**Lab 2**

**Radha Gulhane  
Date: 15 – Nov - 2022**

**Q.1**

(20 points) Investigate two options for running multi-node distributed training with PyTorch and explain your understanding by highlighting the major concepts and components needed for it.

Write a clear, concise paragraph and draw a block diagram to explain your understanding of each option (e.g. DDP, RPC-based, and Horovod for PyTorch Distributed)

Highlight the technologies used in the distributed approach you have used (e.g. MPI collectives, tensor fusion, and hooks for Horovod). What benefits/tradeoffs exist for each option?

=>  
Two options:  
1. Horovod for PyTorch Distributed  
2. Distributed Data Parallel for Pytorch:

Major Components required:  
world\_rank : total number of processes running across all the nodes at one time  
local\_world\_rank : total number of processes running on each node   
process\_group: required to initialize process group and information about communication library  
communication interface: mpi, nccl, etc.  
collective operation: all\_reduce

**Horovod for PyTorch Distributed:**

Horovod is distributed deep learning framework and provides support not only for Pytorch but also Tensorflow, keras, Apache MXNet. The motivation of Horovod is to make distributed deep learning fast and easy to use by allowing minimal changes in script of single node deep learning.

**Block Diagram:  
Please find the Block diagram of Horovod for PyTorch Distributed with explanation of necessary modules.**

Diagram

Description automatically generated

Figure .Horovod for PyTorch Distributed

**Communication interface and MPI collectives:**Horovod supports Gloo, MPI, NCCL, oneCCL

Horovod module(hvd) provides the interface to access [MPI](http://mpi-forum.org/) concepts. Provides operations like hvd.allgather, hvd.allreduce, hvd.alltoall, hvd.broadcast, or hvd.grouped\_allreduce, also many high-level utility objects such as hvd.DistributedOptimizer come with support for process sets.

Besides Horovod’s fundamental operations like hvd.allgather, hvd.allreduce, hvd.alltoall, hvd.broadcast, or hvd.grouped\_allreduce, also many high-level utility objects such as hvd.DistributedOptimizer come with support for process sets.

In Horovod, optimizer can be initialized to hvd.DistributedOptimizer which help gradient computation to original optimizer, averages gradients using MPI collective(allreduce, or user specified method) and then applies those averaged gradients.

**Tensor Fusion:**

Horovod also supports Tensor Fusion, the ability to interleave communication and computation coupled with the ability to batch small allreduce operations which results in performance improvement.

**Hooks:**It can be registered for each tensor or nn.Module and triggered during forward or backword pass. Provides benefit for debugging, modifying gradients during backward pass.

In Horovod: hvd.BroadcastGlobalVariablesHook(0) to broadcast initial variable states from rank 0 to all other processes  
Similarly in a submitted program, hvd.broadcast\_parameters(network.state\_dict(), root\_rank=0) has been used.

**Distributed Data Parallel for Pytorch:**

Major components required:  
Pytorch, torch.distributed

DDP is the class in PyTorch can be used for distributed deep learning and it is based on torch.distributed package. This provides data parallelism and synchronized gradients across each model replica. It uses   
This module utilizes multiprocessing where a process is created for each GPU and thus also provides advantage over DataParallel module provided by PyTorch which uses multithreading.

**Block Diagram:**

**Please find the Block diagram of Distributed Data Parallel for Pytorch with explanation of necessary modules.**

Diagram

Description automatically generated

Figure . Distributed Data Parallel for Pytorch

**Communication interface and MPI collectives:**

torch.distributed supports three built-in backends, gloo , mpi , nccl.

Ideally, it is advisable to use NCCL for distributed GPU training and Gloo backend for distributed CPU training  
While using DDP for distributed training, the backend should be defined while initializing process groups.

**Q. 2.** (10 points) Run the experiments for ResNet-50 using a single node using PyTorch  
1.Run the GPU version  
2.Report throughput (training time) in terms of samples/second or images/second  
3.Vary the batch size and find out the best batch size that gives you the highest throughput.

