Création d'un cluster Raspberry PI avec ClusterHat

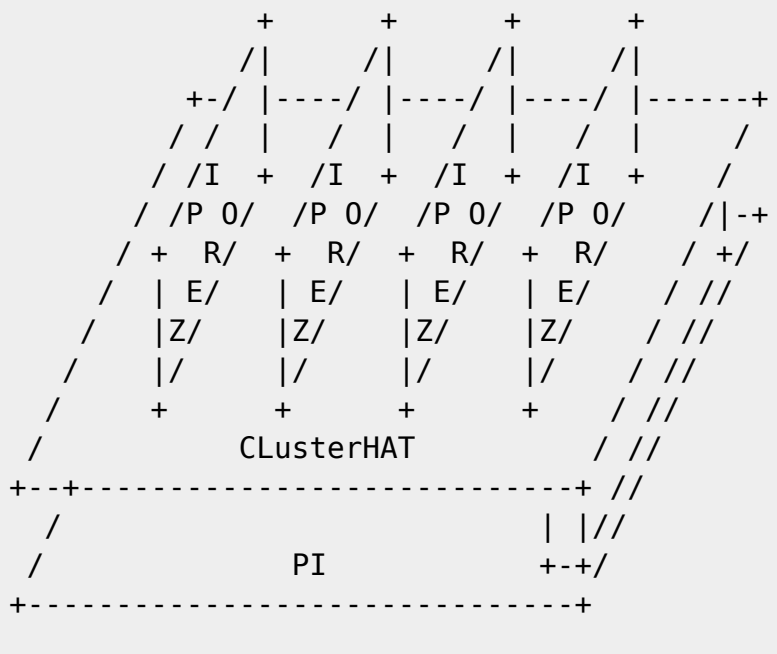
Table of Contents

Un cluster HPC (Haute Performance de Calcul) peut être vu comme un ensemble de machines coopérants pour résoudre un problème non soluble dans un temps raisonnable par une machine seule. Nous allons voir comment créer un cluster sous Linux en utilisant des logiciels libres.

L'utilisateur se connecte à un nœud frontal (contrôleur) où il trouve son environnement ainsi que son home. L'utilisateur interagit ensuite avec un job scheduler localisé sur le nœud frontal pour demander la réalisation de ses travaux. Le job scheduler distribue ensuite le travail sur les nœuds de calcul. L'utilisateur ne peut donc pas se connecter en direct sur les nœuds de calcul. Il doit décrire les ressources qu'il demande (nombre de CPU, de nœuds, quantité de mémoire etc.) ainsi que la façon de lancer ses calculs (exécutable, script, variables d'environnements etc.). C'est ce qu'on appelle un « job ».

Partie I — Création du cluster

Le **ClusterHat** est un shield pour un Raspberry Pi 3 ou 4 qui permet de connecter facilement quatre Raspberry Pi Zero en tant que nœuds à un Raspberry Pi 3/4. Il est extrêmement facile à installer, à configurer, mais il n'y a pas beaucoup de tutoriels sur ce qu'il fallait faire avec un cluster Raspberry Pi.



L'objectif du tutoriel est de construire un cluster basé sur cette topologie et des logiciels libres.

Prérequis

Pour configurer ce cluster, on a besoin des éléments suivants :

- 1 x ClusterHat
- 1 x Raspberry Pi 3 ou 4
- 4 x Raspberry Pi Zero ou Zero W — L'internet sans fil ne sera pas nécessaire mais si on utilise un W on peut l'utiliser
- 5 x cartes MicroSD - Cela contiendra le système d'exploitation pour chacun des Raspberry Pis. Il existe un didacticiel sur le site Web de ClusterHat qui permet d'exécuter les images Nodes sur le réseau. Cependant, bien que cette méthode permette d'économiser un peu d'argent, il fautsavoir que les nœuds fonctionneront probablement plus lentement, car ils exécuteront leur système d'exploitation sur le réseau.
- 1 x clé USB ou disque dur externe - En option, mais recommandé
- PSU - Le contrôleur principal Raspberry Pi alimentera les autres, il faut donc s'assurer que l'alimentation est suffisamment puissante pour alimenter les trois. L'alimentation minimale requise pour l'ensemble du système est de 1 à 1,2 A (selon que l'on utilise des zéros ou des zéro Ws), on peu sans problème exécuter la configuration avec un bloc d'alimentation de 2,5 A

Les Raspberry Pi Zero n'ont pas de port Ethernet, mais il y a une alternative : le mode OTG (On The Go). Pour faire simple, il suffit de brancher un câble USB sur le Raspberry Pi, de le connecter à un PC et il fonctionne comme un adaptateur Ethernet. Ensuite, il devient possible de se connecter en SSH dessus.

Assembler le matériel du cluster

La configuration matérielle du ClusterHat est assez simple. Il est livré avec des entretoises et des vis à installer sur le Raspberry Pi 3 ou 4. Placer le ClusterHat sur les broches GPIO du contrôleur Pi et fixer-le en place avec les entretoises et les vis. Placer les Pi Zero sur le cluster hat, en utilisant le bon port USB sur le Pi Zero (celui étiqueté USB plutôt que l'USB d'alimentation). Le Clusterhat utilise l'USB pour pousser le réseau vers les PI Zeros.

Installer les images OS

Méthode 1: Images ClusterHat Raspbian

Télécharger une image Raspbian pour chaque Pi (Controller/P1/P2/P3/P4). Le site ClusterHat fournit un ensemble d'image **RaspBian** pour le contrôleur et pour les Pis Zero. Pour l'image du controleur on a le choix, avec interface graphique de bureau et logiciel (FULL),avec bureau (STD) et Lite (LITE), et en fonction de la façon de gérer le réseau:

- Les images CBRIDGE relient l'USB Gadget Ethernet des Pi Zeros à eth0 sur le contrôleur leur permettant d'obtenir une adresse IP à partir d'un serveur DHCP sur le réseau local.

- L'image CNAT utilise NAT (Network Address Translation) pour partager automatiquement la connexion Ethernet ou Wifi sur le contrôleur avec les Pi Zeros. Les Pi Zeros recevront automatiquement une adresse IP 172.19.181.X où X est le numéro pX (172.9.181.254 sera utilisé sur le contrôleur). L'image CNAT créer un sous-réseau pour les PI Zeros sur le contrôleur pi, ce qui rend la configuration beaucoup plus facile. Il est également plus sécurisé car les nœuds ne seront visibles que par l'image du contrôleur et non par les périphériques externes.

Graver les images Raspbian sur les cartes SD avec **Imager**.

SSH n'est plus activé par défaut sur aucune des images, pour activer SSH, il faut créer un fichier nommé “ssh” dans la partition de démarrage cela doit être fait sur le contrôleur et les images Pi Zero (à partir du Controller Pi, les Pi Zeros sont accessibles via la console série si SSH est désactivé).

Une fois l'image gravée sur la carte SD, ouvrir la partition de démarrage et créer un fichier vide nommé **ssh** sur la partition de démarrage. Cela permettra de ses connecter en SSH au Raspberry Pis.

Deskpi Pro est un kit matériel pour convertir une Raspberry PI 4 standard à partir d'un SBC nu, avec un stockage limité, en un mini PC complet avec un bouton d'alimentation, un refroidissement, de meilleurs ports et un Mass Storage via SATA USB3 ou M.2.
Pour l'installer sur Raspbian 64bit il faut installer git en premier `apt-get update && apt-get -y install git` puis procéder ainsi:

```
cd ~
git clone https://github.com/DeskPi-Team/deskpi.git
cd ~/deskpi/
./install.sh
```

Méthode 2: Gadget Ethernet

Le gadget Ethernet est un peu plus difficile à configurer, mais il est beaucoup plus puissant car on peut utiliser d'autres OS. Fondamentalement, on aura la possibilité de se connecter à la console ainsi que tout ce qu'on peut faire sur une connexion réseau

Même s'il s'appelle “Ethernet Gadget”, on n'utilise pas de câble Ethernet, Le seul câble est le câble USB micro-B de l'ordinateur au Pi Zero. Le Pi « apparaît » comme un périphérique Ethernet.

