ABCI利用のポイント

国立研究開発法人　産業技術総合研究所
情報・人間工学領域
招聘研究員　萩島功一

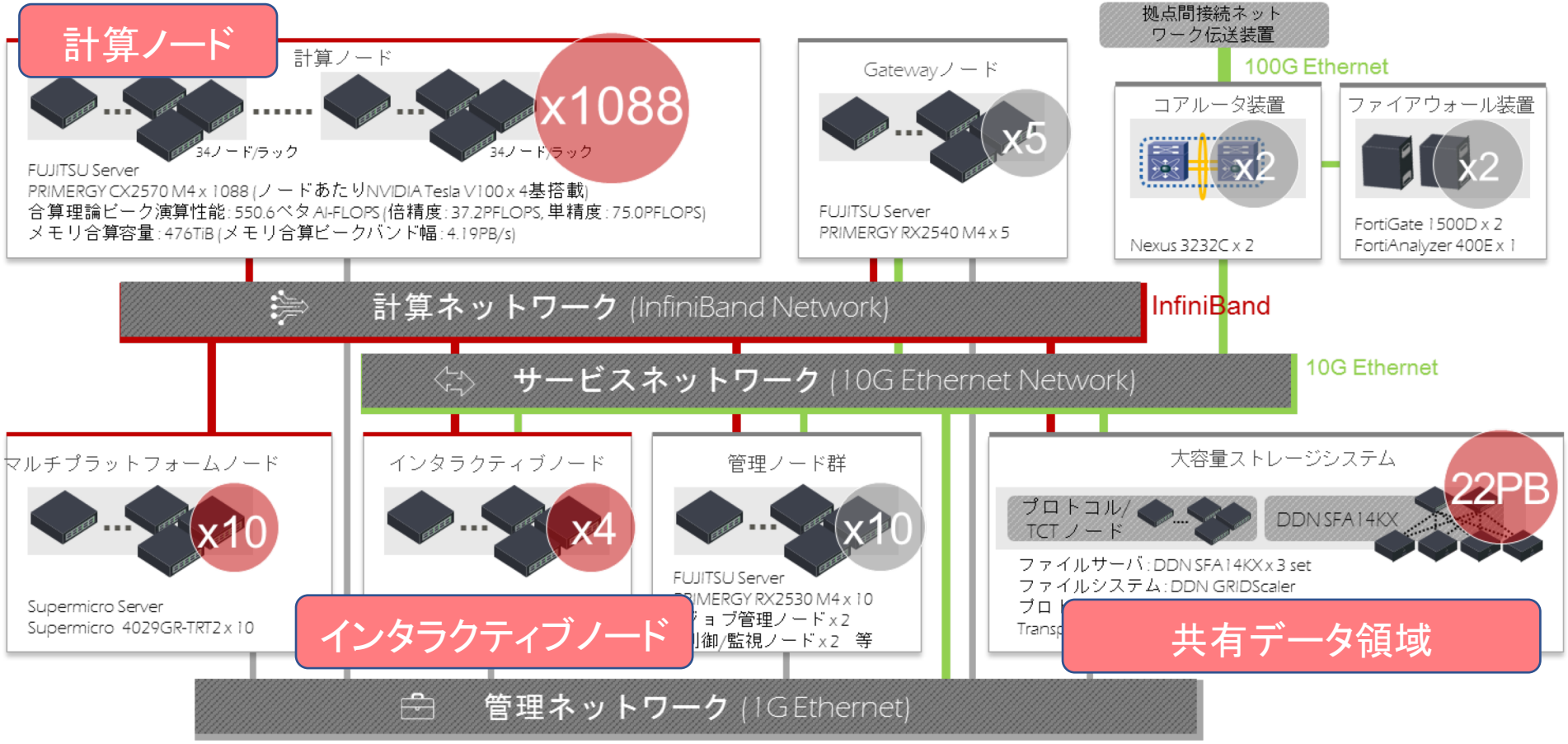
2019年2月27日
2019年3月25日（改定）
2019年3月27日（追加）

