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

La plateforme des notebooks Jupyter au CC-IN2P3

Bernard CHAMBON, journées R&T 2023, Strasbourg, le 7 novembre 2023

Sommaire

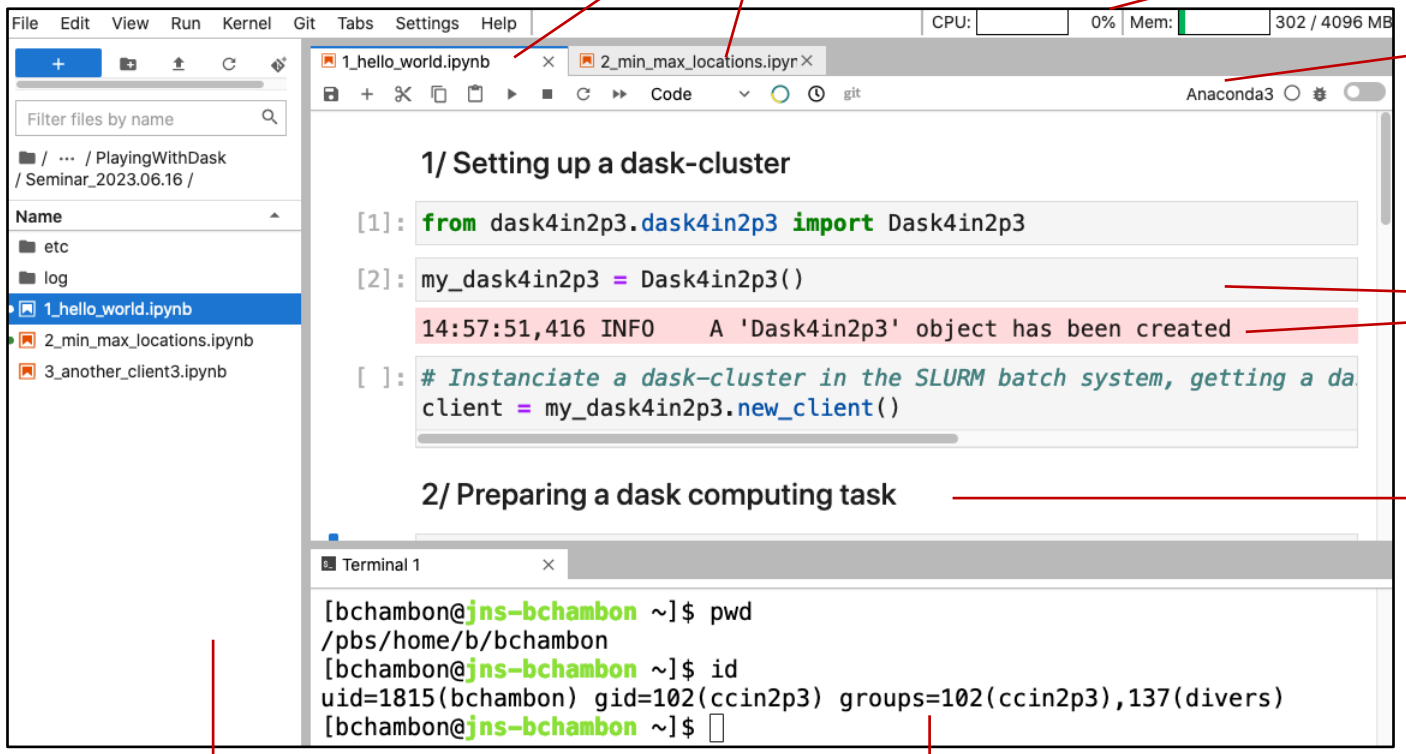
Introduction : 1/2

- Objectif de cette plateforme
  - Fournir un service d'analyse interactive, via les notebooks Jupyter
  - Avec accès aux même systèmes de stockage que ceux disponibles sur la plateforme d'accueil (cca.in2p3.fr)
  - Avec authentification via le SSO du CC-IN2P3