On peut même partager la configuration réseau de l'ordinateur de bureau afin que le Pi puisse accéder à Internet via cet ordinateur via le câble USB.

Depuis mai 2016, Raspbian Jessie ne nécessite pas de nouveau noyau et a la configuration par défaut de raspberrypi.local donc c'est beaucoup plus facile

- **Étape 0** Télécharger et installer la dernière version de Raspbian

Une version Lite suffit, mais la simple Raspbian devrait également fonctionner.

FreeBSD prend en charge le Raspberry Pi original depuis novembre 2012 et Raspberry Pi 2 depuis mars 2015:
- Pour Raspberry Pi Zero utiliser l'image pour **RPI-B**,
- Pour Banana Pi zero (Allwinner H2+, armv7) utiliser l'image pour **RPI2-12+**
- Pour Raspberry Pi 4 et Pi 400, utiliser l'image pour **RPI3** (Pi 400 est connu pour fonctionner avec l'instantané 20210624)

- **Étape 1**: Configuration du nœud principal

Il faut commencer par configurer le nœud maître (le controleur) :

```
ssh pi@192.168.2.18
```

Le mot de passe par défaut est raspberry

Il faut installer quelques mises à jour sur le logiciel principal du Raspberry Pi. La première commande à exécuter est rpi-update.

```
sudo rpi-update
```

Cela mettra à jour le Raspberry Pi avec la dernière version du logiciel de base et du micrologiciel du système - cela prendra quelques minutes.

Une fois la mise à jour éffectuée il faut de nouveau redémarrer le Raspberry Pi pour que cela prenne effet.

```
sudo reboot
```

Cela prendra un peu de temps.

Par défaut, les images de ClusterHat utilisent le schéma **p[1–4]** pour les nœuds et **controler** pour le contrôleur Pi. Pour se conformer à ces spécifications, il est préférable de changer le nom d'hôte des Pis afin de rendre la vie beaucoup plus facile.

```
sudo hostname pi0
sudo vi /etc/hostname # changer le nom d'hôte dans ce fichier
sudo vi /etc/hosts # changer 'contrôleur' □□en pi0
```

Il faut s'assurer de conserver le schéma de numérotation en place. Par exemple, si on préfère qu'ils soient tous appelés node, il faut les nommer node00, node01, node02, node03 et node04.

Au prochain redémarrage, on remarquera que le prompt de la ligne de commande a changé en `pi@your-hostname-here`

Maintenant, il faut configurer l'interface WiFi (wlan0) et l'interface Ethernet (eth0) du réseau.

Modifier le fichier `/etc/dhcpcd.conf` et ajouter une configuration eth0 tout en bas, puis enregistrer.

```
interface eth0
static ip_address=10.0.0.1/8
static domain_name_servers=1.1.1.1,208.67.222.222
nolink
```

Ensuite, configurer l'interface WiFi pour vous connecter au WiFi domestique. Les détails de la connexion WiFi sont enregistrés dans `/etc/wpa_supplicant/wpa_supplicant.conf` mais il est préférable d'utiliser l'outil de configuration intégré (raspi-config) pour effectuer la configuration WiFi.

Dans Options réseau entrer le informations WiFi. Enregistrer/Terminer ensuite.

OPTIONNEL: Il est possible de configurer un serveur DHCP sur ce nœud afin de servir les adresses IP des computes nodes du cluster. Pour cela installer **dnsmasq**:

```
sudo apt update
sudo apt install dnsmasq
sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.backup
```

Une fois installé, configurer le serveur dhcp, dans `/etc/dnsmasq.conf`:
- Supprimer le # au début de la ligne **domain-needed**.
- Supprimer le # au début de la ligne **bogus-priv**.
- Modifier la ligne pour lire commençant par **server=/** par: `server=/cluster/<adresse IP du nœud principal>`
- Remplacer la ligne commençant par **local=/** et par: `local=/cluster/`
- Supprimer le # au début de la ligne **expand-hosts**.
- Remplacer la ligne commençant par **#domaine=** par: `domaine=cluster`
- Remplacer la ligne commençant par **#dhcp-range=** par: `dhcp-range=192.168.2.30,192.168.2.100,14d`. Cela définira la plage d'adresses IP

pouvant être attribuées aux machines clientes (192.168.2.30-192.168.2.100) et pendant combien de temps ces adresses leur sont attribuées avant leur renouvellement, appelée durée de bail, dans ce cas 14 jours.

Démarrer maintenant le service: `sudo service dnsmasq start`

Pour activer le partage internet sur tous les interfaces, d'abord, activer le transfert IP. Modifier le fichier `/etc/sysctl.conf` et décommenter cette ligne :

```
net.ipv4.ip_forward=1
```

Cela permet d'utiliser des règles NAT avec iptables.

Cconfigurer certaines règles POSTROUTING et FORWARD dans iptables pour permettre aux appareils Raspberry Pi sur le réseau 10.0.0.0/8 d'accéder à Internet via l'interface wlan0.

```
sudo iptables -t nat -A POSTROUTING -o wlan0 -j MASQUERADE
sudo iptables -A FORWARD -i wlan0 -o eth0 -m state --state
RELATED,ETABLISHED -j ACCEPT
sudo iptables -A FORWARD -i eth0 -o wlan0 -j ACCEPT
```

Cependant, lorsqu'on redémarre le Pi, ce partage sera perdu.

Pour rendre les iptables persistants et le transfert permanent:
- supprimer le # devant la ligne `#net.ipv4.ip_forward=1` dans le fichier `/etc/sysctl.conf`
- exécuter les commandes suivantes pour que les modifications d'iptables se chargent à chaque fois : `sudo apt-get install iptables-persistent`

- **Étape 2**: Installation de clusterctl

Lorsqu'on utilise l'une des images ClusterCTRL fournies, l'outil **cluserctrl** est installé par défaut, mais on peut l'installer manuellement en suivant les étapes ci-dessous en tant qu'utilisateur root (on peut généralement basculer vers l'utilisateur root en utilisant “sudo -i”).

```
TMPINSTALL=/tmp/clusterhat
apt install git libusb-1.0-0-dev python-usb python-libusb1 python-smbus #
utilisation de Python2 (par exemple Raspberry Pi OS)
apt install git libusb-1.0-0-dev python3-usb python3-libusb1 python3-smbus
python-is-python3 # utilisation de Python3 (par exemple Ubuntu 20.10)
git clone https://github.com/burtyb/clusterhat-image.git $TMPINSTALL
mkdir /usr/share/clusterctrl/
cp $TMPINSTALL/files/usr/sbin/cluster* /usr/sbin/
cp $TMPINSTALL/files/usr/share/clusterctrl/default-clusterctrl
/etc/default/clusterctrl
```

```
cp $TMPINSTALL/files/etc/udev/rules.d/90-clusterctrl.rules
/etc/udev/rules.d/
cp $TMPINSTALL/files/usr/lib/systemd/system/clusterctrl-init.service
/usr/lib/systemd/system/
cp -r $TMPINSTALL/files/usr/share/clusterctrl/python/
/usr/share/clusterctrl/
echo 'TYPE=c' >> /etc/default/clusterctrl
udevadm control --reload-rules
systemctl enable clusterctrl-init
raspi-config nonint do_i2c 0 # Not needed on Ubuntu 20.10
rm -rf $TMPINSTALL
```

Déconnecter et reconnecter maintenant le câble USB, puis vérifier qu'il a trouvé le périphérique.

```
clusterctrl status
```

Alors que le script clusterhat fournit un contrôle de l'alimentation des zéros pi attachés, il ne permet pas de les arrêter proprement avant de couper l'alimentation. Le script fournit sur https://github.com/thagrol/clusterctl tente de fournir ceci.

