CSE 5194.01 Autumn 2020
Lab 2 Report

TF_Horovod: Chen-Chun Chen, Wei Liu, Curtis Isaacson, Lang Xu

1. (20 points) Investigate one or two options for running distributed (multi-node) training for your assigned Deep Learning framework and explain your understanding by highlighting major components/technologies needed for it.

There are a few potential options for running multi-node Tensorflow Horovod, though they can be slightly modified based on if we are training on CPU or GPU. For both GPU and CPU we will use API's that run on top of Message Passing Interface protocols. We will be utilizing the Tensorflow and Horovod interfaces to set up the training environment and will be using NCCL and MVAPICH2 to optimize performance under the hood. Additionally, we will take advantage of OSC Owens Parallel File System to access our data we need for training. We will be training ResNet 50, InceptionV3, and VGG16 on the imdb-wiki dataset

First, we will look at using the Nvidia Collective Communications Library (NCCL, pronounced "Nickel") that we utilize when working on GPU training. NCCL allows us to organize and optimize inter-GPU communication when running multi-training. However, since it is created for inter-GPU communication, we can only use this when running our training on GPU's

Secondly we will look at MVAPICH2 which will assist us under the hood when running on OSC. MVAPICH2 is software that is based on the MPI standard that "delivers the best performance, scalability and fault tolerance for high-end computing systems and servers using InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE networking technologies." This will assist us when running multi-node training.

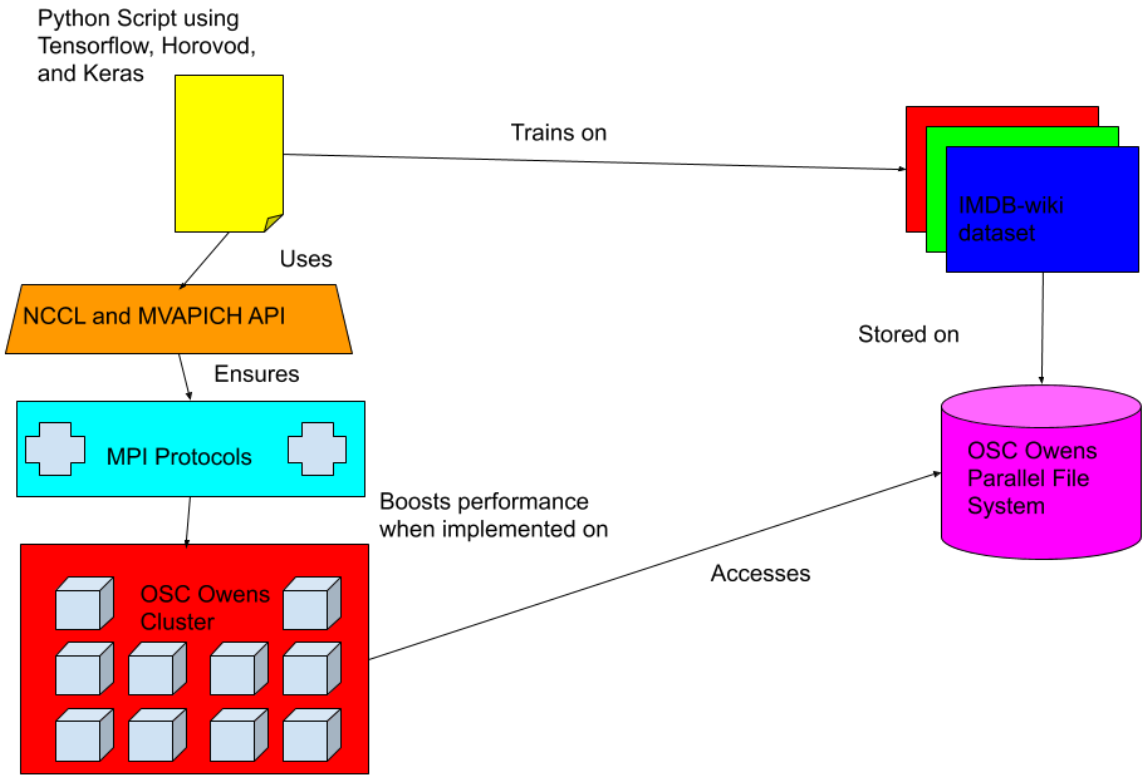

Figure 1

2. Choose one of the following three large-scale datasets.

IMDB-Wiki Dataset (Age Estimation Dataset 460K + 62K images)

