Horovod-MXNet Integration

Note: Please feel free to comment on the Google Doc. We will merge the finalized proposal back here.

https://docs.google.com/document/d/1va79O0Kym0ehyav3hyji8uDwxnHJhJtnKvI8rl0X1OA/edit#

Problem

The existing parameter server approach to distributed MXNet faces limitations in performance and feature completeness (tensor fusion, single-bit gradient compression and ability to use MPI and NCCL).

Horovod is an open-source distributed training framework that has shown 2x speedup compared to distributed TensorFlow using innovative techniques [1, 2].

We propose to add Horovod support to MXNet. This will help our users achieve goal of linear scalability to 256 GPUs and beyond. Naturally, we will support multi-machine CPU training too.

Value Proposition

This project is seeking to provide an alternative distributed training solution for MXNet customers. It offers customers the following value proposition:

1. **Usability** - Users do not have to experiment with number of workers and number of servers to get best performance out-of-the-box.
2. **Performance** - Horovod + Tensorflow has shown 2x performance of Distributed Tensorflow [1], so we expect it to show similar gains.
3. **Cost savings** - Parameter servers are not needed when they use Horovod.
4. **Simplified architecture** - Leverage battle-tested libraries such as MPI and NCCL, as well as network optimizations such as RDMA.
5. **Profiler** - Horovod has an excellent profiler for finding bottlenecks.
6. **Online learning** - Due to its MPI paradigm, Horovod can save checkpoints which enables online learning and fine-tuning of your model. With parameter server, it takes some additional work to save Optimizer state located on servers, but with Horovod this feature comes for free. Note: this feature is not currently supported.
7. **Community** - Horovod is a way for MXNet to leverage the Deep Learning community for advancements in distributed training, and for increasing MXNet's visibility.

Proposed Approach

User Interface

To run the training script, user needs to call:

$ mpirun -np 8 --hostfile ~/hosts --bind-to none --map-by slot -x NCCL_DEBUG=INFO -x NCCL_MIN_NRINGS=4 -x LD_LIBRARY_PATH -x PATH -x MXNET_USE_OPERATOR_TUNING=0 -mca pml ob1 -mca btl ^openib python3 /home/ubuntu/master/3rdparty/horovod/examples/mxnet_imagenet_resnet50.py --benchmark 1 --batch-size=256 --network resnet-v1 --num-layers=50 --num-epochs 1 --kv-store None --dtype float16 --gpus 0

Note: This call is valid when using OpenMPI. To improve user experience, this training script may be wrapped into a Bash script in the future.