目　次

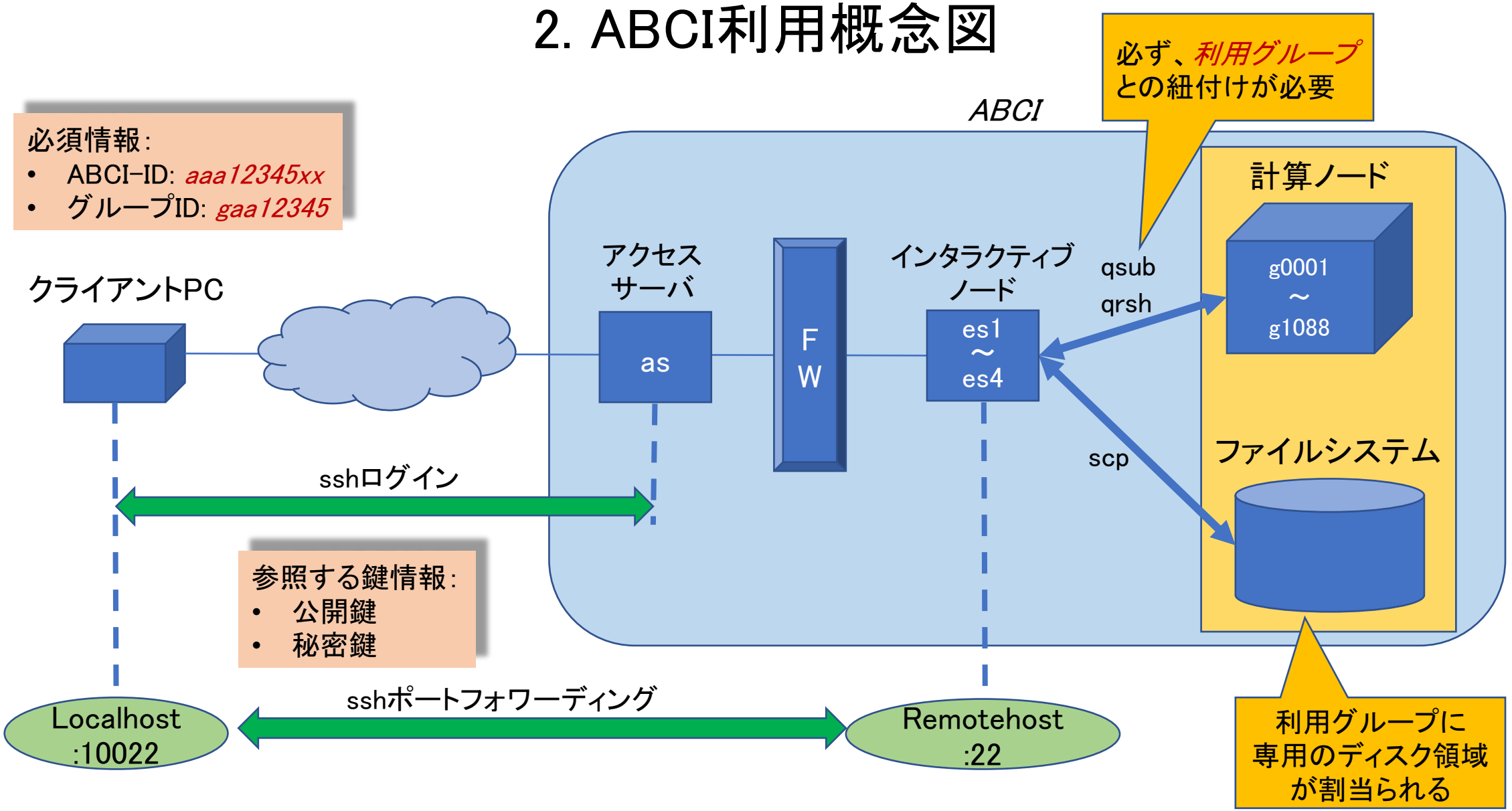
1. ABCIシステムの概要
2. ABCI利用概念図
3. 基本操作
   - ABCIへのログイン(ssh)
   - 計算ノードの利用：ジョブの実行
     - インタラクティブジョブ(qrsh)
     - バッチジョブ(qsub)
   - ファイルのアップロード、ダウンロード(scp)
4. 環境構築
   - Singularityを利用して、TensorFlowの環境を構築
   - qsub（バッチジョブ）での実行例
   - Jupyter Notebookの利用
5. ソフトウェアの利用
6. ABCI料金表
7. 計算ノードの予約利用

1. ABCIシステムの概要

- 1088台の計算ノードと22PBの大容量ストレージを高速ネットワークで接続した高性能計算システム



2. ABCI利用概念図



インタラクティブノード(es)までの2段階認証を行わないと、ABCIリソースを利用できない

3. 基本操作

- ABCIへのログイン(ssh)
- 計算ノードの利用：ジョブの実行
  - インタラクティブジョブ(qrsh)
  - バッチジョブ(qsub)
- ファイルのアップロード、ダウンロード(scp)

ログイン(ssh)

1） ターミナルからアクセスサーバ(*as.abci.ai*)にログインします。

```
yourpc$ ssh -L 10022:es:22 -l aaa12345xx as.abci.ai
The authenticity of host 'as.abci.ai (0.0.0.1)' can't be established.
RSA key fingerprint is XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX. → 初回ログイン時のみ表示される。
Are you sure you want to continue connecting (yes/no)? ← yesを入力
Warning: Permanently added 'XX.XX.XX.XX' (RSA) to the list of known hosts.
Enter passphrase for key '/home/username/.ssh/id_rsa': ← パスフレーズ入力
Welcome to ABCI access server. Please press any key if you disconnect this session.
```

ABCI利用中、
このターミナルは
開いておく！

Warning
上記状態で何らかのキーを入力するとSSH接続が切断されてしますので注意してください。

2） 別のターミナルで、インタラクティブノード(es)にポートフォーワーディングします。

```
yourpc$ ssh -p 10022 -l aaa12345xx localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established. RSA key fingerprint is
XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX. → 初回ログイン時のみ表示される。
Are you sure you want to continue connecting (yes/no)? yes ← yesを入力
Warning: Permanently added 'localhost' (RSA) to the list of known hosts.
Enter passphrase for key '/home/username/.ssh/id_rsa': ← パスフレーズ入力
[aaa12345xx@es1 ~]$
```

ssh -L *10022:es:22* as.abci.ai	localhostのポート番号:remote host:remote hostのポート番号
ssh -p *10022* localhost	localhostのポート番号

計算ノードの利用
ジョブ割当リソース

資源タイプ	資源タイプ名	説明	割り当て物理CPUコア数	割り当てGPU数	メモリ(GiB)	ローカルストレージ(GB)	資源タイプ課金係数	割当可能な個数
Full	rt_F	ノード占有	40	4	360	1440	1.00	1-512 (1-32)*
G.large	rt_G.large	ノード共有GPU利用	20	4	240	720	0.90	1
G.small	rt_G.small	ノード共有GPU利用	5	1	60	180	0.30	1
C.large	rt_C.large	ノード共有CPUのみ利用	20	0	120	720	0.60	1
C.small	rt_C.small	ノード共有CPUのみ利用	5	0	30	180	0.20	1

＊バッチジョブは最大512個、インタラクティブジョブは最大32個

資源タイプ名と数量で資源量を指定
例） 資源タイプ(Full)を1個指定： -l rt_F=1

インタラクティブジョブ(qrsh)

1） インタラクティブノード(es)にログインして実行。

```
[aaa12345xx@es1 ~]$ qrsh -g gaa12345  -l rt_F=1 -l h_rt=01:00:00
```

 # gaa12345: グループ名, rt_F=1 : 計算資源タイプ(フルノードを1個), *h_rt=01:00:00*(最大1時間確保)

2） インタラクティブジョブの状況を参照。

```
[aaa12345xx@es1 ~]$ qstat
job-ID     prior   name       user         state submit/start at     queue                          jclass         slots ja-task-ID
------------------------------------------------------------------------------------------------------------------------------------
151646     0.28027 QRLOGIN    aaa12345xx   r     01/21/2019 09:39:43 gpu@g0371                                      10
```

