Installation de serveurs GPU pour l'encodage déporté

Contexte

> Cette documentation correspond à la documentation de référence pour la mise en place de serveurs utilisant des cartes graphiques (serveurs GPU) pour réaliser les encodages à l'**Université de Montpellier** (UM).
>
> Il sera nécessaire de réaliser des adaptations selon votre configuration (serveurs, architecture, cartes graphiques...).

A l'université de Montpellier, jusqu'au mois de Septembre 2021, la plateforme vidéo utilisait 4 serveurs dédiés à l'encodage (avec 2 workers Celery) et le serveur principal (avec 1 worker Celery).

Au final, 9 vidéos pouvaient être encodées en parallèle. De plus, il fallait compter environ 1h d'encodage par heure de vidéo (selon la qualité, bien entendu).

Pour pouvoir gérer un nombre de vidéos plus importants, et surtout augmenter la rapidité des encodages, nous avons opté pour la mise en place de serveurs GPU.

Ainsi, 2 serveurs R740 ont été achetés, avec 2 cartes graphiques Tesla T4 pour chacun. Ces serveurs ont les caractéristiques suivantes :

- SSD 480 Go
- 128 Go de RAM
- 2 Tesla T4
- 2 alim 1100 W

Ces 2 serveurs tournent sous Debian 11.

Installations diverses et pré-requis

Toutes les installations et commandes ont été réalisées avec le compte **sun** (sauf si c'est explicitement écrit).

Il est indispensable de respecter l'ordre des installations.

Emacs

Personnellement, j'utilise emacs comme éditeur de texte.

```
sudo apt-get -y install emacs
```

Pré-requis

Le package time est nécessaire pour le calcul des temps d'exécution des scripts.

```
sudo apt-get install --reinstall time
```

Installation du driver Nvidia

Liens utiles

https://wiki.debian.org/fr/NvidiaGraphicsDrivers

https://linuxconfig.org/how-to-install-nvidia-driver-on-debian-10-buster-linux

Pré-requis

Vérifier que le repository **non-free** est bien activé dans le fichier /etc/apt/sources.list.

Installation de nvidia-detect

```
sudo apt -y install nvidia-detect
```

Le fait de lancer la commande nvidia-detect aboutira aux lignes suivantes :

```
Your card is also supported by the Tesla 460 drivers series.
Your card is also supported by the Tesla 450 drivers series.
Your card is also supported by the Tesla 418 drivers series.
```

Installation du driver Nvidia

```
sudo apt install nvidia-driver
```

A la fin de l'installation du driver Nvidia, il y a un message avertissant d'un conflit avec le package "nouveau".

Pour résoudre ce conflit, il est nécessaire de désactiver le driver "nouveau" :

```
sudo emacs /etc/modprobe.d/blacklist-nvidia-nouveau.conf

####
blacklist nouveau
options nouveau modeset=0
####

sudo reboot
```

Installation CUDA

Liens utiles

https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html

Installation CUDA 11.4

Il peut-être utile de télécharger la bonne version de CUDA via le package manager : https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64&Distribution=Debian&target_version=10&target_type=deb_network

Dans mon cas, cela n'a pas été nécessaire. L'installation a été réalisée via les commandes suivantes :

```
sudo apt-get install locales
sudo apt-get install -y software-properties-common
sudo apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/cuda/repos/debian10/x86_64/7fa2af80.pub
sudo add-apt-repository "deb https://developer.download.nvidia.com/compute/cuda/repos/debian10/x86_64/ /"
sudo add-apt-repository contrib
sudo apt-get update
sudo apt-get -y install cuda
sudo reboot
```

Il s'agit de la version pour Debian 10; à ce jour, il n'y a pas de version pour Debian 11.

