Introduction au Calcul à Haute Performance
Utilisation d'un cluster avec SLURM

Module de l'Ecole Doctorale SPI UCA

H. Toussaint

11 mars 2025

Présentation générale

- Tous les TP du module ont lieu sur le cluster du LIMOS (comptes temporaires => pensez à récupérer votre travail)
- Un mésocentre accessible à tous les personnels UCA (compte créé sur demande) `https://mesocentre.uca.fr/ressources/`
- Documentation LIMOS + exemples : `https://hpc.isima.fr/`

Cluster SLURM

Accès au cluster :

- Un gestionnaire de ressources : SLURM (Simple Linux Utility for Resource Management)
- Connexion en ssh sur la machine frontale uniquement
- Accès aux noeuds de calcul via SLURM

Intérêt d'un gestionnaire de ressources :

- Possibilité de réserver des ressources, mise en file d'attente si pas de ressource disponible, reproductibilité des temps de calcul (attention à l'hyperthreading, et turbo boost : adapatation dynamique à la charge de travail)

Présentation de SLURM

Qu'est-ce que SLURM ?

- Gestionnaire de ressources et ordonnanceur de tâches
- Répartit au mieux les ressources de calcul (CPU, GPU, RAM) entre utilisateurs en gérant des files d'attente avec priorité
- Confinement des ressources (CPU notamment) : un utilisateur ne peut pas accéder aux ressources réservées par d'autres
- Documentation complète : `https://slurm.schedmd.com/`

Vocabulaire

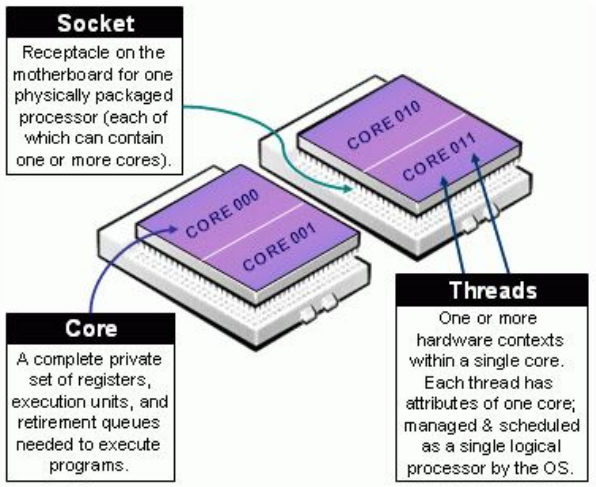

Figure – `https://slurm.schedmd.com/mc_support.html`

- Nœud de calcul (**node**) = une machine physique
- Un noeud contient des processeurs physiques (**sockets**) contenant eux-mêmes des cœurs physiques (**cores**)
- Si hyperthreading activé, chaque cœur physique donne 2 cœurs logiques (**threads**)
- Un **CPU** au sens SLURM = un cœur logique

Vocabulaire

Jobs et Tasks :

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

Vocabulaire

Partition :

- 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

Features :

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

Obtenir des informations sur le cluster

Informations sur les nœuds et les partitions :

- La commande `sinfo` affiche la liste des partitions disponibles
- La commande `sinfo -N` donne l'état des nœuds :
  - **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)
- La commande `sinfo -N --long` donne des détails sur les nœuds

Soumission d'un job

2 solutions :

- Définir le job dans un script et le lancer à l'aide de la commande `sbatch` (**solution conseillée**)
- Lancer directement le job en ligne de commande à l'aide de la commande `srun`

Attention : les jobs ne doivent en aucun cas s'exécuter sur le frontal (sous peine d'être tués par l'administrateur)

Soumission d'un job avec un script

- Ecriture d'un script bash (.sh)
- Possibilité de mettre des options (toutes les options sont facultatives)
- Soumission du script avec la commande `sbatch`
- SLURM attribue un numéro au job
- Quand le job est terminé un fichier `slurm-<numJob>.out` est créé par SLURM, il contient la sortie du programme

Voir exemple1.sh et exempleC

exemple1.sh

```
1  #!/bin/bash
2
3  # ========================================
4  # progamme (ligne de commande à exécuter)
5
6  hostname
```

Ajouter des options

- Par défaut (sur le cluster du LIMOS) SLURM attribue à un job :
  - 1 CPU
  - 2Go de RAM
  - 7 jours max de temps de calcul (partition par défaut)
- On peut modifier ce comportement par défaut à l'aide d'options
- Les options sont précisées dans le script à l'aide du mot clé `#SBATCH`

exemple2.sh

```
1  #!/bin/bash
2
3  #================================
4  # exemples d'options
5
6  #SBATCH --partition=court
7  #SBATCH --time=0:0:30
8  #SBATCH --output=monFichier.out
9
10
11 #================================
12 #exécution du programme
13 hostname
14 sleep 20
```

Quelques options

La commande `sbatch --help` donne une synthèse des options existantes.

