Poseidon Documentation

Release 0.10

Adam Schwab, Hong Wu

Jan 18, 2018

Contents

Poseidon is an easy-to-use and efficient system architecture for large-scale deep learning.

This distribution of Poseidon uses the Tensorflow 1.0.1 client API. Poseidon can distribute python model scripts designed to run on a single node into a cluster with no changes to the scripts themselves. This allows for rapid prototyping of model scripts. Developers can focus on perfecting their models running on a single node and be reasonably confident that the model will run efficiently scaled to 32 nodes or more.

CHAPTER 1

Introduction

Poseidon allows deep learning applications written in popular languages and tested on single GPU nodes to easily scale onto a cluster environment with high performance, correctness, and low resource usage. This release has two packages, one for cpu-only machines and the other for machines with gpus.

Traditionally, distributing deep learning jobs has been difficult for two reasons. First of all, taking a model and parallelizing it has been a manual process that had to be done uniquely for every new deep learning model. Secondly, even if the technical challenge of parallelization can be achieved, speed-ups are not guaranteed because dataflow can bottleneck in many ways. Poseidon is built on top of state of the art research aimed at effectively distributing deep learning and it can automatically distribute most deep learning tasks to provide faster total throughput with no manual intervention.

CHAPTER 2

Contents

2.1 Installation Instructions

2.1.1 Supported Operating Systems

- AWS Ubuntu Server 14.04
- AWS Ubuntu Server 16.04
- AWS CentOS 7.x

2.1.2 Pre-Install steps

There are two Poseidon packages. One is for CPU-only devices, the other runs on GPU devices.

For GPU devices, you must install the nVidia CUDA libraries. The install instructions for CUDA and cuDNN are outlined below:

2.1.2.1 CUDA Installation

TensorFlow uses the Nvidia library CUDA and the machine learning patch cuDNN to run certain computation-heavy operations on GPUs. Thus, in order to launch a Poseidon job using your GPU you must first install CUDA and cuDNN.

CUDA

Refer to this link to download and install the CUDA toolkit. Download the version 8.0 deb/rpm local package and then follow the installation instructions listed underneath the download link. The default installation will place the toolkit into `/usr/local/cuda-8.0` and will create a symbolic folder at `/usr/local/cuda`.

cuDNN

Download cuDNN v5.1 from here. Choose the appropriate library download for your operating system. The following bash instructions will uncompress and copy the cuDNN files into the toolkit directory. Assuming the toolkit is installed in `/usr/local/cuda`, run the following commands (edited to reflect the cuDNN version you downloaded):

```
tar -xvf cudnn-8.0-linux-x64-v5.1-ga.tgz
sudo cp -P cuda/include/cudnn.h /usr/local/cuda/include
sudo cp -P cuda/lib64/libcudnn* /usr/local/cuda/lib64
sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/libcudnn*
```

Finally, run the following command to refresh the shared object cache:

```
sudo ldconfig
```

2.1.2.2 Check NVIDIA Drivers

Sometimes GPU device drivers are incorrectly configured. To make sure your GPU drivers are installed properly, run `nvidia-smi` on the terminal. A box should show up with the installed GPUs and some stats such as temperature, etc. If this fails, you will need to update to the latest drivers.

If your operating system is CentOS 7.2, you can refer to the Troubleshooting section of the tutorial for advice.

2.1.3 Installation Options

Poseidon currently supports two ways of installing:

- Debian packaging system (`apt_get`) for clean Ubuntu systems that will not have conflicting dependencies.
- Pip installation: install directly into an existing python2.7 using pip

The debian packaging system is an easier process, and is preferred when installing many machines in a cluster environment. However, if you have a CentOS machine or a customized python setup (such as `virtualenv`) it would be advisable to use the pip installation method instead.

2.1.3.1 Ubuntu/Debian Installation

If working with GPU machines, don't forget to install CUDA and cuDNN before installation.

