Centre de Calcul
de l'Institut National de Physique Nucléaire
et de Physique des Particules

The Jupyter notebooks platform at CC-IN2P3

Bernard CHAMBON, ATLAS France users meeting, November 21, 2023

Outline

- Introduction
- Architecture
- Focusing on two features
- Demo
- Infrastructure and figures
- Annex

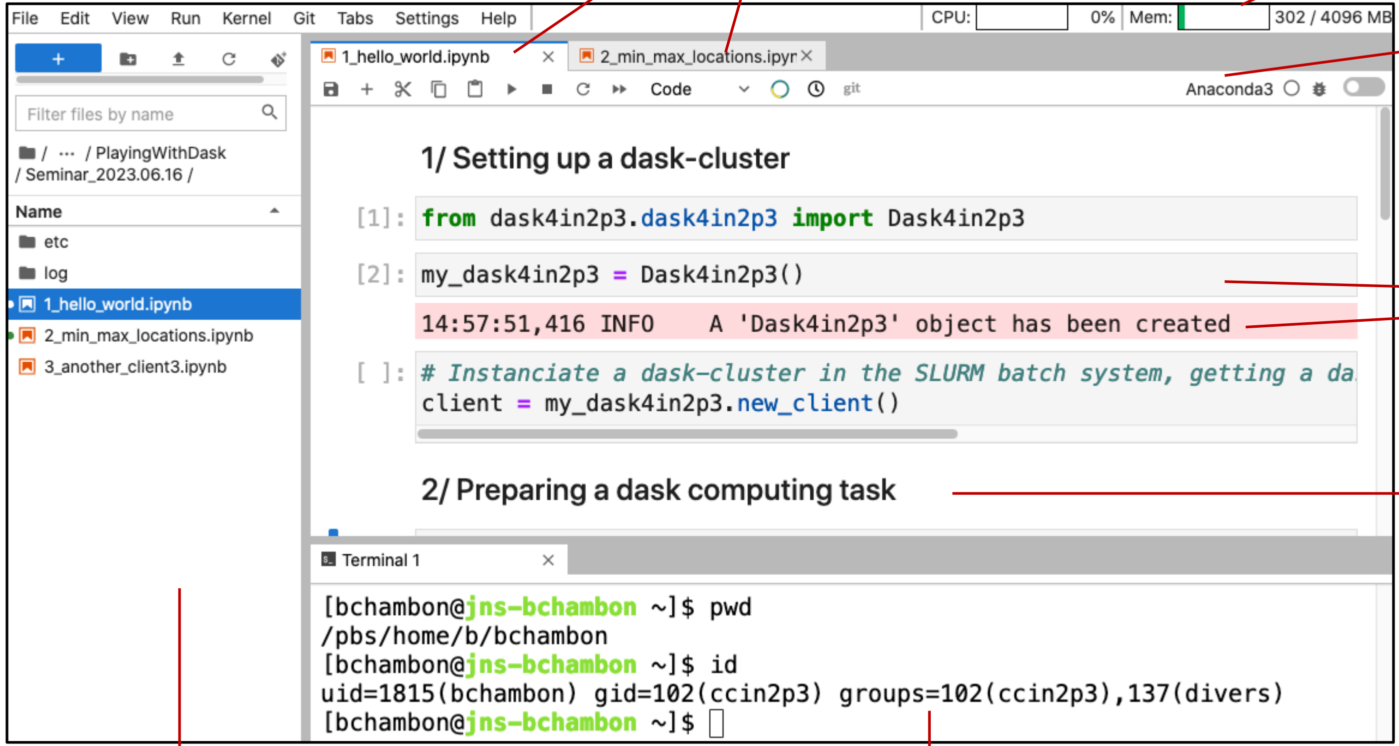
Introduction : 1/2

- Objective
  - Provide an interactive analysis service, via the Jupyter notebooks
  - With access to the same storage systems as those available on the interactive platform (cca.in2p3.fr)
  - Authentication through the SSO of the CC-IN2P3

- Some key points of the Jupyter notebooks
  - Simplicity
    - Running in a web browser
    - Using a same document for code, documentation, results of execution
    - Providing an UNIX terminal (without ssh-ing)
  - Multiple programming languages, via kernels (Jupyter = **Ju**lia, **Pyt**hon, **R)**
  - Richness of ecosystem via extensions

Introduction : 2/2

- My first notebooks server


File Edit View Run Kernel Git Tabs Settings Help
CPU: 0% Mem: 302 / 4096 MB
1_hello_world.ipynb
2_min_max_locations.ipyn
Code
Anaconda3
Filter files by name
/ ··· / PlayingWithDask / Seminar_2023.06.16 /
Name
etc
log
1_hello_world.ipynb
2_min_max_locations.ipynb
3_another_client3.ipynb
Terminal 1

1/ Setting up a dask-cluster

```
[1]: from dask4in2p3.dask4in2p3 import Dask4in2p3

[2]: my_dask4in2p3 = Dask4in2p3()

     14:57:51,416 INFO     A 'Dask4in2p3' object has been created

[ ]: # Instanciate a dask-cluster in the SLURM batch system, getting a da
     client = my_dask4in2p3.new_client()
```

2/ Preparing a dask computing task

```
[bchambon@jns-bchambon ~]$ pwd
/pbs/home/b/bchambon
[bchambon@jns-bchambon ~]$ id
uid=1815(bchambon) gid=102(ccin2p3) groups=102(ccin2p3),137(divers)
[bchambon@jns-bchambon ~]$ 
```

