スパコンで機械学習（Python 環境構築編）

塙 敏博

東京大学情報基盤センター

１．はじめに

近年，東大情報基盤センターで運用しているスパコンを利用して，Deep Learning を始めとした機械学習を実行しているユーザも増えてきました。

2019 年 7 月現在，Reedbush-U/H/L，Oakforest-PACS，Oakbridge-CX の各システムが稼働しており，いずれのシステムにおいても Python 環境やよく使われるフレームワークについては，あらかじめ module ファイルを用意しています。しかし，機械学習フレームワークを始めとして Python モジュールの更新速度は大変速く，新しいバージョンのソフトウェアをいち早く試したいというユーザの方は，自前で Python 環境を構築，維持したいと考えることでしょう。

そこで本稿では，執筆者の経験を元に，東大情報基盤センターのシステムで使うことを念頭においた Python 環境構築方法について説明します。

２．Anaconda の利用

例えば Reedbush システムでは，2019 年 7 月現在 module ファイルとして 200 種類が用意されています[1]。しかし，導入から既に 3 年が経過して，すっかり古くなってしまった物も多数残されています。

Python についても，OS （RedHat/CentOS 7 系）と一緒にインストールされるもの，Intel がコンパイラ群と共に提供しているもの，Anaconda によるもの，といくつかの種類があらかじめ用意されています。いずれも Python 単体で問題なく使うことはできますが，最新の Python モジュールと組み合わせるには困難が伴います。

そこで，ここでは機械学習やデータ解析のための Python 環境として広く使われている **Anaconda** を利用してスパコンで利用できる環境を構築し，管理していくことを考えます。Anaconda はパッケージの内部でモジュール間の依存関係を解決する仕組みを持っており，複数の仮想環境を切り替えて使う機能もあることから，様々なバージョンを使い分けたいような場合にも容易に扱えます。

Anaconda のインストール，初期設定

注意すべき点は，(1)インストール作業は**ログインノード**で全て済ませる必要があること，(2)その際に計算ノードで実行できるディレクトリにインストールすること，です。例えば，Oakforest-PACS や Oakbridge-CX では，**/work** 以下，Reedbush では **/lustre** 以下のディレクトリしか計算ノードは参照できないので，そこにインストールする必要があります。ここでは，Reedbush を例に，ユーザグループ xx12，ユーザ名 y12345 として，Anaconda の一式を /lustre/xx12/y12345/conda にインストールすることにします。（以下プロンプト，例えば

[1] 各システムにおける module コマンドの利用はそれぞれ「利用の手引き」をご覧ください。

[y12345@reedbush-u1 ~]$ を $ と表すことにします。)

```
$ cd  /lustre/xx12/y12345
$ wget  https://repo.anaconda.com/archive/Anaconda3-2019.03-Linux-x86_64.sh
$ chmod  +x  Anaconda3-2019.03-Linux-x86_64.sh
$ ./Anaconda3-2019.03-Linux-x86_64.sh  -p  /lustre/xx12/y12345/anaconda
```

ライセンスやインストール先など確認されますので，承認するとインストールが進みます。最後に，

```
Do you wish the installer to initialize Anaconda3
by running conda init? [yes|no]
```

と聞かれるので，yes と答えてください。すると、~/.bashrc が更新されるので、/lustre の下にコピーしておく必要があります(計算ノードがこちらしか参照できないためです)。

```
$ cp  ~/.bashrc  /lustre/xx12/y12345/
```

その後一旦ログアウトしてからログインし直してください。すると今度はプロンプトの前に(base)がついて表示されます。これで Anaconda が使える状態です。その後 Anaconda 全体を更新する必要があります。以下を実行します。

```
(base)$ conda  update  conda
(base)$ conda  remove  conda-build
(base)$ conda  update  --all
```

確認が求められたら y を選んでください。これで準備は終わりです。

Anaconda の仮想環境

次に Anaconda の仮想環境を作ります。これによって,Python の特定バージョンを指定したり，特定のプロジェクト向けのビルド＝**チャネル**を指定するなどした環境を作ることができます。複数環境で同一のファイルであればハードリンクを使うなど，使われるディスク容量を極力抑えるような工夫もされています。

