Multi-node TensorFlow training on Docker with Horovod

This document gives the instructions to create and run docker containers with horovod for multi-node TensorFlow training.

HW :

- N number of Intel® Xeon® Scalable processors
- A minimum of 8GB memory filled in all DIMM channels i.e >=48 GB
- A minimum of 25Gig Ethernet. (or IB/OPA)
- A storage node with >=1000 GB SSDs

SW:

- TensorFlow (intel-tensorflow==1.15.2)
- Horovod (horovod == 0.16.4)
- Openmpi (latest from linux APT/YUM repositories)
- Docker (latest from linux APT/YUM repositories)

Installation instructions

Prepare N number of Xeon® SP servers with CentOS 7.7 / ubuntu18.04 installed with NW configurations set.

Docker installation:

Instructions for CentOS 7.7 with sudo user

```
sudo yum install -y yum-utils   device-mapper-persistent-data   lvm2
sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
sudo yum install docker-ce docker-ce-cli containerd.io
systemctl restart docker
systemctl enable docker
sudo usermod -aG docker $USER
```

Logout and login

Docker image preparation

Base Dockerfile is taken from https://raw.githubusercontent.com/horovod/horovod/master/Dockerfile.cpu and modified for CPU requirements as given here https://raw.githubusercontent.com/vdevaram/deep_learning_utilities_cpu/master/tensorflow/Dockerfile_ompi_hvd_tf.cpu

- Download the modified Dockerfile for CPU

```
wget https://raw.githubusercontent.com/vdevaram/deep_learning_utilities_cpu/master/tensorflow/Dockerfile_ompi_hvd_tf.cpu
```

- Build the container image

```
docker build -f Dockerfile_ompi_hvd_tf.cpu -t hvd_tf_1.15:1.0 .
```

- After completion please check the image in your system

```
docker images
```

- You should be able to see below result

```
REPOSITORY      TAG          IMAGE ID        CREATED          SIZE
hvd_tf_1.15     0.1          7516b5728c9a    24 hours ago     1.54GB
```

- Either you can push the image to **docker hub** or it can be saved and loaded into all other nodes as below -

```
docker save -o hvd_tf_1.15.tar hvd_tf_1.15:1.0
```

- Copy to the destination nodes

```
docker load -i hvd_tf_1.15.tar
```

- Alternatively push the image to **docker hub**

```
docker tag <image_id> <docker_hub_username>/hvd_tf_1.15:1.0
docker push <docker_hub_username>/hvd_tf_1.15:1.0
```

- This image can be pulled to any of the nodes using the below command

```
docker pull <docker_hub_username>/hvd_tf_1.15:1.0
```

- **docker images** should give similar result as above after load or pull commands given above.
- For experimentation a sample image can be downloaded using **docker pull vdevaram/hvd_tf_1.15:1.0**.

Multi-node preparation

Here are some guidelines needed to check whether all systems are ready for multi node testing

MEMORY BANDWIDTH CHECK

Download the Memory latency checker from Intel® website to check whether all memory channels are filled with DIMMs to get best bandwidth -

Download and copy to your servers from https://software.intel.com/en-us/articles/intelr-memory-latency-checker

untar the file and run mlc check as given below

```
tar -zxvf mlc_v3.8.tgz
cd Linux
./mlc
```

Here is the report for BW which is around 200 GB/s for Intel® Xeon® SP server with 2666 MT/s.

```
Measuring Peak Injection Memory Bandwidths for the system
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using all the threads from each core if Hyper-threading is enabled
Using traffic with the following read-write ratios
ALL Reads        :  209508.2
3:1 Reads-Writes :  193293.0
2:1 Reads-Writes :  190897.9
1:1 Reads-Writes :  179218.4
```

NETWORK BANDWIDTH CHECK

Setup the network link for better latency

```
sudo tuned-adm profile network-latency
```

Iperf3 is a tool which can be used to check the ethernet speed between two nodes. Install and run the iperf3 as below -

```
sudo yum install iperf3 -y
```