2 notebooks in a notebooks server

Widget for view on CPU et RAM

Kernel (anaconda3)

Cells of code and Result of execution of code (text or graph)

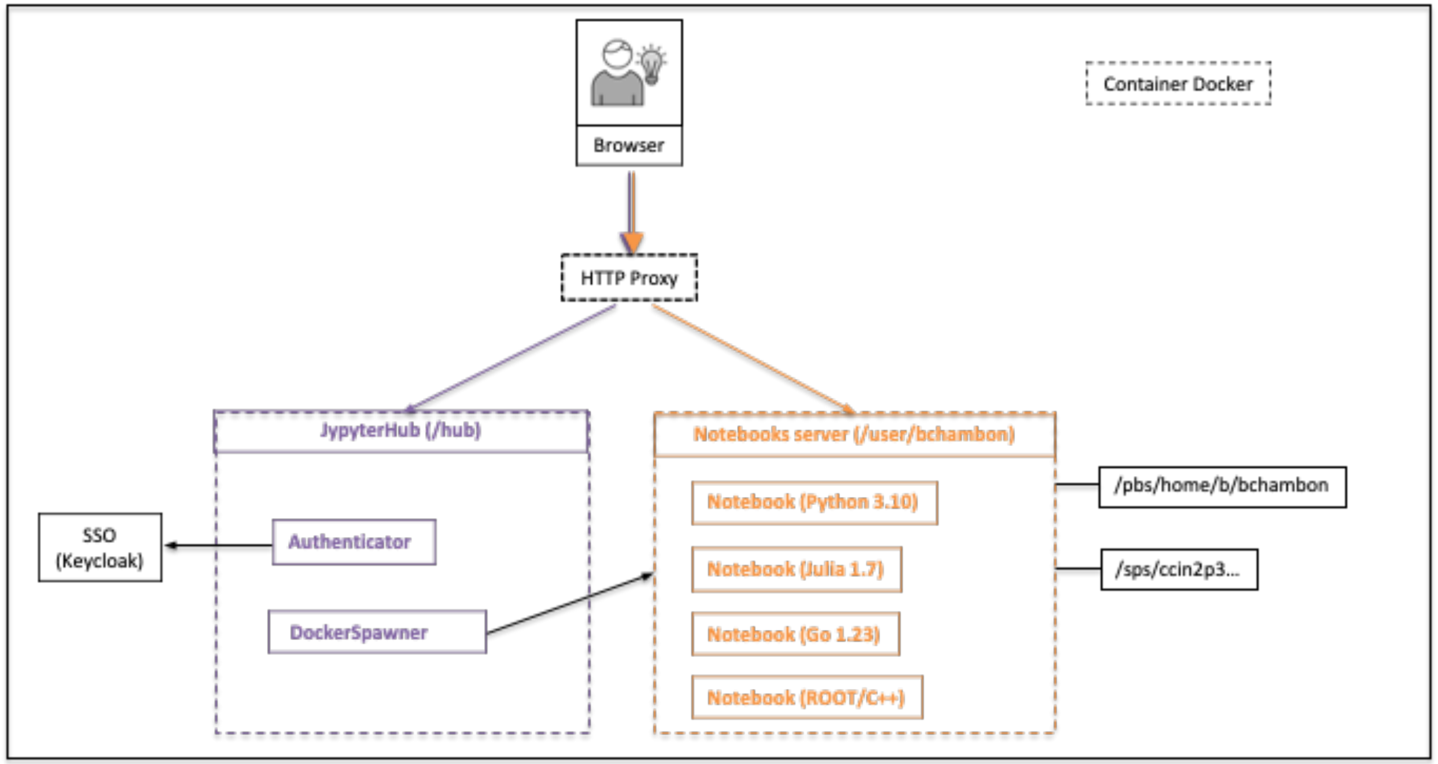
Documentation in markdown

Basic file manager

Unix terminal

Architecture : 1/3

- Built around **JupyterHub**
  - Component allowing to plug an external authentication (OAuth), to provide options forms, to run Docker images
  - Python config file, allowing advanced configuration



- The service is running on a Docker cluster with Swarm as orchestrator to spawn the notebooks servers on the hosts

Architecture : 2/3

- Access & authentication
  - Access allowed for all users having a ‘computing’ account, but GPU feature is restricted to granted users
  - Authentication using OAuth to SSO Keycloak (certificate or login/password)
  - Getting additional information to provide a complete user’s profile (including all secondary groups)

- Launching the Jupyter notebooks server
  - Docker image prepared at CC-IN2P3, based on CentOS 7.6 (same as batch platform and interactive platform)
  - Container running with IDs (uid, gid) of the user
  - With the following storage systems :

• HOME (/pbs/…) area, GROUPS (/sps/…) areas according to primary & secondary groups	Specific paths for each user
• THRONG, SOFTWARE and CVMFS areas	Same paths for all users

Architecture : 3/3

- Limits for RAM, CPU and lifetime
  - RAM
    - Default is 2 GB, higher limit possible per user or per group (if several groups using the max, logon having higher priority than group)
    - 30+ users with 8, 16 or 24 GB, some users with 32 or 64 GB
    - A widget display the memory used and limit
  - CPU
    - No limit for number of CPUs, but …
    - A ready-to-use parameter can be (manually) activated per logon or per group (in case of overload detected)
  - Notebooks server lifetime
    - No usage time limit, but …
    - Notebooks servers are monitored and IDLE ones are stopped after 3 days | 1 day for, respectively, CPU | GPU notebooks servers