予約ノードの指定法：　サブコマンド（ -ar *ar_id* ）

ar_id は「予約ID」（３桁の数字）

予約ID は、「利用ポータル」の「ノード予約・キャンセル」または、qrstatコマンドで参照できる。

ジョブ実行時に、インタラクティブに計算ノードへアクセス可能　-> デバッグ等に便利

バッチジョブ(qsub)

1） インタラクティブノード(es)にログインして実行。

```
[aaa12345xx@es1 ~]$ qsub -g gaa12345  -l rt_C.small=1 sample.sh
```

 # gaa12345: グループ名, rt_C.small=1 : 計算資源タイプ（CPU x 5コア）, *sample.sh:* ジョブスクリプト

```
Your job 151645 ("sample.sh") has been submitted
```

2） バッチジョブの状況を参照。

```
[aaa12345xx@es1 ~]$ qstat
job-ID     prior      name       user         state   submit/start at     queue                  jclass         slots ja-task-ID
-----------------------------------------------------------------------------------------------------------------------------
151646     0.25586    sample.sh  aaa12345xx   r       01/20/2019 15:16:53   gpu@g0002                      10
```

3） バッチジョブの出力（ホームディレクトリ）。

```
[aaa12345xx@es1 ~]$ ls -l
-rw-r--r-- 1 aaa12345xx gaa12345        172  1月 20 15:17 sample.sh.e151646  #エラー出力ファイル（数字はジョブ番号）
-rw-r--r-- 1 aaa12345xx gaa12345          0  1月 20 13:51 sample.sh.o151235  #正常出力ファイル（数字はジョブ番号）
```

予約ノードの指定法：　サブコマンド（ -ar *ar_id* ）

 # ar_id は「予約ID」（３桁の数字）

予約ID は、「利用ポータル」の「ノード予約・キャンセル」または、qrstatコマンドで参照できる。

ジョブをプログラム（スクリプト）化　-> 多数のジョブを一度に投げられる

計算リソース
制限事項

hour:minute:second
時間：分：秒

	インタラクティブジョブ（上限/デフォルト）	バッチジョブ（上限/デフォルト）	予約
rt_F	12:00:00/1:00:00	72:00:00/1:00:00	1日単位 最大30日間
rt_G.large rt_C.large			NA
rt_G.small rt_C.small		168:00:00/1:00:00	
ノード時間積	12	2,304	

- 72時間（3日間）以上必要な学習は、「予約」を利用する。
- 小さなジョブは、一度に多数投入できる（分散処理向き）。

ファイルのアップロード・ダウンロード(scp)

1） ファイルのアップロード： インタラクティブノード(es)にログインし、別のターミナルからscpを実行。

```
yourpc$ scp -P 10022 local-file aaa12345xx@localhost:remote-file
        # local-file : PCからアップしたいファイル、 remote-file : リモートファイル名
Enter passphrase for key: +++++++++ ← パスフレーズを入力
local-file                          100%      file-size    transfer-speed        transfer-time
```

2） ファイルのダウンロード： インタラクティブノード(es)にログインし、別のターミナルからscpを実行。

```
yourpc$ scp -P 10022 aaa12345xx@localhost:sample ./
        # sample: PCへダウンロードしたいファイル
Enter passphrase for key: +++++++++ ← パスフレーズを入力
sample                              100%      file-size    transfer-speed        transfer-time
```

インタラクティブノードにログインした状態でないと、scpできない

4. 環境構築

- Environmentモジュールの利用
- Pythonの利用
- Singularityを利用して、TensorFlowの環境を構築
- Jupyter Notebookの利用

ABCI上に「仮想環境」を構築することにより、利用者のソフトウェア環境を整えることができます。

environment moduleの利用

- アプリ実行環境の適用
  - ABCIで用意されたアプリ実行環境（環境変数）を設定可能

コマンド

```
$ module [avail] [load MODULE] [unload MODULE] [list]
```

オプション	説明
avail	利用可能な environment module の一覧を表示。av に省略可。
load	module の読み込み
unload	module の解除
list	現在読み込んでいる module をリスト表示

実行例

```
[username@g0004 Sample]$  which python3
/usr/bin/which: no python3 in (/apps/…..)
[username@g0004 Sample]$  module load python/3.6/3.6.5
[username@g0004 Sample]$
[username@g0004 Sample]$ which python3
/apps/python/3.6.5/bin/python3
```

Python3環境は未設定

python 3.6.5 環境をロード

Python3の環境が設定された

Pythonの利用 (1/5)