ここでは，MPI による分散学習に対応した Chainer 6.1.0 の構築を例に考えていくことにします。まず chainermn という名前の環境を作ることにします。この際、Intel の MKL (Math Kernel Library)などの最適化されたライブラリを利用できるよう、Intel のチャネルを指定することをお勧めします(特に GPU のない Reedbush-U, Oakforest-PACS, Oakbridge-CX では重要です)。また Intel が用意しているメタパッケージ intelpython3_core を指定すると，利用頻度の高いパッケージが自動的にインストールされます。(intelpython3_full もありますが，数も多いので時間もかかります。)

```
(base)$ conda create -n chainermn -c intel intelpython3_core
```

パッケージのインストールが済んだら以下のように activate で環境を切り替えます。環境が切り替わると環境の一覧は info に --envs をつけて実行します。base に戻る際は，conda deactivate でも構いません。

```
(base)$ conda activate chainermn
(chainermn)$ conda info --envs
# conda environments:
#
base                     /lustre/xx12/y12345/anaconda
chainermn             *  /lustre/xx12/y12345/anaconda/envs/chainermn
```

Anaconda と PyPI 等の混在

他に Anaconda の Intel チャネルで用意されているようなパッケージは，conda install コマンドでインストールできます。そのパッケージ単体で不足するものは，自動的に依存関係を調べて一緒にインストールしてくれます。

その一方で，更新が速く Anaconda のパッケージになっていないものや，環境に大きく依存するようなモジュールは，PyPI (Python Package Index)によってビルド&インストールした方がいい場合もあります。例えば，コンパイラのバージョンに依存したり MPI などのライブラリに依存するようなモジュール(cupy，mpi4py)などです。また，これらのビルドの際には，利用するコンパイラ，MPI 環境などをあらかじめ module load し，pip install に--no-cache-dir オプションをつけるのを忘れないようにしてください。

ソースコードをダウンロードして python setup.py install によってインストールするような場合でも混在は可能です(ただし，インストール先の path については，必要に応じて prefix として /lustre/xx12/y12345/anaconda/envs/chainermn/ を指定するなどしてください)。

また，PyPI でインストールしたパッケージやソースコードからビルド&インストールしたモジュールは，Anaconda では依存性が制御できなくなります。同様に，ソースコードからビルド&インストールしたモジュールは PyPI から制御できなくなります(インストール後 PyPI として登録されるものでは問題ありません)。

Reedbush-H，L で MPI 並列版 Chainer 6.1.0 を構築するには，以下のようにすると良いようです[2]。

```
(chainermn)$ module purge
(chainermn)$ module load intel cuda9/9.1.85 openmpi/gdr/2.1.2/intel pbsutils
(chainermn)$ conda install -c intel cython filelock fastrlock typing protobuf=3.7.1
(chainermn)$ pip install --no-cache-dir cupy-cuda91 mpi4py chainer
```

[2] Python のバージョンの整合性や，pip install で入ったパッケージ名を見て conda で入れ直しています。

パッケージが Anaconda によるものか PyPI によるものか，以下のように容易に判別できます。

```
(chainermn)$ conda  list
# Name                    Version                   Build  Channel
_libgcc_mutex             0.1                        main
bzip2                     1.0.6                        17    intel
...
chainer                   6.1.0                    pypi_0    pypi
```

ここで Channel が空欄のものは Anaconda のデフォルトチャネル，intel は Intel が提供している（-c intel を指定した)もの，pypi は pip install によってインストールされたものを示しています。

他のシステム，他のフレームワークについても同様です。

- Chainer 6.1.0 (Oakbridge-CX, OFP 向け)

  $ conda install -c intel cython filelock fastrlock typing protobuf=3.7.1

  $ pip install --no-cache-dir ideep4py mpi4py chainer

- Keras+Tensorflow[3]

  $ conda install tensorflow=1.14.0 keras (OBCX, OFP, RB-U)

  $ conda install cudatoolkit=9.2 tensorflow-gpu=1.13.1 keras (RB-H, L)[4]

- Tensorflow (nightly build)

  $ pip install --no-cache-dir tf-nightly (OBCX, OFP, RB-U)

  $ conda install cudatoolkit=9.2; pip install --no-cache-dir tf-nightly-gpu (RB-H, L)

- PyTorch 1.1.0

  $ conda install -c pytorch pytorch-cpu torchvision-cpu (OBCX, OFP, RB-U)

