Université de Lorraine

Projet Tutoré

asrall

Systèmes de Fichiers Distribués

Author :
Ricardo RODRÍGUEZ
Julien SCHNEIDER
Jean LUTZ
Yannick LAPREVOTTE

Supervisor :
Stephane CASSET

24 mars 2014

Table des matières

1 Introduction

2 CEPH

2.1 Présentation

Ceph est un système de fichier open-source, développé à partir d'un algorithme de nouvelle génération appelé CRUSH [1]. Il est évolutif (*scalable*) et est capable de fonctionner sur un parc de machines très diverses, on appelle ces parc de machines un cluster Ceph. Son fonctionnement repose sur 3 types de serveurs :

les OSDs *(Object Storage Deamon "OSD")* : sont les serveurs de stockage. Les données seront donc enregistrées sur ces nœuds. Il est préférable d'avoir une bonne quantité d'espace disque, car les OSDs journalisent les opérations d'écritures en donnée ce qui prend de la place de stockage. Il faut au minimum 2 OSDs pour que le cluster Ceph soit opérationnel.

Les Monitors *(Ceph Monitor)* : ceux-ci sont en place pour surveiller que tout fonctionne correctement. Lorsque l'on dispose d'un réseau conséquent en nœuds, il est important de savoir très rapidement quand il y a un soucis quelque part. Il est mieux d'avoir plusieurs Monitors dans le cluster Ceph pour permettre de repérer plus rapidement les erreurs et avoir un bon niveau de tolérance aux pannes. Les monitors sont des daemons relativement léger et ne nécessite pas une grande quantité de mémoire ou d'espace disque.

Les MDS *(MetaData Server)* : C'est le troisième type de serveur. Celui-ci détient toutes les informations permettant de trouver les données demandées et leurs attributs. Ces informations sont très souvent consultées, elle sont dont stockées en mémoires pour en amélioré l'accès. Il faut donc une grande quantité de RAM sur les ordinateur qui hébergent les MDS.

Attention il n'est pas encore recommandé de l'utiliser en production : celui-ci est encore en phase de développement.

1. Ceph est propulsé par CRUSH (Controlled Replication Under Scalable Hashing), un algorithme de hash dérivé de la famille des algorithmes RUSH.

2.2 Architecture du cluster

Nous avons choisit d'installer une configuration nous permettant de tester les performances de Ceph. Comme dit précédemment, Ceph a besoin de 3 types de nœuds différents. Nous avons choisit d'utiliser 6 machines pour 9 noeuds, ces machines seront utilisé sous Debian :

-La première machine sera l'Admin node, elle contiendra aussi le premier monitor et le premier OSD :

```
nom machine : golem
ip : 192.168.1.51
```

- La deuxième machine contiendra le deuxième Monitor et le second OSD :

```
nom machine : rondoudou
ip :  192.168.1.29
```

- La troisième machine contiendra le troisième Monitor et le troisième OSD :

```
nom machine : behemot
ip :    192.168.1.56
```

- La quatrième machine contiendra le premier MDS :

```
nom machine : carapuce
ip : 192.168.1.43
```

- La cinquième machine contiendra le deuxième MDS :

```
nom machine : smogogo
ip : 192.168.1.2
```

- La sixième machine contiendra le troisième MDS :

```
nom machine : porygon
ip : 192.168.1.48
```

Nous avons également dû installer les clients qui utilise ce cluster nous les avons installé sur nos machines personnelles.

2.3 Installation

Ceph a été créer pour fonctionner sur du matériel de base, pas besoin d'avoir de grosse configuration ou de matériel spécifique pour faire tourner ceph, ce qui rend la construction et le maintien d'un cluster de données de l'ordre du péta-octets[2] facilement et économiquement réalisable. Pour l'installation de Ceph, il faut d'abord configurer le matériel que nous allons utiliser.

2.3.1 Installation Matériel

CPU :

MDS : Les serveurs de méta-donnée[3] redistribue dynamiquement leur méta-donnée, ce qui utilise une grande quantité de puissance processeur *CPU*[4]. Les MDS doivent avoir un assez bon processeur (quad-core ou mieux).

OSD : Les serveur de donnée ceph font tourner le service RADOS (*Reliable Autonomic Distributed Object Store*)[5], calculent le placement des datas avec CRUSH, répliquent les données, et maintiennent des copies de la carte[6] (*map*) du cluster. Les serveur de donnée ont un besoin raisonnables de puissance CPU.

Monitor : les monitor n'ont pas besoin de puissance CPU, ils ne font que maintenir une copie de la carte du cluster pour repérer les erreurs dans le cluster.

RAM :

La ram est surtout utilisé par les serveurs de méta-donnée et les monitors, car ils doivent être capable de parcourir leur données rapidement, ils doivent donc avoir une bonne quantité de RAM. Il est recommandé d'avoir 1 GB

2. Un péta-octet contient 10^{15} octets

3. c'est une donnée servant à définir ou décrire une autre donnée quel que soit son support

4. est le composant de l'ordinateur qui exécute les instructions machine des programmes informatiques.

5. RADOS s'occupe de la distribution des objets dans l'ensemble du cluster de stockage et les reproduire pour la tolérance de panne

6. Une série de cartes comportant la carte de moniteur,la carte des OSDs,la carte PG, la carte des MDS et la carte CRUSH

de RAM par daemon[7]. Les Osds n'utilisent pas beaucoup de RAM pour leurs fonctionnements normal, mais ils en utilisent plus si l'on lance une récupération de donnée.

Nous avons installé 6 machines pour créer un cluster Ceph avec un total de 9 noeuds sur ces machines :

MDS : Les MDS ont été installé sur les 3 machines les plus puissantes : carapuce, smogogo, porygon. Elles ont toute un CPU quad-core et assez de RAM pour faire marcher un MDS.

Monitors et OSDs : Les Monitors et les OSDs ont été installé sur les 3 machines restantes : golem, rondoudou, behemot. Elles ont assez d'espace disque et de RAM pour faire marcher ces 2 services.

2.3.2 Configuration Réseau

Pour la configuration réseau du système de fichier distribué, nous avons permit aux machines de se connecter entre elle de manière sécurisé grâce au protocole ssh. Nous avons modifié la configuration ldu fichier /etc/hosts en rajoutant toutes nos machines dedans. Ensuite nous avons créer un utilisateur sur chaque machines qui nous permettra d'installer ceph, Pour finir nous avons échangé les clés de sécurité ssh entre toute les machines pour sécuriser les connexions.

2.3.3 Mise à Jour système

Nous avons mit à jour le système de nos machines en installant et en configurant un serveur de synchronisation d'horloge, le service NTP, pour ne pas avoir de décalage entre les machines ce qui pose problème sur un système de fichier distribué, pour la copie ou la lecture de fichier sur ceph. Et aussi LSB qui standardise la structure des systèmes d'exploitation GNU/Linux, ceph est alors installé sur des systèmes d'exploitations similaires. Pour installer Ceph sur la première machine (golem), nous avons ajouté les dépôts ceph dans le fichier de source /etc/apt/sources/list, et aussi installé les paquets $ceph$, $ceph-deploy$.
La création du répertoire de travail dans le dossier courant de l'utilisateur ceph vas nous permettre d'installer ceph à partir de l'Admin Node.