Iperf3 requires firewall to be disabled. Incase of secured environments modify the iptables to allow TCP packets (not mentioned here)

```
sudo systemctl disable firewalld
sudo systemctl stop firewalld
```

Select one node as server and another as client

On server node start the iperf3 as server mode

```
iperf3 -s
```

On client node start client mode

```
iperf3 -c  <ip of server node>
```

Check the reverse link BW too

```
iperf3 -c  <ip of server node>  -R
```

Here the report which shows around 23Gbps for 25Gig Ethernet link for both directions

```
Connecting to host 192.168.116.103, port 5201
[  4] local 192.168.116.104 port 59510 connected to 192.168.116.103 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec  2.71 GBytes  23.2 Gbits/sec  386    499 KBytes
[  4]   1.00-2.00   sec  2.71 GBytes  23.3 Gbits/sec  276    677 KBytes
[  4]   2.00-3.00   sec  2.73 GBytes  23.4 Gbits/sec   77    557 KBytes
[  4]   3.00-4.00   sec  2.67 GBytes  22.9 Gbits/sec  441    494 KBytes
[  4]   4.00-5.00   sec  2.72 GBytes  23.4 Gbits/sec  248    563 KBytes
[  4]   5.00-6.00   sec  2.72 GBytes  23.4 Gbits/sec  357    766 KBytes
[  4]   6.00-7.00   sec  2.72 GBytes  23.4 Gbits/sec  341    721 KBytes
[  4]   7.00-8.00   sec  2.73 GBytes  23.5 Gbits/sec    0    813 KBytes
[  4]   8.00-9.00   sec  2.73 GBytes  23.4 Gbits/sec   78    621 KBytes
[  4]   9.00-10.00  sec  2.72 GBytes  23.3 Gbits/sec  339    735 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  27.2 GBytes  23.3 Gbits/sec  2543             sender
[  4]   0.00-10.00  sec  27.2 GBytes  23.3 Gbits/sec                  receiver
```

SSH PASSWORD-LESS ACCESS

Keep one node as master worker and all others as slave workers

On the master generate the ssh key and copy the key to all slaves. Please note that since docker runs in the root access follow all these commands in root mode

```
su  root
ssh-keygen -t rsa
ssh-copy-id -i /root/.ssh/id_rsa.pub <user>@<slave_ip_1>
……….
ssh-copy-id -i /root/.ssh/id_rsa.pub <user>@<slave_ip_n>
```

After this each slave worker node should be accessible without password from the master worker node.

