mpi (équivaut à 1 machine avec **28 coeurs** avec maximum **115g** de RAM)	HPC #PBS -q mpi #PBS -l walltime=00:05:00 #PBS -l select=1:ncpus=28:mem=115g	HPC #PBS -q mpi #PBS -l walltime=48:00:00 #PBS -l select=1:ncpus=28:mem=115g

Pour des informations plus précises, reportez-vous à la configuration détaillée des queues.

Exemples de scripts

Vous trouverez des exemples de scripts PBS sur Datarmor dans /appli/services/exemples/pbs et /appli/bioinfo/modeles-de-script

Comment connaître les besoins de vos calculs ?

Sauf si la documentation donne des informations précises, ce qui est très rarement le cas malheureusement, il n'y a pas de réponse simple à cette question.

Méthode 1 (qstat) :

Le mieux est de lancer un premier job avec vos données (ou une partie de celles-ci) et, une fois celui terminé, d'utiliser cette commande :

```
qstat -fx <id-de-job>
```

Regardez alors les trois variables :

- "**resources_used.cpupercent**" : % d'utilisation du CPU ; si >100%, alors le code est parallélisé et vous pouvez utiliser une directive "#PBS -l ncpus=xxx" (si ncpus>1, alors utilisez obligatoirement la queue "omp")
- "**resources_used.mem**" : RAM max utilisée ; cela vous permet de configurer la directive "#PBS -l mem=xxx"
- "**resources_used.walltime**" : temps d'exécution ; cela vous permet de configurer la directive "#PBS -l walltime=xxx"

Exemple :

Si vous lancez votre job sans demander de parallélisation (queue sequentiel), vous pouvez regarder le résultat de la commande "qstat -fx <mon-id-job> :

```
Job Id: 1044016.datarmor0
    resources_used.cpupercent = 527 <-- votre job a utilisé 527 % de cpus
(ce qui équivaut à utiliser 6 cpus), il est donc parallélisable !
    resources_used.walltime = 12:13:12 <-- votre job a duré 12 heures 13
minutes et 12 secondes
    resources_used.mem = 54414096kb <-- votre job a utilisé 54Go de mémoire
```

Le cas précédent vous permet d'ajuster votre configuration. Il aurait donc fallu saisir les paramètres suivants dans votre script PBS :

```
#PBS -q omp <-- pour préciser qu'on va paralléliser le job
#PBS -l ncpus=6 <-- on aura besoin de 6 cpus
#PBS -l walltime=14:00:00 <-- moins de 14 heures d'éxécution
#PBS -l mem=60g <-- et maximum 60Go de mémoire
```

Méthode 2 (pbs-report) :

Voir les ressources utilisées par vos jobs terminés depuis telle date (-b date au format YYYYMMDD) :

```
pbs-report -u lquintri -b 20180419 -v | awk '$2~/lquintri/{print "Job Id :"
$1 "\tMemory Used : " $10 "\tCPU time used : " $12 "s\tWalltime : " $13
"s\tAverage CPU really used : " $12/$13}'
```

Par exemple pour le job 1063390, on a demandé la configuration suivante :

```
#PBS -q omp
#PBS -l walltime=48:00:00
#PBS -l mem=400g
#PBS -l ncpus=12
```

Si on regarde les ressources utilisées (résultat de la commande pbs-resport ci-dessus) :

Job Id : 1063390 **Memory Used** : 26347500kb **CPU time used** : 856153s **Walltime** : 71519s **Average CPU really used** : 11.971

- On a utilisé 26gb sur les 400gb demandés
- On a utilisé 11.971 de CPUS sur les 12 demandés
- Le job a duré 71519/3600 = 19 heures sur les 48 heures demandées

La configuration idéale aurait donc été :

```
#PBS -q omp
#PBS -l walltime=24:00:00
#PBS -l mem=30g
#PBS -l ncpus=12
```

Monitoring des jobs PBS

Avoir les informations sur un job : qstat -fx numero_job

```
qstat -fx 59754.datarmor0
```

Voir les logs du job :

```
tracejob 59754.datarmor0
```

Supprimer un job : qdel numero_job

```
qdel 59754.datarmor0
```

Vos statistiques d'utilisation du cluster sur un intervalle de temps : pbs-report -u login -b BEGINDATE -e ENDDATE

```
pbs-report -u lquintri -b 20170610 -e 20170620
```

Voir l'ensemble de ses jobs en cours : qstat -u login-name

```
qstat -u lquintri
```

Voir l'ensemble de ses jobs (en cours + historique) : qstat -u login-name -x

```
qstat -u lquintri -x
```

Voir sur quels noeuds de calcul nos jobs s'exécutent :

```
qstat -t -n | grep -A 1 lquintri
```

Voir l'ensemble des jobs qui tournent sur le cluster :

```
qstat -t
```

ou

```
wo
```

Technique avancée : les jobs chaînés

Avec PBS, il est possible de lancer des jobs chaînés, qui s'exécutent les uns après les autres : le job n se lance à la fin du job (n-1).

Exemple avec trois jobs :

On soumet le job 1 en le gelant (option -h) ; puis, on soumet les jobs 2 et 3 en leur appliquant une condition (option depend), avant de dégeler le job 1 avec la commande qrls (queue release)

```
qsub -h -N Job1 script1

qsub -N Job2 -W depend=afterany:'qselect -N Job1 -u $USER' script2
qsub -N Job3 -W depend=afterany:'qselect -N Job1 -u $USER' -W
depend=afterany:'qselect -N Job2 -u $USER' script3

qrls 'qselect -N Job1 -u $USER'
```

Ainsi, le job 3 ne s'exécutera qu'après le job 2, lui-même exécuté après le job 1, peu importe l'état de sortie des jobs.

Le paramètre *depend* peut prendre différentes valeurs :

- *afterany* : le job s'exécute quelque soit la terminaison du précédent
- *afterok* : le job ne s'exécute que si le précédent s'est terminé sans erreur
- *afternotok*: le job ne s'exécute que si le précédent s'est terminé avec erreur

Si vous utilisez afterok ou afternotok, vérifiez que vos jobs ne restent pas en attente du fait que la dépendance ne soit pas satisfaite

Partager

Dernière modification le 11/05/2020