- ABCIで利用可能なPython環境
  - module コマンドで確認が可能

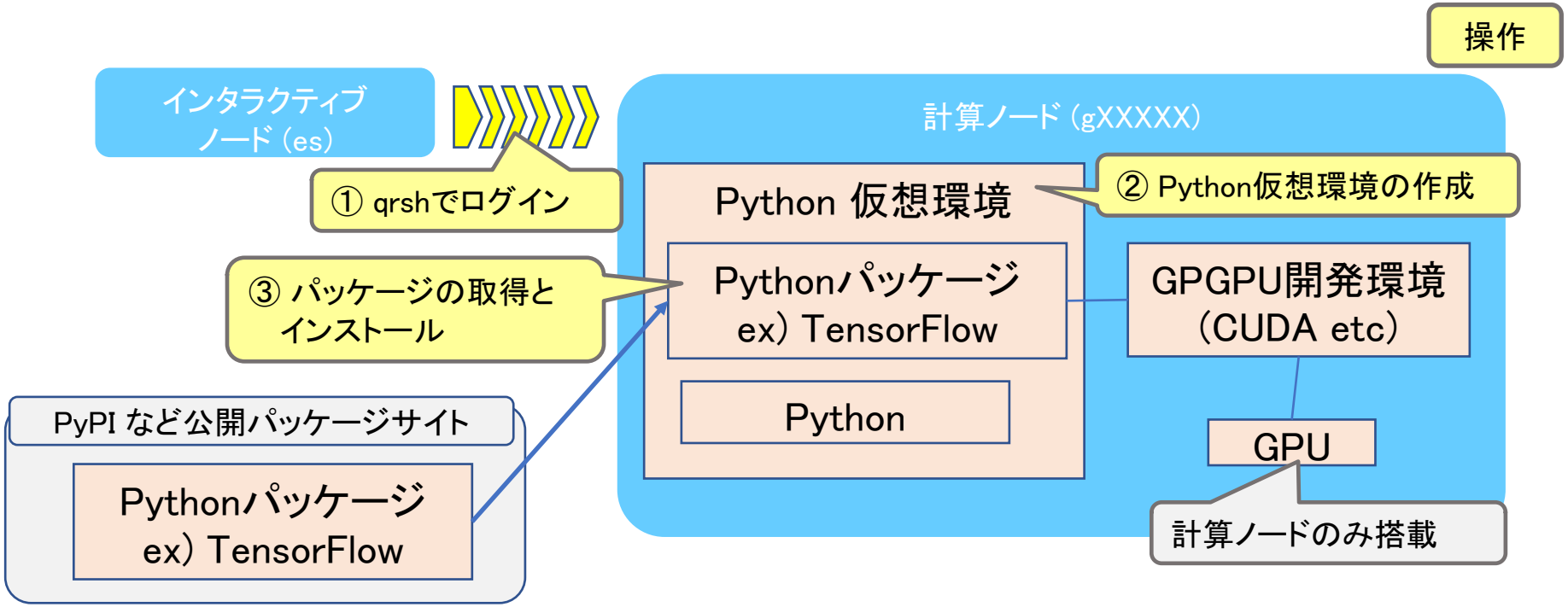
実行例

```
$ module avail
---------------------- /apps/modules/modulefiles/devtools -----------
cmake/3.11.4          openjdk/1.7.0.141     python/3.5/3.5.5
go/1.11.2             openjdk/1.8.0.131     python/3.6/3.6.5
intel-mkl/2018.2.199 python/2.7/2.7.15      R/3.5.0
openjdk/1.6.0.41      python/3.4/3.4.8
```

- Pythonパッケージをインターネットから取得可能
  - インタラクティブノードおよび計算ノードでは、PyPIなどインターネット公開されたPythonパッケージの取得可能
  - AIフレームワークを自分の環境に構築可能

Pythonの利用 (2/5)

- アプリケーションの実行環境は計算ノードで作成を推奨
  - GPGPUは計算ノードに搭載しているため
  - インタラクティブジョブとして計算ノードにログインし環境構築
- Python仮想環境の使用を推奨
  - 用途ごとに個別のアプリ環境を作成可能



Pythonの利用 (3/5)

- TensorFlow GPUのインストールと実行

Python 仮想環境の作成
- ここでは venv を使用

TensorFlow GPUのインストール
- pip install コマンドでインターネットからパッケージを取得、インストール

```
[username@es3 ~]$ qrsh -l rt_G.small -g gaa50091
[username@g0001 ~]$ module load  python/3.6/3.6.5
[username@g0001 ~]$ python3 -m venv ~/Sample/v_tf_gpu
[username@g0001 ~]$ source ~/Sample/v_tf_gpu/bin/activate
(v_tf_gpu) [username@g0001 ~]$ which python
~/Sample/v_tf_gpu/bin/python
(v_tf_gpu) [username@g0001 ~]$ module load cuda/9.0/9.0.176.4 cudnn/7.4/7.4.2
(v_tf_gpu) [username@g0001 ~]$ pip3 install --ignore-installed --upgrade
https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow_gpu-1.12.0-cp36-cp36m-linux_x86_64.whl
Collecting tensorflow-gpu==1.12.0 from https://storage.googleapis.com/tensorflow/linux/gpu/
…
Successfully installed absl-py-0.7.1 astor-0.7.1 gast-0.2.2 grpcio-1.19.0 h5py-2.9.0 keras-applications-1.0.7
keras-preprocessing-1.0.9 markdown-3.0.1 numpy-1.16.2 protobuf-3.7.0 setuptools-40.8.0 six-1.12.0
tensorboard-1.13.1 tensorflow-gpu-1.12.0 termcolor-1.1.0 werkzeug-0.15.1 wheel-0.33.1
(v_tf_gpu) [username@g0001 ~]$ deactivate
```

インタラクティブジョブの投入
計算ノードが割り当てられた
仮想環境の作成
pythonのパスが作成された
GPGPU使用環境の読み込み
Tenforflow_gpu-1.12.0 のインストール
仮想環境の終了

Pythonの利用 (4/5)

サンプルコード (https://www.tensorflow.org/tutorials/?hl=ja)

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Pythonの利用 (5/5)

- PythonでのTensorFlow 実行例

```
[username@g0001 ~]$ python3 -m venv ~/Sample/v_tf_gpu
[username@g0001 ~]$ source ~/Sample/v_tf_gpu/bin/activate
(v_tf_gpu) [axa01001hf@g0003 ~]$ python
Python 3.6.5 (default, Jun  2 2018, 15:49:50)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-16)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> mnist = tf.keras.datasets.mnist
>>> (x_train, y_train),(x_test, y_test) = mnist.load_data()
>>> x_train, x_test = x_train / 255.0, x_test / 255.0
>>>
>>> model = tf.keras.models.Sequential([
...   tf.keras.layers.Flatten(input_shape=(28, 28)),
...   tf.keras.layers.Dense(512, activation=tf.nn.relu),
...   tf.keras.layers.Dropout(0.2),
...   tf.keras.layers.Dense(10, activation=tf.nn.softmax)
... ])
>>> model.compile(optimizer='adam',
...               loss='sparse_categorical_crossentropy',
...              metrics=['accuracy'])
>>> model.fit(x_train, y_train, epochs=5)
Epoch 1/5
2019-02-04 16:38:34.256067: I
tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports
instructions that this TensorFlow binary was not compiled to use: AVX2
AVX512F FMA
2019-02-04 16:38:34.648621: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1432] Found device 0
with properties:
name: Tesla V100-SXM2-16GB major: 7 minor: 0 memoryClockRate(GHz):
1.53
pciBusID: 0000:3d:00.0
totalMemory: 15.78GiB freeMemory: 15.37GiB
2019-02-04 16:38:34.648723: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1511] Adding visible gpu
devices: 0
2019-02-04 16:38:36.224484: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:982] Device interconnect
StreamExecutor with strength 1 edge matrix:
2019-02-04 16:38:36.224530: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:988]      0
2019-02-04 16:38:36.224539: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 0:   N
2019-02-04 16:38:36.224848: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow
device (/job:localhost/replica:0/task:0/device:GPU:0 with 14874
MB memory) -> physical GPU (device: 0, name: Tesla V100-SXM2-16GB, pci bus id:
0000:3d:00.0, compute capability: 7.0)
60000/60000 [==============================] - 8s 127us/step - loss: 0.2215
- acc: 0.9341
Epoch 2/5
60000/60000 [==============================] - 3s 49us/step - loss: 0.0975 -
acc: 0.9691
Epoch 3/5
60000/60000 [==============================] - 3s 50us/step - loss: 0.0702 -
acc: 0.9783
Epoch 4/5
60000/60000 [==============================] - 3s 50us/step - loss: 0.0547 -
acc: 0.9825
Epoch 5/5
60000/60000 [==============================] - 3s 50us/step - loss: 0.0434 -
acc: 0.9860
<tensorflow.python.keras.callbacks.History object at 0x2b8c1553df60>
>>> model.evaluate(x_test, y_test)
10000/10000 [==============================] - 0s 27us/step
[0.07372516659436515, 0.9788]
>>>
```