- Atouts des notebooks Jupyter
  - Simplicité
    - Navigateur WEB
    - Un même document pour du code, de la documentation, des résultats d'exécution
    - Disponibilité d'un terminal UNIX (sans faire ssh)
  - Diversité des langages utilisables, via les kernels (Jupyter = **Ju**lia, **Pyt**hon, **R)**
  - Richesse de l'écosystème via de nombreuses extensions

Introduction : 2/2

- Mon premier serveur de notebooks



2 notebooks dans un serveur de notebooks

Widget pour vue CPU et RAM

Vu du kernel (anaconda3)

Cellules de code et de résultat d'exécution du code (texte ou graphique)

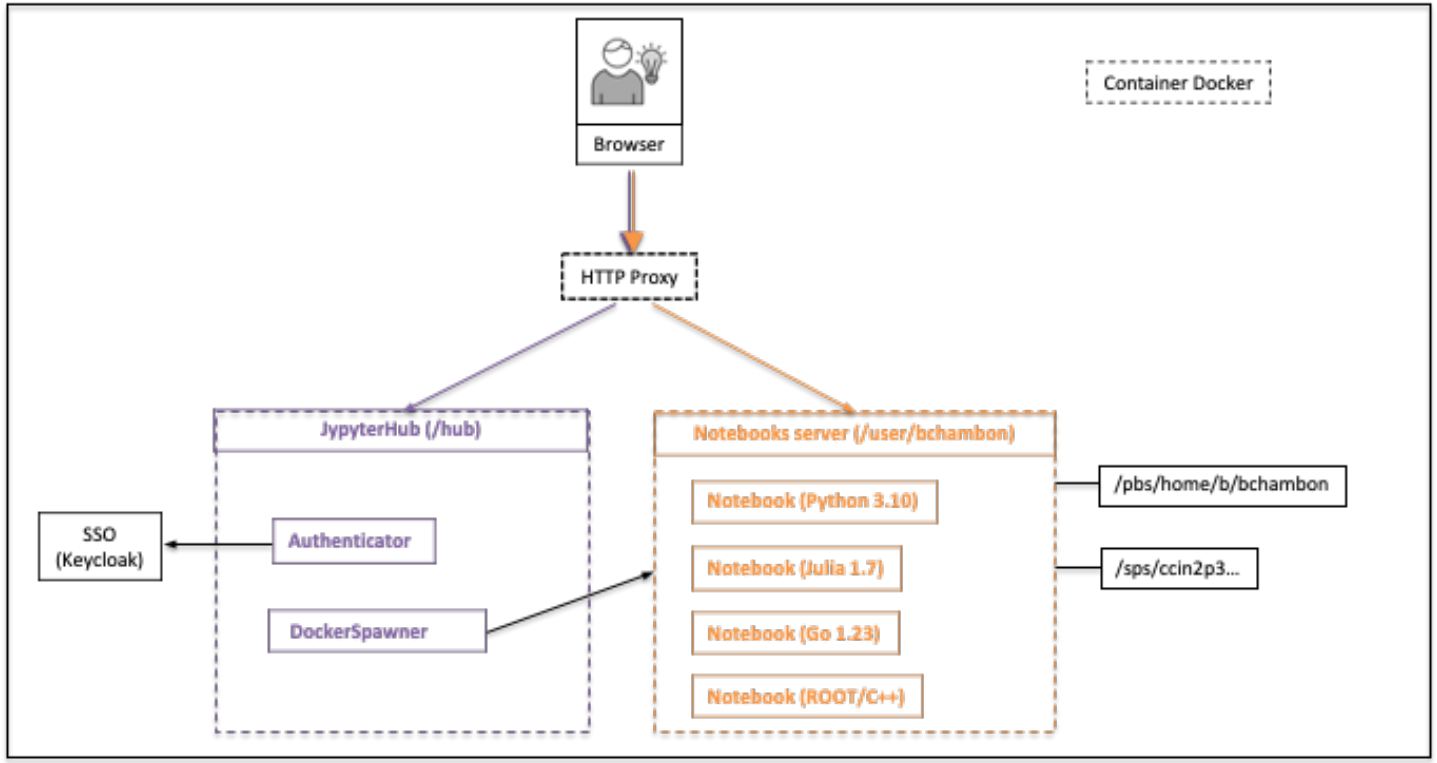
Documentation en markdown

Gestionnaire de fichiers

Terminal Unix

Architecture : 1/3

- Construite autour de **JupyterHub**
  - Composant permettant de définir l'authentification, de construire un formulaire d'options, d'instancier un serveur de notebooks
  - Configuration en Python



- Le service repose sur un cluster Docker avec l'orchestrateur Swarm pour placer les serveurs de notebooks sur les machines

Architecture : 2/3

- Accès au service
  - Ouvert à tout utilisateur disposant d'un compte calcul, mais certaines fonctionnalités nécessitent un accès privilégié (accès aux GPUs)

- Authentification
  - Délégation de l'authentification (OAuth) auprès de du SSO Keycloak (certificat ou login/MdP du compte 'calcul')
  - Ajout d'informations additionnelles (groupes secondaires, MdP expiré, compte bloqué ou expiré)

- A propos du serveur de notebooks
  - Containeur Docker basé sur une image préparée au CC et basée sur CentOS 7
  - Container s'exécute avec les IDs (uid, gid) de l'utilisateur
  - Disponibilités des systèmes de stockage
    - $HOME (/pbs/…), espaces de groupes (/sps/…) selon les groupes primaire et secondaires — Spécifique à chaque utilisateur