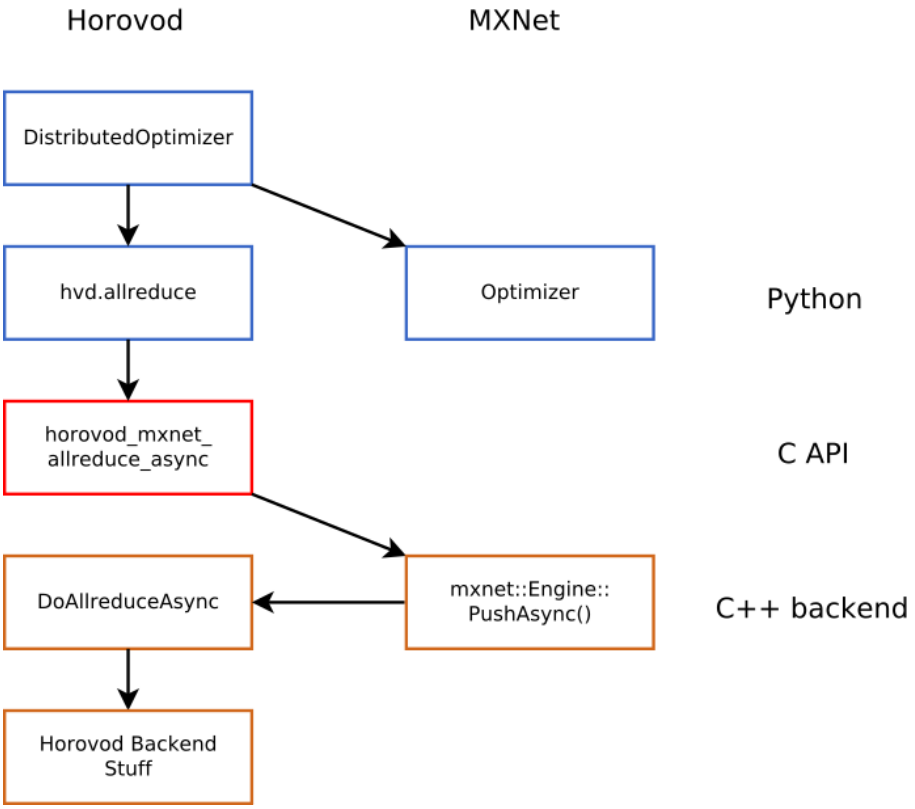
Supported API Calls

Instead of KVStore calls, this API supports the following calls at the Python interface level (for now, in the future we may add other language bindings) after importing the Horovod package (i.e. import horovod as hvd):

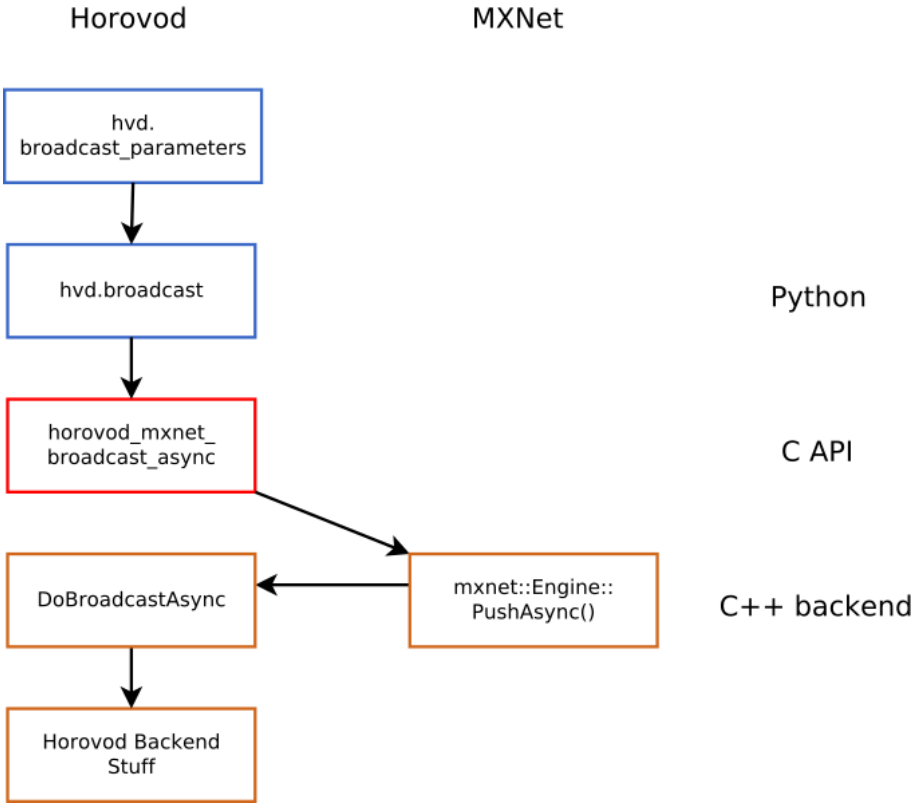
- hvd.allreduce
- hvd.broadcast_parameters
- hvd.local_rank
- hvd.rank
- hvd.local_size
- hvd.size

- hvd.DistributedOptimizer

The following are the two key calls from the Horovod API warrant additional explanation:



(a) hvd.allreduce



(b) hvd.broadcast_parameters

Figure 1. How two key Horovod operations are implemented using Horovod API

Going through how Allreduce works, the DistributedOptimizer is used to wrap an MXNet Optimizer class:

1. In every iteration the DistributedOptimizer wrapper will insert an Allreduce of the gradients before the weight update is done.
2. This is done by calling hvd.allreduce.
3. This calls down into the C API horovod_mxnet_allreduce_async
4. This calls MXNet's PushAsync, which creates a callback for Horovod to call upon completion of the Allreduce.
5. After the Allreduce is complete, Optimizer's weight update is done.