Options de gestion du job

option		utilisation
-a,	--array=<indexes>	valeurs des indices pour un tableau de jobs
-e,	--error=<fic>	nom du fichier de sortie pour les erreurs
-I,	--immediate	le job est tué si les ressources ne sont pas immédiatement disponibles
-J,	--job-name=<jobname>	donne un nom au job
	--mail-type=<type>	notification par mail, type = BEGIN, END, FAIL ou ALL
	--mail-user=<user>	adresse mail à qui envoyer les notifications
-o,	--output=<out>	fichier de sortie
-t,	--time=<time>	temps max alloué au job (format = m:s ou h:m:s ou j-h:m:s)

Quelques options

option		utilisation
-c,	--cpus-per-task=<ncpus>	nombre de CPU pour une tâche
-n,	--ntasks=<ntasks>	nombre de tâches du job
	--ntasks-per-node=<n>	nombre de tâches désirées sur chaque noeud
-N,	--nodes=<N>	nombre de nœuds désirés (N = min[-max]), N = 1 par défaut
-C,	--constraint=<features>	*features* désirées (plusieurs features peuvent être spécifiées, dans ce cas les séparer par &)
	--mem=<MB>	quantité minimale de mémoire pour le job en Mo
	--mincpus=<n>	nombre minimal de CPU (coeur logique) par noeuds = nb de threads
-w,	--nodelist=<hosts>	précise les nœuds désirés (séparés par des virgules), cette liste doit contenir autant que noeuds que le nombre de noeuds réservés par l'option --nodes
-x,	--exclude=<hosts>	exclut certains nœuds de la partition (séparés par des virgules)

Voir exemple2.sh

Suivi des jobs

File d'attente :

- `squeue` affiche la liste des jobs en cours et en attente
  - `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

Suppression d'un job :

- `scancel <jobID>` supprime le job <jobID> (en cours ou en attente)
- `scancel -u <user>` supprime les jobs de l'utilisateur <user> (en cours ou en attente)

Suivi des jobs

Informations détaillées sur l'état d'un **job en cours** :

- `scontrol show job <jobID>` donne des informations détaillées sur le job <jobID> (en cours)
- `sstat -j <jobID>.batch --format=JobID,MaxRSS,AveRSS` pour voir la consommation mémoire d'un job

Suivi des jobs

Informations détaillées sur l'état d'un **job terminé** :

- `sacct -j <jobID>.batch --format=JobID,State,Elapsed,MaxRSS,AllocCPUs,ExitCode`
- `reportseff <jobID>`

Remarque : le job doit durer assez longtemps pour que SLURM ait le temps d'enregistrer la consommation mémoire (il est aussi possible d'utiliser `/usr/bin/time -v` devant le nom de l'exécutable pour des informations plus précises)

Tableaux de jobs

A quoi sert un tableau de jobs ?

- Soumettre à l'aide d'un seul script plusieurs jobs indépendants.
- Utile par exemple pour appliquer le même programme à différentes données d'entrée

Comment le définir ?

- A l'aide de l'option `--array=indiceMin-indiceMax`

Remarque : soyez parcimonieux dans vos demandes d'allocation.

Tableaux de jobs : exemple

Contexte :

- On dispose de 3 instances
- On souhaite exécuter le programme `exe` sur chacune de ces instances

Voir le dossier `exempleTableau`

exemple2.sh | test.c | submitCompil.sh | submitTab.sh

```
1  #!/bin/bash
2
3  #================= OPTIONS (s'applique à chaque job du tableau) ====================
4  #SBATCH --array=0-2            # création d'un tableau de 3 jobs indicés de 0 à 2
5  #SBATCH --ntasks=1             # chaque job possède une seule task
6
7  #========================== execution ================================
8  tab=(instance1.txt instance2.txt instance3.txt)
9  ./exe ${tab[$SLURM_ARRAY_TASK_ID]}
10
```

On lance le script de la manière habituelle avec `sbatch`.
SLURM attribue à chaque job du tableau à un identifiant propre constitué du numéro du job principal et de son indice dans le tableau.

