IUT Nancy-Charlemagne Université Nancy 2
Licence Professionnelle Asrall

Tuteur : Maître de Conférences Lucas Nussbaum

Système de fichiers distribué : comparaison de GlusterFS, MooseFS et Ceph avec déploiement sur la grille de calcul Grid'5000.

Jean-François Garçia, Florent Lévigne,
Maxime Douheret, Vincent Claudel.

Nancy, le 28 mars 2011

Table des matières

1 Introduction **3**
1.1 Contexte 3
1.2 Système de fichiers distribué 3
1.3 Le Grid'5000 4

2 NFS **6**
2.1 Présentation 6
2.2 Aspect technique 6
2.3 Mise en place 7

3 Fuse **8**

4 GlusterFS, véritable serveur d'archivage. **9**
4.1 Présentation 9
4.2 Mise en place 10

5 MooseFS **16**
5.1 Présentation 16
5.2 Aspect technique 17
5.3 Mise en place 17

6 Ceph **24**
6.1 Présentation 24
6.2 Aspect technique 25
6.2.1 Moniteur de cluster 25
6.2.2 Serveurs de métadonnées 25
6.2.3 Serveurs de données 25
6.3 Mise en place 26

7 Comparaison **28**
7.1 Test de performances 28
7.1.1 Cinq serveurs 29
7.1.2 Vingt serveurs 31
7.1.3 Cinquante serveurs 31
7.1.4 Analyse des résultats 33

8 Conclusion **34**

A Organisation du travail 35
A.1 Outils utilisés . . . 35
A.2 Répartition des taches . . . 36
A.2.1 Florent Lévigne . . . 36
A.2.2 Jean-François Garcia . . . 36
A.2.3 Maxime Douhéret . . . 36
A.2.4 Vincent Claudel . . . 36
A.3 Tutoriel d'utilisation du Grid'5000 . . . 36

B Scripts 38
B.1 GlusterFS . . . 38
B.2 MooseFs . . . 40
B.3 Ceph . . . 44
B.4 NFS . . . 47
B.5 Benchmark . . . 48
B.6 Meta scripts . . . 51

C Résultats relevés 54
C.1 NFS . . . 54
C.2 GlusterFS . . . 55
C.3 MooseFS . . . 56
C.4 Ceph . . . 57

D Sources 58

Partie 1

Introduction

1.1 Contexte

Étudiants en licence professionnelle ASRALL (Administration de Systèmes, Réseaux, et Applications à base de Logiciels libres), notre formation prévoie une période de deux mois à mis temps pour la réalisation d'un projet tuteuré.

Le projet que nous avons choisis consiste à comparer diverses solutions de systèmes de fichiers distribués.

1.2 Système de fichiers distribué

Un système de fichiers (file system en anglais) est une façon de stocker des informations et de les organiser dans des fichiers, sur des périphériques comme un disque dur, un CD-ROM, une clé USB, etc. Il existe de nombreux systèmes de fichiers (certains ayant des avantages sur d'autres), dont entre autre l'ext (Extented FS), le NTFS (New Technology FileSystem), ZFS (Zettabyte FS), FAT (File Allocation Table).

Un système de fichiers distribué est un système de fichiers permettant le partage de données à plusieurs clients au travers du réseau. Contrairement à un système de fichier local, le client n'a pas accès au système de stockage sous-jacent, et interagit avec le système de fichier via un protocole adéquat.

Un système de fichier distribué est donc utilisé par plusieurs machines en même temps (les machines peuvent ainsi avoir accès à des fichiers distants, l'espace de noms est mis en commun). Un tel système permet donc de partager des données entre plusieurs clients, et pour certains de répartir la charge entre plusieurs machines, et de gérer la sécurité des données (par réplication).

1.3 Le Grid'5000

Le Grid'5000 est une infrastructure distribuée dédié à la recherche sur des systèmes distribuées à grande échelle.

Des ingénieurs assurent le développement et le support de l'infrastructure jour après jour, venant pour la plupart de l'INRIA [1].

Le Grid'5000 est réparti sur onze sites, dont neuf en France.

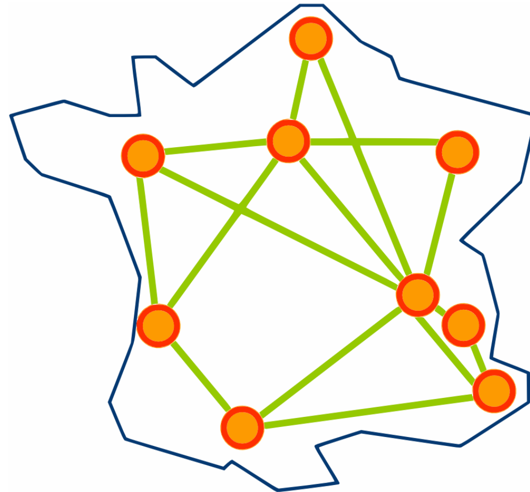

FIGURE 1.1 – Les sites français du Grid'5000

Chaque site possède un ou plusieurs clusters [2]. Par exemple, le site de Nancy possède deux clusters :

graphene : 144 nœuds contenants :
1 CPU Intel@2.53GHz
4 cores/CPU
16GB RAM
278GB DISK

griffon : 92 nœuds contenants :
2 CPUs Intel@2.5GHz
4 cores/CPU
16GB RAM
278GB DISK

1. Institut National de Recherche en Informatique et en Automatique
2. Grappe de serveurs

Pour travailler sur le Grid'5000, on se connecte en SSH [3] à une machine appelé le « frontend »présent sur chaque site. A partir de ce frontend, il est possible de déployer des environnements sur des machines du Grid (cf Partie A.3 page 36). Un certain nombre d'images (OS) sont disponibles. L'ensemble de nos tests ont été réalisés avec une Debian Squeeze (6), que nous avons construit à partir d'une Debian Lenny (5), la version 6 n'étant pas disponible. Un système de réservation permet de réserver à une date et heure voulue des nœuds, et si on le souhaite, d'exécuter un script.

FIGURE 1.2 – Un cluster de Nancy

3. Secure Shell : protocole de communication sécurisé

Partie 2

NFS

2.1 Présentation

NFS (Network File System, système de fichiers en réseau en français) est un système de fichiers développé par Sun Microsystem, permettant de partager des données par le réseau. NFS est aujourd'hui la méthode standard de partage de disques entre machines Unix. C'est une solution simple et pratique, quoique peu sécurisée.

2.2 Aspect technique

NFS utilise généralement le protocole non connecté UDP (en opposition à TCP qui est un protocole connecté). Toutefois, certains clients et serveurs NFS (depuis la version 3) peuvent aussi utiliser TCP, ce qui permet, entre autre, de fiabiliser les échanges.

On retrouve pour NFS différentes versions :
NFSv1 et v2 définies dans la RFC 1094 prévues pour fonctionner sur UDP.
NFSv3 définie dans la RFC 1813 prenant en charge le TCP.
NFSv4 définie dans la RFC 3010 puis réviser dans la RFC 3530. L'ensemble du protocole est repensé, et les codes sont réécrits. Cette version intègre une meilleur gestion de la sécurité, de la montée en charge, et un système de maintenance simplifié. NFSv4 supporte les protocoles de transports TCP (par défaut) et RDMA, ce qui augmente la fiabilité. Nous utiliserons donc cette version NFS dans l'ensemble de nos tests.

Au delà des évolutions entre les versions 3 et 4 du protocole, c'est la cohérence du système de nommage qui distingue la version 4 du système de fichiers réseau. Il s'agit de garantir qu'un objet (fichier ou répertoire) est représenté de la même manière sur un serveur et sur ses clients.

2.3 Mise en place

Le rôle du serveur NFS est de mettre à disposition sur le réseau une partie de son arborescence locale. On parle d'« exportation ».
Il existe plusieurs implémentations libres de serveur NFS. On se limite ici à l'utilisation du logiciel lié au noyau Linux.
Dans cette configuration nous traitons de l'installation d'un serveur NFS en version 4 dont le but est d'exporter le contenu d'un répertoire /tmp vers les clients. Utilisant une image de Debian Squeeze, On recherche dans la liste des paquets disponibles, ceux dont le nom débute par 'nfs'.

```
# aptitude search ?name"(^nfs)"
v   nfs-client          -
i   nfs-common          - NFS support files common to client and server
i   nfs-kernel-server - support for NFS kernel server
v   nfs-server          -
p   nfs4-acl-tools      - Commandline and GUI ACL utilities for the NFSv4 client
p   nfswatch            - Program to monitor NFS traffic for the console
```

Dans la liste ci-dessus, on identifie les paquets nfs-common et nfs-kernel-server qui correspondent bien aux fonctions souhaitées pour le client et le serveur NFS. En exploitant la documentation Nfsv4 configuration et l'exemple donné dans le fichier de configuration, on applique la configuration suivante dans le fichier /etc/exports :

```
        /tmp  *.nancy.grid5000.fr(rw,fsid=0,insecure,subtree_check)
```

Pour les besoins de ces travaux pratiques, les fonctions de sécurité Kerberos ne sont pas utilisées. On utilise l'appartenance au domaine nancy.grid5000.fr comme critère de contrôle d'accès ; ce qui correspond à un niveau de sécurité faible, puisque toutes les machines déployées sur la plateforme de Nancy peuvent accéder au répertoire.
rw : autorise les requêtes en lecture et écriture ;
fsid=0 : (fsid=root ou fsid=0) cette option est propre à NFSv4. Elle définie le point de partage racine. Ce qui permet de monter des partages en fonction d'une racine virtuelle.
insecure : l'option insecure permet l'accès aux clients dont l'implémentation NFS n'utilisent pas un port réservé.
subtree_check : permet de limiter l'accès exclusivement au répertoire partager.

Après la modification de ce fichier, il suffit de redémarrer le service, puis de monter les clients sur le répertoire partager à l'aide de la commande suivante :

```
        mount serveur:/ -t nfs4 /tmp/partage
```

Afin d'utiliser la version 4 de NFS, il est nécessaire de le spécifier dans le type lors du montage.
Depuis le serveur, la commande exportfs est chargée de la gestion de la liste des systèmes de fichiers partagés. L'exécution de cette commande sans argument affiche la liste des répertoires exportés. Ainsi on peut vérifier que les directives données dans le fichier /etc/exports sont effectivement appliquées. Sur le client, nous avons le programme appelé showmount associé à l'option -e.

Partie 3

Fuse

FUSE est un logiciel libre signifiant « Filesystem in Userspace ». Il s'agit d'un module, disponible pour les kernels 2.4 et 2.6, grâce auquel il est possible d'implémenter toutes les fonctionnalités d'un système de fichier dans un espace utilisateur. Ces fonctionnalités incluent :

- une API de librairie simple ;
- une installation simple (pas besoin de patcher ou recompiler le noyau) ;
- une implémentation sécurisée ;
- utilisable dans l'espace utilisateur.

Pour pouvoir monter un système de fichier, il faut être administrateur à moins que les informations de montage aient été renseignées sur le fichier /etc/fstab.
FUSE permet à un utilisateur de monter lui-même un système de fichier. Pour profiter de FUSE, il faut des programmes qui exploitent sa bibliothèque et ces programmes sont nombreux [1].

L'installation de FUSE est réalisé avec la commande suivante :

```
apt-get install fuse-utils libfuse2
```

Avant de pouvoir utiliser ce paquet, il faut charger le module « fuse » en mémoire :

```
modprobe fuse
```

Pour charger le module automatiquement à chaque démarrage de l'ordinateur, il faut ajouter « fuse » dans le fichier « /etc/modules ».

```
nano /etc/modules
```

1. http ://fuse.sourceforge.net

Partie 4

GlusterFS, véritable serveur d'archivage.

4.1 Présentation

GlusterFS est un système de fichiers distribués libre (GPLv3), à la fois très simple à mettre en œuvre et dont les capacités de stockage peuvent monter jusqu'à plusieurs petabytes (1,000 milliards octets).
GlusterFS est composé de deux éléments logiciels : un serveur et un client.
GlusterFS supporte plusieurs protocoles de communication tels que TCP/IP, InfiniBand RDMA.

Les serveurs de stockage gèrent la réplication des données stockées dans le volume distribué, permettant une reprise rapide du service en cas d'indisponibilité de l'un des nœuds.
Les clients reposent sur Fuse pour monter localement l'espace de stockage fusionné, donc facilement administrable.
Le nombre de serveur et le nombre de clients ne sont pas limité.

Le fonctionnement de GlusterFS est très modulaire et peut être configuré pour :

- fusionner l'espace de tous les nœuds,
- assurer une redondance des données,
- découper les gros fichiers en tronçons plus petits qui seront répartis sur différents serveurs de stockage (stripping),
- fonctionner malgré la perte d'un ou plusieurs nœuds de stockage qui seront automatiquement réintégrée dans le cluster et resynchronisés dès qu'ils seront de nouveau actifs ...

4.2 Mise en place

Installation de GlusterFS

Pour vous présenter la méthodologie d'installation des deux parties Serveur et Client, nous allons réaliser la plus simple architecture, à savoir deux serveurs qui abriterons le volume distribué et le montage de celle-ci par une machine cliente.
La distribution choisie est une Debian Squeeze pour l'ensemble de la maquette, et le choix des paquets glusterfs-server et glusterfs-client en version 3.1.2 officiellement stable.

Actuellement, l'ajout du dépôt Sid [1] est obligatoire pour pouvoir accès à cette version de Gluster.

