Chapitre 1

Mise en place d'un cluster GFS

1.1 Principe

1.1.1 Principe d'un cluster

Cluster est le terme utilisé en Anglais pour designer une **grappe de serveurs**, soit un ensemble de serveurs ou d'ordinateurs indépdendants (apellés noeuds, **node** en Anglais) partageant des ressources afin d'optimiser le traitement des taches et augmenter la disponibilité des ressources.
Une grappe de serveurs est donc un groupe de serveurs indépendants fonctionnant comme un seul et même système. Un client dialogue avec un cluster comme s'il s'agissait d'une machine unique. Leur utilisation est de plus en plus importante dans le monde

professionnel, où les besoins en calculs à haute performance sont toujours croissants tout en minimisant l'impact d'une panne de serveur. En effet disposant de ressources communes, une architecture Cluster permet d'effectuer régulièrement des procédures de redondances, afin de vérifier l'état de ses noeuds.
Lors de la défaillance d'un serveur, le logiciel de clustering réagit en transférant les tâches exécutées sur le système défaillant vers les autres serveurs de la grappe.
Le même principe est mis en oeuvre pour le partage des tâches d'un serveur surchargé à un autre. Les clusters consituent également une solution économique interessante, puis-

qu'ils permettent de minimiser l'investissement dans du matériel récent ; la répartition équilibrée des tâches permet une décharge de la mémoire à moindre coût.

(schéma)

1.1.2 Principe de GFS

Global File System (GFS), est un système de fichiers, sous licence publique GNU, pour UNIX/Linux à haute dispnibilité, c'est à dire qu'il assure une disponibilité des services 99% du temps par an.

Le GFS est utilisé pour synchroniser l'accès aux ressources partagées.
Etant symétrique, il défini les mêmes regles en lecture/écriture pour tous les noeuds du cluster. Sa seconde particularité est le support la journalisation et la récupération de données suite à des défaillances de clients.
GFS utilise en général la *Fibre Channel* pour établire une communication à grande vitesse sur des périphériques *SCSI*.
Bien que performante, une telle installation nécessite un investissement financier important. heureusement il est possible de remédier à des solutions logicielles et libres, qui permettent le transport de protocoles à haut niveau (tels que SCSI) sur une installation Ethernet standard.

(schéma)

1.2 Objectifs du projet

Il existe plusieurs systèmes de fichiers parmis lesquels NBD, NFS, NAS qui sont couramment utilisés pour le partage de données entre serveurs.
L'objectif de ce projet est de démontrer les capacités de GFS face aux autres protocoles de partages, à savoir :

- De meilleures performances en lecture et écriture
- Une installation sécurisée
- Une haute disponibilité des services

Pour cela, ce projet se déroule en deux parties :

- Implémentation de Iscsi, pour le partage de la ressource
- Configuration du Cluster et installation de GFS

1.3 Implémentation du protocole iSCSI

1.3.1 Principe de iSCSI

iSCSI est un protocole de partage de ressources concu par IBM au milieu des années 90 et standardisé par l'IETF en Avril 2004.
Il a été concu afin de transporter des commandes *SCSI* sur un réseau de type *Ethernet*.
Comme grand nombre d'autres protocoles de partages tels que *Samba, NFS,...* iSCSI fonctionne selon une architecture Client/Serveur.
Il permet l'export et le montage d'une ressource (volume physique ou logique) sur le réseau.
Ainsi il sera possible d'effectuer des opérations de lecture/écriture sur le média commun comme si il s'agissait d'une partition locale.
Ceci offre de nombreux avantages, comme le montage automatique de la partition au démarrage de la machine, mais aussi une plus grande souplesse dans l'accés aux données
Le protocole iSCSI se divise en deux parties :

– La cible : exporte la ressource (partition, fichier, ...)
– L'initiateur : importe la ressource pour la monter localement
Durant la mise en oeuvre de ce service, l'architecture suivante à été réalisée :

(schéma de iSCSI)
On a choisi d'exporter un volume logique de 10 Gigaoctets sur la station **target**, ne disposant que d'une seule partition.
Le volume logique "export" est a été créé à l'aide de l'outil **Logical-Volume-Manager** *(Voir Annexes. Création d'un volume logique)*, son emplacement est */dev/VolGroup00/export*.

