Poseidon Documentation

Release 0.10

Adam Schwab, Hong Wu

May 19, 2017

Contents

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

This distribution of Poseidon uses the Tensorflow 0.10 client API.

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 runs the TensorFlow 0.10 api on distributed GPU clusters - greatly improving convenience, efficiency and scalability over the standard opensource TensorFlow software.

CHAPTER 2

Contents

Installation Instructions

Supported Operating Systems

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

Pre-Install steps

Poseidon runs on GPU clusters. The following steps outline the installation of GPU libraries CUDA and cuDNN:

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 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
```

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.

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.

Ubuntu/Debian Installation

Don't forget to install CUDA and cuDNN. Once this is complete, follow these steps.

Install

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

This creates an `apt-get` repository on your system with a few dependencies in it as local `deb` files. The final step is to install the core Poseidon project, and it will automatically install all the dependencies.

```
wget -O poseidon-repo-ubuntu1404_0.10_amd64.deb https://github.com/petuum-inc/storage/
↪blob/master/poseidon/deb/ubuntu/poseidon-repo-ubuntu1404_0.10_amd64.deb?raw=true
sudo dpkg -i poseidon-repo-ubuntu1404_0.10_amd64.deb
sudo apt-get update
sudo apt-get install poseidon
```

Uninstall

Uninstalling with the debian packaging manager simply means removing the core package, called `poseidon`:

```
sudo apt-get remove poseidon

# Note you can also remove the debian install repository:
sudo apt-get purge poseidon-repo
```

Pip Installation

Don't forget to install CUDA and cuDNN. 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
```

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.

```
pip install --upgrade setuptools==30.1.0 protobuf==3.1.0 numpy paramiko

pip install https://github.com/petuum-inc/storage/blob/master/poseidon/wheel/linux/
↪gpu/poseidon-0.10.0-cp27-none-linux_x86_64.whl?raw=true
```

Uninstall

To uninstall, run the following command:

```
pip uninstall poseidon
```

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.

Poseidon Tutorial

Poseidon is designed to be able to run on a variety of different clusters. Below we have two examples: AWS and a private cluster.

For installation, see the installation instructions.

Quick Tutorial: AWS Cluster

This is a quick tutorial to run a distributed Poseidon task on an AWS 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). More detailed instructions on how to get started are available at: https://www.tensorflow.org/versions/r0.10/tutorials/deep_cnn/.

Data

The dataset will download automatically when you run the training code with the default options. You can also put your dataset in a distributed file system.

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:

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

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

Execution

We can now launch Posiedon with the following command. The script, `cifar10_train.py` is an example model script included with the Poseidon installation.

```
# The model is in the Poseidon install directory. This line gets the Poseidon home.
POSEIDON_HOME=`python -c 'import os; import tensorflow; print os.path.
↪dirname(tensorflow.__file__)'`

psd_run -c config.json "python $POSEIDON_HOME/models/image/cifar10/cifar10_train.py --
↪max_steps 1000"
```

Note that the above script for cifarNet is included in the Poseidon release. If you wish to view the model, it is located in `$POSEIDON_HOME/models/image/cifar10`.

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 purpose created in `poseidon_log_$TIMESTAMP_SUFFIX` folder.

Evaluating

Poseidon's evaluating procedure is the same as TensorFlow's. Please follow the tutorial here.

For installation, see the installation instructions.

Quick Tutorial: Private Cluster

This is a quick tutorial to run a distributed Poseidon task on 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). More detailed instructions on how to get started are available at: https://www.tensorflow.org/versions/r0.10/tutorials/deep_cnn/.

Data

The dataset will download automatically when you run the training code with the default options. You can also put your dataset in a distributed file system.

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 Posiedon with the following command. The script, `cifar10_train.py` is an example model script included with the Poseidon installation.

```
# The model is in the Poseidon install directory. This line gets the Poseidon home.
POSEIDON_HOME=`python -c 'import os; import tensorflow; print os.path.
↪dirname(tensorflow.__file__)'`
```

```
psd_run -c config.json "python $POSEIDON_HOME/models/image/cifar10/cifar10_train.py --
↪max_steps 1000"
```

Note that the above script for cifarNet is included in the Poseidon release. If you wish to view the model, it is located in `$POSEIDON_HOME/models/image/cifar10`.

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 purpose created in `poseidon_log_$TIMESTAMP_SUFFIX` folder.

Evaluating

Poseidon's evaluating procedure is the same as TensorFlow's. Please follow the tutorial here.

In order to run python scripts using Poseidon, slight adjustments must be made. The page below walks throught an example TensorFlow script and shows how it can be altered for Poseidon.

Port TensorFlow Script to Poseidon

Posiedon worker scripts are almost identical to native TensorFlow, but with an extended session initialization object. In order to demonstrate the changes, we have a native TensorFlow script with a corresponding Poseidon script.