RAM, CPU and I/O consumptions are monitored for internal usage only (using cAdvisor, Prometheus and Grafana tools)

About two features

- Using GPUs

- Using ‘Dask+SLURM’

GPU feature : 1/2

- Objective
  - Allow users to run GPU code via a Jupyter notebook

- How to
  - Granted access upon request (possible per user or per group)
  - Option form to select the model of GPU, the number of GPUs
  - Also possible to select the amount of RAM of the notebooks server

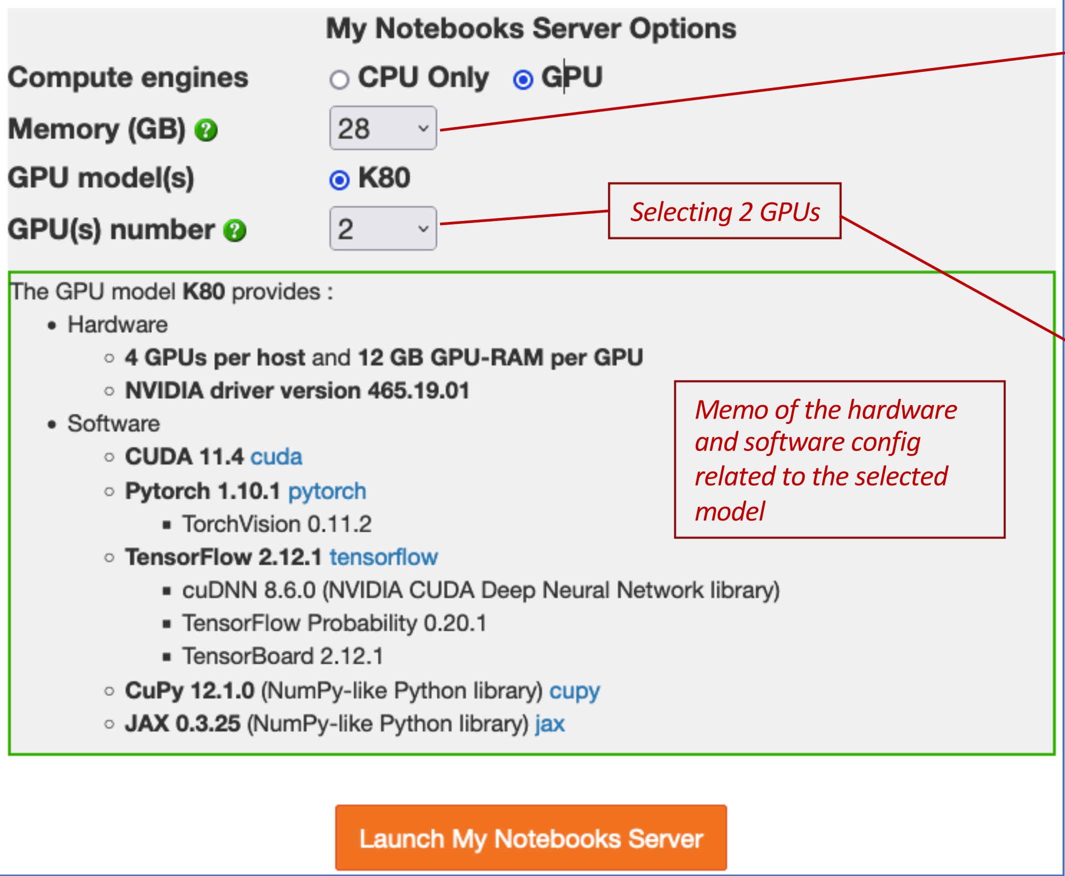
- User will obtain
  - A running notebooks server with dedicated GPUs
  - With ML frameworks + libs (already installed inside the Docker image)
    - TensorFlow + TensorBoard + TensorFlow Probability,
    - cuDNN
    - Pytorch
    - JAX (NumPy-like Python library)

  - On Nov 2023, 7 GPU hosts, model K80, 4 GPUs /host => up to 28 simultaneous users, each one having one GPU

GPU feature : 2/2

Options form and running a GPU notebooks server

Options form



My Notebooks Server Options

Compute engines	CPU Only [✓ GPU]
Memory (GB)	28
GPU model(s)	[✓ K80]
GPU(s) number	2

The GPU model **K80** provides :

- Hardware
  - **4 GPUs per host** and **12 GB GPU-RAM per GPU**
  - **NVIDIA driver version 465.19.01**
- Software
  - **CUDA 11.4** cuda
  - **Pytorch 1.10.1** pytorch
    - TorchVision 0.11.2
  - **TensorFlow 2.12.1** tensorflow
    - cuDNN 8.6.0 (NVIDIA CUDA Deep Neural Network library)
    - TensorFlow Probability 0.20.1
    - TensorBoard 2.12.1
  - **CuPy 12.1.0** (NumPy-like Python library) cupy
  - **JAX 0.3.25** (NumPy-like Python library) jax

