共同利用計算機「ディープラーニングシステム」の導入と運用

杉山 耕一朗*, 岩澤全規*, 岡田 康†, 池田 総一郎†,
川見 昌春†, 稲葉 洋*, 原 元司*, 廣瀬 誠*, 金山 典世*

* 松江高専・情報工学科
†松江高専・実践教育支援センター

2021/03/03 高専フォーラム OS-13

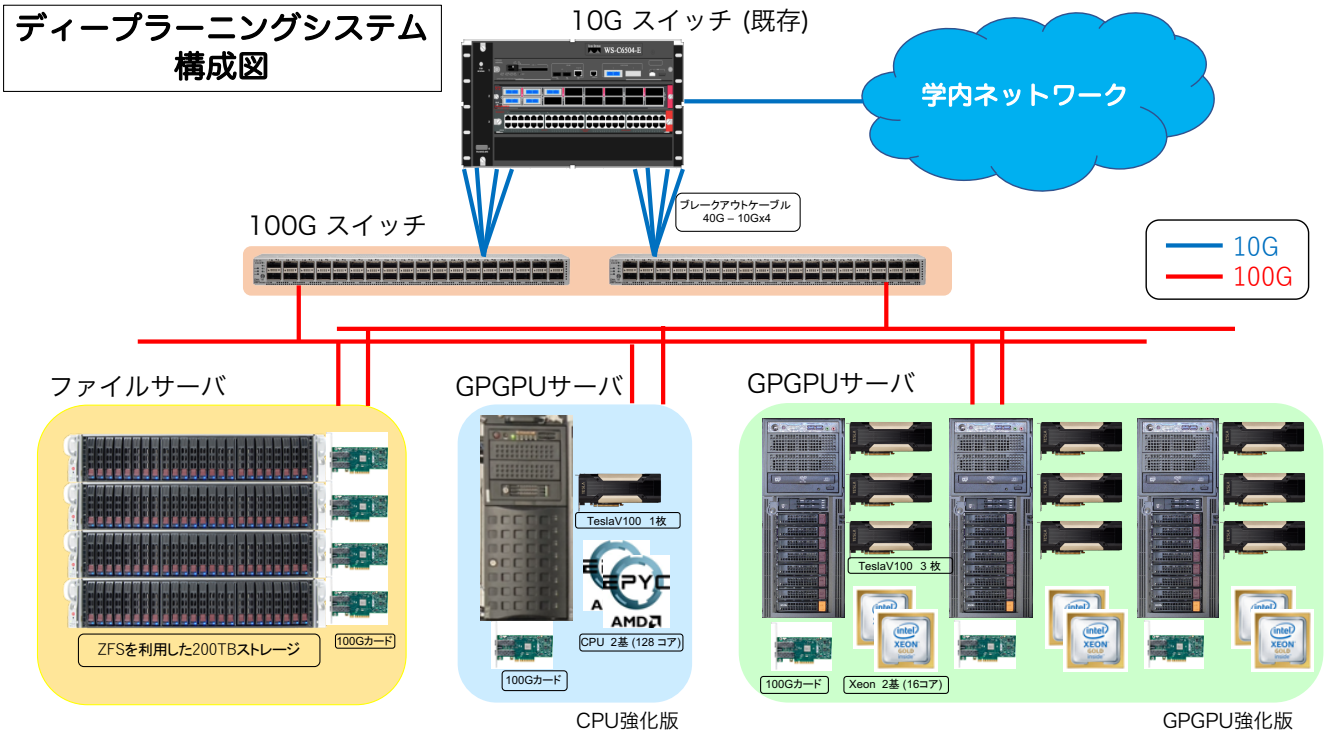
はじめに

- コロナ禍であっても/だからこそ，教育研究を下支えする情報基盤の導入と運用は必須である.
- 近年，分野や学科を問わず，ディープラーニングに関連する教育研究が盛んに行われている.
- ディープラーニング用計算機の学内的なニーズは高い.
- 松江高専情報処理センターでは，共同利用計算機としてディープラーニングシステムを導入・運用することになったので，その報告する.