Horovod Interaction with MXNet

The way we plan accomplish decoupling between Horovod and MXNet while ensuring the MXNet engine correctly recognizes dependencies on the NDArrays that are being Allreduce'd together is by calling MXNet engine from Horovod source code. This introduces a compile-time dependency on MXNet when compiling Horovod.

This behaviour is the same as when doing pip install horovod for TensorFlow and PyTorch support. When those libraries are not present, Horovod installation will fail.

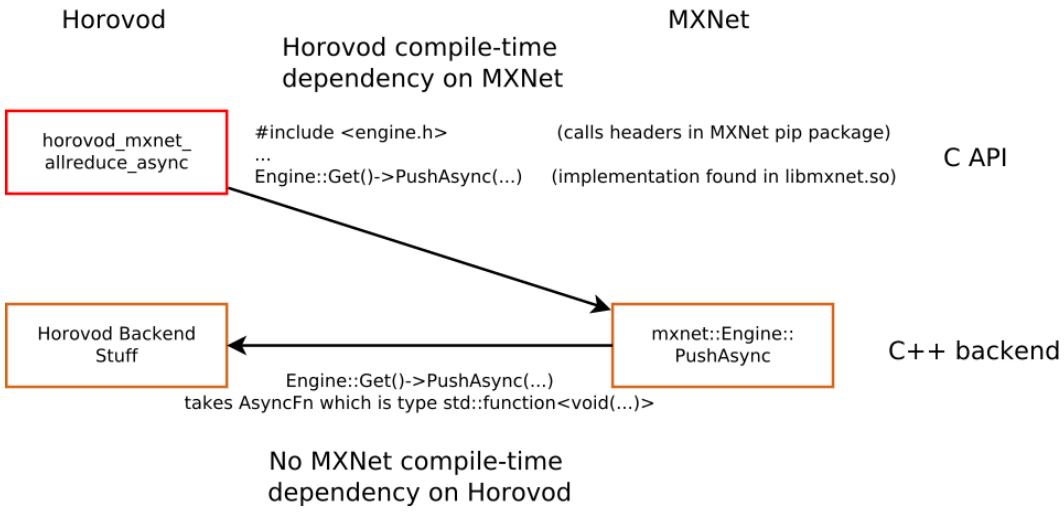


Figure 2. How Horovod interacts with MXNet engine

Code Changes to Horovod Repository

We need to:

1. We will need to modify "setup.py" to install Horovod by linking to MXNet shared library "libmxnet.so"
2. Add "mxnet_imagenet_resnet50.py" and "mxnet_mnist.py" scripts to "horovod/examples". These scripts must expose the Optimizer object.
3. We need to add a folder called "horovod/mxnet" parallel to "horovod/pytorch" and "horovod/tensorflow" that will:
    1. wrap the NDArray objects
    2. wrap the mxnet.Optimizer object

This will allow the source code that supports Horovod core functionalities (tensor fusion, gradient compression, MPI and NCCL support) to be reused for MXNet.

Code Changes to MXNet Repository

We need to:

1. Introduce the new C API method MXWaitforHorovodAllreduce and MXWaitforHorovodBroadcast
2. Make a folder called "include" that will be located in the same folder as the MXNet pip package. This will allow Horovod to call into MXNet for both the NDArray operations and the MXWaitForHorovod call that tells the MXNet engine which NDArrays must be locked for the duration of the Allreduce and Broadcast.