7. désigne un type de programme informatique, un processus ou un ensemble de processus qui s'exécute en arrière-plan plutôt que sous le contrôle direct d'un utilisateur.

2.3.4 Création du cluster et installation des monitors

En premier nous allons installer le cluster et les 3 Monitors avec l'utilisation de la commande *ceph – deploy* :

— D'abord l'installation des nodes sur les machines avec :

```
ceph-deploy new golem rondoudou behemot
```

— L'installation de ceph sur les autres machines à partir de l'Admin Node (golem), où nous avons ajouté une option pour ignorer le proxy du réseau de la salle de classe : –no-adjust-repos.

— Et la création des Monitors avec :

```
ceph-deploy mon create-initial
```

Cette commande génère les monitors et créer la configuration initial du cluster.

2.3.5 Installation des OSDs

Pour l'installation, nous avons créer et formater une partition en xfs pour stocker les données enregistré sur le cluster avec l'aide Gparted, les partitions sur les machines font chacune 10 Go. Ce qui nous donnent en tous 30 Go d'espace disque pour le cluster. Avec la préparation et l'activation des OSDs en utilisant les partitons en xfs :

```
ceph-deploy osd prepare <nomdemachine>:<partitions_xfs>
ceph-deploy osd activate <nomdemachine>:<partiton xfs>
```

Après ces commandes les OSDs sont activé et propre. Notre cluster commence a être opérationnel.

2.3.6 Installation des MDS

Les MDS sont installé avec la commande :

```
ceph-deploy mds create <nomdemachine>
```

Maintenant le cluster a 1 MDS activé et 2 en standby ils attendent jusqu'à ce que le MDS actif crash ou lag pour s'activer et mettre le MDS instable en standby. Mais nous voulons 2 mds actif, car les MDS sont des goulots d'étranglement pour l'enregistrement des fichiers sur ceph. 2 MDS permettraient de pourvoir réaliser des benchmarks avec de meilleurs performances. Nous utilisons alors la commande :

```
ceph mds set_max_mds_ 2
```

2.3.7 Installation du client

Le client vas être sur une machine que l'on a ajouté dans le réseau de ceph avec l'aide de ssh, la création d'un utilisateur ceph, et la copie de la configuration du cluster. Nous avons alors créer notre premier bloc de donnée dans une piscine du cluster ceph il a 3 piscines de base, nous utilisons rbd :

```
ceph-deploy create <nom du bloc> --size <espace disque> --pool
 <piscine utilisé>
```

Nous avons ajouté le bloc de donnée dans la carte de rbd, pour que le bloc puisse être repéré.

```
rbd map <nom du bloc> -pool <piscine utilisé>
```

Ensuite nous avons ajouté un système de fichier sur le bloc de donné créer.

```
mkfs.rxt4 -m0 /dev/rbd/<nom du bloc>
```

Et nous avons monté la partition créer sur le client.

```
mount /dev/rbd/<nom du bloc> /mnt/rbd_foo
```

Ce qui nous permet d'utiliser ceph comme une partition de notre système, mais enregistrer sur 3 serveurs de données ce qui permet une haut-accessibilité et une haute-tolérance aux pannes. Le cluster ceph est entièrement installé, nous pouvons commencer les benchmarks.

2.4 Erreurs rencontrées lors de l'installation de Ceph

2.4.1 Erreur lors de l'installation du premier MDS

Pendant notre 1ère installation de Ceph, nous avions réussit à installer :
1 Monitor sur golem et 2 OSDs sur golem et rondoudou, mais lorsque nous avons tenté d'installer le 1èr MDS avec la commande :

```
ceph@golem:~/cluster$ceph-deploy mds create carapuce
```

Il y a eu des erreurs :

```
[carapuce][WARNIN] No data was received after 300 seconds, disconnecting...
Traceback (most recent call last):
  File "/usr/bin/ceph-deploy", line 21, in <module>
    sys.exit(main())
 File "/usr/lib/python2.7/dist-packages/ceph_deploy/util/decorators.py",
  line 62, in newfunc
    return f(*a, **kw)
 File "/usr/lib/python2.7/dist-packages/ceph_deploy/cli.py", line 138,
  in main
 return args.func(args)
File "/usr/lib/python2.7/dist-packages/ceph_deploy/mds.py", line 169,
 in mds
    mds_create(args)
 File "/usr/lib/python2.7/dist-packages/ceph_deploy/mds.py", line 157,
  in mds_create
    create_mds(distro.conn, name, args.cluster, distro.init)
 File "/usr/lib/python2.7/dist-packages/ceph_deploy/mds.py", line 55,
 in create_mds
    os.path.join(keypath),
  TypeError: 'NoneType' object is not iterable
```

Après cette erreur le cluster que nous avions installé n'était plus utilisable , la commande :

```
ceph@golem:~/cluster$ceph status
```

nous a renvoyé :

```
10/02/2014 : erreur lors de la commande ceph status et ceph health
 pour vérifier le fonctionnement de ceph,les commandes renvoient :

    root@golem:~# ceph status

    2014-02-10 15:58:38.386216 7fdab0128700  0 -- :/1030939 >>
     192.168.1.51:6789/0 pipe(0x185e3c0 sd=3 :0 s=1 pgs=0 cs=0 l=1
     c=0x185e620).fault

    2014-02-10 15:58:41.381650 7fdaa8d83700  0 -- :/1030939 >>
     192.168.1.51:6789/0 pipe(0x185d510 sd=3 :0 s=1 pgs=0 cs=0 l=1
     c=0x185d770).fault

    2014-02-10 15:58:44.381919 7fdab0128700  0 -- :/1030939 >>
     192.168.1.51:6789/0 pipe(0x185b000 sd=3 :0 s=1 pgs=0 cs=0 l=1
     c=0x185b260).fault

    ^CError connecting to cluster: InterruptedOrTimeoutError
```

Nous avons supprimer le cluster ceph avec purge autoremove, nous avons également supprimer le cluster de l'Admin Node et les fichier de conf et nous avons lancé une réinstallation complète de ceph. Pour cette réinstallation, nous avons suivi le tutoriel du site officiel pour recommencer.

2.4.2 Problème de cluster

Problème de cluster nous avions 3 cluster de ceph installé sur golem :
$Un\ dans\ /home/root/cluster\ fsid\ =\ 490b1d93-f413-430b-b5d7-d632b7556f35$
$Un\ dans\ /home/ceph\ fsid\ =\ 0346f25b-53f7-43ce-9018-8e37181f844d$
$Un\ dans\ /home/ceph/cluster\ fsid\ =\ d4a63950-a12f-4e4d-983b-57583bf09bac$
Mais lors de nos manipulations sur notre 1ère installation, nous avons lancé des commandes ceph-deploy en-dehors de notre répertoire cluster, ce qui a créer de nouveau fichier de configuration et un autre cluster instable avec un nom par défaut. Mais nous n'avions pas changer le nom par défaut de notre cluster et donc

plusieurs cluster du même nom ont été créer, et les cluster en-dehors du répertoire cluster était instable, ce qui a posé problème lors du démarrage du service ceph.
On a alors purge toute l'installation de ceph avec :

```
ceph-deploy purgedata golem rondoudou carapuce behemot porygon smogogo
ceph-deploy forgetkeys
ceph-deploy purge golem rondoudou carapuce behemot porygon smogogo
```