本発表の内容

- 導入したシステムの概要 (2020年11月末 納品)
  - スペック，特徴
- 運用方法の検討
  - 世間の動向・学内動向を踏まえて
- テスト運用 (2020年12月末～)
  - 第1フェーズ，第2フェーズ
- 現在の利用方法のイメージ
- まとめ，今後の予定

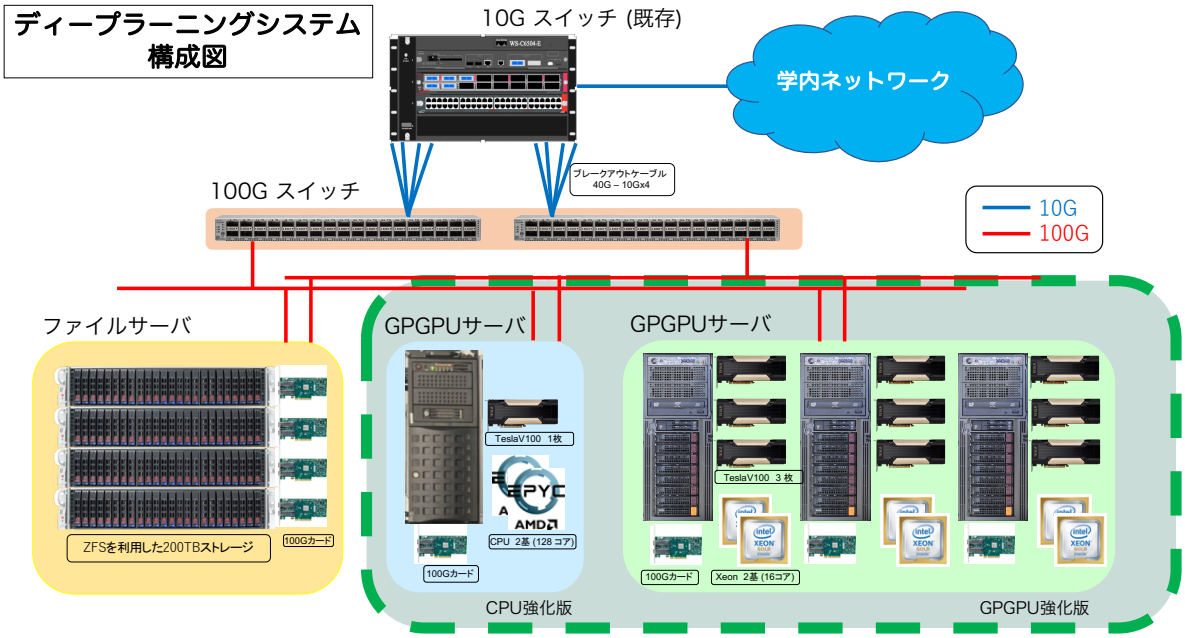
システムの概要



高専マスタープランで採択・導入．2020年11月末に納品．

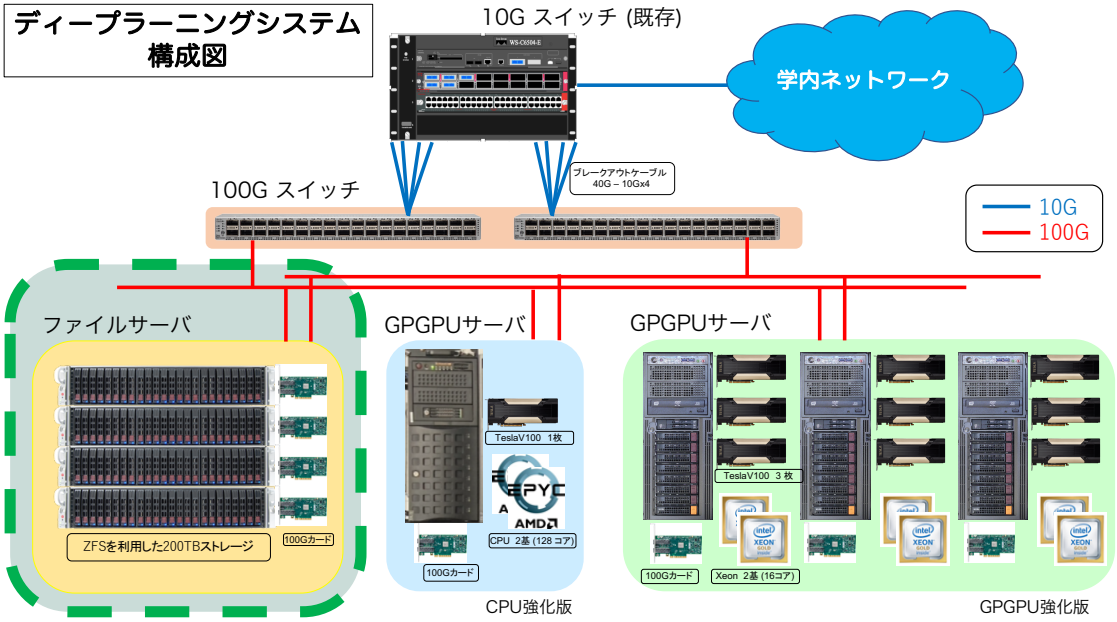
システムの特徴

- 2 種類の GPGPU サーバ
  - GPGPU 強化版 : Intel Xeon + GPGPU(Tesla V100S) 3基
  - CPU 強化版 : AMD EPYC (128コア) + GPGPU(Tesla V100S) 1基