Modification du fichier /etc/apt/sources.list pour l'ajout des dépôts de Debian Sid, suivi de la gestion des priorités /etc/apt/preferences.

```
Ajout des depots Sid
deb http://ftp.fr.debian.org/debian squeeze main contrib
deb http://security.debian.org/ squeeze/updates main contrib
deb http://ftp.fr.debian.org/debian sid main contrib

gestion des priorites
Package: *
Pin: release a=stable
Pin-Priority: 700
Package: *
Pin: release a=unstable
Pin-Priority: 600
```

Installation de GlusterFS Serveur Actualisez le contenu du catalogue de paquets, installez le module serveur sur les deux machines.

```
aptitude update;aptitude install glusterfs-server/sid
Mettre a jour la librairie libglusterfs0 a partir de sid :
aptitude install libglusterfs0/sid
```

Choix du paquet différent pour le client.

```
aptitude update;aptitude install glusterfs-client/sid
Mettre a jour la librairie libglusterfs0 a partir de sid :
aptitude install libglusterfs0/sid
```

Nous passons à la configuration du volume distribué. Nous créons un volume qui sera la concaténation des deux volumes offerts par nos deux serveurs.
Le montage du volume sera réalisé dans /sharespace. Création de ce répertoire :

```
mkdir -p /sharespace
```

La génération des fichiers de configuration s'effectuera à partir de notre premier serveur grâce à la commande glusterfs-volgen.
Voila les IP que nous allons utiliser pour une meilleur compréhension :

- IP serveurs : 172.16.65.91 et 172.16.65.92

1. Version de Debian « instalble » , en développement

– IP cliente : 172.16.65.90

```
glusterfs-volgen --name mystore 172.16.65.92:/sharespace 172.16.65.91:/sharespace
```

Retour de serveur :

```
Generating server volfiles.. for server '172.16.65.92'
Generating server volfiles.. for server '172.16.65.91'
Generating client volfiles.. for transport 'tcp'
```

Vous pouvez constater que la méthode de transport par défaut est le protocole TCP.

En cas d'infrastructure équipé de l'InfiniBand (bus à haut débit), nous aurions pu le définir à cette étape par le biais de l'option -t ib-verbs, sans oublier d'installer les librairies nécessaires. Nous aurions eu :

```
apt-get install libibverbs1 librdmacm1
glusterfs-volgen --name mystore -t ib-verbs 172.16.65.92:/sharespace 172.16.65.91:/
    sharespace
```

La commande glusterfs-volgen génère les fichiers de configuration pour les serveurs et le client. Chaque fichier porte l'IP de la machine concerné dans son nom, nous allons les exporter dans leur répertoire d'exploitation et les renommer pour qu'ils soient pris en compte.

```
cp 172.16.65.91-mystore-export.vol /etc/glusterfs/glusterfsd.vol
cp mystore-tcp.vol /etc/glusterfs/glusterfs.vol

pour le second serveur
scp 172.16.65.92-mystore-export.vol 172.16.65.92:/etc/glusterfs/glusterfsd.vol
scp mystore-tcp.vol 172.16.65.92:/etc/glusterfs/glusterfs.vol
```

Une copie du fichier de transport ou fichier de configuration client doit être stocké par au moins un serveur.
A ce moment, nous sommes prêt à mettre en marche les services de GlusterFS sur chaque serveur :

```
/etc/init.d/glusterfs-server start
```

Montage d'un poste client.

Petit aparté pour vous spécifier que le protocole en natif est fuse, mais que l'applicatif est capable de supporter d'autres protocoles, NFS, CIFS ou encore Dav.
Le paquet gluster étant installé, passons à la création du point de montage.

```
mkdir -p /mnt/monglusterfsvolume
```

Notre client va maintenant monter le volume réparti, il suffit d'employer la commande mount.

```
mount -t glusterfs 172.16.65.91:6996 /mnt/monglusterfsvolume/
```

Lors du montage, GlusterFS va initier une connexion vers le serveur spécifié afin de récupérer le fichier de configuration du client que nous avons stocké précédemment sous /etc/glusterfs/glusterfs.vol.

ATTENTION

Nous avons pu constater que cette étape ne fonctionnait pas toujours très bien. Cela a pour conséquence de générer des dysfonctionnements dues à des problèmes de droits. Nous aurons droit à la lecture et à la modification des fichiers présent dans le volume.
Pour résoudre cette mauvaise initialisation, copions le fichier de transport manuellement dans /etc/glusterfs/ et remontons le volume pour recharger le nouveau fichier de configuration avec la commande :

```
mount -t glusterfs /etc/glusterfs/glusterfs.vol /mnt/monglusterfsvolume/
```

Pour le monter automatiquement à chaque démarrage, il faut ajouter la ligne suivante au fichier /etc/fstab Pour un serveur :

```
172.16.65.91  /mnt/monglusterfsvolume  glusterfs  defaults,_netdev  0  0
```

Pour un client :

```
/etc/glusterfs/glusterfs.vol  /mnt/monglusterfsvolume  glusterfs  defaults,_netdev
    0  0
```

Vérifions le volume :

```
df -h /mnt/monglusterfsvolume
Filesystem             Size  Used  Avail  Use% Mounted on
/etc/glusterfs/glusterfs.vol
                       9.4G  831M  8.1G  10% /mnt/monglusterfsvolume
```

ou encore :

```
df -h | grep 172.16.65.91
172.16.65.91 5.6G 2.1G 3.3G 39% /mnt/monglusterfsvolume
```

A des fins de test, concevons un fichier de 10 Mo :

```
dd if=/dev/urandom of=test bs=10M count=1
1+0 enregistrements lus
1+0 enregistrements ecrits
10485760 octets (10 MB) copies, 1,45815 s, 7,2 MB/s
```

Copions le fichier test sur le FS distribué :

```
cp test /mnt/monglusterfsvolume
```

Examinons le contenu des répertoires /mnt/monglusterfsvolume et /sharespace des deux serveurs.

```
ls -lh /mnt/monglusterfsvolume\\
ls -lh /sharespace\\
```

On constate l'absence du fichier test sur l'un des nœuds, ce qui est totalement cohérent puisque la configuration par défaut de GlusterFS est la mise en œuvre de la concaténation des volumes physiques et sans duplication ou réplication des données.
Nous utilisons donc l'espace des deux volumes sur les serveurs, qui est vu comme un seul par les clients. Vous trouverez ci-dessous des informations complémentaires sur les possibilités des fichiers de configuration même si nous n'y toucherons pas pour notre travail.

Listing du fichier glusterfsd.vol

```
volume posix
     type storage/posix
     option directory /mount/glusterfs
end-volume
volume plocks
 type features/posix-locks
 subvolumes posix
end-volume
volume brick
 type performance/io-threads
 option thread-count 4
 option cache-size 64MB
 subvolumes plocks
end-volume
volume brick-ns
     type storage/posix
     option directory /mount/glusterfs-ns
end-volume
volume server
     type protocol/server
     option transport-type tcp/server
     option listen-port 6996
     option bind-address 127.0.0.1
     subvolumes brick brick-ns
     option auth.ip.brick.allow 127.0.0.1
end-volume
```

Le module “posix” pointe vers l’espace de stockage des données “/mount/glusterfs”.
Le module “plocks” hérite du module “posix” et y ajoute de verrous de fichiers.
Le module “brick” hérite du module “plocks” et y ajoute un cache en lecture.
Le module “brick-ns” pointe vers l’espace de stockage de l’annuaire de fichiers (”namespace”) “/mount/glusterfs-ns”.
Le module “server” hérite des modules “bricks” et “brick-ns” et les publie sur le port TCP 6996 et pour le PC “localhost” uniquement.
Nous aurions par exemple pu ajouter un cache en écriture et un cache en lecture/écriture sur “brick-ns”.

Il est toujours primordial de bien définir le niveau de log :

```
glusterfsd -f server1.vol -L DEBUG -l server1.log
glusterfsd -f server2.vol -L DEBUG -l server2.log
```

L’option « -f » pointe vers le fichier de configuration, « -L » indique le niveau de journalisation souhaité et « -l » donne le nom du fichier de journaux par instance de « GlusterFS ».

Listing du fichier /etc/glusterfs/client.vol.

```
### Ajoutez la caracteristique du client et rattachez au subvolume des servers
volume brick1
type protocol/client
option transport-type tcp/client
option remote-host 172.16.65.91
option remote-subvolume brick
end-volume

volume brick2
type protocol/client
option transport-type tcp/client
option remote-host 172.16.65.92
option remote-subvolume brick
end-volume

### Fichier d'index du serveur1
volume brick1-ns
type protocol/client
option transport-type tcp/client
option remote-host 172.16.65.91
option remote-subvolume brick-ns
end-volume

### Fichier d'index du serveur2
volume brick2-ns
type protocol/client
option transport-type tcp/client
option remote-host 172.16.65.92
option remote-subvolume brick-ns
end-volume

###Volume repique pour les donnees
volume afr1
type cluster/afr
subvolumes brick1 brick2
end-volume

###Volume repique pour les index
volume afr-ns
type cluster/afr
subvolumes brick1-ns brick2-ns
end-volume

###Unification des volumes afr (utilises quans il y a plus de deux serveurs)
volume unify
type cluster/unify
option scheduler rr # round robin
option namespace afr-ns
subvolumes afr1
end-volume
```

Un dialogue est établi entre le client et les serveurs par le port par défaut de glusterts : 6996. Les espaces de stockage nommés « brick » de chacun de ses serveurs seront fusionnés par le module « cluster/unify ».

L'amélioration des performances passe par l'écriture à tour de rôle vers les différents serveurs ; vous aurez reconnu l'utilisation de l'algorithme de répartition de charge round robin (rr).
Les données de l'annuaire de données quand à elles sont écrites en parallèle sur les espaces de stockage « brick-ns », directive du module « cluster/afr ».

Nous pourrions rajouter des directives à des fins de performances, voici un exemple :

```
volume iot
 type performance/io-threads
 option thread-count 2
 subvolumes bricks
end-volume

### Ajout de la directive writebehind pour la structure de donnees
volume writebehind
type performance/write-behind
option aggregate-size 131072 # unite en bytes
subvolumes iot
end-volume

### Ajout de la directive readhead pour la structure de donnees
volume readahead
 type performance/read-ahead
 option page-size 65536   # unit in bytes
 option page-count 16     # cache per file = (page-count x page-size)
 subvolumes writebehind
end-volume
```

Le tableau suivant répertorie les options de volume ainsi que sa description et la valeur par défaut (d'autres options existent) :

Option	Description	Valeur par défaut
performance.cache-refresh-timeout	Les données mises en cache pour un fichier sera conservé x secondes, après quoi une revalidation des données est effectuée.	1 sec
performance.cache-size	Taille du cache de lecture.	32 MB
performance.write-behind-window-size	taille du per-file écrire derriere le buffer	1 MB
network.frame-timeout	Le délai après lequel l'opération doit être déclaré comme mort, si le serveur ne répond pas à une opération particulière.	1800 (30 mn)
network.ping-timeout	La durée pendant laquelle le client attend de vérifier si le serveur est sensible.	42 secs
auth.allow	Les adresses IP ou nom d'hôte qui devraient être autorisés à accéder au volume	* (permis à tous)
auth.reject	Les adresses IP ou nom d'hôte qui devraient se voir refuser l'accès au volume.	NONE (aucun rejet)
diagnostics.brick-log-level	Changements du niveau de journalisation des serveurs.	NORMAL
diagnostics.client-log-level	Changements du niveau de journalisation des clients	NORMAL

Partie 5

MooseFS

5.1 Présentation

MooseFS (Moose File System) est un système de fichiers répartis à tolérance de panne, c'est à dire résistant aux pannes d'un ou plusieurs nœuds de stockage. Développé par Gemius SA. Le code préalablement propriétaire a été libéré et mis à disposition publiquement le 5 mai 2008 sous licence GPLv3. Disponible pour Linux, FreeBSD, OpenSolaris et MacOS X.

Les développeurs sont réactifs et suivent de très près la liste de diffusion et les remontés de bugs de leurs utilisateurs.

MooseFS permet de déployer assez facilement un espace de stockage réseau, répartit sur plusieurs serveurs. Les fichiers sont découpés en morceaux (appelés chunks) puis répartis sur les différents chunkservers (nœuds de stockage) : selon le paramétrage voulu pour ces fichiers, ou pour le répertoire dans lequel ils se trouvent, chaque chunk sera répliqué sur x nœuds (différents).

Cette répartition permet de gérer la disponibilité des données, lors des montées en charge ou lors d'incident technique sur un serveur. L'atout principal de MooseFS, au-delà du fait qu'il s'agisse d'un logiciel libre, est sa simplicité d'administration et de sa mise en œuvre.

D'autres qualités sont à son actif :
- Le respect de la norme Posix.
- L'utilisation de Fuse en espace client.
- La mise à disposition d'une poubelle automatique à durée de rétention modifiable à souhait.
- Une commande unique « mfssetgoal » pour régler le nombre de répliquas, sans redémarrer le service, par fichier ou par répertoire. Disponibilité des données ou tolérance aux pannes en cas de réplication positionné à plus de 1 pour les données rendra transparent la perte d'un chunkserver au détriment d'un coût en matière de place.
- Redimensionnement simple de l'espace de stockage, sans arrêt des serveurs, quelque soit la configuration, comprenant la ré-indexation et la répartition des données vers ces nouveaux espaces sans avoir à intervenir (cela peut prendre plusieurs heures).

– Le système est réparti : les fichiers supérieurs à 64Mo sont découpés en paquets de 64Mo et distribués sur les différents serveurs de données dit « chunkserver ».

5.2 Aspect technique

Les montages du système de fichiers par les clients se fait à l'aide de FUSE. MooseFS est constitué de trois types de serveurs :

1. Un serveur de métadonnées (MDS) : ce serveur gère la répartition des différents fichiers
2. Un serveur métajournal (Metalogger server) : ce serveur récupère régulièrement les métadonnées du MDS et les stocke en tant que sauvegarde.
3. Des serveurs Chunk (CSS) : ce sont ces serveurs qui stockent les données des utilisateurs.

Le point le plus important étant de bien dimensionner le serveur Master (qui stocke les métadonnées) afin de ne pas être limité par le suite.

MooseFS permet de partager des données sur plusieurs machines de manière rapide, fiable et sécurisée.

