openreview.net/pdf?id=q7kEN7WoXU8LEkD3t7BQ

[9]Distributed Deep Learning Training: Model And Data Parallelism InTensorflow
https://theaisummer.com/distributed-training/
[10]Distributed Training
https://svn.apache.org/repos//infra/websites/production/singa/content/v0.3.0/distributed-training.html