Gestion de la persistance

Activer le mode persistant

Pour activer le mode persistant des GPU, il suffit de lancer les commandes suivantes :

```
sudo -i
nvidia-smi -pm 1
```

Pour vérifier, il est possible de lancer :

```
systemctl list-units --type service --all | grep nvidia
# Cela doit renvoyer :
nvidia-persistenced.service loaded active running NVIDIA Persistence Daemon
```

Daemon de persistance Nvidia (*non utilisé*)

Si besoin, on peut aussi utiliser un daemon de persistance Nvidia, à activer via les commandes suivantes :

```
sudo systemctl enable nvidia-persistenced
sudo /usr/bin/nvidia-persistenced --verbose
```

Dans mon cas, cela renvoie une erreur. A étudier si nécessaire (le mode persistant me paraît suffisant à l'heure actuelle).

Activation CUDA 11.4

Il est nécessaire d'éditer le .bashrc du compte sun pour activer et utiliser CUDA 11.4.

```
cd
emacs .bashrc

############
# Manually added by Loic to enable CUDA:
export LD_LIBRARY_PATH=/usr/local/cuda-11.4/lib64:$LD_LIBRARY_PATH
export LIBRARY_PATH=/usr/local/cuda-11.4/lib64:$LIBRARY_PATH
export PATH=/usr/local/cuda-11.4/bin${PATH:+:${PATH}}
export CUDA_PATH=/usr/local/cuda-11.4
export CUDA_VISIBLE_DEVICES=0,1
export CUDA_DEVICE_ORDER=PCI_BUS_ID
############

source .bashrc
```

Les lignes CUDA_VISIBLE_DEVICES=0,1 et CUDA_DEVICE_ORDER=PCI_BUS_ID sont utiles dans notre cas, car le serveur utilise 2 GPU avec le type select/cons_tres, configuré avec du MPS (cf. paragraphe sur SLURM).

Utilisation de ffmpeg

Liens utiles

Pour la compilation de ffmpeg : https://docs.nvidia.com/video-technologies/video-codec-sdk/ffmpeg-with-nvidia-gpu/

Pour Lame : https://stackoverflow.com/questions/35937403/error-libmp3lame-3-98-3-not-found

Compilation de ffmpeg

Il n'est pas possible d'utiliser une version de base de ffmpeg avec des GPU. La compilation est indispensable.

ffmpeg doit être compilé avec utilisation de cuvid | nvenc | cuda | nvdec (et libnpp).

Pré-requis de ffmpeg

Il faut récupérer les sources et installer les pré-requis via les commandes suivantes :

```
cd
sudo apt-get install -y pkg-config
sudo apt-get install git -y
git clone https://git.videolan.org/git/ffmpeg/nv-codec-headers.git
cd nv-codec-headers && sudo make install && cd ..
git clone https://git.ffmpeg.org/ffmpeg.git ffmpeg/
sudo apt-get install build-essential yasm cmake libtool libc6 libc6-dev unzip wget libnuma1 libnuma-dev
```

Pré-requis de ffmpeg : Lame

Télécharger la dernière version stable de Lame; à ce jour, la 3.100 : https://sourceforge.net/projects/lame/files/lame/3.100/

Il ne reste plus qu'à l'extraire et l'installer :

```
cd
tar -zxvf lame-3.100.tar.gz
cd lame-3.100
./configure
make
sudo make install
```

Pré-requis de ffmpeg : libopus

```
sudo apt-get install -y libopus-dev
```

Compilation des sources

Il faut compiler les sources via les commandes suivantes :

```
cd ~/ffmpeg

./configure --enable-cuda --enable-nonfree --enable-cuda-nvcc --enable-cuvid --enable-nvenc --enable-nvdec --
enable-libnpp --disable-x86asm --enable-libmp3lame --extra-cflags=-I/usr/local/cuda/include --extra-ldflags=-L
/usr/local/cuda/lib64 --pkg-config-flags="--static" --enable-libopus

make (ou si besoin: make -j 8)

sudo make install

# Déplacement des binaires ffmpeg et ffprobe
sudo mv /usr/local/bin/ffmpeg /usr/bin
sudo mv /usr/local/bin/ffprobe /usr/bin

# Vérification
ffmpeg -version
```