5.3 Mise en place

Le système utilisé est une Debian Squeeze. MooseFs ne faisant pas partie des dépôts de la distribution, nous avons installé MooseFS en compilant les sources de la dernière version disponible (1.6.20) sur le site officiel.
http ://www.moosefs.org/download.html

Méthodologie d'installation pour la mise en œuvre des différents serveurs ou services sur des machines dédiées :

Commençons par **l'installation d'un Master serveur**

Récupérons l'archive sur le site de MooseFs et décompressons la dans le répertoire /usr/src/,
tar zxvf mfs-1.6.20-2.tar.gz
Pour des questions d'isolement nous créons un utilisateur et un groupe pour faire tourner le futur service.

```
groupadd mfs;useradd mfs -s /sbin/nologin
```

Positionnons nous dans le répertoire de l'archive, cd /usr/src/mfs-1.6.20-2
Nous compilons avec la prise en compte de notre nouvel utilisateur et nous désactivons les services non nécessaire à notre master. Désactivation du service mfschunck (stockage de données) et mfsmount (montage côté client)

```
./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var/lib --with-default
    -user=mfs --with-default-group=mfs --disable-mfschunkserver --disable-mfsmount
```

```
make;make install
```

Nous recopions les fichiers exemples du répertoire /etc pour la configuration du service, au nombre de trois :

```
cd /etc
cp mfsmaster.cfg.dist mfsmaster.cfg
cp mfsmetalogger.cfg.dist mfsmetalogger.cfg
cp mfsexports.cfg.dist mfsexports.cfg
```

Si nous ne modifions pas ces fichiers, les directives par défaut seront alors appliquées. Pour information toutes les directives sont très bien documentées dans le page de man (exp : man mfsmaster.cfg).
Nous ne toucherons pas au fichier mfsmaster.cfg, celui-ci défini les principaux arrangements du serveur. Nous adaptons la configuration du fichier mfsexports.cfg qui permet de fixer la plage d'IP ou les ordinateurs qui pourrons monter le système de fichiers et de convenir des privilèges. La première ligne non commenté nous intéresse, nous remplaçons l'astérisque (*) en spécifions la plage IP et son masque que nous allons employer pour notre maquette. Cela aura pour effet de donner accès à toutes les machines de cette plage d'IP à la totalité de la structure en lecture et en écriture.
Exemple de la ligne modifiée :

```
192.168.2.0/24 / rw,alldirs,maproot=0
```

Les binaires de méta-données et les fichiers de changelog sont maintenus dans un dossier spécifié lors de la compilation /var/lib, mais lors de l'installation le fichier de metadata est initialisé vide. Nous allons changer cela :

```
cd /var/lib/mfs
cp metadata.mfs.empty metadata.mfs
```

Avant le démarrage du service nous devons penser à la résolution de nom du serveur. Nous renseignons l'adresse du serveur maître dans le fichier /etc/hosts. A titre d'exemple :

```
192.168.1.1 mfsmaster
```

A ce stade nous pouvons démarrer le master serveur, le service démarrera avec le compte mfs

```
usr/sbin/mfsmaster start
```

En cas de mise en production pensez à modifier votre configuration pour un démarrage automatique du service lors des lancements de votre serveur.

Pour conclure cette section, un cgi[1] est disponible pour suivre l'état de fonctionnement du système, disponible sous votre navigateur à l'adresse http ://IP_Master_Server :9425

```
/usr/sbin/mfscgiserv
```

Remarque : le contenu sera vide tant que les serveurs de données ne seront pas démarrés.

1. Common Gateway Interface : interface utilisée par les serveurs HTTP

Installation du Serveur de Backup Metalogger.

Le principe de ce serveur est de récupérer tous les événements du système de fichiers et peut permettre, si le maître est vraiment mal en point (perte de ses métadonnées, ou de son historique), de lui faire récupérer ses données pour que le volume reste cohérent.
Cette solution repose sur le protocole Carp « Common Address Redundancy Protocol », et permet d'admettre à un groupe d'hôtes sur la même partie du réseau (ici mfsmaster et mfsmetalogger) de partager une même adresse IP (un maître et un esclave). La méthodologie d'installation est similaire au serveur maître.
Voila les commandes à suivre :

```
groupadd mfs
useradd -g mfs mfs
cd /usr/src
tar -zxvf mfs-1.6.20-2.tar.gz
cd mfs-1.6.20-2
```

Désactivation du service mfschunck (stockage de données) et mfsmount (montage côté client)

```
./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var/lib --with-default
    -user=mfs --with-default-group=mfs --disable-mfschunkserver --disable-mfsmount
make;make install
cd /etc
cp mfsmetalogger.cfg.dist mfsmetalogger.cfg
```

Déclaration du serveur master auprès de notre fichier /etc/hosts : 192.168.1.1 mfsmaster
Le démarrage automatique n'est pas configuré. Voila la commande de lancement :

```
/usr/sbin/mfsmetalogger start
```

Installation d'un Serveur de Données Chunk servers.

La procédure d'installation toujours similaire :
Désactivation du service mfsmaster

```
groupadd mfs
useradd -g mfs mfs
cd /usr/src
tar -zxvf mfs-1.6.20-2.tar.gz
cd mfs-1.6.20-2
./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var/lib --with-default
    -user=mfs --with-default-group=mfs --disable-mfsmaster
make;make install
```

Préparation des fichiers de configurations :

```
cd /etc/
cp mfschunkserver.cfg.dist mfschunkserver.cfg
cp mfshdd.cfg.dist mfshdd.cfg
```

Les directives du fichier mfshdd.cfg indique les dossiers dans lesquels seront stockés les données.

Exemple de mise en œuvre pour deux emplacements :
Ajoutons les deux lignes suivantes dans le fichier mfshdd.cfg.
/mnt/mfschunks1
/mnt/mfschunks2
Positionnons les droits d'écritures nécessaires pour la création des dossiers de verrouillage : .lock :

```
mkdir /mnt/mfschunks1
mkdir /mnt/mfschunks2
chown -R mfs:mfs /mnt/mfschunks1
chown -R mfs:mfs /mnt/mfschunks2
```

Si nous ne souhaitons pas utiliser la totalité de l'espace libre de nos disques, nous pouvons utiliser une autre méthode permettant de contrôler le volume utilisé du disque.
Création d'un volume de deux Go en ext3.

```
dd if=/dev/zero of=disk1 bs=1024 count=1 seek=$((2*1024*1024-1))
dd if=/dev/zero of=disk2 bs=1024 count=1 seek=$((2*1024*1024-1))
mkfs -t ext3 disk1
mkfs -t ext3 disk2
```

Nous montons nos nouveaux disques, sans oublier les droits d'écriture.

```
mkdir /mnt/hd1
mkdir /mnt/hd2
mount -t ext3 -o loop disk1 hd1
mount -t ext3 -o loop disk2 hd2
chown -R mfs:mfs /mnt/hd1
chown -R mfs:mfs /mnt/hd2
```

Nous informons le fichier /etc/hosts de la présence du master serveur : 192.168.1.1 mfsmaster
Il est maintenant prêt à être démarré :

```
/usr/sbin/mfschunkserver start
```

Installation d'un Poste Client.

Afin de monter le système de fichiers basé sur Fuse, la version du paquet doit être égal ou supérieur à 2.7.2
http ://sourceforge.net/projects/fuse/
Si le paquet n'est pas disponible dans la bonne version dans les dépôts, on peut le compiler :

```
cd /usr/src
tar -zxvf fuse-2.8.5.tar.gz
cd fuse-2.8.5
./configure;make;make install
modprobe fuse
```

Afin d'installer le service mfsmount nous réalisons les étapes suivantes :

```
tar -zxvf mfs-1.6.20-2.tar.gz
cd mfs-1.6.20-2
./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var/lib --with-default
   -user=mfs --with-default-group=mfs --disable-mfsmaster --disable-mfschunkserver
make;make install
```

Nous annonçons le serveur maître au fichier /etc/hosts : 192.168.1.1 mfsmaster
Supposons que nous souhaitons monter le système dans le dossier /mnt/mfs sur notre machine cliente.

```
mkdir -p /mnt/mfs
chown -R mfs:mfs /mnt/mfs
/usr/bin/mfsmount /mnt/mfs -H mfsmaster
```

Nous pouvons vérifier l'espace occupé par les systèmes de fichiers avec la commande df, option h améliora la lisibilité.
exemple : df -h | grep mfs

```
/storage/mfschunks/mfschunks1
                            2.0G    69M 1.9G 4% /mnt/mfschunks1
/storage/mfschunks/mfschunks2
                            2.0G    69M 1.9G 4% /mnt/mfschunks2
mfs#mfsmaster:9421          3.2G      0 3.2G 0% /mnt/mfs
```

Commandes Usuelles

Montage de la Poubelle :

```
mkdir -p /mnt/mfsmeta
chown -R mfs:mfs /mnt/mfsmeta
mfsmount -m /mnt/mfsmeta
```

Le temps est indiqué en secondes (valeurs utiles : 1 heure égal à 3600 secondes, 24h = 86400 secondes, 1 semaine = 604800 secondes).
Si nous définissons la valeur à 0 cela signifie une suppression immédiate du dossier et sa récupération ne sera plus possible.
Positionnons la rétention désiré pour la corbeille mfssettrashtime (option r permet la récursivité sur les répertoires) et vérifions la rétention appliqué mfsgettrashtime :

```
mfsgettrashtime -r /mnt/mfs-test/test2
/mnt/mfs-test/test2:
files with trashtime 0 : 36
directories with trashtime 604800 : 1

mfssettrashtime -r 1209600 /mnt/mfs-test/test2
/mnt/mfs-test/test2:
inodes with trashtime changed: 37
inodes with trashtime not changed: 0
inodes with permission denied: 0

mfsgettrashtime -r /mnt/mfs-test/test2
```

```
/mnt/mfs-test/test2:
files with trashtime 1209600 : 36
directories with trashtime 1209600 : 1
```

Définissons le nombre de copies que l'on désir par répertoire avec le commande mfssetgoal –r :

```
 mfssetgoal -r 1 /mnt/mfs/folder1
/mnt/mfs/folder1:
 inodes with goal changed:        0
 inodes with goal not changed:    1
 inodes with permission denied:   0

Remarquer la valeur 2 qui suit l'option -r qui
permet de fixer le nombre de copie a 2

mfssetgoal -r 2 /mnt/mfs/folder2
/mnt/mfs/folder2:
 inodes with goal changed:        0
 inodes with goal not changed:    1
 inodes with permission denied:   0
```

Vérifions la redondance avec la fonction mfsgetgoal :

```
mfsgetgoal /mnt/mfs/folder1
/mnt/mfs/folder1: 1

mfssetgoal /mnt/mfs/folder2
/mnt/mfs/folder2: 2
```

Le nombre réel de copies d'un dossier peut être vérifié avec les commandes mfscheckfile et de mfsfileinfo :

```
mfscheckfile /mnt/mfs/folder1

/mnt/mfs/folder1:
3 copies: 1 chunks


mfsfileinfo /mnt/mfs/folder1

/mnt/mfs/folder1:
chunk 0: 00000000000520DF_00000001 / (id:336095 ver:1)
copy 1: 192.168.0.12:9622
copy 2: 192.168.0.52:9622
copy 3: 192.168.0.54:9622
```

Testons le positionnement :

```
cp /usr/src/mfs-1.6.20-2.tar.gz /mnt/mfs/folder1
cp /usr/src/mfs-1.6.20-2.tar.gz /mnt/mfs/folder2
```

La commande mfscheckfile est employée pour la vérification du nombre de copies :

```
mfscheckfile /mnt/mfs/folder1/mfs-1.6.20-2.tar.gz
/mnt/mfs/folder1/mfs-1.6.20-2.tar.gz:
1 copies: 1 chunks

mfscheckfile /mnt/mfs/folder2/mfs-1.6.20-2.tar.gz
/mnt/mfs/folder2/mfs-1.6.20-2.tar.gz:
2 copies: 1 chunks
```

Le résumé du contenu de l'arbre entier du système de fichiers peut être appelé avec la commande mfsdirinfo :

```
mfsdirinfo /mnt/mfs/folder1/:
inodes: 15
directories: 4
files: 8
chunks: 6
length: 270604
size: 620544
realsize: 1170432
```

Arrêt de MooseFS

```
umount /mnt/mfs)
/usr/sbin/mfschunkserver stop
/usr/sbin/mfsmetalogger stop
/usr/sbin/mfsmaster stop
```

Dans tous les systèmes de fichiers contemporains, les fichiers sont écrits par le biais d'un buffer (cache d'écriture). En conséquence, l'exécution de la requête d'écriture elle-même transfère seulement les données au buffer (cache), l'écriture réelle n'ayant donc pas lieu. En conséquence, la confirmation de la requête d'écriture ne signifie pas que les données ont été correctement écrites sur un disque.
Elle est seulement invoquer et l'aboutissement passe par la commande fsync qui provoque le déchargement de toutes les données gardées en cache pour obtenir l'écriture physique sur le disque. Cela est problématique en cas d'erreur avant la finalisation de cette phase, un retour d'erreur de la commande fsync peut avoir lieu, voir une perte des données.

Partie 6

Ceph

6.1 Présentation

Ceph est un système de fichiers distribué sous Licence LGPL, créé initialement par Sage Weill en 2007 pendant ses études à l'université de Californie à Santa Cruz. Le but principal de Ceph est d'être compatible POSIX et d'être complètement distribué sans un seul point de défaillance. Les données sont répliquées de façon transparente ce qui fait de Ceph un système tolérant aux pannes. Mais Ceph est encore en phase de développement et il est fortement conseillé de ne pas l'utiliser en production.

Tout comme avec MooseFS les données sont découpées, réparties sur les différents serveurs et répliquées sur x nœuds. Cette répartition permet à Ceph d'avoir une grande tolérance vis-à-vis de la perte de nœuds stockant les données. De plus Ceph permet de rajouter ou d'enlever des serveurs de données pendant le fonctionnement de Ceph sans être obligé de couper le service.