Install

Currently quick installation is only possible with Ubuntu through the debian packaging system.

```
# CPU-only package:
wget -O poseidon-ubuntu1404_1.0.1_amd64.deb https://github.com/petuum-inc/storage/
↪blob/master/poseidon/deb/ubuntu/cpu/poseidon-ubuntu1404_1.0.1_amd64.deb?raw=true
sudo dpkg -i poseidon-ubuntu1404_1.0.1_amd64.deb

# GPU-enabled package:
wget -O poseidon-gpu-ubuntu1404_1.0.1_amd64.deb https://github.com/petuum-inc/storage/
↪blob/master/poseidon/deb/ubuntu/gpu/poseidon-gpu-ubuntu1404_1.0.1_amd64.deb?raw=true
sudo dpkg -i poseidon-gpu-ubuntu1404_1.0.1_amd64.deb
```

Uninstall

Uninstalling with the debian packaging manager simply means removing the package, called `poseidon` or `poseidon-gpu`:

```
# CPU-only package:
sudo apt-get remove poseidon

# GPU-enabled package:
sudo apt-get remove poseidon-gpu
```

2.1.3.2 Pip Installation

If working with GPU machines, don't forget to install CUDA and cuDNN. More details can be found here. Once this is complete, follow these steps.

Setup

Installation using pip will work on many Unix machines. We support Ubuntu and CentOS (7.x).

First, install `pip` as well as some OS specific core dependencies:

```
# Redhat/CentOS 7.x 64-bit
sudo yum install wget
sudo wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
sudo rpm -ivh epel-release-latest-7.noarch.rpm
sudo yum makecache
sudo yum install python-pip libffi-devel python-devel openssl-devel

# Ubuntu 14.04 and 16.04 64-bit
sudo apt-get update
sudo apt-get install python-pip libssl-dev python-dev libffi-dev python-paramiko
```

Virtualenv (Optional)

Use virtualenv if you wish to install poseidon and tensorflow to a clean python environment.

Note: if you already have tensorflow installed on the machine, this is highly recommended.

The following will install virtualenv and set up a virtual python environment under `~/sandbox`:

```
sudo pip install virtualenv
mkdir ~/sandbox
cd ~/sandbox
virtualenv .
. bin/activate
```

Once `activate` has been run, virtualenv sets `~/sandbox/bin` as the first path for bash. Now `which python` and `which pip` should point to this directory, and anything installed using `pip` will go here.

Note: to uninstall virtualenv, all you have to do is remove the sandbox.

```
deactivate # If you are using a bash that has been 'activate'd
rm -r ~/sandbox
```

Install

Note: if you are not using virtualenv, you may need to `sudo` the following instructions.

```
# CPU-only package:
pip install https://github.com/petuum-inc/storage/raw/master/poseidon/wheel/linux/cpu/
↪poseidon-1.0.1-cp27-cp27mu-linux_x86_64.whl

# GPU-enabled package:
pip install https://github.com/petuum-inc/storage/raw/master/poseidon/wheel/linux/gpu/
↪poseidon_gpu-1.0.1-cp27-cp27mu-linux_x86_64.whl
```

Uninstall

To uninstall, run the following command:

```
# CPU-only package:
pip uninstall poseidon

# GPU-enabled package:
pip uninstall poseidon-gpu
```

2.1.4 Running on a Cluster

Running Poseidon in a cluster environment is simple and is outlined in the next section. Beforehand, install Poseidon using the steps above for each node in your cluster.

For installation, see the installation instructions.

2.2 Poseidon Tutorial

Poseidon is designed to be able to run on a variety of different clusters. This tutorial contains two cluster examples: AWS and a private cluster.

In this tutorial, we will use CIFAR-10, which is a common benchmark in machine learning for image recognition using convolutional neural networks (CNN). The following url provides a description of TensorFlow's implementation: https://www.tensorflow.org/versions/r1.0/tutorials/deep_cnn/.

2.2.1 Download the Model Script

The model we will use for Cifar10 is created by the TensorFlow team. Clone the git repository for each node you will use for Poseidon. Since Poseidon is currently set up for v1.0.1, we must also roll back the model repository (the repository isn't versioned along with TensorFlow).

```
git clone https://github.com/tensorflow/models.git
git reset --hard 4364390adbf16b57c093a05217897831f48da7d3
```

For reference later, save the model directory that you just cloned into `$TF_MODEL_HOME` for reference later:

```
export TF_MODEL_HOME="`pwd`/models"
```

The script is designed to download Cifar10 automatically. To test everything is installed correctly (and download the data), we can run cifar10_train.py directly: `python $TF_MODEL_HOME/tutorials/image/cifar10/cifar10_train.py`

2.2.2 Data

Once you have tested cifar10_train.py with the above call, the data should have downloaded to `/tmp/cifar10_data`. Copy this data onto each node you wish to run Poseidon on to skip the download step during Poseidon execution.

