# Assignment - 2
# CS 744 - PyTorch
# Course: Big Data Systems

## Group 10: Team Members

Divy Patel (dspatel6@wisc.edu)
Smit Shah (spshah25@wisc.edu)
Sujay Chandra Shekara Sharma (schandrashe5@wisc.edu)
Venkata Abhijeeth Balabhadruni (balabhadruni@wisc.edu)

## Contributions

Divy:

1. Sync gradient across nodes using all-reduce Gloo Back-end
2. Setup Pytorch in distributed mode using distributed module

Smit:

1. Sync gradient across nodes using gather and scatter Gloo Back-end
2. Use Pytorch's distributed in-built training module for part 3

Sujay:

1. PyTorch forward pass, backward pass computation VGG-11 for CIFAR-10 dataset
2. Experimenting with multiple nodes for data parallel training

Abhijeeth:

1. PyTorch loss computation, optimizer step for VGG-11 for CIFAR-10 dataset
2. Experiment with multiple nodes using distributed in-built training module

Common work done by all the members:

1. Brainstorming on the problem
2. Setting up Pytorch, repository, and VMs

## Environment and Setup Information

- For all the parts, we used the seed value of 744
- Batch size for part 1 is 256, and batch size for remaining parts is 64, such that the total batch size is 256.
- For all parts, We are printing loss after every 20 iterations and running the model for a total of 196 iterations.
- The average iteration time used to create graphs are for 40 iterations (excluding the 1st iteration)
- For the scalability part, we varied the batch size according to the number of nodes, such that the total batch size is 300.

## Part - 1: Training VGG-11 on CIFAR-10

In this part, we implemented the standard training loop consisting of forward and backward passes, loss computation, and optimizer step. We also printed the loss after every 20 iterations. The loss values and the resulting test accuracy are illustrated in the below screenshot. We observe a final average loss of 2.1737 and a test accuracy of 17%.

```
Files already downloaded and verified
Files already downloaded and verified
Iteration Number:  0 , loss:  2.451446533203125
Iteration Number:  20 , loss:  2.870509624481201
Average Iteration time:  1.461957076923077
Iteration Number:  40 , loss:  2.3516321182250977
Iteration Number:  60 , loss:  2.2551450729370117
Iteration Number:  80 , loss:  2.3916430473327637
Iteration Number:  100 , loss:  2.2172696590423584
Iteration Number:  120 , loss:  2.240690231323242
Iteration Number:  140 , loss:  2.197807788848877
Iteration Number:  160 , loss:  2.1832666397094727
Iteration Number:  180 , loss:  2.1817855834960938
Test set: Average loss: 2.1737, Accuracy: 1696/10000 (17%)
```