    - Espaces THRONG, SOFTWARE, CVMFS — Identique pour tous les utilisateurs

Architecture : 3/3

- Ressources RAM, CPU et durée d'utilisation
  - RAM
    - 2 GB par défaut, extensible sur demande, par logon ou par groupe (si plusieurs groupes on considère le max, logon prioritaire sur le groupe).
    - ~ 30 d'utilisateurs ayant 8, 16 ou 24 GB, quelques utilisateurs à 32 ou 64 GB
    - Un widget permet à l'utilisateur de voir la mémoire max disponible et la consommation instantanée
  - CPU
    - Pas de limitation en nombre de CPUs, mais
    - Un mécanisme de limitation est disponible et activable, par logon ou par groupe, en cas de consommation excessive
  - Durée d'utilisation
    - Pas de limitation, mais
    - Les serveurs de notebooks IDLE sont monitorés et arrêtés sur timeout de 3 | 1 jour(s) pour respectivement un serveur de notebooks CPU|GPU

- Monitoring des consommations
  - Utilisation de cAdvisor, Prometheus, Grafana pour la collecte, le stockage et la visualisation (à usage interne)

Présentation de 2 fonctionnalités

- Utilisation des GPUs

- Utilisation de ‘Dask+SLURM’

Utilisation des GPUs : 1/2

- Objectif
  - Permettre de faire de l'analyse en utilisant les GPUs

- Moyen
  - Disposer du droit d'accès à cette fonctionnalité. En faire la demande auprès du support du CC
  - Accès à un formulaire d'options permettant de choisir le modèle et le nombre de GPUs ainsi que la RAM du serveur de notebooks (voir slide suivant)

- L'utilisateur disposera alors
  - D'un serveur de notebooks restreint au nombre de GPUs choisis (max = 4, car sur une même machine)
  - Avec des frameworks + libs de machine learning (ML) déjà installés dans l'image Docker
    - TensorFlow + TensorBoard + TensorFlow Probability,
    - cuDNN
    - Pytorch
    - JAX (NumPy-like Python library)

