BigData Assignment 2
Report

Input Dataset
CIFAR 10 dataset exposed by tensorflow. The dataset contains 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

Project Goal
In all the tasks below, we tried to study the effect of Distributed DataParallel training on a DeepLearning Model. In particular, we used a VGG11 model with 8 Convolution layers, 5 MaxPool layers and 1 Fully Connect Linear Output Layer.

Hardware Used
A 4 node cluster with each node having 5 cpu with 1 core was used. The bandwidth available between the nodes was around 7 Gbits/sec.

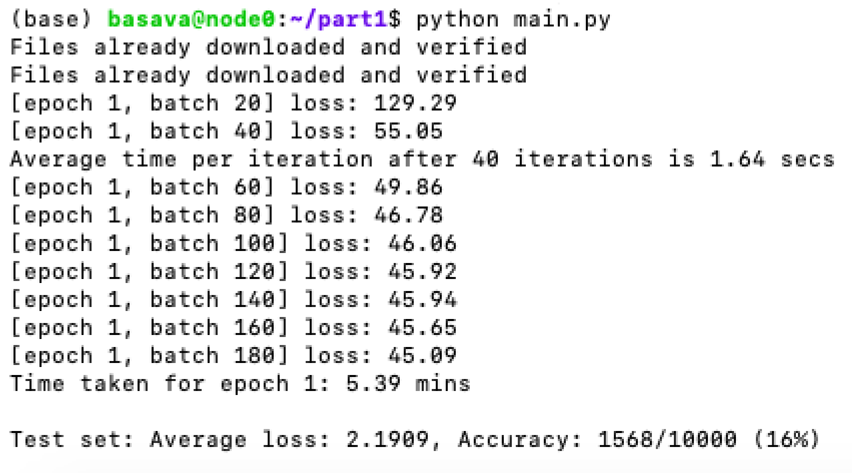
TASK 1

In this task, we check the performance of the model in local training mode. We trained the model on a single node and for 1 epoch.

Parameters used:

- Batchsize: 256
- Random Seed: 2317

Observations:



```
(base) basava@node0:~/part1$ python main.py
Files already downloaded and verified
Files already downloaded and verified
[epoch 1, batch 20] loss: 129.29
[epoch 1, batch 40] loss: 55.05
Average time per iteration after 40 iterations is 1.64 secs
[epoch 1, batch 60] loss: 49.86
[epoch 1, batch 80] loss: 46.78
[epoch 1, batch 100] loss: 46.06
[epoch 1, batch 120] loss: 45.92
[epoch 1, batch 140] loss: 45.94
[epoch 1, batch 160] loss: 45.65
[epoch 1, batch 180] loss: 45.09
Time taken for epoch 1: 5.39 mins

Test set: Average loss: 2.1909, Accuracy: 1568/10000 (16%)
```

Fig 1.1 Local training mode

- Accuracy: 16%
- Average Test Loss: 2.19
- Average time per iteration (for the first 40 iterations excluding the first one): 1.64 secs
- Time taken for 1 full epoch: 5.39 mins