1.3.2 Configuration de iSCSI

Configuration de la cible

Ne disposant pas d'un véritable périphérique *SCSI* compatible, on se propose de réaliser ce projet à l'aide d'une solution logicielle.
L'application **iSCSi-target**, issue de la communauté *Open-source*, disponible à l'adresse *http ://sourceforge.net/projects/iscsitarget/* permet l'export de disques en iSCSI sur un réseau *Ethernet*.
Ce logiciel contient un *démon* **ietd** (IET = iSCSI Enterprise Target) ainsi qu'une commande de contrôle **ietadm** permettant la découverte et la connexion des initiateurs à la cible.

Une fois installée *(Voir Annexes. Installation de iSCSI-target*, il faut configurer l'application **iSCSI-target**, et définir dans le fichier **/etc/ietd.conf** le nom iSCSI de la cible, et la ressource à exporter. On entre donc les lignes suivantes dans ce fichier :

```
Target iqn.2008-03.fr.polytechnique.lix:storage.disk2

Lun 0 Path=/dev/VolGroup00/export,Type=blockio
```

La première ligne correspond au nom de la cible au format iSCSI.
Ce nom commence toujours par "iqn." suivit d'une date de type "yyyy-mm" (ici 2008-03) et du nom de domaine réseau, inscris à l'envers, sur lequel se trouve la machine (dans notre cas *lix.polytechnique.fr*).
Il faut également préciser dans le nom de la cible, que l'on exporte des ressources, avec la ligne *storage.disk2* toujours précédée des " :" (*disk2* étant un nom arbitraire, il est possible de le remplacer par un nom quelconque). La seconde commande nous permet

de préciser le chemin d'accès à la ressource qui sera exportée, avec le paramètre **Path**.

On déclare aussi le type de ressource après **Type**= :
- blockio : disque amovible, partition ou volume logique
- fileio : fichier

On prend soin de laisser le reste du fichier en commentaires (lignes précédées du caractère "#").
La cible est désormais configurée pour exporter le volume "export".
Il reste cependant à définir les règles de permissions pour l'accès des initiateurs.
On stipule dans le fichier **/etc/initiators.allow** que les initiateurs d'adresses IP *192.168.113.200, 192.168.113.201, et 192.168.113.204* sont autorisés avec la commande suivante :

```
iqn.2008-03.fr.polytechnique.lix:storage.disk2 192.168.113.200,
192.168.113.201, 192.168.113.204
```

On observe que cette ligne est simplement constituée du nom iSCSI de la cible suivi des adresses IP des initiateurs.
La politique étant de tout interdire par défaut, il n'est pas nécessaire de configurer le fichier **/etc/initiators.deny** qui gère les interdictions (la syntaxe de ce fichier est identique à **initiators.allow**).
L'accès au volume "export" sera donc strictement réservé à nos trois initiateurs.
La station **target** étant correctement configurée, on peut dès à présent activer le partage

des ressources en éxécutant le service iscsi-target via les commandes :

```
service iscsi-target start
```

Si tout s'est déroulé correctement le logiciel répond :

```
[root@virtbarracuda1 /]# service iscsi-target start
Starting iSCSI target service:                             [  OK  ]
```

On peut afficher la liste des ressources partagées par iSCSI grâce à la commande *cat /proc/net/iet/volume* :

```
[root@virtbarracuda1 ~]# cat /proc/net/iet/volume
tid:1 name:iqn.2008-03.fr.polytechnique.lix:storage.disk2
        lun:0 state:0 iotype:blockio iomode:wt path:/dev/VolGroup00/export
```

La ressource est à présent accessible sur le réseau, il faut maintenant paramétrer les initiateurs.

Configuration des initiateurs

Sur les stations initiateurs, respectivement **virtlavardin1, virtcypres1, virtcypres3** il faut installer le logiciel client **Open-iscsi**. Ce dernier est disponible sous forme de *package* **iscsi-initiator-utils** pour les distributions Red Hat Enterprise, Fedora Core, et CentOS. Une fois installé, on dispose dès à présent des composants suivants :

Démons, fichiers, répertoires	Description
iscsid	Démon permettant la découverte et la connexion
iscsiadm	Commande d'administration pour découvrir et se conn
iscsi-iname	Commande utilisée pour générer un nom un
/etc/iscsi/iscsi.conf	Fichier de configuration principal
/etc/iscsi/initiatorname.iscsi	Fichier contenant le nom de l'initiate
/var/lib/iscsi/	Base de données initiateur comprenant les informatio

Au même titre que la cible, une station initiateur doit posséder un nom iSCSI unique pour être identifiée.
Ce nom se situe dans le fichier **/etc/iscsi/initiatorname.iscsi**, sa syntaxe est de la forme :

```
InitiatorName=iqn.2008-03.fr.polytechnique.lix:client.nom_iscsi
```