D'autres tests de configuration de ffmpeg ont été réalisées ou récupérées sur le Web; au cas où, je laisse les commandes :

```
/////// TESTS: configuration: --prefix=/home/ysh/ffmpeg_build --pkg-config-flags=–static --extra-cflags=-I/home
/ysh/ffmpeg_build/include --extra-ldflags=-L/home/ysh/ffmpeg_build/lib --extra-libs='-lpthread -lm' --ld=g++ --
bindir=/usr/local/bin --enable-gpl --enable-gnutls --enable-libass --enable-libfdk-aac --enable-libfreetype --
enable-libmp3lame --enable-libopus --enable-libvorbis --enable-libx264 --enable-libx265 --enable-nonfree --
enable-cuda --enable-cuvid --enable-nvdec --enable-nvenc --enable-nonfree --enable-libnpp --extra-cflags=-I/usr
/local/cuda/include --extra-ldflags=-L/usr/local/cuda/lib64
/// Configuration de celui installé par sudo apt install ffmpeg :
/// configuration: --prefix=/usr --extra-version='1~deb10u1' --toolchain=hardened --libdir=/usr/lib/x86_64-
linux-gnu --incdir=/usr/include/x86_64-linux-gnu --arch=amd64 --enable-gpl --disable-stripping --enable-
avresample --disable-filter=resample --enable-avisynth --enable-gnutls --enable-ladspa --enable-libaom --enable-
libass --enable-libbluray --enable-libbs2b --enable-libcaca --enable-libcdio --enable-libcodec2 --enable-
libflite --enable-libfontconfig --enable-libfreetype --enable-libfribidi --enable-libgme --enable-libgsm --
enable-libjack --enable-libmp3lame --enable-libmysofa --enable-libopenjpeg --enable-libopenmpt --enable-libopus
--enable-libpulse --enable-librsvg --enable-librubberband --enable-libshine --enable-libsnappy --enable-libsoxr
--enable-libspeex --enable-libssh --enable-libtheora --enable-libtwolame --enable-libvidstab --enable-libvorbis
--enable-libvpx --enable-libwavpack --enable-libwebp --enable-libx265 --enable-libxml2 --enable-libxvid --
enable-libzmq --enable-libzvbi --enable-lv2 --enable-omx --enable-openal --enable-opengl --enable-sdl2 --enable-
libdc1394 --enable-libdrm --enable-libiec61883 --enable-chromaprint --enable-frei0r --enable-libx264 --enable-
shared
```

Installation et configuration de Slurm

Slurm est un système de gestion de cluster et de planification des tâches open source, tolérant aux pannes et hautement évolutif pour les clusters Linux grands et petits.

Slurm ne nécessite aucune modification du noyau pour son fonctionnement et est relativement autonome.

En tant que gestionnaire de charge de travail de cluster, Slurm a trois fonctions clés :

- Premièrement, il alloue un accès exclusif et/ou non exclusif aux ressources (nœuds de calcul) aux utilisateurs pendant une certaine durée afin qu'ils puissent effectuer un travail.
- Deuxièmement, il fournit un cadre pour démarrer, exécuter et surveiller le travail (normalement un travail parallèle) sur l'ensemble des nœuds alloués.
- Enfin, il arbitre les conflits de ressources en gérant une file d'attente de travail en attente. Des plug-ins facultatifs peuvent être utilisés pour la comptabilité, les réservations avancées, la planification des gangs (partage du temps pour les travaux parallèles), la planification des renvois, la sélection de ressources optimisée pour la topologie, les limites de ressources par utilisateur ou compte bancaire et les algorithmes sophistiqués de hiérarchisation des travaux multifacteur.

Slurm doit être installé et configuré après l'installation du driver Nvidia et CUDA pour éviter des erreurs NVML.

