Tutoriel Atelier MAIA MBB

Mardi 02 juillet 2019

Pages utiles

Page d'accueil de la plateforme MBB
https://mbb.univ-montp2.fr/MBB/
La documentation en ligne
https://gitlab.mbb.univ-montp2.fr/docs/doc_user/docs/_book/articles/index.html
La page de demande d'assistance
https://kimura.univ-montp2.fr/calcul/helpdesk_NewTicket.html

Prérequis

Création d'un compte sur le cluster MBB

Ouvrir un ticket de création de compte sur la page de demande d'assistance de la plateforme MBB
https://kimura.univ-montp2.fr/calcul/helpdesk_NewTicket.html
Pour vérifier que le compte est actif se connecter sur la page principale de la plateforme MBB avec le bouton Login en haut à droite de la page https://mbb.univ-montp2.fr/MBB/

Installation du client FileZilla

Cette étape n'est pas nécessaire pour les personnes familières avec le transfert de fichiers distants en ligne de commande (scp) ou tout autre logiciel sFTP de leur choix.
Télécharger le client FileZilla à cette adresse
https://filezilla-project.org/download.php?type=client
Installer le client sur votre machine.

Installation du client SSH Putty (utilisateurs Windows)

Le client Putty permet aux utilisateurs Windows de se connecter au cluster en invite de commande.
Télécharger le binaire (.exe) https://putty.org/
Il n'y a pas d'installation à faire, le binaire est directement utilisable en double-cliquant dessus.

Pour les utilisateurs Mac et Linux il suffira d'ouvrir un terminal pour se connecter au MBB.

Première connexion au cluster

Les identifiants

L'URL du noeud de login (appelé le master) cluster-mbb.mbb.univ-montp2.fr
Ou son adresse IP 162.38.181.18
Votre identifiant de connexion : communiqué par les administrateurs de la plateforme, en général de la forme première lettre du prénom nom (sans espace) par exemple Philippe Verley => pverley. A la suite du tutoriel nous y ferons référence avec le mot *login*.
Mot de passe : communiqué par les administrateurs de la plateforme ou réinitialisé par vos soins.

Mac/Linux

La connexion entre le le noeud de login du cluster MBB et la machine locale s'opère avec le protocle SSH. SSH est un protocole de communication sécurisé (crypté) entre deux machines distantes.
Ouvrir un terminal (si vous ne savez pas comment faire, demandez de l'aide à votre voisin ou votre moteur de recherche).

```
ssh -Y login@cluster-mbb.mbb.univ-montp2.fr
```

et saisir le mot de passe.
Remarque 1 : pour la première connexion la commande scp demandera confirmation pour ajouter l'adresse du cluster aux hôtes connus (et de confiance). Confirmer.
Remarque 2 : rien ne s'affiche à l'écran à la saisie du mot de passe, pas même des *, c'est normal.
Pour aller plus loin (facultatif pour le tutoriel, pratique dans le monde réel)

- Comment créer un alias SSH pour ne pas avoir à taper l'adresse complète du cluster à chaque fois ? Par exemple ssh mbb (mbb étant l'alias pour login@cluster-mbb.mbb.univ-montp2.fr)