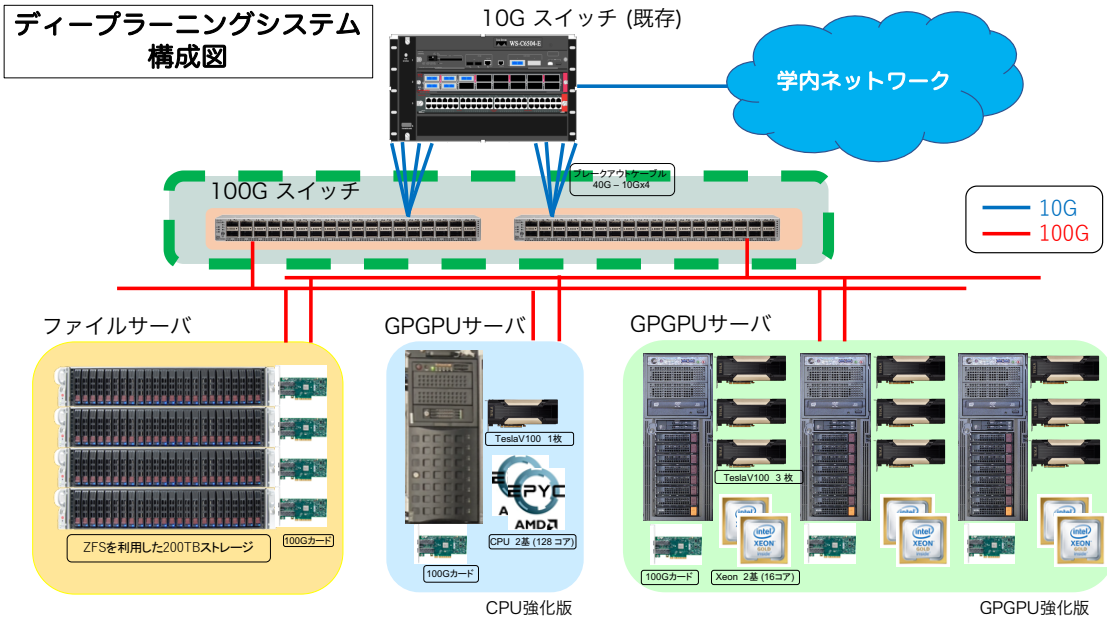
システムの特徴

- 大容量ファイルサーバ
  - ZFS で SSD 4TB x 12台を束ねる
  - NFS で GPGPUサーバと接続



システムの特徴

- バックボーンとしての100Gネットワーク
  - ブレークアウトケーブルで既存10Gスイッチと接続
  - 将来的には本校の基幹ネットワーク中核としての利用も



導入・運用にあたって: 学内の動向調査

- 研究室で購入している GPGPU は 10 万円程度が多い
  - メモリ容量は 4 ~ 8 GB 程度.

➡ 2 種類のGPGPUサーバ
最上位GPGPUカードの導入

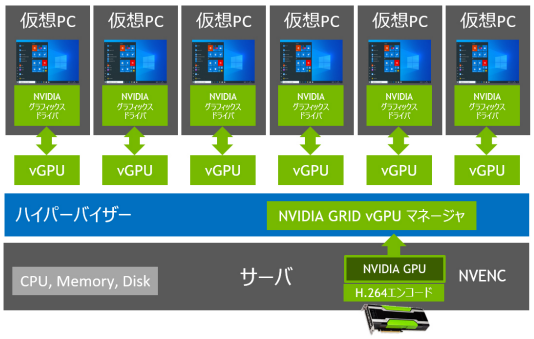
- 教員の希望は大きく分けて 2 グループ
  - GPU 性能・数を重視. CPU には大きなこだわりない. (大多数)
  - GPU 性能だけでなく CPU コア数も重視 (少数)
    - 数値シミュレーションの加速器としての利用など

- 言語は Python, 利用するライブラリは研究室毎に多様.
  - 研究室内で複数のライブラリ環境を使っているケースも
    tensorflow, pytorch, chainer, keras, matplotlib, opencv, numpy, cupy, …

- OS：Linux (Ubuntu) が多数.
  - Linux のコマンドラインからPythonプログラムを実行
  - Windows の GUI, Google Colaboratory (Web UI)を使っている研究室もある.

