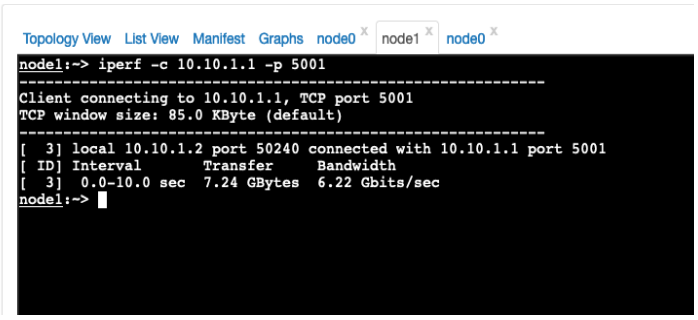
BIG DATA SYSTEMS
Assignment -2

Introduction:

In this assignment we have implemented different approaches of distributed data parallelization on VGG11 model (Deep learning neural network model) using pytorch and compared them.

The data set we used is the CIFAR 10 dataset which consists of 60000 32x32 color images with 10 output labels. In that we are using 50000 for training and remaining 10000 for testing.

We are using a 4 node cluster to do the distributed data parallel and bandwidth between each of these nodes is 6.22 Gbits/sec.



Model Description:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 32, 32]           1,792
              ReLU-2          [-1, 64, 32, 32]               0
         MaxPool2d-3          [-1, 64, 16, 16]               0
            Conv2d-4         [-1, 128, 16, 16]          73,856
              ReLU-5         [-1, 128, 16, 16]               0
         MaxPool2d-6           [-1, 128, 8, 8]               0
            Conv2d-7           [-1, 256, 8, 8]         295,168
              ReLU-8           [-1, 256, 8, 8]               0
            Conv2d-9           [-1, 256, 8, 8]         590,080
             ReLU-10           [-1, 256, 8, 8]               0
        MaxPool2d-11           [-1, 256, 4, 4]               0
           Conv2d-12           [-1, 512, 4, 4]       1,180,160
             ReLU-13           [-1, 512, 4, 4]               0
           Conv2d-14           [-1, 512, 4, 4]       2,359,808
             ReLU-15           [-1, 512, 4, 4]               0
        MaxPool2d-16           [-1, 512, 2, 2]               0
           Conv2d-17           [-1, 512, 2, 2]       2,359,808
             ReLU-18           [-1, 512, 2, 2]               0
           Conv2d-19           [-1, 512, 2, 2]       2,359,808
             ReLU-20           [-1, 512, 2, 2]               0
        MaxPool2d-21           [-1, 512, 1, 1]               0
           Linear-22                 [-1, 10]           5,130
================================================================
Total params: 9,225,610
Trainable params: 9,225,610
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 2.55
Params size (MB): 35.19
Estimated Total Size (MB): 37.76
----------------------------------------------------------------
```

We can find more details of the model used from the above diagram. This model uses around 9M parameters with a total estimated model size 38MB. This VGG11 model has a total of 22 layers spread across Convolution2D, ReLU and MaxPool2d. We removed the batch norm layer from the model to observe the same accuracy across different nodes.

Part-1

In this part of the assignment, We trained the model using a single node for 40 iterations and checked its performance.

Parameters used:

- Batch size : 256
- Random seed : 69143

Observations :

```
Loss at 20th batch is 2.305149793624878
Loss at 40th batch is 2.3005731105804443
Total execution time is : 93.44245481491089 seconds
Average execution time is  : 2.39596037986951 seconds
Test set: Average loss: 2.3031, Accuracy: 1000/10000 (10%)
```

- Accuracy is 10%.
- Average Test loss is 2.3031
- The total time for executing 39 iterations is 93.44 seconds.
- Average execution time for 39 iterations is 2.39 seconds
- Everything is handled by a single worker node so the average execution time is solely based on the computation latency.

Part-2 &3

In this task, we have trained the model using data parallel technique i.e by distributing the data across 4 nodes and using different kinds of collective communication techniques to sync the gradients.

Implementation:

- We are initialising the weights using seed so that all nodes have the same initial weights and we are also using a distributed data sampler which helps in sending that data to the 4 nodes.

- Once we send the data we are computing the gradients of the model in each node after one iteration. Then we sync the gradients using different kinds of collective communication technique like gather-scatter or ring reduce.