Avec "nom_iscsi" specifique à chaque initiateur, qui peut être généré avec la commande **iscsi-iname** :

```
[root@virtlavardin1 ~]# iscsi-iname
iqn.2008-03.fr.polytechnique.lix:client.a4a623d6036
```

Le paramètre "InitiatorName" en début de ligne, indique au logiciel le nom de l'initiateur, au même titre que "Target" était nécessaire pour déclarer le nom de la cible.
Il est important que la partie *iqn.2008-03.fr.polytechnique.lix* soit commune dans le nom de l'initiateur et celui de la cible, afin que l'identification des initiateurs par la cible puisse avoir lieu. On défini alors les noms suivants pour les initiateurs :

- virtlavardin1 : InitiatorName=iqn.2008-03.fr.polytechnique.lix :client.a4a623d6036

- virtcypres1 : InitiatorName=iqn.2008-03.fr.polytechnique.lix :client.74b57f5c4f70
- virtcypres3 : InitiatorName=iqn.2008-03.fr.polytechnique.lix :client.a4a623d6036

Les initiateurs sont désormais prets à procéder à la découverte de la cible puis à la connexion, il faut alors lancer le service iscsi avec la commande *service iscsi start* :

```
[root@virtlavardin1 ~]# service iscsi start
iscsid dead but pid file exists
Turning off network shutdown. Starting iSCSI daemon:       [  OK  ]
                                                           [  OK  ]
iscsiadm: No records found
```

Au premier démarrage l'application rapporte le message "No records found" qui nous indique qu'aucune cible n'est répertoriée dans la base de données de l'initiateur.
Ceci est du au fait que nous n'avons pas encore défini aux initiateurs qui était la cible.
Cette procédure est la "découverte de la cible".

1.3.3 Découverte de la cible

Les services **iscsi** et **iscsi-target** sont actifs, ce qui siginifie que les initiateurs et la cible sont prets à communiquer.
L'initiateur doit récupérer des informations sur la cible avant de s'y connecter (nom, type de périphérique exporté, etc...).
La découverte de la cible s'éxécute avec la commande **iscsiadm**.
Sa syntaxe générale est la suivante :
iscsiadm -m discovery -t st -p a.b.c.d

Avec **a.b.c.d** adresse IP de la cible.

Dans notre cas on souhaite "décrouvrir" la station **target** d'adresse IP 192.168.113.203, on entre alors la commande *iscsiadm -m discovery -t st -p 192.168.113.203*.
Lorsque la découverte est terminée, le logiciel retourne le nom et l'adresse IP de la cible, ainsi que le numéro de port (3260) sur lequel a eu lieu la communication :

```
[root@virtlavardin1 ~]# iscsiadm -m discovery -t st -p 192.168.113.203
192.168.113.203:3260,1 iqn.2008-03.fr.polytechnique.lix:storage.disk2
```

L'initiateur peut désormais se connecter sur la cible.

1.3.4 Connexion sur la cible

De manière similaire, les procédures de découverte et de connexion sur la cible utilisent toutes deux la commande **iscsiadm**.
Néanmoins, leur syntaxe varie quelque peu, ainsi pour se connecter sur la cible, il faudra entrer la commande suivante sur un terminal initiateur :

```
iscsiadm -m node -T iqn.2008-03.fr.polytechnique.lix:storage.disk2
-p 192.168.113.203 -l
```

On a ici renseigné le nom iSCSI et l'adresse IP de la sation cible (**Target**), si la connexion s'est déroulée correctement, la requête nous adresse le rapport suivant :

```
Login session [iface: default, target:
iqn.2008-03.fr.polytechnique.lix:storage.disk2, portal: 129.104.11.98,3260]
```

La connexion est alors établie entre les initiateurs et la cible, et ceux ont désormais accès au volume "export" partagé par **Target**.
Il est possible, dans un premier temps, de vérifier les sessions établies sur la cible, via la commande *cat /proc/net/iet/session* :

```
tid:1 name:iqn.2008-03.fr.polytechnique.lix:storage.disk2
       sid:69805794312126976 initiator:iqn.2008-03.fr.polytechnique.lix:client.15b567403d95
               cid:0 ip:192.168.113.204 state:active hd:none dd:none
       sid:69524319268307456 initiator:iqn.2008-03.fr.polytechnique.lix:client.74b57f5c4f70
               cid:0 ip:192.168.113.201 state:active hd:none dd:none
       sid:64739244664226304 initiator:iqn.2008-03.fr.polytechnique.lix:client.a4a623d6036
               cid:0 ip:192.168.113.200 state:active hd:none dd:none
```