Liens utiles

Slurm quickstart : https://slurm.schedmd.com/quickstart_admin.html

Installation Munge : http://wiki.sc3.uis.edu.co/index.php/Slurm_Installation_on_Debian

Création du user /group slurm

Par défaut, un compte slurm/slurm est utile. Dans notre cas, nous utiliserons le compte **sun/sun** comme utilisateur de Slurm.

> ***Commandes inutiles, mais laissés au cas où**. En cas d'installation du package slurm, le user/group slurm peut exister avec l'id spécifique 64030.*
>
> *Si besoin : sudo useradd -m -u 64030 slurm*
>
> *sudo emacs /etc/passwd*
>
> *slurm:x:64030:64030::/nonexistent:/usr/bin/nologin*

Installation de Slurm et de ses pré-requis

Il est nécessaire d'installer une version v20+ de Slurmd pour une gestion correcte des GPU et de l'option indispensable **SelectType=select/cons_tres**.

Cette installation doit être faite avec le compte *sun* (qui sera utilisé comme slurmuser).

> Commentaires pour Debian 10 : surtout ne pas installer la version de base trop ancienne (v18 via sudo apt-get install -y slurmd).
>
> Pour Debian 11, il est possible d'installer directement le package Slurmd de base, en version 20.11, via sudo apt-get install -y slurmd
>
> Dans notre cas, j'ai suivi la documentation de référence, qui explicite l'installation via les sources : https://slurm.schedmd.com/quickstart_admin.html

Installation Munge

La documentation de référence pour l'installation de Munge est la suivante : http://wiki.sc3.uis.edu.co/index.php/Slurm_Installation_on_Debian

Au final, voici les commandes réalisées :

```
sudo apt-get install -y libmunge-dev libmunge2 munge
```

Générer la clé munge (à ne faire que pour le serveur contrôleur GPU1) :

```
sudo -i
dd if=/dev/urandom bs=1 count=1024 > /etc/munge/munge.key
```

Pour les autres serveurs de calcul (typiquement GPU2 dans notre cas) : copier la clé **/etc/munge/munge.key** du serveur GPU1 et la mettre sur chaque serveur de calcul au même endroit.

Sécuriser la clé :

```
sudo chown munge:munge /etc/munge/munge.key
sudo chmod 400 /etc/munge/munge.key
```

Personnellement, pour que le user munge soit identique sur tous les serveurs, je réalise directement une modification du user munge via :

```
emacs /etc/passwd

#######
munge:x:501:501::/var/run/munge;/usr/sbin/nologin
#######
```

Par la suite, pour éviter tout problème de droit, j'exécute les commandes suivantes :

```
sudo chown munge:munge /etc/munge -R
sudo chown munge:munge /var/log/munge -R
sudo chown munge:munge /var/lib/munge -R
sudo chown munge:munge /run/munge -R
```

Et enfin j'active et je démarre le service munge :

```
sudo systemctl enable munge
sudo systemctl start munge
```

Ne pas oublier de faire de même pour tous les serveurs GPUs, **excepté le fait d'utiliser la même clé sur chaque serveur (/etc/munge/munge.key).**

Installation de Slurm

Voici les étapes à réaliser pour installer slurm :

```
# Téléchargement
mkdir ~/downloads
=> télécharger slurm dans homedirectory/downloads via https://www.schedmd.com/downloads.php. Version 21.08 à ce
jour

# Dézip
cd downloads
tar --bzip -x -f slurm*tar.bz2

# Droits
sudo chown sun:sun /home/sun/downloads/slurm-21.08.0 -R

# Compilation des sources et installation effective
cd slurm-21.08.0
./configure --prefix=/usr/local --sysconfdir=/etc/slurm/ --with-hdf5=no
sudo make
sudo make install
sudo ldconfig -n /usr/local/lib
```

Configuration Slurm

Gestion des droits initiaux :

```
sudo chown sun:sun /etc/slurm/ -R
```

Mettre les fichiers de configuration, déjà travaillés en test, dans le répertoire */etc/slurm*, à savoir :

Nom des fichiers	Fichiers sources

cgroup.conf	
gres.conf	
slurm.conf	

Ces fichiers sont identiques sur l'ensemble des noeuds du cluster.