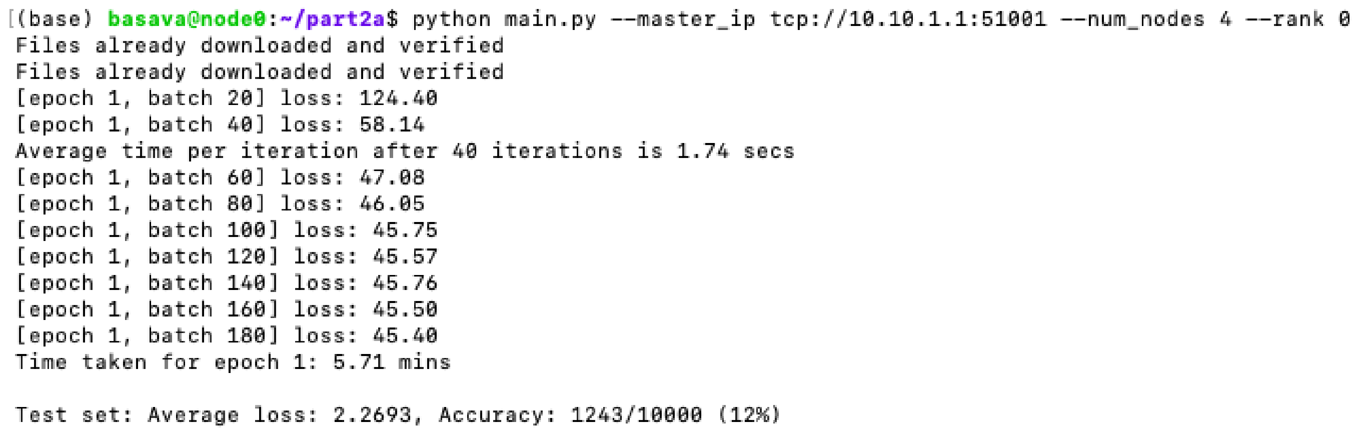
TASK 2a

In this task, we tried to analyze the effect of splitting the data across 4 nodes and conduct DataParallel training. After each minibatch is processed, the rank 0 node gathers gradients (using torch *gather* call) from all the nodes and averages the values. The Mean gradient tensor is scattered back to the nodes using torch *scatter* call. Post this, *optimizer.step()* is called which updates the weights based on the mean gradient available in the model parameters.

Parameters used:

- Batchsize on each node: 64
- Random Seed: 2317

Observations:

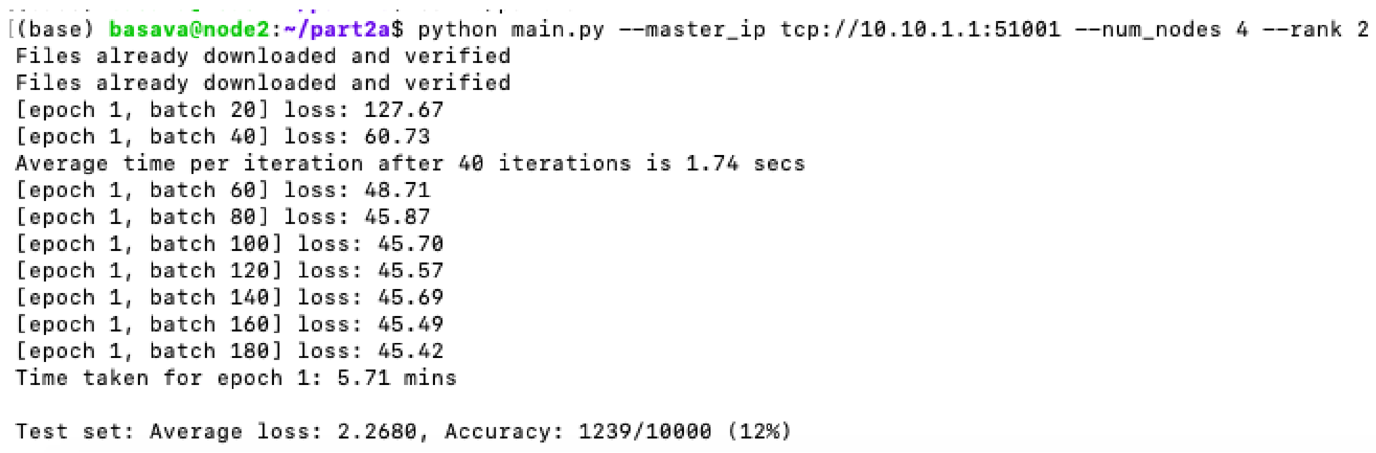


```
[(base) basava@node0:~/part2a$ python main.py --master_ip tcp://10.10.1.1:51001 --num_nodes 4 --rank 0
Files already downloaded and verified
Files already downloaded and verified
[epoch 1, batch 20] loss: 124.40
[epoch 1, batch 40] loss: 58.14
Average time per iteration after 40 iterations is 1.74 secs
[epoch 1, batch 60] loss: 47.08
[epoch 1, batch 80] loss: 46.05
[epoch 1, batch 100] loss: 45.75
[epoch 1, batch 120] loss: 45.57
[epoch 1, batch 140] loss: 45.76
[epoch 1, batch 160] loss: 45.50
[epoch 1, batch 180] loss: 45.40
Time taken for epoch 1: 5.71 mins

Test set: Average loss: 2.2693, Accuracy: 1243/10000 (12%)
```