Pythonで、TensorFlow(GPU)の環境を構築

1） 計算ノード(G.samll)にログインし、TensorFlow-gpuをインストール（一度、実施すればいい）。

CUDA9.0ではtensorflow-latestをサポートしていないので、tensorflow_gpu:1.12.0をインストール

```
[username@es3 ~]$ qrsh -l rt_G.small -g gaa50091
[username@g0001 ~]$ module load  python/3.6/3.6.5
[username@g0001 ~]$ python3 -m venv ~/Sample/v_tf_gpu
[username@g0001 ~]$ source ~/Sample/v_tf_gpu/bin/activate
(v_tf_gpu) [username@g0001 ~]$ which python
~/Sample/v_tf_gpu/bin/python
(v_tf_gpu) [username@g0001 ~]$ module load cuda/9.0/9.0.176.4 cudnn/7.4/7.4.2
(v_tf_gpu) [username@g0001 ~]$ pip3 install --ignore-installed --upgrade
https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow_gpu-1.12.0-cp36-cp36m-linux_x86_64.whl
Collecting tensorflow-gpu==1.12.0 from https://storage.googleapis.com/tensorflow/linux/gpu/
…
Successfully installed absl-py-0.7.1 astor-0.7.1 gast-0.2.2 grpcio-1.19.0 h5py-2.9.0 keras-applications-1.0.7 keras-preprocessing-1.0.9
markdown-3.0.1 numpy-1.16.2 protobuf-3.7.0 setuptools-40.8.0 six-1.12.0 tensorboard-1.13.1 tensorflow-gpu-1.12.0 termcolor-1.1.0
werkzeug-0.15.1 wheel-0.33.1
(v_tf_gpu) [username@g0001 ~]$ deactivate
```

2） PythonでTensorFlowを実行(qrsh)。

```
[username@g0001 ~]$ python3 -m venv ~/Sample/v_tf_gpu
[username@g0001 ~]$ source ~/Sample/v_tf_gpu/bin/activate
(v_tf_gpu) [username@g0001 ~]$ ~> python
Python 3.6.5 (default, Jun  2 2018, 15:49:50)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-16)] on linux
>>> import tensorflow as tf
>>>
                # TensorFlow (GPU)が利用できるようになる
```

qsubでの実行例

1） スクリプト例(run_mnist.sh)

```
#!/bin/bash
 # Environment Modules の初期化
    source /etc/profile.d/modules.sh
 # モジュールのロード
    module load python/3.6/3.6.5
    module load cuda/9.0/9.0.176.4
    module load cudnn/7.4/7.4.2
 # venv 環境の有効化
    source ~/Sample/v_tf_gpu/bin/activate
 # 計算を実行(スクリプト)
    cd ~/Sample/v_tf_gpu
    python sample_mnist.py
```

2） プログラム例(sample_mnist.py)

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

def create_model():
 model = tf.keras.models.Sequential([
   tf.keras.layers.Flatten(input_shape=(28, 28)),
   tf.keras.layers.Dense(512, activation=tf.nn.relu),
   tf.keras.layers.Dropout(0.2),
   tf.keras.layers.Dense(10, activation=tf.nn.softmax)
 ])
 return model
model = create_model()
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=3)
model.save("./mnist_model.hdf5")
print(model.evaluate(x_test, y_test))
print("finished")
```

3） スクリプト、プログラムをscpでアップロード

```
yourpc$ scp -P 10022 run_mnist.sh aaa12345xx@localhost:run_mnist.sh
yourpc$ scp -P 10022 sample_mnist.py aaa12345xx@localhost:sample_mnist.py

# ~/Sample/v_tf_gpuフォルダへ移動
```

4） インタラクティブノードでのコマンド

```
# スクリプト、プログラムに実行権を付与
[username@es3 v_tf_gpu]$ chmod u+x run_mnist.sh
[username @es3 v_tf_gpu]$ chmod u+x sample_mnist.py
[username @es3 v_tf_gpu]$ ls -l
-rwxr----- 1 aaa12345xx aaa12345xx     417  3月 27 11:13 run_mnist.sh
-rwxr----- 1 aaa12345xx aaa12345xx     720  3月 27 10:05 sample_mnist.py
# -rwxr- となっていることを確認

[username @es3 v_tf_gpu]$ qsub -g gaa12345 -l rt_G.small=1 run_mnist.sh
```

Singularityを利用して、TensorFlow(GPU)の環境を構築

1） インタラクティブノード(es)にログインし、TensorFlowのDockerイメージを取得（一度、実施すればいい）。

以下のイメージを使用 (tag: -gpu-py3)
https://hub.docker.com/r/tensorflow/tensorflow/