Configuration du contrôleur:
- Démarrer le contrôleur (maître)
- Se connecter en tant que pi
- Générer des clés ssh : `ssh-keygen -t rsa` (accepter la valeur par défaut pour “Enter file in which to save the key ” et ne pas définir de phrase secrète.
- Installer les dépendances : `sudo apt install python-pip python-gpiozero git`
- Télécharger les fichiers **clusterctl**: `git clone https://github.com/thagrol/clusterctl`
- ou alors:
`wget https://raw.githubusercontent.com/thagrol/clusterctl/master/clusterctl.py`
`chmod a+x clusterctl.py`
`wget https://raw.githubusercontent.com/thagrol/clusterctl/master/gpiosetup.sh`
corriger les droits d'exécution: `chmod a+x gpiosetup.sh`
- Configurer **gpiosetup.sh** pour qu'il s'exécute à chaque démarrage en ajoutant `/path/to/gpiosetup.sh &` immédiatement au-dessus de la ligne `exit(0)`dans le fichier `/etc/rc.local`
- Éteindre le maître : `sudo poweroff`
- Débrancher l'alimentation.
- S'il n'est pas déjà connecté, attacher clusterhat et pi zéro(s).
- Rebrancher l'alimentation et démarrer le maître.
Configuration des computes nodes (esclaves):
- Mettre le compute node sous tension : `/path/to/clusterctl.py -n 1 on`
- Se connectere via ssh : `ssh p1.local`
- Générer des clés ssh : `ssh-keygen -t rsa` (accepter la valeur par défaut pour “Enter

file in which to save the key ” et ne pas définir de phrase secrète)
- Copier la clé publique du maître : `scp pi@controller.local:/home/pi/.ssh/idras.pub ~/.ssh/authorized_keys`
- Se déconnecter.
- De retour sur le contrôleur (maître), mettre l'esclave hors tension : `/path/to/clusterctl.py -n 1 stop`
\\Ces étapes doivent être répétées pour chaque pi esclave. Pour éviter les problèmes causés par un bloc d'alimentation inadéquat, un seul esclave est mis sous tension à la fois. Modifier les noms d'hôtes et les numéros si nécessaire

- **Étape 3**: configuration des computes nodes

Après avoir gravé la carte SD, ne pas l'éjecter et utiliser un éditeur de texte pour ajouter la ligne suivante comme dernière ligne dans le fichier `config.txt`:

```
dtoverlay=dwc2
```

Ensuite, ajouter dans le fichier `cmdline.txt`, après **rootwait** (le dernier mot sur la première ligne), avec une espace devant:

```
modules-load=dwc2,g_ether
```

Sur **FreeBSD**, pour activer le device ethernet de Gadget USB, ajouter ces lignes à `/boot/loader.conf`, en le créant s'il n'existe pas déjà :

```
if_cdce_load="YES"
hw.usb.template=1
```

Pour charger le module et définir le modèle sans redémarrer, utiliser :

```
# kldload if_cdce
# sysctl hw.usb.template=1
```

Enfin, ajouter un fichier `ssh` à la racine de la partition (vide, c’est sans importance).

Une fois que les images sont gravées et SSH activé, placer les cartes SD dans leur Raspberry Pis respectif.

Maintenant, on peut démarrer le Raspberry Pi avec la carte. Après quelques secondes (c’est un peu plus long que d’habitude), il devrait apparaître comme un adaptateur Ethernet (RNDIS/Ethernet Gadget) dans les connexions de l'ordinnateur. Il suffit ensuite d’ouvrir le Terminal et de taper la ligne suivante (le mot de passe sera raspberry).

```
ssh pi@raspberrypi.local
```

Il faut brancher le câble USB de l'ordinateur sur le port du connecteur “USB” du Pi Zero, et non sur le connecteur PWR.

Bonjour doit être installé sur l'ordinateur local afin qu'il sache quoi faire avec les noms .local

Il faut installer quelques mises à jour sur le logiciel principal du Raspberry Pi. La première commande à exécuter est rpi-update.

```
sudo rpi-update
```

Cela mettra à jour le Raspberry Pi avec la dernière version du logiciel de base et du micrologiciel du système - cela prendra quelques minutes.

Une fois la mise à jour éffectuée ilf faut de nouveau redémarrer le Raspberry Pi pour que cela prenne effet.

```
sudo reboot
```

Cela prendra un peu de temps.

Comme pour le contrôleur, il est préférable de changer le nom d'hôte des Pis afin de rendre la vie beaucoup plus facile.

```
sudo hostname pi1
sudo vi /etc/hostname # changer le nom d'hôte dans ce fichier
sudo vi /etc/hosts # changer 'contrôleur' □□en pi1
```

Maintenant, il faut configurer l'interface WiFi (wlan0) et l'interface Ethernet (eth0) du réseau.

Modifier le fichier `/etc/dhcpcd.conf` et ajouter une configuration eth0 tout en bas, puis enregistrer.

```
interface eth0
static ip_address=10.0.0.1/8
static domain_name_servers=1.1.1.1,208.67.222.222
nolink
```

Ensuite, configurer l'interface WiFi pour vous connecter au WiFi domestique. Les détails de la connexion WiFi sont enregistrés dans `/etc/wpa_supplicant/wpa_supplicant.conf` mais il est préférable d'utiliser l'outil de configuration intégré (raspi-config) pour effectuer la configuration WiFi.

Dans Options réseau entrer le informations WiFi. Enregistrer/Terminer ensuite.

Premier démarrage

Lorsqu'on utilise une image ClusterHat Raspbian le nom d'utilisateur par défaut est **pi** et le mot de passe est **clusterctrl**

Avant de faire quoi que ce soit d'autre, il faut modifier le mot de passe par défaut sur l'image du contrôleur. Lorsqu'on utilise l'image CBRIDGE à l'étape précédente, il faut également le faire sur chacun des PI Zero.

Une fois connecter taper la commande `sudo raspi-config` et appuyer sur Entrée/Retour, Cela exécutera un programme de configuration Raspberry Pi fourni.

La commande sudo signifie que toutes les commandes suivantes seront exécutées en tant que super-utilisateur autorisé à apporter des modifications au système plutôt que de simplement modifier les fichiers d'un utilisateur.

Lorsque le menu apparaît :

- Choisir l'option 3 Boot Options.
- Choisir maintenant l'option B1 Text console et appuyer sur Entrée/Retour.

Cela signifie que le Raspberry Pi démarrera toujours sur une ligne de commande plutôt que sur une interface graphique:

- Choisir maintenant l'option 9 Options avancées.
- Choisir le nom d'hôte de l'option A2 et appuyer sur entrée/retour.

Sur cet écran, il y a un nom d'hôte, un nom facile à reconnaître, pour le Raspberry Pi auquel on ciens de se connecté. Pour le rendre plus identifiable et personnaliser le cluster, on peut choisir un nom pour le Raspberry Pi.

On va changer le nom en **p0** et appuyer sur entrée/retour pour le confirmer.

Il faut définir le bon fuseau horaire pour que la date et l'heure sur le Pi soient correctes.

- Choisir l'option 5 Options d'internationalisation.
- Choisir l'option T2 Changer le fuseau horaire.
- Dans la liste Choisir la région, (**Europe**).
- Ensuite, Choisir l'emplacement le plus proche, (**Paris**).

Il est très important de définir les options appropriées pour le clavier et dee fuseau horaire de tous les Raspberry Pis, car ils devront avoir la même heure système plus tard.

Une fois de retour dans le menu principal, utiliser les touches curseur/flèche pour accéder au bouton

qui dit Terminer et appuyer sur entrée/retour, puis Choisir oui et redémarrer ce Pi.

Se reconnecter au Raspberry Pi à l'aide de la commande ssh précédemment utilisée et saisir le mot de passe.

Configurer SSH

Après avoir défini les paramètres sur le contrôleur, redémarrer le Pi. Une fois le Pi redémarré, configurer une clé SSH avec les commandes suivantes :

```
ssh-keygen -t rsa -b 4096
cat ~/.ssh/id_rsa.pub
```

Après avoir généré une clé ssh et l'avoir affichée, copier-la car on en aura besoin dans les étapes suivantes.

Configurer un fichier de configuration pour les noms SSH **~/.ssh/config** avec le contenu suivant (si on utilise l'image CBRIDGE, ignorer cette étape) :

```
Host p1
    Hostname 172.19.181.1
    User piHost p2
    Hostname 172.19.181.2
    User piHost p3
    Hostname 172.19.181.3
    User piHost p4
    Hostname 172.19.181.4
    User pi
```

Cette étape est superflue, mais elle permet de se connecter rapidement en ssh à l'un des nœuds du contrôleur Pi avec la commande ssh hostname où hostname est le nom d'hôte des nœuds : p1, p2, p3 ou pi4.