3. Choose three different DNN models available for your chosen dataset.

ResNet, Inception, VGG

4. (10 points) Run the experiments for each DNN using a single node (If you have these numbers from Lab #1, please re-use these to save SUs. Otherwise, you can re-run these experiments.)

For single node run, we both ran the test under a single-node CPU environment and a single-node GPU environment.

For single-node CPU run, we first conducted the experiments with a varied number of processes and threads and compared the three DNN models in terms of their throughput (images per second). To find the best process/thread combination, we fixed the batch size to be 128.

Based on the same dataset (IMDB-Wiki), we were able to run tests in three process/thread combinations: 1 process with 28 threads, 2 processes with 14 threads each and 4 processes with 7 threads each. The tests are guaranteed on all the three DNN models: InceptionV3, ResNet50 and VGG16. As we can see from Figure 2, among all models, the throughput showed an increase as we changed from 1 process with 28 threads to 4 processes with 7 threads each. Furthermore, they reached a quite stable point at 4 processes with 7 threads each. To pick a most satisfactory configuration, we looked to the specific numbers. For InceptionV3, it reached 19.46 images per second, which is the highest for this model. For ResNet50, it reached 22.97 images per second, slightly less than its previous configuration (23.34 images per second). For VGG16, because of its enormous model size (528MB) and number of trainable parameters (138,357,544), it reached only 8.57 images per second, which is also the maximum across configurations. As a result, two out of three models staged a peak throughput when placed under 4 processes with 7 threads each and we picked that as our optimal configuration.