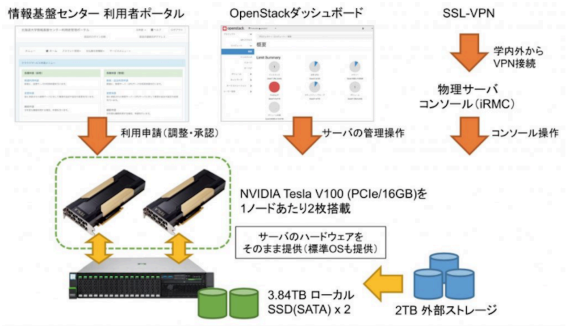
運用方法の検討：世間の動向

- VM (Virtual Machine)



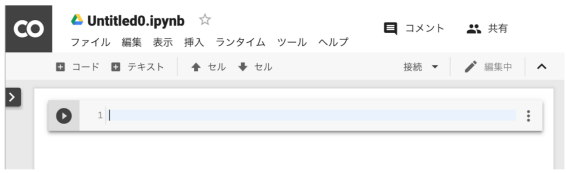
https://www.fujitsu.com/jp/products/computing/servers/validation/case/gpu/index.html

- ベアメタルの提供



北大 情報基盤センター

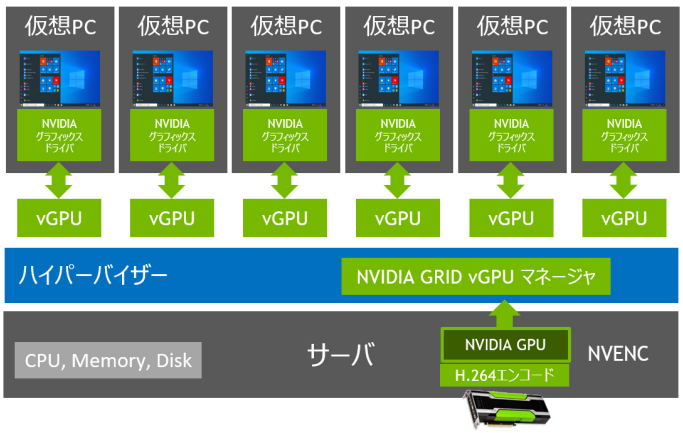
- Web UI (Jupyter など)
  - Google Colaboratory



- ジョブ管理システム
  - スパコンの GPU ノードなどで使われる形態
  - 管理ノードにログイン・ジョブ投入
  - 計算機資源の厳密な管理が可能

運用方法：入札時点での想定

- 利用を希望する研究室にVMを提供
  - OSの選択，ライブラリ環境構築は各研究室に任せる
  - ハイパーバイザ：VMware vSphere
- vGPU で 1 つの GPGPU を 2~4 研究室で分割
  - GPGPU メモリ 32GB を分割
    - GPGPU の数 < 研究室の数