```
[aaa12345xx@es3 ~]$ module load  singularity/2.6.1
[aaa12345xx@es3 ~]$ singularity pull docker://tensorflow/tensorflow:1.12.0-gpu-py3
Docker image path: index.docker.io/tensorflow/tensorflow:1.12.0-gpu-py3
Cache folder set to /fs3/home/axa01001hf/.singularity/docker
[17/17] |==================================| 100.0%
                          :
Done. Container is at: ./tensorflow-1.12.0-gpu-py3.simg
[aaa12345xx@es3 ~]$ ls
tensorflow-1.12.0-gpu-py3.simg
```

2） インタラクティブジョブ（計算ノード）で、Singularityを実行し、TensorFlowの環境を構築。

```
[aaa12345xx@es3 ~]$ qrsh -l rt_F=1 -l h_rt=01:00:00 -g gaa12345
[aaa12345xx@g0003 ~]$ module load  singularity/2.6.1
[aaa12345xx@g0003 ~]$ singularity shell --nv ./tensorflow-1.12.0-gpu-py3.simg
Singularity: Invoking an interactive shell within container...

Singularity tensorflow-1.12.0-gpu-py3.simg:~> python
>>> import tensorflow as tf
>>>
                # TensorFlow (GPU)が利用できるようになる
```

＊コンテナイメージは、Docker Hub以外からもpullすることが可能です。

TensorFlowの実行例

```
hagishima — aaa10005yk@g0195:~ — ssh -p 10022 -l aaa10005yk localhost — 193×70
[aaa10005yk@es3 ~]$ qrsh -l rt_F=1 -l h_rt=01:00:00 -g gaa50069
[aaa10005yk@g0195 ~]$ module load singularity/2.6.1
[aaa10005yk@g0195 ~]$ singularity shell --nv ./tensorflow-latest-gpu-py3.simg
Singularity: Invoking an interactive shell within container...

Singularity tensorflow-latest-gpu-py3.simg:~> python
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> mnist = tf.keras.datasets.mnist
>>> (x_train, y_train),(x_test, y_test) = mnist.load_data()
>>> x_train, x_test = x_train / 255.0, x_test / 255.0
>>> model = tf.keras.models.Sequential([
...   tf.keras.layers.Flatten(input_shape=(28, 28)),
...   tf.keras.layers.Dense(512, activation=tf.nn.relu),
...   tf.keras.layers.Dropout(0.2),
...   tf.keras.layers.Dense(10, activation=tf.nn.softmax)
... ])
>>> model.compile(optimizer='adam',
...               loss='sparse_categorical_crossentropy',
...               metrics=['accuracy'])
>>> model.fit(x_train, y_train, epochs=5)
Epoch 1/5
2019-01-31 05:41:33.601800: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 AVX512F FMA
2019-01-31 05:41:34.114172: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1432] Found device 0 with properties:
name: Tesla V100-SXM2-16GB major: 7 minor: 0 memoryClockRate(GHz): 1.53
pciBusID: 0000:3d:00.0
totalMemory: 15.78GiB freeMemory: 15.37GiB
2019-01-31 05:41:34.450015: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1432] Found device 1 with properties:
name: Tesla V100-SXM2-16GB major: 7 minor: 0 memoryClockRate(GHz): 1.53
pciBusID: 0000:3e:00.0
totalMemory: 15.78GiB freeMemory: 15.37GiB
2019-01-31 05:41:34.789030: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1432] Found device 2 with properties:
name: Tesla V100-SXM2-16GB major: 7 minor: 0 memoryClockRate(GHz): 1.53
pciBusID: 0000:b1:00.0
totalMemory: 15.78GiB freeMemory: 15.37GiB
2019-01-31 05:41:35.138469: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1432] Found device 3 with properties:
name: Tesla V100-SXM2-16GB major: 7 minor: 0 memoryClockRate(GHz): 1.53
pciBusID: 0000:b2:00.0
totalMemory: 15.78GiB freeMemory: 15.37GiB
2019-01-31 05:41:35.138560: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1511] Adding visible gpu devices: 0, 1, 2, 3
2019-01-31 05:41:38.702250: I tensorflow/core/common_runtime/gpu/gpu_device.cc:982] Device interconnect StreamExecutor with strength 1 edge matrix:
2019-01-31 05:41:38.702304: I tensorflow/core/common_runtime/gpu/gpu_device.cc:988]      0 1 2 3
2019-01-31 05:41:38.702314: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 0:   N Y Y Y
2019-01-31 05:41:38.702321: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 1:   Y N Y Y
2019-01-31 05:41:38.702328: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 2:   Y Y N Y
2019-01-31 05:41:38.702334: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 3:   Y Y Y N
2019-01-31 05:41:38.703068: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 14874 MB memory) -> physical GP
U (device: 0, name: Tesla V100-SXM2-16GB, pci bus id: 0000:3d:00.0, compute capability: 7.0)
2019-01-31 05:41:38.704461: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:1 with 14874 MB memory) -> physical GP
U (device: 1, name: Tesla V100-SXM2-16GB, pci bus id: 0000:3e:00.0, compute capability: 7.0)
2019-01-31 05:41:38.704741: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:2 with 14874 MB memory) -> physical GP
U (device: 2, name: Tesla V100-SXM2-16GB, pci bus id: 0000:b1:00.0, compute capability: 7.0)
2019-01-31 05:41:38.705017: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:3 with 14874 MB memory) -> physical GP
U (device: 3, name: Tesla V100-SXM2-16GB, pci bus id: 0000:b2:00.0, compute capability: 7.0)
60000/60000 [==============================] - 10s 171us/step - loss: 0.2208 - acc: 0.9353
Epoch 2/5
60000/60000 [==============================] - 4s 65us/step - loss: 0.0966 - acc: 0.9708
Epoch 3/5
60000/60000 [==============================] - 4s 65us/step - loss: 0.0679 - acc: 0.9785
Epoch 4/5
60000/60000 [==============================] - 4s 66us/step - loss: 0.0529 - acc: 0.9826
Epoch 5/5
60000/60000 [==============================] - 4s 66us/step - loss: 0.0418 - acc: 0.9863
<tensorflow.python.keras.callbacks.History object at 0x2b85720acef0>
>>> model.evaluate(x_test, y_test)
10000/10000 [==============================] - 0s 33us/step
[0.06713192030405335, 0.9799]
>>>
```

参照：_index.ipynb (Google)
https://www.tensorflow.org/tutorials/?hl=ja

Jupyter Notebookの利用

1） インタラクティブノード(es)にログインし、Jupyter Notebookをインストール（一度、実施すればいい）。