Activer les nœuds

On peut maintenant démarrer les nœuds. Après avoir exécuté la commande suivante, on doit voir les voyants du nœud s'allumer en orange un par un à mesure que chaque Pi Zero se met en ligne :

```
sudo clusterhat on
```

Chaque fois qu'on doit arrêter les nœuds, on peut utiliser la commande suivante pour les désactiver. C'est très utile lorsqu'il faut les redémarrer tous en même temps :

```
sudo clusterhat off
```

Cette commande effectue un démarrage en dur qui coupe l'alimentation des nœuds. Si on exécute cette commande pendant qu'ils écrivent sur la carte SD, il y a un risque de corrompre la carte SD, ce qui est catastrophique.

Configuration des noeuds

On peut maintenant se connecter à chacun des nœuds pour les configurer.

Mot de passe

Répéter le mot de passe et la localisation configurés pour chaque nœud. Il faut également mettre la clé ssh copiée précédemment dans le fichier `~/.ssh/authorized_keys`:

```
ssh p1
# le mot de passe par défaut pour les nœuds est clusterctl

echo [paste the ssh key here] >> ~/.ssh/authorized_keys
sudo raspi-config
```

Après avoir configuré la localisation et ssh sur chacun des nœuds, installer le package **ntpdate**. Cela gardera l'heure système synchronisée sur les nœuds en arrière-plan :

```
sudo apt-get install -y ntpdate
```

Noms d'utilisateurs

Si on veut modifier le nom d'utilisateur par défaut, il faut le faire maintenant, mais les utilistaeurs doivent avoir le même UID sur tous les nœuds et le contrôleur pi, et les choses seront plus fluides si tous les utilisateurs ont le même nom d'utilisateur. Il est préférable de rester avec l'utilisateur pi par défaut.

Configurer un Drive partagé

Il n'est pas nécessaire d'avoir un lecteur partagé pour exécuter des commandes sur le cluster, mais il est extrêmement utile d'avoir une partition pour contenir des données que l'ensemble du cluster peut utiliser.On peut configurer et utiliser une partition sur la carte SD du contrôleur, une clé USB supplémentaire ou un partage réseau.

Pour ce tuturiel on va configurer un partage NFS avec le contrôleur Pi, pour savoir où le périphérique est chargé dans `/dev`, exécuter la commande **lsblk** et vous devrier voir quelque chose comme ceci :

```

```

```
$ lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sda 8:0 1 119.5G 0 disk
└─sda1 8:1 1 119.5G 0 part
mmcblk0 179:0 0 29.7G 0 disk
├─mmcblk0p1 179:1 0 256M 0 part /boot
└─mmcblk0p2 179:2 0 29.5G 0 part /
```

Le périphérique se trouve dans `/dev/sda1`. Ensuite il faut partitionner le lecteur :

```
sudo mkfs.ext4 /dev/sda1
```

Après avoir formaté le lecteur, configurer un répertoire pour le monter. Il est préférable dutiliser le dossier `/media` pour configurer les lecteurs externes, mais on peut placer le dossier n'importe où sur le système de fichiers. Il faut simplement s'assurer que ce sera le même dossier sur tous les nœuds.

```
sudo mkdir /media/Storage
sudo chown nobody.nogroup -R /media/Storage
sudo chmod -R 777 /media/Storage
```

On a configuré les autorisations les plus lâches pour ce lecteur. Toute personne ayant accès au pi pourra lire, modifier ou supprimer le contenu de ce lecteur. Pour utiliser cette machine dans un environnement de production ou pour traiter des données sensibles, il faut utiliser différentes autorisations dans cette étape précédente.

Après avoir configuré cela, exécuter la commande **blkid** pour obtenir l'UUID du lecteur afin que l'on puisse configurer le montage automatique du lecteur à chaque démarrage du pi, rechercher une ligne comme celle-ci :

```
/dev/sda1: LABEL=”sammy” UUID=”a13c2fad-7d3d-44ca-b704-ebdc0369260e”
TYPE=”ext4" PARTLABEL=”primary” PARTUUID=”d310f698-d7ae-4141-
bcdb-5b909b4eb147"
```

Bien que la partie la plus importante soit l'UUID, `UUID="a13c2fad-7d3d-44ca-b704-ebdc0369260e"`. Modifier fstab pour qu'il contienne la ligne suivante, tout en s'assurant de remplacer l'UUID des lecteurs à l'emplacement approprié :

```
sudo vi /etc/fstab# Ajouter la ligne suivante au bas du fichier fstab :
UUID=a13c2fad-7d3d-44ca-b704-ebdc0369260e /media/Storage ext4 defaults 0 22
```

Installer le serveur NFS sur le contrôleur :

```
sudo apt-get install -y nfs-kernel-server
```

Il faut ensuite mettre à jour le fichier `/etc/exports` pour qu'il contienne la ligne suivante en bas :

```
/media/Storage 172.19.181.0/24(rw,sync,no_root_squash,no_subtree_check)
```

Si on utilise l'image CBRIDGE, il faut utiliser le schéma d'adresse IP due réseau. Par exemple, si le réseau utilise 192.168.1.X, il faut remplacer 172.19.181.0 par 192.168.1.0 dans l'exemple ci-dessus.

Après avoir modifié le fichier d'exportation, exécuter la commande `sudo exportfs -a` pour mettre à jour le serveur NFS.

On peut maintenant monter le partage NFS que l'on a configuré sur chacun des nœuds Pis. Exécuter le jeu de commandes suivant sur chaque nœud :

```
sudo apt-get install -y nfs-common# Créer le dossier de montage, en
utilisant le même dossier de montage ci-dessus.
# Si vous aver utilisé des autorisations différentes, utiliser les mêmes
autorisations ici.sudo mkdir /media/Storage
sudo chown nobody.nogroup /media/Storage
sudo chmod -R 777 /media/Storage# Configurer le montage automatique en
éditant votre /etc/fstab:sudo vi /etc/fstab# Ajouter cette ligne en bas :
172.19.181.254:/media/Storage /media/Storage nfs par défaut 0 0
```

Puis exécuter la commande `sudo mount -a` et le lecteur NFS sera monté sur le nœud. Si il y a des erreurs, vérifier les fichiers `/etc/fstab` dans les nœuds et le fichier `/etc/exports` sur le contrôleur. Créer un fichier de test dans le répertoire de montage NFS pour s'assurer que l'on peut le voir le fichier sur tous les nœuds :

```
echo "Ceci est un test" >> /media/Storage/test
```

On doit pouvoir voir ce fichier sur tous les nœuds et le modifier également.

Configuration du clustering

SLURM est l'acronyme de **S**imple **L**inux **U**tility for **R**esource **M**anagement. Développé à l'origine sur Linux en gardant à l'esprit les leçons des anciens planificateurs de tâches, SLURM est est un job scheduler, il gère les ressources telles que les cœurs de processeur et la mémoire sur un cluster, son rôle est de maintenir une image aussi précise que possible de l'état d'utilisation des ressources sur le cluster. Il se base sur cette image pour placer les jobs des utilisateurs en fonction des ressources libres.

L'utilisateur se connecte au contrôleur et utilise la commande **srun** ou **sbatch** pour envoyer un job sur le cluster. Une connexion à un service **slurmctld** en exécution sur le nœud d'administration est réalisée. Ce service connaît la topologie du cluster ainsi que l'état d'utilisation des ressources. Si les

ressources demandées par l'utilisateur sont libres, **slurmctld** demande au(x) service(s) **slurmd** de(s) noeud(s) de calcul(s) réservé(s) d'exécuter le job.

Le service **slurmd** accepte la requête du service **slurmctld** et fork immédiatement un processus **slurmstepd** qui va manager la consommation de ressources du job ainsi que ses entrées/sorties. Le job est un processus fils de **slurmstepd** exécuté sous l'identité de l'utilisateur (setuid).