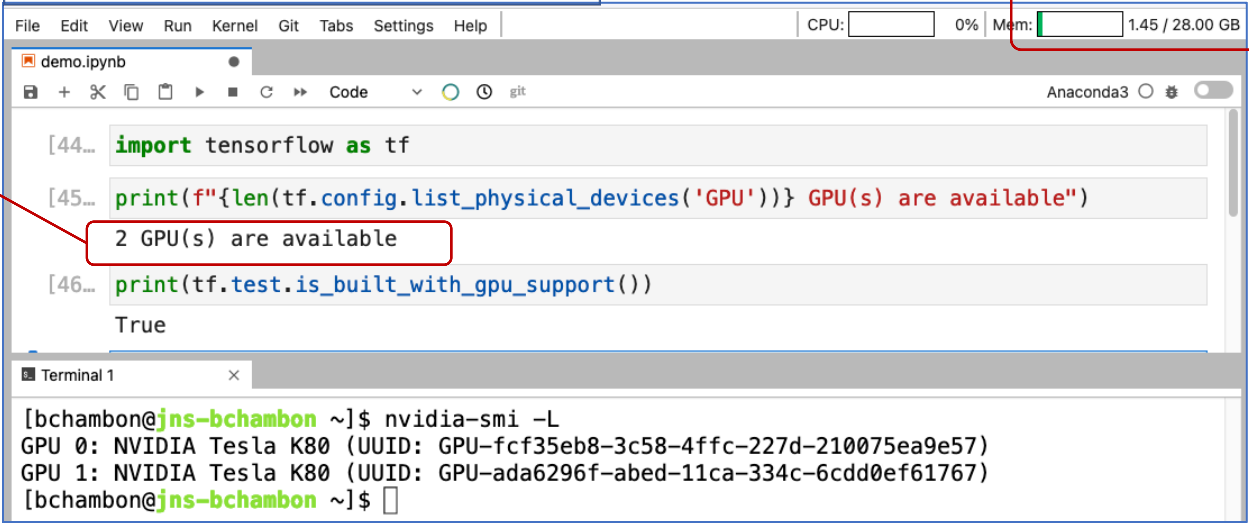
Launch My Notebooks Server

Selecting 28 GB de RAM

Selecting 2 GPUs

Memo of the hardware and software config related to the selected model

A 2 GPUs notebooks server with 28 GB of RAM



```
[44… import tensorflow as tf
[45… print(f"{len(tf.config.list_physical_devices('GPU'))} GPU(s) are available")
     2 GPU(s) are available
[46… print(tf.test.is_built_with_gpu_support())
     True
```

```
[bchambon@jns-bchambon ~]$ nvidia-smi -L
GPU 0: NVIDIA Tesla K80 (UUID: GPU-fcf35eb8-3c58-4ffc-227d-210075ea9e57)
GPU 1: NVIDIA Tesla K80 (UUID: GPU-ada6296f-abed-11ca-334c-6cdd0ef61767)
[bchambon@jns-bchambon ~]$
```