- After this, we are using optimizer.step() to update the model parameters (weights) based on the new aggregated gradients available after the gradient synchronization.

Sanity Checks:

- We have tested that the model parameters (weights) changed by changing the seed of the model.
- The accuracy of the model comes out to be similar while using different seeds (*Note: The seed value is same across all the nodes in a given run).*
- When we use different seeds across nodes in the same run, the accuracy of the model comes out to be different as we start with different initial model parameters.

Parameters Used:
- Batch size on each node : 64
- Random seed : 69143

Part-2a

In this task, we have trained the model using data parallel technique i.e by distributing the data across 4 nodes and using gather and scatter techniques to sync the gradients. In this technique, first we are gathering all the gradients in node 0 and computing the sum of them and then scattering the new gradients from node 0 to all the nodes.

Results & Observations:

```
Loss at 20th batch is 2.3083415031433105
Loss at 40th batch is 2.3153371810913086
Total execution time is : 47.23314929008484 seconds
Average execution time is  : 1.2111063920534575 seconds
Test set: Average loss: 2.2933, Accuracy: 1304/10000 (13%)
```

- The total time for executing 39 iterations is 47.23 seconds and the average execution time is 1.21 seconds.
- The training loss for 4 nodes is also almost the same.
- Initially there is a small fluctuation in accuracy across all the nodes which is very small (<1% difference) but this happened because of the hidden parameters in the batch norm which couldn't be synchronized. So we

removed that layer from the model and after that we observed that accuracy is the same in all four nodes.

- The average training loss for the mini batch is almost the same with the single node training method (Task1) and this behaviour is observed to be the same for all iterations.
- When compared with Task1 the total execution time and avg execution per iteration has decreased a lot because we are using data parallel technique and updating the model parameter.
- The average test loss and accuracy are almost similar with the Task1.

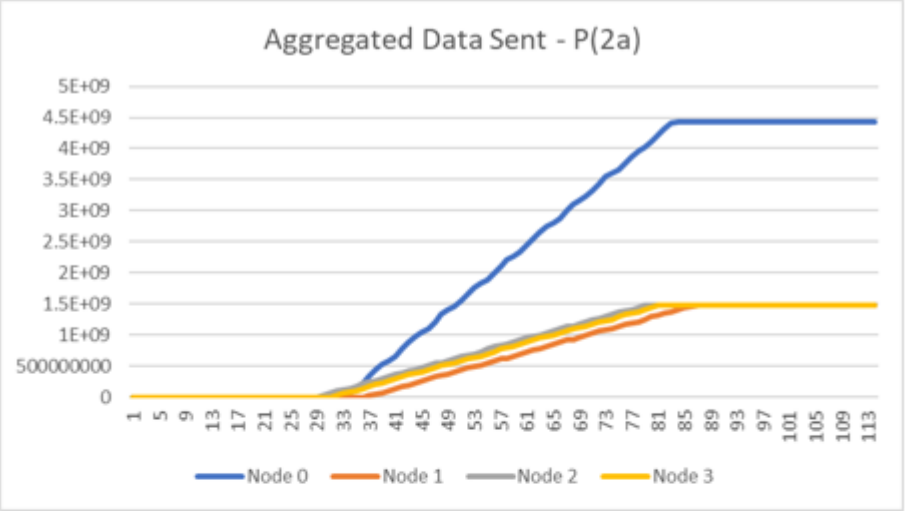


Figure 1: Aggregated Data Sent (in bits) Vs time (in sec) across nodes

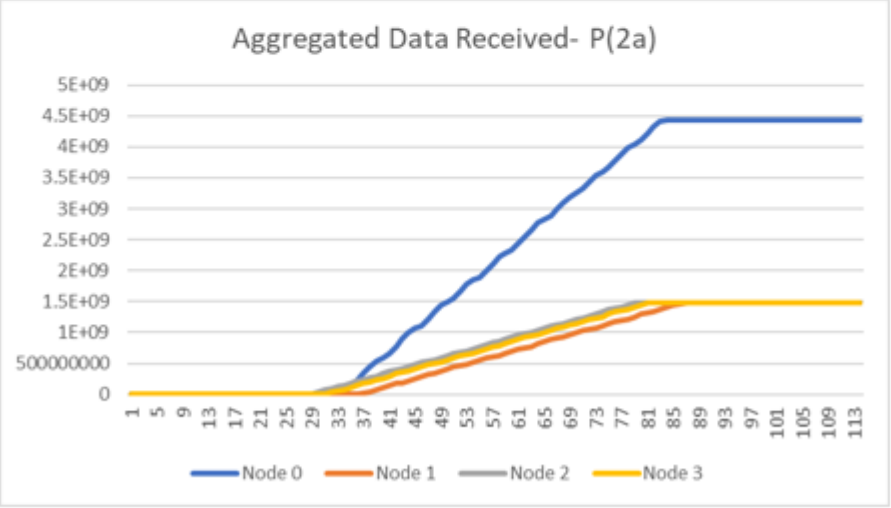


Figure 2: Aggregated Data Received (in bits) Vs time (in sec) across nodes

- We can see from network bandwidth send/receive (Fig:1 &2) that node 0 has aggregated data thrice the size of other nodes. This is expected as in gather/scatter communication all the data is sent/received from node 0 to other nodes.