On retrouve bien ici, les noms des trois initiateurs.
Il est également possible de visualiser les ressources SCSI disponibles sur chaque initiateur.
Ceci se fait à l'aide de la commande *cat /proc/scsi/scsi* :

```
  [root@virtlavardin1 ~]# cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: IET      Model: VIRTUAL-DISK     Rev: 0
  Type:   Direct-Access                    ANSI SCSI revision: 04
```

On observe qu'un disque virtuel SCSI a bien éte reconnu. On peut obtenir des informations supplémentaires sur cette ressource, en entrant la commande **fdisk -l** sur le terminal d'un initiateur :

```
Disk /dev/xvda: 10.7 GB, 10737418240 bytes
255 heads, 63 sectors/track, 1305 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

    Device Boot      Start         End      Blocks   Id  System
/dev/xvda1   *           1          13      104391   83  Linux
/dev/xvda2              14        1305    10377990   8e  Linux LVM

Disk /dev/sda: 10.7 GB, 10737418240 bytes
64 heads, 32 sectors/track, 10240 cylinders
Units = cylinders of 2048 * 512 = 1048576 bytes

Disk /dev/sda doesn't contain a valid partition table
```

La nouvelle partition située sur */dev/sda/* a bien une taille de 10 Gigabytes.
On obtient également des informations sur le nombre de **cylindres** et de **secteurs** qui composent cette partition.
Le message Disk /dev/sda doesn't contain a valid partition table nous indique que cette partition n'a pas de table de partitionnement valide, du fait qu'elle n'a pas encore été initialisée.
Il faut donc allouer ce nouvel espace libre dans l'arborescence des partitions de Linux avec un **volume groupe** "VolGroup01" de 10 Gigabytes puis un **volume logique** "export" de même taille *(Voir Annexes. Initialisation d'une partiton)* . Une fois cette opération terminée, le volume "export" est disponible via le chemin */dev/VolGroup01/export*.
On peut visualiser les volumes groupes disponibles sur la station avec la commande **pvscan** :

```
[root@virtlavardin1 ~]# pvscan
  PV /dev/xvda2   VG VolGroup00   lvm2 [9.88 GB / 1.03 GB free]
  PV /dev/sda1    VG VolGroup01   lvm2 [10.00 GB / 0    free]
  Total: 2 [19.87 GB] / in use: 2 [19.87 GB] / in no VG: 0 [0   ]
```

On remarque qu'un nouveau **volume groupe** de 10 Gigabytes a éte créé.
Par ailleurs il est possible d'obtenir à nouveau des informations concernant les disques en entrant une nouvelle fois la commande **fdisk -l** :

```
[root@virtlavardin1 ~]# fdisk -l

Disk /dev/xvda: 10.7 GB, 10737418240 bytes
255 heads, 63 sectors/track, 1305 cylinders
```

```
Units = cylinders of 16065 * 512 = 8225280 bytes

    Device Boot      Start         End      Blocks   Id  System
/dev/xvda1   *           1          13      104391   83  Linux
/dev/xvda2              14        1305    10377990   8e  Linux LVM

Disk /dev/sda: 10.7 GB, 10737418240 bytes
64 heads, 32 sectors/track, 10240 cylinders
Units = cylinders of 2048 * 512 = 1048576 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1               1       10240    10485759+  8e  Linux LVM
```

Cette fois-ci le rapport ne précise plus l'absence de table de partition valide.
La ressource étant correctement reconnue sur chaque initiateur, on peut désormais effectuer des opérations supplémentaires, notemment pour renforcer la sécurité.

1.3.5 Mise en place d'une authentification

On souhaite sécuriser l'accès au volume "export" à l'aide d'une authentification par nom d'utilisateur et mot de passe.
iSCSI propose deux types d'authentification :

- IN : La cible s'authentifie auprès de l'initiateur
- OUT : L'initiateur s'authentifie auprès de la cible

Il est possible de configurer une authentification IN et/ou OUT aussi bien pour la découverte que pour la connexion.
On choisit de sécuriser la connexion avec une authentification OUT, afin de minimiser les manipulations sur la cible.
Il faut renseigner le nom d'utilisateur et le mot de passe dans le fichier de configuration **/etc/ietd.conf** dans la ligne **IncomingUser**.
On choisit *lix* comme nom de session et *iscsi* comme mot de passe :

```
 IncomingUser lix iscsi
```