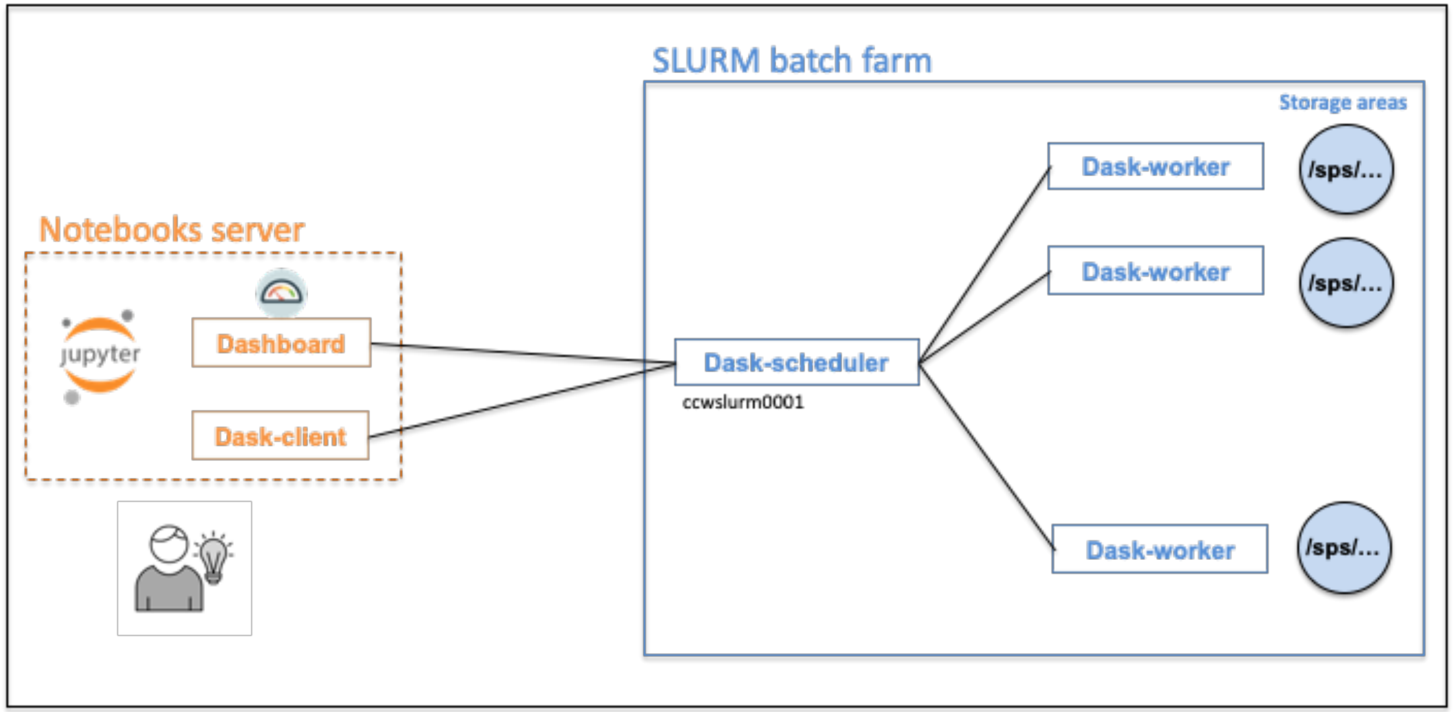
‘Dask+SLURM’ feature

- Objective and architecture
- How to use it
- *Demo*

‘Dask+SLURM’ feature : 1/3

- Objectives
  - Allow interactive analysis for huge amount of data via parallel processing
  - From notebooks server (for interactivity) and by using resources from SLURM batch farm (for performance)
  - By spreading computing tasks, with Dask, over potentially several hundred of SLURM jobs

- Architecture



‘Dask+SLURM’ feature : 2/3

- How-to
  - This feature is not restricted (= available for everyone using the Jupyter notebooks platform)
  - By writing Dask code (= in Python) and using the ‘dask4in2p3’ package (already installed in the Docker image)

- User will be able to specify :
  - The number of jobs, the RAM and elapse time of the jobs (same values for all jobs)
  - A virtual environment (where the package ‘dask4in2p3’ is installed, since it’s also required on ‘SLURM batch farm’ side)
  - Other parameters (timeouts, etc.) See docstring of methods (*via Shift-Tab after selecting the method*)

Each parameter having a default value, it’s usable without any specification

- User will get
  - A dask-client connected to the dask-scheduler
  - A dashboard showing live metrics related to dask-workers (via ‘dask-labextension’)
  - The possibility to connect another dask-client(s) on the running dask-cluster (only one dask-cluster /user)

‘Dask+SLURM’ feature : 3/3

- *Demo*
  - Processing of 200 images (edge detection)
  - Objective of the demo
    - Impact of parallelization (100 jobs in our case) to reduce the overall processing time
    - Availability of a dashboard providing live metrics
    - Availability to connect another dask-client (from another notebook) but using the same dask-cluster
  - Silent video of 2’30

Figures on the infrastructure

- Hardware
  - 1 server : VM : 8 CPUs, 16 GB RAM — where JupyterHub runs
  - 19 workers : — where notebooks servers run
    - **11** VMs : 8 CPUs, 32 or 64 GB RAM per VM
      - 7 VMs dedicated for computing
      - 4 VMs dedicated for training
    - **9** bare metal hosts : 16 CPUs, 130 GB RAM, 1 Gbps I/O, per host
      - 7 dedicated for computing on GPU (model K80, 4 GPUs/host)
      - 2 dedicated for computing on CPU (for users with high requirements in terms of RAM or I/O)

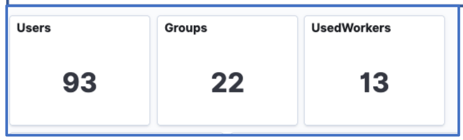
- This infrastructure can serve **100+** users (CPU + GPU) including 28 users each one having one GPU

- Easily extendable by using VMs form Openstack

Figures on the use of the service

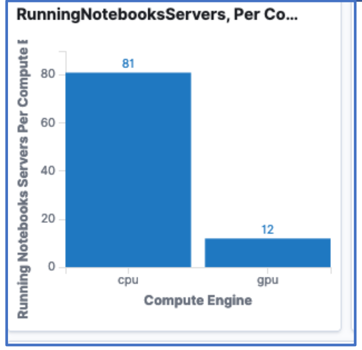
Snapshot view, mid-October 2023

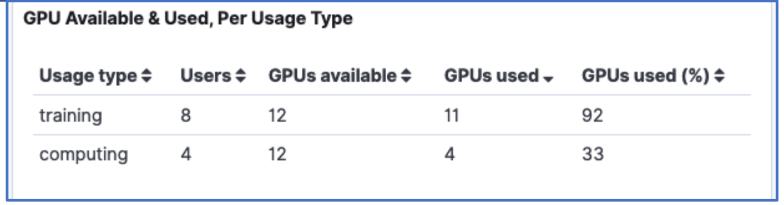
__93__ users from __22__ groups and using __13__ hosts



Users	Groups	UsedWorkers
93	22	13

__81__ CPU users; __12__ GPU users (__4__ in 'computing' and __8__ in 'training')

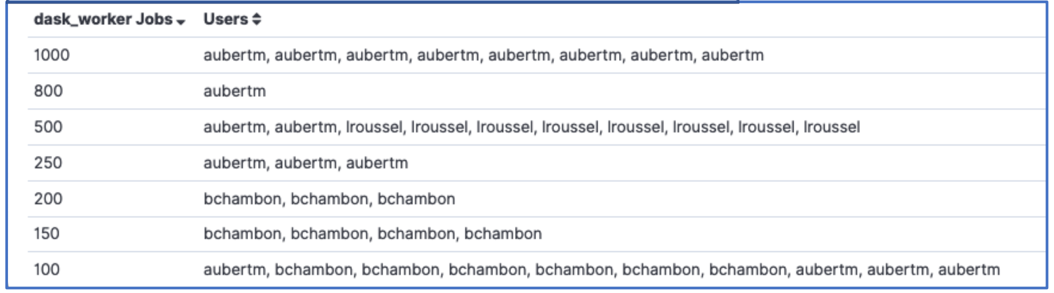




GPU Available & Used, Per Usage Type

Usage type	Users	GPUs available	GPUs used	GPUs used (%)
training	8	12	11	92
computing	4	12	4	33