Ceph remplit deux lacunes importantes des systèmes de fichiers actuellement disponible :

- robustesse du stockage distribué open-source. Ceph fournit une variété de fonctionnalités clés, généralement manquante dans les systèmes de fichiers open source, tel que l'évolutivité transparente (capacité d'ajouter simplement un disque pour étendre le volume de stockage), la répartition de charge intelligente
- évolutivité en terme de charge de travail ou de capacité de stockage. Ceph a été créé pour supporter des charges de l'ordre de dizaines de milliers de clients accédant au même fichier ou au même dossier. La capacité de stockage va du gigaoctet au petaoctet [1] et au-delà.

1. 1 petaoctet = 1024 teraoctet

6.2 Aspect technique

La partie serveur de Ceph utilise trois types distinct de démons :
- moniteur de cluster, qui garde trace des nœuds actifs et défaillants
- serveurs de métadonnées, qui stockent les métadonnées des inoeuds et des répertoires
- serveurs de données, qui stockent les données.

6.2.1 Moniteur de cluster

Le moniteur gère l'administration centrale, la configuration et l'état du cluster. C'est à partir du moniteur que l'on modifie le cluster : ajout ou suppression de serveurs de données et métadonnées. Les clients ont accès au données gérées par Ceph par l'intermédiaire d'un moniteur. Il garde en mémoire une carte du cluster, ce qu'il lui permet de communiquer aux clients sur quels serveurs de métadonnées et de données se connecter pour modifier, ajouter ou supprimer des fichiers.

6.2.2 Serveurs de métadonnées

Le démon gérant les serveurs de métadonnées agit en tant que cache cohérent et distribué du système de fichiers. Il contient les données concernant l'arborescence des fichiers et leurs métadonnées (nom, taille, emplacement mémoire sur les serveur de données). Avoir plusieurs serveurs de métadonnées a un certain avantage car cela permet d'équilibrer la charge de travail entre les différents serveurs et donc de gérer plus de clients au même moment.

6.2.3 Serveurs de données

Les serveurs de données stockent les fichiers du cluster. Ils sont découpés en morceaux et répliqués plusieurs fois sur des serveurs différents. Cela permet une certaine tolérance vis-à-vis de la perte de serveurs. Si jamais un serveur tombe en panne, les données sont toujours accessible car elles sont présente sur un autre serveur de données.

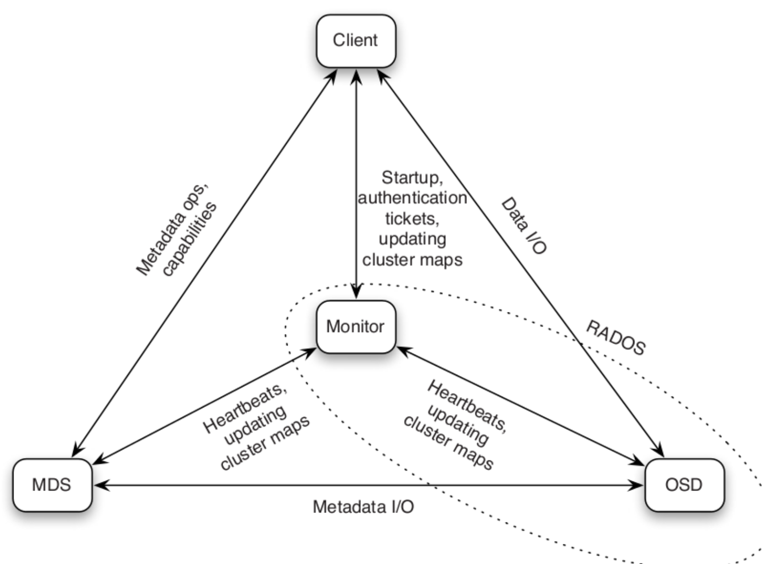

Figure 6.1 – Fonctionnement de Ceph

6.3 Mise en place

Avant toute configuration, il nous faut déterminiez les paquets qui nous seront utiles.Utilisant une image de Debian Squeeze, On recherche dans la liste des paquets disponibles, ceux dont le nom débute par 'ceph'.

```
# aptitude search ?name"(^ceph)
i   ceph                     - distributed storage and file system
i A ceph-client-tools        - utilities to mount a ceph filesystem with the kernel
    client
p   ceph-client-tools-dbg    - debugging symbols for ceph-client-tools
p   ceph-dbg                 - debugging symbols for ceph
i A ceph-fuse                - FUSE-based client for the Ceph distributed file system
p   ceph-fuse-dbg            - debugging symbols for ceph-fuse
```

Dans la liste ci-dessus, on identifie les paquets ceph, ceph-client-tools et ceph-fuse qui correspondent bien aux fonctions souhaitées pour le client et le serveur Ceph. En exploitant les documentations de Ceph (cf. Partie D page 58) on peut élaborer une configuration type.

Afin de mettre en place Ceph, il faut connaitre le nombre exacte de serveurs qui seront utilisés. Dans cet exemple de mise en place, nous prenons deux serveurs, node0 et node1. node0 sera le serveur principal, que l'on appel aussi moniteur. La totalité de la configuration, hormis le montage des clients, se fait à partir du moniteur. Pour commencer, il faut créer un fichier de configuration ceph.conf qui contiendra une grande partie des données nécessaires au bon fonctionnement des serveurs :

```
[global]
        pid file = /var/run/ceph/$name.pid
        debug ms = 1
        keyring = /etc/ceph/keyring.bin
[mon]
        mon data = /tmp/partage/mon$id
[mon0]
        host = node0
        mon addr = 10.0.0.10:6789
[mds]
        debug mds = 1
        keyring = /etc/ceph/keyring.$name
[mds0]
        host = node0
[mds1]
        host = node1
[osd]
        sudo = true
        osd data = /tmp/partage/osd$id
        keyring = /etc/ceph/keyring.$name
        debug osd = 1
        debug filstore = 1
        osd journal = /tmp/partage/osd$id/journal
        osd journal size = 1000
[osd0]
        host = node0
[osd1]
        host = node1
```

Ce fichier de configuration permet donc globalement d'indiquer la position des fichiers pid, keyring et le niveau de debug. Il défini les répertoires du moniteur dans lesquels les fichiers seront stockés. **[mon0]** contient le nom du serveur principal avec son adresse IP et le port utilisé. On spécifie ensuite la position du fichier keyring pour les serveurs de métadonnées **[mds]** et leur identification. **[osd]** contient la position de stockage des données, les journaux et le niveau de debug des serveurs de données spécifiés par la suite. Le nombre de **[osd]** et de **[mds]** varieront suivant le nombre de serveurs de données et de métadonnées voulus, en sachant qu'il est impératif de spécifier le rôle de chaque serveur.

Etant donné que la configuration ne se fait que sur le moniteur, il est alors nécessaire de copier la clé ssh[2] publique sur le second serveur (node1) afin de rendre possible la copie automatique des fichiers de configuration sans entrer de mot de passe :

```
ssh-copy-id -i ~/.ssh/id\_rsa.pub root@node1
```

Il est possible de générer par la suite un fichier keyring, qui n'est pas présent dans tout les exemples de configuration que l'on a pu rencontrer dans les documentations. Ce fichier keyring permet de déterminer les différents droits (lecture, écriture, exécution) des clients sur les différents serveurs quand la fonction d'authentification de CephFS est utilisé. Il se génère à l'aide des commandes suivantes :

```
cauthtool --create-keyring -n client.admin --gen-key keyring.bin
cauthtool -n client.admin --cap mds 'allow' --cap osd 'allow *' --cap mon 'allow
    rwx' keyring.bin
```

Pour permettre aux serveurs de fonctionner correctement, il est nécessaire de monter des droits spécifiques sur le répertoire de stockage, qui est ici /tmp :

```
mount -o remount,user\_xattr /tmp              # sur le moniteur
ssh node1 mount -o remount,user\_xattr /tmp    # sur le second serveur depuis le
    moniteur
```

Cette option « user_xattr » permet d'activer le support des attributs étendus sur les fichiers.
Puis l'on crée le système de fichier avant de démarrer le service :

```
mkcephfs -c /etc/ceph/ceph.conf --allhosts -v -k /etc/ceph/keyring.bin
```

Démarrage du service Ceph :

```
/etc/init.d/ceph -a start
```

L'option « -a », permet de démarrer le service sur tout les serveurs, ici node0 et node1.

Du côté des clients, il suffit de créer un répertoire dans lequel on monte le répertoire partagé par les serveurs à l'aide de fuse (cf. Partie 3 page 8) :

```
mkdir /ceph
cfuse -m 172.16.64.7 /ceph
```

Ce procédé est automatisé à l'aide du script DeploiementCeph.rb (cf. Partie B.3 page 44).

2. Secure Shell : protocole de communication sécurisé

Partie 7

Comparaison

7.1 Test de performances

Afin de ne pas avoir de différence de matériel lors nos tests, ceux-ci ont tous été réalisés sur un même cluster du Grid'5000 : Graphene.

Ce cluster est composé de 144 nœuds, avec pour caractéristiques :
- 1 CPU Intel de quatre cœurs cadencé à 2.53 GHz
- 16 Go de RAM
- 278 Go d'espace disque

Nous avons développé un benchmark mesurant les performances (débit) de quatre type d'opérations sur le système de fichier distribué :

Écriture de petits fichiers : écriture des sources du noyau Linux (décompressé).

Écriture de gros fichiers : écriture de trois fichiers de 1 Go[1]. Les trois fichiers sont différents.

Lecture de petits fichiers : lecture des fichiers du noyau Linux. Pour cela, nous avons compressé le dossier contenant le noyau (impliquant la lecture des fichiers), en redirigeant la sortie vers /dev/nul (afin que les performances du disque ne rentrent pas en jeux).

Lecture de gros fichiers : lecture des fichiers de 1 Go. Opération réalisé en faisant un « cat » des fichiers, et en redirigeant la sortie vers /dev/nul afin de ne pas « polluer » le terminal.

Les scripts de benchmark se découpent en deux parties :

Quatre petits scripts Bash décrivant les opérations à réaliser, décrite ci dessus.

Un script écrit en Ruby contrôlant l'ensemble du benchmark. Ce script se charge de distribuer les scripts Bash aux clients par SCP[2], et leur demande de les exécuter. Afin que l'ensemble des clients effectuent ces opérations simultanément, il a fallu mutlithreader le script (en utilisant la fonction fork[3]). Le déroulement du benchmark peut être suivis de manière précise sur le terminal, et les résultats sont consultable dans un fichier texte.

1. Fichier créé avec la commande : dd if=/dev/urandom of=/lieu/voulu bs=1G count=1
2. Secure copy : désigne un transfert sécurisé de fichiers entre deux ordinateurs utilisant le protocole de communication SSH (Wikipédia)
3. Cette fonction permet à un processus de donner naissance à un nouveau processus, par exemple en vue de réaliser un second traitement parallèlement au premier. (Wikipédia)

7.1.1 Cinq serveurs

Les tests réalisés avec NFS ont bien sur été effectués sur un seul serveur.

