TP HPC/Big Data

Dask

Supaéro
Module FITR35

Rédigé par
Gaëlle Usseglio / Annaïg Pedrono

2025/2026

Préparation du TP

Le code source se situe sur ce dépôt.
Vous pouvez le rapatrier en local via la commande *git clone*.

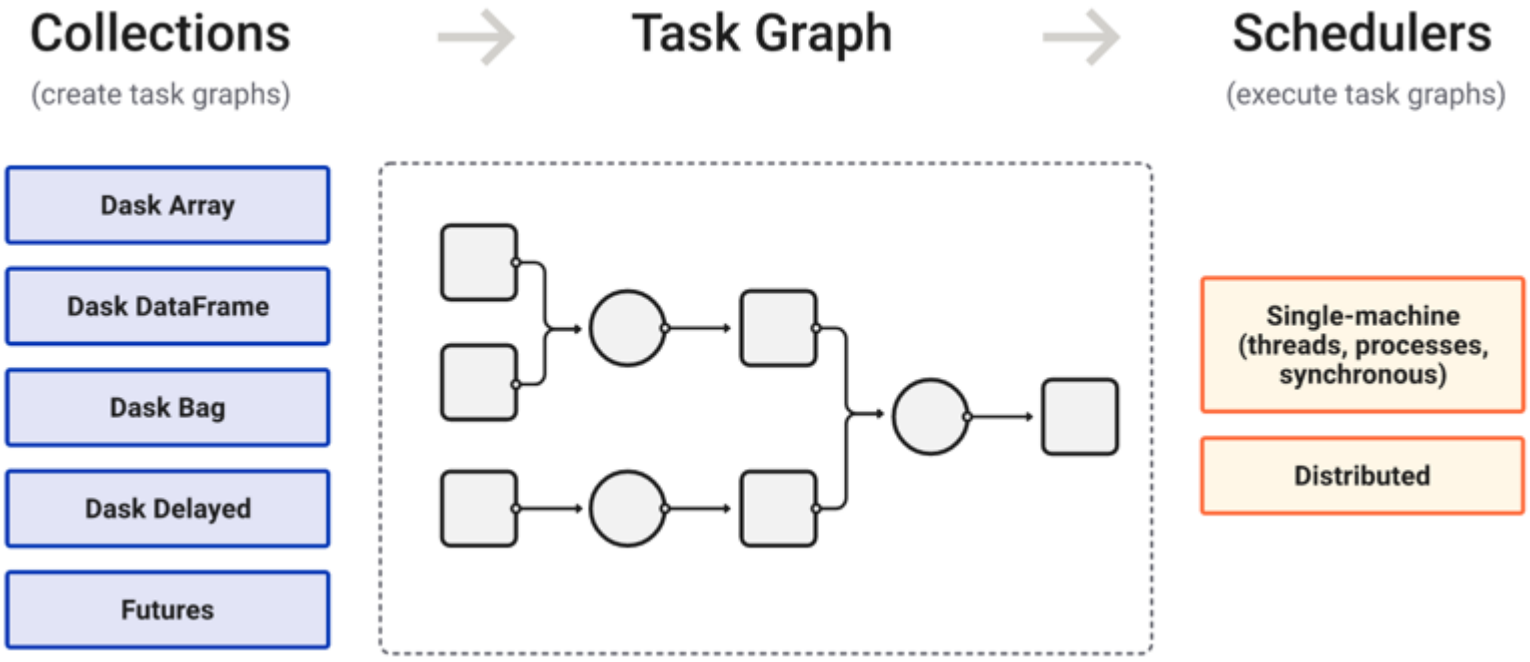
Le TP peut s'éxcuter sur le cluster TREX via l'URL https ://jupyterhub.cnes.fr. Plus d'explication dans la notice du JupyterHub CNES
Pour une utilisation sur le cluster TREX, la récupération du dépôt se fait via *git clone file ://///work/C3/formation-isae/repositories/TP_ Dask.git*. Une exception vous demandera, peut-être, d'ajouter ce réportoire comme "safe".

Dask

1 Rappel des concepts

Dask manipule les concepts suivants :

- Abstractions mémoire : Des collections/interfaces mises à disposition par Dask pour manipuler les données. Ces abstractions sont issues du "monde" Python avec les *Dask Array*, *Dask DataFrame* ou encore les *Delayed* et *Futures*.
- Notion de Lazy : Aucune exécution n'est réalisée jusqu'au déclenchement d'une action. Il s'agit de la différenciation entre transformations et actions. Une liste d'actions et tâches sont associées à chaque type collection.
- Graphe de tâches : Une graphe d'exécution est généré par Dask enchainant les tâches et les actions afin d'optimiser les traitements.
- Configuration Serveur/Client :
  - Serveur : Dask gère également l'exécution des graphes avec un Scheduler interne qui organise la réalisation des tâches sur un pool de ressources appelé Workers
  - Notion de client : APIs permettant de soumettre des actions sur les ressources (map, gather, sudmit ...)



2 Compilation - Run Time

Avant tout chose, il est important de vérifier son environnement.

- Première condition, avoir l'interpréteur Python ! Regarder si python est disponible.
- Seconde condition, installer Dask via *pip* ou *conda*. Il est possible de vérifier l'installation via les commandes *conda list* ou *pip list*.

Les principales APIs sont décrites dans le lien suivant **Dask cheat sheet**.

3 Exemples

Plusieurs exemples sont disponibles et permettent de mieux les abstractions de Dask (ne pas hésiter à se référer à la documentation de Dask) :

- *delayed* :
  - Voit-on une // parfaite ou y-a-t-il une dépendance entre les opérations ?
  - Est-il possible d'accumuler les delayed avant de lancer toutes les opérations d'un coup ?
  - Y-a-t-il une réutilisation des calculs précédemment fait s'ils sont redemandés ?
- *futures* :
  - Quelle est la différence entre delayed et futures ? Les calculs sont-ils lancés dès la déclarations du futures
  - Y-a-t-il un moyen d'attendre les fonctions du *Client* ?
- *array* :
  - A-t-on des APIs similaires entre np.array et dask.array ?
  - Comment la notion de *blocked algorithms* permet-elle de paralléliser des calculs et d'optimiser la consommation mémoire ?
- *dataframe* :
  - A-t-on des APIs similaires entre la librairie panda et les dask dataframe ?
  - De quoi les dask dataframe sont-ils contitués et quelle organisation trouve-t-on couramment ?
- *cluster* : différence entre cluster local et distribué

4 Exercices

Essayer de paralléliser les codes suivants :

- Utilisation des delayed et futures pour paralléliser une boucle **matricielle** : Iteration sur une liste d'entrée. Si l'entrée est paire alors la fonction *inc* est appelée sinon la fonction *double*. La fonction *is_ even* est rapide et n'a besoin d'être inclus dans le graphe de tâche.
- Analyse : Calcul du NDVI. L'indice de végétation permet de visualiser la santé des plantes/arbres par calcul de différence normalisée. L'objectif est de récupérer des images d'une même zone géographique prises à différentes dates.