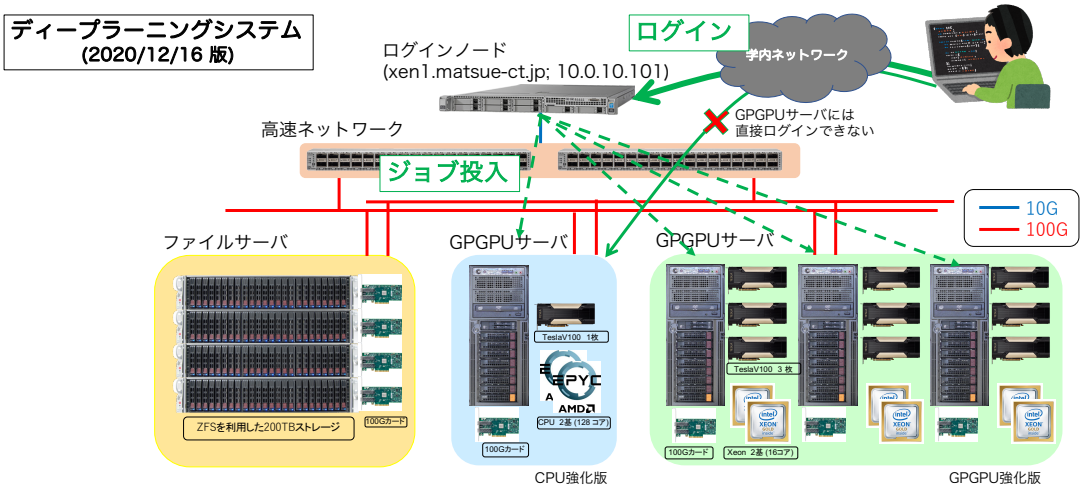


https://www.fujitsu.com/jp/products/computing/servers/validation/case/gpu/index.html

運用方法：テスト運用 第1フェーズ

2020年12月末～

- 構築が思惑通りにいかず…
  - 管理者：VMware や vGPU の設定方法が全然理解できなくて
  - ユーザ：GPGPU のメモリを細かく分割しないで欲しいとの声
- 運用方法：ジョブ管理システム
  - 多様性：Python 環境の仮想化 (ユーザが構築)



Python 環境の仮想化

- レベルの異なる 2 種類の仮想化をサポート
  - 必要とされるライブラリは, Python, cuda, cuDNN などのバージョンに強く依存

- Python venv
  - Python の標準機能
  - ホストマシンの Python, cuda cuDNNなどを利用.
  - ユーザ権限でライブラリのインストールが可能

- docker
  - コンテナ型の仮想環境
  - ホストマシンのカーネルを利用
  - ユーザ権限でゲスト OSの動作が可能
    - Pythonや各種ライブラリのバージョン選択の自由度が高い

運用方法：テスト運用 第2フェーズ

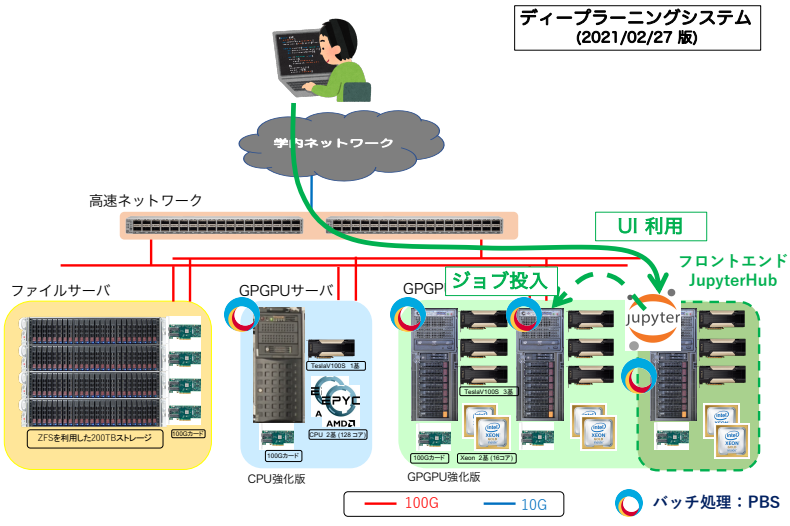
2021年3月～

- 第1フェーズにおけるユーザからの声
  - 管理ノードに SSH でログイン&ジョブ投入は敷居が高い
  - Python の仮想環境を自分で構築するのが難しい/出来ない
    - 先輩が作った計算実行環境をそのまま利用してきたユーザが多い
- 運用方法：Web UI + ジョブ管理システムのハイブリッド
  - Web UI (JupyterHub) によって利用の敷居を下げる
    - Google Colaboratory と類似の利用方法に
  - システム側で Python 仮想環境を複数提供
    - ほとんどのユーザは仮想環境を構築しなくて済む
    - 自前で仮想環境を構築することも引き続き可能.
  - 計算資源の使い分け
    - テスト計算，短時間の計算 → Web UI
    - 長時間計算 (バッチ処理) → ジョブ管理システム

運用方法：テスト運用 第2フェーズ

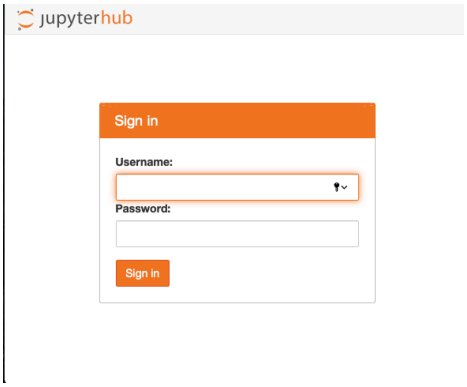
2021年3月～

- 第1フェーズにおけるユーザからの声
  - 管理ノードに SSH でログイン&ジョブ投入は敷居が高い
  - Python の仮想環境を自分で構築するのが難しい/出来ない
    - 先輩が作った計算実行環境をそのまま利用してきたユーザが多い
- 運用方法：Web UI + ジョブ管理システムのハイブリッド

