Python 環境(2021 年版)

塙 敏博

東京大学情報基盤センター

１．はじめに

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ユーザが増えていることから，2019 年 7 月に「スパコンで機械学習 (Python 環境構築編)」として記事を掲載しました[1]。多くの方に参考にしていただいたようで，お役に立てて幸いです。

ところが，当該記事でベースに用いた Anaconda[2]について，2020 年 4 月に利用規約が変更され，無償利用の条件が厳しくなっております。東大情報基盤センターのシステムでも注意喚起と回避策の概略についてご紹介します。

また，5 月には，東大情報基盤センターに導入された Wisteria/BDEC-01 が導入され，Odyssey ノードでも Aquarius ノードでも Python は利用可能です。ここでは Aquarius ノードでの利用を想定して，miniconda，JupyterHub/JupyterLab，singularity による利用方法を説明します。

２．Anaconda の利用規約

現在，Anaconda の利用規約は 2021 年 5 月 26 日版となっています[3]。利用規約では，無償利用では商用(commercial activity)は許可されないとあり，以下は commercial activity ではない，としています。

- 個人がビジネス以外の個人的な目的で使用する場合
- 教育機関の学生または従業員が教育活動に関連して使用する場合
- 非営利団体の従業員やボランティアが、慈善事業に関連して使用する場合
- 非営利研究機関が、非商業的な活動に関連して使用する場合で、収益につながらない場合
- 総従業員数が 200 人未満の企業にて使用する場合

ここから考えると，当センターのスパコンシステム(JCAHPC の OFP も含む)では，大半のユーザは commercial activity とみなされることになりそうです。

そこで 2021 年度より，Anaconda は取りやめ，Anaconda 互換の Miniconda および conda-forge の利用に切り替えました。

３．Miniconda と conda-forge の利用

Miniconda は，Anaconda でも使われている管理コマンド conda のみを含む最小限のパッケージです[4]。conda-forge とは，*conda で利用可能なパッケージのリポジトリで，Anaconda 社ではなくコミュニティーで管理されているものです[5]。Anaconda 社としても，Miniconda と conda-forge の組み合わせであればライセンスを侵害しない，と回答しています[6]。

４．Python 環境の選択，環境構築

4.1 インストール済みモジュールの利用

当センターのスパコンシステムでは，ユーザ環境の切り替えのために Environment Modules [7]を用いています。Reedbush，Oakforest-PACS，Oakbridge-CX，Wisteria/BDEC-01 全てのシステムで，python を用いるためのモジュールを用意しています。また，Reedbush，Oakbridge-CX，Wisteria/BDEC-01 では，miniconda のモジュールを用意しています。運良く，使いたいバージョンのソフトウェアがモジュールとして存在していれば，すぐに使用することができます。

- モジュールの一覧表示，その後使いたいモジュールをロード（Reedbush の例）

```
$ module avail
…
-------- /lustre/app/modulefiles/cuda10 -------
…   miniconda3/py39_4.9.2  … pytorch/1.8.1 …
…
$ module load pytorch/1.8.1
All modules were reset, cuda10/10.0.130 miniconda3/py39_4.9.2 pbsutils intel/19.4.243
openmpi/3.1.4/intel gnu/gcc_7.5.0 are loaded
```

※ プロンプト，例えば [y12345@reedbush-h1 ~]$ を $ と表す。また，コマンドラインで入力する文字列は下線で表す。以下同様。

- モジュールの一覧表示（Oakforest-PACS，Oakbridge-CX，Wisteria/BDEC-01 で可能）

```
$ show module
ApplicationName   ModuleName                      Node           BaseCompiler/MPI
-------------------------------------------------------------------------------
…
Miniconda         miniconda/py38_4.9.2            aquarius
…
PyTorch+Horovod   pytorch-horovod/1.8.1-0.21.3   aquarius       cuda/11.1
```

4.2 Miniconda の利用

機械学習やデータ解析でよく用いられる Python ですが，前提にしている Python のモジュール・ライブラリのバージョンがツールごとに異なり，依存関係がたちまち成立しなくなるため，一つの環境にまとめることは，ほぼ不可能です。そこで，使用するツールごとに環境を分ける必要が生じてきます。

Python の仮想環境を作って切り替えるツールには，知る限りでは，pyenv，virtualenv，venv，{Ana,mini}conda,最近では pipenv と言ったものが存在しているようです。この中で，conda 系は，BLAS などのライブラリ群が Intel MKL をベースに構成される(pip 系では OpenBLAS がデフォルト)という点や，チャネルを指定して提供元を切り替えられる(intel や pytorch など，conda-forge 以外にもそれぞれチャネルが選べる)という利点があります。また，共用のスパコンで使う上では，ビルド済みバイナリで提供されること(pip ではソースからビルドする物