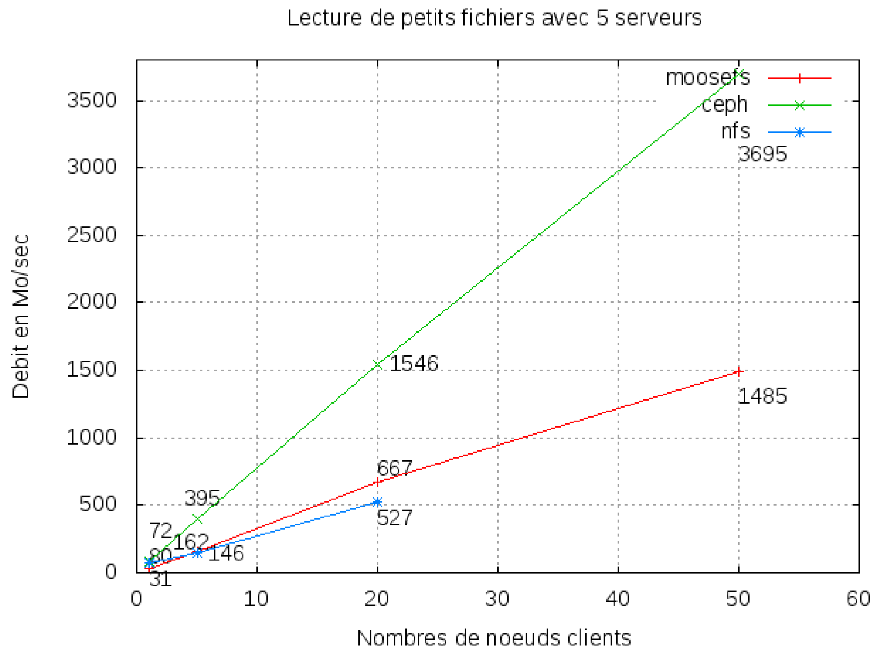

FIGURE 7.1 – Lecture de petits fichiers avec 5 serveurs

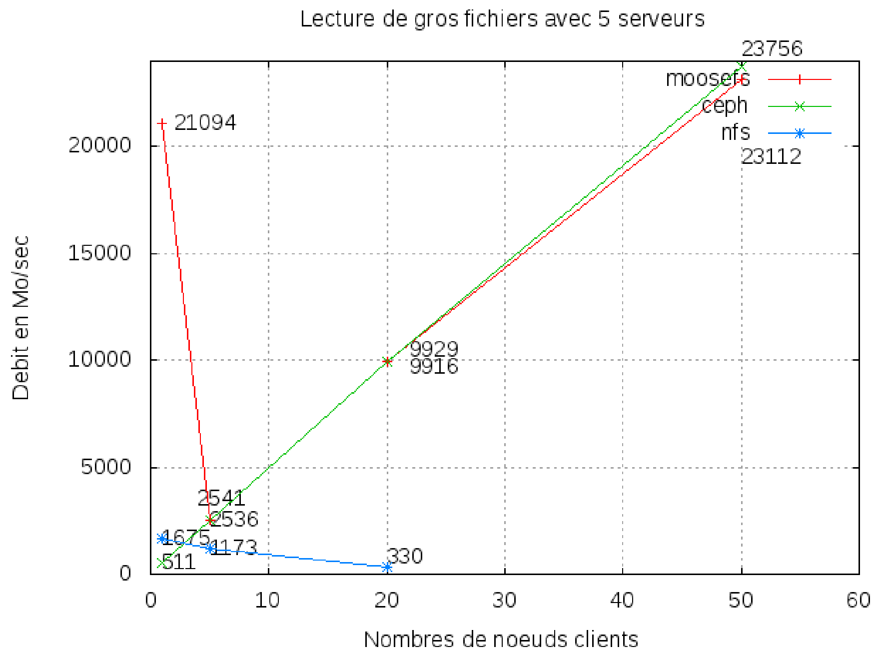

FIGURE 7.2 – Lecture de gros fichiers avec 5 serveurs

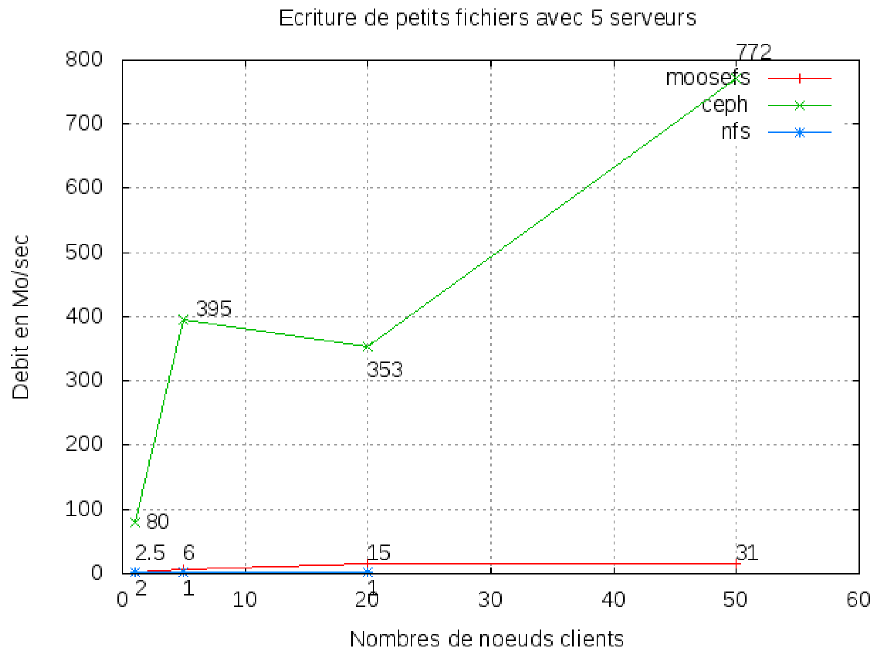

FIGURE 7.3 – Écriture de petits fichiers avec 5 serveurs

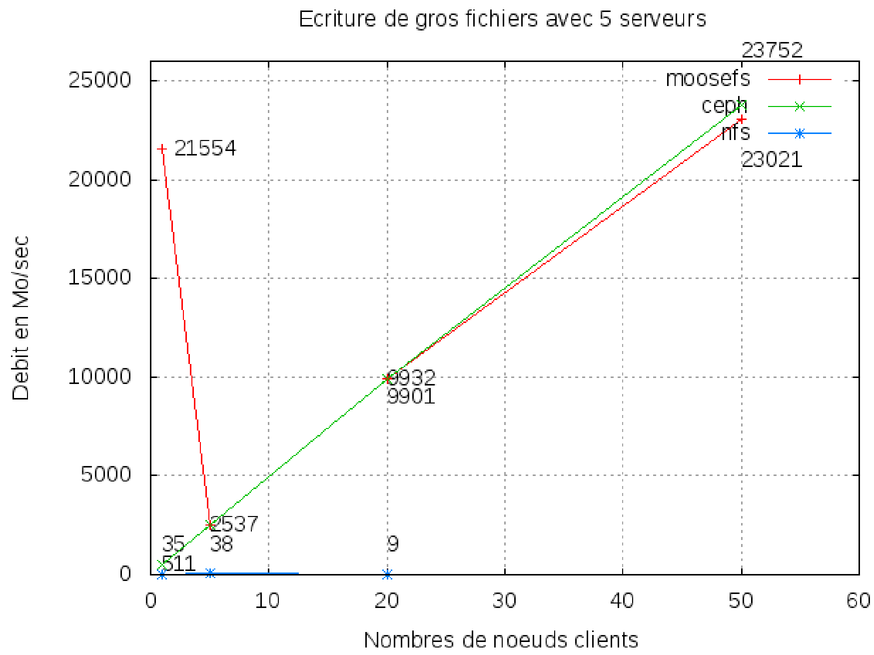

FIGURE 7.4 – Écriture de gros fichiers avec 5 serveurs

7.1.2 Vingt serveurs

7.1.3 Cinquante serveurs

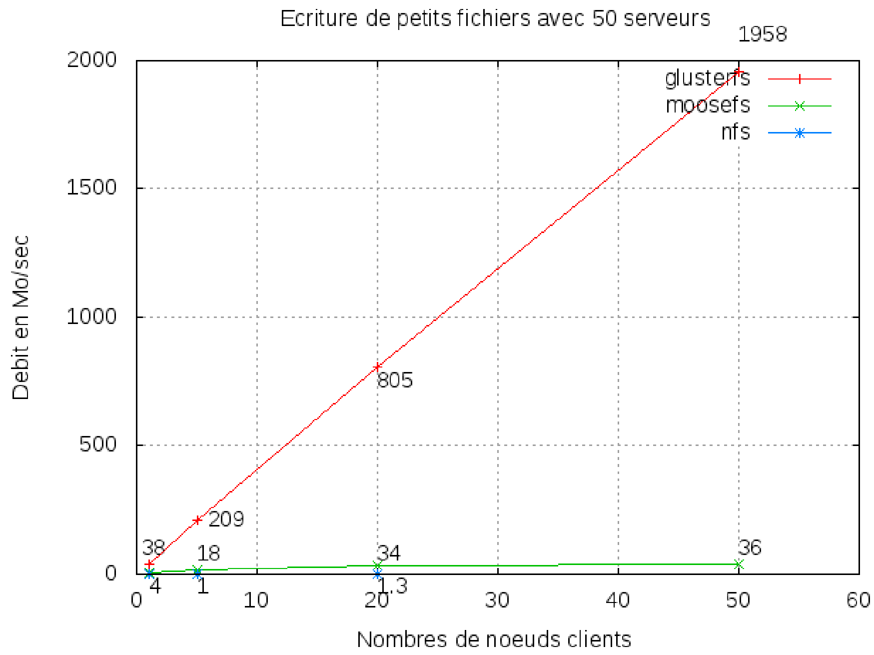

FIGURE 7.5 – Écriture de petits fichiers avec 50 serveurs

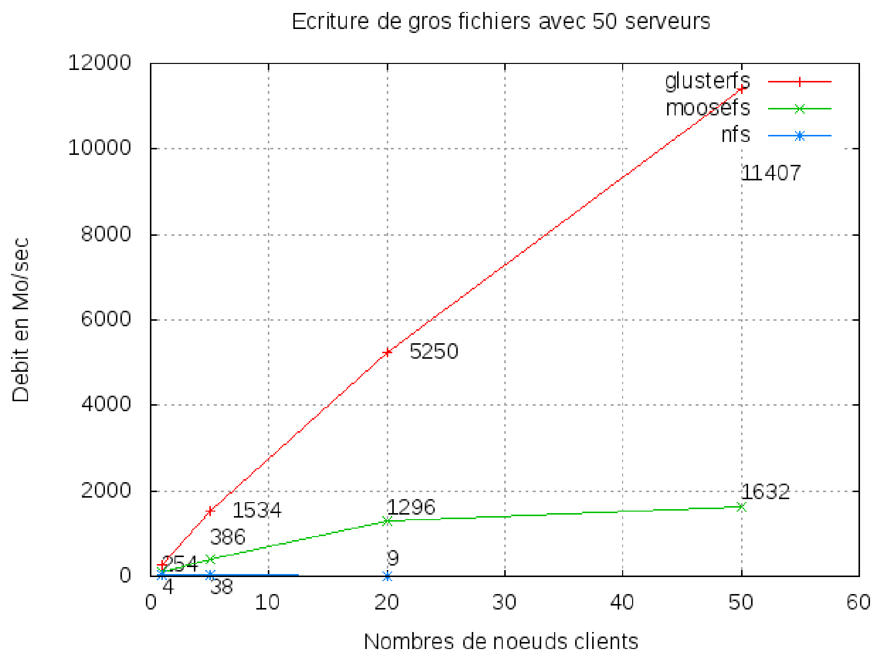

FIGURE 7.6 – Écriture de gros fichiers avec 50 serveurs

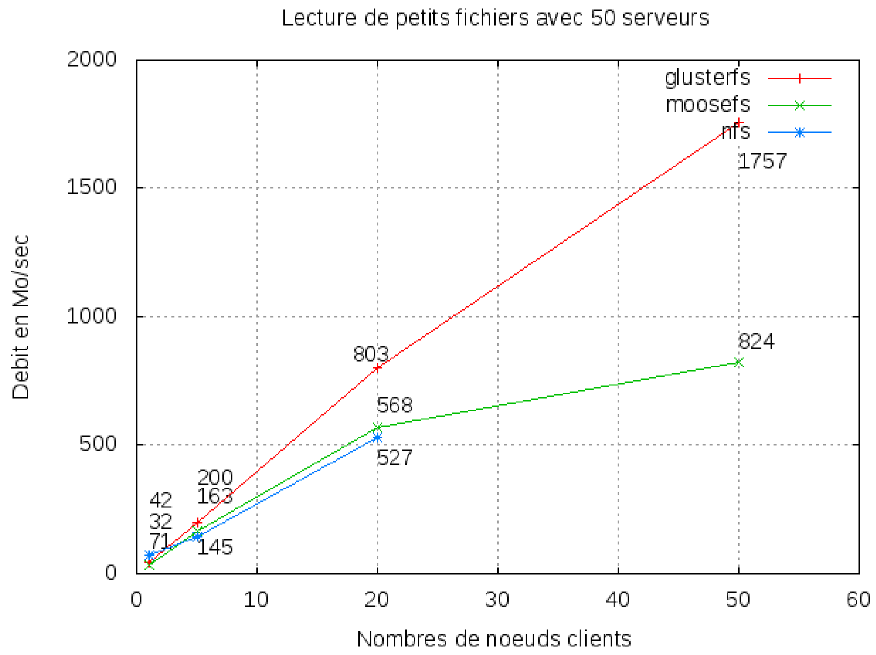

FIGURE 7.7 – Lecture de petits fichiers avec 50 serveurs

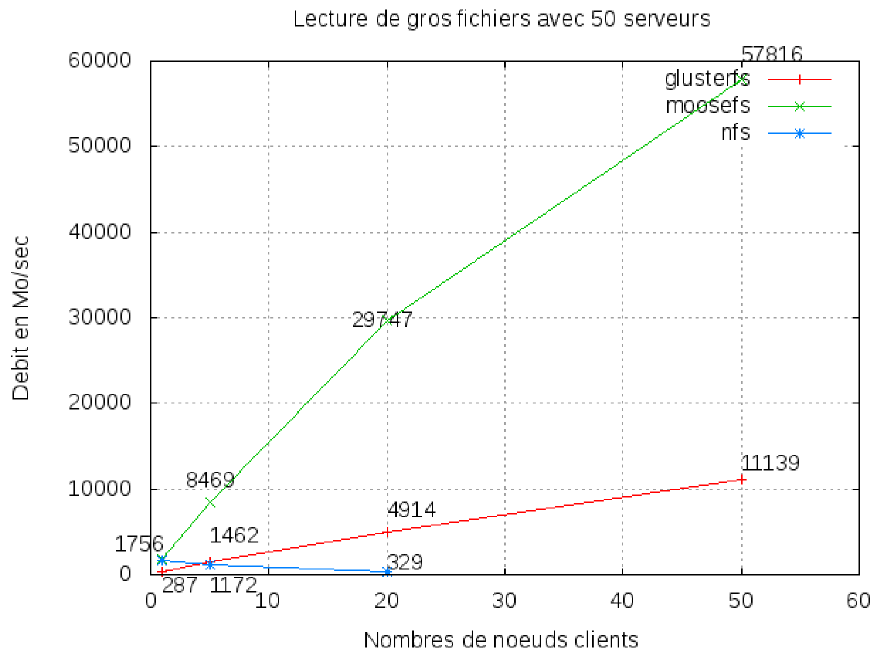

FIGURE 7.8 – Lecture de gros fichiers avec 50 serveurs

7.1.4 Analyse des résultats

Sur les tests que nous avons effectués, nous remarquons des débits très importants, parfois supérieurs au débit du réseau présent sur le cluster. Nous nous sommes donc questionné, afin de trouver pourquoi nous obtenons de tels résultats, et nous avons émis les hypothèses suivantes :

- Lors de la lecture de fichiers (petits ou gros), le client lit des fichiers dont il possède une copie sur sa machine. Le systèmes de fichier distribué est-il capable de le détecter et d'éviter d'aller lire le fichier sur un serveur distant ?
- Le cache présent sur les systèmes de fichiers est-il, à lui seul, responsable de tel résultats pour la lecture ?
- Lors des tests d'écriture, tous les clients écrivent les mêmes fichiers, le système de fichier distribué est-il capable de le détecter, et de limiter les transferts des clients vers les serveurs ?