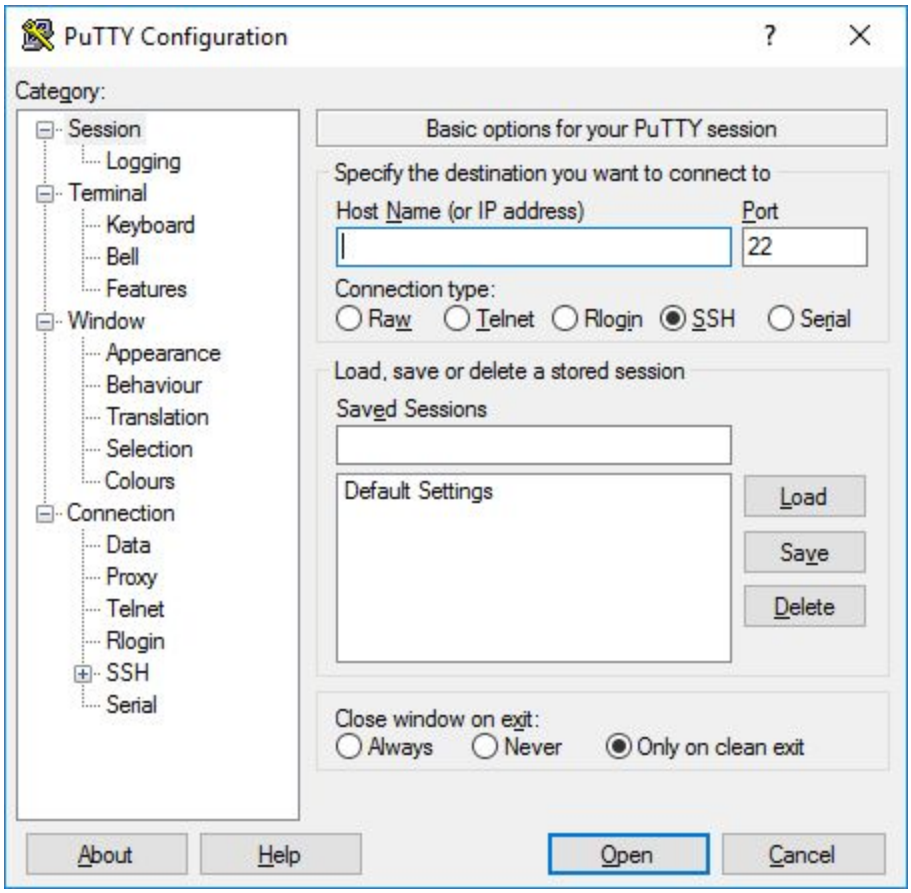
Éditer ou créer le fichier ~/.ssh/config et ajouter le contenu

```
Host mbb
  User login
  Hostname cluster-mbb.mbb.univ-montp2.fr
```

- Comment établir une connexion sans avoir à saisir le mot de passe à chaque fois ? (faire une recherche sur internet, non trivial)

Windows

Ouvrir le client Putty.exe et remplir les champs avec les identifiants de connexion. Laisser les valeurs par défaut si hésitation.
Enregistrer la session sous nom MBB pour ne pas avoir à ressaisir les identifiants lors d'une prochaine connexion et ouvrir la session pour faire apparaître le terminal de connexion.



Remarque 1 : pour la première connexion la commande scp demandera confirmation pour ajouter l'adresse du cluster aux hôtes connus (et de confiance). Confirmer.
Remarque 2 : rien ne s'affiche à l'écran à la saisie du mot de passe, pas même des *, c'est normal.

Premières commandes

Vous êtes connecté-e-s au noeud de login du cluster MBB, le point d'entrée, appelé encore le master. Aucun calcul ne seront lancé directement sur le master. Le master permet de gérer les fichiers présents sur son compte, compiler des codes sources et installer des packages R en local et gérer les calculs qui seront distribuées sur les noeuds de calcul à l'aide de

l’ordonnanceur de tâches. Le master ne possède pas d’interface graphique, on interagit via l’invite de commande et un langage de script.
Vérifier l'identité de la machine distante :

`hostname`

Vérifier votre identité :

`whoami`

Où suis-je sur la machine ?

`pwd` (print working directory)

Système d’exploitation du noeud de login :