も多い)，create した環境同士は基本的に干渉せず独立に扱えるなどの点からも，conda 系が扱いやすいと考えています。

2019 年 7 月号に Anaconda による環境構築を執筆しましたが，Miniconda でも同じように実現できます。但し，Miniconda もデフォルトチャネルは anaconda のレポジトリになっていますので，conda-forge に限る必要があります。

```
$ conda config --add channels conda-forge
$ conda config --remove channels defaults
```

この操作をすると，結果的に以下の“.condarc”の channels:の項のようになるはずですのでご確認ください。

さて，今回は，なるべくストレージ容量を節約し，高速に環境を構築することを念頭に，インストール済みの miniconda 環境をローカルに clone（可能な場合にはシンボリックリンク）して使うことを考えてみます。注意すべき点は，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，Wisteria/BDEC-01 では **/data** または**/work** 以下，Oakforest-PACS，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。

Wisteria/BDEC-01 を例に，ユーザグループ gg12，ユーザ名 a12345 として，/work/gg12/a12345/miniconda にインストールすることにします。

準備

システムにインストール済みの Miniconda をモジュールでロードします。

```
$ module load aquarius miniconda/py39_4.9.2
$ conda info
…
base environment : /work/opt/local/x86_64/cores/miniconda/py39_4.9.2  (read only)
…
```

もし，ホームディレクトリに“.conda”，“.condarc”がある場合には，元のファイルをリネームして，/work 領域とシンボリックリンクを張って共通化しておきます。

```
$ mv  ~/.conda  ~/.conda.bak   (あれば)
$ mv  ~/.condarc  ~/.condarc.bak   (あれば)
$ touch  /work/gg12/a12345/.condarc
$ mkdir  /work/gg12/a12345/.conda
$ ln  -s  /work/gg12/a12345/.conda*  ~/
```

“.condarc”は以下のように編集します。もし不具合が起こるようでしたら，ファイル中の *softlinks を削除した後に環境の再構築をお試しください。

```
channels:
  - conda-forge
allow_softlinks: true # 容量を節約したい場合
always_softlink: true # 容量を節約したい場合
# root_prefix: /work/gg12/a12345/miniconda # minicondaをインストールするpath
```

クローン

上記の base environment のパスから，自分のディレクトリにクローンします。

```
$ conda create -p /work/gg12/a12345/miniconda --clone base
$ conda install -p /work/gg12/a12345/miniconda conda
```

conda および全モジュールをアップデートします。

```
$ conda update -p /work/gg12/a12345/miniconda conda
$ conda update -p /work/gg12/a12345/miniconda --all
```

その上で“.condarc”中の root_prefixの前のコメント # を削除し，有効にします。

使い方

以下のようにしてconda環境を有効にします。

```
$ source /work/gg12/a12345/miniconda/etc/profile.d/conda.sh
```

これによってcondaコマンドも使えるようになります。さらにactivateすれば準備完了です。

```
$ conda info --envs
# conda environments:
#
base              *  /work/gg12/a12345/miniconda
$ conda activate
(base) $
```

ここでは，実際に執筆時点で最新版のpytorchをインストールしてみます。

```
(base) $ conda create -n pytorch pytorch=1.10.0=cuda112py39h4e14dd4_0 cudatoolkit=11.2
(base) $ conda activate pytorch
(pytorch) $
```

以下にcondaコマンドの主な使い方をまとめておきます。

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 create -p パス [-c チャネル] -clone 元環境	元環境をクローンしてパスの名前の環境を作成
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] [--info]	パッケージの検索，バリエーションや依存関係を表示
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

５．JupyterHub の利用

Wisteria/BDEC-01システムでは，Webブラウザを用いてPythonを利用可能なJupyterHub環境を提供しています[8]。以下のURLにWebブラウザで接続することで利用できます。サインインの際は，ポータルと同じパスワードを利用してください。なお，Mac環境の方は，Safariの場合にはターミナルで文字が入力できない場合があります。Firefox，Chromeなどをご利用ください。

https://wisteria08.cc.u-tokyo.ac.jp:8000/jupyterhub/

Wisteria/BDEC-01 システムでは，JupyterHub 中の Jupyter Notebook から，pysub コマンドを利用して，Notebook 上に記載されている Python コードの行番号を指定してジョブ投入ができます。

あらかじめ，以下のテンプレートを ~/.notebook/template_job.sh に用意しておきます。

```
#!/bin/sh
#PJM -L rscgrp=share-a
#PJM -L gpu=1
```

また，カレントディレクトリを/work に移動しておきます。

```
[1] !pwd
/home/a12345/.notebook
[2] %cd /work/gg12/a12345
/work/01/gg12/a12345
```