> Ces fichiers sont totalement spécifiques à l'architecture mise en place de l'Université de Montpellier et nécessite d'être adapté à votre configuration.

Ne pas oublier de mettre les adresses IP des serveurs de calcul dans les fichiers */etc/hosts* de chaque serveur :

```
127.0.0.1 localhost
xxx.xxx.xxx.xxx prod-gpu1.univ.fr prod-gpu1
xxx.xxx.xxx.xxx prod-gpu2.univ.fr prod-gpu2
```

Gestion des droits :

```
chmod 660 /etc/slurm/slurm.conf
chmod 660 /etc/slurm/gres.conf
```

⚠️ Le paramétrage de slurm n'est pas du tout évident et nécessite de nombreux tests et une lecture approfondie de la documentation officielle.

Les éléments importants du paramétrage de slurm.conf :

- *SlurmctldHost=prod-gpu1* : cela correspond au nom du contrôleur (celui qui exécute slurmctld). Dans notre cas, c'est le serveur GPU1.
- *GresTypes=gpu,mps* : cela correspond à l'utilisation du GPU pour les processus, et mps permet d'avoir plusieurs threads en parrallèle sur chaque GPU.
- *ProctrackType=proctrack/cgroup* : pour la gestion des groupes de base.
- ReturnToService=1 : par défaut

- *SlurmctldPidFile=/run/slurm/slurmctld.pid* : répertoire à créer avant le 1° démarrage
- *SlurmctldPort=6817* : par défaut
- *SlurmdPidFile=/run/slurm/slurmd.pid* : répertoire à créer avant le 1° démarrage
- *SlurmdPort=6818* : par défaut
- *SlurmdSpoolDir=/var/lib/slurm/slurmd* : répertoire à créer avant le 1° démarrage
- *SlurmUser=sun* : ℹ Très important. Ainsi, tous les répertoires, fichiers, logs... de Slurm appartiennent au compte *sun* et doivent être writeable par le user *sun*.
- *StateSaveLocation=/var/lib/slurm/slurmctld* : répertoire à créer avant le 1° démarrage
- *SwitchType=switch/none* : par défaut
- *TaskPlugin=task/cgroup,task/affinity* : pour la gestion des groupes de base.
- *InactiveLimit=0* : par défaut
- *KillWait=30* : par défaut
- *MinJobAge=300* : par défaut
- *SlurmctldTimeout=120* : par défaut
- *SlurmdTimeout=300* : par défaut
- *Waittime=0* : par défaut
- *SchedulerType=sched/backfill* : par défaut
- *SelectType=select/cons_tres* : ℹ Très important. Cela permet d'utiliser les GPU comme ressources.
- *SelectTypeParameters=CR_Core* : par défaut
- *PriorityType=priority/basic* : finalement, je n'utilise pas le système de priorité de Slurm, trop complexe pour nos besoins. Les priorités sont directement misent en place dans le script principal.
- *AccountingStorageType=accounting_storage/none* : aux vues des besoins, il n'est pas nécessaire d'utiliser le système d'accounting de slurm.
- *AccountingStoreFlags=job_comment* : aux vues des besoins, il n'est pas nécessaire d'utiliser le système d'accounting de slurm.
- *ClusterName=cluster* : aux vues des besoins, il n'est pas nécessaire d'utiliser le système d'accounting de slurm.
- *JobCompLoc=/var/log/slurm/job_completions.log* : quand les tâches sont réalisées, les infos sont inscrites dans ce fichier de log.
- *JobCompType=jobcomp/filetxt* : quand les tâches sont réalisées, les infos seront inscrites dans un fichier de log texte.
- *JobAcctGatherType=jobacct_gather/linux* : par défaut
- *SlurmctldDebug=info* : par défaut
- *SlurmctldLogFile=/var/log/slurm/slurmctld.log* : répertoire des logs du contrôleur à créer avant le 1° démarrage
- *SlurmdDebug=info* : par défaut
- *SlurmdLogFile=/var/log/slurm/slurmd.log* : répertoire des logs du noeud de calcul à créer avant le 1° démarrage
- *NodeName=prod-gpu1 NodeHostname=prod-gpu1 NodeAddr=xxx.xxx.xxx.xxx Gres=gpu:2,mps:200 CPUs=40 Sockets=2 CoresPerSocket=10 ThreadsPerCore=2 RealMemory=120000*
  ✅ Configuration d'un noeud de calcul, selon les caractéristiques techniques de celui-ci. Il s'agit du 1° serveur GPU (*attention : NodeAddr est important*), qui utilise ces 2 cartes graphiques, chacune pouvant utiliser un total de 100 mps (*Gres=gpu:2,mps:200).*
  On pourra changer La valeur du mps si nécessaire.
- NodeName=prod-gpu2 NodeHostname=prod-gpu2 NodeAddr=xxx.xxx.xxx.xxx Gres=gpu:2,mps:200 CPUs=40 Sockets=2 CoresPerSocket=10 ThreadsPerCore=2 RealMemory=120000
  ✅ Configuration d'un noeud de calcul, selon les caractéristiques techniques de celui-ci. Il s'agit du 2° serveur GPU (*attention : NodeAddr est important*), qui utilise ces 2 cartes graphiques, chacune pouvant utiliser un total de 100 mps (*Gres=gpu:2,mps:200).*
  On pourra changer La valeur du mps si nécessaire.
- PartitionName=pod Nodes=prodsun-gpu[1-2] Default=YES MaxTime=INFINITE State=UP
  La partition s'appelle *pod* et utilise les 2 noeuds configurés ci-dessus *(prod-gpu1 et prod-gpu2).*