## Part - 2a: Sync gradient with gather and scatter call using Gloo backend

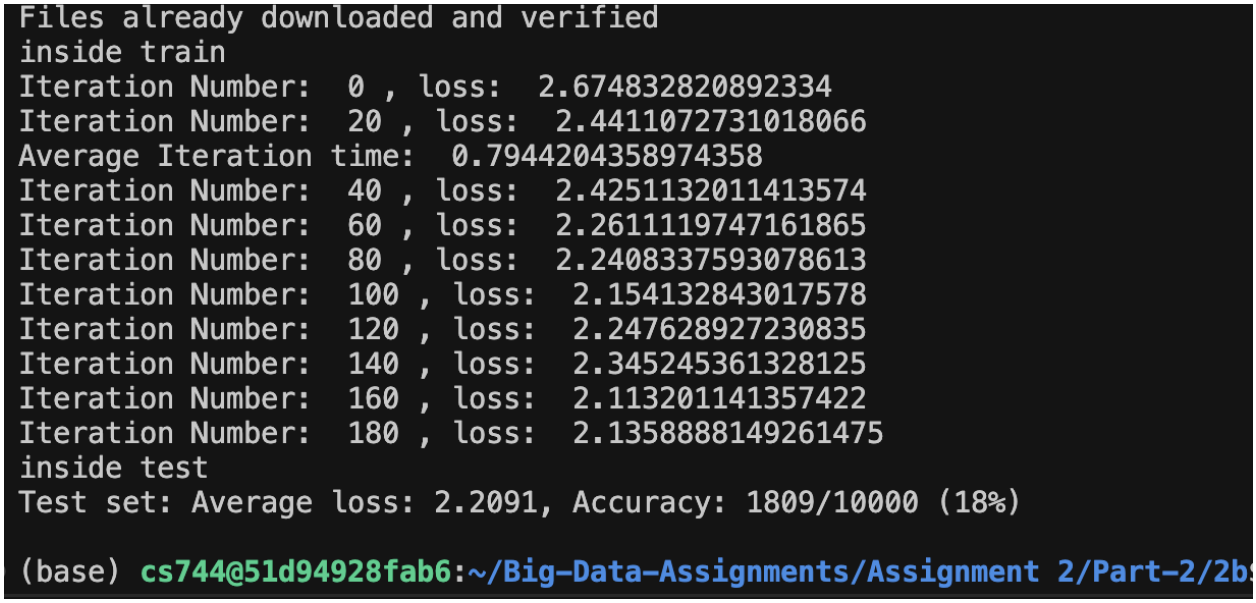
In this part, we modified the implementation to train the model on 4 different nodes in a data parallel manner. In order to communicate between the nodes, we used the Scatter and Gather primitive with Gloo backend. We also printed the loss after every 20 iterations. The loss values and the resulting test accuracy are illustrated in the below screenshot. We observe a final average loss of 2.6755 and a test accuracy of 19%.

```
Files already downloaded and verified
Files already downloaded and verified
inside train
Iteration Number:  0 , loss:  2.674832820892334
Iteration Number:  20 , loss:  2.4063479900360107
Average Iteration time:  0.916540794871795
Iteration Number:  40 , loss:  2.475839376449585
Iteration Number:  60 , loss:  2.26484751701355
Iteration Number:  80 , loss:  2.2465951442718506
Iteration Number:  100 , loss:  2.1489877700805664
Iteration Number:  120 , loss:  2.248079538345337
Iteration Number:  140 , loss:  2.354888916015625
Iteration Number:  160 , loss:  2.142300605773926
Iteration Number:  180 , loss:  2.096667528152466
inside test
Test set: Average loss: 2.6755, Accuracy: 1863/10000 (19%)

(base) cs744@51d94928fab6:~/Big-Data-Assignments/Assignment 2/Part-2/2a$
```

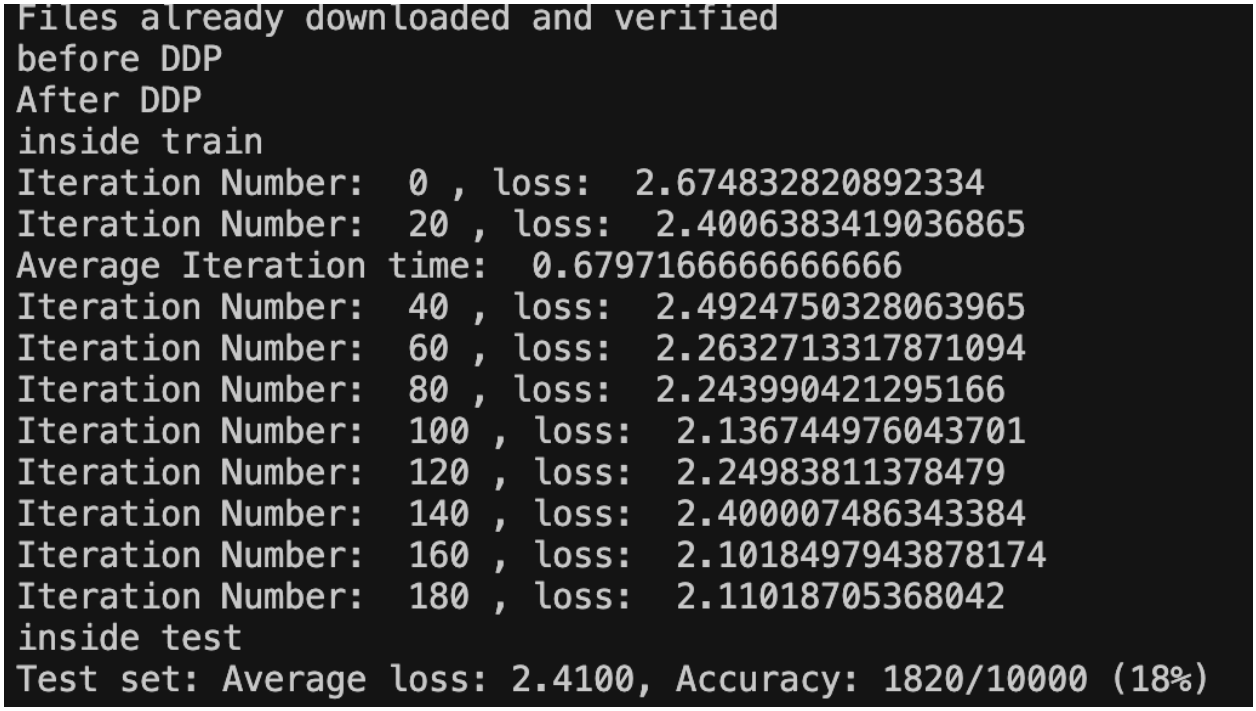
## Part - 2b: Sync gradient with allreduce using Gloo backend