NODE SETUP

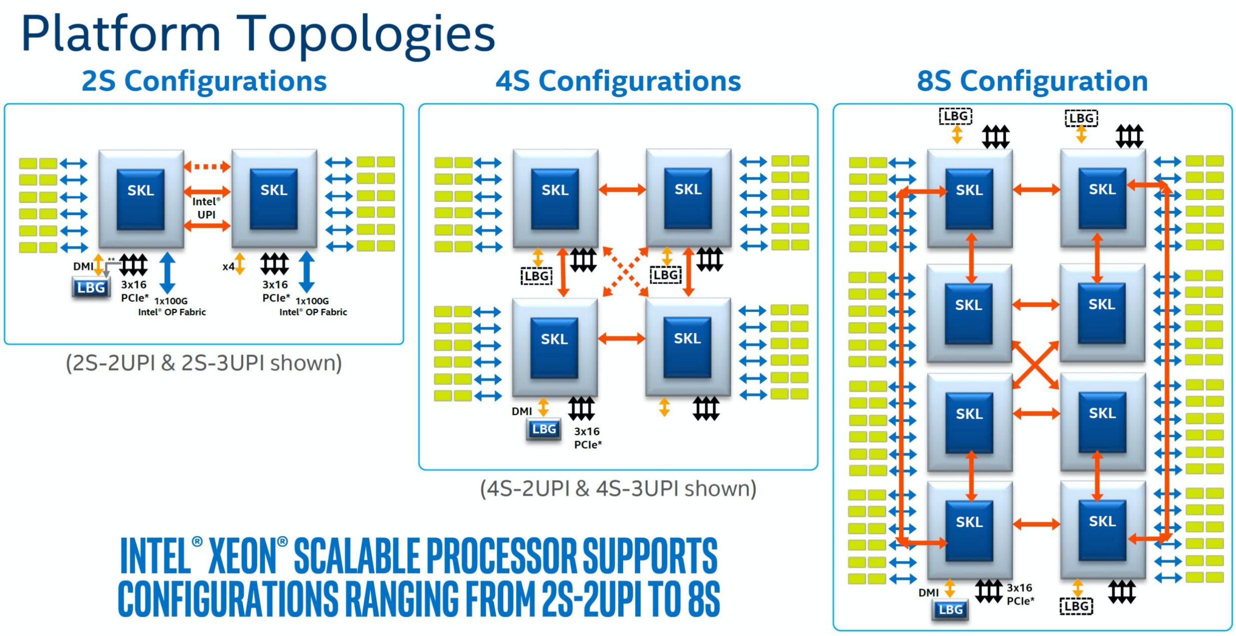

Fig1: Topological view of Xeon® SP processors.

For the experimental purposes, two socket Intel® Xeon® Gold 6248 CPU @ 2.50GHz is chosen. Get the cpu info with below command.

```
lscpu
```

Above command gives the details of the cpu as below. Please note the highlighted regions. Usually Intel® Xeon® processors comes as two socket servers. Each socket is independent CPU system with dedicated memory and both sockets are connected with **UPI** links. Each socket is termed as a numa node and can be controlled individually with **numactl** commands. For below experiments **hyper threading is disabled** for optimised performance. Also as highlighted the flags section should contain **AVX-512** flags for optimum usage of the server. If not seen, check the BIOS settings or hypervisor settings.

```
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              40
On-line CPU(s) list: 0-39
Thread(s) per core:  1
Core(s) per socket:  20
Socket(s):           2
NUMA node(s):        2
Vendor ID:           GenuineIntel
CPU family:          6
Model:               85
Model name:          Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz
Stepping:            7
CPU MHz:             3199.768
CPU max MHz:         3900.0000
```

```
CPU min MHz:           1000.0000
BogoMIPS:              5000.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              28160K
NUMA node0 CPU(s):     0-19
NUMA node1 CPU(s):     20-39
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art
arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni
pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid dca
sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm
3dnowprefetch epb cat_l3 cdp_l3 intel_pt ssbd mba ibrs ibpb stibp ibrs_enhanced tpr_shadow
vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm cqm
mpx rdt_a avx512f avx512dq rdseed adx smap clflushopt clwb avx512cd avx512bw avx512vl
xsaveopt xsavec xgetbv1 cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local dtherm ida
arat pln pts hwp hwp_act_window hwp_epp hwp_pkg_req pku ospke avx512_vnni md_clear
spec_ctrl intel_stibp flush_l1d arch_capabilities
```

Once the above tests are done for all nodes and showing the expected results, your nodes are ready for multi-node deep learning training.

Multi-node benchmarking

In this document TensorFlow benchmarks are chosen for deep learning training with classification workloads on Intel® Xeon® SP platform. Here guideline for manual invocation of containers is given which can be automated with Kubernetes further when using large cluster of nodes.

Download the TensorFlow benchmarks from GitHub to your workspace

```
export WORKSPACE=$HOME/workspace
mkdir -p $HOME/workspace
git clone https://github.com/tensorflow/benchmarks.git
cd benchmarks
git checkout cnn_tf_v1.15_compatible
```

Run the docker container on all workers with below command. Please keep your dataset mapped to storage node and mount the path to the container with **-v** option. If the containers needs to be started with **Kubernetes**, make sure the network is properly exposed if in case **privileged** option is not supported in your environment

```
docker run --name <temp_name> -v $WORKSPACE:/workspace -v /root/.ssh:/root/.ssh --network=host  --privileged -it <tf_hvd_container_name>
```

Once the containers are started in interactive mode use the below commands to start the multi-node training -