Fig 2.a.1 Node 1 statistics

- The average training running loss for a mini-batch remained almost the same in comparison to the local mode. This behavior is seen across iterations.
- The training losses across all 4 nodes display similar trends.



```
[(base) basava@node2:~/part2a$ python main.py --master_ip tcp://10.10.1.1:51001 --num_nodes 4 --rank 2
Files already downloaded and verified
Files already downloaded and verified
[epoch 1, batch 20] loss: 127.67
[epoch 1, batch 40] loss: 60.73
Average time per iteration after 40 iterations is 1.74 secs
[epoch 1, batch 60] loss: 48.71
[epoch 1, batch 80] loss: 45.87
[epoch 1, batch 100] loss: 45.70
[epoch 1, batch 120] loss: 45.57
[epoch 1, batch 140] loss: 45.69
[epoch 1, batch 160] loss: 45.49
[epoch 1, batch 180] loss: 45.42
Time taken for epoch 1: 5.71 mins

Test set: Average loss: 2.2680, Accuracy: 1239/10000 (12%)
```

Fig 2.a.2 Node 3 statistics

- Test accuracy took a minor hit, but we believe that the difference will get negligible over epochs.
- The average time per iteration (for the first 40 iterations excluding the first one) increased slightly as compared to the local mode. We believe that this is due to inefficient usage of gather and scatter calls.
- As a result total time taken for 1 epoch increased slightly to 5.71 mins in comparison to 5.39 mins elapsed in local training mode. We believe that there is no significant gain in total time here because the dataset is small. There would have been significant reduction in total time had the dataset been huge.
- The available Network throughput is not being used efficiently in this case as the rank 0th node is doing all the processing of gathering and scattering the vectors. This can be seen from *Fig 4.1* and *Fig 4.2* (at the end of the report).

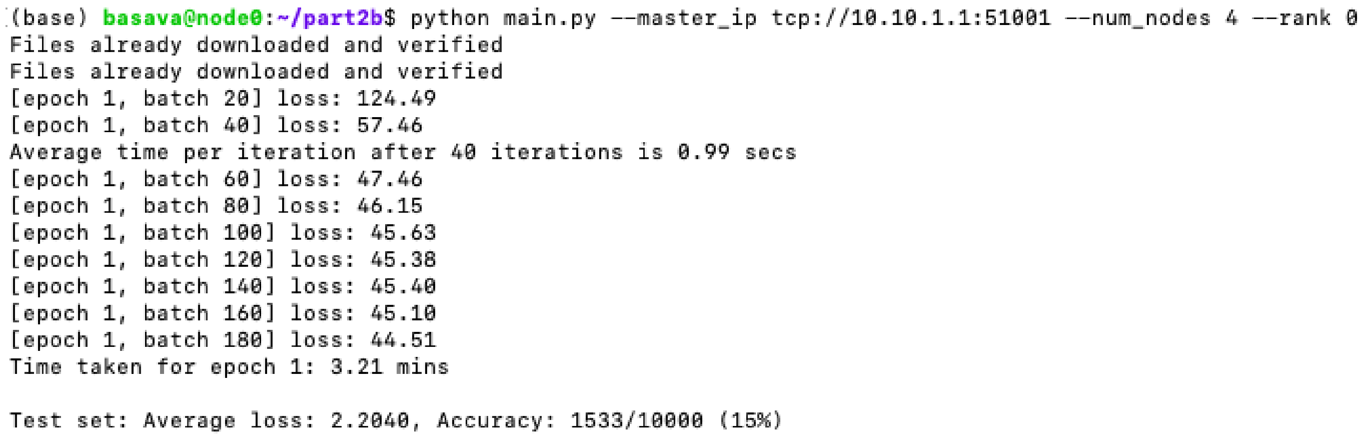
TASK 2b

In this task, we tried to improvise upon 2a and better utilize the network. After each minibatch is processed, the nodes call the torch *all_reduce* function to accumulate the gradients across all the nodes. Post this, each node divides the tensor by the number of nodes to obtain the mean gradient. Finally, *optimizer.step()* is called which updates the weights based on the mean gradient available in the model parameters.