In this part, we modified the implementation to train the model on 4 different nodes in a data parallel manner. In order to communicate between the nodes, we used the All Reduce primitive with Gloo backend. We also printed the loss after every 20 iterations. The loss values and the resulting test accuracy are illustrated in the below screenshot. We observe a final average loss of 2.2091 and a test accuracy of 18%.



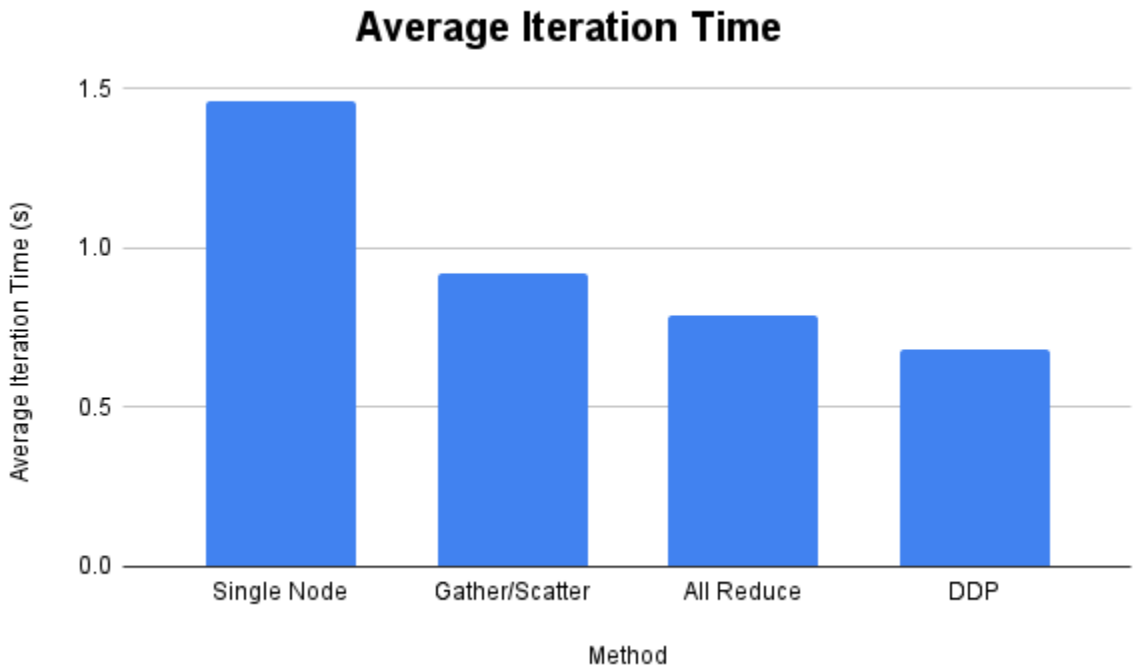
## Part - 3 : Distributed Data Parallel Training using Built in Module

In this part, we modified the implementation to train the model on 4 different nodes in a data parallel manner. We used the in-built DDP module in PyTorch. We also printed the loss after every 20 iterations. The loss values and the resulting test accuracy are illustrated in the below screenshot. We observe a final average loss of 2.41 and a test accuracy of 18%.