Part 2b

In this part, we have trained the model using data parallel technique i.e by distributing the data across 4 nodes and using all reduce technique to sync the gradients.

Results & Observations

```
Loss at 20th batch is 2.3052637577056885
Loss at 40th batch is 2.2833316326141357
Total execution time is : 36.44679880142212 seconds
Average execution time is  : 0.9345333026005671 seconds
Test set: Average loss: 2.2928, Accuracy: 1301/10000 (13%)
```

- The total time for executing 39 iterations is 36.44 seconds. As expected this is less than part 2a because we use ring reduce collective communication for which the latency is independent of the number of nodes whereas other collective communication kinds like gather/scatter (2a) is proportional to the number of nodes.
- The total execution time and average loss is similar across all nodes. Since we tested it on the same data across all nodes using the same model and the accuracy is exactly the same.
- Accuracy of this part is similar to part 1 of the model which justifies the mathematical equivalence of the model.
- Same as 2a, here also we removed the batch norm layer from the model and after that we observed that accuracy is the same in all four nodes.
- We can see from network bandwidth (Fig:3 &4) that all nodes have similar data sent/received. This is expected as in ring reduce communication every chunk travels all around the ring and accumulates a chunk in each node. Also, we can observe that the network overhead for each node is scaled by 4 (9/4 =2.25Gbits). This is less compared to the overhead of node 0 in part 2a, hence the communication time for 2b is less than 2a.

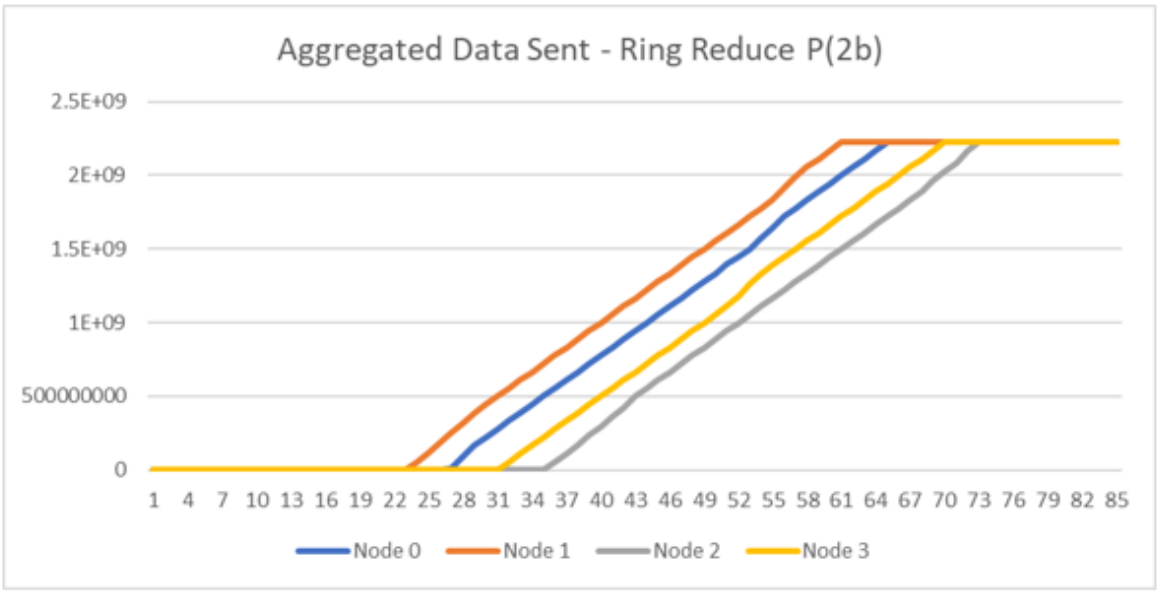


Figure 3: Aggregated Data Sent (in bits) Vs time (in sec) across nodes

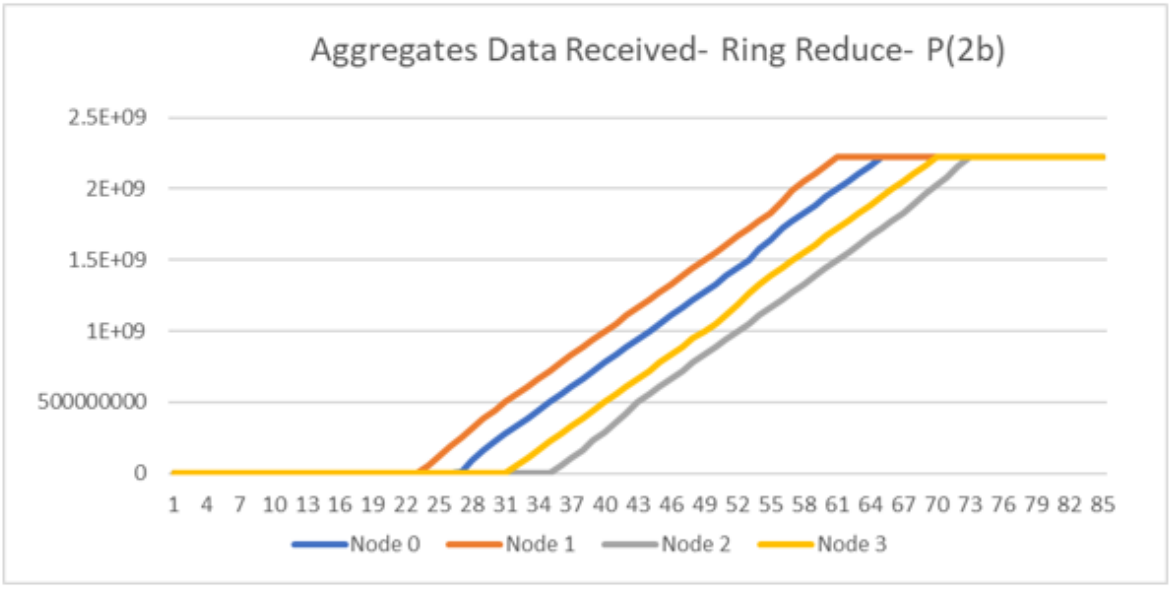
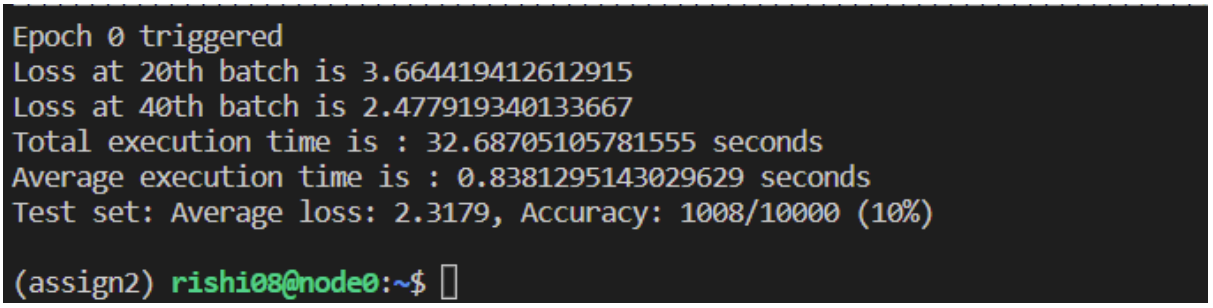


Figure 4: Aggregated Data Received (in bits) Vs time (in sec) across nodes

Part 3

In this part, instead of implementing our own collective communication method, we used the distributed functionality provided by PyTorch which internally synchronizes gradients by computing the average across different nodes using ring based all reduce operation. Gradient bucket of 25MB has been utilized to increase the overlap between computation and communication.