  - Au 11/2023, on a 6 machines, modèle K80, de 4 GPUs /machine

Utilisation des GPUs : 2/2

Formulaire d'options pour GPU et instanciation d'un serveur de notebooks GPU

Formulaire d'options

My Notebooks Server Options

Compute engines ○ CPU Only ◉ GPU

Memory (GB) 28

GPU model(s) ◉ K80

GPU(s) number 2

Sélection de 28 GB de RAM

Sélection de 2 GPUs

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

Rappel de la config hardware et software relativement au modèle de GPU choisi

Launch My Notebooks Server

Un serveur de notebooks GPU instancié avec 2 GPUs et 28 GB de RAM

File Edit View Run Kernel Git Tabs Settings Help | CPU: 0% | Mem: 1.45 / 28.00 GB

demo.ipynb

Code Anaconda3

```
[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
```

Terminal 1

```
[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 ~]$
```

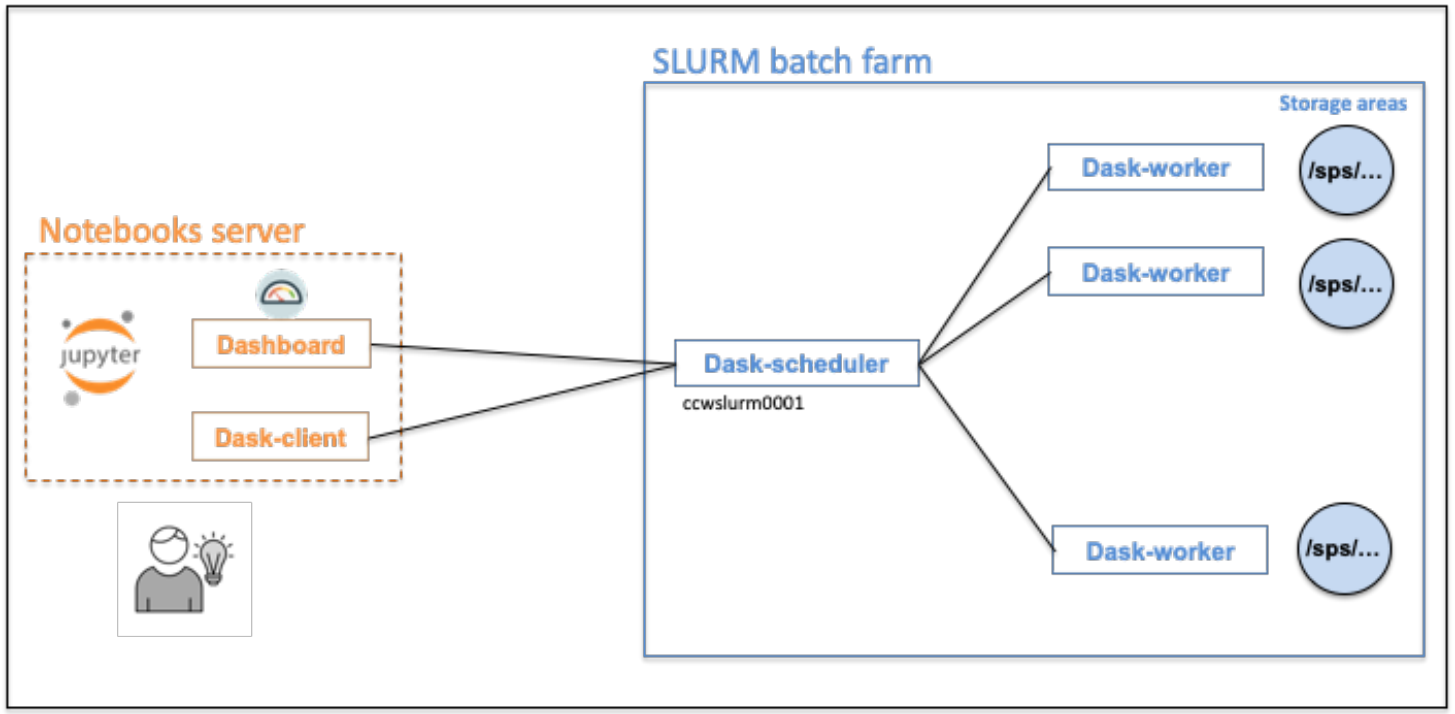
Fonctionnalité ‘Dask+SLURM’