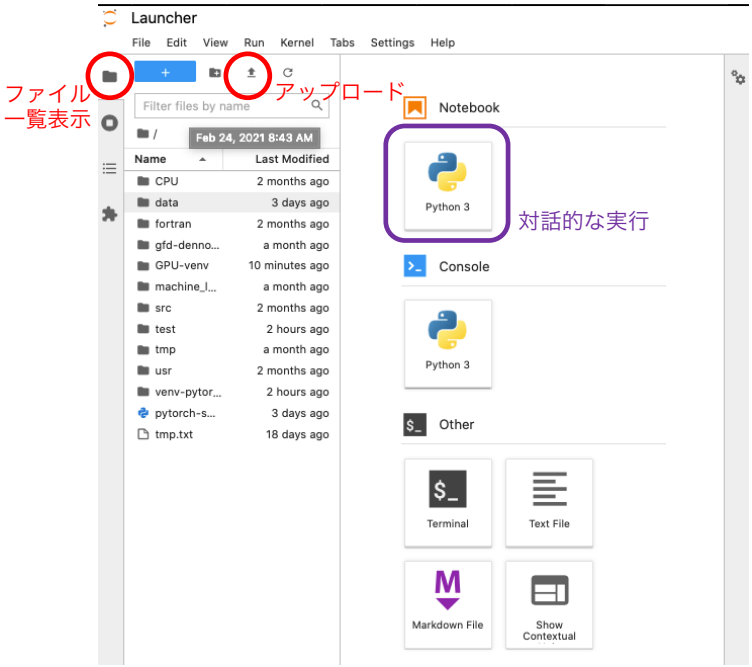


利用のイメージ

(1) ログイン画面



(2) ランチャー画面 (メニュー: File → New Launcher)

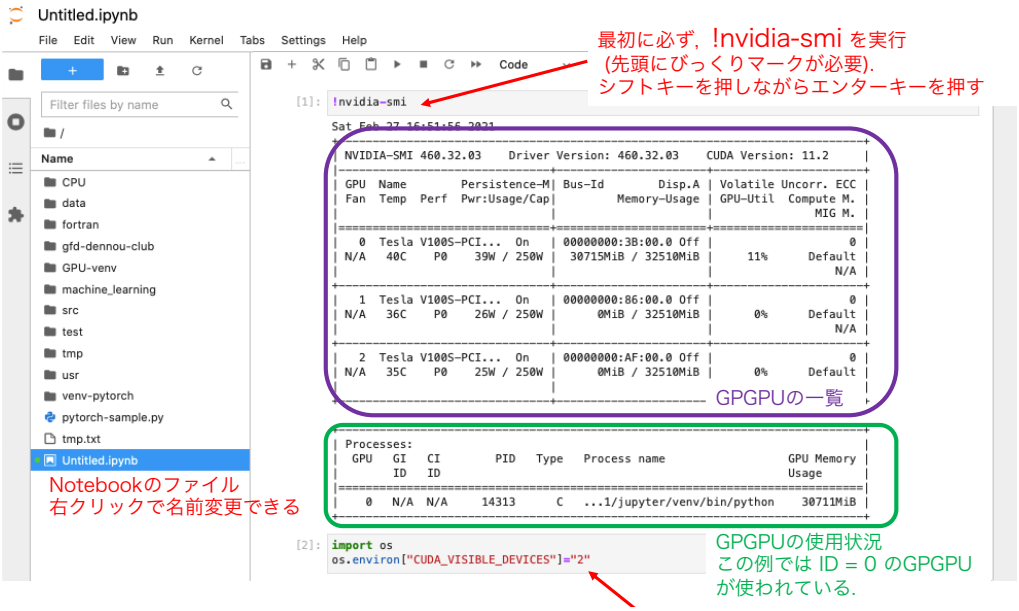


利用のイメージ

(1) Notebook を起動

クリック

(2) GPGPU の空き状況を確認し，使う GPGPU の ID を指定する.