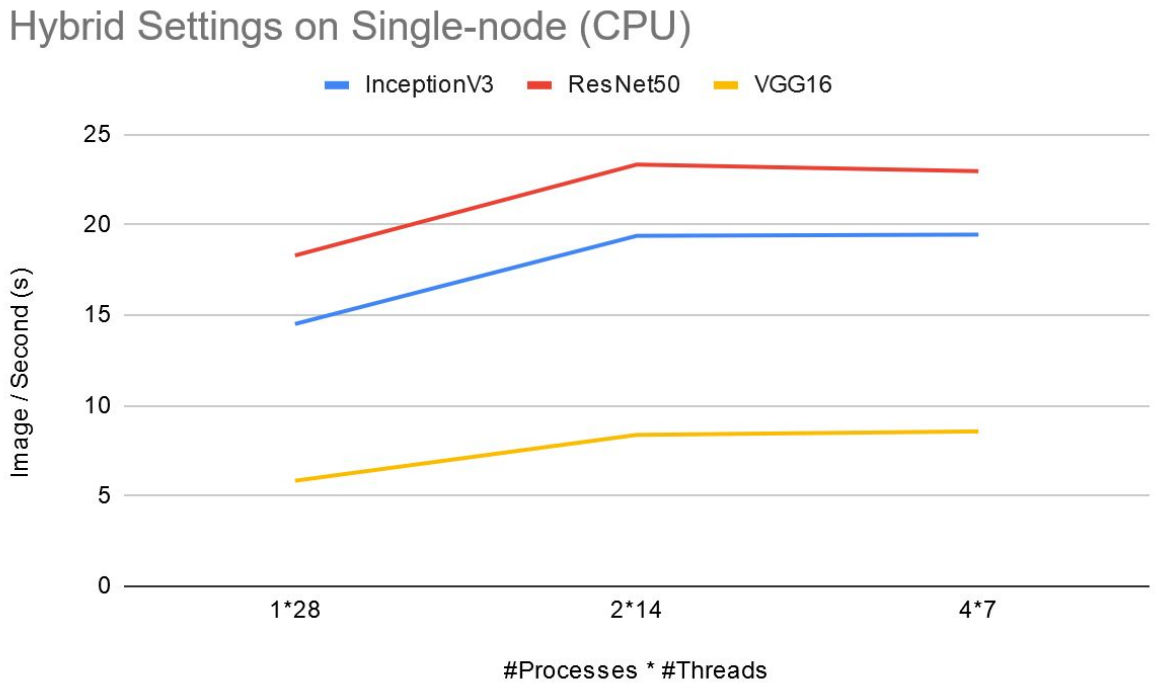

Figure 2

Next, to find the best global batch size for a single-node CPU run, we ran the test under a different number of global batch sizes. From Figure 3, during the early phase of the increase in batch size, we noticed a significant boost in throughput until a global batch size of 8. Following that, we experienced a flat trend according to Figure 3. It is worth mentioning that we chose the global batch size to be 128 because of that for the following strong scaling tests, we will be dividing global batch size according to different numbers of nodes up to 8. To make the test sufficient in resource background, we chose global batch size to be 128 so that each process in each node will be having a reasonable amount of local batch size.

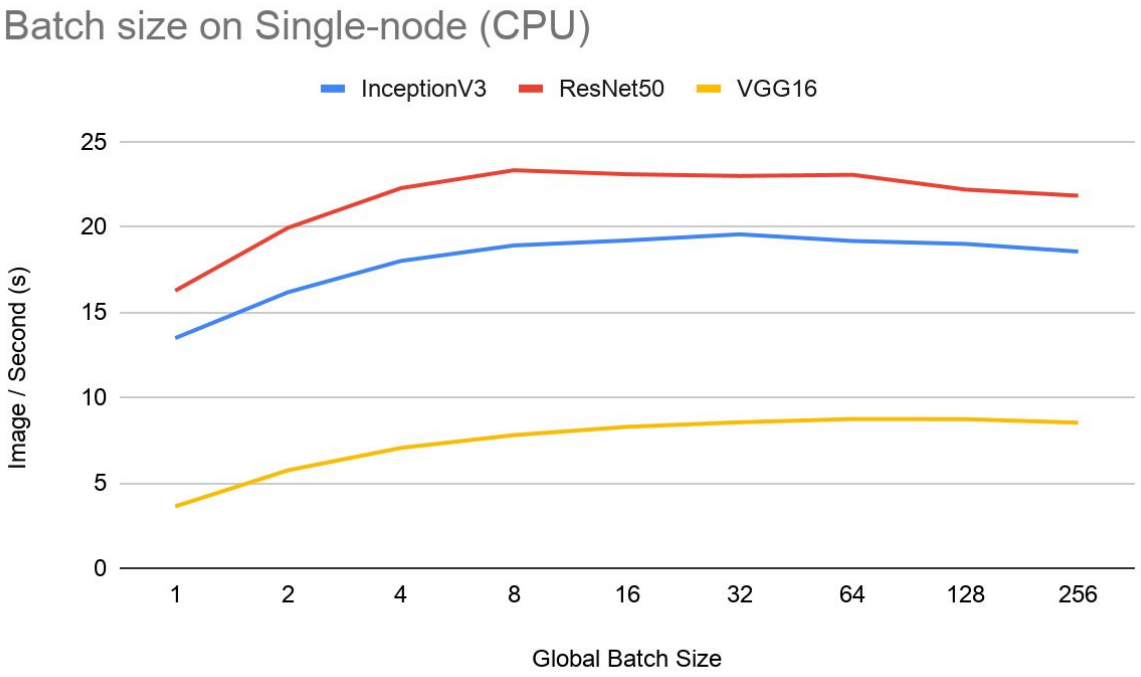

Figure 3

For the next step, a single-node GPU test is conducted to find out the optimal global batch size on a single GPU. As we can see from Figure 4, the throughput of each model is in a gradual increase across all DNN models. At the point where global batch size is set to 128, we observed that InceptionV3 was showing a throughput of 44.4 images per second, which was slightly less than 64 global batch size. ResNet50 showed a peak throughput of 66.11 images per second while VGG16 also showed a peak throughput of 63.88 images per second. Two out of three DNN models performed at their peak at a global batch size of 128, we thus chose this batch size as our desired configuration.

