Utilisation du cluster du LIMOS

Hélène Toussaint, 30 mai 2017

Table des matières

1 Présentation générale

Un cluster est un regroupement de machines dédiées au calcul. Il est composé d'une machine frontale ("porte d'entrée" du cluster) et de plusieurs machines de calcul, appelées *nœuds,* et éventuellement hétérogènes.

Le cluster du LIMOS dispose actuellement de 13 nœuds qui représentent un total de 272 cœurs physiques. Il dispose d'un gestionnaire de ressources SLURM qui permet à un utilisateur de réserver une partie des ressources de calcul totales (CPU, RAM) en fonction de ses besoins pour un temps donné.

Remarque : Les processus d'un utilisateur sont confinés aux ressources (CPU notamment) qu'il a réservées : il ne peut pas accéder aux CPU réservés par d'autres utilisateurs. Cela permet d'assurer une certaine reproductibilité des exécutions d'un même programme, notamment au niveau des temps de calcul (contrairement à ce qu'il se passe sur des machines en libre accès).

1.1 Composition actuelle du cluster du LIMOS

Différentes machines de calcul ont été groupées pour former le cluster : 11 machines appelées *node* et numérotées de 22 à 32, ainsi que 4 machines hétérogènes.

Le tableau suivant donne des informations sur chacune de ces machines.

Noeuds	nb coeurs physiques	hyperthread activé ?	proc	Cache	RAM
node22 à node29	16	non	intel Xeon E5-2670 2.6 GHz	20 Mo	62.5 Go
node30 à node32	20	non	IntelXeonCPU E5-2670 v2 @ 2.50GHz	25 Mo	62.5 Go
molene168	16	non	Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz	20 Mo	31 Go
dellware	16	oui -> 32 coeurs logiques	Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz	20 Mo	386 Go
kephren	72	non	Intel(R) Xeon(R) CPU E7-8890 v3 @ 2.50GHz	46 Mo	3To
bob	24	non	Intel(R) Xeon(R) E5-2687Wv4 @ 3.00GHz	30 Mo	768 Go

Tableau 1: Machines composant le cluster du LIMOS

Remarque : node27 et node29 sont actuellement indisponibles.

1.2 Accès et utilisation

On accède au cluster en se connectant (en ssh) sur le serveur (frontal) frontalhpc.rcisima.isima.fr et en utilisant le login / mot de passe de son compte habituel. Pour exécuter un programme sur un nœud

de calcul, l'utilisateur doit obligatoirement passer par l'ordonnanceur de tâches SLURM (voir section 2).

Remarque 1 : on se connecte pas directement sur les nœuds

Remarque 2 : le frontal n'est pas une machine de calcul, il sert seulement à la connexion et éventuellement à la compilation.

2 Utilisation du gestionnaire de ressources SLURM

2.1 Qu'est-ce que SLURM ?

SLURM (*Simple Linux Utility for Resource Management)* est un gestionnaire de ressources et ordonnanceur de tâches pour des clusters LINUX. Il permet de répartir au mieux les ressources de calcul (CPU, GPU, RAM) entre utilisateurs en gérant des files d'attente avec priorité.

Une documentation complète est disponible sur le site de SLURM http://slurm.schedmd.com/.

Le vocabulaire :

1. **Les nœuds et CPUs**

- Un **nœud** de calcul correspond à une machine de calcul du cluster (i.e. une machine physique),
- un nœud contient des **processeurs physiques** (*sockets*) contenant eux-mêmes des **cœurs physiques** (*cores*),
- dans le cas où l'hyperthreading est activé sur le nœud alors chaque cœur physique donne 2 **cœurs logiques** (*threads*). Autrement dit, si l'hyperthreading est activé alors un cœur physique permet d'exécuter 2 threads en parallèle.

Lorsqu'on parle de **CPU** dans SLURM on fait référence à une unité de calcul logique (i.e. un cœur logique), donc :

- sans l'hyperthreading : 1 cœur physique = 1 CPU SLURM
- avec l'hyperthreading : 1 cœur physique = 2 CPU SLURM

2. **Les jobs et les tasks**

Dans SLURM un **job** correspond à une requête d'allocation de ressources (CPU, RAM, temps de calcul) par un utilisateur. Un job est composé d'étapes (*steps*) et chaque étape effectue une tâche (*task*) ou plusieurs en parallèle. Une ***task*** correspond à un processus, une *task* peut utiliser un ou plusieurs CPU.