Pour avoir un seul cluster il faut lancer toutes les commandes d'installation de ceph dans un seul dossier : /home/ceph/cluster
on a ajouter dans le .bashrc de ceph des alias :
alias ceph-deploy='cd /home/ceph/cluster ; ceph-deploy'
alias ceph='cd /home/ceph/cluster ; ceph'

2.4.3 Problème de l'horloge système des machines

Vers la fin de notre projet nos machines on eu un décalage d'horloge (*clock skew*) de quelque seconde, ce qui empêché ceph de fonctionner normalement, Ceph est comme un file system[8] mais sur plusieurs machines si ces machines n'ont pas une horloge synchronisé, la copie de fichier peut causer des problèmes. Pour palier ce problème nous avons dû reconfigure ntp :

Serveur NTP *(Network Time Protocol "NTP")* : On a fait un serveur ntp sur golem en modifiant le fichier /etc/ntp.conf en enlevant les serveur de temps extérieur et en rajoutant golem comme serveur :

```
  Fichier /etc/ntp.conf
server 127.127.1.0 # adresse localhost utilisé par ntp
server 192.168.1.51 # adresse de golem pour le nommé comme serveur ntp
fudge 127.127.1.0 stratum 4 # ajouter l'horloge système comme source
de temps par le paramètre fudge et pour le mettre en stratum 4

restrict 192.168.1.0 mask 255.255.255.0  nomodify # pour restricter
l'accès au serveur et ne permettre qu'aux machine sur le réseau
de se synchroniser sur le serveur ntp
```

8. c'est une façon de stocker les informations et de les organiser dans des fichiers

```
broadcast 192.168.1.255 # l'adresse de broadcast
```

Le serveur ntp est fonctionnel, il permet de diffuser son horloge sur les clients.

Client NTP : L'installation du client ntp se fait avec apt :

```
apt-get install ntpdate
```

Nous avons ensuite lancé une mise à jour de l'horloge des client à partir du serveur ntp sur golem avec l'aide de :

```
ntpdate 192.168.1.51
```

(nous avons lancé plusieurs fois cette commande pour accélérer la synchronisation de toute les horloges) et ensuite nous avons modifié le fichier /etc/ntp.conf pour rajouter le serveur ntp sur les clients :

```
  Fichier /etc/ntp.conf
server 192.168.1.51
```

Nos machines sont donc toutes synchronisée à partir de golem et nous n'avons plus de problème décalage d'horloge sur Ceph.

2.5 Test de disponibilités

Ceph permet de facilement ajouter/supprimer de nouveaux nœud dans le cluster : comme des monitors, des osds ou des mds.

2.5.1 Test de surpression d'un MDS

Vérification du système sur ceph avec le mds planté :

```
ceph@golem:~/cluster-cheese$ceph health
    HEALTH_WARN mds carapuce is laggy
    ceph@golem:~$ceph status
    cluster dde8d8a9-e880-4c81-8bc6-8ede2adbe71
     health HEALTH_WARN mds carapuce is laggy
     monmap e2: 3 mons at {behemot=192.168.1.56:6789/0,golem=
     192.168.1.51:6789/0,rondoudou=192.168.1.29:67891}, election
     epoch 30,quorum 0,1,2 behemot,golem,rondoudou
     mdsmap e9: 2/2/2 up {0=smogogo=up:active,1=porygon=up:active},
```

```
 1 down:standby(laggy or crashed)
 osdmap e44: 3 osds: 3 up, 3 in
  pgmap v188: 192 pgs, 3 pools, 17276 bytes data, 37 objects
        15470 MB used, 15004 MB / 30475 MB avail
             192 active+clean
```

Si le mds est enlevé du cluster, il suffit de le reconnecter au réseau et de relancer le service ceph pour qu'il se réactive. (carapuce a planter, on l'as relancer mais la carte réseau ne c'est pas relancer, on la lancer avec ifup eth1 et le mds remarche) quand le mds à redémarré j'ai refait un ceph status sur golem :

```
ceph@golem:~/cluster-cheese$ceph status
cluster dde8d8a9-e880-4c81-8bc6-8ede2adbe71
  health HEALTH_OK
 monmap e2: 3 mons at {behemot=192.168.1.56:6789/0,golem=
 192.168.1.51:6789/0,rondoudou=192.168.1.29:67891}, election
 epoch 30,quorum 0,1,2 behemot,golem,rondoudou
 mdsmap e9: 2/2/2 up {0=smogogo=up:active,1=porygon=up:active},
  1 up:standby
  osdmap e44: 3 osds: 3 up, 3 in
  pgmap v188: 192 pgs, 3 pools, 17276 bytes data, 37 objects
        15470 MB used, 15004 MB / 30475 MB avail
             192 active+clean
```

Le MDS est de nouveau utilisable, mais il reste en standby car le mds qui l'as remplacé fonctionne bien et contient les métadonnées nécessaire pour permettre de surveiller Ceph.

2.5.2 Test de suppression d'un OSD

Nous avons enlevé 1 OSD de notre cluster, nous vérifions l'état du cluster pour voir s'il fonctionne encore :

```
ceph@golem:~/cluster-cheese$ceph status
    cluster dde8d8a9-e880-4c81-8bc6-8ede2adbe71
       health HEALTH_WARN 192 pgs degraded; 192 pgs stale; 192 pgs stuck
```

```
     stale; 192 pgs stuck unclean; 1/1 in osds are down
     monmap e2: 3 mons at {behemot=192.168.1.56:6789/0,golem=
     192.168.1.51:6789/0,rondoudou=192.168.1.29:67891}, election
     epoch 30,quorum 0,1,2 behemot,golem,rondoudou
     mdsmap e9: 2/2/2 up {0=smogogo=up:active,1=porygon=up:active},
      1 up:standby
      osdmap e44: 3 osds: 2 up, 2 in
      pgmap v188: 192 pgs, 3 pools, 12476 bytes data, 37 objects
            10235 MB used, 10002 MB / 20350 MB avail
                 192 stale+clean+degraded
```

on a donc réactivé l'osd manquant :

```
ceph-deploy osd activate rondoudou:sda3
```

le système ceph est de nouveau "fonctionnel" après ces manipulations.

```
ceph@golem:~/cluster-cheese$ceph status
    cluster dde8d8a9-e880-4c81-8bc6-8ede2adbe71
      health HEALTH_OK
     monmap e2: 3 mons at {behemot=192.168.1.56:6789/0,golem=
     192.168.1.51:6789/0,rondoudou=192.168.1.29:67891}, election
     epoch 30,quorum 0,1,2 behemot,golem,rondoudou
     mdsmap e9: 2/2/2 up {0=smogogo=up:active,1=porygon=up:active},
      1 up:standby
      osdmap e44: 3 osds: 3 up, 3 in
      pgmap v188: 192 pgs, 3 pools, 17276 bytes data, 37 objects
            15470 MB used, 15004 MB / 30475 MB avail
                 192 active+clean
```

2.6 Conclusion

Ceph est un système de fichier distribué hautement évolutif. Il n'est pas encore recommandé de l'utiliser en production : celui-ci est encore en phase de développement. Mais a un bel avenir en vus des fonctionnalité intéressante qu'il offre :
— Il est fonctionnel sur du matériel de base. Coût d'installation peu élevé.