2.2.3 Launch Poseidon

The first tutorial covers configurations specific to Amazon Web Service EC2 service. The second describes configurations when running on a local network.

2.2.3.1 Poseidon on AWS Cluster

Training

The executable for running Poseidon tasks is `psd_run`. Running `psd_run -h` should print this:

```
usage: psd_run [-h] [-c,--cluster_config CLUSTER_CONFIG]
               [-o,--out OUTPUT_FOLDER]
               cmd

psd_run runs Poseidon for distributed machine learning on a GPU cluster. The␣
↪following are its command line arguments.

positional arguments:
  cmd

optional arguments:
  -h, --help            show this help message and exit
  -c,--cluster_config CLUSTER_CONFIG
                        configuration file for cluster environment: specify workers␣
↪in each machine. See example:
                        {
                          "virtualenv": "/home/ubuntu/sandbox",
                          "username": "ubuntu",
                          "pem_file": "/path/to/pem-file.pem",
                          "master_node": "localhost",
                          "worker_nodes": [
                            "192.168.1.11",
                            "192.168.1.15"
                          ],
                          "server_nodes": [
                            "192.168.1.70",
                            "192.168.1.80"
                          ]
                        }
  -o,--out OUTPUT_FOLDER
                        output log folder
```

Setup

We must create a `config.json` to specify our cluster configurations. If running a single node on Ubuntu within an AWS instance (with no virtualenv), the configurations are very simple:

Note: replace `cluster-key.pem` with a path to your AWS pem file.

```
{
  "pem_file": "cluster-key.pem",
  "worker_nodes": [
    "127.0.0.1"
  ],
  "server_nodes": [
    "127.0.0.1"
  ]
}
```

Note: if running on multiple AWS nodes, add each node's private IP in the worker_nodes and server_nodes lists within config.json.

Execution

We can now launch Poseidon with the following command:

```
psd_run -c config.json -o logs "python $TF_MODEL_HOME/tutorials/image/cifar10/cifar10_
↪train.py --max_steps 1000"
```

Poseidon Logs

After running Poseidon, you can check the execution log `poseidon_run.log` in the same path you ran `psd_run`. There are also output log files for debugging and monitoring purposes created in `poseidon_log_$TIMESTAMP_SUFFIX` folder which will reside in a `logs` folder in your current directory (the -o directive).

2.2.3.2 Poseidon on a Private Cluster

Training

The executable for running Poseidon tasks is `psd_run`. Running `psd_run -h` should print this:

```
usage: psd_run [-h] [-c,--cluster_config CLUSTER_CONFIG]
               [-o,--out OUTPUT_FOLDER]
               cmd

psd_run runs Poseidon for distributed machine learning on a GPU cluster. The␣
↪following are its command line arguments.

positional arguments:
  cmd

optional arguments:
  -h, --help            show this help message and exit
  -c,--cluster_config CLUSTER_CONFIG
```

```
                        configuration file for cluster environment: specify workers␣
↪in each machine. See example:
                        {
                          "virtualenv": "/home/ubuntu/sandbox",
                          "username": "ubuntu",
                          "pem_file": "/path/to/pem-file.pem",
                          "master_node": "localhost",
                          "worker_nodes": [
                            "192.168.1.11",
                            "192.168.1.15"
                          ],
                          "server_nodes": [
                            "192.168.1.70",
                            "192.168.1.80"
                          ]
                        }
  -o,--out OUTPUT_FOLDER
                        output log folder
```

Setup

Say we wish to run Poseidon on two nodes, IP1 and IP2. We must create a `config.json` to specify our configurations. The runner `psd_run` uses ssh to communicate with the cluster, so certain options must be added, such as username. If you wish to use a virtualenv, you can specify it using the json as well. The path should correspond to $VIRTUAL_ENV environment variable (after virtualenv `activate` script has been run).