Remember that for this release, Poseidon scripts must be compatible with TensorFlow 0.10 API.

Below is an example TensorFlow script, we'll call it `mymodel_train.py`:

```
import tensorflow as tf

FLAGS = tf.app.flags.FLAGS

tf.app.flags.DEFINE_integer('max_steps', 5, 'Loop count')

def model():
  x1 = tf.Variable(1.5, name='x1')
  x2 = tf.Variable(3.5, name='x2')
  out = x1 + 2 * x2
  grads = tf.gradients(ys=[out], xs=[x1, x2])
  return grads

def train():
  model_op = model()
  init = tf.initialize_all_variables()
  sess = tf.Session()
  sess.run(init)
  for step in xrange(FLAGS.max_steps):
    model_out = sess.run(model_op)
    print 'step ' + str(step) + ' ' + str(model_out)

def main(argv=None):
  train()
```

```
if __name__ == '__main__':
  tf.app.run()
```

The above could be executed with the following command:

```
python mymodel_train.py --max_steps 10
```

A modified `mymodel_train.py` for Poseidon will have three changes.

1. Add three commandline arguments - distributed, master_address and client_id:

```
tf.app.flags.DEFINE_boolean('distributed', False, "Use Poseidon")
tf.app.flags.DEFINE_string('master_address', "tcp://0.0.0.0:5555", "master address")
tf.app.flags.DEFINE_integer('client_id', -1, "client id")
```

2. Build a tf.ConfigProto() using these commandline args:

```
config = tf.ConfigProto()
config.master_address = FLAGS.master_address
config.client_id = FLAGS.client_id
config.distributed = FLAGS.distributed
```

3. Call tf.Session with the new config object:

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

Here, if you use other interfaces like `tf.contrib.slim` interface, you can pass this config object as the argument of `slim.learning.train` function.

Putting the three changes together, we get the script below.

```
import tensorflow as tf

FLAGS = tf.app.flags.FLAGS

# Three new command line options
tf.app.flags.DEFINE_boolean('distributed', False, "Use Poseidon")
tf.app.flags.DEFINE_string('master_address', "tcp://0.0.0.0:5555", "master address")
tf.app.flags.DEFINE_integer('client_id', -1, "client id")

tf.app.flags.DEFINE_integer('max_steps', 5, 'Loop count')

def model():
  x1 = tf.Variable(1.5, name='x1')
  x2 = tf.Variable(3.5, name='x2')
  out = x1 + 2 * x2
  grads = tf.gradients(ys=[out], xs=[x1, x2])
  return grads

def train():
  model_op = model()
  init = tf.initialize_all_variables()

  # Add a config variable to pass Poseidon settings
  config = tf.ConfigProto()
  config.master_address = FLAGS.master_address
  config.client_id = FLAGS.client_id
  config.distributed = FLAGS.distributed
```

```
  # Initialize tf.Session with the config
  sess = tf.Session(config = config)
  sess.run(init)
  for step in xrange(FLAGS.max_steps):
    model_out = sess.run(model_op)
    print 'step ' + str(step) + ' ' + str(model_out)

def main(argv=None):
  train()

if __name__ == '__main__':
  tf.app.run()
```

Given a config.json file, Poseidon can be executed using the above script with the following command (remember to change /path/to/mymodel_train.py):

```
psd_run -c config.json -o ~/logs "python /path/to/mymodel_train.py --max_steps 10"
```

Note: `psd_run` adds the following flags to the worker when it launches:

- distributed
- master_address
- client_id

Reference

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).

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.

ConfigProto Posiedon Options

This table demonstrates the Poseidon settings, and `psd_run` defaults.

Arg Name	Set by psd_run	Default
distributed	True	True
master_address	True	master_address in json or localhost
client_id	True	0 -> N-1 (N is number of workers)
num_push_threads	False	1
num_pull_threads	False	1
block_size	False	4 MB
use_sfb	False	False

Troubleshooting

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.

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.

The following figures show Poseidon's performance on four widely adopted neural networks (see the table for their configurations) using distributed GPU clusters. All of these neural networks use the TensorFlow api (0.10), and the benchmarks compare the Poseidon software against the standard TensorFlow engine.

Name	#parameters	Dataset	Batchsize (single node)
Inception-V3	27M	ILSVRC12	32
VGG19	143M	ILSVRC12	32
VGG12-22k	229M	ILSVRC12	32
ResNet-152	60.2M	ILSVRC12	32

For distributed execution, Poseidon consistently delivers near-linear increases in throughput across various models and engines. For example, Poseidon registered a 31.5x speedup on training the Inception-V3 network on 32 nodes, which is a 50% improvement upon the original TensorFlow (20x speedup). When training a 229M parameter network (VGG19-22K), Poseidon still achieves near-linear speedup (30x on 32 nodes), while distributed TensorFlow sometimes experiences negative scaling with additional machines.

CHAPTER 4

Contact

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