— Il a une haute tolérance au panne.
— Et peut être modifier/réparer sans interruption du système.

3 Lustre

FIGURE 1 – Lustre F.S.

3.1 Introduction

Lustre est un système de fichiers distribué libre(sous licence GPL v2), utilisé pour de très grandes grappes de serveurs. Son nom vient de la combinaison de deux mots, «Linux» et «cluster». Le projet a commencé comme un projet de recherche en 1999 par Peter Braam, en 2007 Sun Microsystems a continué son développement, en 2010 avec l'acquisition de Sun, a commencé à manternir et sortir nouvelles versions de Lustre, en décembre du 2010 Oracle a arrêté le développement du système de fichiers version 2.x, plusieurs nouvelles organisations surgirent à fournir un soutien et de développement dans un modèle de développement communautaire ouvert, comprenant *Whamcloud*, *Open Scalable File Systes, Inc. (OpenSFS)*, *EUROPEAN Open FileSystems (EOFS)* et autres. En 2012 WhamCloud a été rachetée par Intel.

Lustre a la possibilité de accueillir des dizaines de milliers de systèmes client et une grande quantité de données, pétaoctets[9] de stockage et des centaines de gigaoctets(Go) de débit Entrée/Sortie.

3.1.1 Les composants de Lustre

Serveur de Métadonnées *(Metadata Server «MDS»)* : Le serveur MDS rend les métadonnées, stockées dans un ou plusieurs MDT, disponibles pour des

9. Un pétaoctet contient 10^{15} octets

clients Lustre. Chaque MDS gère les noms et répertoires dans le systèmes de fichiers Lustre.

Cible de Métadonnées *(Metadata Target «MDT»)* : Le MDT stocke les métadonnées(tels que les noms de fichiers, les répertoires et la structure des fichiers) sur un MDS. Chaque système de fichiers possède un MDT. Une MDT sur une cible de stockage partagé peut être disponible pour de nombreux MDS, bien que l'on devrait utiliser effectivement. Si une MDS actif échoue, un MDS passif peut servir le MDT et le rendre disponible aux clients. Ceci est appelé *MDS failover*.

Serveur de Stockage d'Objets *(Object Storage Server «OSS»)* : L'OSS fournit un servicce d'Entrée/Sortie de fichiers et *network request handling* pour un ou plusieurs OST locaux. Typiquement, un OSS sert entre 2 et 8 OST, jusqu'à 8 To [10]

Cible de Stockage d'Objets *(Object Storage Target «OST»)* : L'OST stocke les données(des morceaux de fichiers de l'utilisateur) comme des objets de données sur un ou plusieurs OSS. Un seul système de fichiers Lustre peut avoir plusieurs OST, servant chacun un sous-ensamble de fichiers données. Il n'est pas nécessairement une correspondance 1 :1 entre un fichier et un OST. Pour optimiser les performances, un fichiers peut être distribué sur plusieurs OST.

Serveur de Gestion *(Management Server «MGS»)* : Le MGS stocke les informations de configuration pour tous les systèmes de fichiers Lustre dans une grappe de serveurs. Chaque component Lustre contacte le MGS pour fournir information, et les Lustre clients contactent le MGS pour obtenir information.

Lustre clients : Lustre clients sont noeuds éxecutant le logiciel Lustre qui leur permet de monter le système de fichiers Lustre.

Le logiciel client de Lustre est composé d'une interfacce entre le *Linux Virtual File System* et les serveurs Lustre. Chaque cible(composant de Lustre) a une contrapartie client : Client de Métadonnées (MDC), Client de Stockage d'Objets(OSC) et un Client de Gestion(MGC). Un groupe de OSC sont intégrés dans un Volume d'objets logique(*Logical Object Volume «LOV»*). Travaillant en collaboration, les OSC fournissent un accès transparent au système de fichiers.

10. Un téraoctet contient 10^{12}octets.

Les clients qui montent le système de fichiers Lustre, voient un seul, cohérente, espace de noms(*namespace*), synchronisé toujours. Différents clients peuvent écrire aux différentes parties d'un même fichier en même temps, tandis que d'autres clients peuvent lire le fichier.

3.1.2 Principales Caractéristiques de Lustre

Les principales caractéristiques de Lustre comprennent :

Évolutivité : Lustre échelles haut ou le bas par rapport au nombre de postes clients, stockage sur disque et bande passante. Actuellement , Lustre est en cours d'exécution dans des environnements de production avec un maximum de 26 000 postes clients, avec de nombreux grappes d'entre 10,000 et 20,000 clients. Autres installations fournissent Lustre d'espace de stockage en disque agrégé et bande passsante allant jusqu'à 1000 OST fonctionnant sur plus de 450 OSS. Plusieurs systèmes de fichiers Lustre avec une capacité de 1 pétaoctet ou plus (permettant le stockage de jusqu'à 2 milliards de fichiers) ont été en usage depuis 2006.

Performance : Lustre déploiements dans des environnements de production offrent actuellement la performance de jusqu'à 100 Go par seconde. Dans un environnement de test, une performance de 130 Go par seconde a étée soutenue. Lustre seul débit de poste client a été mesurée à 2 Go par seconde (max) et OSS débit de 2,5 Go par seconde (max). Lustre a été exécuté à 240 Go par seconde sur le système de fichiers d'araignée(Spider File System) à Oak Ridge National Laboratories.

Conformité POSIX : Dans un cluster, la conformité POSIX signifie que la plupart des opérations sont atomiques et les clients ne voient jamais des données périmées ou métadonnées .

Haute Disponibilité : Lustre propose des partitions de stockage partagés pour les cibles de l'OSS (OST), et une partition de stockage partagé pour la cible MDS (MDT).

Sécurité : En Lustre, il s'agit d'une option pour les connexions TCP seulement de ports privilégiés. Manipulation des membres du groupe est basée sur le serveur. POSIX listes de contrôle d'accès (ACL) sont pris en charge .

Open Source : Lustre est sous licence GNU GPL v2.

3.1.3 Lustre Network

Lustre Network (LNET) est une API de réseau personnalisé qui fournit l'infrastructure de communication qui gère l'Entrée/Sortie des métadonnées et fichiers de données pour les serveurs et les clients du système de fichiers Lustre.
LNET prend en charge plusieurs types de réseaux couramment utilisés, tels que les réseaux *InfiniBand* et *IP*, et permet la disponibilité simultanée sur plusieurs types de réseaux avec routage entre eux. Accès à la mémoire direct à distance (RDMA) est autorisée lorsqu'elle est soutenue par des réseaux sous-jacentes utilisant le pilote réseau Lustre approprié (*Lustre Network Driver «LND»*). Haute disponibilité et de récupération permettent la récupération transparente avec les serveurs de basculement. Un LND est un pilote enfichable qui fournit un support pour un type de réseau particulier, par exemple *ksocklnd* est le pilote qui implémente le*TCP Scocket LND* qui prend en charge les réseaux TCP. LNDs sont chargés dans la pile de pilotes, avec une LND pour chaque type de réseau en cours d'utilisation.

