Install and Configure Distributed MXNet

Hongyi Wang

April 6, 2017

Abstract

This tutorial mainly focus on how to install MXNet with GPU configuration enabled. And how to launch distributed ML deep net training clusters on MXNet frame.

1 Introduction

MXNet is a deep learning framework designed for both efficiency and flexibility. It allows you to mix symbolic and imperative programming to maximize efficiency and productivity. At its core, MXNet contains a dynamic dependency scheduler that automatically parallelizes both symbolic and imperative operations on the fly. A graph optimization layer on top of that makes symbolic execution fast and memory efficient. MXNet is portable and lightweight, scaling effectively to multiple GPUs and multiple machines.
MXNet is also more than a deep learning project. It is also a collection of blue prints and guidelines for building deep learning systems, and interesting insights of DL systems for hackers.

2 Install MXNet with GPU enabled on AWS EC2

To setup MXNet with GPU enabled, you will need following dependencies:

- C++ compiler with C++ 11 supported, e.g. gcc >= 4.8
- GUDA toolkit with version higher than 7.0 and cuDNN
- BLAS for CPU linear algebra
- openCV for image preprocessing and augmentations
- curl and openSSL for ability to read/write to Amazon S3

For CUDA and cuDNN installation you may want to check this tutorial:
https://github.com/BVLC/caffe/wiki/Install-Caffe-on-EC2-from-scratch-(Ubuntu,-CUDA-7,-cuDNN-3)
For installation of other dependencies, you can refer to:
http://mxnet.io/how_to/cloud.html#set-up-an-ec2-gpu-instance

Once all dependencies are installed, one can build MXNet from source (i.e. fetch the GitHub repo, and do make yourself) following the foregoing guide.
Note: Please make sure you do the following line during make the MXNet on your machine since this command make sure you can run distributed training with MXNet frame

```
echo "USE_DIST_KVSTORE = 1" >> config.mk
```

It is not complete with the foregoing installation instruction provided by MXNet, which is you still need to build python package yourself after building the MXNet workspace. For building python packages, you can simply run commands in following:

```
cd python
```

```
python setup.py install
```

If you encounter no error, you will need to use MXNet with CPU enabled on AWS EC2 instances, to make sure everything is correct, you can run a simple test:

```
INPUT: a = mx.nd.ones((100, 100), mx.gpu(0))
       a

OUTPUT: <NDArray 100x100 @gpu(0)>
```

If you can see output like this, you should be fine to run any GPU enabled examples provided by MXNet:
https://github.com/dmlc/mxnet/tree/master/example

I already have all the fore-mentioned packages installed and tested, and created a new AMI for this, which helps you using GPU enabled MXNet with simply launching an EC2 instance from that AMI:
AMI ID: ami-a1198fc1
AMI Name: MXNet with GPU
AMI Description: installed MXNet with GPU and OpenCV enabled

3 Launch distributed MXNet tasks on AWS EC2

Basically, you can find the official tutorial from MXNet for doing this:
http://mxnet.io/how_to/multi_devices.html#distributed-training-with-multiple-machines
However, this one is really not quite helpful. So, let's see how to do that in detail.
To enable distributed train or build a distributed training cluster on MXNet, the key step is to set:
kv-store = **dist_sync** or **kv-store** = **dist_async** (which represent distributed synchronous training and distributed asynchronous training respectively).
We first need to launch $n + 1$ instances on EC2, in which n represents number of workers in our cluster. Here I use the script provided by TensorFlow to launch and setup NFS on EC2 instances, which can be found at:
https://github.com/hwang595/distributed-MXNet/blob/master/tools/tf_ec2.py
actually, I need to do some changes in this script to make it more suitable for distributed MXNet.
After changing some parameters based on your application, you can just run:

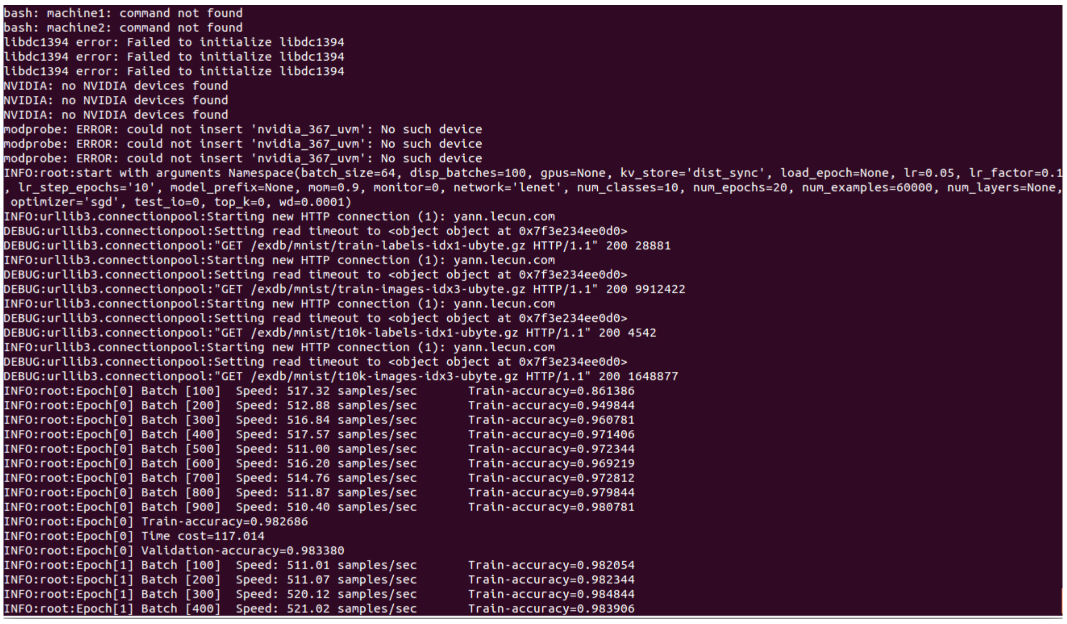
```
python tf_ec2.py launch
```

And please make sure to run

```
python tf_ec2.py shutdown
```

after a training task is done. After launching the $n + 1$ EC2 instances, you will need to ssh to one of them, then this one will be the **host machine**, which is kind of the "parameter server" in TensorFlow, but I'm not totally sure the difference between these two types of machines.
After ssh-ing to the host machine, you will need to do the following steps:

- launch all workers from the host machine:
  `ssh -i your_own_ec2_key.pem usr_name@worker_ip`
- scp your own EC2 key file to each machine:
  `scp -i your_dir:your_own_ec2_key.pem usr_name@worker_ip:remote_dir`
- generate authentication key one each machine by doing:
  `ssh-keygen in /home/.ssh directory`
- write all each generated authentication key files into **etc/hosts** files of each machine (include all workers and host machine)
  `cat id_rsa.pub >> worker_ip:/home/.ssh/authorized_keys`



```
bash: machine1: command not found
bash: machine2: command not found
libdc1394 error: Failed to initialize libdc1394
libdc1394 error: Failed to initialize libdc1394
libdc1394 error: Failed to initialize libdc1394
NVIDIA: no NVIDIA devices found
NVIDIA: no NVIDIA devices found
NVIDIA: no NVIDIA devices found
modprobe: ERROR: could not insert 'nvidia_367_uvm': No such device
modprobe: ERROR: could not insert 'nvidia_367_uvm': No such device
modprobe: ERROR: could not insert 'nvidia_367_uvm': No such device
INFO:root:start with arguments Namespace(batch_size=64, disp_batches=100, gpus=None, kv_store='dist_sync', load_epoch=None, lr=0.05, lr_factor=0.1
, lr_step_epochs='10', model_prefix=None, mom=0.9, monitor=0, network='lenet', num_classes=10, num_epochs=20, num_examples=60000, num_layers=None,
 optimizer='sgd', test_io=0, top_k=0, wd=0.0001)
INFO:urllib3.connectionpool:Starting new HTTP connection (1): yann.lecun.com
DEBUG:urllib3.connectionpool:Setting read timeout to <object object at 0x7f3e234ee0d0>
DEBUG:urllib3.connectionpool:"GET /exdb/mnist/train-labels-idx1-ubyte.gz HTTP/1.1" 200 28881
INFO:urllib3.connectionpool:Starting new HTTP connection (1): yann.lecun.com
DEBUG:urllib3.connectionpool:Setting read timeout to <object object at 0x7f3e234ee0d0>
DEBUG:urllib3.connectionpool:"GET /exdb/mnist/train-images-idx3-ubyte.gz HTTP/1.1" 200 9912422
INFO:urllib3.connectionpool:Starting new HTTP connection (1): yann.lecun.com
DEBUG:urllib3.connectionpool:Setting read timeout to <object object at 0x7f3e234ee0d0>
DEBUG:urllib3.connectionpool:"GET /exdb/mnist/t10k-labels-idx1-ubyte.gz HTTP/1.1" 200 4542
INFO:urllib3.connectionpool:Starting new HTTP connection (1): yann.lecun.com
DEBUG:urllib3.connectionpool:Setting read timeout to <object object at 0x7f3e234ee0d0>
DEBUG:urllib3.connectionpool:"GET /exdb/mnist/t10k-images-idx3-ubyte.gz HTTP/1.1" 200 1648877
INFO:root:Epoch[0] Batch [100]  Speed: 517.32 samples/sec       Train-accuracy=0.861386
INFO:root:Epoch[0] Batch [200]  Speed: 512.88 samples/sec       Train-accuracy=0.949844
INFO:root:Epoch[0] Batch [300]  Speed: 516.84 samples/sec       Train-accuracy=0.960781
INFO:root:Epoch[0] Batch [400]  Speed: 517.57 samples/sec       Train-accuracy=0.971406
INFO:root:Epoch[0] Batch [500]  Speed: 511.00 samples/sec       Train-accuracy=0.972344
INFO:root:Epoch[0] Batch [600]  Speed: 516.20 samples/sec       Train-accuracy=0.969219
INFO:root:Epoch[0] Batch [700]  Speed: 514.76 samples/sec       Train-accuracy=0.972812
INFO:root:Epoch[0] Batch [800]  Speed: 511.87 samples/sec       Train-accuracy=0.979844
INFO:root:Epoch[0] Batch [900]  Speed: 510.40 samples/sec       Train-accuracy=0.980781
INFO:root:Epoch[0] Train-accuracy=0.982686
INFO:root:Epoch[0] Time cost=117.014
INFO:root:Epoch[0] Validation-accuracy=0.983380
INFO:root:Epoch[1] Batch [100]  Speed: 511.01 samples/sec       Train-accuracy=0.982054
INFO:root:Epoch[1] Batch [200]  Speed: 511.07 samples/sec       Train-accuracy=0.982344
INFO:root:Epoch[1] Batch [300]  Speed: 520.12 samples/sec       Train-accuracy=0.984844
INFO:root:Epoch[1] Batch [400]  Speed: 521.02 samples/sec       Train-accuracy=0.983906
```

Figure 1: train logging

- copy the **host** file to /**home** directory.

Till now, we are almost good to go, for next step, we can just run:
python ../../tools/launch.py -n 2 –launcher ssh -H hosts python train_mnist.py –network lenet –kv-store dist_sync
The launch.py file can be found at:
https://github.com/dmlc/mxnet/blob/master/tools/launch.py
And n represents number of workers in the current cluster.
Once the task is begin, you can see something like following:

4 For next step

- The launching method now is really time and effort costing, we will need a more flexible and effective launch script here for launching large number of workers.
- Figure out how parameter server and workers work in MXNet, I'm not sure if the host machine can be just use as parameter server, because they have their own parameter server project:
  https://github.com/dmlc/ps-lite/tree/acdb698fa3bb80929ef83bb37c705f025e119b82
- It seems, we will still need to implement the soft kill idea in MXNet, but hopefully it will work.