Parameters used:

- Batchsize on each node: 64
- Random Seed: 2317

Observations:

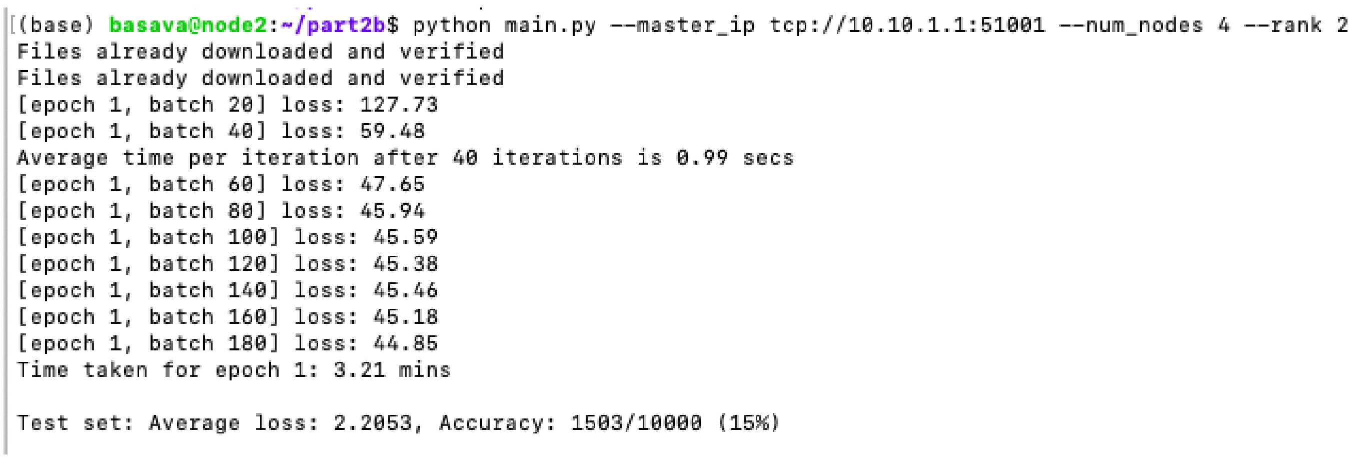


```
(base) basava@node0:~/part2b$ python main.py --master_ip tcp://10.10.1.1:51001 --num_nodes 4 --rank 0
Files already downloaded and verified
Files already downloaded and verified
[epoch 1, batch 20] loss: 124.49
[epoch 1, batch 40] loss: 57.46
Average time per iteration after 40 iterations is 0.99 secs
[epoch 1, batch 60] loss: 47.46
[epoch 1, batch 80] loss: 46.15
[epoch 1, batch 100] loss: 45.63
[epoch 1, batch 120] loss: 45.38
[epoch 1, batch 140] loss: 45.40
[epoch 1, batch 160] loss: 45.10
[epoch 1, batch 180] loss: 44.51
Time taken for epoch 1: 3.21 mins

Test set: Average loss: 2.2040, Accuracy: 1533/10000 (15%)
```

Fig 2.b.1 Node 1 statistics

- The average time per iteration (for the first 40 iterations excluding the first one) is significantly less as compared to both previous parts i.e *local mode* and *gather-scatter* collective communication mode. This improvement can be attributed to the fact that we are using *ring-reduce* algo based *all_reduce* calls in pytorch where any node just needs to interact with neighboring nodes, which inturn reduces the communication overhead. The efficient distribution of network load can be seen in the figure *Fig 4.2* (at the end of the report)
- The total accuracy is almost similar to the local mode, which proves the mathematical equivalence of Distributed DataParallel Training, discussed in the Pytorch paper. We also see some minor variability in accuracy across different setups, which we believe will reduce with the increasing number of epochs.



```
(base) basava@node2:~/part2b$ python main.py --master_ip tcp://10.10.1.1:51001 --num_nodes 4 --rank 2
Files already downloaded and verified
Files already downloaded and verified
[epoch 1, batch 20] loss: 127.73
[epoch 1, batch 40] loss: 59.48
Average time per iteration after 40 iterations is 0.99 secs
[epoch 1, batch 60] loss: 47.65
[epoch 1, batch 80] loss: 45.94
[epoch 1, batch 100] loss: 45.59
[epoch 1, batch 120] loss: 45.38
[epoch 1, batch 140] loss: 45.46
[epoch 1, batch 160] loss: 45.18
[epoch 1, batch 180] loss: 44.85
Time taken for epoch 1: 3.21 mins

Test set: Average loss: 2.2053, Accuracy: 1503/10000 (15%)
```