3.1.4 Réseaux Lustre

Un réseau Lustre est composé de clients et serveurs exécutant le logiciel Lustre. Il n'a pas besoin d' être limité à un sous-réseau de LNET mais peut s'étendre sur plusieurs réseaux, le routage est possible entre les réseaux. D'une manière similaire, un réseau unique peut avoir plusieurs sous-réseaux LNET.

3.1.5 Lustre cluster

À grande échelle, un cluster de système de fichiers Lustre peut inclure des centaines de OSS et des milliers de clients, Figure 2, Plus d'un type de réseau peut être utilisé dans un cluster Lustre. Le stockage partagé entre OSS permet la fonction de basculement*(failover)*.

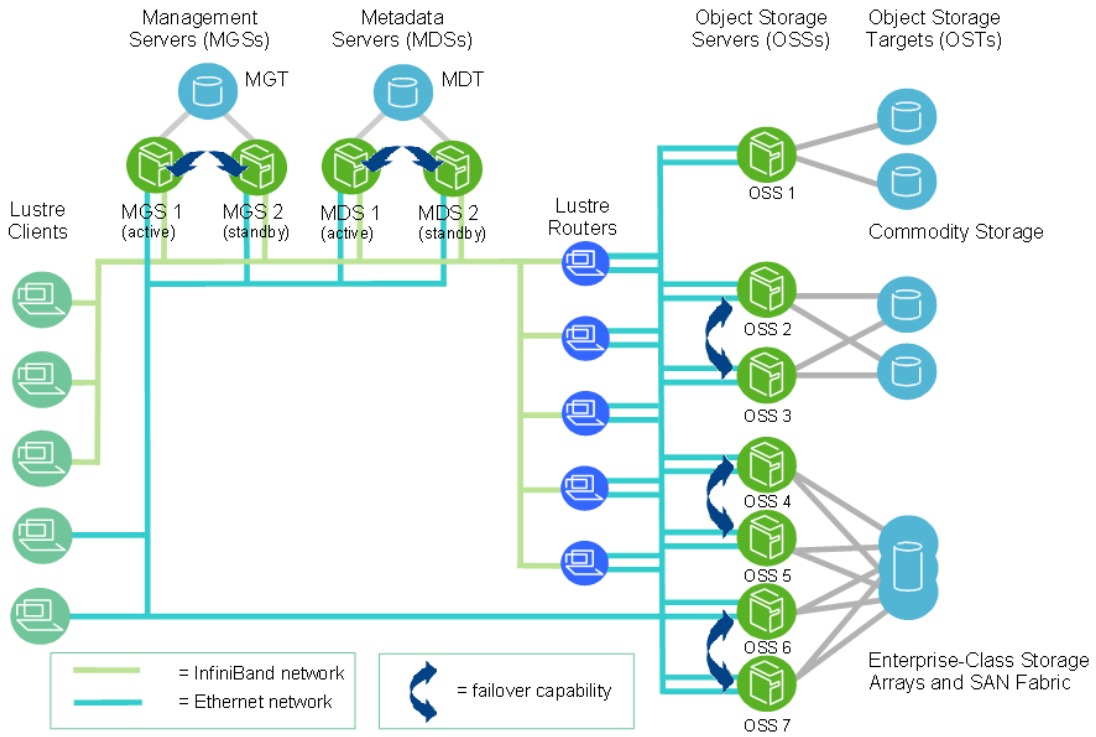

FIGURE 2 – Scaled Cluster

3.1.6 Stockage des données avec Lustre

Dans Lustre version 2.0, les identifiants de fichiers Lustre *(File Identifier «FID»)* ont été introduites pour remplacer les numéros d'inodes UNIX pour identifier des fichiers ou des objets. Un FID est un identificateur de 128 bits qui contient un numéro unique de 64 bits séquence, un 32-bit ID objet *(Object Identifier «OID»)*, et un numéro de version 32 bits. Le numéro de séquence est unique parmi tous les objectifs Lustre dans un système de fichiers (OST et EMD). Ce changement a permis le soutien futur de plusieurs équipes multidisciplinaires (introduits dans Lustre version 2.3 du logiciel) et ZFS [11](introduits dans Lustre logiciel version 2.4).

Informations sur l'endroit où les données de fichier se trouve sur le OST, est stockée comme un attribut étendu appelé disposition EA dans un objet MDT identifié par la FID pour le fichier(figure 3). Si le fichier est un fichier de données (pas un répertoire ou un lien symbolique), l'objet MDT pointe de 1 à N OST objet(s) sur

11. http ://fr.wikipedia.org/wiki/ZFS

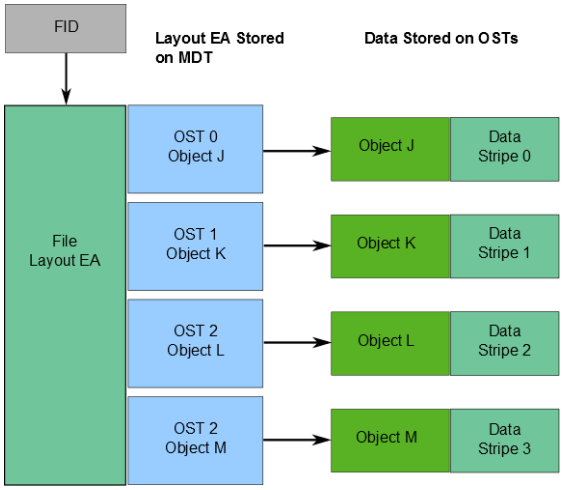

FIGURE 3 – Disposition EA sur MDT pointant vers le fichier de données sur OST

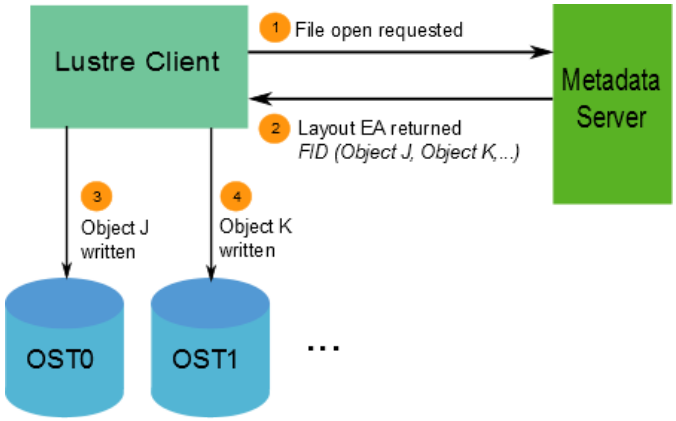

FIGURE 4 – Lustre client demandant des données

le(s) OST(s) qui contiennent les données de fichiers. Si le fichier MDT pointe à un objet, toutes les données de fichier sont stockées dans cet objet. Si le MDT pointe à plus d'un objet, les données du fichier sont réparties sur les objets à l'aide RAID 0, et chaque objet est stocké sur un OST différent.

Quand un client veut lire ou écrire dans un fichier, il récupère tout d'abord la mise en EA de l'objet MDT pour le fichier. Le client utilise ensuite ces informations pour effectuer des Entrées/Sorties sur le fichier, interagir directement avec les nœuds OSS où les objets sont stockés(Figure 4).