Variables d'environnement

Les variables d'environnement peuvent être utilisées dans les scripts pour afficher des informations sur le job :

- `SLURM_JOB_ID` : numéro du job
- `SLURM_JOB_NAME` : nom du job
- `SLURM_JOB_NUM_NODES` : nombre total de nœuds alloués pour le job
- `SLURM_SUBMIT_DIR` : répertoire à partir duquel le job est soumis
- `SLURMD_NODENAME` : nom du nœuds sur lequel le job s'exécute
- `SLURM_JOB_PARTITION` : nom de la partition sur laquelle le job s'exécute

Variables d'environnement

Exemple d'utilisation :

exemple3.sh

```
1  #!/bin/bash
2
3  # ================================================
4  # OPTIONS
5  #SBATCH --job-name=testVarEnv
6
7  # ================================================
8  # progamme (ligne de commande à exécuter)
9  echo mon job $SLURM_JOB_NAME, num $SLURM_JOB_ID,
10 echo "s'execute" dans la partition $SLURM_JOB_PARTITION sur le noeud $SLURMD_NODENAME
11 hostname
12 sleep 60
```

Variables d'environnement

D'autres variables sont liées aux tableaux de jobs :

- `SLURM_ARRAY_JOB_ID` : numéro du job principal
- `SLURM_ARRAY_TASK_ID` : indice de la tâche dans le tableau
- `SLURM_ARRAY_TASK_COUNT` : nombre total de tâches dans un tableau de jobs
- `SLURM_ARRAY_TASK_MAX` : indice max du tableau de jobs
- `SLURM_ARRAY_TASK_MIN` : indice min du tableau de jobs

Scripts pour programme multithread

Mémoire partagée (type openMP)

- une task ...
- ... qui utilise plusieurs CPU

Voir l'exemple dans le dossier `exempleOpenMP`

```
#!/bin/bash

#=======================  OPTIONS de sbatch ====================================
#SBATCH --job-name=openMP
#SBATCH --ntasks=1                  # on a une seule tache, elle utilise 8 coeurs (mémoire partagée)
#SBATCH --cpus-per-task=8           # Allocation de 8 CPU  (mettre le meme nombre que dans le code !)
#SBATCH --mem-per-cpu=2000          # RAM par CPU (en Mo)

#SBATCH --time=10:00

#=======================  programme a executer=================================

./exe
```

Scripts pour programme multithread

Mémoire partagée (type openMP)

Pour avoir de bonnes performances :

- Le nombre de threads utilisés par openMP ne doit pas excéder le nombre de CPU réservé par SLURM

Attention aux librairies "multithreadées" :

- Certaines utilisent par défaut autant de threads que de cœurs disponibles (Cplex par exemple)
- Il faut toujours leur imposer un nombre de threads max $\leq$ au nombre de CPU réservé par SLURM (sinon perte de performances)

Scripts pour programme multiprocessus

Mémoire distribuée (type MPI)

Pour un programme à mémoire distribuée on a :

- n tasks ($\approx n$ processus)
- Possibilité de distribuer les tasks sur plusieurs nœuds d'une même partition
- Bonne pratique : attribuer 1 task par CPU

Voir le dossier `exempleMPI` (attention modif à faire dans le `.bashrc`).

submit.sh

```
1  #!/bin/bash
2
3  # REMARQUE : SLURM distribue automatiquement les processus de MPI
4  #sur les noeuds -> inutile de préciser l'option -np
5
6  #SBATCH --ntasks=20
7  #SBATCH --ntasks-per-core=1
8  #SBATCH --job-name=testMPI
9
10 mpirun ./hello
```

Les shells interactifs

On peut utiliser un job SLURM pour ouvrir un shell interactif (i.e. une session bash) sur un noeud.

Utilité :

- Débogage
- Compilations longues
- Programmes nécessitant une interaction avec l'utilisateur
- Suivre la consommation CPU / RAM en temps réel (commande `top` ou `htop`)

Important : Pensez toujours 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.

Les shells interactifs

Demander un shell interactif :

- En ligne de commande
- Commande basique : `srun --pty bash`
- Possibilité d'ajouter des options (les mêmes que dans les scripts)
- Exemple : `srun --ntasks=1 --cpus-per-task=2 --partition=court --pty bash`

Important : Pensez toujours 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.

Accès au Mésocentre

- Demande d'accès : `https://mesocentre.uca.fr/ressources/acces-aux-ressources`
- Documentation :`https://hub.mesocentre.uca.fr/docs/cluster/access/`
- Connexion par clé (pas de mot de passe) : la partie publique est à déposer dans `https://hub.mesocentre.uca.fr/`