## Experiment Results, Observations and Reasoning:

### Evaluation of Average Training Time for Various Setups




### Observation

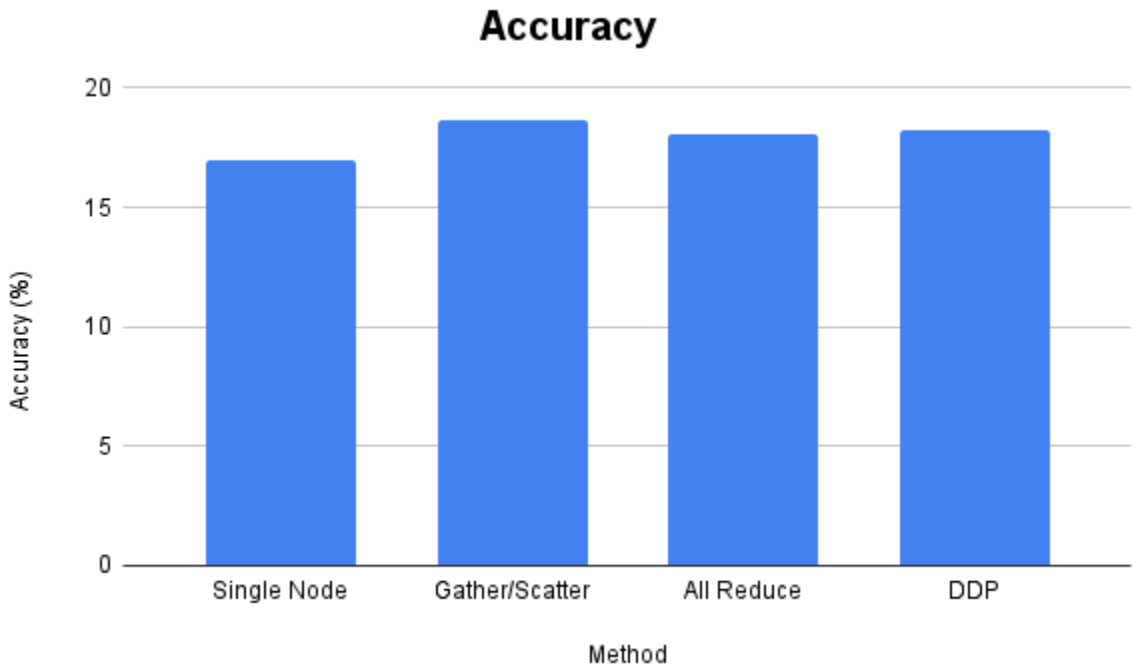
- When training VGG11 on a single node, the completion of an iteration during training takes **1.46 seconds**
- When training VGG11 in a distributed setting on 4 separate isolated containers with the Scatter and Gather primitives of the Gloo Collective Communication Library(CCL) backend of PyTorch we saw an average iteration time of **0.91 seconds,** which is a **37.67%** improvement over the single-node setting
- When training VGG11 in a distributed setting on 4 separate isolated containers with the AllReduce primitive of the Gloo Collective Communication Library(CCL) backend of PyTorch we saw an average iteration time of **0.79 seconds,** which is a **13.18%** improvement over the Scatter/Gather CCL primitive used for Distributed training
- When training VGG11 in a distributed setting on 4 separate isolated containers with the Distributed Data-Parallel training API of PyTorch which uses AllReduce primitive bundled with optimisations like Gradient Bucketing we saw an average

iteration time of **0.68 seconds,** which is a **13.92%** improvement over the vanilla AllReduce CCL primitive used for Distributed training