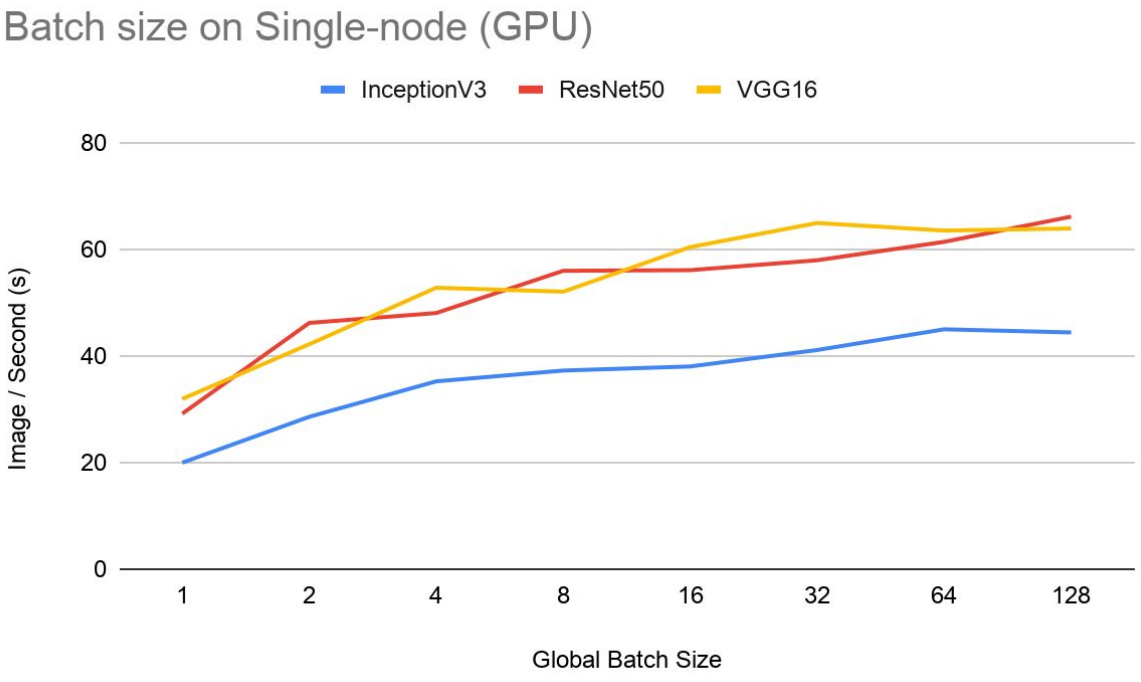

Figure 4

5. (25 points) After you have the data for the best batch size for a single node, run the experiment with this best batch size per processor for 1, 2, 4, and 8 nodes (Weak Scaling) [batch size per GPU/CPU remains same]

As we concluded a best global batch size of 128 for a single node both under CPU run and GPU run, we try to utilize such batch size to study the throughput boost under a weak scaling environment. In a weak scaling preset, we tried to maintain the batch size passed to each process the same while increasing the number of nodes we utilize. To be specific, we divide the best global batch size by the best process number configuration to get the best local batch size for each process which is 32. In a CPU weak scaling preset, one node contains 4 processes and each process will be assigned a local batch size of 32. We are not changing the local batch size but rather increase the number of allocated nodes.

We first ran our test on a multi-node CPU platform. From Figure 5, we can see that when we maintained the same local batch size, we could see a huge increase in throughput as we increased the number of nodes. For InceptionV3, we had a throughput of 136.51 images per second at 8 nodes. Compared to 1 node (19.4 images per second), we observed a 7 times throughput boost. For ResNet50, we had a throughput of 156.33 images per second at 8 nodes, that was 6.84 times throughput boost compared to 1 node (22.84 images per second). For VGG16, we had a throughput of 66.25 images per second, that was 7.51 times throughput boost compared to 1 node (8.82 images per second).