Fig 2.b.2 Node 3 statistics

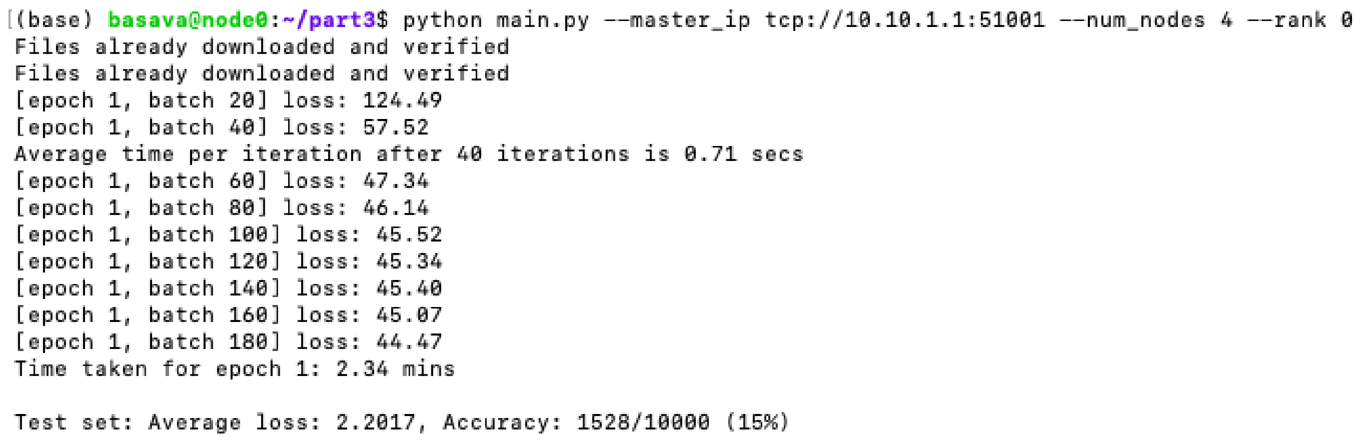
- The total time for 1 epoch is around 3.21 mins which is less in comparison to both *local mode* and *gather-scatter* mode.
- The above two points prove the purpose of data parallel training where we not only take less time to train, but also achieve scalability by distributing data across 4 nodes.

TASK 3

In this task, we used Pytorch's inbuilt DistributedDataParallel module for training the model. This setup splits the input across the specified devices. The model is replicated on each machine, and each replica handles a portion of the input. Post the *backward* pass and before the *optimizer.step()*, gradients from each node are averaged.

Parameters used:

- Batchsize on each node: 64
- Random Seed: 2317



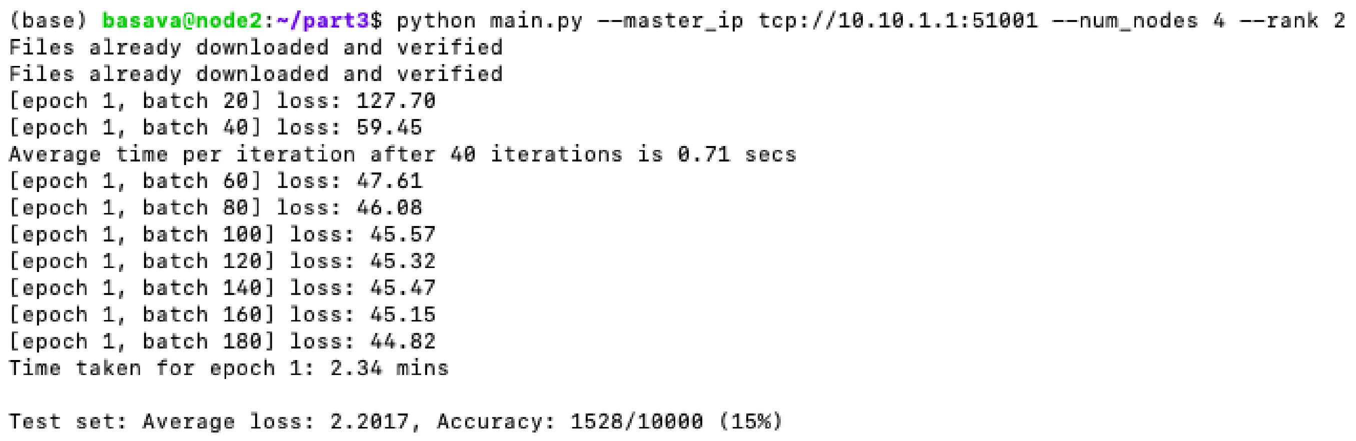
```
[(base) basava@node0:~/part3$ python main.py --master_ip tcp://10.10.1.1:51001 --num_nodes 4 --rank 0
Files already downloaded and verified
Files already downloaded and verified
[epoch 1, batch 20] loss: 124.49
[epoch 1, batch 40] loss: 57.52
Average time per iteration after 40 iterations is 0.71 secs
[epoch 1, batch 60] loss: 47.34
[epoch 1, batch 80] loss: 46.14
[epoch 1, batch 100] loss: 45.52
[epoch 1, batch 120] loss: 45.34
[epoch 1, batch 140] loss: 45.40
[epoch 1, batch 160] loss: 45.07
[epoch 1, batch 180] loss: 44.47
Time taken for epoch 1: 2.34 mins

Test set: Average loss: 2.2017, Accuracy: 1528/10000 (15%)
```

Fig 3.1 Node 1 Statistics

Observations:

- The average time per iteration (for the first 40 iterations excluding the first one) is 0.71 sec, which is lowest among all the previous setups.
- The total time for 1 epoch is around 2.34 mins which is also the lowest among all the previous setups.
- The above two observations can be attributed to the use of gradient bucketing by PyTorch. It waits for the gradients of a bucket and then performs the reduction step. We expect this setup to take the least amount of time compared to the above setups because it overlaps communication and computation phases in an optimal way.
- The training losses across all 4 nodes display similar trends. Same goes for test accuracy as well.



```
(base) basava@node2:~/part3$ python main.py --master_ip tcp://10.10.1.1:51001 --num_nodes 4 --rank 2
Files already downloaded and verified
Files already downloaded and verified
[epoch 1, batch 20] loss: 127.70
[epoch 1, batch 40] loss: 59.45
Average time per iteration after 40 iterations is 0.71 secs
[epoch 1, batch 60] loss: 47.61
[epoch 1, batch 80] loss: 46.08
[epoch 1, batch 100] loss: 45.57
[epoch 1, batch 120] loss: 45.32
[epoch 1, batch 140] loss: 45.47
[epoch 1, batch 160] loss: 45.15
[epoch 1, batch 180] loss: 44.82
Time taken for epoch 1: 2.34 mins

Test set: Average loss: 2.2017, Accuracy: 1528/10000 (15%)
```

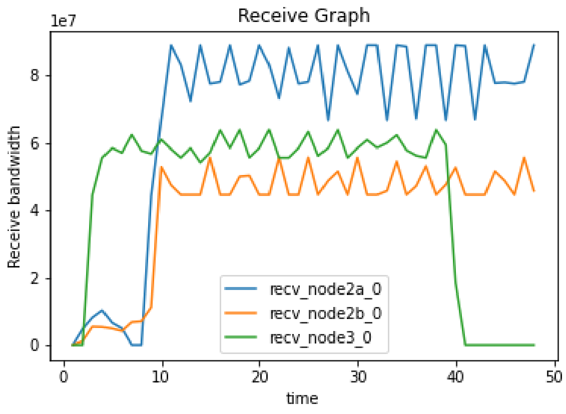
Fig 3.2 Node 3 Statistics

- Test accuracy is around 15 percent, which is approximately similar to the local training mode. It is expected because we are using Pytorch in-built DDP module which guarantees mathematical equivalence.