Results & Observations



Observations:

- The total time for executing 39 iterations is 32.68 seconds. As expected this is less than other parts because of the following reasons:
    a. Gradient bucket of 25MB has been utilized to increase the overlap between computation and communication. This is the reason for less communication time in part 3 compared to part 2b.
    b. We use ring reduce for which the compute time is independent of the number of nodes whereas other reduce kinds like gather/scatter (2a) is proportional to the number of nodes.

- The total execution time and average loss is similar across all nodes. Since we tested it on the same data across all nodes using the same model and the accuracy is exactly the same.
- Accuracy of this part is similar to part 1 of the model which justifies the mathematical equivalence of the model.
- We use ring reduce for which the compute time is independent of the number of nodes whereas other reduce kinds like gather/scatter (2a) is proportional to the number of nodes.

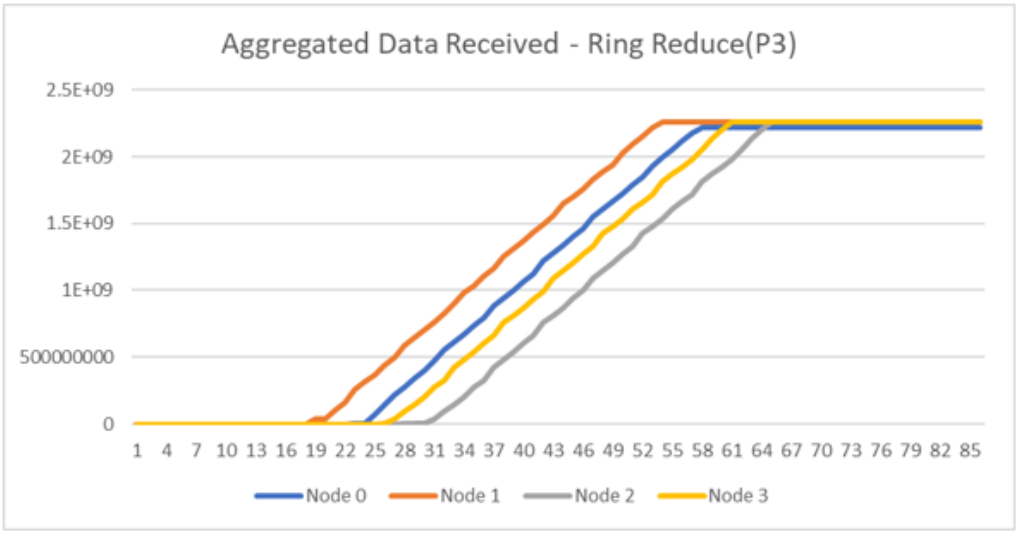


Figure 5: Aggregated Data Sent (in bits) Vs time (in sec) across nodes

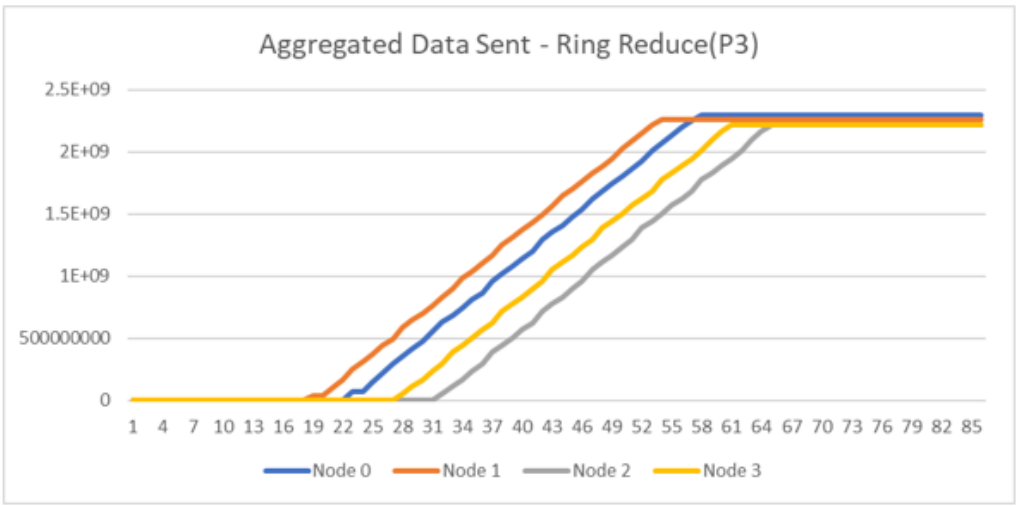


Figure 6: Aggregated Data Received (in bits) Vs time (in sec) across nodes

CPU Utilization

From the below plots (Fig:7-9) of CPU utilization for different reduce methods, we observe the following:

- **CPU utilization in Part 2a is less than 2b** since 2b uses **ring reduce collective communication** which reduces the overall communications latencies and increase the average CPU utilization(as the total execution time is comparatively very less). In gather and scatter all the nodes need to wait till node 0 does the computation and also the network overhead is high as discussed before.