Alternative Approaches (Prototype Design, KVStore API)

We built another approach whose served the following purposes (Note: this prototype design will not be released):

- Proof of concept that MXNet can be made to work with Horovod

- Code reuse (~90% of the code from the prototype will go into the final design)
- Fast prototyping possible due to sidestepping challenges such as:
    - Compilation of Horovod pip package separate from MXNet, which requires some arcane mechanisms such as CFFI
    - Building DistributedOptimizer class that wraps Optimizer class in MXNet

Our prototype uses the KVStore API to call Horovod backend. We expose a new KVStore class that can be selected by the user.

Supported API Calls

In addition to the standard KVStore calls, this API supports the following calls at the Python interface level:

- kv.pushpull
- kv.broadcast
- kv.local_rank
- kv.num_local_workers

Shown below is the kv.broadcast method. The other method are called using a similar manner and are thus not shown:

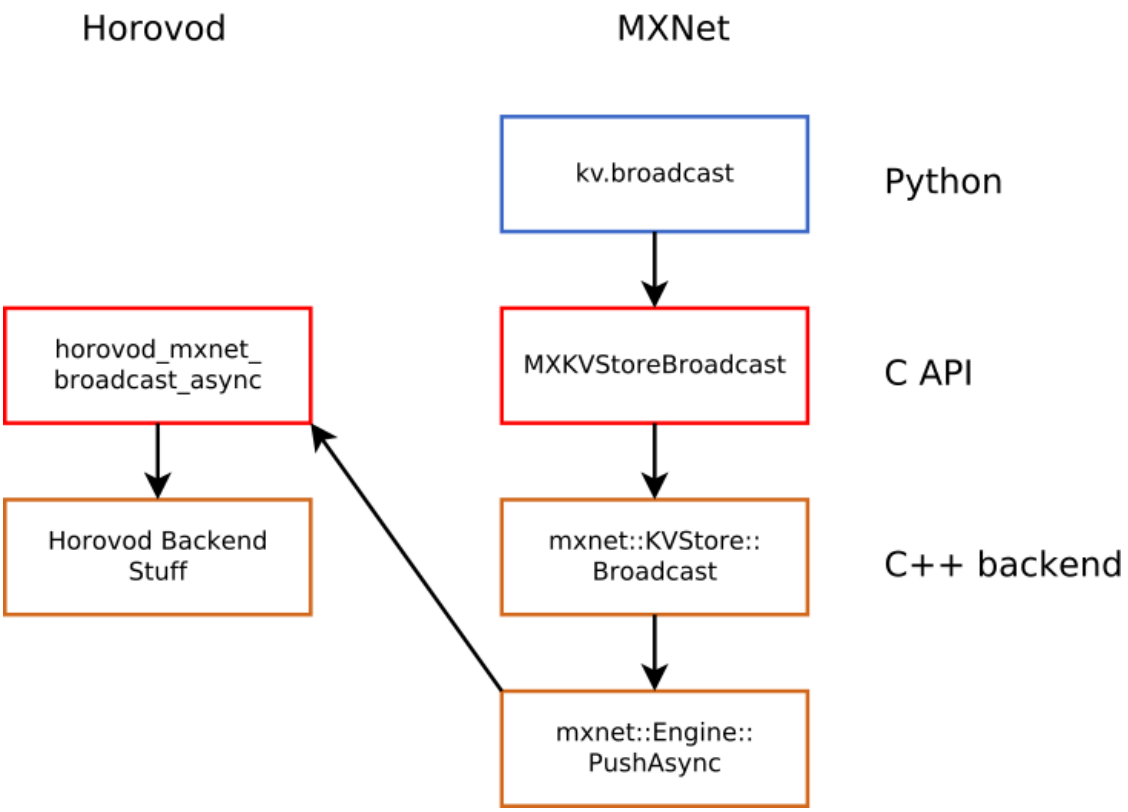


Figure 3. How kv.broadcast is implemented using KVStore API

Pros and Cons of Final Design vs. Prototype Design

Pros

- Easier integration with Horovod repository, because it shares a common API with TensorFlow, PyTorch and Keras
- User does not have to recompile MXNet every time they change Horovod versions, or recompile Horovod every time they change MXNet versions
- Users who are familiar with Horovod API can jump right in to training with MXNet
- With the prototype design, we are creating a kvstore_horovod object, but it is not really a "KVStore". The naming is a misnomer and may confuse some users.