`cat /etc/centos-release`

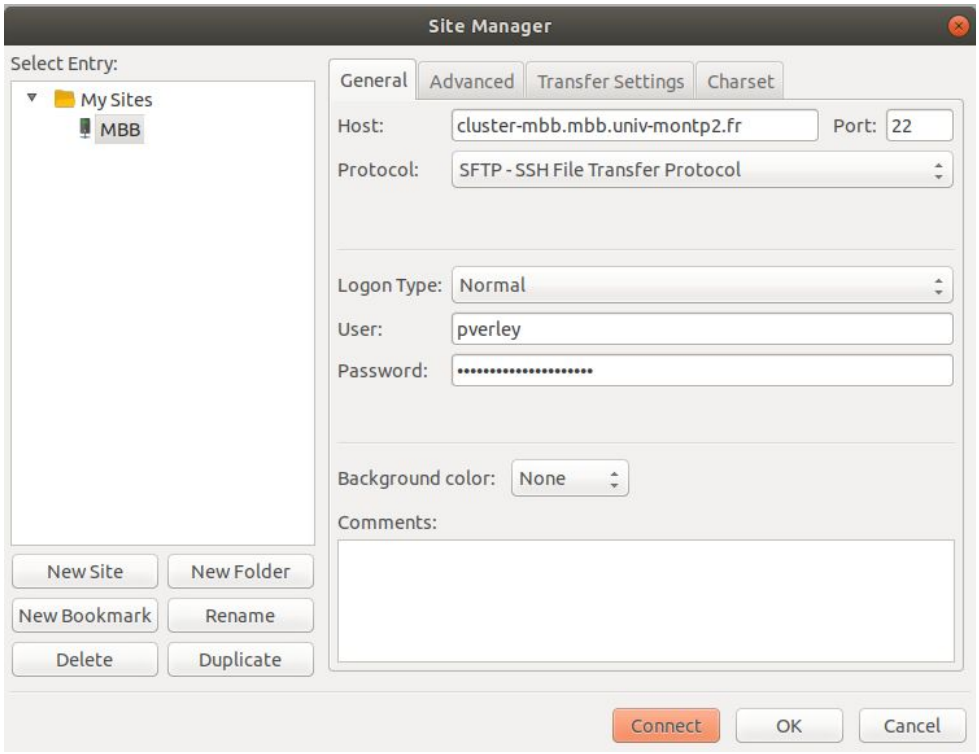
Premier transfert de fichier

Avec le client FileZilla

Ouvrir FileZilla puis Menu > File > Site Manager. Depuis la fenêtre de gestionnaire de site cliquer sur “New site”.
Nommer le site MBB

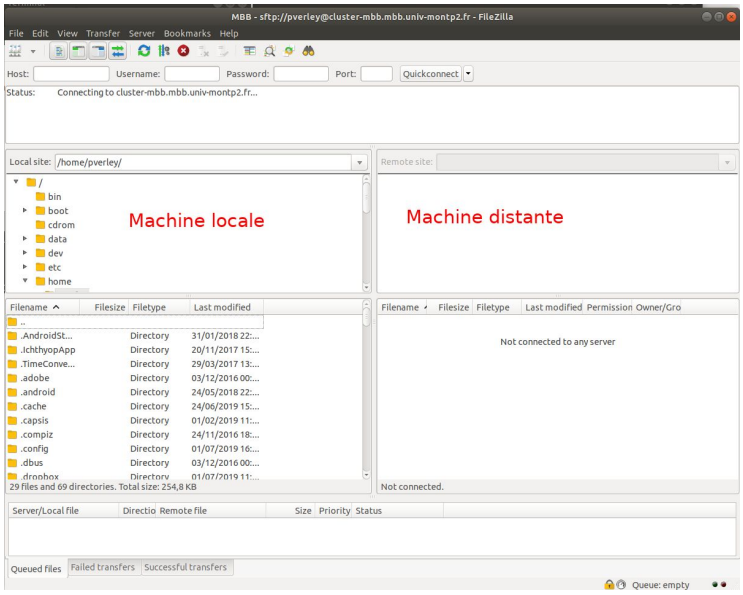
- Host : cluster-mbb.mbb.univ-montp2.fr
- Port : 22
- Protocol : SFTP
- Logon type : Normal
- User : votre login
- Password : votre mot de passe

Cliquer sur Connect puis OK.



Une fois connecté au cluster MBB la fenêtre principale de FileZilla est divisée en deux fenêtre principales :

- Gauche l'arborescence de fichiers sur votre machine locale.
- Droite : l'arborescence de fichiers sur la machine distante.