- Objectif et architecture
- Comment l’utiliser
- *Démo*
- Cas d’usage d’utilisateurs

Fonctionnalité ‘Dask+SLURM’ : Architecture

- Objectif
  - Permettre l’analyse interactive de gros volumes de données via des traitements en parallèle
  - Depuis le notebook (pour l’interactivité) avec utilisation des ressources de la ferme de batch SLURM (pour la performance)
  - Par agrégation des ressources matérielles d'autant de machines distinctes que possible

- Architecture



Fonctionnalité ‘Dask+SLURM’ : Comment l’utiliser

- Moyen
  - Fonctionnalité non privilégiée (= disponible pour utilisateur de la plateforme des notebooks Jupyter)
  - Ecrire du code Dask (= en Python) et utiliser le package ‘dask4in2p3’ (déjà installé dans l’image Docker)

- L’utilisateur pourra spécifier :
  - Le nombre de jobs, la RAM et le temps de résidence des jobs SLURM (même valeur pour tous les jobs)
  - Un environnement virtuel (où le package ‘dask4in2p3’, est installé, car il est aussi nécessaire ‘coté SLURM’)
  - D’autres paramètres (timeouts, etc.) Voir les docstring des méthodes (*via Shift-Tab en pointant la méthode*)

Chaque paramètre a un valeur par défaut, donc testable sans rien préciser

- L’utilisateur disposera alors
  - D’un dask-client connecté au dask-scheduler
  - D’un dashboard fournissant des métriques relativement aux dask-workers (via l’extension JupyterLab ‘dask-labextension’)
  - De la possibilité de connecter d’autres dask-client(s) sur le dask-cluster existant (il n’y a qu’un dask-cluster /utilisateur)

Fonctionnalité ‘Dask+SLURM’ : *Démo*

- *Démo*
  - Traitement indépendant sur 200 images (détection de contours)
  - Objectifs de la démo
    - Impact de la parallélisation (100 jobs dans notre cas) pour diminuer le temps de traitement global
    - Disponibilité d’un dashboard remontant des métriques instantanées
    - Possibilité de connecter un autre dask-client (dans un autre notebook) mais adressant le même dask-cluster
  - Vidéo (muette) de 2’30

Fonctionnalité ‘Dask+SLURM’ : Cas d’usage d’utilisateurs

- Cas 1

Traitement des données issues d'une simulation cosmologique.

Dominique Boutigny, LAPP Annecy, Rubin/LSST

- Cas 2 :

Traitement d’images ZTF.