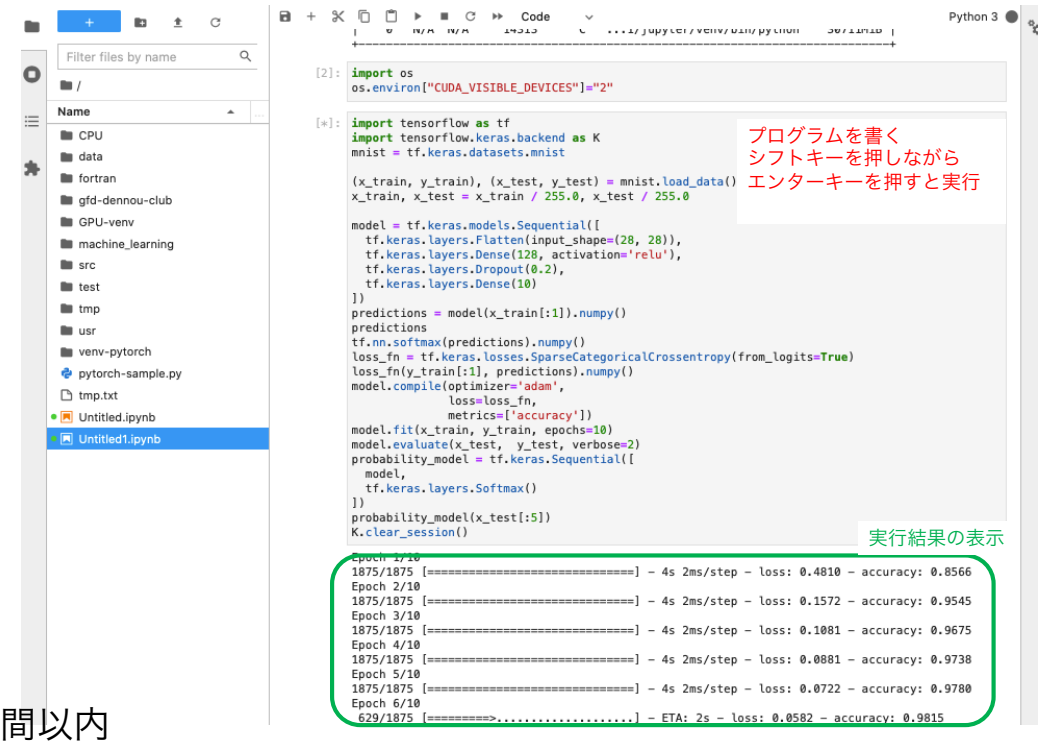


計算時間：36 時間以内

プロセス監視：Sleep なプロセスは強制終了 (Jupyter が GPU を握り続けるため)

利用のイメージ

(3) プログラムを書いて実行する.



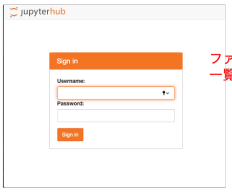

```
[2]: import os
os.environ["CUDA_VISIBLE_DEVICES"]="2"

[*]: import tensorflow as tf
import tensorflow.keras.backend as K
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(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
predictions = model(x_train[:1]).numpy()
predictions
tf.nn.softmax(predictions).numpy()
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
loss_fn(y_train[:1], predictions).numpy()
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10)
model.evaluate(x_test,  y_test, verbose=2)
probability_model = tf.keras.Sequential([
  model,
  tf.keras.layers.Softmax()
])
probability_model(x_test[:5])
K.clear_session()
```