Nous avons remarqués, après coup, certaines faiblesses de notre benchmark, qui devrait être plus complexe que nous n'avions pensé. Mais si l'on se base sur nos résultats, nous constatons que les performance de GlusterFS et de MooseFS sont de loin les meilleurs. MooseFS s'est sort très bien dans certains cas, avec un nombre plus réduit de serveurs. Ceph est plutôt destiné aux clusters de très grandes tailles (plusieurs milliers de serveurs).

Les performances seuls sont bien sur insuffisantes pour choisir un système de fichiers distribué, il y a plus de paramètres à prendre en comptes, en fonction de l'usage que l'on veut en faire. Préfère-t-on privilégier la sécurité ? (bonne réaction face à la perte d'un ou plusieurs serveurs) Les performances ? La facilité de mise en œuvre ?

Partie 8

Conclusion

Durant ce projet tuteuré nous avons eut la chance de travailler sur une grande structure : le Grid'5000. Après un premier temps de prise en main, nous avons pu commencer à mettre en place des solutions de systèmes de fichiers distribués, à créer des scripts automatisant leurs déploiements, et de réaliser un benchmark.

Nous avons rencontré un certains nombre de difficultés, (en plus de la mise en place des systèmes de fichiers distribués) propre au Grid'5000 :

- Nos test devaient être réalisés sur un même cluster, mais nous n'étions pas seuls à travailler dessus, et nous ne disposions pas toujours de toutes les machines dont nous avions besoins.
- Nous avons aussi rencontré des erreurs lors de déploiement de machines, indépendant de nous, qui ont « cassé » des benchmarks programmés à l'avance.
- Et nous avons aussi commis quelques erreurs humaines :
  - Suppression d'une réservation « importante » lors du nettoyage de l'espace de travail.
  - Oubli de donner les droits d'exécution à un script de pilotage, lors d'une réservation « importante ».

A cause de ces divers problèmes, nous n'avons pas obtenus tous les résultats que nous avions prévus d'avoir, ce qui ne permet pas une comparaison aussi approfondi que nous avions souhaité.

Le script de benchmark s'est aussi montré, après coup, imparfait, du fait que les clients lisent un fichier distant, dont ils ont une copie sur leurs machines, ce qui a pu fausser nos résultats.

Malgré ces problèmes, nous avons quand même pu effectuer une comparaison de différents systèmes de fichiers distribués. Ce travail nous a offert une bonne expérience pratique, d'un sujet technique, avant notre départ en stage.

Partie A

Organisation du travail

A.1 Outils utilisés

Afin d'organiser au mieux notre travail, nous avons mis en place un projet sur GitHub [1]. Nous avons utilisé le wiki [2] mis à notre disposition afin de mettre en commun nos connaissances et nos découvertes sur notre projet.

Nous avons aussi utilisé le dépôt Git [3] pour gérer le développement de nos scripts, et notre rapport de projet.

1. GitHub est un service web d'hébergement et de gestion de développement de logiciels (Wikipédia).
2. Un wiki est un site Web dont les pages comportent des hyperliens les unes vers les autres et sont modifiables par les visiteurs afin de permettre l'écriture et l'illustration collaboratives des documents numériques qu'il contient. (Wikipédia)
3. Git est un logiciel de gestion de versions, écrit par Linus Torvalds.

A.2 Répartition des taches

A.2.1 Florent Lévigne

- Étude sur la mise en place de GlusterFS
- Réalisation d'un script de déploiement de GlusterFS
- Étude sur la mise en place de MooseFS
- Réalisation d'un script de déploiement de MooseFS
- Réalisation d'un script de benchmark pour système de fichiers distribué
- Rédaction du rapport

A.2.2 Jean-François Garcia

- Étude sur la mise en place de GlusterFS
- Étude sur la mise en place de NFS
- Réalisation d'un script de déploiement de NFS
- Étude sur la mise en place de Ceph
- Réalisation d'un script de déploiement de Ceph
- Rédaction du rapport

A.2.3 Maxime Douhéret

- Étude sur la mise en place de GlusterFS
- Étude sur la mise en place de Ceph
- Aide à la réalisation des scripts de déploiement de Ceph et de benchmark
- Rédaction du rapport

A.2.4 Vincent Claudel

- Étude sur la mise en place de GlusterFS
- Étude sur la mise en place de MooseFS
- Rédaction du rapport
- Réalisation des graphiques en GnuPlot

A.3 Tutoriel d'utilisation du Grid'5000

Avant toute tentative de connexion, il est nécessaire de s'inscrire sur le site du Grid'5000 [4] par le biais de l'un des administrateurs avec l'intention d'effectuer de vraies recherches. Le Grid ayant été présenté, en partie 1.3 à la page 4, nous ne nous attarderons pas sur la description de celui-ci. Ce tutoriel à pour but de montrer le fonctionnement basique [5] de la plateforme.
L'accès au Grid ce fait à l'aide de SSH, et se fait en deux étapes : connexion au site, puis à son frontend.

```
ssh login@access.site.grid5000.fr
# depuis la machine :
ssh frontend
```

4. https ://www.grid5000.fr/mediawiki/index.php/Grid5000 :Home
5. Car il est possible de faire bien plus, le nombre de fonctionnalités disponibles est plutôt important !

Le but de cette plateforme, est la possibilité de réserver des machines faisant partie d'un cluster que l'on appelle des nœuds. Un certain nombre de commandes sont disponibles pour connaitre le statut de ces nœuds, leurs durée d'utilisation, ou encore de permettre leur réservation :

```
oarsub -I            # reservation d'une machine pour une heure par defaut
oarsub -I -l nodes=n # reservation de n machines pour une heure
oarsub -I -t deploy -l nodes=1,walltime=n # reservation de n machines pour n
    heures pour un d ploiement
```

La commande « oarsub » nous renvoie un numéro de réservation appelé « JOB_ID » et le nom des nœuds réservés est stockés dans la variable « OAR_FILE_NODES ». Dans le cadre du projet, il a fallu réserver des machines à l'avance dans le but d'effectuer des mesures de façon automatique durant la nuit pour éviter de monopoliser la totalité des machines d'un sites en journée. Heureusement, la commande oarsub peut effectuer cette tâche en lui spécifiant le script à exécuter avec un passage de paramètre :

```
oarsub -r '2010-03-24 19:30:00' -l nodes=55,walltime=6:30:00 "~/Ceph/
    DeploiementCeph.rb 5"
```

Il est possible à l'aide de différentes commandes, de connaitre des informations parfois vitales sur son environnement de travail :

```
# Sur l'environnement :
env
# Sur les noeud (les coeurs) (necessite d'avoir fait une reservation):
cat $OAR_NODEFILE
# Sur un job particulier (sur le frontend):
oarstat -f -j OAR\_JOB\_ID
# Sur le statut d'un job :
oarstat -s -j OAR\_JOB\_ID
# Sur les reservations d'un utilisateur :
oarstat -u login
```

Suite à la réservation d'un nœud, il est possible (et même vivement conseillé) de déployer une image (OS) sur celui-ci, dans le but d'effectuer des recherches. Cette image peut être fournie par le Grid'5000 ou être une image personnalisé. Ici, ce sera une image personnalisé.

```
kadeploy3 -f $OAR\_NODEFILE -a mylenny-x64-base.env -k
```

L'option « -f » permet de spécifier les nœuds sur lesquels on veut déployer l'image.
L'option « -a » spécifie l'image à utiliser.
et l'option « -k » envoie la clef plublic SSH à la machine déployée.
La commande « kadeploy3 » renvoie le statut du déployement en cours, puis si celui-ci c'est déroulé correctement, le nom des machines déployées. A la suite de se déploiement, il est possible de se connecter au nœud via la commande SSH :

```
ssh user@node.site.grid5000.fr
```

Dans le cas où les recherches sont terminées avant la fin de la réservation, il est possible de supprimer le JOB réservé à partir du frontend :

```
oardel $OAR_JOB_ID
```

Attention ! Dans ce cas, c'est la totalité de la réservation qui est supprimée, donc les nœuds déployés sur ce JOB.

Partie B

Scripts

B.1 GlusterFS

Fichier deploimentGluster.rb :

```
#!/usr/bin/ruby -w
# encoding: utf-8

# reservation des noeuds (a lancer manuellement)
# oarsub -I -t deploy -l nodes=8,walltime=2
# oarsub -I -t deploy -l nodes=8,walltime=2 -p "cluster='graphene'"

if ARGV[0] == nil
  puts "doit prendre en parametre le nombre de serveurs"
  exit(1)
end


# doit concorder avec la commande oarsub
numberOfServers = "#{ARGV[0]}".to_i

infiniband = 1 # 1 : active, 0 : non active (ne change rien pour l'instant)

# creation d'un fichier contenant la liste des noeuds reserves
`touch listOfNodes`
File.open("listOfNodes", 'w') do |file|
  file << `cat $OAR_FILE_NODES | sort -u`
end

# creation de deux fichiers contenant la liste des serveurs, et des clients
`touch listOfClients listOfServers`
serverWrited = 0
File.open("listOfNodes", 'r') do |node|
  File.open("listOfServers", 'w') do |server|
    File.open("listOfClients", 'w') do |client|
      while line = node.gets
        if serverWrited < numberOfServers
          server << line
          serverWrited += 1
        else
          client << line
        end
      end
    end
```

```
40     end
41   end
42
43   # deploiement des machines
44   puts "Machines en cours de deploiement..."
45   `kadeploy3 -k -e squeeze-collective -u flevigne -f listOfNodes` # image collective
46
47
48   # Creation d'un repertoire dans /tmp/sharedspace sur les serveurs
49   File.open("listOfServers", 'r') do |file|
50     while line = file.gets
51       machine = line.split.join("\n")
52       `ssh root@#{machine} mkdir /tmp/sharedspace`
53     end
54   end
55
56   # Creation d'un repertoire dans /media/glusterfs sur les clients
57   File.open("listOfClients", 'r') do |file|
58     while line = file.gets
59       machine = line.split.join("\n")
60       `ssh root@#{machine} mkdir /media/glusterfs`
61     end
62   end
63
64   masterServer = `head -n 1 listOfServers`.split.join("\n")
65
66   # generation des fichiers de conf, et envoie des fichiers de conf aux machines (
       serveurs et clients)
67   puts "Configuration des serveurs et des clients..."
68   `scp listOfServers root@#{masterServer}:`
69   `scp listOfClients root@#{masterServer}:`
70   `scp glusterfs-volgen.rb root@#{masterServer}:`
71   `ssh root@#{masterServer} ./glusterfs-volgen.rb`
72
73   # demarrage des serveurs
74   puts "Demarrage des serveurs..."
75   File.open("listOfServers", 'r') do |file|
76     while line = file.gets
77       machine = line.split.join("\n")
78       `ssh root@#{machine} /etc/init.d/glusterfs-server start`
79     end
80   end
81
82   # montage du repertoire par les clients
83   puts "Montage du repertoire par les clients..."
84   File.open("listOfClients", 'r') do |file|
85     while line = file.gets
86       machine = line.split.join("\n")
87       `ssh root@#{machine} mount -t glusterfs /etc/glusterfs/glusterfs.vol /media/
           glusterfs`
88     end
89   end
90
91   # resume des machines
92   puts "GlusterFS operationnel"
93   puts "\nMachines clients :"
```

```
94  puts `cat listOfClients`
95
96  puts "\nMachines serveurs :"
97  puts `cat listOfServers`
98
99  puts "\nServeur maitre : #{masterServer}"
100
101 # nettoyage
102 #`rm listOfNodes listOfClients listOfServers`
```

B.2 MooseFs

Fichier deploiementMoose.rb :

```
1  #!/usr/bin/ruby -w
2
3  # reservation des noeuds (a lancer manuellement)
4  # oarsub -I -t deploy -l nodes=9,walltime=2
5  # oarsub -I -t deploy -l nodes=9,walltime=2 -p "cluster='graphene'"
6
7  if ARGV[0] == nil
8    puts "doit prendre en parametre le nombre de serveurs (3 min)"
9    exit(1)
10 end
11
12 # doit concorder avec la commande oarsub
13 numberOfClients = 5 # inutile : prend les machines restantes en clients
14 numberOfServers = "#{ARGV[0]}".to_i # 3 serveurs minimum
15
16
17 # MooseFS et infinibande ?
18
19 # creation d'un fichier contenant la liste des noeuds reserves
20 `touch listOfNodes`
21 `cat $OAR_File_NODES`
22 File.open("listOfNodes", 'w') do |file|
23   file << `cat $OAR_FILE_NODES | sort -u`
24 end
25
26 # creation de deux fichiers contenant la liste des serveurs, et des clients
27 `touch listOfClients listOfServers`
28 serverWrited = 0
29 File.open("listOfNodes", 'r') do |node|
30   File.open("listOfServers", 'w') do |server|
31     File.open("listOfClients", 'w') do |client|
32       while line = node.gets
33         if serverWrited < numberOfServers
34           server << line
35           serverWrited += 1
36         else
37           client << line
38         end
39       end
40     end
41   end
```

```
42 end
43
44 # deploiement des machines
45 puts "Machines en cours de deploiement..."
46 `kadeploy3 -e squeeze-collective -u flevigne -f listOfNodes` # image collective
47
48 masterServer = `head -1 listOfServers`.strip # 1ere ligne du fichier
49 `sed -i 1d listOfServers` # supression de la 1ere ligne
50 metaloggerServer = `head -1 listOfServers`.strip
51 `sed -i 1d listOfServers`
52
53 masterServerIp = `ssh root@#{masterServer} hostname -i`.strip
54
55 # configuration du serveur maitre
56 puts "\nConfiguration du serveur maitre..."
57 `scp masterServer.sh root@#{masterServer}:/root`
58 `ssh root@#{masterServer} ./masterServer.sh`
59
60 # configuration du serveur de metadonnees
61 puts "\nConfiguration du serveur de metadonnees..."
62 `scp metaloggerServer.sh root@#{metaloggerServer}:/root`
63 `ssh root@#{metaloggerServer} ./metaloggerServer.sh #{masterServerIp}`
64
65 # configuration des chunks
66 puts "\nConfigurations des serveurs chunk..."
67 numberOfChunk = open("listOfServers").read.count("\n").to_i # numberOfChunk inutile
       ?
68
69 puts "Nombre de Chunk : #{numberOfChunk}"
70
71 # configuration des chunks
72 chunkConfiguration = 1
73 File.open("listOfServers", 'r') do |file|
74   while line = file.gets
75     machine = line.strip
76     `scp chunkServer.rb root@#{machine}:/root`
77     #`ssh root@#{machine} ./chunkServer.sh #{masterServerIp} #{numberOfChunk}`
78     `ssh root@#{machine} ./chunkServer.rb #{masterServerIp} #{numberOfChunk} #{
         chunkConfiguration}`
79     chunkConfiguration += 1
80   end
81 end
82
83 # demarrage des chunks
84 puts "\nDemarrage des serveurs chunks"
85 File.open("listOfServers", 'r') do |file|
86   while line = file.gets
87     machine = line.strip
88     `ssh root@#{machine} /usr/sbin/mfschunkserver start`
89   end
90 end
91
92 puts "\nConfigurations des clients..."
93 # configuration des clients
94 File.open("listOfClients", 'r') do |file|
95   while line = file.gets
```

```
96      machine = line.strip
97      `scp client.sh root@#{machine}:/root`
98      `ssh root@#{machine} ./client.sh #{masterServerIp}`
99    end
100 end
101
102 # resume
103 puts "\nmaster server :"
104 puts "#{masterServer} : #{masterServerIp}"
105
106 puts "\nmetalloger server :"
107 puts "#{metaloggerServer}"
108
109 puts "\nchunk servers :"
110 puts `cat listOfServers`
111
112 puts "\nMachines clients :"
113 puts `cat listOfClients`
```