Cons

- Easier integration with other MXNet bindings, because those bindings already support KVStore
- User does not have to install another dependency in order to do distributed training, because MXNet build tool includes Horovod source code as a 3rd party dependency.
    - However, there is a trade-off, because then the Horovod source code would need to be maintained to ensure there are no regressions
- Language bindings for languages other than Python are available without additional work

Performance Benchmarks

Final API Benchmarks

Model: resnet-v1, 50 layers

Dataset: synthetic

Dtype: float32

Instance types: Horovod+X (32 p3.16xlarge), parameter server (32 p3.16xlarge)

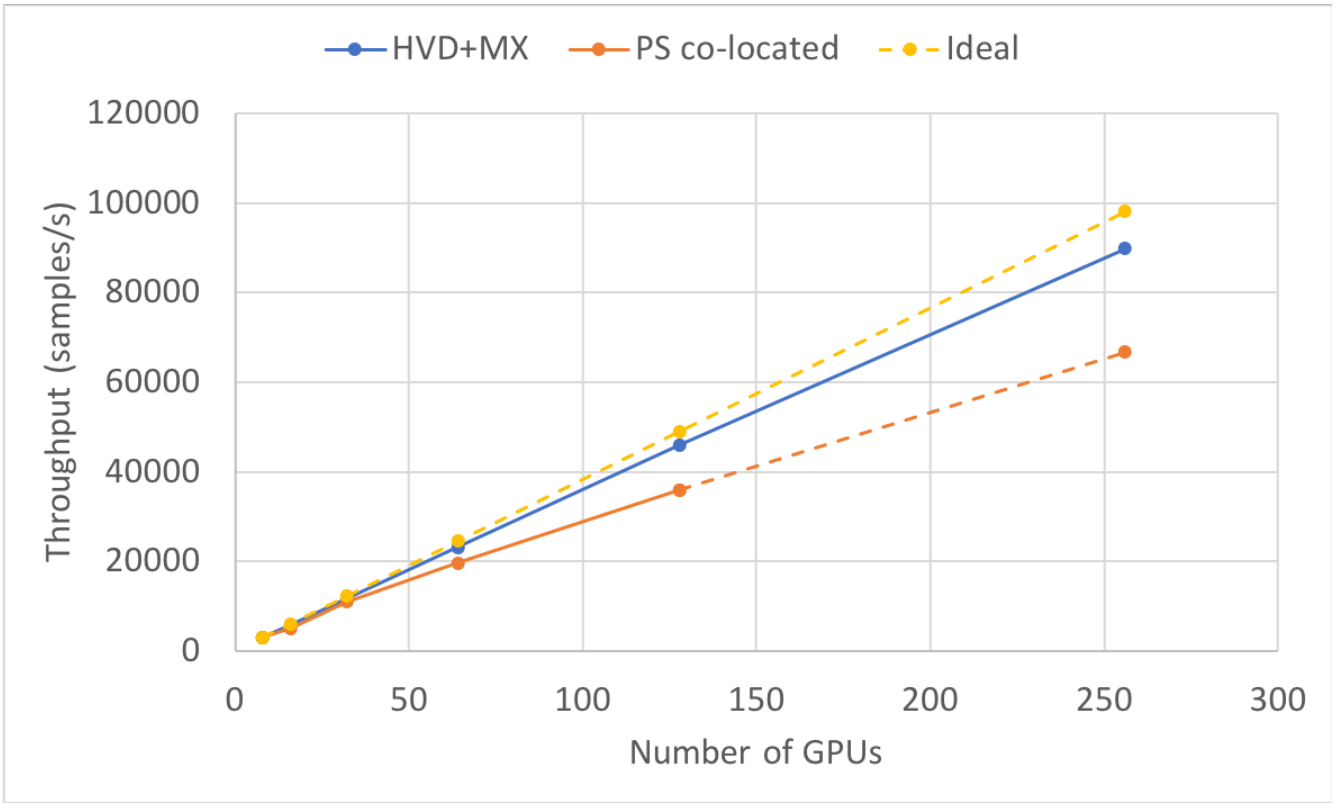


Figure 4. Preliminary benchmark on synthetic data comparing parameter server co-located (servers on same node as workers) and Horovod+MXNet

Prototype Benchmarks

Model: resnet-v1, 50 layers

Dataset: synthetic

Dtype: float32

Instance types: Horovod+X (16 p3.16xlarge), parameter server (16 p3.16xlarge, 32 r4.16xlarge).

