TP Slurm 1/2

Nom et prénom :

Veuillez recopier les résultats des commandes exécutées (et/ou inclure des captures d'écran) et répondre aux questions de ce document. Les réponses peuvent être rendues dans le format de votre choix (Word,PDF séparé, papier puis scanné), tant que les numérotations des questions/réponses sont respectées.

Le document contenant les réponses est à envoyer à ***lise.jolicoeur@cea.fr*** *en fin de TP* ***ET****, si non terminé, avant le 06/02/2023 à 12h.*

Merci d'indique votre NOM dans le nom de fichier du document envoyé.

.0. Préparation et validation de l'environnement

Dans cette introduction, nous allons démarrer 4 machines virtuelles :

- **vm0** comme leader et worker slurm
- **vm[1-3]** comme workers slurm seulement

1. Démarrer **4** machines virtuelles avec PCOCC avec le *template* du TP Slurm 1.

```
pcocc alloc -c2 mycentos74-tp-slurm1:4
```

2. Se connecter à la première machine virtuelle **vm0** et vérifier l'état du cluster avec la commande **sinfo -l**.

```
pcocc ssh vm0  (long, si n'aboutit pas ctrl+C et relancer deuxième fois un peu plus tard)
```

3. Décrire le cluster d'après la sortie de la commande sinfo -l

4. Vérifier le bon fonctionnement de **slurm** en réalisant une première exécution interactive.

```
srun -n 4 -N 4 hostname
```

```
si message d’erreur srun: error: Unable to allocate resources: Invalid account ...ajouter votre login en tant que user avec la commande qui suit :

sudo  sacctmgr -i add user name=«prenom.nom» cluster=ensiie account=guests fairshare=10 qos=normal,debug
```

.1. Premier script batch

Dans un premier temps nous allons réaliser un premier script shell à utiliser dans un job batch.

1. Créer un premier script affichant le nom du nœud host ‘hostname’ et se terminant avec un code retour 0. L’exécuter directement dans la session en cours.

```
vim script-1.sh

chmod +x script-1.sh ; ./script-1.sh
```

2. L’exécuter à l’aide de la commande **sbatch** puis observer le contenu du répertoire courant et l’état des statistiques d’utilisation de **slurm.**

```
sbatch script-1.sh

remarquer le numéro du job (%jobid) « Submitted batch job no »

ls -ltr | tail

sacct -j %jobid obtenu au moment du sbatch%

cat slurm-%jobid.out
```

3. Modifier le script puis l’exécuter afin d’obtenir un état « FAILED » plutôt que l’état « COMPLETED ».

```
vim script-1.sh

cat script-1.sh ; sbatch script-1.sh

sacct -j %jobid%
```

4. Le modifier à nouveau pour retrouver un code d’erreur « COMPLETED » puis demander l’allocation de 4 cœurs de calcul et constater la différence au niveau

de l'accounting.

```
sbatch -n 4 script-1.sh
sacct -j%jobid%
```

.2. Intégration de « step » parallèles

L’exécution d’un script batch demandant l’allocation de plusieurs cœurs nécessite le lancement d’un « jobstep » afin de pouvoir les utiliser en parallèle. Le lancement d’un step se fait par l’intermédiaire d’une commande **srun** intégrée dans le script batch.

Vérifier que plus rien ne tourne entre chaque partie.

1. Dupliquer le premier script batch et le modifier pour intégrer l’exécution d’un step affichant le nom des machines. Ensuite, afficher le contenu du « listing » obtenu et les données d’accounting. Commenter les résultats pour nodelist et alloccpus.

```
cp script-1.sh script-2.1.sh
vim script-2.1.sh
cat script-2.1.sh
sbatch -n 4 script-2.1.sh
cat %fichier_listing%
sacct -j %jobid% -o jobid,jobname,nodelist,alloccpus,state
```

2. Dupliquer le script précédent et ajouter un second step exécutant une attente de 120 secondes (sleep 120). Constater et expliquer le déroulement de l’exécution à l’aide des commandes squeue et sacct puis afficher les données d'accounting une fois le job batch terminé.

```
cp script-2.1.sh script-2.2.sh
vim script-2.2.sh
cat script-2.2.sh
sbatch -n 4 script-2.2.sh
squeue -s -j %jobid%
sacct -j%jobid%
squeue -s -i 30 -j %jobid%
sacct -j %jobid%
sacct -j  %jobid% -o jobid,jobname,nodelist,alloccpus,state,start,end,elapsed
```