```
[aaa12345xx@es3 ~]$ module load python/3.6/3.6.5
[aaa12345xx@es3 ~]$ python3 -m venv ~/lib/pyenv/jupyter_test
[aaa12345xx@es3 ~]$ source ~/lib/pyenv/jupyter_test/bin/activate
 (jupyter_test) es3 $ pip install --upgrade pip
 (jupyter_test) es3 $ pip install jupyter
 (jupyter_test) es3 $ deactivate
```

2） インタラクティブジョブ(qrsh)で、Jupyter Notebookを起動する。

```
[aaa12345xx@es3 ~]$ qrsh -g gaa12345 -l rt_F=1 -l h_rt=01:00:00
[aaa12345xx@g0019 ~]$ module load python/3.6/3.6.5
                # aaa12345xx: username, g0019: 割当てられた計算ノードリソース
[aaa12345xx@g0019 ~]$ source ~/lib/pyenv/jupyter_test/bin/activate
 (jupyter_test) [aaa12345xx@g0019 ~]$ jupyter notebook --no-browser --ip=`hostname` >> jupyter.log 2>&1 &
 (jupyter_test) [aaa12345xx@g0019 ~]$ jupyter notebook list
Currently running servers:
http://g0004.abci.local:8888/?token=e7f0ba979d4ffd9eeb7e6debf5a326f853fc289583f92dc5 :: /fs3/home/aaa12345xx
```

3） 別ターミナルで。

```
yourpc$ ssh -L 10022:es:22 -l aaa12345xx as.abci.ai
```

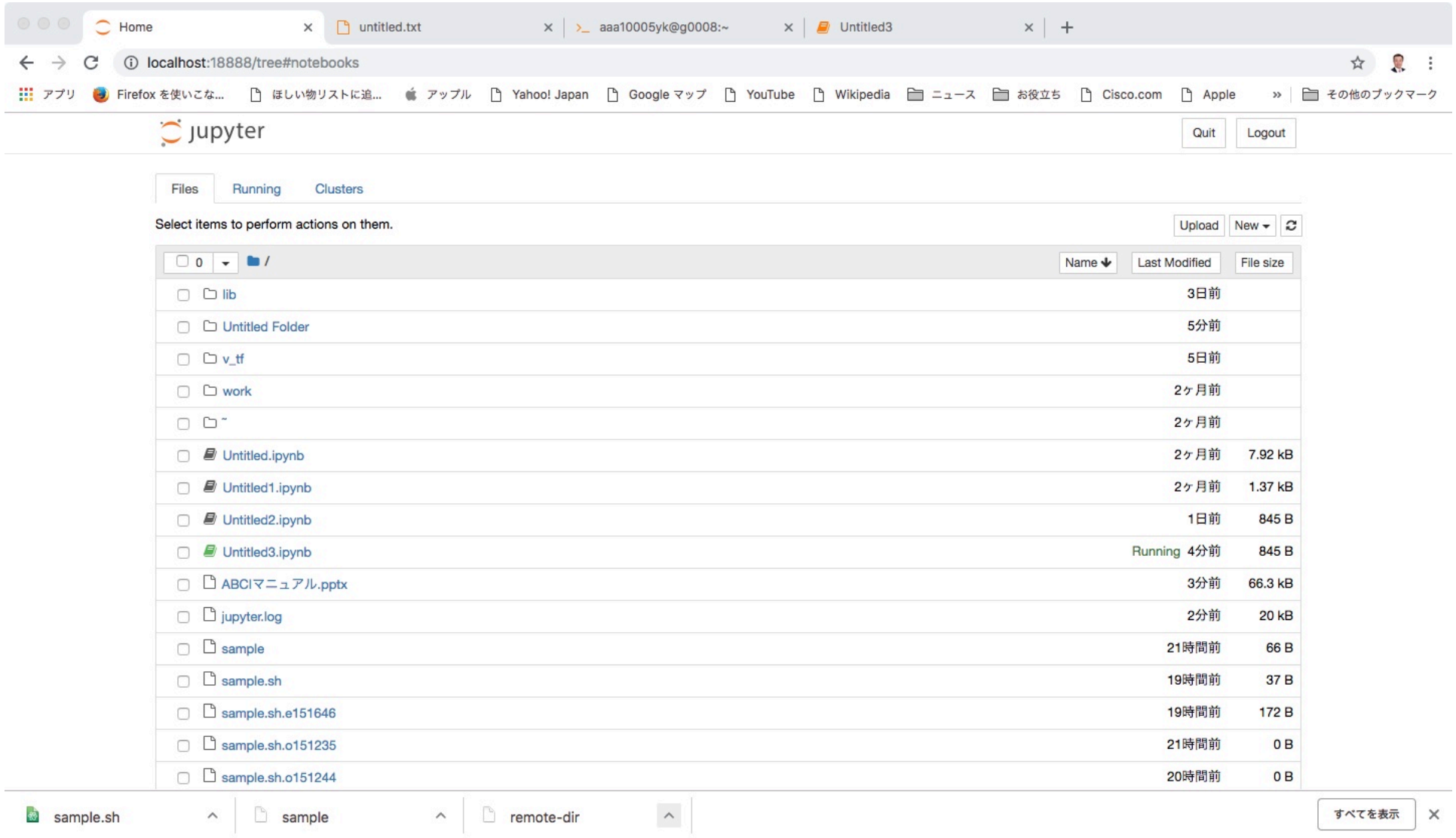
4） さらに、別のターミナルで。

```
yourpc$ ssh -L 18888:g0004:8888 -l aaa12345xx (-i ~/.ssh/id_rsa) -p 10022 localhost
                # -i：秘密鍵オプションは省略可
```

5） ブラウザでアクセス（トークンは、2）をコピペ）。