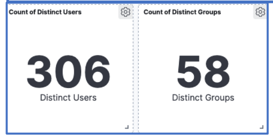
Dask+SLURM feature, up to __1000__ jobs (dask-workers)



dask_worker Jobs	Users
1000	aubertm, aubertm, aubertm, aubertm, aubertm, aubertm, aubertm, aubertm
800	aubertm
500	aubertm, aubertm, lroussel, lroussel, lroussel, lroussel, lroussel, lroussel, lroussel, lroussel
250	aubertm, aubertm, aubertm
200	bchambon, bchambon, bchambon
150	bchambon, bchambon, bchambon, bchambon
100	aubertm, bchambon, bchambon, bchambon, bchambon, bchambon, bchambon, aubertm, aubertm, aubertm

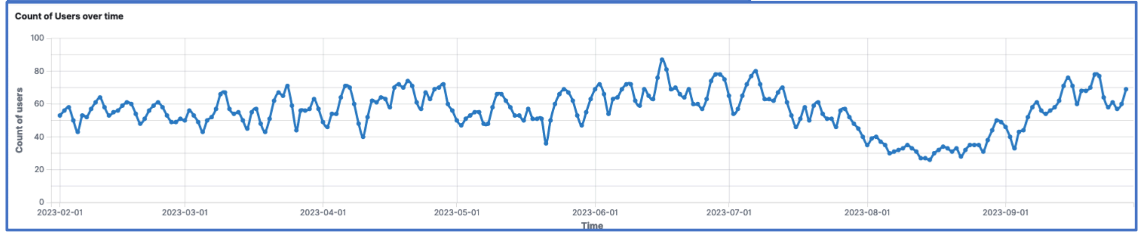
8-month view (February 1 to September 30, 2023)

__300+__ users in __58__ distinct groups

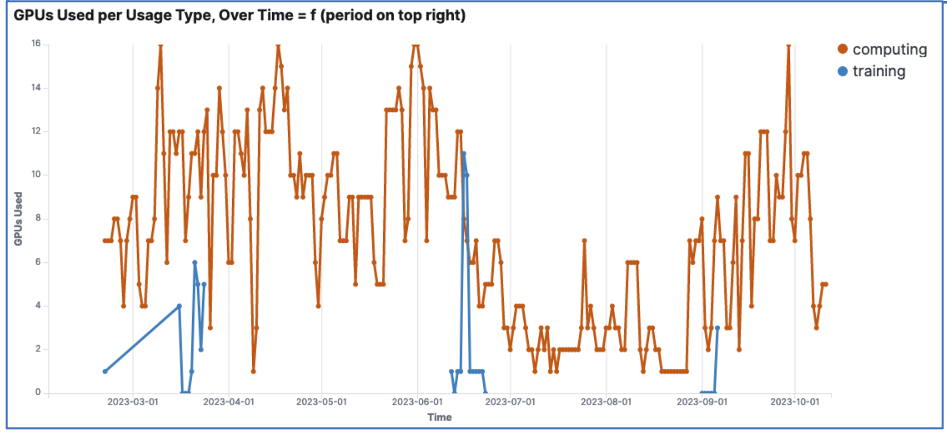


Count of Distinct Users	Count of Distinct Groups
306 Distinct Users	58 Distinct Groups

Number of CPU + GPU users (peak at __87__ in June 2023)



GPUs usage for computing (most of the time) or training (from time to time)



Outcome

- A new service at CC-IN2P3 (since July 2020)
  - Available for all users having a 'computing' account
  - Configured to serve various needs
    - For data analysis, for training sessions
    - Providing both CPU or GPU resources
    - Providing CPU resources of the SLURM batch farm, by using the Dask framework
  - And with a reactive support
- URLs
  - Read the documentation Jupyter Notebooks Platform
  - Access to the service https://notebook.cc.in2p3.fr/
  - Ask for support https://support.cc.in2p3.fr/
  - Specific to the 'Dask+SLURM' feature
    - The documentation Dask usage
    - The project Dask4in2p3
    - Notebooks as examples Demodask4in2p3