Les communications entre **slurmctld** et **slurmd** sont authentifiées avec Munge (clé privée partagée par les nœuds). Il faut donc installer Munge sur tous les nœuds du cluster.

Installation et configuration de Munge

Pour configurer **munge**, il faut d'abord éditer le fichier `/etc/hosts` pour qu'il contienne les adresses et les noms d'hôtes des autres nœuds du cluster. Cela rendra la résolution de noms beaucoup plus facile et éliminera tout travail de devinette pour le Pis. Il faut éditer le fichier `/etc/hosts` sur chacun des nœuds et le contrôleur pour y mettre toutes les adresses IP et les noms d'hôte, à l'exception de celui de la machine.

Par exemple, ajouter ces lignes au fichier `/etc/hosts` du contrôleur :

```
172.19.181.1 pi1
172.19.181.2 pi2
172.19.181.3 pi3
172.19.181.4 pi4
```

Sur le premier nœud (pi1), ajouter ces lignes :

```
172.19.181.254 controller
172.19.181.2 pi2
172.19.181.3 pi3
172.19.181.4 pi4
```

Répéter ce processus pour les trois autres nœuds. Après avoir modifié le fichier hosts, installer et configurer **munge** sur chacun des PI. On va commencer par le contrôleur.

```
sudo apt-get install -y munge
```

> WRAP round help>Sur **FreeBSD**, on peut installer le dernier **munge** stable sous forme de paquet binaire en utilisant :\\ \\ `pkg install munge`\\ \\ Ou, il peut être construit et installé à partir de la source en utilisant :\\ \\ `cd /usr/ports/security/munge && make install`</WRAP>

Après cela, activer et démarrer le service munge :

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

Vérifier l'état pour s'assurer que **munge** a démarré correctement sans aucun problème :

```
sudo systemctl status munge
```

Après avoir configuré cette configuration sur le contrôleur Pi, copier le fichier de clé Munge sur le NFS configuré précédemment. on aura besoin de cette même clé sur chacun des nœuds :

```
sudo cp /etc/munge/munge.key /media/Storage
```

Si munge ne démarre pas normalement il faut vérifier la propriété du fichier de l'ensemble de l'arborescence racine. Par exemple lorsque l'arborescence racine appartient à l'utilisateur par défaut (UID=1000), tmpfiles.d ne créera pas les répertoires nécessaires à munge pour démarrer. Pour résoudre ce problème, exécuter les commandes suivantes :

```
sudo chown root:root /
sudo chown root:root /etc
sudo chown root:root /sbin
sudo chown root:root /usr
```

Cela rétablira la propriété de la structure du fichier là où elle doit être. Il faut le faire sur tous les nœuds et configurer munge

Il faut s'assurer que tous les nœuds du cluster ont le même `munge.key`, répéter les étapes ci-dessus sur chacun des nœuds du cluster, sauf qu'au lieu de copier la clé du nœud vers le NFS, copier la clé du contrôleur vers le nœud, en écrasant la clé installée par munge :

```
sudo cp /media/Storage/munge.key /etc/munge
```

Il faut s'assurer que l'utilisateur et le groupe munge sont les seuls utilisateurs à disposer d'autorisations de lecture sur la clé munge sur chacun des nœuds en exécutant `sudo ls -la /etc/munge`. La sortie devrait ressembler à ceci :

```
$ sudo ls -la /etc/mungetotal 12
drwx------  2 munge munge 4096 Aug 3 20:52 .
drwxr-xr-x 93 dhuck dhuck 4096 Aug 3 21:37 ..
-r--------  1 munge munge 1024 Aug 3 20:52 munge.key
```

Afin de tester que munge est correctement configuré, exécuter la commande suivante :

```
sudo apt-get install -y munge
```

Tant qu'on n'est pas sur pi1, on doit voir quelque chose qui ressemble à cette sortie :

```
```

```
STATUS: Success (0)
ENCODE_HOST: medusa (127.0.1.1)
ENCODE_TIME: 2019-08-05 20:19:54 +0100 (1565032794)
DECODE_TIME: 2019-08-05 20:19:54 +0100 (1565032794)
TTL: 300
CIPHER: aes128 (4)
MAC: sha256 (5)
ZIP: none (0)
UID: dhuck (1000)
GID: dhuck (1000)
LENGTH: 0
```

Installation et configuration de Slurm

On est maintenant prêts à installer slurm et à obtenir le clustering

Sur le contrôleur, exécuter les commandes suivantes:

- installer slurm :

```
sudo apt-get install -y slurm-wlm
```

Cela prendra un moment. Une fois terminé, on utilisera la configuration de slurm par défaut et la modifiera pour répondre aux besoins.

Sur **FreeBSD**, on peut installer le dernier **Slurm** stable sous forme de paquet binaire en utilisant :

```
pkg installer slurm-wlm
```

Ou, il peut être construit et installé à partir de la source en utilisant :

```
cd /usr/ports/sysutils/slurm-wlm && make install
```

Le package binaire installe une configuration **Slurm minimale** adaptée aux nœuds de calcul typiques. L'installation à partir des sources permet à l'utilisateur d'activer des options telles que les outils **mysql** et **gui** via un menu de configuration.

- Copier le fichier de configuration depuis le dossier de documentation slurm :

```
cd /etc/slurm-llnl
cp /usr/share/doc/slurm-client/examples/slurm.conf.simple.gz .
gzip -d slurm.conf.simple.gz
mv slurm.conf.simple slurm.conf
```

Ouvrir le fichier `/etc/slurm-llnl/slurm.conf` et effectuer les modifications suivantes :

1. Définir les informations de la machine de contrôle : `SlurmctldHost=controller(172.19.181.254` ceci peut changer lorsqu'on utilise le CBRIDGE ou si on modifié le nom d'hôte
2. Définir le paramètre **SelectType** sur les valeurs suivantes : `SelectType=select/cons_res`
3. Définir le paramètre **SelectTypeParameters** sur les valeurs suivantes : `SelectTypeParameters=CR_Core`
4. Pour modifier ou définir le nom du cluster, le définir avec le paramètre **ClusterName** : `ClusterName=cluster`
5. À la fin du fichier:
    1. supprimer l'entrée par défaut pour **PartitionName** à la fin du fichier et la remplacer par le nom de partition personnalisé. Le bloc de code suivant doit être entièrement sur une seule ligne: `PartitionName=mycluster Nodes=p[1–4] default=YES MaxTime=INFINITE State=UP`
    2. il doit y avoir une entrée pour un nœud de calcul, le supprimer et mettre à sa place :

```
NodeName=p0 NodeAddr=172.19.181.254 CPU=2 Weight=2 State=UNKNOWN
NodeName=p1 NodeAddr=172.19.181.1 CPUs=1 Weight=1 State=INCONNU
NodeName=p2 NodeAddr=172.19.181.2 CPUs=1 Weight=1 State=INCONNU
NodeName=p3 NodeAddr=172.19.181.3 CPUs=1 Weight=1 State=INCONNU
NodeName=p4 NodeAddr=172.19.181.4 CPUs=1 Weight=1 State=UNKNOWN
```

Si on utilise l'image CBRIDGE pour le contrôleur, les adresses IP seront différentes. Idem si on a changé l'un des noms d'hôte

Dans cet exemple on n'alloue que 2 processeurs du contrôleur pi vers le cluster. Si on veutr ajouter tous les CPU, changer CPUs=2 en CPUs=4. Cependant, il est recommandé de laisser quelques-uns des processeurs sur le contrôleur Pi pour gérer le cluster.