続いて以下のように Python のスクリプトを入力したとします。一旦入力を完了させて行番号が振られたことを確認します。

```
[3] import os
    import numpy as np
    A=np.random.rand(5,5)
    B=np.random.rand(5,5)
    print (A,B)
    C=np.dot(A,B)
    print (C)
```

続いてこの行番号 3 を使って pysub でジョブをサブミットします。その前に一旦左上の save ボタンを押してこの notebook を保存しておきます。この notebook のファイル名が“Untitled.ipynb”だとすると，以下のように実行することができます。

```
[4] !pysub  -f  Untitled.ipynb  3
```

JupyterHub には，ターミナル機能もありますので，“New Launcher”からターミナルを起動して結果を確認することも容易です。ただし，この結果出力を再び Python で使用して解析を行う，といった連携はまだできるようになっていません（実現方法は現在検討中です）。

自分好みの JupyterLab を構成する

JupyterLab では，様々な拡張機能が利用可能です。デフォルトでは，拡張機能が利用できるように見えますが，共有領域に設定を書き込もうとしてしまうため，うまくいきません。ここではそれを回避する方法を説明します。

まず，**4.2** 節で述べた方法で，JupyterLab の環境を構築します。拡張機能に必要なモジュールもあらかじめインストールしておきます。

```
(base) $ conda create -n myjupyter jupyterlab
(base) $ conda activate myjupyter
(myjupyter) $
```

次に，~/.jupyter/jupyter_lab_config.py を作成し，上で構成した myjupyter 環境における JupyterLab のパスをセットします。ファイルの内容は以下の 1 行です。

```
c.LabApp.app_dir = '/work/gg12/a12345/miniconda/envs/myjupyter/share/jupyter/lab'
```

最後に，JupyterLab 上で，“File”→ “Hub Control Panel” → “Stop My Server” → “Start My Server”の順に再起動すると，自分が構築した JupyterLab 環境を使用して起動します。拡張機能のいくつかは，JupyterLab の環境とバージョンが合わずに動作しないものも見受

けられますが，是非ご利用いただき，便利な使い方がありましたらご紹介ください。

注意点としては，ブラウザから切断しただけでは，環境が起動中のまま残ることです。明らかに不要な場合には，“File”→“Log Out”を選んでください。逆に言えば，明示的にログアウトするまでは処理がそのまま継続できます。また，複数のユーザの Jupyter 環境が同じサーバ上で動作しますので，あまり重い処理は実行しないようにお願いいたします。

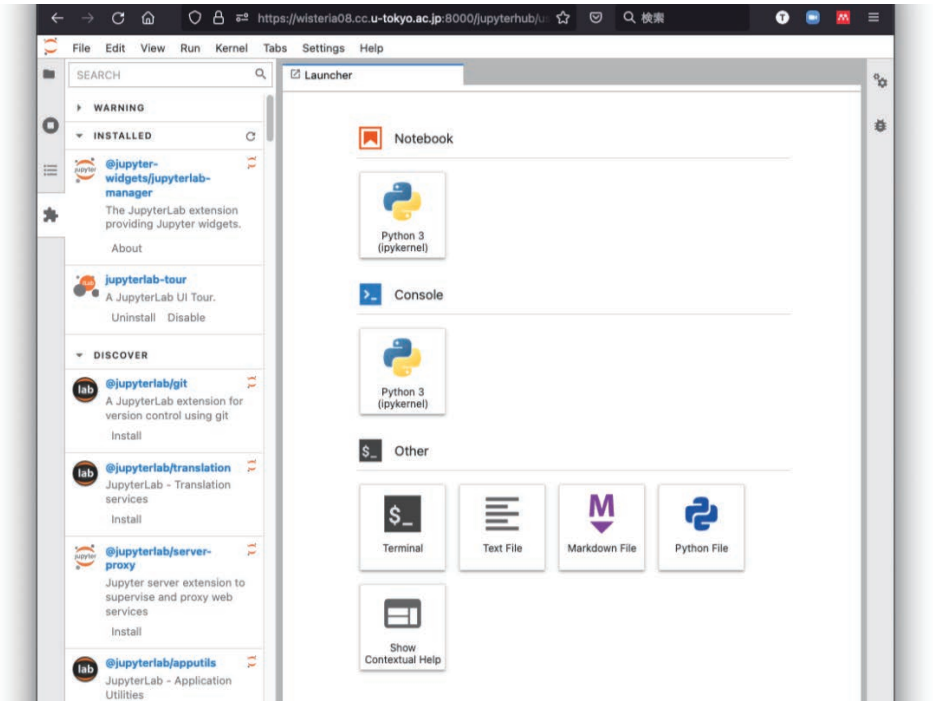

図 1 JupyterLab の画面

６．Singularity の利用

例えば TensorFlow の場合，CUDA ライブラリ等の依存関係の問題で，conda，pip 等で環境構築するのは非常に困難です。このような場合，コンテナ仮想化によって，開発元などが作成したイメージ一式を使い，確実に動く環境を容易に手に入れることができます。