Thank you for your attention

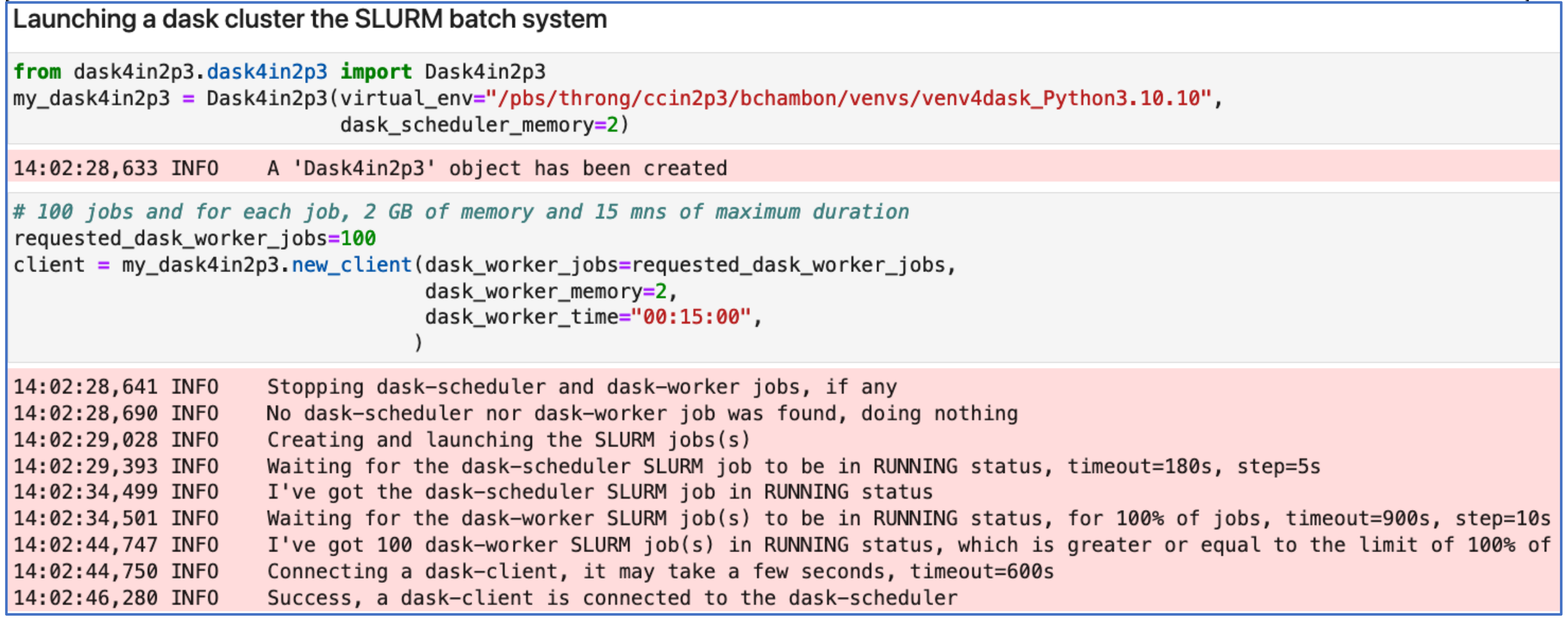
Annexs

- 2 screenshots summarising the demo

- Feature ‘Dask+SLURM’ : About the package ‘dask4in2p3’

Summarising the demo (‘Dask+SLURM’) : 1/2

*1) Running a dask-cluster in the SLURM batch farm, with **100** dask-workers of 2 GB and 15 minutes max*



Launching a dask cluster the SLURM batch system

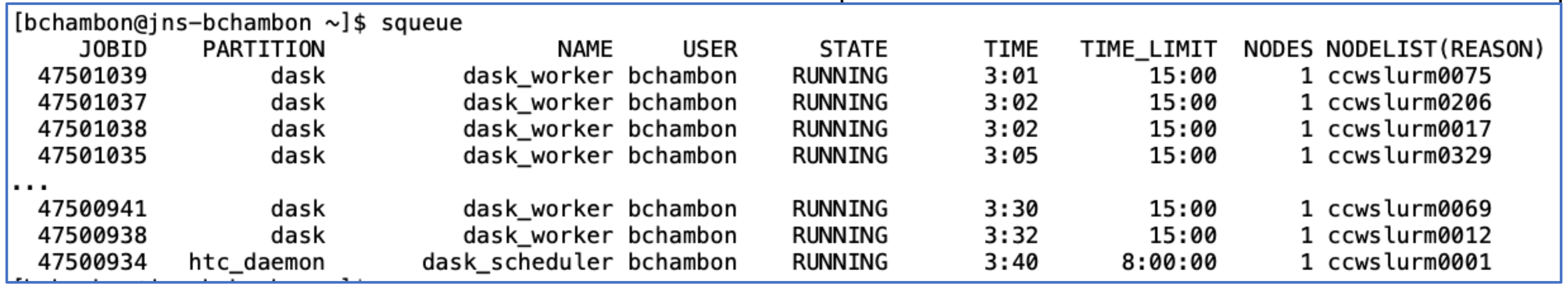
```
from dask4in2p3.dask4in2p3 import Dask4in2p3
my_dask4in2p3 = Dask4in2p3(virtual_env="/pbs/throng/ccin2p3/bchambon/venvs/venv4dask_Python3.10.10",
                           dask_scheduler_memory=2)
```

```
14:02:28,633 INFO    A 'Dask4in2p3' object has been created
```

```
# 100 jobs and for each job, 2 GB of memory and 15 mns of maximum duration
requested_dask_worker_jobs=100
client = my_dask4in2p3.new_client(dask_worker_jobs=requested_dask_worker_jobs,
                                  dask_worker_memory=2,
                                  dask_worker_time="00:15:00",
                                 )
```

```
14:02:28,641 INFO    Stopping dask-scheduler and dask-worker jobs, if any
14:02:28,690 INFO    No dask-scheduler nor dask-worker job was found, doing nothing
14:02:29,028 INFO    Creating and launching the SLURM jobs(s)
14:02:29,393 INFO    Waiting for the dask-scheduler SLURM job to be in RUNNING status, timeout=180s, step=5s
14:02:34,499 INFO    I've got the dask-scheduler SLURM job in RUNNING status
14:02:34,501 INFO    Waiting for the dask-worker SLURM job(s) to be in RUNNING status, for 100% of jobs, timeout=900s, step=10s
14:02:44,747 INFO    I've got 100 dask-worker SLURM job(s) in RUNNING status, which is greater or equal to the limit of 100% of
14:02:44,750 INFO    Connecting a dask-client, it may take a few seconds, timeout=600s
14:02:46,280 INFO    Success, a dask-client is connected to the dask-scheduler
```