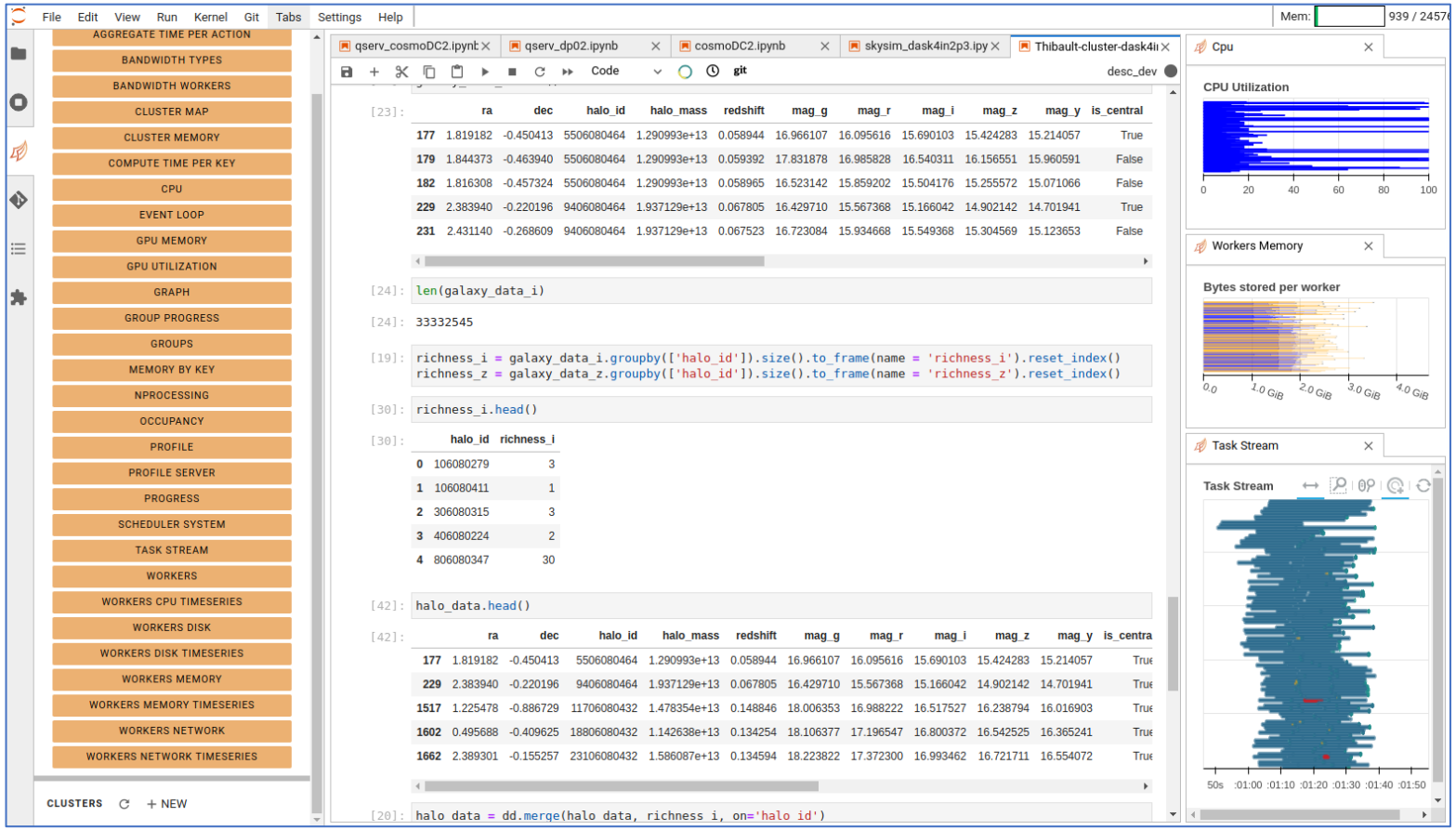
Mickaël Rigault, IP2I Lyon, ZTF

Cas 1 : Traiter des données issues d'une simulation cosmologique

Dominique Boutigny, LAPP Annecy, Rubin/LSST

L'objectif est de filtrer et enrichir un catalogue de galaxies issu d'une simulation cosmologique réaliste (simulation de l'évolution d'un univers dans le cadre d'un modèle cosmologique donné)
29 TB de données (1572 fichiers parquet de 20 GB / fichier)

*Le traitement est distribué sur **100 jobs SLURM** avec 5 GB de RAM par job*
Calcul exécuté en 7.5 minutes
À la fin on récupère un tableau de 5 millions d'amas de galaxies avec leurs caractéristiques



Cas 2 : Traitement d'images pour ZTF

Mickael Rigault, IP2I Lyon, ZTF

Traitement d'un grand nombre (50 10^6) d'images
*Déjà testé avec **500 jobs** et **2 GB** de RAM par job. Estimation à 1 jour de traitement avec **1000 jobs***
Intérêt :
*- **Voir en direct où j'en suis (bar de progression), les conso CPU / Memory / IO.***
*- **Voir en direct les résultats des jobs qui ont fonctionné***
*- **Savoir facilement lesquels ont craché, pourquoi et les relancer en direct si besoin / possible***