3.1.7 Lustre et l'entrelacement

L'un des principaux facteurs menant à la haute performance des systèmes de fichiers Lustre est la capacité de répartir les données sur plusieurs OST dans un mode round-robin. Les utilisateurs peuvent éventuellement configurer pour chaque fichier le nombre de rayures, taille de bande, et OST qui sont utilisés.

L'entrelacement*(Striping)* peut être utilisé pour améliorer les performances lorsque la bande passante globale à un fichier unique excède la bande passante d'un seul OST. La capacité de bande est également utile lorsque l'OST n'a pas assez d'espace pour contenir un fichier entier.
Le Stripping permet segments ou «morceaux» de données dans un fichier pour être stocké sur différents OST, dans le système de fichiers Lustre, une configuration RAID 0 est utilisée dans laquelle des données sont «striped» à travers un certain nombre d'objets. Le nombre d'objets dans un seul fichier est appelé *stripe_ count*.
Chaque objet contient un bloc de données à partir du fichier. Lorsque le bloc de données en cours d'écriture à un objet particulier dépasse la *stripe_ size*, le prochain bloc de données dans le fichier est stocké sur l'objet suivant.
Les valeurs par défaut pour *stripe_ count* et *stripe_ size* sont fixés pour le système de fichiers. La valeur par défaut pour *stripe_ count* est une bande de fichier et la valeur par défaut pour *stripe_ size* est 1Mo. L'utilisateur peut modifier ces valeurs sur une base par répertoire ou par fichier.
La taille maximale de fichier n'est pas limité par la taille d'une cible unique. Dans un système de fichiers Lustre, les fichiers peuvent être réparties sur plusieurs objets (jusqu'en 2000), et chaque objet peut être jusqu'à 16 To en taille avec ldiskfs [12]. Cela conduit à une taille de fichier maximale de 31,25 pétaoctets. (Notez qu'un système de fichiers Lustre peut supporter des fichiers jusqu'à 2^{64} octets selon le stockage de sauvegarde utilisé par OST.)

12. http ://wiki.lustre.org/lid/ulfi/ulfi_ldiskfs.html

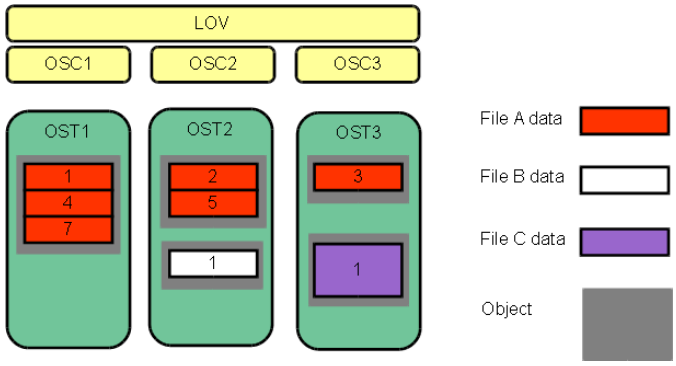

FIGURE 5 – Striping

3.2 Installation

3.2.1 Ubuntu-Lustre

Parallèlement à la première tentative d'installation sur CentOS 6.5, en suivant la recommandation de notre tuteur, nous avons decidé de déployer les client Lustre dans une architecture virtualisée, mais en utilisant le système d'exploitation Ubuntu 13.10 cette fois.

Nous avons travaillé également avec KVM et l'outil graphique virt-manager pour créer une machine Virtuelle à laquelle nous avons attibué 7 cpus, 2 Go de mémoire vive et 20 Go de disque dur, cette machine etait destinée principalement à la compilation du noyau Linux version 3.13.6, qui donne la posibilité d'activer les frivers compatibles avec Lustre.

Le plan était compiler et installer le nouveau nouyau su cette machine et despuis les sources compiler les paquets nécesaires pour l'installation de Lustre sur chaque noeud, cloner la machine 2 fois, réduire ses resources pour les donner 2 cpus à chaqune et 1 Go de mémoire vive, deployer sur les machines clonées et la première machine les clients Lustre.

Nous avons commencé pour la compilation du noyau Linux v13.13.6, la dernière version «stable», nous sommes allés sur le site officiel [13] pour télécharger les paquets sources.

On a obtenu le fichier «linux-3.13.6.tar.xz» dans notre machine virtuelle, pour extraire le contenu nous avons utilisé la commande :

```
rsmrg@LUT:$tar -Jxvf linux-3.13.6.tar.xz
```

Ensuite on a changé le répertoire pour nous déplacer vers celui que nous avons extrait du fichier, aussi on a installé les outils de compilation :

```
rsmrg@LUT:$cd linux-3.16.6 ; sudo apt-get install debconf-utils dpkg-dev
debhelper  build-essential kernel-package libncurses5-dev
```

Nous avons copié la configuration du noyau précédant, après on a exécuté, make oldconfig pour répondre aux nouvelles questions de configurations du noyau :

```
rsmrg@LUT:$sudo cp /boot/config- 'uname -r' .config
rsmrg@LUT:$make oldconfig
```

13. https ://www.kernel.org/

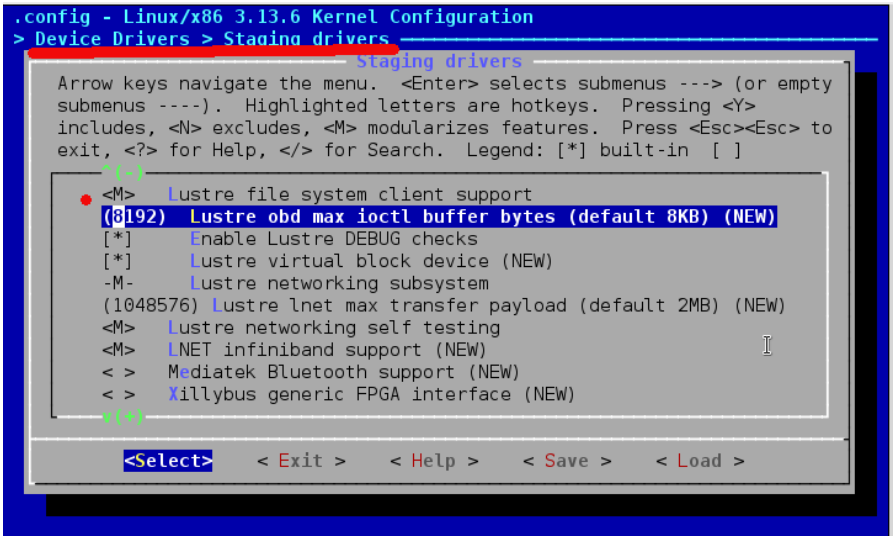

FIGURE 6 – Lustre Drivers

Pour activer les options Lustre on est allé au menu de configuration de noyau dans la partie «Staging drivers» (figure 7).

```
rsmrg@LUT:$make menuconfig
```

Après sauvegarder les manipulations réalisées, il fallait compiler le noyau en utilisant les 7 cpus et générer les paquets «.deb» :

```
rsmrg@LUT:$sudo make -j7 deb-pkg
```

Installation du nouveau kernel depuis les paquets génères :

```
rsmrg@LUT:$sudo dpkg -i ../*.deb
```

Après la compilation du noyau de Linux on a essayé d'installer les paquets Lustre sur le système d'exploitation, mais sans succès, après avoir parlé avec le tuteur du projet, il a recommandé l'arrêt de travail avec Ubuntu pour ne pas perdre plus de temps et de nous consacrer entièrement au déploimeent de Lustre avec logiciels plus anciens, mais supportés, comme c'est le cas du système d'exploitation CentOS 6.3.