Après avoir réalisé le paramétrage, et avant de démarrer slurmd et slurmctld, il est nécessaire de créer les répertoires, avec les droits adéquats :

```
sudo mkdir /var/log/slurm
sudo chown sun:sun /var/log/slurm
sudo mkdir /var/lib/slurm
sudo chown sun:sun /var/lib/slurm
sudo mkdir /run/slurm
sudo chown sun:sun /run/slurm
```

Services slurmd et slurmctld

✅ Au final :

- GPU1 fait office de contrôleur et de noeud de calcul : il faut donc activer et démarrer et *slurmctld* et *slurmd*.
- GPU2 fait office de noeud de calcul : il faut donc activer et démarrer seulement *slurmd*.

Il est nécessaire, dans un premier temps, de copier les fichiers de service :

```
sudo cp /home/sun/downloads/slurm-21.08.0/etc/slurmd.service /usr/lib/systemd/system/
sudo cp /home/sun/downloads/slurm-21.08.0/etc/slurmctld.service /usr/lib/systemd/system/
sudo systemctl enable slurmctld
sudo systemctl enable slurmd
# Au cas où
#sudo systemctl unmask slurmctld
#sudo systemctl unmask slurmd
# A faire sur GPU1
sudo systemctl start slurmctld
# A faire sur GPU1 et GPU2
sudo systemctl start slurmd
```

Après avoir démarré slurm, en profiter pour remettre les droits au user *sun* (par défaut pour slurmd , c'est *root*) :

```
sudo chown sun:sun /var/lib/slurm -R
sudo chown sun:sun /var/log/slurm -R
sudo chown sun:sun /run/slurm -R
```

La solution d'encodage de Pod

La communication entre serveurs distants

Il est indispensable que les serveurs de Pod (**le serveur principal ainsi que les serveurs Web**) puissent communiquer avec les serveurs d'encodage GPU, et vice-versa.

Pour ce faire, tout se réalise via un système de clé SSH.