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.4810 - accuracy: 0.8566
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1572 - accuracy: 0.9545
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1081 - accuracy: 0.9675
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0881 - accuracy: 0.9738
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0722 - accuracy: 0.9780
Epoch 6/10
 629/1875 [=========>....................] - ETA: 2s - loss: 0.0582 - accuracy: 0.9815
```

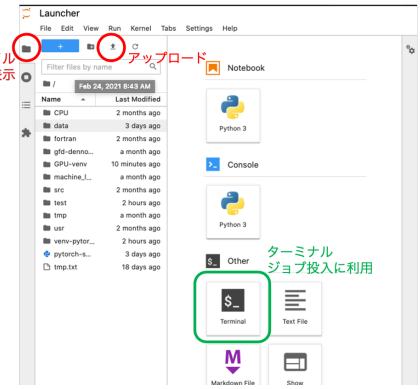
計算時間：36 時間以内
プロセス監視：Sleep なプロセスは強制終了 (Jupyter が GPU を握り続けるため)

利用のイメージ (ジョブ投入)

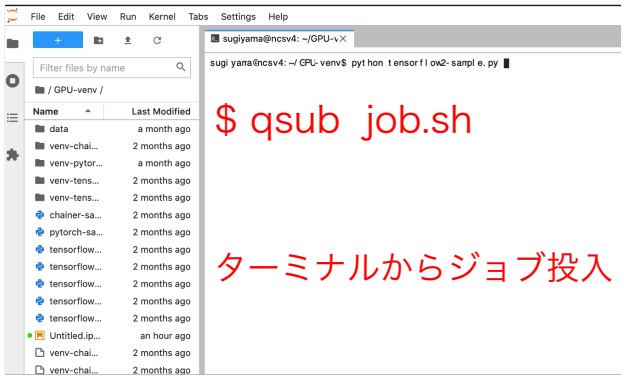
(1) ログイン画面



(2) ランチャー画面 (メニュー: File → New Launcher)



(3) ターミナルの起動



ジョブスクリプトの例 (python venv)

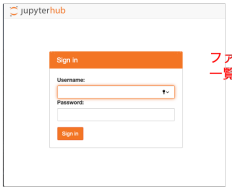
```
#!/bin/sh
#PBS -q gpu
#PBS –l  select=1:ncpus=1:ngpus=1

cd $PBS_O_WORKDIR

. venv/bin/activate    # venv 仮想環境利用
python test.py         # プログラム実行
deactivate
```

ジョブスクリプトの例 (python + docker)

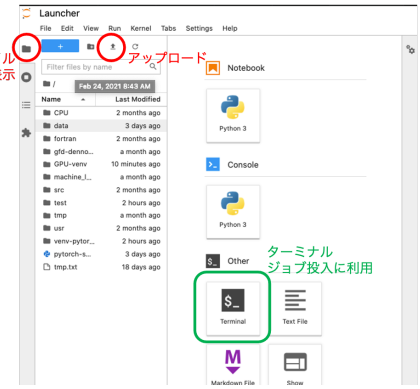
```
#!/bin/sh
#PBS -q gpu
#PBS –l  select=1:ncpus=1:ngpus=1

cd $PBS_O_WORKDIR

# docker の仮想環境を利用して実行
docker run --gpus all -v $PWD:/tmp ¥
-w /tmp --rm chainer/matsue-ct      ¥
python3 test.py
```

利用のイメージ (ジョブ投入)

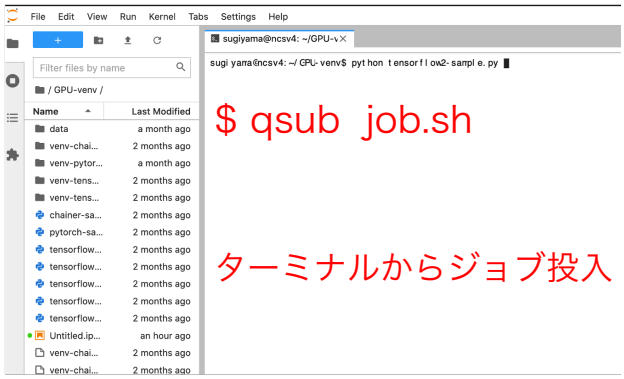
(1) ログイン画面



(2) ランチャー画面 (メニュー: File → New Launcher)



(3) ターミナルの起動



ジョブスクリプトの例 (C 言語, Fortran)

```
#!/bin/sh
#PBS -q cpu  #キューの指定
#PBS –l select=1:ncpus=8:mpiprocs=8

cd $PBS_O_WORKDIR

# 実行
mpirun ./c_omphello
```

利用可能なキュー

gpu
- 利用可能 CPU : 1 ~ 6
- 利用可能 GPU : 1 基
- 研究室単位での同時実行可能数に制限あり (実行可能数 3).
- 実行時間の制限あり (72 時間)

cpu :
- 利用可能 CPU : 1 ~ 64
- GPU 利用不可
- 研究室単位での同時実行可能数に制限あり (実行可能数 2).
- 実行時間の制限あり (72 時間)

まとめ

- 共同利用計算機としてのディープラーニングシステムの導入と運用を開始した.
- ユーザの利用状況を見ながら運用方法をアップデート中
- 現時点の運用方法
  - Web UI + ジョブ管理システム
  - Python環境の仮想化 (venv, docker)
- テスト運用第2フェーズを経て, 本格運用を開始 (2021年4月末を予定).
  - ユーザの義務：システムを利用した研究発表の報告