## Reasoning

- Single node setup being the one taking the most time to train seemed intuitive as training a Deep Neural Network like VGG11 is a compute-bound workload and parallelising intra-iteration computations will do better in most cases for large datasets like CIFAR-10
- With Scatter and Gather primitives being used to synchronize the gradients performed better than the single-node setup even with its inefficient network pattern among nodes. This shows that the workload is much more compute-bound than network-bound
- With AllReduce primitive being used to synchronize the gradients performed better than the Scatter and Gather version of Distributed training because of its lower cross-node communication requirements and hence incurring lesser network overhead
- With DDP API provided by PyTorch, we not only leverage the communication efficiency of AllReduce primitive but also utilize optimizations like Gradient Bucketing which trades off some memory by bucketing gradients of the group of operators and communicating at once in a batch, amortizing the cost of memory did usage

## Evaluation of Accuracy for various Setups



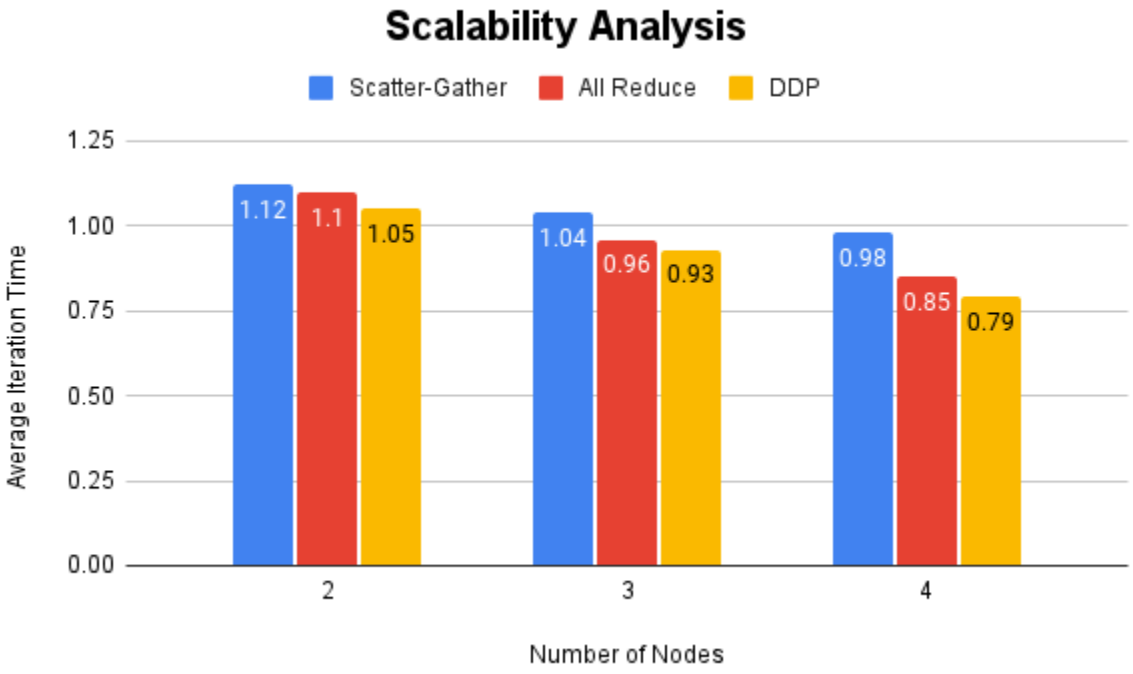

## Observation

- Accuracy on the test data remains similar for all the setups with respect to the traditional single-node training setups

## Reasoning

- Similar accuracies of the training setups are attributed to the statistical equivalence of the Distributed Data-Parallel training method with respect to the traditional single-node training
- The communication primitives do not change how the gradients are being synchronized. Hence, the parameters, more or less, remain the same after each iteration across all the setups

## Evaluation of Scalability of Various Setups with Increasing Number of Nodes in Cluster




### Observation

- When training VGG11 in a distributed setting on setups of 2, 3, and 4 containers, we saw a decrease in average training time as the number of nodes increased for all the different data parallel setups.

### Reasoning

- With all the setups, even though there is an increase in the cross node communication as we increase the number of nodes, overall we see a decrease in the average iteration time as we are able to parallelize the task better with more nodes.