Sur les serveurs GPU (GPU1 et GPU2)

- Générer une paire de clés pour l'utilisateur sun :

```
ssh-keygen -t rsa -b 2048
```

ℹ️ *Laisser les chemins par défaut et ne pas mettre de passphrase.*

- Copier le contenu de la clé publique (*/.ssh/id_rsa.pub de ces serveurs GPU1 et GPU2*) sur le serveur pod v2 principal, ainsi que les serveurs Web (Nginx) dans le fichier *.ssh/authorized_key*

Sur le serveur principal Pod, et les serveurs Web Nginx

- Générer une paire de clés pour l'utilisateur pod (ou podprep en préproduction) sur le serveur podv2 principal :

```
ssh-keygen -t rsa -b 2048
```

ℹ️ *Laisser les chemins par défaut et ne pas mettre de passphrase.*

- Copier le contenu de la clé publique (*/.ssh/id_rsa.pub du serveur podv2 principal*) sur les serveurs d'encodage distant GPU, dans le fichier *.ssh /authorized_key*

⚠️ *Ouvrir le firewall de manière à ce que les serveurs puissent communiquer.*

Il est nécessaire la 1° fois de valider la confiance entre les différents serveurs. Le mieux est d'alors de lancer les commandes suivantes :

Commandes à faire la 1° fois depuis le serveur web1 :
ssh -i /data/www/pod/.ssh/id_rsa_web11 sun@prod-gpu1.univ.fr "df -h"
ssh -i /data/www/pod/.ssh/id_rsa_web11 sun@prod-gpu2.univ.fr "df -h"

Commandes à faire la 1° fois depuis le serveur web2 :
ssh -i /data/www/pod/.ssh/id_rsa_web12 sun@prod-gpu1.univ.fr "df -h"
ssh -i /data/www/pod/.ssh/id_rsa_web12 sun@prod-gpu2.univ.fr "df -h"

Commandes à faire la 1° fois depuis le serveur prod-video1 :
ssh -i /home/pod/.ssh/id_rsa sun@prod-gpu1.univ.fr "df -h"
ssh -i /home/pod/.ssh/id_rsa sun@prod-gpu2.univ.fr "df -h"

Commandes à faire la 1° fois depuis les serveurs GPU1 et GPU2 :
ssh -i /home/sun/.ssh/id_rsa pod@prod-video1.univ.fr "df -h"
ssh -i /home/sun/.ssh/id_rsa pod@prod-web1.univ.fr "df -h"
ssh -i /home/sun/.ssh/id_rsa pod@prod-web2.univ.fr "df -h"

Sinon, il est aussi possible de:

- récupérer la commande exécutée dans l'administration de Pod, rubrique Vidéo, Journaux des encodages et de la lancer à la main depuis les serveurs Web et le serveur podv2 principal
- récupérer la commande depuis les serveurs GPU, dans /var/pod/encoding/encoding-xxx/encode.job, et la lancer depuis les serveurs GPU.

La solution sur les serveurs d'encodage GPU

J'ai modifié les différents scripts de base pour que cela corresponde à mes besoins (utilisation de 2 cartes GPU....).

Voici les informations applicatives :

Valeur	Commentaires	Fichiers sources

/var/pod	Répertoire de base, contenant les scripts applicatifs, à savoir : • ***scripts_gpu/encode.template*** : template d'encodage pour Slurm • ***scripts_gpu/encode_gpu.py*** : script qui réalise l'encodage GPU voire CPU si besoin • ***scripts_gpu/script.sh*** : script de base de la solution, appelé dans Pod :	  
/var/pod/encoding	Répertoire contenant les jobs des encodages générés	
/var/pod/slurm-encoding-out	Répertoire utilisé par slurm pour les logs	
/tmp	Répertoire temporaire pour les encodages. ⚠️ *Nécessite donc pas mal de stockage.*	

Pour la gestion des répertoires et des droits, il est nécessaire d'exécuter les commandes suivantes :

```
sudo mkdir /var/pod
sudo chown sun:sun /var/pod
mkdir /var/pod/encoding
mkdir /var/pod/slurm-encoding-out
chmod 744 /var/pod/scripts_gpu/script.sh
```