On each slave worker node run **sshd** service before starting the master worker

```
/usr/sbin/sshd -p <any private port id>; sleep infinity
```

On master worker run the training command -

```
HOROVOD_FUSION_THRESHOLD=<a value suggested by Horovod> mpirun -np <number of processes>  --map-by ppr:<parts per socket>:socket:pe=<number of cores per process> --allow-run-as-root --mca plm_rsh_args "-p <any private port id> " -mca btl_tcp_if_include <interface for multi-node training> -mca btl ^openib -mca pml ob1 -H <master_ip>:<port_id>,<slave_1_ip>:<port_id>, ....,<slave_n_ip>:<port_id>   --oversubscribe --report-bindings -x LD_LIBRARY_PATH -x HOROVOD_FUSION_THRESHOLD -x OMP_NUM_THREADS=<num_cores_per_process>-1 --batch_size <batch_size>  --num_batches <total batches for training> --distortions=False —num_intra_threads <num_cores_per_process> --num_inter_threads <num_threads_per_core>  --local_parameter_device cpu --variable_update horovod --horovod_device cpu
```

Explanation on parameters more relevant for CPU settings in above commands -

- **--map-by** ppr:<parts per socket>:socket:pe=<number of cores per process>

Suggested to use minimum one worker per socket to reduce the load on UPI during training. Also choose right performance saturation point by increasing the ppr and balancing with batch size.

- **--mca** plm_rsh_args

ssh args to be provided. Here any private port like ‘12345’ can be used for ssh communication between the workers. Please make sure sshd service on same port (ex: 12345) is open on all slave workers

- **--mca** btl_tcp_if_include

This is to set the inferface to be used for all workers communication. It is required if TCP over ethernet is used for communication.
All other **mca** options are to disable unnecessary network interfaces. Refer to openmpi man page for more information https://www.open-mpi.org/doc/v2.1/man1/mpirun.1.php

- OMP_NUM_THREADS

To fix the number of cores to be used for each process. Provide a number less than allocated in the **--map-by** command.

- KMP_BLOCKTIME

Allow the cores to not to go to sleep to take up the execution of next layer immediately. Usually 1 sec is good for many cases

- KMP_AFFINITY

To set the CPU affinity for workload to reduce the core switch over

Example for 6248 server is given below :

As mentioned in previous section two socket 6248 consists of 20 cores per socket is used for experimentation.

On master:

```
HOROVOD_FUSION_THRESHOLD=134217728 mpirun -np 8 --map-by ppr:2:socket:pe=10 --
allow-run-as-root --mca plm_rsh_args "-p 12345" -mca btl_tcp_if_include bond0.123 -mca btl
^openib -mca pml ob1 -H 192.168.116.103:9999,192.168.116.104:9999  --oversubscribe --
report-bindings -x LD_LIBRARY_PATH -x HOROVOD_FUSION_THRESHOLD -x
OMP_NUM_THREADS=9 -x KMP_BLOCKTIME=1  -x
KMP_AFFINITY=granularity=fine,verbose,compact,1,0 python3 -u /workspace/benchmarks/
scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py --model resnet50 --batch_size 64 --
num_batches 40 --distortions=False --num_intra_threads 10 --num_inter_threads 1  --
local_parameter_device cpu --variable_update horovod --horovod_device cpu
```

- Our experiments show two parts per socket with BS=64 gives the best. So **ppr** is selected as 2 resulting in 10 cores per socket.
- Two nodes, one for master and one for slave is selected. As per above ppr, each node consists of 4 workers. So **np** is 8.
- Due to this, **OMP_NUM_THREADS** is fixed at 9 (10-1).
- **num_intra_threads** should be selected accordingly as 10.
- As hyper threading is disabled, **num_inter_threads** to be selected as 1

On slave:

```
/usr/sbin/sshd -p 12345; sleep infinity
```

Wait for the horovod communication over sshd until the training is over. Please read the https://www.intel.ai/ai/wp-content/uploads/sites/69/ScalingDLwithTensorFlow_WP.pdf for more info on other environments.