  $ conda install -c pytorch cudatoolkit=9.0 pytorch=1.1.0 torchvision (RB-H, L)[5]

最後に，Anaconda，PyPI におけるコマンドをまとめておきます（[]は省略可能)。

コマンド	意味
conda create -n 環境 [-c チャネル] パッケージ	指定された名前の環境の作成
conda info [--envs]	環境やインストールパスなどの表示
conda install [-c チャネル] パッケージ[=バージョン等]	現在の環境にパッケージをインストール
conda list	パッケージ一覧表示
conda search パッケージ [-c チャネル] [--	パッケージの検索，バリエーション

[3] Anaconda によるビルドの方がコンパイラの最適化が効いているので，敢えて Intel チャネルを外しています。

[4] CUDA 9.2.148 で動作確認しました。

[5] CUDA 9.1.85 で動作確認しました。

info]	や依存関係を表示
conda remove パッケージ	パッケージの削除
conda remove -n 環境 --all	環境の削除
conda update [-c チャネル] パッケージ or --all	パッケージの更新
pip install --no-cache-dir パッケージ	(PyPI)パッケージのインストール
pip uninstall パッケージ	(PyPI)パッケージの削除
pip search パッケージ	(PyPI)パッケージの検索

３．ジョブの実行

ジョブの実行の際には，.bashrc の指定，module の指定，Anaconda の環境の指定などを全て適切に行う必要があります。
以下に Reedbush-H で Chainer を使って 4 ノード 8 GPU で動作させて MNIST の学習をする例を示します。以下のように Chainer 6.1.0 のソース一式をダウンロードして展開します（詳細は chainer のホームページをご覧ください)。

```
$ wget https://github.com/chainer/chainer/archive/v6.1.0.tar.gz
$ tar xvfz v6.1.0.tar.gz
```

```
#PBS -q h-regular
#PBS -l select=4:mpiprocs=2:ompthreads=1
#PBS -W group_list=xx12
#PBS -l walltime=02:00:00
cd $PBS_O_WORKDIR
source /lustre/xx12/y12345/.bashrc
conda activate chainermn
export CHAINER_DATASET_ROOT=/lustre/app/acc/cuda9/chainermn/1.0.0/dataset
module purge
module load intel cuda9/9.1.85 openmpi/gdr/2.1.2/intel
mpirun python chainer-6.1.0/examples/chainermn/mnist/train_mnist.py ¥
-g --communicator pure_nccl |& tee mnist.log
```

図 1: MNIST training のジョブスクリプトファイル例

４．インストール済み Anaconda の利用

特定のモジュールだけ使えればいい場合や，新しいモジュールへの依存が少ない場合にはインストール済みの Anaconda で十分用が足りる場合もあります。Chainer 6.1.0 を Reedbush にインストールするには，

```
$ module purge
$ module load intel cuda9/9.1.85 openmpi/gdr/2.1.2/intel pbsutils anaconda3
```

```
$ export PYTHONUSERHOME=/lustre/xx12/y12345/.python
$ mkdir  $PYTHONUSERHOME
$ pip  install --no-cache-dir --user  cupy-cuda91  mpi4py  chainer
```

などとしても問題なくインストールできます。この場合は，ジョブスクリプト中でPYTHONUSERHOME 環境変数も忘れずに指定してください。

５．おわりに

本稿では，最新 Python 環境の構築方法について述べました。しかしながら，Python の各モジュール間には大変複雑な依存関係があります。一方のモジュールのバージョンをアップデートしようとすると，もう片方がダウングレードされてしまったり，モジュールがビルドされた OS が Ubuntu 系で，スパコンにインストールされている RedHat / CentOS では正常に動作しない，など，簡単には解決できないことも起きます。

このような環境の違いを吸収するための仕組みとして，Docker に代表されるコンテナ仮想化技術があります。本センターのスパコンシステムでは，コンテナ仮想化ソフトウェアとして Docker コンテナイメージも扱える Singularity というソフトウェアを導入しています。

次号では，この Singularity を使った実行例についてご紹介する予定です。

参　考　文　献

Anaconda Inc., Anaconda Distribution, https://www.anaconda.com/distribution/

Python Software Foundation, Python Package Index, https://pypi.org

Chainer, https://chainer.org

Tensorflow, https://www.tensorflow.org

PyTorch, https://pytorch.org