Looking at the results we obtained from the above tests, it is worth mentioning that due to the exceptional large scale of VGG16 and the enormous number of neurons, the base throughput of VGG16 is rather low on CPU platforms but were still able to speed up to 7 times and more when we weak scale based on the number of nodes allocated.

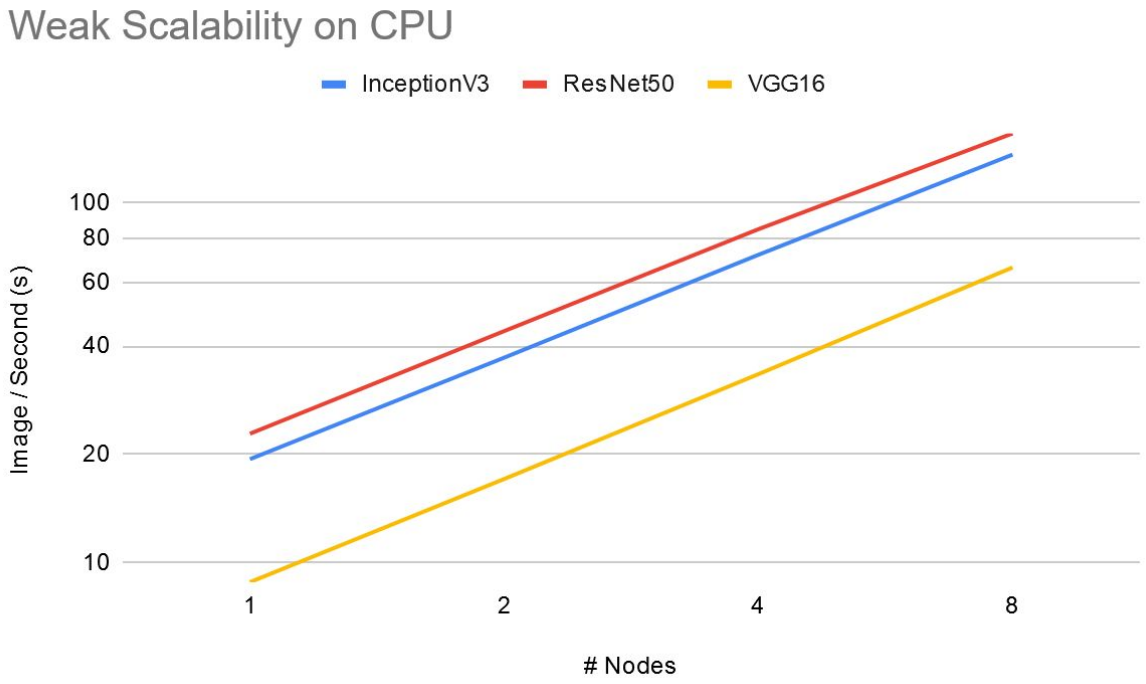

Figure 5

Next, we conducted our weak scaling test on a multi-node GPU platform. Since each GPU has only one process, we will make the best global batch size of 128 a local batch size and assign it to each GPU. Each node has one GPU, thus we will try a different number of nodes.

As we can see from Figure 6, we observed a sharp increase in throughput from 1 node to 8 nodes. To be specific, InceptionV3 had a throughput of 312.17 images per second at 8 nodes which was a 6.6 times boost compared to 1 node. ResNet50 had a throughput of 508.88 images per second at 8 nodes which was an 8.03 times boost compared to 1 node. VGG16 had a throughput of 671.09 images per second at 8 nodes which was a 10.56 times boost compared to 1 node.