Il suffit ensuite de passer les fichiers de l'un vers l'autre. Pour de gros fichiers cela peut prendre du temps.

Commande scp (Mac/Linux)

Ouvrir un terminal.
Pour un fichier

touch tmp.txt (création d'un fichier vide à la racine de votre $HOME
scp temp.txt login@cluster-mbb.mbb.univ-montp2.fr:~ (noter le ~ qui est un raccourci pour indiquer à la commande scp de copier le fichier temp.txt à la racine de votre $HOME sur la machine distante)

Pour un dossier

mkdir tmp
mv tmp.txt tmp/
scp -r tmp login@cluster-mbb.mbb.univ-montp2.fr:~

Cas particulier des fichiers / dossiers volumineux

Votre répertoire personnel sur le cluster s'appelle votre $HOME. Le chemin d'accès est le suivant /home/login (login étant votre identifiant de connexion). Le répertoire personnel ne permet pas de stocker plus que quelques Go de données.
Si vous envisagez de travailler avec des données volumineuses, AMAP possèdent de l'espace disque supplémentaire sur le cluster (~10To) accessible à l'adresse /share/nas2-amap
Accéder à cet emplacement avec FileZilla
Créer à votre convenance un nouveau répertoire personnel (par exemple pverley) avec FileZilla à cet emplacement.

Se familiariser à l'invite de commande

Structure générale d'une commande bash

commande -o --option arg1 arg2

Commandes essentielles

Savoir où vous vous trouvez dans l'arborescence de fichier : pwd = print working directory

pwd

Lister les fichiers du répertoire courant: ls pour list

ls . (ou juste ls, . étant un raccourci pour le répertoire courant)

Lister les fichiers d'un répertoire en particulier

ls /share/nas2-amap

Retourner à la racine de répertoire personnel (/home/login/)

cd

Se déplacer dans un répertoire : cd pour change directory

`cd /absolute/path/directory`
`cd relative/path/directory`

Pour des explications plus poussées sur l'arborescence de fichier Linux
https://doc.ubuntu-fr.org/arborescence
Revenir un cran en arrière dans l'arborescence :
`cd ..`
Consulter l'aide sur une commande en particulier : man pour manual
`man ls`

Commandes utiles

Copier un fichier/dossier
`cp source destination`
`Cp -r dossier destination` (-r pour récursif)
Déplacer un fichier/dossier (ou le renommer)
`mv source destination`
`mv tmp.txt tmp2.txt`
Supprimer un fichier (attention pas de corbeille pour récupérer un fichier supprimé)
`rm source`
`rm -r dossier` (-r pour récursif)
Effacer l'écran
`clear`
Pour les utilisateurs Mac/Linux, éditer un fichier texte directement sur le cluster
`gedit fichier.txt &`

Pour aller plus loin

Sur la documentation en ligne du MBB vous trouverez une liste étoffée de commandes linux
https://gitlab.mbb.univ-montp2.fr/docs/doc_user/docs/_book/articles/linux-base.html
De façon générale en tapant dans un moteur de recherche "Comment faire ceci ou cela en invite de commande" vous trouverez plusieurs réponses.
Pour un toturiel complet sur le langage de script BASH
https://gitlab.mbb.univ-montp2.fr/docs/doc_user/formations/_book/form1/bash.html

Découvrir le cluster

Votre quota disque

Requêter le détail du quota et utilisation disque :
`get_myquota.sh`
Rappel : espace disque supplémentaire disponible /share/nas2-amap

Les queues de soumission

Lister les queues actives :

`qconf -sql`

Le détail d'une queue :

`qconf -sq cemeb.q`

Les noeuds de calcul

Lister l'appartenance des machines et leurs caractéristiques par queue :

`qhost -q`

Les environnements parallèles

Lister les environnements parallèles :

`qconf -spl`

Les environnements parallèles pour une queue en particulier :

`qconf -sq cemeb.q |grep -A1 pe_`

Le détail d'un environnement parallèle :