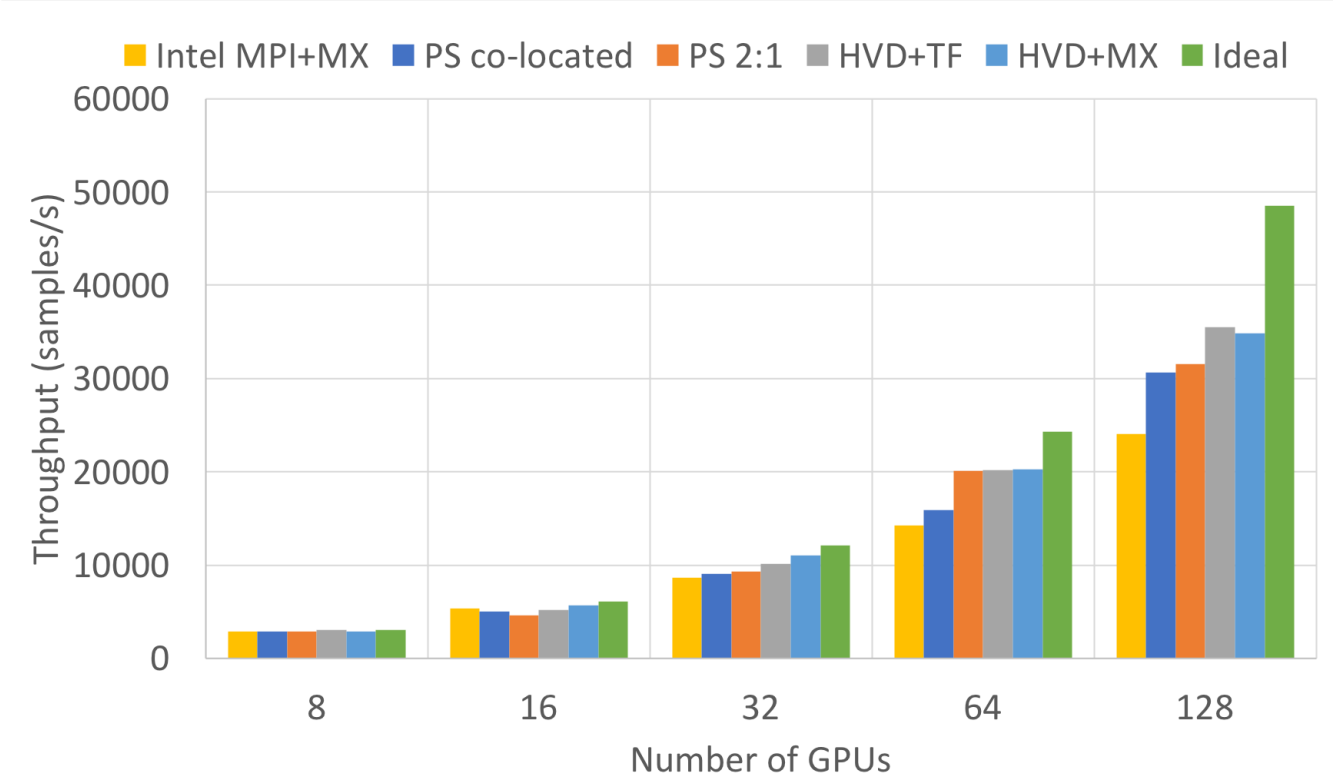


Figure 5. Preliminary benchmark on synthetic data comparing parameter server co-located (servers on same node as workers), parameter server 2 servers:1 worker, Intel MPI+MXNet, Horovod+Tensorflow, and Horovod+MXNet.

CPU support and GPU fp16 support

Since CPU support and GPU fp16 support are listed as experimental at the moment, we do not have performance numbers for them.

Test Plan

Functionality Tests

We will be introducing unit tests for most public Horovod functions at the Python level:

- hvd.allreduce
- hvd.broadcast_parameters
- hvd.local_rank
- hvd.rank
- hvd.local_size
- hvd.size

These will be contained under the path "horovod/test/test_mxnet.py", next to "test_tensorflow.py" and "test_torch.py". To run the test:

$ mpirun -np 8 --hostfile ~/hosts --bind-to none --map-by slot -x NCCL_DEBUG=INFO -x NCCL_MIN_NRINGS=4 -x LD_LIBRARY_PATH -x PATH -x MXNET_USE_OPERATOR_TUNING=0 -mca pml ob1 -mca btl ^openib test_mxnet.py

Performance Tests

Automated performance tests will be outside the scope of this project. For example, the Horovod repository itself does not have any performance tests. They provide a pointer to the repo from https://github.com/tensorflow/benchmarks, and say that the user will be able to replicate the performance numbers shown in the paper [1].

Technical Challenges

MXNet not intended to be used in 1 process/1 GPU mode

Unlike PyTorch and TensorFlow, MXNet is not designed to be used in this manner. Typical MXNet usage involves controlling all GPUs on a single-machine from 1 master process. Therefore, if one calls MXNet from 8 different processes, the memory usage will look as follows:

blocked URL

Figure 5. Memory usage of MXNet when running 8 GPUs in 1 GPU/process mode.