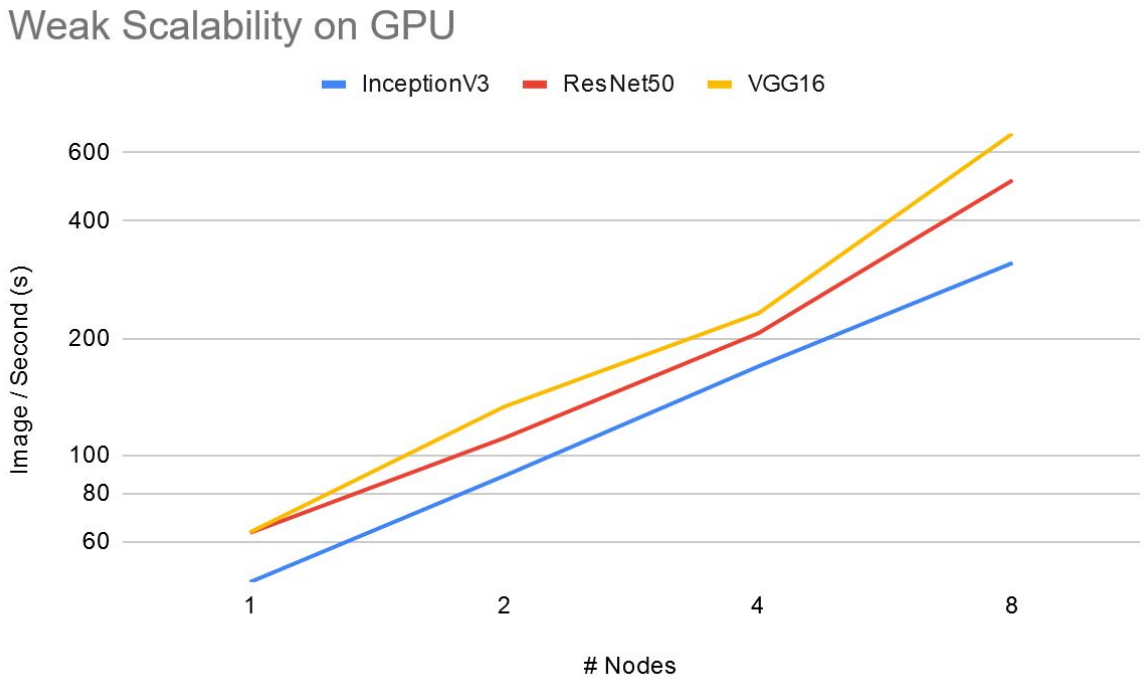

Figure 6

Compare the result of CPU and GPU, we have an interesting finding. The throughput of VGG16 is the worst in CPU running, but it becomes the best if we use GPU. It's because despite the largest model size and the number of trainable parameters, VGG16 model is a relatively simple model, which means it's not that deep and complex to the other models. It makes it benefit from the increasing computing power from CPU to GPU.

6. (25 points) Select a reasonable batch size for a single node, run the experiment by keeping the same effective batch size as one GPU for 1, 2, 4, and 8 nodes (Strong Scaling) [e.g. Choose a batch size of 64 for 1 processor, then use BS (per processor) = [16, 8, 4] for [2, 4, 8] nodes, respectively. Therefore, the effective batch size (EBS = #processors x BS) stays the same].

In this part, we select global batch size 128 for a single node and run the experiments. So when running with CPUs, there are 4 processors per node. So if we run with 1 node, the local batch size is 32 for each processor. When we run with 2 nodes, the local batch size is 16 for each processor. When we run with GPUs, there are 1 processor per node. So if we run with 1 node, the local batch size is 128 for each node. If we run with 2 nodes, the local batch size is 64 for each node.

First, we run the experiments with different numbers of nodes without GPU. Figure 7 shows the throughput and scalability when we run for CPU only. From this figure, we could see that when we increase the number of nodes, the throughput (images/second) of training these models will also increase. From the throughput of training different models, we could see that the scalability of the ResNet50 model is the best, and the speedup is 7.2. On the other hand, the speedup of the InceptionV3 and VGG16 are 6.8 and 6.3. These are good numbers for the scalabilities of the 3 models.