2) Partial view , via squeue, of the SLURM jobs
***100** dask-workers + dask-scheduler*



```
[bchambon@jns-bchambon ~]$ squeue
     JOBID    PARTITION               NAME      USER      STATE       TIME   TIME_LIMIT  NODES NODELIST(REASON)
  47501039         dask        dask_worker  bchambon    RUNNING       3:01        15:00      1 ccwslurm0075
  47501037         dask        dask_worker  bchambon    RUNNING       3:02        15:00      1 ccwslurm0206
  47501038         dask        dask_worker  bchambon    RUNNING       3:02        15:00      1 ccwslurm0017
  47501035         dask        dask_worker  bchambon    RUNNING       3:05        15:00      1 ccwslurm0329
...
  47500941         dask        dask_worker  bchambon    RUNNING       3:30        15:00      1 ccwslurm0069
  47500938         dask        dask_worker  bchambon    RUNNING       3:32        15:00      1 ccwslurm0012
  47500934   htc_daemon     dask_scheduler  bchambon    RUNNING       3:40      8:00:00      1 ccwslurm0001
```

Summarising the demo (‘Dask+SLURM’) : 2/2

CCIN2P3

3) Defining and submitting the computing tasks

```python
# Defining the method to process a list of images
def do_segmentation(file_names):
  processing_durations=[]
  for file_name in file_names:
    try :
      image_name= file_name.split(os.sep)[-1] # image_name from absolute filepath
      logger.info(f"Start reading & processing image {image_name} ")

      t0 = time.time()
      # 1/
      src_images = dask_image.imread.imread(f"{file_name}") # src_images will be  dask.ar
      src_image = next(iter(src_images)) # To get the only one image of this array
      # 2/
      grayscale_image = grayscale(src_image)
      # 3/
      smoothed_image =  dask_image.ndfilters.gaussian_filter(grayscale_image, sigma=[2,2]
      # 4/
      threshold_value = 0.75 * da.max(smoothed_image)
      segmented_image = smoothed_image > threshold_value
      # 5/
      plt.imsave(f"{DEST_PATH}/segmented_{image_name}", segmented_image, cmap='gray')
```

```python
# Preparing a array of tasks (one task = do_segmentation function defined above)
#  getting the list of files
filename_list = glob.glob(f"{SRC_PATH}/*.jpg")
#  splitting the list of files according to dask_worker_jobs
filename_sublists = np.array_split(filename_list, min(dask_worker_jobs, len(filename_list)))

# Filling an array of tasks, with the (delayed)
#  do_segmentation() function defined above
from dask import delayed
tasks=[delayed(do_segmentation)(x.tolist()) for x in filename_sublists]
```

```python
try:
  t0 = time.time()
  logger.info(f"Submitting an array of {len(tasks)} computing task(s)
  futures = client.compute(tasks)
  results = client.gather(futures)
  overall_processing_duration = time.time() - t0
  logger.info(f"Results are available")
```

```
14:03:26,231 INFO    Submitting an array of 100 computing task(s) to 100 dask-worker(s), waiting
for the results...
14:03:36,461 INFO    Results are available
14:03:36,462 INFO    Average processing duration per dask-worker    5.89 s
14:03:36,463 INFO    Overall processing duration (including send tasks & recv results)  10.23 s
```

Feature 'Dask+SLURM' : About the package 'dask4in2p3'

- Package required on Jupyter notebooks side
  - Interaction with SLURM (sbatch, squeue, scancel) and connect the dask-client
  - Setup of the scripts of the dask-scheduler and dask-worker jobs
  - Managing certificate used to authenticate connections between dask-client, dask-scheduler, dask-worker
  - Managing directories etc/ et log/ (inside log/ are written stdout+err of SLURM jobs)
  - Tracking user's requirements and logging : number of jobs, amount of resources per job, etc.

This package is installed into the Docker image providing the notebooks server

- Package is also required on SLURM batch farm side
  - Must be installed into a Python virtual environment, specified via the constructor of the class Dask4in2p3

    my_dask4in2p3 = Dask4in2p3(virtual_env="/pbs/…")

  - But … there are ready-to-use Python virtual environment, for Python 3.8.5, 3.10.10 et 3.11.3

Python 3.8.5	/pbs/software/centos-7-x86_64/jnp/dask/venv4daskdemo	(This is the default one)
Python 3.10.10 | 3.11.3	/pbs/software/centos-7-x86_64/jnp/dask/venv4daskdemo_Python3.10.10 | 3.11.3	

The 'Dask+SLURM' feature can be tested without anything to install ☺