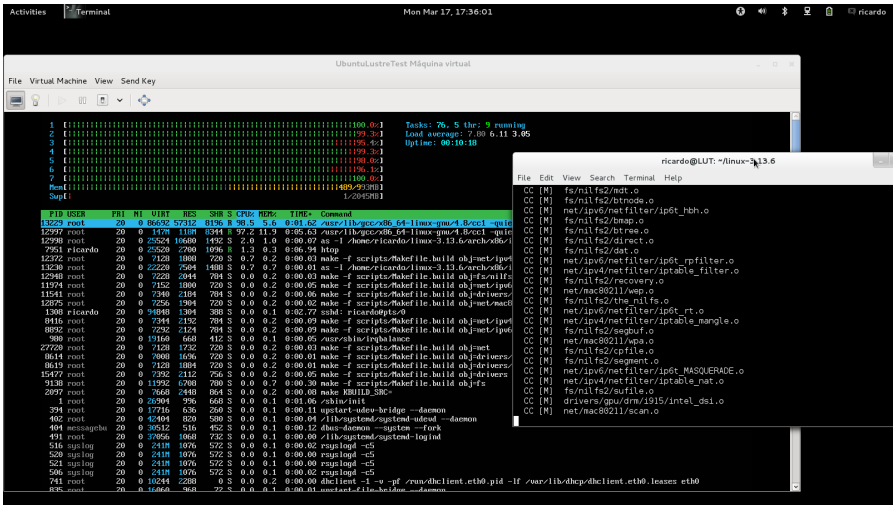

FIGURE 7 – Compilation Linux-13.13.6

3.3 QQLiéeàLustre

3.4 MásPerca

3.5 Lustre

3.6 Faru

4 Annexe

4.1 How to install ceph

4.1.1 Partitionnement

En premier, on a partitionné les machines sur lesquels un OSD sera installé, nous avons créer de nouvelle partition en xfs [14] pour stocker les données du cluster avec le logiciel Gparted. Sur chaque machines nous avons fait une partition de environs 10 Go.

14. est un système de fichiers 64-bit journalisé de haute performance

4.1.2 Configuration du Réseau

Sur chaque machine nous avons modifié le fichier /etc/hosts ajoutant l'alias et l'adresse IP [15] de chaque machine ainsi elles peuvent facilement se connecter entre elles en indiquant ses alias.

```
  Fichier /etc/hosts :
#Ceph cluster
192.168.1.51 golem
192.168.1.29 rondoudou
192.168.1.56 behemot
192.168.1.43 carapuce
192.168.1.2 smogogo
192.168.1.48 porygon
```

Création d'utilisateur ceph

Ceph nécessite un utilisateur spécial pour la configuration et l'administration du cluster à partir de la machine d'administration, nous avons crée l'utilisateur ceph avec les droits d'administrateur du système.

```
root@golem:~/sudo useradd -d /home/ceph -m ceph
  Fichier /etc/sudoers :
ceph ALL = (root) NOPASSWD:ALL
```

Configuration ssh [16]

Pour effectuer la gestion du cluster, les machines doivent se communiquer entre elles avec des tunnels ssh, avec l'utilisateur ceph il faut générer les clés publiques pour s'identifier avec les autres machines.

```
su ceph
ceph@golem:~/ssh-keygen
```

Copier les clés sur tous les autres postes.

```
ceph@golem:~/ssh-copy-id ceph@nomdemachine
```

15. est une famille de protocoles de communication dans le domaine du réseau informatique conçus pour être utilisés par Internet.

16. est à la fois un programme informatique et un protocole de communication sécurisé.

Modifier le fichier /.ssh/config pour se connecter par les tunnels ssh avec l'utilisateur ceph par défaut.

Fichier config :

```
Host golem
User ceph
Host rondoudou
User ceph
Host behemot
User ceph
Host carapuce
User ceph
Host smogogo
User ceph
Host porygon
User ceph
```

Mise à Jour du système :
Afin de prévenir un décalage d'horloge entre les nodes du cluster nous avons installé le serveur NTP sur golem qui nous permet de synchroniser l'horloge de toute les nodes :

```
ceph@golem: sudo apt-get install ntp
ceph@golem: sudo /etc/init.d/ntp restart
```

Nous avons également installé lsb sur toute les machines, il permet de standardiser la structure interne des systèmes d'exploitation basés sur GNU/Linux :

```
ceph@golem:~$ sudo apt-get install lsb
```

Sous Debian, l'installation de Ceph est simple car les développeurs mettent régulièrement les paquets à disposition mais les sources le sont autant. La première étape consiste à rajouter les dépôts de Ceph pour apt-get. Pour ce faire, rajoutez les deux lignes suivantes à la fin de votre

Fichier '/etc/apt/sources-list':

```
deb http://ceph.net/debian/ wheezy main
deb-src http://ceph.net/debian/ wheezy main
```

Seconde étape, mettre a jour apt-get pour la prise en compte de ces nouveaux dépôts :

```
ceph@golem:~$sudo apt-get update
```

Enfin, nous avons utilisez apt-get pour installer les paquets :

```
ceph@golem:~$ apt-get install ceph ceph-deploy
```

On a ensuite créer un répertoire de travail, il est utilisé par l'Admin Node, nous avons créer ce répertoire dans le dossier courant de l'utilisateur ceph :

```
ceph@golem:~$ mkdir cluster-cheese
ceph@golem:~$ cd cluster-cheese
```

A partir de maintenant toute les commandes utilisé pour créer le système de fichier distribué devrons être lancer dans ce dossier, sinon il est possible que l'utilisation d'une de ces commandes créer un deuxième cluster et des problèmes surviennent, si les 2 cluster ont le même nom.

4.1.3 Création cluster et installation des monitors

Nous allons commencer à installer les différents nodes de notre cluster ceph avec la commande :

```
ceph@golem:~/cluster-cheese$ceph-deploy new golem
 rondoudou behemot
```

cette commande permet de déclarer les nodes du cluster.
Ensuite il y a l'installation de ceph sur ces nodes : Pour lancer cette commande nous avons dû rajouter l'option –no-adjust-repos qui permet de passer les toutes les modifications du proxy [17] sur le paquet et ira directement à l'installation du paquet :

```
ceph@golem:~/cluster-cheese$ceph-deploy install --no-adjust-repos
 rondoudou behemot
```

17. est un composant logiciel informatique qui joue le rôle d'intermédiaire en se plaçant entre deux autres pour faciliter ou surveiller leurs échanges.