Note below that the virtualenv keyword is optional. Remove if you installed without virtualenv.

```
{
  "virtualenv": "/path/to/virtualenv",
  "username": "<cluster username>",
  "worker_nodes": [
    "<IP1>",
    "<IP2>"
  ],
  "server_nodes": [
    "<IP1>",
    "<IP2>"
  ]
}
```

For security reasons, the script does not allow passwords in ssh. Therefore, no-password ssh must be enabled for the username. On <IP1> (our master node), generate a key-pair. Then copy it onto <IP2>:

```
ssh-keygen
# Use defaults (press ENTER three times)

ssh-copy-id -i ~/.ssh/id_rsa.pub <cluster username>@<IP1>
ssh-copy-id -i ~/.ssh/id_rsa.pub <cluster username>@<IP2>
# Enter password

# Test, ssh should not prompt for a password
ssh <cluster username>@<IP2>
exit
```

Execution

We can now launch Poseidon with the following command:

```
psd_run -c config.json -o logs "python $TF_MODEL_HOME/tutorials/image/cifar10/cifar10_
↪train.py --max_steps 1000"
```

Poseidon Logs

After running Poseidon, you can check the execution log `poseidon_run.log` in the same path you ran `psd_run`. There are also output log files for debugging and monitoring purposes created in `poseidon_log_$TIMESTAMP_SUFFIX` folder which will reside in a `logs` folder in your current directory (the -o directive).

2.2.4 Evaluating

Poseidon's evaluating procedure is the same as TensorFlow's. Run the evaluation script:

```
python $TF_MODEL_HOME/tutorials/image/cifar10/cifar10_eval.py
```

2.3 Reference

2.3.1 Command Line Options for psd_run

psd_run options "<TensorFlow command>"

Options:

- -c (–cluster-config): path to the json configuration file. More details below.
- -o OUTPUT_FOLDER (–out=OUTPUT_FOLDER): the folder base path where log files will be saved (on each node).

Advanced usage:

- If you want specify different entry commands for different workers, you can generate a commands string splits with `,`. Please make sure the number of commands in the last argument is consistent with the number of workers. For example, if you want to specify two different data partitions for the two workers, you can run as: `psd_run -c conf.json "python model.py --dataset=/dat/partition1, python model.py --dataset=/dat/partition2"`.

2.3.2 JSON Cluster Config

Within the cluster configuration json there are two required fields and several optional ones.

Required:

- worker_nodes - required list of IP addresses to run the tensorflow script files.
- server_nodes - required list of IP addresses for parameter servers.

Note: The number of worker and server nodes currently must be equal.

Optional:

- master_node - IP of the master, by default this is set to be local machine
- pem_file - If using aws, use this to point to the pem file required for ssh auth
- virtualenv - If using python within a virtualenv, point to the virtualenv root directory (equivalent to echo $VIRTUAL_ENV).
- username - Set global username for all processes/communication. The username should be set for ssh no-password authentication on all machines. The default is ubuntu.

2.4 Troubleshooting

2.4.1 No Nvidia GPU device drivers found for CentOS 7.2 (AWS)

You can check if your drivers are installed correctly by calling `nvidia-smi` via the terminal. If the drivers are installed, this should result with a box showing GPU temperature, processes, etc. If it returns in error, then you will need to install (or update) your Nvidia drivers.

Try the below command. Advisable only on fresh instance, it updates the kernel to a newer version that is supported under Nvidia device drivers.

```
sudo rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org
sudo rpm -Uvh http://www.elrepo.org/elrepo-release-7.0-2.el7.elrepo.noarch.rpm
sudo yum install kmod-nvidia
```

Finally, reboot.

Try `nvidia-smi` to verify the installation was successful.

2.4.2 Program Hangs After session.run() call (AWS p2.xlarge)

In ConfigProto (the object we pass to a new session object), set the following options:

```
config = ConfigProto()
...

config.num_push_threads = 1
config.num_pull_threads = 1
...

sess = tf.Session(config = config)
...
```

CHAPTER 3

Performance at a Glance

Poseidon can scale almost linearly in total throughput with additional machines while simultaneously incurring little additional overhead.

CHAPTER 4

Contact

- Adam Schwab - adam.schwab@petuum.com
- Hong Wu - hong.wu@petuum.com
- Hao Zhang - hao.zhang@petuum.com