Analyse des SNeIa (type de supernova) de ZTF
Durée de quelques minutes pour traiter 4000 SNeIa avec 200 jobs et ~3GB /job
Intérêt : Interactivité, en diminuant le temps global de traitement

Etude du plan focal de ZTF et combinaison d'images
*Utilisation d'un outil (propre développement) pour traiter et afficher de manière **très fluide** avec 10 jobs et 1 GB /job*
Intérêt : Fluidité, grâce à l'agrégation de puissance des ressources de la ferme de batch

*Je me sers de Dask dès qu'il faut traiter des **tâches en parallèle** ou dès que j'ai des **besoins importants en terme de RAM***

***Outil** aussi **indispensable** que numpy ou pandas pour beaucoup de choses que je fais*

Mickael Rigault

Quelques chiffres sur l'infrastructure matérielle

- Matériel
  - 1 serveur : VM de 16 GB RAM, 8 CPUs. Où est déployé JupyterHub
  - 19 workers : Où s'exécutent les serveurs de notebooks
    - **11** VMs : 8 CPUs et 32 ou 64 GB RAM par VM
      - 7 VMs pour le computing
      - 4 VMs réservées aux formations
    - **8** machines physiques : 16 CPUs et 130 GB RAM par machine
      - 6 dédiées au computing sur GPU (modèle K80, 4 GPUs/machine)
      - 2 dédiées au computing sur CPU (pour des utilisateurs ayant des besoins en RAM et/ou E/S)
  - Cette infrastructure peut servir **100+** utilisateurs (CPU + GPU) dont **24** qui prendraient un seul GPU chacun
  - Extensible facilement par utilisation de VMs dans Openstack

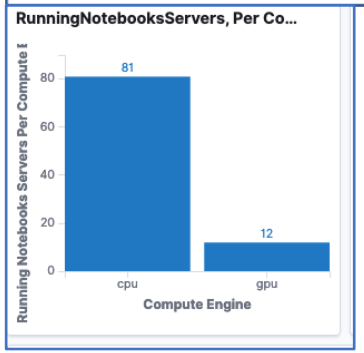
Quelques chiffres sur l'utilisation du service

Vue instantanée, mi-octobre 2023

***93** utilisateurs repartis **22** groupes et utilisant **13** machines*

Users	Groups	UsedWorkers
93	22	13

***81** utilisateurs de CPU; **12** de GPU (dont **4** en computing et **8** en 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

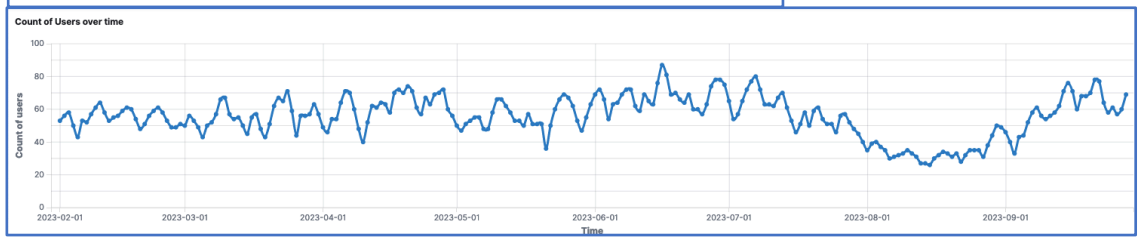
*Fonctionnalité Dask+SLURM, jusqu'à **1000** jobs (dask-workers)*