`qconf -sp multithread60 | grep allocation`

- "pe_slots", i.e., tous les slots sont alloués sur le même noeud
- "round robin", i.e. les slots sont distribués sur les noeuds un à la fois à tour de rôle.
- "fill up", i.e. les slots sont alloués sur un premier noeud jusqu'à son remplissage avant de passer aux suivants

Pour en savoir plus
https://gitlab.mbb.univ-montp2.fr/docs/doc_user/docs/_book/articles/cluster.html#comment-connaitre-les-environnements-paralleles-disponibles-sur-le-cluster

Exécution d'un script R en séquentiel

Le MBB propose une section d'aide dédiée à R
https://gitlab.mbb.univ-montp2.fr/docs/doc_user/docs/_book/articles/r.html

Choix de la version de R avec la commande module

Plusieurs versions de R sont installées sur le cluster. Pour lister les applications disponibles :

`module avail`

Pour charger la version R de votre choix

`module load R-3.2.0`

Pour lister les modules chargés localement

`module list`

Pour décharger un module

```
module unload R-3.2.0
```

Pour décharger tous les modules

```
module purge
```

Pour la suite du tutoriel nous utiliserons la version 3.2.0

Exécution séquentielle d'un script sans arguments

Préparation du script R et du script d'exécution en local

Créer un fichier mon_script_1.R

```
rm(list=ls())
squareFunc <- function(n) { return (n*n) }
squareFunc(c(1:10))
```

Créer un fichier mon_script_1.sh

```
#!/bin/bash
# options pour le job scheduler
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -l h_rt=00:01:00
#$ -N script_1
module purge
module load R-3.2.0
R --vanilla --slave < mon_script_1.R
```

Dans ce script les lignes qui commencent avec #$ correspondent aux options du job scheduler. A quelques détails près on trouvera toujours une structure similaire pour tous les jobs soumis sur le cluster.

- -S le langage de script utilisé est le bash
- -cwd le script s'exécute dans le répertoire courant (là où sera appelé la commande qsub)
- -j y regroupé les fichiers de sortie erreur et output
- -l h_rt le temps maximum que vous autoriser pour l'exécution de votre job (une petite valeur permet à l'ordonnanceur de caler un petit job entre des plus gros jobs)
- -N le nom du job

Téléverser les fichiers sur le cluster

Téléverser les fichiers mon_script_1.R et mon_script_1.sh sur le cluster dans un nouveau dossier atelier_maia/ avec FileZilla.

Exécuter le script

```
cd atelier_maia
qsub -q mem.q mon_script_1.sh
```

Les utilisateurs du groupe AMAP sont prioritaires ce matin sur la queue mem.q (pour minimiser notre temps d'attente, l'administrateur a créé une réservation). En temps normal la queue de soumission se fait sur cemeb.q ou cemeb20.q (noeuds avec beaucoup de coeurs)
Pour vérifier l'état d'avancement du script

```
qstat
```

```
job-ID  prior   name    user         state submit/start at     queue                          slots ja-task-ID
------------------------------------------------------------------------------------------------------------------------
 657351 0.00000 script_1 pverley      qw    07/01/2019 22:17:48                                    1
```

La variable "state" renseigne sur l'état du job. Qw pour queuing, R pour running. Dans ce cas précis le job est tellement court que vous ne verrez probablement même pas l'état R.
Quand le job est terminé vous verrez un nouveau fichier résultat dans le dossier atelier_maia script_1.o657** les chiffres étant ceux du job-ID. Pour afficher son contenu :

```
cat script_1.o657**
[1]   1   4   9  16  25  36  49  64  81 100
```

Pour renoncer à l'exécution d'un script placé en file d'attente

```
qdel job-ID
```

Exécution séquentielle d'un script avec argument

Créer un fichier mon_script_2.R

```
rm(list=ls())
squareFunc <- function(n) { return (n*n) }
args <- commandArgs(TRUE)
value=as.numeric(args[1])
squareFunc(value)
```

Créer un fichier mon_script_2.sh

```
#!/bin/bash
# options pour le job scheduler
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -l h_rt=00:01:00
#$ -N script_2
module purge
module load R-3.2.0
R --vanilla --slave --args 10 < mon_script_2.R
```