=>

|  |  |  |
| --- | --- | --- |
| ResNet-50 using a single node using PyTorch | | |
|  |  |  |
| Batch Size | epoch | Throghput |
|  |  |  |
| 128 | epoch 1 | 1271 |
| epoch 2 | 1315 |
| epoch 3 | 1304 |
| epoch 4 | 1304 |
| epoch 5 | 1306 |
| **Throughput (Images/ seconds)** | | **1300** |
|  |  |  |
| 256 | epoch 1 | 2053 |
| epoch 2 | 2142 |
| epoch 3 | 2147 |
| epoch 4 | 2132 |
| epoch 5 | 2132 |
| **Throughput (Images/ seconds)** | | **2121** |
|  |  |  |
| 512 | epoch 1 | 2169 |
| epoch 2 | 2256 |
| epoch 3 | 2256 |
| epoch 4 | 2256 |
| epoch 5 | 2255 |
| **Throughput (Images/ seconds)** | | **2238** |

**Conclusion: Best batch size which gives you highest throughput => 512**

**Q. 3.** (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, and 4 nodes using both (Weak Scaling) [batch size per GPU remains same]  
1. Run GPU experiments for both Horovod and PyTorch distributed  
2. Report the throughput for both frameworks  
3. Report the scalability/speedup for multiple nodes by creating a graph that presents images/second on the y-axis and #nodes on x-axis.

=>

**Batch size: 512**

|  |  |  |
| --- | --- | --- |
| **# of Nodes** | **Throughput** | |
|  | **Horovod** | **PyTorch distributed** |
| 1 | 1689 | 2076 |
| 2 | 2948 | 3035 |
| 4 | 5706 | 5549 |

**Q. 4.** (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, and 4 nodes (Strong Scaling) [e.g. Choose a batch size of 32 for 1 processor, then use BS (per processor) = [16, 8] for [2, 4] nodes, respectively. Therefore, the effective batch size (EBS = #processors x BS) stays the same].  
1. Run GPU experiments for both Horovod and PyTorch distributed  
2. Report the throughput for both frameworks  
3. Report the scalability/speedup for multiple nodes by creating a graph that presents images/second on the y-axis and #nodes on x-axis.

=>  
**Effective Batch Size = 512**

|  |  |  |
| --- | --- | --- |
| **# of Nodes** | **Throughput** | |
|  | **Horovod** | **PyTorch distributed** |
| 1 | **1657** | **2076** |
| 2 | **2674** | **2829** |
| 4 | **3253** | **3323** |

**Q. 5.** (20 points) What can you conclude from this study? Write a few paragraphs to explain your results and compare Horovod and PyTorch Distributed.

=>  
**Analysis of throughput with respective to number of nodes:**  
As we increase the nodes, throughput is increasing for both strong and weak scaling. But as this experiment was limited to only 4 nodes, no significant communication overhead was observed as increasing no. of nodes and thus no decrease in throughput. To conclude, as increase with number of nodes till 4 nodes, we are able to achieve parallelism.

**Comparison of Horovod and PyTorch Distributed for Weak Scaling:**

For Weak scaling, with single node, PyTorch Distributed is performing slightly better than Horovod in terms of throughput. with 2 node GPU, the throhgput for both is nearly same. However, with 4 node distributed training Horovod is giving slightly greater throughput than PyTorch Distributed.  
  
For Strong Scaling, it is observed that PyTorch Distributed is giving more throughput than Horovod. However, as we increase the node the difference between the throughput by both the options is decreasing gradually.

Horovod seems to me more synchronous than Pytorch Distributed, DDP overlaps backward computation with communication. But horovod synchronizes models in the optimizer.step() and don’t overlap with backward computation. Though Horovod provide support for making this process asynchronous way, but by default, it is synchronous.