Fichier masterServer.sh :

```
1  #!/bin/bash
2
3  # placement au bon endroit
4  cd /usr/src/mfs-1.6.20-2
5
6  # compilation avec les options qui vont bien
7  ./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var/lib --with-default
       -user=mfs --with-default-group=mfs --disable-mfschunkserver --disable-mfsmount
8
9  make
10
11 make install
12
13 # "creation" des fichiers de conf
14 cd /etc
15 cp mfsmaster.cfg.dist mfsmaster.cfg
16 cp mfsmetalogger.cfg.dist mfsmetalogger.cfg
17 cp mfsexports.cfg.dist mfsexports.cfg
18
19 cd /var/lib/mfs
20 cp metadata.mfs.empty metadata.mfs
21
22 # ajout du serveur maitre dans /etc/hosts
23 ipserver=`hostname -i`
24 echo "$ipserver mfsmaster" >> /etc/hosts
25
26 # demarrage du serveur
27 /usr/sbin/mfsmaster start
```

Fichier chunkServer.rb :

```ruby
#!/usr/bin/ruby -w

serverMasterIp = "#{ARGV[0]}"
numberOfChunk = "#{ARGV[1]}".to_i
numOfChunk = "#{ARGV[2]}".to_s

# placement au bon endroit
Dir.chdir("/usr/src/mfs-1.6.20-2")
#`cd /usr/src/mfs-1.6.20-2`

# compilation avec les options qui vont bien
`./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var/lib --with-default-user=mfs --with-default-group=mfs --disable-mfsmaster`

`make`

`make install`

# "creation" des fichiers de conf
Dir.chdir("/etc")
`cp mfschunkserver.cfg.dist mfschunkserver.cfg`
`cp mfshdd.cfg.dist mfshdd.cfg`

# conf fichier /etc/mfshdd.cfg
`mkdir /tmp/mfschunks#{numOfChunk}`

`chown -R mfs:mfs /tmp/mfschunks#{numOfChunk}`

#numberOfChunk.times do |i|
# numChunk = i + 1
# `echo "/tmp/mfschunks#{numChunk}" >> /etc/mfshdd.cfg`
#end

`echo "/tmp/mfschunks#{numOfChunk}" >> /etc/mfshdd.cfg`

# ajout du serveur maitre dans /etc/hosts
`echo "#{serverMasterIp} mfsmaster" >> /etc/hosts`
```

Fichier metaloggerServer.sh :

```bash
#!/bin/bash

serverMasterIp=$1

# placement au bon endroit
cd /usr/src/mfs-1.6.20-2

# compilation avec les options qui vont bien
./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var/lib --with-default-user=mfs --with-default-group=mfs --disable-mfschunkserver --disable-mfsmount

make

make install

# "creation" des fichiers de conf
```

```
cd /etc
cp mfsmetalogger.cfg.dist mfsmetalogger.cfg

# ajout du serveur maitre dans /etc/hosts
echo "$serverMasterIp mfsmaster" >> /etc/hosts

# demarrage du serveur
/usr/sbin/mfsmetalogger start
```

Fichier client.sh :

```
#!/bin/bash

serverMasterIp=$1

# placement au bon endroit
cd /usr/src/mfs-1.6.20-2

# compilation avec les options qui vont bien
./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var/lib --with-default
    -user=mfs --with-default-group=mfs --disable-mfsmaster --disable-mfschunkserver

make

make install

# ajout du serveur maitre dans /etc/hosts
echo "$serverMasterIp mfsmaster" >> /etc/hosts

# rep de montage
mkdir /media/mfs
# montage
/usr/bin/mfsmount /media/mfs -H mfsmaster
```

B.3 Ceph

Fichier deploimentCeph.rb :

```
#!/usr/bin/ruby -w
# encoding: utf-8

##############################################

# File Name : deploiementCeph.rb

# Creation Date : 11-03-2011

# Last Modified : dim. 27 mars 2011 22:15:42 CEST

# Created By : Helldar

##############################################

# doit concorder avec la commande oarsub

```

```
18 if ARGV[0] != nil
19   numberOfServers = ARGV[0].to_i
20   puts "Nb serveur : #{numberOfServers}\n"
21 else
22   puts "Veuillez relancer le script avec les bons parametres!\nUsage : <nombre_de_
       serveur>"
23   exit
24 end
25
26 # creation d'un fichier contenant la liste des noeuds reserves
27 `touch listOfNodes`
28 File.open("listOfNodes", 'w') do |file|
29         file << `cat $OAR_FILE_NODES | sort -u`
30 end
31 # creation de deux fichiers contenant la liste des serveurs, et des clients
32
33 `touch listOfClients listOfServers`
34 serverWrited = 0
35 File.open("listOfNodes", 'r') do |node|
36         File.open("listOfServers", 'w') do |server|
37                 File.open("listOfClients", 'w') do |client|
38                         while line = node.gets
39                                 if serverWrited < numberOfServers
40                                         server << line
41                                         serverWrited += 1
42                                 else
43                                         client << line
44                                 end
45                         end
46                 end
47         end
48 end
49
50 # deploiement des machines
51 puts "Machines en cour de deploiement..."
52 `kadeploy3 -k -e squeeze-collective -u flevigne -f listOfNodes` # image collective
53
54 # configuration du serveur
55 serveur_1 = `head -1 listOfServers | cut -d "." -f1`.strip
56 ip_serveur = `ssh root@#{serveur_1}  ifconfig eth0 |grep inet\  | cut -d ":" -f2 |
     cut -d ' ' -f1`.strip
57
58 # generation du fichier de ceph.conf
59
60 `touch ceph.conf`
61 File.open("ceph.conf", 'w') do |file|
62      file << "[global]
63              pid file = /var/run/ceph/$name.pid
64              debug ms = 1
65              keyring = /etc/ceph/keyring.bin
66 [mon]
67              mon data = /tmp/partage/mon$id
68 [mon0]
69              host = #{serveur_1}
70              mon addr = #{ip_serveur}:6789
71 [mds]
```

```
72                debug mds = 1
73                keyring = /etc/ceph/keyring.$name"
74    if numberOfServers > 3
75      1.upto(3) { |i|
76        file << "
77  [mds#{i - 1}]"
78        host = `sed -n #{i + 1}p listOfServers | cut -d '.' -f1`.strip
79        puts "host #{i} : #{host}\n"
80        file << "
81         #{host}"
82      }
83    else
84      file << "[mds0]"
85      host = `sed -n 2p listOfServers | cut -d '.' -f1`.strip
86      file << "
87         #{host}"
88    end
89    file << "
90  [osd]
91         sudo = true
92         osd data = /tmp/partage/osd$id
93         keyring = /etc/ceph/keyring.$name
94         debug osd = 1
95         debug filstore = 1
96         osd journal = /tmp/partage/osd$id/journal
97         osd journal size = 1000"
98    1.upto(numberOfServers) { |i|
99      file << "
100 [osd#{i - 1}]"
101     host = `sed -n #{i}p listOfServers | cut -d '.' -f1`.strip
102     file << "
103        #{host}"
104   }
105 end
106
107 # copie du fichier ceph.conf vers le serveur
108 `scp ceph.conf root@#{serveur_1}:/etc/ceph`
109 puts "Envoye!"
110
111 # generation du fichier keyring.bin
112 `ssh root@#{serveur_1} cauthtool --create-keyring -n client.admin --gen-key keyring
        .bin`
113 `ssh root@#{serveur_1} cauthtool -n client.admin --cap mds 'allow' --cap osd 'allow
        *' --cap mon 'allow rwx' keyring.bin`
114 `ssh root@#{serveur_1} mv keyring.bin /etc/ceph/`
115 puts "Keyring genere!"
116
117 # montage
118 `ssh root@#{serveur_1} mount -o remount,user_xattr /tmp`
119 1.upto(numberOfServers - 1) { |i|
120   serveurs = `sed -n #{i + 1}p listOfServers | cut -d "." -f1`.strip
121   `ssh root@#{serveurs} mount -o remount,user_xattr /tmp`
122 }
123 puts "Montage fait!"
124
125 # demarrage du serveur
```

```
126 `ssh root@#{serveur_1} mkcephfs -c /etc/ceph/ceph.conf --allhosts -v -k /etc/ceph/
        keyring.bin`
127 `ssh root@#{serveur_1} /etc/init.d/ceph -a start`
128 puts "Serveur ceph demarre!"
129
130 # configuration des clients
131 1.upto(`wc -l listOfClients`.to_i) { |i|
132   clients = `sed -n #{i}p listOfClients | cut -d "." -f1`.strip
133   `ssh root@#{clients} mkdir /ceph`
134   `ssh root@#{clients} cfuse -m #{ip_serveur} /ceph`
135 }
136 puts "Clients montes!"
```

B.4 NFS

Fichier deploimentNFS.rb :

```
1  #!/usr/bin/ruby -w
2  # encoding: utf-8
3
4  #################################################
5
6  # File Name : deploiementNFS.rb
7
8  # Creation Date : 17-03-2011
9
10 # Last Modified : dim. 27 mars 2011 22:15:59 CEST
11
12 # Created By : Helldar
13
14 #################################################
15
16 `cat $OAR_FILE_NODES | sort -u > listOfNodes`
17
18 # Deploiement des machines
19 #puts "Machines en cour de deploiement...\n"
20 #`kadeploy3 -k -e squeeze-collective -u flevigne -f listOfNodes # image collective`
21
22 serveur = `head -1 listOfNodes`.strip
23 puts "Le serveur : #{serveur}!\n"
24 # Suppression du serveur de la liste
25 `sed -i 1d listOfNodes`
26
27 puts "Configuration du serveur...\n"
28 `scp exports root@#{serveur}:/etc/`
29 `ssh root@#{serveur} /etc/init.d/nfs-kernel-server restart`
30
31 puts "Configuration des clients...\n"
32 line = `wc -l listOfNodes | cut -d ' ' -f1`.strip.to_i
33 puts "Il y a #{line} nodes"
34 1.upto(line) { |i| clients = `sed -n #{i}p listOfNodes | cut -d "." -f1`.strip
35   `ssh root@#{clients} mkdir /tmp/partage`
36   `ssh root@#{clients} mount -t nfs4 #{serveur}:/ /tmp/partage` }
```