- **CPU utilizationin Part 2b is less than 3** because in part 3 we use **Gradient bucketing** which starts reducing method once the parameters in a given

bucket are available unlike 2b where all the parameters in an iteration have to be computed before synchronisation.

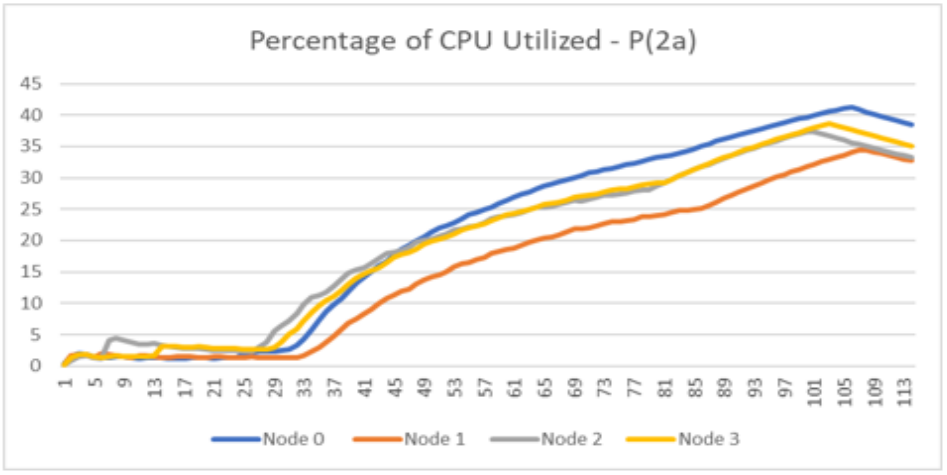


Figure 7: % of CPU utilized Vs time (in sec) across nodes for 2a

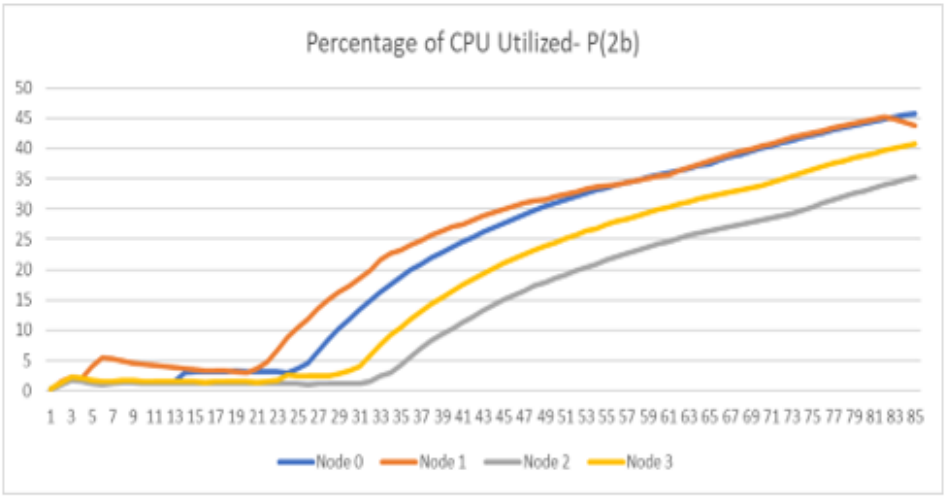


Figure 8: % of CPU utilized Vs time (in sec) across nodes for 2b

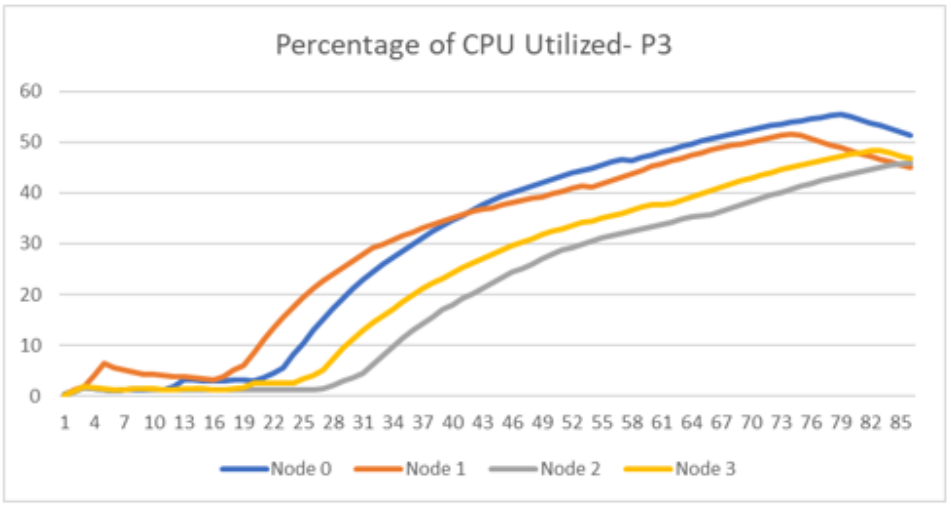
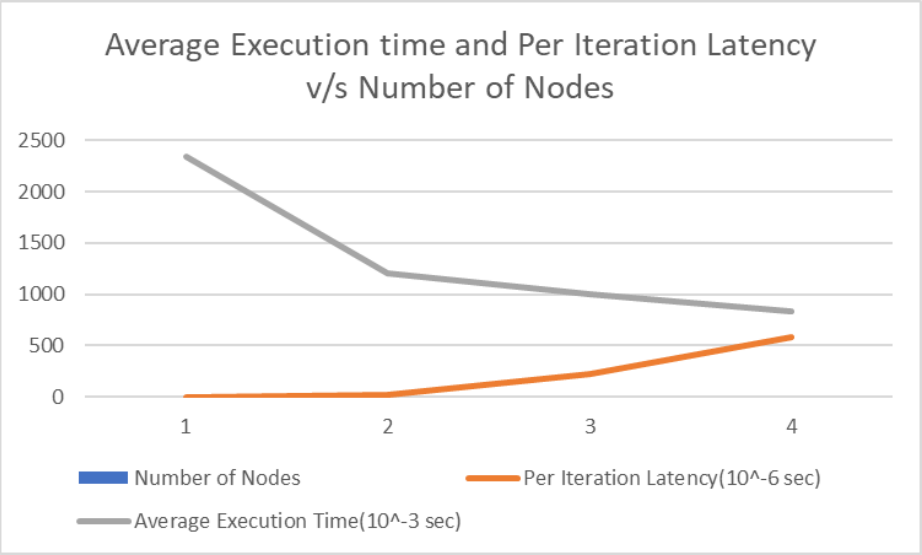


Figure 9: % of CPU utilized Vs time (in sec) across nodes for 3

Scalability

In order to test the scalability of the model, we have run the model across different number of CPUs and then compared the average execution and latency time per iteration. Here latency time refers to the communication time. We computed the communication time by subtracting the computation time when only one node was used. Following are some observations based on below figure:

- Execution time reduces with the number of CPUs as we are getting the benefit of data parallel training.
- Communication time increases with number of CPUs as the number of ring reduce operations are increased.
- Given 4x computational power, the scale up factor effectively turns out to be around 2.8x because of communication latencies.



Contributions

All three of us have worked on the assignment independently. We used to have regular discussions on how to implement and once we are done with our implementations we used to compare our results and comment on them.

- Rishideep Reddy Rallabandi - Task 1, 2a, 2b, 3, Report.
- Girish Jonnavithula - Task 1, 2a, 2b, 3, Report, Plots.
- Akhil Perumal Reddy -Task 1, 2a, 2b, 3, Report.