3. Dupliquer le script précédent et remplacer les deux steps par une boucle de 3 step d'attente de 120 secondes. Le lancer et constater le fonctionnement et utiliser la commande **scancel** pour arrêter prématurément les 3 steps. Expliquer la sortie de sacct.

```
cp script-2.2.sh script-2.3.sh

vim script-2.3.sh

cat script-2.3.sh

sbatch -n 4 script-2.3.sh

squeue -s -j %jobid%

scancel %jobid%.0

sacct -j %jobid%

scancel %jobid %

sacct -j%jobid % (-o state%30)
```

.3. Utilisation des QOS

L'objectif est ici de comprendre le comportement des politiques d'ordonnancement en utilisant différentes QOS pour favoriser le démarrage de certains jobs. Le cluster Slurm du TP est configuré de manière à autoriser 2 QOS, une par défaut et une optionnelle. Vérifier que plus rien ne tourne entre chaque partie.

1. Vérifier l'état de l'élément **slurmdbd** nécessaire au fonctionnement de sacctmgr. Décrire les différences entre les deux QOS. La/Lesquell/es pouvez-vous utiliser ? Quelle est votre QOS par défaut ? Vérifier que c'est bien le QOS par défaut.

   Le script 'parse_sacctmgr.sh' dans /home/shared/lise.jolicoeur permet de visualiser de manière plus lisible la sortie de la commande 'sacctmgr'. N'hésitez pas à le copier dans votre VM (pcocc scp) et à l'utiliser/le modifier (ex : ./parse_sacctmgr.sh show qos)

```
systemctl status slurmdbd

sacctmgr show qos

sacctmgr show assoc format=cluster,account,user%30,qos,defaultqos

srun -n 1 hostname

sacct -j %jobid% -o jobid,user,jobname,qos
```

2. Lancer deux fois de suite le script script-2.2.sh en demandant une allocation de ressources « exclusive » de 4 tâches sur 4 nœuds, puis une troisième fois en demandant l'utilisation de la QOS « debug ». Constater les priorités et expliquer l'ordre d'exécution des 3 lancements à l'aide des commandes « sprio » et « squeue ».

```
sbatch -n4 -N4 --exclusive script-2.2.sh
sbatch -n4 -N4 --exclusive script-2.2.sh
sbatch -n4 -N4 --exclusive --qos debug script-2.2.sh
squeue -O jobid,state,qos,timeused
sprio
```

3. Lancer maintenant deux fois de suite le même script **sans** allocation exclusive de ressources et **sans** utiliser la QOS debug. Que constatez-vous ?

```
sbatch -n4 -N4 script-2.2.sh
sbatch -n4 -N4 script-2.2.sh
squeue -O jobid,qos,timeused,state,reason
```

4. Faites la même chose en utilisant la QOS debug. Que constatez-vous ? Pourquoi ?

```
sbatch -n4 -N4 --qos debug script-2.2.sh
sbatch -n4 -N4 --qos debug script-2.2.sh
squeue -O jobid,qos,timeused,state,reason
```

.4. Comportements aux limites

L'idée est maintenant de constater le comportement de Slurm lorsqu'un job batch ne se comporte pas de la façon attendue ou que le système lui même introduit des problèmes. Vérifier que plus rien ne tourne entre chaque partie.

1. Lancer le script précédent en demandant un temps d'allocation de ressources de 1 minute seulement. Constater le comportement à l'aide des commandes squeue et sacct jusqu'à sa terminaison. Quel est le « state » associé au job une fois terminé ? Combien de steps ont été exécutées ?

```
 sbatch -t 1 -n 4 script-2.3.sh
…
```

2. Lancer un script consommant 128Mo de mémoire dans un step en spécifiant

une demande de 100Mo de mémoire seulement. Constater et expliquer les états de sortie ainsi que le contenu du listing obtenu.

```
% cat gen100M.sh
#!/bin/bash
srun dd if=/dev/zero of=/dev/shm/128Mo bs=1M count=128
sleep 30
rm /dev/shm/128Mo
exit 0
% sbatch --mem-per-cpu=100 -n1 gen100M.sh
% squeue
% sacct
…
```

3. Lancer le script script-2.3.sh sans restriction de temps d'exécution ni de mémoire puis provoquer le reboot de la vm1.

```
sbatch -n 4 script-2.3.sh
sudo ssh vm1 reboot
```

Constater le comportement à l'aide des commandes squeue et sacct jusqu'à sa terminaison. Quel est le « state » associé au job une fois terminé ?

```
Sinfo, squeue, scontrol show jobid=%jobid (au cours du job)
sacct -j %jobid
```

.5. Gestion des nœuds

L'état actuel du cluster n'est pas complètement opérationnel. Le reboot de vm1 à changé son état et ne le rend plus éligible à l'allocation. Il est donc nécessaire de corriger ce problème pour pouvoir à nouveau l'utiliser. Vérifier que plus rien ne tourne entre chaque partie.