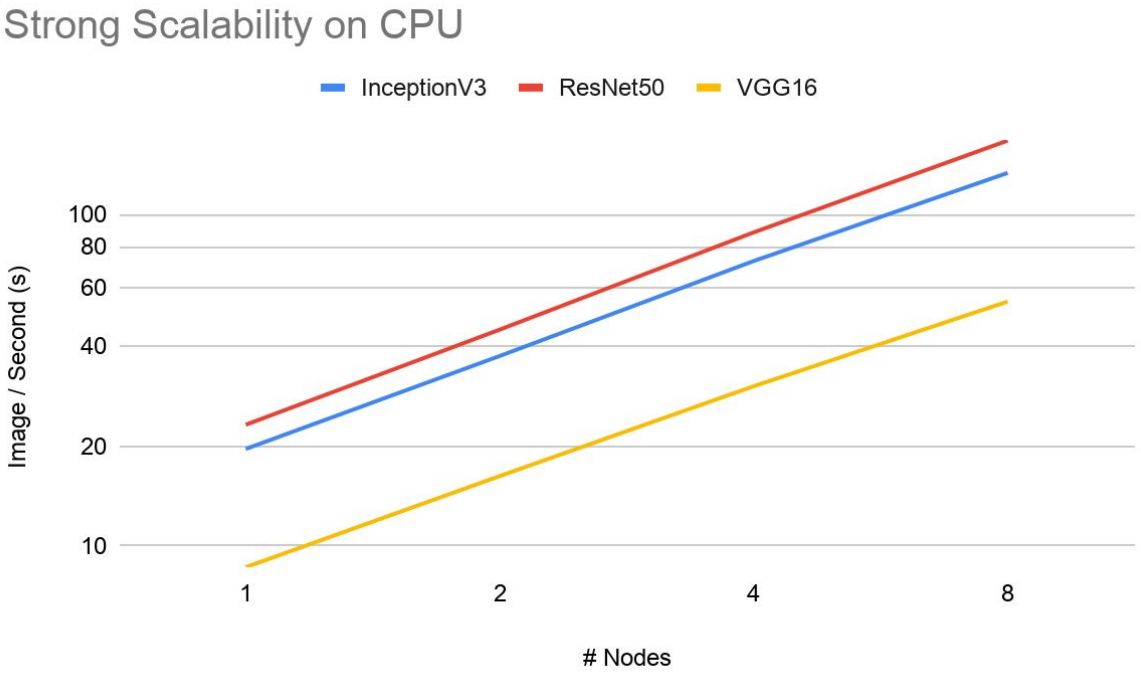

Figure 7

Figure 8 shows the throughput and scalability when we run for GPU only. From this figure, we could see that when we increase the number of nodes, the throughput (images/second) of training these 3 models will also increase. From the throughput of training different models, we could see that the scalability of training the InceptionV3 model with GPUs is the best. When we increase the number of nodes, the throughput of training the InceptionV3 model improves a lot. For the ResNet50 model, the speedup difference is small when increasing the number of nodes. The trend is not that linear after 4 nodes. We think that's because the ResNet50 model is more complex and deeper, which makes it hard to paralleled and gains less benefit from the growth of computing power.