3. **Les partitions**

Une partition dans SLURM correspond à un groupement logique de nœuds de calcul. Chaque partition est associée à des contraintes en termes de ressources (en particulier le temps de calcul maximum d'un job). Une partition peut être vue comme une file d'attente : l'utilisateur choisit la partition dans laquelle il soumet ses jobs en fonction de ses besoins.

4. **Les *features***

L'administrateur peut affecter des *features (tags)* aux nœuds en fonction de leurs caractéristiques. L'utilisateur peut alors préciser les *features* nécessaires pour ses jobs afin que SLURM affecte les jobs uniquement aux nœuds possédant le / les *features* demandés.

2.2 Soumettre un job

Pour soumettre un job (demande d'allocation de ressources pour exécuter un programme), l'utilisateur doit :

- soit définir son job dans un script et le lancer à l'aide de la commande **`sbatch`** (voir section 2.2.1);
- soit lancer directement son job en ligne de commande à l'aide de la commande `srun` (voir section 2.2.2).

2.2.1 Soumettre un job en utilisant un script

L'utilisateur crée un script bash (.sh) dans lequel il précise les ressources dont il a besoin puis appelle son programme.

Ce script est ensuite soumis à SLURM via la commande **`sbatch`** qui affecte un numéro au job.

1. Description du job dans un fichier bash (.sh)

Un script de soumission SLURM est composé de deux parties :

- un ensemble d'options, correspondant à des options de la commande **`sbatch`**, et permettant de préciser certaines caractéristiques du job telles que la quantité de ressources requise (CPU, RAM, temps) pour le programme que l'on souhaite exécuter,
- la ligne de commande du programme à exécuter. On peut aussi définir plusieurs étapes dans un job (et donc lancer plusieurs programmes en parallèle ou en séquentiel) via la commande **`srun.`**

Les options dans le script doivent être précédées de la directive **`#SBATCH`**. Un exemple de script permettant d'exécuter un programme monothread est donné ci-dessous :

```
#!/bin/bash

#=============================================================================
# exemples d'options

#SBATCH --partition=normal    # choix de la partition où soumettre le job
#SBATCH --time=10:0           # temps max alloué au job (format = m:s ou h:m:s ou j-h:m:s)
#SBATCH --ntasks=1            # nb de tasks total pour le job
#SBATCH --cpus-per-task=1     # 1 seul CPU pour une task
#SBATCH --mem=1000            # mémoire nécessaire (par noeud) en Mo

#=============================================================================
#exécution du programme (remplacer exe par le nom du programme à exécuter)
./exe
```

Exemple de script (`submit.sh`) de soumission pour un programme monothread

Remarque 1 : Les ressources demandées par un job lui sont allouées pour toute la durée de son exécution. Un job ne peut pas avoir accès à plus de ressources que celles demandées, en particulier s'il

dépasse le temps précisé en option (ou à défaut le temps max autorisé par la partition) il est automatiquement tué.

Remarque 2 : si l'utilisateur ne précise pas d'option ce sont celles par défaut qui sont appliquées :

- 1 CPU
- 2Go de RAM par CPU
- soumission dans la partition " normal" : jobs limités à 7 jours

Remarque 3 : la priorité d'un job dépend, entre autre, de la quantité de ressources demandées : plus un utilisateur demande de ressources, plus la priorité de son job tend à être faible.

2. Soumission du job via la commande `sbatch`

Le job décrit dans un script est soumis via la commande **`sbatch`** suivie du nom du script. SLURM affecte alors un numéro au job et le place dans la file d'attente (celle par défaut si l'utilisateur n'a pas précisé dans quelle partition il désirait soumettre son job, celle correspondant à la partition demandée sinon). Le job est exécuté quand les ressources sont disponibles.

```
[toussain@frontalhpc ~]$ sbatch submit.sh
Submitted batch job 2906757
```

Soumission du job décrit dans le script submit.sh

La soumission d'un job entraîne la génération d'un fichier de sortie, par défaut il est nommé **`slurm-<JobID>.out`** où **`JobID`** est le numéro affecté au job par SLURM. L'option **`--output=<NomDuFichier>`** permet de choisir un nom différent pour le fichier de sortie.

2.2.2 Soumettre un job directement en utilisant la commande srun

La commande **`srun`** permet de définir une étape dans un script mais on peut aussi l'utiliser en ligne de commande pour soumettre directement un job (sans écrire de script). Le résultat d'exécution s'affiche alors directement à l'écran (il n'y a pas de création de fichier résultat). Les options sont les mêmes qu'avec le script, elles sont passées en ligne de commande.