1. Lancer un script nécessitant les 4 nœuds exclusivement et constater l'état du job à l'aide de la commande :

```
sinfo
```

```
squeue
```

2. Remettre le nœud vm1 en « production » à l’aide de la commande scontrol et commenter les conséquences sur l’état du job précédemment lancé.

```
sinfo -Rl

sudo scontrol update nodename=vm1 state=idle
sinfo

squeue
scontrol show jobid=%jobid (au cours du job)
```

3. Réaliser la même opération de 4.3 mais en ajoutant le paramètre « --requeue » (voir « man sbatch ») à sbatch au moment de la soumission. Remettre ensuite le nœud en état de « production ». Que constatez-vous pour le job? Quelle est la différence avec 4.3 sans « --requeue » ?

   Décrivez les commandes utilisées afin d’observer l’état du cluster et leurs sorties.

4. Lancer deux fois de suite le script script-2.2.sh en demandant une allocation de ressources « exclusive » de 4 tâches sur 4 nœuds, Puis « drainer » un nœud à l’aide de la commande scontrol. Décrire l’état des nœuds à l’aide de la commande sinfo avant et après la fin du premier job et l’état du second job une fois le premier terminé. Remettre ensuite le nœud « drained » en production (regarder dans « man scontrol » pour trouver les syntaxes exactes).

   Décrivez les commandes utilisées afin d’observer l’état du cluster et leurs sorties.

5. Lancer le script script-2.2.sh en demandant une allocation de ressources de 4 tâches sur 4 nœuds puis passer un des nœuds à l’état « down ». Donner l’état des nœuds. Que constatez-vous sur l’état du job ?

   Décrivez les commandes utilisées afin d’observer l’état du cluster et leurs sorties .

6. En déduire d’après 4 et 5 la différence entre mettre « down » un nœud et « drainer » un nœud. Remettre ensuite le nœud en état de « production ».

.6. Test du Backfilling

Le cluster est pleinement opérationnel sans aucun job en cours d’exécution. Il est maintenant possible de monter un scénario permettant d’illustrer le fonctionnement du « backfilling ». Vérifier que plus rien ne tourne entre chaque partie. Pour cela, il suffit de créer:

- 1 job « bkf-1.sh » nécessitant une partie des ressources pour une période de temps relativement longue.
- 1 job « bkf-2.sh » nécessitant l’intégralité des ressources du cluster et de

grande priorité.

- 1 job « bkf-3.sh » moins prioritaire que le second nécessitant l'espace non occupé (ou moins) du premier job pour une plage de temps suffisamment longue pour ne pas être « backfilled ».
- 1 job « bkf-4.sh » moins prioritaire que le second nécessitant l'espace non occupé (ou moins) du premier job pour une plage de temps suffisamment courte pour être « backfilled ».

1. Mettre en place le scénario illustrant le principe du « backfilling » :

   Lancer les 4 jobs dans l'ordre qui vous semble approprié pour créer un scénario de « backfilling ». Afficher les priorités des jobs en attente avec sprio pour vérifier la plus grande priorité du 2ième job.

   Fournissez l'ordre de passage des 4 jobs pour montrer que le 4ieme job a été « backfillé » (passé avant un job plus prioritaire). Décrivez les commandes utilisées afin d'observer l'état du cluster. N'hésitez pas à inclure la sortie des commandes effectuées et/ou des impressions d'écran montrant la suite des évènements.
2. Réaliser à nouveau le scénario en annulant l'exécution du premier job une fois la mise en exécution d'un nouveau job. Qu'en déduisez-vous sur les limitations du « backfilling » ?
3. Réaliser à nouveau le scénario sans spécifier les temps d'allocation souhaités. Qu'en déduisez-vous sur les besoins du « backfilling » ?