Exploitation

Liens utiles

https://nvidia.custhelp.com/app/answers/detail/a_id/3751/~/useful-nvidia-smi-queries

Architecture simplifiée

Compte UM	sun
Serveur contrôleur slurm	prod-gpu1.univ.fr adresse xxx.xxx.xxx.xxx
Serveurs noeuds de calcul	prod-gpu1.univ.fr adresse xxx.xxx.xxx.xxx prod-gpu2.univ.fr adresse xxx.xxx.xxx.xxx
Répertoire de base des scripts	/var/pod
Répertoires des logs	/var/pod/slurm-encoding-out /var/log/slurm
Répertoire temporaire des encodages	/tmp
Base de données	Aucune
Fichier principal de configuration de slurm	/etc/slurm/slurm.conf

Commandes utiles

Commandes vis-à-vis des GPU, CUDA...

Commandes	Commentaires
for i in encoders decoders filters; do echo $i:; ffmpeg -hide_banner -${i} | egrep -i "npp|cuvid|nvenc|cuda|nvdec"; done	Liste des encodeurs/décodeurs Nvidia
ffmpeg -hwaccels -hide_banner	Liste des accélérateurs ffmpeg
ffmpeg -y -vsync 0 -hwaccel cuda -hwaccel_output_format cuda -i input.mp4 -c:a copy -c:v h264_nvenc -b:v 5M output.mp4	Test d'encodage avec utilisation CUDA
lspci | grep -i nvidia	Vérification des cartes GPU
⭐⭐⭐ nvidia-smi	Liste des processus pris en charge par les GPUs
nvcc --version	Version de CUDA
srun env | grep CUDA	Path de CUDA, configuré dans le .bashrc
lscpu	Architecture du serveur

Commandes vis-à-vis de Slurm et des noeuds de calcul

Commandes	Commentaires
srun	Exécute une commande
sbatch	Exécute une commande
sinfo	Informations sur les noeuds du cluster
sinfo --Node --long	Informations plus détaillées sur les noeuds du cluster
⭐⭐⭐ squeue	Liste d'attente des jobs
scontrol show jobid -dd 140	
scancel 140	Annuler un job en cours d'exécution (140 dans ce cas)
sinfo 140	Informations sur un job en cours d'exécution (140 dans ce cas)
sstat 140	Statistiques sur un job en cours d'exécution (140 dans ce cas)
salloc -n 4 -c 2 -gres=gpu:1	
srun --jobid 89 --pty watch -n 30 nvidia-smi	Lance un job n°89 qui va exécuter nvidia-smi toutes les 30 secondes

scontrol show nodes	Affiche les informations de tous les noeuds
⭐⭐⭐ scontrol show node prod-gpu1 ⭐⭐⭐ scontrol show node prod-gpu2	Affiche les informations d'un noeud
scontrol reconfigure	
scontrol update NodeName=prod-gpu2 State=Down Reason=" Undraining"	Commande à faire si un noeud n'est pas dans un bon état et si l'on souhaite supprimer tous les jobs en cours sur ce noeud. Il faut après faire la commande ci-dessous (State=Resume) pour que cela fonctionne à nouveau (en ayant perdu l'historique des jobs).
⭐⭐⭐ scontrol update NodeName=prod-gpu1 State=Resume ⭐⭐⭐ scontrol update NodeName=prod-gpu2 State=Resume	Commande à faire si un noeud est dans state=drain (https://stackoverflow.com/questions/29535118/how-to-undrain-slurm-nodes-in-drain-state)

Ne pas tuer un travail srun pour annuler un travail SLURM ! Cela ne fait que terminer srun. Les tâches continuent de s'exécuter, mais pas sous la gestion SLURM. Si vous tuez/qualifiez un travail srun, vous pouvez utiliser *squeue* pour obtenir l'identifiant du travail, puis soit annuler le travail, soit utiliser *srun -p <partition> -a <jobid> -j*, pour rattacher srun au travail, puis vous pouvez utiliser Ctrl-C pour l'annuler.