- Créer le fichier suivant, `/etc/slurm-llnl/cgroup.conf` avec les lignes suivantes pour indiquer à slurm à quelles ressources il peut accéder sur les nœuds:

```
CgroupMountpoint=”/sys/fs/cgroup”
CgroupAutomount=yes
CgroupReleaseAgentDir=”/etc/slurm-llnl/cgroup”
AllowedDevicesFile=”/etc/slurm-llnl/cgroup_allowed_devices_file.conf”
ConstrainCores=no
TaskAffinity=no
ConstrainRAMSpace=yes
ConstrainSwapSpace=no
ConstrainDevices=no
AllowedRamSpace=100
AllowedSwapSpace=0
MaxRAMPercent=100
MaxSwapPercent=100
```

```
MinRAMSpace=30
```

- Créer et écrire les lignes suivantes afin de mettre en liste blanche les périphériques système dans `/etc/slurm-llnl/cgroup_allowed_devices_file.conf`:

```
/dev/null
/dev/urandom
/dev/zero
/dev/sda*
/dev/cpu/*/*
/dev/pts/*
/media/Storage* # nom du lecteur NFS à adapter
```

Ce sont des valeurs avec lesquelles on peut jouer pour explorer comment le cluster computing affectera les ressources. Il s'agit d'une configuration très permissive qui peut être modifiée.

- Copier ces fichiers de configuration sur le lecteur NFS que l'on a configuré précédemment :

```
sudo cp slurm.conf cgroup.conf cgroup_allowed_devices_file.conf
/media/Storage
```

- Enfin, activer et démarrer le démon slurm et le démon de contrôle slurm sur le contrôleur Pi :

```
sudo systemctl enable slurmd
sudo systemctl start slurmd

sudo systemctl enable slurmctld
sudo systemctl start slurmctld
```

Sur chacun des compute nodes il faut effectuer les opérations suivantes:

- Installer le client slurm sur les nœuds :

```
sudo apt-get install -y slurmd slurm-client
```

- Copier les fichiers de configuration sur chacun des nœuds :

```
sudo cp /clusterfs/slurm.conf /etc/slurm-llnl/slurm.conf
sudo cp /clusterfs/cgroup* /etc/slurm-llnl
```

- Enfin, activer et démarrer le démon slurm sur chaque nœud :

```
sudo systemctl enable slurmd
sudo systemctl start slurmd
```

- Se connecter au nœud de contrôleur et tester slurm pour s'assurer qu'il fonctionne.

Exécuter la commande `sinfo` et vous doit obtenir le résultat suivant :

```
PARTITION  AVAIL TIMELIMIT NODES STATE NODELIST
mycluster*    up  infinite     4  idle   p[1–4]
```

De plus, on peut exécuter la commande **hostname** sur tous les nœuds avec la commande suivante :

```
srun --nodes=5 hostname
```

Ce qui devrait produire un résultat similaire à ce qui suit :

```
p0
p4
p2
p1
p3
```

Pour voir combien de tâches simultanées on peut exécuter, utiliser la commande suivante pour vérifier :

```
srun --ntasks hostname
```

Ce qui devrait produire un résultat similaire à ce qui suit :

```
p0
p0
p4
p2
p1
p3
```

Dans la commande, p0 apparaît deux fois, car il contribue à 2 cœurs au cluster.

Partie II — Quelques tâches simples

Maintenant que le cluster est opérationnel, On peut commencer à exécuter des tâches dessus. Dans cette partie, on va voir les bases de Slurm, configurer facilement des logiciels sur le cluster et créer des exemples de tâches qui exécutent de nombreuses tâches individuelles à l'aide du planificateur.

Les commandes Slurm de base

Comme indiqué précédemment, Slurm est un logiciel appelé ordonnanceur. Cela permet de soumettre des tâches qui demandent une quantité spécifique de ressources, telles que des cœurs de processeur, de la mémoire ou des nœuds de calcul entiers. Le planificateur exécutera chaque tâche au fur et à mesure que les ressources seront disponibles. Cela signifie qu'on peut laisser autant de

ressource que l'on veut, et il le découvrira.

sinfo

Slurm fournit plusieurs outils de ligne de commande utiles qu'on utilisera pour l'interface avec le cluster. Se connecter au nœud maître/contrôleur:

```
ssh pi@node01
```

La première commande que l'on examinera est sinfo. C'est asser simple, il fournit juste des informations sur le cluster :

```
$ sinfo

PARTITION   AVAIL   TIMELIMIT   NODES   STATE   NODELIST
glmdev*     up      infinite    3       mix     node[1–3]
```

Ici, on voit le nom de la partition, si elle peut être utilisée, le délai par défaut, le nombre de nœuds et leurs états. L'état « mix » se produit lorsqu'un nœud a une tâche en cours d'exécution, mais qu'il a encore des ressources disponibles. (Comme lorsqu'un seul noyau est utilisé.)

srun — ordonnancement des commandes

La commande srun est utilisée pour exécuter directement une commande sur le nombre de nœuds/cœurs que l'on souhaite:

```
$ srun --nodes=3 hostname
node1
node2
node3
```

Ici, on exécute la commande **hostname** sur 3 nœuds. C'est différent de l'exécuter sur 3 cœurs, qui peuvent tous être sur le même nœud :

```
$ srun --ntasks=3 hostname
node1
node1
node1
```

Ici, **ntasks** fait référence au nombre de processus. C'est effectivement le nombre de cœurs sur lesquels la commande doit être exécutée. Ce ne sont pas nécessairement sur des machines différentes. Slurm attrape juste les prochains cœurs disponibles.

On peut aussi combiner les deux :

```
$ srun --nodes=2 --ntasks-per-node=3 hostname
node1
node2
node2
node1
node2
node1
```

Cela exécute la commande sur 2 nœuds et lance 3 tâches par nœud, effectivement 6 tâches.

squeue — afficher les tâches planifiées

Lorsqu'on exécute des tâches de plus en plus longues, il est utile de vérifier leur statut. Pour ce faire, exécuter la commande **squeue. Par défaut**, il affiche toutes les tâches soumises par tous les utilisateurs et leurs états :

```
$ squeue
JOBID PARTITION  NAME     USER ST    TIME  NODES NODELIST(REASON)
  609 glmdev     24.sub.s pi   R     10:16     1 node2
```

La plupart de ces informations sont assez explicites, la colonne ST, indique l'état du travail. R signifie que le travail est en cours d'exécution. Voici la liste complète des codes d'état.

scancel — annuler une tâche planifiée

Une fois qu'une tâche a été planifiée, elle peut être annulée à l'aide de la commande scancel :

```
$ scancel 609
```

(où 609 est le JOBID que l'on souhaite annuler) On ne peut annuler que les travaux démarrés par l'utilisateur.

sbatch — planifier un script batch

sbatch est utilisé le plus souvent lorsqu'on veut planifier une tâche à exécuter sur le cluster. Cette commande prend un certain nombre d'indicateurs et de configuration, ainsi qu'un fichier shell. Ce fichier shell est ensuite exécuté une fois et toutes les ressources demandées (nœuds/cœurs/etc) sont mises à sa disposition. .

A titre d'exemple on va créer un travail de base:

Créer le fichier `/clusterfs/helloworld.sh` (ce fichier batch est généralement un script bash qui exécute le travail, mais il semble un peu différent):

```
#!/bin/bash
```

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --partition=<partition name>
cd $SLURM_SUBMIT_DIR
echo "Hello, World!" > helloworld.txt
```

- Le fichier commence par un shebang. Ceci est nécessaire, car il indique à Slurm comment exécuter lee travail
- Ceci est suivi d'un certain nombre de flags qui prennent la forme suivante :`#SBATCH <flag>` (ces indicateurs indiquent simplement tous les paramètres pouvant être transmis à la commande sbatch)
- Le cd `$SLURM_SUBMIT_DIR` garantit que le travail s'exécute dans le répertoire à partir duquel il a été soumis. Dans le cas, il s'agit de /clusterfs.

Les tâches sont presque identiques à celles utilisées par la commande srun, mais avec une différence principale:
- les tâches ne sont pas automatiquement relancées sur chaque nœud/cœur spécifié.
- Au lieu de cela, chaque tâche est exécutée sur le premier cœur du premier nœud qui lui est alloué, mais la tâche a accès aux autres nœuds qu'elle a demandé.

Maintenant, On peut dire à Slurm de planifier et d'exécuter le travail :