Téléverser les fichiers sur le cluster et exécuter le script 2 comme le script 1.

```
qsub -q cemeb.q (ou mem.q) mon_script_2.sh
```

Vérifier la bonne exécution du script et le résultat (100 attendu)

Exécution d'un script R en série

Garder le même script mon_script_2.R
Créer un fichier mon_script_3.sh

```
#!/bin/bash
# options pour le job scheduler
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -l h_rt=00:01:00
#$ -N script_3
#$ -t 1-10
module purge
module load R-3.2.0
R --vanilla --slave --args $SGE_TASK_ID < mon_script_2.R
```

Noter l'option -t 1-10 qui indique que nous voulons exécuter un "array job". Cela signifie que le script mon_script_3.sh sera exécuté 10 fois avec une variable d'état $SGE_TASK_ID qui prendra les valeurs de [1, 10]. Ici l'exemple est trivial mais on peut imaginer que dans le monde réel la variable $SGE_TASK_ID transmise en argument à R permette ensuite de piocher dans une liste de fichiers de configuration ou d'espèces, etc.
Noter en sortie les fichiers résultats script_3.o657***.[1-10] qui sont autant de réalisations indépendantes du script mon_script_3.sh

Exécution d'un script R en parallèle

En mode multithread

[attention cet exemple ne fonctionne pas sur le cluster… sans explication évidente. Pas grave toutefois puisque le mode distribué présenté à la section suivante est une généralisation de la parallélisation multithread]
Tous les CPUs sont sur le même noeud.
Créer un fichier mon_script_4.R

```
rm(list=ls())
squareFunc <- function(n) { return (n*n) }
args <- commandArgs(TRUE)
ncore=as.numeric(args[1])
library(parallel)
cl <- makeCluster(getOption("cl.cores", ncore))
```

```
clusterExport(cl, “squareFunc”)
res <- parSApply(cl, 1:100, squareFunc)
stopCluster(cl)
print(res)
```

Créer un fichier mon_script_4.sh

```
#!/bin/bash
# options pour le job scheduler
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -l h_rt=00:01:00
#$ -N script_4
#$ -pe multithread8 2
module purge
module load R-3.2.0
R --vanilla --slave --args 2 < mon_script_4.R
```

En mode distribué

Installation du package snow. Exécuter les commandes suivantes depuis le master :

```
cd
module purge
module load R-3.2.0
mkdir -p ~/R/x86_64-unknown-linux-gnu-library/3.2
wget https://cran.r-project.org/src/contrib/snow_0.4-3.tar.gz
R CMD INSTALL snow_0.4-3.tar.gz -l ~/R/x86_64-unknown-linux-gnu-library/3.2/
```

Les CPUS sont répartis sur différents noeuds en fonction des disponibilités.

Créer un fichier mon_script_5.R (

```
library(snow,lib.loc="~/R/x86_64-unknown-linux-gnu-library/3.2")
squareFunc <- function(n) { return (n*n) }
values <- 1:100
#lecture du fichier contenant la liste des machines
args <- commandArgs(TRUE)
peFile=args[1]
#construction de la liste des slots pour le "cluster"
liste_Nodes=read.table(peFile, sep=" ",header=F, stringsAsFactors=F)
node_Names=liste_Nodes[,1]
nb_slots=liste_Nodes[,2]
workers=rep(node_Names, nb_slots)
#We will run in parallel mode (socket) with
cl <- makeSOCKcluster(workers)
res <- parLapply(cl, values, squareFunc)
```

```
stopCluster(cl)
print(unlist(res))
```

Créer un fichier mon_script_5.sh

```
#!/bin/bash
# options pour le job scheduler
#$ -S /bin/bash
#$ -cwd
#$ -j y
#$ -l h_rt=00:02:00
#$ -N script_5
#$ -pe robin 4
module purge
module load R-3.2.0
R --vanilla --slave --args $PE_HOSTFILE < mon_script_5.R
```

Maintenant à vous de jouer avec vos scripts !