コンテナ仮想化環境として，お手元の PC では，Docker コンテナを利用することが多いと思いますが，スパコンでは Docker ではなく，Singularity というソフトウェアを利用します[9]。

これ以降は，Docker Hub に登録されている Tensorflow コンテナ，具体的には https://hub.docker.com/r/tensorflow/tensorflow の“latest-gpu”タグで登録されているものを，tf-image.file というファイル名でダウンロードする場合を例に説明します。

```
$ module load singularity
$ singularity build tf-image.file docker://tensorflow/tensorflow:latest-gpu
```

（注）ファイルサイズは 3 GB 弱ほどになりますので実行の際はご注意ください。

ダウンロード元としては以下のような既存のコンテナリポジトリの指定が可能です。

- `docker://`　Docker Hub
- `shub://`　Singularity レジストリ

実行する際には，以下のようにします(Wisteria/BDEC-01 の場合です。Singularity のバージョンによってオプションが異なる場合があります)。

```
$ singularity exec --nv --bind $PWD tf-image.file python スクリプト名
```

Singularity の主なコマンドの使い方を以下にまとめておきます。

コマンド	意味
`singularity build [--sandbox] container.img ソース`	コンテナを `singularity` 用に `build` して `container.img` に保存する。`--sandbox` の場合は，`container.img` 以下のディレクトリに展開される。
`singularity shell [--nv] [--writable] [--fakeroot] container.img`	コンテナ環境内で閉じて実行する。`--nv` は `NVIDIA GPU` を利用する場合に用いる。`--writable` はコンテナイメージを書き込み可能にする場合，`--fakeroot` と合わせて使用する。
`singularity exec [--nv] [--writable] [--fakeroot] [--bind dir1[:dir2]] container.img コマンドライン`	コンテナ環境内にあるコマンドを用いてコマンドラインを実行する。`--bind` はコンテナ外の `dir1` をコンテナ内の `dir2` にマップする。`dir2` を省略すると `dir1` がコンテナ内でも使われる。その他のオプションは `shell` の場合と同じ。

Wisteria/BDEC-01 におけるジョブスクリプトは以下のようになります。カレントディレクトリの keras-tf-mnist.py を使う例です。特に jobenv の指定は忘れないようにしてください。

```
#!/bin/bash
#PJM -L rg=share-a
#PJM -L gpu=1
#PJM -g gg12
#PJM -L jobenv=singularity
#PJM -L elapse=00:15:00
#PJM -j
module load aquarius cuda singularity
singularity exec --bind $PWD /work/gg12/a12345/tf-image.file python keras-tf-mnist.py ¥
>& keras-tf-mnist.log.$PJM_JOBID
```

５．おわりに

本稿では，Wisteria/BDEC-01 の環境を中心に，Python 環境の構築方法について，最新の状況を説明しました。

新しい UI として準備した JupyterHub は，スパコンに対するハードルを下げ，利便性を高める環境として我々も期待しています。

また，Singularity は，環境構築のコストを下げる切り札として，スパコンでも広く使われるようになり，NVIDIA によるスパコン向けベンチマークの配布もコンテナイメージを使って行われるようになっています[10]。当センターでは，早くから全システムに Singularity を整備してきました。以前は MPI 環境への対応に難がありましたが，Wisteria/BDEC-01 では特に不都合なく使えています。

Python 環境は(Jupyter も)，アップデートのサイクルが非常に早いので，また大きな変化がありましたらご報告したいと思います。本執筆記事の感想，コメントをお待ちしております。

参 考 文 献

[1] 塙 敏博，スパコンで機械学習(Python 環境構築編)，スーパーコンピューティングニュース，Vol. 21，No. 4，2019 年 7 月
https://www.cc.u-tokyo.ac.jp/public/VOL21/No4/11.201907python.pdf

[2] Anaconda Inc.，Anaconda Installers & Packages，https://repo.anaconda.com

[3] Anaconda Inc.，Terms of Service，https://www.anaconda.com/terms-of-service

[4] Miniconda，https://docs.conda.io/en/latest/miniconda.html

[5] conda-forge，https://conda-forge.org/

[6] Anaconda CEO の回答，
https://www.reddit.com/r/Python/comments/iqsk3y/anaconda_is_not_free_for_commercial_use_anymore/g4xuabr/

[7] Environment Modules，http://modules.sourceforge.net/

[8] Jupyter，JupyterHub，https://jupyter.org/hub

[9] Sylabs.io，Singularity，https://sylabs.io/singularity/

[10] NVIDIA Corp.，NVIDIA HPC Benchmarks，
https://ngc.nvidia.com/catalog/containers/nvidia:hpc-benchmarks