A OSDI'20 Artifact Evaluation Appendix

A.1 Abstract

This appendix helps readers to reproduce the main evaluation results in our OSDI'20 paper: A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. Our system BytePS is open-sourced at GitHub, with all design and implementation points in the paper included. Users can find the documents on GitHub for basic usage and tutorials. Here, we provide a specific guideline to reproduce our main experimental results including:

- **Expr1**: End-to-end performance with different number of CPU machines. (Fig. 12)
- **Expr2**: Topology-aware intra-machine communication. (Fig. 13)
- **Expr3**: Scalability. (Fig. 14 and 15)

A.2 Artifact check-list

- **Run-time environment:** Ubuntu 18.04, CUDA 10.0, NCCL 2.4.8, tensorflow-gpu==1.15.0, torch==1.4.0, mxnet-cu100==1.5.0. Note that these versions are recommended but not mandatory. We have confirmed BytePS also work with other versions. Please see the documents on our public link for more details.
- **Hardware:** The most large scale experiments in this paper need 32 GPU machines each with eight V100 GPUs, and 32 CPU machines. PCIe-only and NVLink-based GPU machines are needed for Expr1 and Expr2. We use 100 Gbps RDMA (RoCEv2) network with full bisection bandwidth. But if resources are not available, readers are encouraged to perform experiments at smaller scale.
- **Metrics:** Images/sec, Tokens/sec.
- **Public link:** `https://github.com/bytedance/byteps`.
- **Experiments:** The code to reproduce the results is available at `https://github.com/byteps/examples`.
- **How much time is needed to prepare workflow (approximately)?:** Install BytePS only takes several minutes. Users can quickly get the example running in 10 minutes by following the tutorial (see below). Downloading the data sets may take some hours, but some of the experiments can use synthetic data without relying on real data sets.
- **How much time is needed to complete experiments (approximately)?:** Each experiment listed above should take at most a few minutes to get the speed metrics.

A.3 Description

A.3.1 How to access

- System code and documentations: `https://github.com/bytedance/byteps`
- Experiment code: `https://github.com/byteps/examples`

A.3.2 Hardware dependencies

Evaluating BytePS requires at least one machine with more than one GPU. However, one may expects higher gain from BytePS with more GPU machines. We encourage the readers to perform large scale experiments if resources are available. We used 100 Gbps RDMA network in the experiments. While BytePS also works with TCP, the performance is not shown in our paper. Readers can find the TCP performance here: `https://github.com/bytedance/byteps/blob/master/docs/performance.md`.

A.3.3 Software dependencies

Before installing BytePS, we assume one or more of the following frameworks have been installed: TensorFlow / PyTorch / MXNet. CUDA and NCCL are necessary for BytePS. The recommended (not mandatory) software versions are Ubuntu 18.04, CUDA 10.0, NCCL 2.4.8, tensorflow-gpu==1.15.0, torch==1.4.0, mxnet-cu100==1.5.0. For other dependencies, please refer to this Dockerfile for details: `https://github.com/bytedance/byteps/blob/master/docker/Dockerfile`.

A.3.4 Data sets and storage

Some experiments such as TensorFlow ResNet-50 (Fig. 14a) and MXNet VGG-16 (Fig. 14b) can support training with synthetic data. Other experiments require manually downloading the data sets (links provided). Usually we use a shared file systems (e.g., CephFS or HDFS) for storage, so that each machine can access the data.

A.4 Installation

Follow the install instructions here: `https://github.com/bytedance/byteps#quick-start`.

A.5 Basic tutorial

Follow this tutorial to launch BytePS step by step: `https://github.com/bytedance/byteps/blob/master/docs/step-by-step-tutorial.md`. Readers will get familiar with the whole workflow (including how to set up the environment variables) after successfully running a distributed training job. There are a few concepts for distributed training that we explain here: `https://github.com/bytedance/ps-lite/tree/byteps#concepts`.

Note that **we strongly recommend readers to walk through this basic tutorial first, and then move on to the next section for final evaluation**. In the following, we omit the details to set up the necessary environment.