(ceph est déjà installé sur golem car c'est aussi notre Admin Node.)
Nous passons maintenant à l'installation des monitors, pour cela nous allons créer 1 monitors sur chacun des nodes présents (golem, rondoudou, behemot) avec la commande :

```
ceph@golem:~/cluster-cheese$ceph-deploy mon create-initial
```

cette commande génère de nouveau fichier dans notre répertoire cluster-cheese :

```
ceph@golem:~/cluster-cheese$ls
```

```
ceph.conf ceph.log ceph.mon.keyring
```

si on regarde ceph.conf :

```
  Fichier ceph.conf
[global]
fsid = 10c95f01-2dd2-4863-affa-60c4eafcd8d2
mon_ initial_members = golem, rondoudou, behemot
mon_host = 192.168.1.51, 192.168.1.29, 192.168.1.56
auth cluster required = cephx
auth service required = cephx
auth client required = cephx
osd_journal_size = 1024
```

On vois que nos 3 monitors ont été ajouté dans la configuration du cluster.

4.1.4 Installation des Osds

Pour l'installation des OSDs nous avons créé une partition de type xfs sur les machines golem, rondoudou et behemot, nous avons utilisé la commande :

```
ceph@golem:~/cluster-cheese$ceph-deploy disk list <nomdemachine>
```

Qui permet de voir les partitions et leur système de partitionnement sur les différents nodes.
Ensuite depuis golem on a formaté ces partitions avec les commandes :

```
ceph@golem:~/cluster-cheese$ceph-deploy disk zap golem:sda3
ceph@golem:~/cluster-cheese$ceph-deploy disk zap rondoudou:sda3
ceph@golem:~/cluster-cheese$ceph-deploy disk zap behemot:sda6
```

Après avoir formaté les partitions nous avons préparé et activé les Osds :

```
ceph@golem:~/cluster-cheese$ceph-deploy osd prepare golem:sda3
ceph@golem:~/cluster-cheese$ceph-deploy osd activate golem:sda3
ceph@golem:~/cluster-cheese$ceph-deploy osd prepare rondoudou:sda3
ceph@golem:~/cluster-cheese$ceph-deploy osd activate rondoudou:sda3
ceph@golem:~/cluster-cheese$ceph-deploy osd prepare behemot:sda3
ceph@golem:~/cluster-cheese$ceph-deploy osd activate behemot:sda3
```

(Pour certaine de ces commandes nous avons du rajouter l'option –overwrite-conf, pour modifier la configuration de ceph sur les nodes, ex : ceph-deploy –overwrite-conf osd prepare golem :sda3)
Après avoir installé les OSDs nous pouvons déjà regardé si ceph est installé correctement avec la commande :

```
ceph@golem:~/cluster-cheese$ceph status
    cluster dde8d8a9-e880-4c81-8bc6-8ede2adbe71
      health HEALTH_OK
     monmap e2: 3 mons at {behemot=192.168.1.56:6789/0,golem=
     192.168.1.51:6789/0,rondoudou=192.168.1.29:67891}, election
     epoch 30,quorum 0,1,2 behemot,golem,rondoudou
      osdmap e44: 3 osds: 3 up, 3 in
      pgmap v188: 192 pgs, 3 pools, 17276 bytes data, 37 objects
            15470 MB used, 15004 MB / 30475 MB avail
                 192 active+clean
```

4.1.5 Installation des MDS

Nous arrivons à la dernière partie de l'installation où nous allons installé les mds avec la commandes :

```
ceph@golem:~/cluster-cheese$ceph-deploy mds create carapuce smogogo
 porygon
```

et nous voulons 2 mds d'actif, pour l'instant il y en a un d'actif de base on utilise la commande suivante :

```
ceph@golem:~/cluster-cheese$ceph mds set_max_mds 2
```

Ensuite nous avons fait un ceph status pour voir si le système ceph était bien installé :

```
ceph@golem:~/cluster-cheese$ceph status
    cluster dde8d8a9-e880-4c81-8bc6-8ede2adbe71
      health HEALTH_OK
     monmap e2: 3 mons at {behemot=192.168.1.56:6789/0,golem=
     192.168.1.51:6789/0,rondoudou=192.168.1.29:67891}, election
     epoch 30,quorum 0,1,2 behemot,golem,rondoudou
     mdsmap e9: 2/2/2 up {0=smogogo=up:active,1=porygon=up:active},
      1 up:standby
      osdmap e44: 3 osds: 3 up, 3 in
      pgmap v188: 192 pgs, 3 pools, 17276 bytes data, 37 objects
            15470 MB used, 15004 MB / 30475 MB avail
                 192 active+clean
```

Nous avons un système ceph fonctionnel, qui nous permet de réaliser des test de disponibilité et de performance.
Avec la commande :

```
ceph@golem:~/cluster-cheese$ceph osd lspools
```

Nous pouvons voir qu'il y a 3 "piscines" [18] sur notre cluster ceph, nous allons utilisé la pool rbd [19] (*Ceph's RADOS Block Devices*), pour créer un file system est pouvoir stocké des données sur Ceph.

4.1.6 Installation du client

Pour l'installation du client, on a d'abord ajouté le client dans le réseau ceph grâce à ssh, on a créé un utilisateur ceph et on a installé ceph sur la machine-client à partir de golem (Admin Node) :

18. Les piscines sont des partitions logiques pour stocker des objets.
19. fournit un accès à la couche Rados à travers un module noyaux

```
ceph@ronflex:~/$ceph-deploy install ceph-client
```

On a ensuite copier la configuration de notre cluster de golem vers le client avec la commande :

```
ceph@ronflex:~/$ceph-deploy admin ceph-client
```

cette commande à copier le ceph.conf et le ceph.client.admin.keyring dans le dossier /etc/ceph sur le ceph-client.
Nous allons créer dans notre piscine rbd un nouveau périphérique bloc [20] qui nous permettra de stocké des données, nous créons ici un bloc de 10Go dans la piscine rbd

```
ceph@ronflex:~/$prbd create foo --size 10096 --pool rbd
```

Nous pouvons voir avec :

```
ceph@ronflex:~/$rbd ls rbd
ceph@ronflex:~/$rbd --image foo -p rbd info
```

que notre bloc a bien été créer.
Et maintenant nous lançons la commande :

```
ceph@ronflex:~/$rbd map foo -pool rbd
```

pour ajouter à la map de rbd le nouveau bloc, on peut le voir avec la commande rbd showmapped, on a ensuite mit un système de fichier sur le bloc :

```
ceph@ronflex:~/$mkfs.ext4 -m0 /dev/rbd/rbd/foo
```

et nous avons finalement monter le bloc sur le système client avec :

```
ceph@ronflex:~/$mkdir /mnt/rbd_foo
ceph@ronflex:~/$mount /dev/rbd/rbd/foo /mnt/rbd_foo
```

A partir d'ici nous pouvons écrite/lire des fichiers sur notre système de fichier distribué Ceph à partir d'un client et réaliser les test de performances.

20. organise les données de telle façon qu'il en résulte une amélioration de la flexibilité, des performances et de la fiabilité.

5 Conclusion

http ://doc.ubuntu-fr.org/tutoriel/compiler_linux http ://wiki.lustre.org/manual/LustreManual18
http ://wiki.lustre.org/index.php/Debian_Install https ://wiki.debian.org/Lustre
http ://downloads.whamcloud.com/public/lustre/lustre-2.5.1/el6/client/RPMS/x86_64/
http ://ocubom.wordpress.com/2008/05/21/estructuras-basicas-de-latex/ http ://en.wikipedia.org/