This means that if we want to run MXNet at the optimal batch size for performance, which requires each GPU to be near its memory limit, all those extra memory allocations will quickly cause an out-of-memory error in GPU0. We tracked down the issue to CPUPinned always allocating memory to GPU0 even if the user requests it to be allocated to a different GPU. We are still working on our PR fix here: https://github.com/apache/incubator-mxnet/pull/12031

For a better, longterm solution, it may be necessary to introduce a mechanism that uses CUDA_VISIBLE_DEVICES macro to only make given GPUs visible to each process.

Linking to MXNet shared library

Since we are linking with the MXNet shared library, we need to include the correct headers in the PyPi package. In order to avoid ABI compatibility issues, we may need to add additional APIs (e.g. mx.config.get_compile_flags or mx.config.get_link_flags) that return the compilation and linker flags respectively. Then, the Horovod installation can proceed using the exact same flags.

Gluon support

Gluon support can be added by:

1. making sure to pass the DistributedOptimizer object into Trainer instead of into Module
2. using hvd.broadcast_parameters on the exposed initialized parameters

Milestones

Aug. 10, 2018: Prototype API available for testing: https://github.com/ctcyang/horovod/tree/mxnet_fp16_divide_before_sum/examples/mxnet

- GPU fp32 support has been tested
- GPU fp16 support is still experimental
- CPU support is experimental

Oct. 5, 2018: Beta release of Final API

References

[1] Sergeev, Alexander, and Mike Del Balso. "Horovod: fast and easy distributed deep learning in TensorFlow." arXiv preprint arXiv:1802.05799 (2018). https://arxiv.org/pdf/1802.05799.pdf

[2] Distributed training framework for TensorFlow, Keras, and PyTorch. https://github.com/uber/horovod