A.6 Experiment workflow

Find the evaluation code at `https://github.com/byteps/examples`. There are altogether six models. Under the folder of each model, there is a script named "run.sh" that demonstrates the basic usage to run the model. Note that this script does not set up the necessary environment variables needed to launch BytePS. Therefore, readers need extra efforts to set them up for different processes, which is illustrated in the previously mentioned step-by-step tutorial. All experiments use all-reduce as the baseline. Our code is also compatible with Horovod. It is easy to switch to Horovod and test the performance for comparisons (see the instructions on GitHub).

Once a BytePS job has been successfully launched, the speed metrics (e.g., images/sec) will be printed in stdout/stderr. Note that the metrics indicate the speed of a single GPU, which means we need to multiply it by the number of GPUs to get the total speed. If we are running NLP models, the metrics are typically "steps/sec". We can manually convert it to "Tokens/sec" using this equation: Tokens/sec = steps/sec * batch_size * max_seq_len * num_gpu.

The following procedures is illustrated using the same setup as used in the paper (e.g., 8 GPU machines in Expr1). If readers do not have that much available resources, they can scale down (e.g., using 4 GPU machines with 4 additional CPU machines at most for Expr1) and should observe similar results. To run with RDMA with high performance, we need to `export DMLC_ENABLE_RDMA=1` and `export BYTEPS_ENABLE_IPC=1` for all processes.

A.6.1 Reproduce Expr1

- First, `export BYTEPS_ENABLE_MIXED_MODE=1` for all processes.
- Each GPU machine should be allocated with a worker process and a server process. Meanwhile, each CPU machine should be allocated with a server process. The scheduler process can be launched on any machine. The launch script is the "run.sh" of each model. Besides, users need to manually specify the path of the datasets.
- To reproduce Fig. 12(a), make sure the GPU machines do not have NVLinks. Then set `BYTEPS_PCIE_SWITCH_SIZE=4`, which indicates that there are 4 GPUs under the same PCIe switch.
- To reproduce Fig. 12(b), the GPU machines need to have NVLinks. For the topology like Fig. 7, we set `export BYTEPS_REDUCE_ROOTS=2,3`, which indicates that we need to reduce all tensors to GPU-2 and GPU-3 as shown in Fig. 7, so as to avoid the contention with the NIC's traffic.
- Change the number of CPU machines used to see the performance change.

A.6.2 Reproduce Expr2

- We do not need additional CPU machines in this experiment. So just allocate a worker and a server process on each GPU machine.
- To reproduce Fig. 13(a), make sure the GPU machines do not have NVLinks. Then set `BYTEPS_PCIE_SWITCH_SIZE` to 8 and 4, which represents the strawman and optimal solutions, respectively.
- To reproduce Fig. 13(b), the GPU machines need to have NVLinks. Change the value of `BYTEPS_REDUCE_ROOTS` to (1) "0,1,2,3,4,5,6,7" for "root=all" bar, (2) "0" for "root=0" bar, (3) "2,3" for "root=2,3" bar, and (4) "2" for "root=2" bar.

A.6.3 Reproduce Expr3

- The scalability experiments use NVLink-based machines. So we need `export BYTEPS_REDUCE_ROOTS=2,3` to enable the optimal intra-machine communication.
- To reproduce the performance of "BytePS w/o CPU machines", no additional CPU machine is required. It needs at most 32 GPU machines (each with 8 GPUs). On each GPU machine, we need to allocate a worker and a server process.
- To reproduce the performance of "BytePS", we need at most 32 additional CPU machines besides the GPU machines used. The number of GPU machines and CPU machines should be identical. We allocate a worker and a server process on each GPU machine, and only allocate a server process on each CPU machine.
- Here is an example showing how to interpret and get the final speed metrics. After we have launched the ResNet-50 example in Fig. 14(a) with 256 GPUs, we find the stdout of worker-0 shows that GPU-0 has the speed "351 images/sec", then the total speed is 351 * 256 = 89856 images/sec.