```
http://localhost:18888/?token=e7f0ba979d4ffd9eeb7e6debf5a326f853fc289583f92dc5
```

Jupyter Notebook画面例



5. ソフトウェアの利用

- 深層学習フレームワーク
- ビッグデータ解析フレームワーク

インストール済みのソフトウェアは、moduleコマンドで利用可能。

深層学習フレームワーク

ABCIシステムで深層学習フレームワークを利用する場合、利用者がホーム領域またはグループ領域にインストールする必要があります。
深層学習フレームワークのインストール方法は下記のサイトをご参照ください。

https://portal.abci.ai/docs/ja/11/

フレームワーク
Caffe
Caffe2
TensorFlow
Theano
Torch
PyTorch
CNTK
MXNet
Chainer
Keras

ビッグデータ解析フレームワーク

ABCIシステムでは、Hadoopが利用可能です。
Hadoopの環境設定は、Moduleコマンドで行います。

1） インタラクティブノード(es)にログインし、Hadoopをインストール（一度、実施すればいい）。

```
[aaa12345xx@es3~]$ module load openjdk/1.8.0.131
[aaa12345xx@es3~]$ module load hadoop/2.9.1
```

2） インタラクティブジョブ(qrsh)でのHadoop実行サンプル。

```
[aaa12345xx@es3 ~]$ qrsh -g gaa12345 -l rt_F=1 -l h_rt=01:00:00
[aaa12345xx@g0019 ~]$ module load openjdk/1.8.0.131
[aaa12345xx@g0019 ~]$ module load hadoop/2.9.1
[aaa12345xx@g0019 ~]$ mkdir input
[aaa12345xx@g0019 ~]$ cp /apps/hadoop/2.9.1/etc/hadoop/*.xml input
[aaa12345xx@g0019 ~]$ hadoop jar /apps/hadoop/2.9.1/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.9.1.jar grep input output
'dfs[a-z.]+'
[aaa12345xx@g0019 ~]$ cat output/part-r-00000
1       dfsadmin
```

インストール済みソフトウェア

ソフトウェア	w/o CUDA	CUDA-8.0.61.2	CUDA-9.0.176.2	CUDA-9.1.85.3	CUDA-9.2.88.1
OpenMPI-2.1.3 (for GCC)	○	○	○	○	○
OpenMPI-3.1.0 (for GCC)	○	○	○	○	○*1
MVAPICH2 (for GCC)	○	○	○	○	○
NCCL-1.3.5	X	○	○	○	○*2

*1 OpenMPI-3.1.0 (CUDA-9.2.88.1) のサンプルコマンド。

```
[aaa12345xx@es3 ~]$ wget https://download.open-mpi.org/release/open-mpi/v3.1/openmpi-3.1.0.tar.bz2
[aaa12345xx@es3 ~]$ tar xvjf openmpi-3.1.0.tar.bz2

[aaa12345xx@es3 ~]$ cd openmpi-3.1.0
[aaa12345xx@es3 openmpi-3.1.0]$ module load cuda/9.2/9.2.88.1
```

*2 NCCL-1.3.5 (CUDA-9.2.88.1) のサンプルコマンド。

```
[aaa12345xx@es3 ~]$ git clone https://github.com/NVIDA/nccl.git
[aaa12345xx@es3 ~]$ cd nccl
[aaa12345xx@es3 nccl]$ module load cuda/9.2/9.2.88.1
```

利用者はルート権限(su)を持てないので、よく利用されるソフトウェアはプリインストールしてある。

利用可能ソフトウェア一覧

項目	ソフトウェア	バージョン	
OS	CentOS	7.4	
開発環境	Intel Parallel Stduio XE Cluster Edition	2018.2.046	
	PGI Professional Edition	18.5	
	NVIDIA CUDA SDK	8.0.61.2 9.0.176.2 9.0.176.3 9.0.176.4	9.1.85.3 9.2.88.1 9.2.148.1
	GCC	4.8.5	
	Python	2.7.15 3.4.8	3.5.5 3.6.5
	Ruby	2.0.0.648-33	
	R	3.5.0	
	Java	1.8.0_131	
	Scala	1.27-248	
	Lua	5.1.4	
	Perl	5.16.3	
ファイルシステム	DDN GRIDScaler	4.2.3-8	
	BeeOND	6.18	
コンテナ	Docker	17.12.0	
	Singularity	2.6.1	

6. ABCI料金表
H31年度(2019年度)

		インタラクティブジョブ	バッチジョブ	予約
計算ノード	rt_F	1.0 ポイント/時間		36ポイント/日 (1.5ポイント/時間)
	rt_G.large	0.9ポイント/時間		NA
	rt_C.large	0.6ポイント/時間		
	rt_G.small	0.3ポイント/時間		
	rt_C.small	0.2ポイント/時間		
ストレージ	共有ディスク	5 ポイント/TB・月 (デフォルト200MBは無償)		
	セキュア オブジェクトストレージ	(H31年度から提供予定)		

7. 計算ノードの予約利用

項目	説明
最小予約日数	1日
最大予約日数	30日
システムあたりの最大同時予約可能ノード数	442ノード
1予約あたりの最大予約ノード数	32
1予約あたりの最大予約ノード時間積	12,288ノード時間積
予約受付開始時刻	30日前の午前10時
予約受付締切時刻	予約開始前日の午後9時
予約取消受付期間	予約開始前日の午後9時
予約開始時刻	予約開始日の午前10時
予約終了時刻	予約終了日の午前9時30分

予約ノードを利用する際は、qsubコマンド又はqrshコマンドにて、「-ar ar_id」オプションを使う。

ABCI参考サイト

より詳細な情報については、以下を参照下さい。

- ABCIユーザサポート
  https://abci.ai/ja/how_to_use/user_support.html
- ABCI利用に関するFAQ
  https://abci.ai/ja/how_to_use/yakkan.html
- 利用の手引き
  https://portal.abci.ai/docs/ja/
- 利用ポータル
  https://portal.abci.ai/user/
- 利用ポータルの手引き
  https://portal.abci.ai/docs/portal/ja/