Exemple de soumission d'un exécutable (exe) dans la partition court pour 10 minutes :

```
[toussain@frontalhpc ~]$ srun --partition=court --time=10:0 ./exe
```

2.3 Les partitions et *features* définies actuellement sur le cluster

On a actuellement 3 partitions :

- court : jobs < 24h
- normal : jobs < 7 jours
- long : jobs < 30 jours

Une quatrième partition a été créée pour la cryptographie : elle est utilisable par tout le monde pour des jobs < 48h mais elle peut être réservée pendant plusieurs mois par les membres de l'équipe cryptographie et ainsi devenir indisponible sur une période assez longue.

- crypto : jobs < 48h (attention, partition pas toujours disponible)

La répartition des nœuds par partition et les *features* sont donnés dans le tableau ci-dessous. Chaque nœud possède son propre nom comme feature.

Noeuds	partition	features (en dehors du nom de la machine)
node22 à node25	normal et long	xeon26
node26 à node28 (sauf 27)	normal	xeon26
node30 à node32	normal	xeon25
molene168	normal	
dellware	court	
kephren	court	
bob (quand dispo)	crypto	

2.4 Annuler un job

Un job (en cours d'exécution ou en attente) peut être annulé par la commande **`scancel`** suivie du numéro du job.

2.5 Les tableaux de jobs

Les tableaux de jobs (*Job arrays)* offrent une manière très simple pour soumettre un grand nombre de jobs indépendants. Ils peuvent être typiquement utilisés pour appliquer le même programme à différentes données d'entrée.

La soumission d'un tableau de jobs s'effectue simplement en ajoutant l'option **`--array=<indices>`** à la liste des options de la commande **`sbatch`**. Par exemple l'option **`--array=0-9`** crée un tableau de jobs indicé de 0 à 9.

La variable d'environnement **`SLURM_ARRAY_TASK_ID`** donne l'indice du job courant dans le tableau. L'exemple suivant montre comment créer un tableau de 10 jobs et utiliser la variable **`SLURM_ARRAY_TASK_ID`** pour que chaque job s'exécute sur une instance (donnée d'entrée) différente. Les instances s'appellent insA.txt à insJ.txt et sont passées en argument de la ligne de commande.

```
#!/bin/bash

#================= OPTIONS (s'applique à chaque job du tableau) ========================
#SBATCH --array=0-9            # création d'un tableau de 10 jobs indicés de 0 à 9
#SBATCH --partition=court      # choix de la partition
#SBATCH --ntasks=1             # chaque job possède une seule task
#SBATCH --cpus-per-task=4      # une task nécessite 4 CPU
#SBATCH --mem-per-cpu=2048     # 2 Go de RAM par CPU

#========================== TASKS ================================

tab=(A B C D E F G H I J)
./exe ins${tab[$SLURM_ARRAY_TASK_ID]}.txt
```

Exemple de définition d'un job array de 10 jobs, chaque job lance un programme sur une instance différente

Le script est soumis de la manière habituelle à l'aide de la commande **sbatch**. Les 10 jobs seront alors envoyés dans la file d'attente et exécutés au fur et à mesure que des ressources sont disponibles. A la soumission du script, SLURM affecte un numéro de job pour le tableau (disponible via la variable d'environnement **SLURM_ARRAY_JOB_ID**) et un numéro par job du tableau (**SLURM_ARRAY_TASK_ID**).

2.6 Quelques options de sbatch

Une synthèse des options disponibles pour **sbatch** est disponible via la commande **sbatch --help** et la page de man donne d'avantage de précisions sur les options.

-a, --array=indexes	Crée un tableau de jobs et en précise les indices
--begin=time	Retarde le job jusqu'à la date précisée (HH:MM MM/DD/YY)
-c, --cpus-per-task=ncpus	Nombre de CPU nécessaire par tâche
-e, --error=err	Fichier de sortie pour les erreurs
-J, --job-name=jobname	Nom du job
--mail-type=type	Type des événements à notifier (BEGIN, END, FAIL, ALL)
--mail-user=user	Mail pour l'envoi des notifications
-n, --ntasks=ntasks	Nombre de tasks à exécuter
-N, --nodes=N	Nombre de nœuds sur lesquels exécuter le job (N = min[-max])
-o, --output=out	Fichier de sortie (à la place de slurm-jobID.out)
-p, --partition=partition	Nom de la partition souhaitée
-t, --time=minutes	Temps limite (format = m:s ou h:m:s ou j-h:m:s)
--mem=MB	Quantité max de mémoire par noeud à allouer au job
--mincpus=n	Nombre minimal de CPU par nœud
--mem-per-cpu=MB	Quantité maximum de mémoire par CPU requise par le job
--ntasks-per-core=n	Nombre de task sur chaque core (n=1 interdit l'hyperthreading)

Plus de précisions sur la gestion des ressources en fonction des options sélectionnées sont disponibles sur la page de SLURM : https://slurm.schedmd.com/cpu_management.html.