B.5 Benchmark

Fichier benchmark.rb :

```ruby
#!/usr/bin/ruby -w
# encoding: utf-8


if ARGV[0] == nil || ARGV[1] == nil || ARGV[2] == nil
  puts "Usage correcte :"
  puts "param1 : nombe de clients participant au bench"
  puts "param2 : fichier de sortie"
  puts "param3 : url de la liste des clients"
  puts "param4 : lieu d'ecriture du bench"
  exit(1)
end

$clientsOfBench = "#{ARGV[0]}"

# chemin du fichier contenant la liste des clients
#listOfClients = "/home/flevigne/glusterFs/listOfClients"
listOfClients = "#{ARGV[2]}"

# chemin ou ecrire les donnees du benchmark
#whereToWrite = "/media/glusterfs"
whereToWrite = "#{ARGV[3]}"

# chemin du fichier contenant les resultats
$outputRes = "#{ARGV[1]}"

# le client doit avoir dans /home/flevigne :
# - linux-2.6.37.tar.bz2 : noyau linux compresse
# - bigFile : un fichier de 3 Go

# fichier contenant la liste des clients participant au benchmark
`touch clientOfBench`
`head -#{$clientsOfBench} #{listOfClients} > clientOfBench`

# si le fichier $outputRes n'existe pas, on le cree.
if !File.exist?($outputRes)
  `touch #{$outputRes}`
end

`echo "\nBenchmark sur #{$clientsOfBench} clients" >> #{$outputRes}`

$numberOfClients = open("clientOfBench").read.count("\n").to_i
puts "nombre de clients : #{$numberOfClients}"

puts "Lancement du benchmarck sur #{$numberOfClients} clients."


# lance un travail
# parametres :
# - name : nom du travail (str)
# - work : chemin du script de travail (str)
# - whereToWrite : chemin ou ecrire les donnees du benchmark (str)
# - size : taille (en Mo) du/des fichier(s) a ecrire/lire (float)
```

```
54 def startBench(name, work, whereToWrite, size)
55   puts "bench : #{name} en cours..."
56
57   totalSize = size.to_i * $clientsOfBench.to_i
58   workFinished = 0
59   startOfBench = Time.now
60
61   # execution du sript pour tous les clients
62   File.open("clientOfBench", 'r') do |file|
63     while line = file.gets
64       fork do
65         machine = line.split.join("\n")
66         `scp #{work} root@#{machine}:/root`
67         `ssh root@#{machine} ./#{work} #{whereToWrite}`
68         exit(0)
69       end
70     end
71   end
72
73   # on attend que tous les clients aient fini leur travail
74   1.upto($numberOfClients) do
75     pid = Process.wait
76     workFinished += 1
77     puts "Machine(s) ayant termine leur travail : #{workFinished}"
78   end
79
80   endOfBench = Time.now
81   duration = endOfBench - startOfBench
82
83   puts "Toute les machines ont termine leur travail."
84
85   puts " --> Le benchmark \"#{name}\" a dure #{duration} secondes. (debit : #{
       totalSize / duration} Mo/s)"
86
87   `echo "#{name} : #{duration} sec : #{totalSize / duration} Mo/s" >> #{$outputRes
       }`
88 end
89
90
91 # lancement du benchmark
92 startBench("ecriture de petits fichiers", "writingSmallFiles.sh", whereToWrite,
     479)
93 startBench("ecriture de gros fichiers", "writingBigFiles.sh", whereToWrite, 3076)
94 startBench("lecture de petits fichiers", "readingSmallFiles.sh", whereToWrite, 479)
95 startBench("lecture de gros fichiers", "readingBigFile.sh", whereToWrite, 3076)
96
97 # nettoyage du systeme de fichier distribue (necessaire pour enchainer les
     benchmark)
98 puts "Nettoyage de l'espace de travail..."
99 oneClient = `head -1 clientOfBench`.strip
100 `ssh root@#{oneClient} rm -r #{whereToWrite}/*`
101
102 puts "\nBenchmark termine"
```

Fichier writingSmallFiles.sh :

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

```
2
3 whereToWrite=$1
4
5 nameOfMachine=`uname -n`
6
7 # creation du repertoire de travail de la machine
8 mkdir "$whereToWrite/$nameOfMachine"
9
10 # decompression dans ce repertoire
11 cd "$whereToWrite/$nameOfMachine"
12 tar -xf /home/flevigne/linux-2.6.37.tar.bz2
```

Fichier writingBigFiles.sh :

```
1 #!/bin/bash
2
3 whereToWrite=$1
4
5 nameOfMachine=`uname -n`
6
7 # on copie les gros fichiers au lieu voulu
8 cp /home/flevigne/bigFile1 "$whereToWrite/$nameOfMachine"
9 cp /home/flevigne/bigFile2 "$whereToWrite/$nameOfMachine"
10 cp /home/flevigne/bigFile3 "$whereToWrite/$nameOfMachine"
```

Fichier readingSmallFiles.sh :

```
1 #!/bin/bash
2
3 whereToWrite=$1
4
5 nameOfMachine=`uname -n`
6
7 cd "$whereToWrite/$nameOfMachine"
8
9 # lecture des fichiers du noyau linux (compression (donc lecture) redirigevers /dev
    /nul)
10 tar -cf /dev/null linux-2.6.37
```

Fichier readingBigFile.sh :

```
1 #!/bin/bash
2
3 whereToWrite=$1
4
5 nameOfMachine=`uname -n`
6
7 cd "$whereToWrite/$nameOfMachine"
8
9 # lecture des gros fichiers
10 cat bigFile1 > /dev/nul
11 cat bigFile2 > /dev/nul
12 cat bigFile3 > /dev/nul
```

B.6 Meta scripts

Fichier meta_NFS.sh :

```
#!/bin/bash

# la reservation doit etre faite a la main :
# oarsub -I -t deploy -l nodes=10,walltime=2
# oarsub -I -t deploy -l nodes=10,walltime=2 -p "cluster='graphene'"

# NFS
cd ~/NFS
echo "./deploiementNFS.rb"
./deploiementNFS.rb

# benchmark sur 1 client
cd ~/benchmark
./benchmark.rb 1 "~/resOfBench/gluster" ~/NFS/listOfNodes /tmp/partage
# benchmark sur 5 clients
./benchmark.rb 5 "~/resOfBench/gluster" ~/NFS/listOfNodes /tmp/partage
# benchmark sur 20 clients
./benchmark.rb 20 "~/resOfBench/gluster" ~/NFS/listOfNodes /tmp/partage
# benchmark sur 50 clients
./benchmark.rb 50 "~/resOfBench/gluster" ~/NFS/listOfNodes /tmp/partage
```

Fichier meta_gluster.sh :

```
#!/bin/bash

# la reservation doit etre faite a la main :
# oarsub -I -t deploy -l nodes=10,walltime=2
# oarsub -I -t deploy -l nodes=10,walltime=2 -p "cluster='graphene'"

if [ -z $1 ]
then
    echo -e "Usage : <nombre de serveurs>"
    exit 1
fi

nb_serveur=$1

# gluster n serveurs
cd ~/glusterFS
echo "./deploiementGluster.rb $nb_serveur"
./deploiementGluster.rb $nb_serveur

# benchmark sur 1 client
cd ~/benchmark
./benchmark.rb 1 "~/resOfBench/gluster$nb_serveur" ~/glusterFS/listOfClients /media
    /glusterfs
# benchmark sur 5 clients
./benchmark.rb 5 "~/resOfBench/gluster$nb_serveur" ~/glusterFS/listOfClients /media
    /glusterfs
# benchmark sur 20 clients
./benchmark.rb 20 "~/resOfBench/gluster$nb_serveur" ~/glusterFS/listOfClients /
    media/glusterfs
# benchmark sur 50 clients
```

```
./benchmark.rb 50 "~/resOfBench/gluster$nb_serveur" ~/glusterFS/listOfClients /
    media/glusterfs
```

Fichier meta_ceph.sh :

```bash
#!/bin/bash

# la reservation doit etre faite a la main :
# oarsub -I -t deploy -l nodes=10,walltime=2
# oarsub -I -t deploy -l nodes=10,walltime=2 -p "cluster='graphene'"

if [ -z $1 ]
then
    echo -e "Usage : <nombre de serveurs>"
    exit 1
fi

nb_serveur=$1

# ceph 5 serveurs
cd ~/cephFS
./deploiementCeph.rb $nb_serveur

# benchmark sur 1 client
cd ~/benchmark
./benchmark.rb 1 "~/resOfBench/ceph$nb_serveur" ~/cephFS/listOfClients /ceph
# benchmark sur 5 clients
./benchmark.rb 5 "~/resOfBench/ceph$nb_serveur" ~/cephFS/listOfClients /ceph
# benchmark sur 20 clients
./benchmark.rb 20 "~/resOfBench/ceph$nb_serveur" ~/cephFS/listOfClients /ceph
# benchmark sur 50 clients
./benchmark.rb 50 "~/resOfBench/ceph$nb_serveur" ~/cephFS/listOfClients /ceph
```

Fichier meta_moose.sh :

```bash
#!/bin/bash

# la reservation doit etre faite a la main :
# oarsub -I -t deploy -l nodes=10,walltime=2
# oarsub -I -t deploy -l nodes=10,walltime=2 -p "cluster='graphene'"

if [ -z $1 ]
then
    echo -e "Usage : <nombre de serveurs>"
    exit 1
fi

nb_serveur=$1

# mooseFs 5 serveurs
cd ~/mooseFs
./deploiementMoose.rb $nb_serveur

# benchmark sur 2 clients
cd ~/benchmark
#echo "./benchmark.rb 2 \"~/resOfBench/moose$nb_serveur\" ~/mooseFs/listOfClients /
    media/mfs"
./benchmark.rb 1 "~/resOfBench/moose$nb_serveur" ~/mooseFs/listOfClients /media/mfs
```

```
23 # benchmark sur 5 clients
24 ./benchmark.rb 5 "~/resOfBench/moose$nb_serveur" ~/mooseFs/listOfClients /media/mfs
25 # benchmark sur 20 clients
26 ./benchmark.rb 20 "~/resOfBench/moose$nb_serveur" ~/mooseFs/listOfClients /media/
     mfs
27 # benchmark sur 50 clients
28 ./benchmark.rb 50 "~/resOfBench/moose$nb_serveur" ~/mooseFs/listOfClients /media/
     mfs
```

Partie C

Résultats relevés

C.1 NFS

Écriture de petits fichiers avec NFS (débit en Mo/sec) :

	1 client	5 clients	20 clients	50 clients
1 serveur	1.76	1.17	1.31	

Écriture de gros fichiers avec NFS (débit en Mo/sec) :

	1 client	5 clients	20 clients	50 clients
1 serveur	35.11	37.83	8.54	

Lecture de petits fichiers avec NFS (débit en Mo/sec) :

	1 client	5 clients	20 clients	50 clients
1 serveur	71.53	145.78	527.97	

Lecture de gros fichiers avec NFS (débit en Mo/sec) :

	1 client	5 clients	20 clients	50 clients
1 serveur	1675.29	1172.55	329.87	

C.2 GlusterFS

Écriture de petits fichiers avec GlusterFS (débit en Mo/sec) :

	1 client	5 clients	20 clients	50 clients
5 serveurs				
20 serveurs				
50 serveurs	38.55	209.48	804.68	1957.51

Écriture de gros fichiers avec GlusterFS (débit en Mo/sec) :

	1 client	5 clients	20 clients	50 clients
5 serveurs				
20 serveurs				
50 serveurs	254.01	1533.95	5249.86	11407.30

Lecture de petits fichiers avec GlusterFS (débit en Mo/sec) :

	1 client	5 clients	20 clients	50 clients
5 serveurs				
20 serveurs				
50 serveurs	42.82	200.15	802.95	1756.84

Lecture de gros fichiers avec GlusterFS (débit en Mo/sec) :

	1 client	5 clients	20 clients	50 clients
5 serveurs				
20 serveurs				
50 serveurs	287.22	1461.58	4914.09	11139.07

C.3 MooseFS

Les résultats de MooseFS avec 5 serveurs ont été obtenus sur un autre cluster.

Écriture de petits fichiers avec MooseFS (débit en Mo/sec) :

	1 client	5 clients	20 clients	50 clients
5 serveurs	2.43	5.58	15.14	14.53
20 serveurs	4.37	13.93	17.93	17.80
50 serveurs	4.17	18.01	34.35	36.22

Écriture de gros fichiers avec MooseFS (débit en Mo/sec) :

	1 client	5 clients	20 clients	50 clients
5 serveurs	21554.35	2536.51	9901.15	23020.77
20 serveurs	79.83	308.10	573.01	545.86
50 serveurs	92.33	385.99	1295.74	1631.78

Lecture de petits fichiers avec MooseFS (débit en Mo/sec) :

	1 client	5 clients	20 clients	50 clients
5 serveurs	31.54	162.36	667.75	1485.40
20 serveurs	35.03	172.10	569.37	853.36
50 serveurs	31.97	163.32	567.77	823.54

Lecture de gros fichiers avec MooseFS (débit en Mo/sec) :

	1 client	5 clients	20 clients	50 clients
5 serveurs	21093.63	2540.53	9928.70	23112.12
20 serveurs	1779.99	8525.03	31322.11	64509.88
50 serveurs	1755.61	8468.74	29746.91	57816.35

C.4 Ceph

Les résultats de Ceph avec 5 serveurs ont été obtenus sur un autre cluster.

Écriture de petits fichiers avec Ceph (débit en Mo/sec) :

	1 client	5 clients	20 clients	50 clients
5 serveurs	79.58	395.13	353.26	771.53
20 serveurs				
50 serveurs				

Écriture de gros fichiers avec Ceph (débit en Mo/sec) :

	1 client	5 clients	20 clients	50 clients
5 serveurs	501.88	2537.34	9930.57	23751.55
20 serveurs				
50 serveurs				

Lecture de petits fichiers avec Ceph (débit en Mo/sec) :

	1 client	5 clients	20 clients	50 clients
5 serveurs	79.66	395.13	1545.80	3695.22
20 serveurs				
50 serveurs				

Lecture de gros fichiers avec Ceph (débit en Mo/sec) :

	1 client	5 clients	20 clients	50 clients
5 serveurs	510.82	2536.10	9915.92	23755.81
20 serveurs				
50 serveurs				

Partie D

Sources

- Pour l'ensemble de notre travail : Wikipédia Français et Anglais
- NFS :
  - Documentation de Wikipedia anglais [1]
  - Tutoriel de Ubuntu-fr [2]
  - Documentation de Sourceforge [3]
- GlusterFS :
  - Documentation de GlusterFS [4]
  - Gnu/Linux Magazine numéro 133 : « Introduction à GlusterFS »
- MooseFS :
  - Documentation de MooseFS [5], en particulier « Installing MooseFS Step by Step Tutorial » [6]
- Ceph :
  - Site internet et Wiki de Ceph [7]
  - Documentations de Ceph [8] [9]

1. http ://en.wikipedia.org/wiki/Distributed_file_system
2. http ://doc.ubuntu-fr.org/nfs
3. http ://nfs.sourceforge.net/
4. http ://gluster.com/community/documentation/index.php/Main_Page
5. http ://www.moosefs.org/reference-guide.html
6. http ://www.moosefs.org/tl_files/manpageszip/moosefs-step-by-step-tutorial-v.1.1.pdf
7. http ://ceph.newdream.net/ http ://ceph.newdream.net/wiki/Main_Page
8. http ://www.ssrc.ucsc.edu/Papers/weil-osdi06.pdf
9. http ://www.usenix.org/publications/login/2010-08/openpdfs/maltzahn.pdf