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

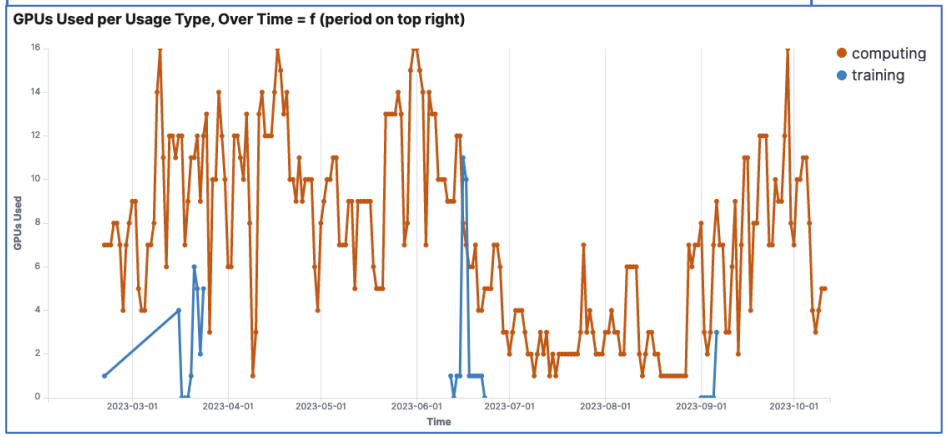
Vue sur 8 mois (du 1 février au 30 septembre 2023)

***300+** utilisateurs dans **58** groupes distincts*

*Nombre d'utilisateurs CPU + GPU (pic à **87** en juin 2023)*



*Utilisation des **GPUs** pour du calcul ou pour des formations*



Conclusion

- Un nouveau service du CC-IN2P3 (depuis juillet 2020)
  - Ouvert à tout utilisateur disposant d'un compte 'calcul' au CC
  - Configuré pour servir des besoins variées
    - Pour de l'analyse de données, pour faire des formations
    - Pour utiliser les ressources du service en CPU | GPU, ou via Dask, des ressources de la ferme de batch SLURM
  - Avec un support à l'écoute et réactif
- URLs
  - Consulter la documentation Jupyter Notebooks Platform
  - Accéder au service https://notebook.cc.in2p3.fr/
  - Contacter le support https://support.cc.in2p3.fr/
  - Fonctionnalité 'Dask+SLURM'
    - La documentation Dask usage
    - Le projet Dask4in2p3
    - Notebooks d'exemples Demodask4in2p3

Merci de votre attention

Annexes

- 2 screenshots de résumé de la démo

- Fonctionnalité ‘Dask+SLURM’ : A propos du package ‘dask4in2p3’

Résumé de la démo (‘Dask+SLURM’) : 1/2

*1) Instanciation d’un dask-cluster dans la ferme de batch SLURM, avec **100** dask-workers de 2 GB et 15 mns 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) La vue (partielle), via squeue, des jobs SLURM **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
```

Résumé de la démo (‘Dask+SLURM’) : 2/2

3) Définition et soumission des tâches de calcul

```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
```

Fonctionnalité ‘Dask+SLURM’ : A propos du package ‘dask4in2p3’

- Nécessaire coté de notebooks Jupyter
  - Interaction avec SLURM (sbatch, squeue, scancel) et connecter le dask-client
  - Préparation des scripts des jobs dask-scheduler et dask-worker
  - Gestion d’un certificat utilisé pour authentifier les connexions entre dask-client, dask-scheduler, dask-worker
  - Gestion des répertoires etc/ et log/ (dans log/ sont écrit les stdout+err des jobs SLURM)
  - Monitoring des spécifications de l’utilisateur en terme de nombre de jobs et leurs caractéristiques (RAM, CPU, temps)

Ce package est déjà disponible dans l’image Docker qui donne le serveur de notebooks

- Aussi nécessaire coté système de batch SLURM
  - Doit être installé dans un environnement virtuel Python, spécifié dans le constructeur

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

  - Mais … il existe des environnements virtuel Python tout prêts, pour Python 3.8.5, 3.10.10 et 3.11.3

Python 3.8.5	/pbs/software/centos-7-x86_64/jnp/dask/venv4daskdemo
Python 3.10.10 | 3.11.3	/pbs/software/centos-7-x86_64/jnp/dask/venv4daskdemo_Python3.10.10 | 3.11.3

La fonctionnalité ‘Dask+SLURM’ peut être testée sans rien avoir à installer ☺