2.7 shell interactif

On peut utiliser un job pour demander un *shell interactif* (i.e. une session bash sur un nœud de calcul). Ceci peut être intéressant pour des programmes en cours de débogage, nécessitant une interaction avec l'utilisateur ou des compilations longues. La requête pour un shell interactif s'effectue en ligne de commande avec **srun** et l'option **--pty**. Par exemple la commande :

```
srun --ntasks=1 --cpus-per-task=2 --pty bash
```

demande un shell bash interactif utilisant 2 CPU.

Après avoir entré cette commande, le job est mis en file d'attente, comme n'importe quel job classique, jusqu'à ce que les ressources demandées soient disponibles. Une fois les ressources disponibles on obtient une session interactive sur un nœud de calcul. On peut alors exécuter des commandes comme dans une session classique. La commande **exit** permet de quitter la session et de libérer les ressources qui lui avaient été allouées.

Au cas où les ressources ne seraient pas disponibles on peut ajouter une option à la ligne de commande pour être prévenu par mail au moment où la session démarre :

```
srun --mail-user=user@xxx.fr --mail-type=BEGIN --pty bash
```

> Ne pas oublier l'option --pty qui permet d'obtenir une session semblable à une session "classique".
>
> Penser à bien quitter la session interactive à l'aide de la commande **`exit`** pour éviter de laisser des ressources oisives et indisponibles pour les autres utilisateurs.

3 Suivi des jobs, état des nœuds et des partitions

1. **sacct** affiche l'état des jobs de l'utilisateur qu'ils soient en cours ou déjà terminés. Les principaux états possibles sont :
   - CA, cancelled : le job a été annulé par l'utilisateur ou l'administrateur
   - CD, completed : le job s'est terminé avec succès
   - CG, completing : job en cours
   - F, failed : le job s'est terminé avec un échec
   - PD, pending : le job attend des ressources
   - R, running : le job est en cours d'exécution
   - TO, timeout : le job s'est terminé car il a atteint son temps d'exécution limite

2. **sinfo** affiche les partitions existantes

sinfo -N affiche l'état des nœuds, les états possibles sont :
- alloc : le nœud est entièrement utilisé
- mix : le nœud est en partie utilisé
- idle : aucun job ne tourne sur le nœud
- drain : le nœud termine les jobs qui lui ont été soumis mais n'en accepte plus d'autres (typiquement le nœud est sur le point d'être arrêté pour une opération de maintenance)

3. **squeue** affiche les jobs en cours et en attente dans l'ordre de leur priorité
   squeue -u <user> affiche les jobs en cours et en attente pour l'utilisateur user
   squeue -p <nomPart> affiche les jobs en cours et en attente pour la partition demandée
   squeue -i <sec> actualise la liste des jobs en cours toutes les sec secondes

4. **sprio** donne la priorité des jobs en attente (les jobs les plus prioritaires ont la priorité la plus élevée)

5. **sstat <jobID>** donne des informations sur la consommation de ressources d'un job en cours d'exécution

4 Exemples de scripts

4.1 Script pour un programme multithread (type openMP)

Le script suivant montre comment réserver 8 CPU pour un programme (typiquement un programme utilisant 8 threads en mémoire partagée : programme écrit avec openMP par exemple ou logiciel multithread comme Cplex).

```
#!/bin/bash

# ================== options de sbatch ===================================

#SBATCH --job-name=jobMemPartagee
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8  # réservation de 8 CPUs

# ====================ligne de commande ================================

# nom du programme précédé de la commande time pour avoir les temps de calcul
time ./exe
```

Réservation de 8 CPU pour un programme multithread de type openMP (mémoire partagée)

4.2 Script pour un programme multithread (type MPI)

SLURM est fait pour fonctionner avec MPI. Par défaut, il lancera autant de processus que de cœurs disponibles. On peut utiliser l'option **`--bind-to-core`** de mpirun pour attacher les processus MPI sur les cœurs.

```
#!/bin/bash

# ============= options de sbatch (ici réservation de 32 cores)==================
#SBATCH --ntasks=32
#SBATCH --ntasks-per-core=1

# ====================ligne de commande ================================
mpirun ./exe
```

Réservation de 32 CPU pour un programme multithread de type MPI (mémoire distribuée)