PYTORCH DISTRIBUTED – ARULNITHI P (12210280)

Distributed training with single GPUs in a cluster of machines is particularly useful when working with large datasets or complex models. It allows for the distribution of the computational workload across multiple machines, reducing the overall training time while also enabling the handling of larger datasets. This can be particularly important in domains such as natural language processing, computer vision, and deep reinforcement learning. However, distributed training comes with several challenges. For example, data needs to be split across machines, and the communication between them needs to be carefully managed to ensure that the model is trained efficiently. Additionally, hardware failures or network issues can occur, making it necessary to implement fault-tolerance mechanisms to ensure that training is not interrupted.

To implement distributed training effectively, it is necessary to use a distributed computing framework that can handle these challenges. PyTorch and TensorFlow are two popular deep learning frameworks that provide distributed training capabilities. PyTorch uses the torch.distributed package, which provides support for various distributed backends such as TCP. TensorFlow, on the other hand, uses its built-in distributed computing API, which supports various architectures, including parameter servers, peer-to-peer, and collective communication. One critical concept in distributed training is the use of a global rank. This is a unique identifier for each process, which is used to determine its position in the distributed system. The global rank helps to coordinate the communication between the machines, enabling them to work together effectively.

In summary, distributed training with single GPUs in a cluster of machines can be a powerful way to accelerate the training of deep learning models. However, it comes with several challenges that need to be addressed, such as data splitting, communication, and fault-tolerance. Using a distributed computing framework like PyTorch can help to overcome these challenges, while the global rank is an essential concept for coordinating the communication between the machines.

Torchrun

I have used torchrun for fault tolerance it makes it easy for differencing number of GPUs, it sets up the world size and ID number of the GPU. It helps us to load the last saved snapshot and resume the operation from there.

torchrun --nproc_per_node=1 --nnodes=World_size --node_rank=0--rdzv_id=456--rdzv_backend=c10d rdzv_endpoint=103.147.138.252:29603 multinode.py 50 10

--nproc_per_node=1 – describes the number of GPUs in single machine.

nnodes – determines the world size(total number of machines in the cluter.

rdzv_id=456 – random number for identifying the process.

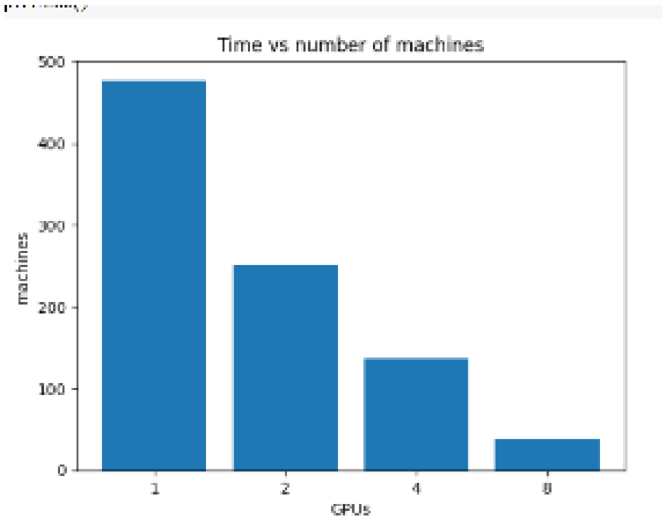
rdzv_backend=c10d - communication protocol for sharing of the gradients across the machines.

rdzv_endpoint=103.147.138.252:29603 – The gradients are shared by the TCP layer. This Ip address should be the IP address of the master node.

Experimental setup

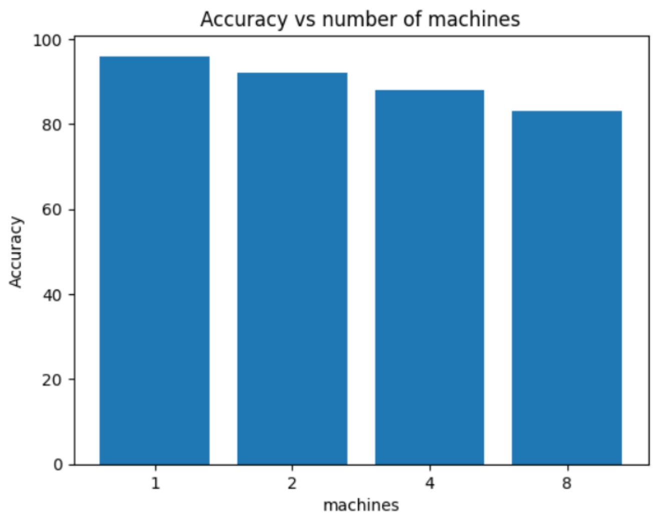
I have run the model in Nvidia RTX 3090 GPUs. To run code in multiple machines we need to clone the code in multiple machines and execute the torchrun command keeping one node for aggregation of gradients. The performance ultization in this machine will be less compared to other nodes because it to compute aggregate gradients.

Result



As we can see that time taken to train the model has significantly reduced.

Accuracy



As the number of machines increase, there is a drop in accuracy, which is an important consideration in distributed computing. However, it is worth noting that when training on two machines, the accuracy was 92%. In contrast, when using a single machine with multiple GPUs, the accuracy increased to 94%. Although the GPUs used were the same, there was a 2% decrease in accuracy when using a multi-node setup due to the additional communication delay.

Similarly, when training on four machines, the accuracy dropped to 88%, but when using a single machine with four GPUs, the accuracy increased to 90%. This difference can be attributed to the communication overhead between the machines in the distributed setup.

Therefore, it is crucial to carefully consider the trade-offs between accuracy and performance when designing distributed computing systems. While scaling up the number of machines can increase the computational power, it may come at the cost of accuracy due to communication overhead. On the other hand, using a single machine with multiple GPUs can provide a balance between performance and accuracy.

So it is better to train on 8 GPU on single machine than 8 machines with a single GPU.