Simulation paramétrique : Passer d'un job array à Dask

Guillaume Eynard-Bontemps, CNES

cnes
CENTRE NATIONAL D'ÉTUDES SPATIALES

Sommaire

Simulation paramétrique et LUT Sentinel3

Version Jobs, Jobs array et problèmes

Version Dask, avantages et limitations

Et le Cloud ?

Simulation paramétrique

Qu'est-ce que j'entends par là ?

- Un code de calcul, une méthode à appeler
- Sur des milliers ou millions de paramètres différents
- Consolider les résultats

Exemples

- Evaluer une fonction sur une grille de paramètres (GridSearch en machine learning)
- Simulation Monte-carlo, ou les paramètres sont tirés aléatoirement

L'exemple : Génération des LUT Sentinel avec Sixs

Le besoin métier

- Générer pour chaque bande spectrale de Sentinel 3 un fichier Look Up Table (LUT) permettant de lier le Pixel à la rétrodiffusion de la scène, utilisé à des fins de calibration

Environnement technique

- Cluster de calcul du CNES (HAL).
- Code : 6SV Routine : basic RT code used for calculation of lookup tables in the MODIS atmospheric correction algorithm.
- Scripts shell, job array via PBS Pro

Les stats

- 3123120 lancements par bande spectrale
- 654-700 hcpu par bande,
- soit 0,8s par lancement.
- 200Mo générés par bande

```
#Création des paramètres de simulation
lines = []
for u03 in tab_uO3:
    for uw in tab_uw:
        for ep_opt in tab_ep_opt:
            for psurf in tab_psurf:
                for alti in tab_alti:
                    for phi in tab_phi:
                        for thetas in tab_thetas:
                            for thetav in tab_thetav:
                                lines += [' '.join((runLUTtool, REP_in6SV,
```

Implémentation HPC : la naïve (ou stagiaire)

Sur mon PC, je fais une boucle for

- Et j'attends un mois ?

Sur le cluster, je fais une boucle for…

```
%%bash -s "$ficJobArrayTiny"
while read line; do
  qsub -- /bin/bash "$line"
done < $1
```

Et je reçois un mail du support… Heureusement, on vous limite...

Implémentation HPC : Le job array

3 phases :

- Génération des paramètres : création d'un fichier
- Lancement du job array sur les paramètres, chaque job produit un résultat.
- Récupération des résultats et ordonnancement pour produire le fichier LUT.

```
lines = []
for u03 in tab_u03:
    for uw in tab_uw:
        for ep_opt in tab_ep_opt:
            for psurf in tab_psurf:
                for alti in tab_alti:
                    for phi in tab_phi:
                        for thetas in tab_thetas:
                            for thetav in tab_thet
                                lines += [' '.join

open(ficJobArray, "w").write("\n".join(lines))
```

```
#!/bin/bash
#PBS -N jobArray
#PBS -J 0-20:1
#PBS -l select=1:ncpus=1:mem=1000mb
#PBS -l walltime=00:15:00
l_start=$PBS_ARRAY_INDEX
l_end=$PBS_ARRAY_INDEX
cmd="$(sed -n "${l_start},${l_end}p" /home
cd /home/eh/eynardbg/scratch/lut_work
eval ${cmd} > out_$PBS_ARRAY_INDEX
```

DEMO

```
result = []
for i in range(21):
    result += [open(work_dir + '/out_%d' % i, "r").readline()]
print("".join(result))
```

Implémentation HPC : Le job array les problèmes

Limitations niveau scheduler PBS :

- Une tâche 6SV dure en moyenne moins d'1s : jobs trop courts
- 3 millions de jobs dans l'array : trop de jobs
- Il faut regrouper plusieurs lancement dans un même job
- Gestion manuelle des erreurs (nœud planté, ou walltime atteint)

DoS attack

Et les résultats, je les récupère comment ?

- Un fichier créé par tâche… dans le système de fichier distribué.
- Relecture de tous les fichiers ensuite pour consolidation…

DDoS attack

Le code

- Beaucoup de shell rien que pour enchainer les différentes phases
- Encore plus pour gérer les cas en erreur

Implémentation Dask

4 phases :

- Génération des paramètres : en mémoire
- Réservation de nœuds pour lancer un cluster Dask
- Itération sur les paramètres et soumission de tâches
- Récupération des résultats ordonnés en mémoire

Démo

- Notebook
- Batch

Implémentation Dask : avantages et limites

Avantages :

- Plus de surcharge PBS, des jobs long
- Pas d'impact sur le système de fichier, tous les échanges se font via réseau
- Gère très bien les tâches d'une seconde.
- Code propre et simple, un script gère l'ensemble.
- Portabilité (cf la suite)
- Dask relance une tâche en erreur un nombre de fois configurable

Limites :

- 3M tâches, c'est trop aussi pour Dask
  - On peut, avec une ligne de plus, regrouper les traitements.
- Dask-jobqueue, c'est bien, mais comment peut-on s'assurer d'avoir les ressources le temps qu'il faut ?
  - Adaptive Mode et as_completed, attention à la durée du job parent qui démarre le Scheduler
  - Dask-mpi !

Implémentation Dask : le code en batch

```
logging.info('Création des paramètres de simulation')
lines = []
for u03 in tab_u03:
    for uw in tab_uw:
        for ep_opt in tab_ep_opt:
            for psurf in tab_psurf:
                for alti in tab_alti:
                    for phi in tab_phi:
                        for thetas in tab_thetas:
                            for thetav in tab_thetav:
                                lines += [' '.join((runLUTtool, REP_in6SV, satellite, capteur, bande, AER_NOM,
                                                    AER_TYPE, thetav, phi, thetas, ep_opt, uw, u03, psurf, alti,
                                                    NOMFICBANDE, REP_out6SV, REP_CODE_TR, REP_WORK))]
logging.info('%d paramètre de lancement générés', len(lines))

logging.info('Démarrage d\'un cluster Dask')
cluster = PBSCluster(processes=1, cores=4, memory="10GB", local_directory='$TMPDIR',
                     project='Dask_LUT_demo_batch', walltime='06:00:00', interface='ib0',
                     name='Dask_LUT_demo_batch')
#On veut un cluster de 20 noeuds, soit 480 coeurs, soit 480/4=120 process workers avec la configuration "bouche trou
sus
cluster.scale(120)

logging.info('Connexion d\'un client au cluster. A ne pas oublier sinon on va tout lancer en local!')
# Timeout plus long pour laisser le temps au cluster de démarrer
client = Client(cluster, timeout=10)

logging.info('Lancement du calcul')
# Disons qu'on veut environ 100 enregistrements par batch
npartitions = len(lines) // 100 + 1
b = db.from_sequence(lines, npartitions=npartitions)
result = b.map(run_process).compute()
print(len(result))

logging.info('Ecriture du résultat')
result_df = pd.read_csv(io.StringIO(''.join(result)), delim_whitespace=True, header=None, dtype='str')
result_path = work_dir + ('/OUTLUT_%s_%s_%s.csv' % (satellite, capteur, bande))
result_df.to_csv(result_path, sep=' ', index=False, header=False)
```

Implémentation Dask : et dans le cloud ?

Prérequis :

- Une plateforme Pangeo dans le cloud
  - Cluster Kubernetes (de préférence avec autoscaling)
  - Helm chart Pangeo (Jupyterhub + autorisation RBAC pour Dask)

Ajouts/modifications :

- Une image Docker contenant 6SV
- Remplacer PBSCluster par KubeCluster

Démo Binder : https://github.com/guillaumeeb/demo-lut-cloud