```
$ sbatch ./helloworld.sh
Submitted batch job 639
```

Puisque le travail est très simple, il devrait être fait essentiellement immédiatement. Si tout s'est déroulé comme prévu, on doit voir le fichier `/clusterfs/helloworld.txt` que l'on a créé.

Le travail ne renvoie rien au shell, ce qui est logique. Si on a un travail qui dure des heures, il n'est pas très utile d'avoir un terminal ouvert tout le temps pour obtenir une sortie. Au lieu de cela, Slurm génère l'erreur standard et la sortie standard dans un fichier au format slurm-XXX.out où XXX est le numéro d'identification de la tâche.

Partie III — OpenMPI et logiciels partagés

Dans cette partie, on va configurer OpenMPI, installer une version de Python patagée par tous les nœuds et examiner l'exécution de certains travaux en parallèle pour utiliser les multiples nœuds du cluster.

OpenMPI

OpenMPI est une implémentation open source du concept d'interface de passage de messages. Un MPI est un logiciel qui connecte les processus exécutés sur plusieurs ordinateurs et leur permet de communiquer pendant leur exécution. C'est ce qui permet à un seul script d'exécuter un travail réparti sur plusieurs nœuds de cluster.

On va installer OpenMPI en toute simplicité. Bien qu'il soit possible de l'installer à partir des sources, il est plus difficile de le faire fonctionner correctement avec SLURM, de sorte que SLURM configure automatiquement l'environnement lorsqu'un travail est en cours d'exécution afin qu'OpenMPI ait accès à toutes les ressources que SLURM a allouées au travail.

Installation d'OpenMPI

Pour installer OpenMPI, se connecter en SSH au nœud principal du cluster et utiliser **srun** pour installer OpenMPI sur chacun des nœuds :

```
$ sudo su -
# srun --nodes=3 apt install openmpi-bin openmpi-common libopenmpi3
libopenmpi-dev -y
```

Test de programme MPI

Maintenant, pour le tester, on va créer un programme C très basique qui crée un cluster MPI avec les ressources que SLURM alloue à le travail. Ensuite, il va appeler une simple commande d'impression sur chaque processus:

Créer le fichier /clusterfs/hello_mpi.c

avec le contenu suivant :

```
#include <stdio.h>
#include <mpi.h>int main(int argc, char** argv){
    int node;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &node);    printf("Hello World from Node
%d!\n", node);    MPI_Finalize();
}
```

Ici, on inclus la bibliothèque **mpi.h** fournie par OpenMPI. Ensuite, dans la fonction principale, on:

1. initialise le cluster MPI,
2. obtiens le numéro du nœud sur lequel le processus en cours s'exécutera,
3. affiche un message et ferme le cluster MPI.

Compiler le programme.

Il faut compiler le programme C pour l'exécuter sur le cluster. Cependant, contrairement à un programme C normal, on n'utilisera pas gcc, mais le compilateur fournit OpenMPI, qui liera automatiquement les bibliothèques MPI.

Parce qu'on doit utiliser le compilateur fourni par OpenMPI, il faut récupérer une instance de shell à partir de l'un des nœuds :

```
login1$ srun --pty bash
node1$ cd /clusterfs
node1$ mpicc hello_mpi.c
node1$ ls
a.out* hello_mpi.c
node1$ exit
```

Le fichier `a.out` est le programme compilé qui sera exécuté par le cluster.

Créer un script de soumission.

Maintenant, créer le script de soumission qui exécute le programme sur le cluster. Créer le fichier `/clusterfs/sub_mpi.sh`:

```
#!/bin/bashcd $SLURM_SUBMIT_DIR
# Affiche le nœud qui démarre le processus
echo "Master node: $(hostname)"
# Exécute le programme en utilisant OpenMPI.
# OpenMPI découvrira automatiquement les ressources de SLURM.
mpirun a.out
```

Exécuter le travail.

Exécuter la tâche en la soumettant à SLURM et en demandant quelques nœuds et processus :

```
$ cd /clusterfs
$ sbatch --nodes=3 --ntasks-per-node=2 sub_mpi.sh
Submitted batch job 1211
```

Cela indique à SLURM d'obtenir 3 nœuds et 2 cœurs sur chacun de ces nœuds. Si tout fonctionne correctement, cela devrait créer un cluster MPI avec 6 nœuds. En supposant que cela fonctionne, on doit voir une sortie dans le fichier slurm-XXX.out :

```
Master node: node1
Hello World from Node 0!
```

```
Hello World from Node 1!
Hello World from Node 2!
Hello World from Node 3!
Hello World from Node 4!
Hello World from Node 5!
```

Installation d'un logiciel partagé

Jusqu'à présent, lorsqu'on a installé un logiciel sur le cluster, on l'a essentiellement fait individuellement sur chaque nœud. Bien que cela fonctionne, cela devient rapidement inefficace. Au lieu de dupliquer les efforts en essayant de s'assurer que les mêmes versions logicielles et le même environnement sont disponibles sur chaque nœud, on peut le compiler à partir de la source et le configurer pour qu'il soit installé dans un répertoire du stockage partagé. L'architecture ds nœuds étant identique, ils peuvent tous exécuter le logiciel à partir d'un stockage partagé.

Ceci est utile car cela signifie qu'on n'a à maintenir qu'une seule installation d'un logiciel et sa configuration. En revanche, la compilation à partir des sources est beaucoup plus lente que l'installation de packages pré-construits. Il est également plus difficile à mettre à jour.

Dans cette section, on va installer Python3 à partir des sources et l'utiliser sur less différents nœuds.

Prérequis

Pour que la construction Python se termine avec succès, il faut s'assurer que les bibliothèques nécessaires sont installées sur l'un des nœuds. On les installera que sur un nœud et on veillera à ne construire Python que sur ce nœud :

```
$ srun --nodelist=node1 bash
node1$ sudo apt install -y build-essential python-dev python-setuptools
python-pip python-smbus libncursesw5-dev libgdbm-dev libc6-dev zlib1g-dev
libsqlite3-dev tk-dev libssl-dev openssl libffi-dev
```

Bien que l'on puisse techniquement créer Python lui-même sans exécuter cette étape, on veut pouvoir accéder à Pip et à un certain nombre d'autres outils supplémentaires fournis avec Python. Ces outils ne compileront que si leurs dépendances sont disponibles.

Ces dépendances n'ont pas besoin d'être présentes pour utiliser le nouvelle installation Python, juste pour la compiler.

Télécharger Python

Télécharger une copie des fichiers source Python afin de pouvoir les construire. Créer un répertoire de construction dans le stockage partagé et y extraire les fichiers :

```
$ cd /clusterfs && mkdir build && cd build
$ wget https://www.python.org/ftp/python/3.7.3/Python-3.7.3.tgz
$ tar xvzf Python-3.7.3.tgz
... tar output ...
$ cd Python-3.7.3
```

À ce stade, on aura la source Python extraite dans le répertoire `/clusterfs/build/Python-3.7.3`.

Configurer Python

La première étape de la construction de Python consiste à configurer la construction dans l'environnement. Cela se fait avec la commande `./configure`. L'exécuter tout seul configurera Python pour qu'il s'installe dans le répertoire par défaut. Cependant, on ne veut pas cela, il faut donc lui passer l'indicateur `--prefix=`. Cela indiquera à Python d'installer dans un dossier du stockage partagé, et cela peut prendre un certain temps :

```
$ mkdir /clusterfs/usr        # directory Python will install to
$ cd /clusterfs/build/Python-3.7.3
$ srun --nodelist=node1 bash  # configure will be run on node1
node1$ ./configure \
            --enable-optimizations \
            --prefix=/clusterfs/usr \
            --with-ensurepip=install
...configure output...
```

Construire Python

Maintenant que l'on a configuré Python pour l'environnement, on doit réellement compiler les binaires et les préparer à fonctionner. On va le faire avec la commande **make**. Cependant, étant donné que Python est un programme asser volumineux et que le RPi n'est pas exactement le plus gros bourreau de travail au monde, la compilation prendra un peu de temps.

Donc, plutôt que de laisser un terminal ouvert tout le temps de la compilation de Python, on va utiliser le planificateur. On peut soumettre un travail qui le compilera et on peut simplement attendre la fin du travail. Pour cela, créer un script de soumission dans le dossier source Python :

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
#SBATCH --nodelist=node1

cd $SLURM_SUBMIT_DIR
make -j4
```