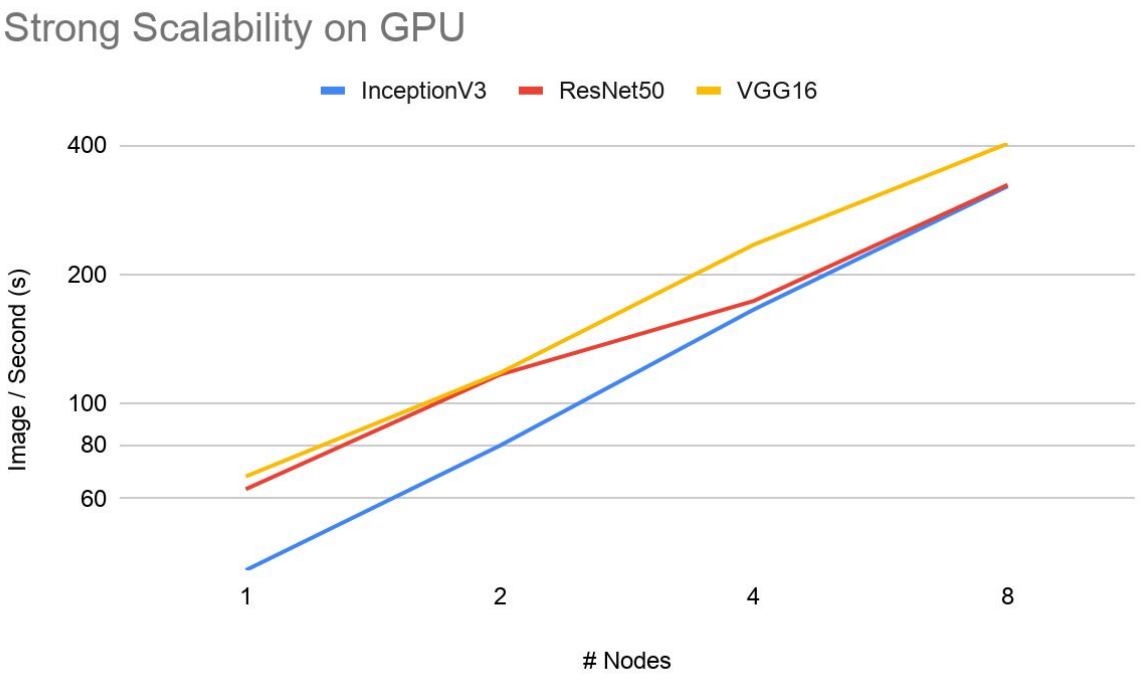

Figure 8

Compared to the results of CPU and GPU, there is a similar trend in VGG16 which is discussed in the Weak Scaling part.

7. (20 points) What can you conclude from this study? Write a few paragraphs to explain your results and observations.

In this lab, we selected a large scale data set and trained 3 models (ResNet50, InceptionV3, VGG16) with different settings of batch size, number of nodes, with or without GPUs.

First, we trained the 3 models with our selected dataset on a single node. We varied the batch size when training these models. From our experiments, we found that during the early phase of the increase in batch size, we noticed a significant boost in throughput until a global batch size of 8. For the following parts of the lab, we selected the best batch size 128.

Then we ran the experiments for different numbers of nodes with the same batch size. When training models only with CPUs, we could see a huge increase in throughput as we increased the number of nodes when we maintained the same local batch size. The speedup increased linearly when we increased the number of nodes if we kept the batch size the same for all the processes. When we ran the experiments with GPUs, the result was roughly the same. The scalability was good when we increased the number of nodes of running on CPUs or GPUs. It's to say if we can increase the effective (global) batch size as large as possible and keep the local batch size the same, the scalability will remain good within these 3 models under this architecture.

Then we ran the experiments for different numbers of nodes with the same effective batch size. In the experiments, the scalabilities of these 3 models on CPU and GPU are still good. It performs almost linearly if we increase resources to the same problem size. It's to say if we can not keep increasing the effective (global) batch size, then adding resources to let each worker responsible for a smaller dataset is still effective for training these models, at least under 8 nodes. The only exception is running ResNet50 on GPU. The performance decrease compared to other 2 models, which indicates adding GPU nodes to this model is not as effective as other models.

From this lab, we could find that when we run the experiments with more nodes, the speed of training models will also increase a lot. The scalability of training DNN models using TensorFlow with Horovod is good. It delivers greater computational power when the amount of resources is increased. But if we use the same effective batch size for all nodes, the parallelization efficiency will decrease as the number of nodes increases. The upper limit of speedup is determined by the fraction of parallized code. So, parallel computing with many processors is useful for highly parallelized programs. But when we use the same batch size for all the processes, the size of the problem will increase. The parallel part in the code scales linearly with the number of nodes. The serial part does not increase with respect to the size of the problem. Therefore, when we increase the number of nodes, the speedup will also increase linearly when we run the experiments either with CPUs or with GPUs.