Overall, we also observed variance in accuracy and loss when we used different values for the *seed*. This is because the data is sampled differently by the DistributedSampler and the loss should eventually converge if we run it for more epochs. Also the initial weights of the VGG11 model are dependent on the value of the *seed.*

Network trends over all the tasks

Observations:



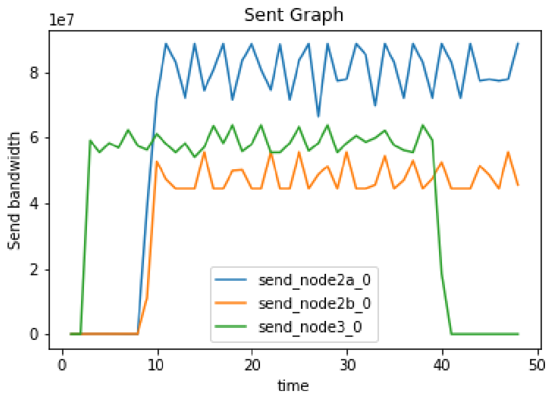
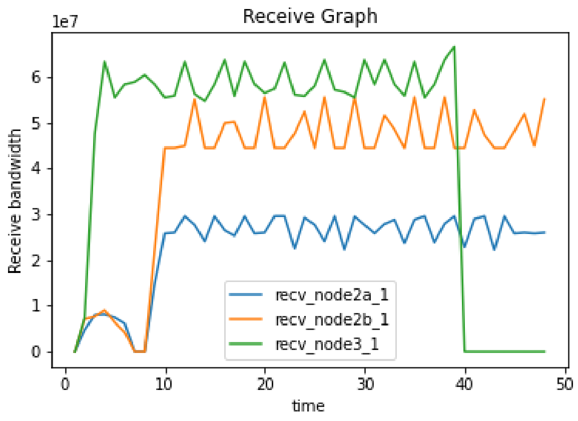


Fig 4.1 Node 0 network stats across all tasks

- In task 2a, network is not being utilised efficiently as both gather and scatter operations are executed with destination/source as node 0. This is not the case in tasks 2b and 3, where we see the network throughput across all nodes is somewhat similar.



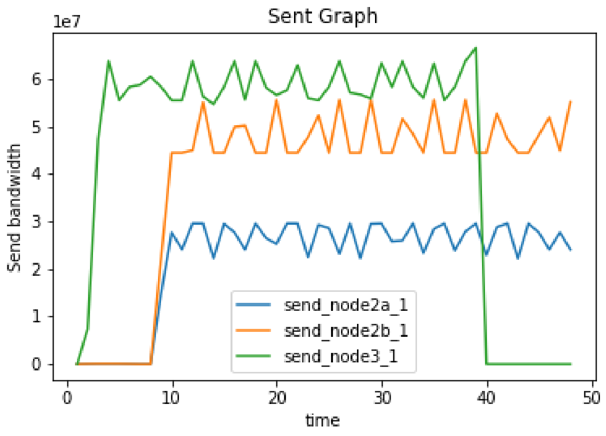


Fig 4.2 Node 1 network stats across all tasks

- The network utilization across the remaining nodes is highest in task 3. Task 2b reaches similar levels of network throughput, but doesn't reach the peak level (we believe this is due to not having gradient bucketing in 2b). We see that the network utilization is lowest among other nodes in task 2a.

- One more important observation that we made was, the exact accuracy in 2a and 2b differ (minutely though) over the nodes. But it isn't the case in task 3, where we use Pytorch's DDP module. On further investigation, we found that the difference is due to the mean and variance stored in the BatchNorm Layer of the local model in each node, which is dependent on the data seen by that node. However in the case of Pytorch, we got to know that the module also syncs the mean and variance for BatchNorm Layer across nodes. It does this by broadcasting the values available in Node 0 local model's BatchNorm Layer before each forward iteration. This synchronization is supported in DDP through Model Buffers as mentioned in the research paper.

Contributions

All three of us did most of the tasks independently because we wanted to get hands-on experience with Pytorch and we also wanted to make sure we were on the right track by comparing our results.

- Pankaj Kumar - Task 1, 2b, 3, Report, Plots
- Salman Munaf - Task 1, 2b, 3, Report
- Basava Kolagani - Task 1, 2a, 2b, 3, Report