Ce script demandera 4 cœurs sur node1 et exécutera la commande **make** sur ces cœurs. Maintenant, il suffit de soumettre la tâche à partir du nœud de connexion :

```
$ cd /clusterfs/build/Python-3.7.3
$ sbatch sub_build_python.sh
Submitted batch job 1212
```

Maintenant, il faut juste attendre que le travail se termine. On peut visualiser sa progression à l'aide de la commande **squeue** et en consultant le fichier de sortie SLURM :

```
$ tail -f slurm-1212.out        # remplacer "1212" par l'ID du travail
```

Installer Python

Enfin, installer Python dans le répertoire `/clusterfs/usr`. Cela prendra également un certain temps, mais pas autant que la compilation. On peut utiliser le planificateur pour cette tâche. Créer un script de soumission dans le répertoire source :

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --nodelist=node1

cd $SLURM_SUBMIT_DIR
make install
```

Cependant, on ne veut pas que n'importe quel programme puisse modifier ou supprimer les fichiers d'installation de Python. Ainsi, comme avec n'importe quel programme normal, on va installer Python en tant que root afin qu'il ne puisse pas être modifié par les utilisateurs normaux. Pour ce faire, soumettre la tâche d'installation en tant qu'utilisateur root :

```
$ sudo su -
# cd /clusterfs/build/Python-3.7.3
# sbatch sub_install_python.sh
Submitted batch job 1213
```

Encore une fois, on peut surveiller l'état du travail. Une fois terminé, on aura un Python fonctionnel installé.

Tester Python.

On doit maintenant pouvoir utiliser l'installation Python à partir de n'importe quel nœud. Comme premier test de base, On peut exécuter une commande sur tous les nœuds :

```
$ srun --nodes=3 /clusterfs/usr/bin/python3 -c "print('Hello')"
Hello
Hello
```

```
Hello
```

On doit également avoir accès à pip :

```
$ srun --nodes=1 /clusterfs/usr/bin/pip3 --version
pip 19.0.3 from /clusterfs/usr/lib/python3.7/site-packages/pip (python 3.7)
```

La même installation Python devrait maintenant être accessible à partir de tous les nœuds. Ceci est utile car, lorsqu'on souhaite utiliser une bibliothèque pour un travail, on peut l'installer une fois sur cette installation et tous les nœuds peuvent l'utiliser. C'est plus propre à entretenir.

Un Hello-World Python MPI

Enfin, pour tester les nouvelles installations OpenMPI et Python, on va préparer un travail Python rapide qui utilise OpenMPI. Pour s'interfacer avec OpenMPI en Python, on va utiliser une bibliothèque appelée mpi4py.

Pour le démo, on va utiliser l'un des programmes de démonstration du référentiel mpi4py permettant de calculer la valeur de pi (le nombre) en parallèle.

Prérequis

Avant de pouvoir écrire le script, il faut installer les bibliothèques **mpi4py**, et **numpy**[1]. On peut installer ces bibliothèques via pip, en utilisant un travail par lots. Créer le fichier `/clusterfs/calc-pi/sub_install_pip.sh` :

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1/clusterfs/usr/bin/pip3 install numpy mpi4py
```

Ensuite, soumettre le travail, en tant que root car cela modifiera l'installation Python :

```
$ cd /clusterfs/calc-pi
$ sudo su
# sbatch sub_install_pip.sh
Submitted batch job 1214
```

Une foisle travail soit terminé on doit pouvoir utiliser les bibliothèques **mpi4py** et **numpy** :

```
$ srun bash
node1$ /clusterfs/usr/bin/python3
Python 3.7.3 (default, Mar 27 2019, 13:41:07)
[GCC 8.3.1 20190223 (Red Hat 8.3.1-2)] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> import numpy
>>> from mpi4py import MPI
```

Créer le programme Python.

Comme mentionné ci-dessus, on va utiliser l'un des programmes de démonstration fournis dans le référentiel **mpi4py**. Cependant, comme on l'exécutera via le planificateur, il faut le modifier pour ne nécessiter aucune entrée de l'utilisateur. Créer le fichier `/clusterfs/calc-pi/calculate.py` :

```
from mpi4py import MPI
from math   import pi as PI
from numpy  import array

def comp_pi(n, myrank=0, nprocs=1):
    h = 1.0 / n
    s = 0.0
    for i in range(myrank + 1, n + 1, nprocs):
        x = h * (i - 0.5)
        s += 4.0 / (1.0 + x**2)
    return s * h

def prn_pi(pi, PI):
    message = "pi is approximately %.16f, error is %.16f"
    print  (message % (pi, abs(pi - PI)))

comm = MPI.COMM_WORLD
nprocs = comm.Get_size()
myrank = comm.Get_rank()

n    = array(0, dtype=int)
pi   = array(0, dtype=float)
mypi = array(0, dtype=float)

if myrank == 0:
    _n = 20 # Enter the number of intervals
    n.fill(_n)
comm.Bcast([n, MPI.INT], root=0)
_mypi = comp_pi(n, myrank, nprocs)
mypi.fill(_mypi)
comm.Reduce([mypi, MPI.DOUBLE], [pi, MPI.DOUBLE],
            op=MPI.SUM, root=0)
if myrank == 0:
    prn_pi(pi, PI)
```

Ce programme divisera le travail de calcul de l'approximation de pi selon le nombre de processus qu'on lui fourni. Ensuite, il affichera la valeur calculée de pi, ainsi que l'erreur de la valeur stockée de pi.

Créer et soumettre le travail.

On peut exécuter le travail en utilisant le planificateur. On demandera un certain nombre de cœurs au cluster et SLURM préconfigura l'environnement MPI avec ces cœurs. Ensuite, on exécutera simplement le programme Python en utilisant OpenMPI. Créer le fichier de soumission `/clusterfs/calc-pi/sub_calc_pi.sh`:

```
#!/bin/bash
#SBATCH --ntasks=6

cd $SLURM_SUBMIT_DIR

mpiexec -n 6 /clusterfs/usr/bin/python3 calculate.py
```

L'indicateur **–ntasks** permet de demander un nombre spécifique de cœurs au total (l'indicateur **–ntasks-per-node** permet de demander un certain nombre de cœurs pour chaque nœud). Parce qu'on utilise MPI, on peut avoir des cœurs sur toutes les machines. Par conséquent, On peut simplement demander le nombre de cœurs que l'on veut. Dans ce cas, ons demande 6 cœurs.

Pour exécuter le programme réel, on utilise **mpiexec** en lui indiquant qu'il dispose de 6 cœurs et on utilise la version de Python patagée.

On peut ajuster le nombre de cœurs pour qu'il soit supérieur/inférieur en s'assurant simplement de modifier l'indicateur `mpiexec -n ##` pour qu'il corresponde.

Enfin, on peut exécuter le travail :

```
$ cd /clusterfs/calc-pi
$ sbatch sub_calc_pi.sh
Submitted batch job 1215
```

Le calcul ne devrait prendre que quelques secondes sur le cluster. Lorsque le travail est terminé (On peut le surveiller avec **squeue**), on doit voir une sortie dans le fichier `slurm-####.out` :

```
$ cd /clusterfs/calc-pi
$ cat slurm-1215.out
pi is approximately 3.1418009868930934, error is 0.0002083333033003
```

On peut modifier le programme pour calculer une valeur plus précise de pi en augmentant le nombre d'intervalles sur lesquels le calcul est exécuté. Pour ce faire, modifier le fichier `calculate.py` :

```
if myrank == 0:
    _n = 20        # changer ce nombre pour contrôler les intervalles
    n.fill(_n)
```

Par exemple, voici le calcul exécuté sur 500 intervalles :

```
pi is approximately 3.1415929869231265, error is 0.0000003333333334
```

1)

NumPy est un package qui contient de nombreuses structures et opérations utiles utilisées pour le calcul scientifique en Python

From:
http://www.ouarte.garden/ - **dwndoc**

Permanent link:
http://www.ouarte.garden/doku.php?id=prive:rpi-clusterhat

Last update: **2025/02/19 10:59**