Il ne reste plus qu'à redémarrer le service **iscsi-target** pour que ces modifiations soient prises en compte.
Pour vérifier si l'accès aux ressources est à présent sécurisé, on effectue une nouvelle découverte et connexion sur la cible.
On relance le service **iscsi** sur chaque initiateur :

```
[root@virtlavardin1 ~]# service iscsi start
Turning off network shutdown. Starting iSCSI daemon:       [  OK  ]
                                                           [  OK  ]
Setting up iSCSI targets: Login session [iface: default, target:
```

```
iqn.2008-03.fr.polytechnique.lix:storage.disk2, portal: 129.104.11.98,3260]
                                                          [  FAILLED ]
```

La présence de *[FAILLED]* à la fin du rapport indique que la connexion sur la cible a échouée, il n'est donc plus possible de s'y connecter.
Il faut maintenant renseigner le nom d'utilisateur et le mot de passe pour les initiateurs, ceux-ci doivent être stockés avec l'ensemble des informations de la cible dans le fichier **/var/lib/iscsi/nodes/nom de la cible/adresse IP de la cible/default** (dans notre cas */var/lib/iscsi/nodes/iqn.2008-03.fr.polytechnique.lix : storage.disk2/192.168.113.203,3260,1/default*).
Dans ce fichier on ajoute les lignes suivantes :

```
node.session.auth.authmethod = CHAP
node.session.authusermane = lix
node.session.authpassword = iscsi
```

On observe avec la première ligne, qu'on utilise **CHAP** comme protocole d'authentification.
Il serait aussi possible d'utiliser le protocole **PAP**, mais ce dernier contrairement à **CHAP** ne permet pas un cryptage des paramètres d'authetification, il est donc moins fiable.
On peut à présent, tenter une nouvelle connexion sur la cible :

```
[root@virtlavardin1 ~]# service iscsi start
Turning off network shutdown. Starting iSCSI daemon:       [  OK  ]
                                                           [  OK  ]
Setting up iSCSI targets: Login session [iface: default, target:
iqn.2008-03.fr.polytechnique.lix:storage.disk2, portal: 129.104.11.98,3260]
                                                           [  OK  ]
```

L'authentification s'est bien établie, et la ressource "export" est désormais sécurisée et prête à être utilisée dans le Cluster.

1.4 Configuration du cluster

1.4.1 Schéma et packages

Nous disposons d'un volume partagé avec iSCSI sur trois stations.
L'objectif est de paramétrer un cluster pour implémenter un futur système de fichiers GFS.
La configuration du cluster nécessite les *packages* suivants :

- cman

- clvmd
- system-config-cluster

Le package **cman** intègre le *démon* **cmand**, qui permet la mise en oeuvre du cluster.
Il nécessite le fichier de configuration **/etc/cluster/cluster.conf** pour fonctionner.
Ce fichier doit être identique sur chaque machine, une station disposant d'un fichier différent ne sera pas membre du cluster.
Clvmd est un **démon** qui permet de mettre à jour et de distribuer les **metadatas LVM** au sein du cluster.
Pour configurer le cluster, nous utilisons le logiciel **system-config-cluster**, qui paramètre graphiquement le fichier **/etc/cluster/cluster.conf**.

Ainsi l'architecture suivante a été réalisée :

(Schéma de larchitetcure)

1.4.2 Définition du nom

Lorsqu'on exécute **system-config-cluster** la première fois, le logiciel indique qu'il ne trouve pas le fichier **cluster.conf**, il faut donc créer une nouvelle configuration.
On sélectionne l'option "Create a new configuration" qui va nous afficher la fenêtre ci-dessous :

Dans cette fenêtre on configure le nom du cluster, l'adresse multicast du cluster ainsi que les options concernant l'utilisation d'un **quorum-disk**.
On défini **gfslix** comme nom pour le cluster.
L'adresse multicast permet au logiciel de transférer les informations contenues dans le fichier **cluster.conf** sur toutes les stations présentes sur le brin réseau.
Par défaut, **cman** utilise l'adresse multicast du brin réseau, il n'ets donc pas nécessaire d'indiquer une adresse multicast car le brin sur lequel se situe le cluster (192.168.113.0) comporte uniquement des stations membres du cluster.
Un **quorum-disk** est généralement utilisé pour éviter les conflits de priorité entre les nodes. Ceux-ci peuvent survenir par exemple, lors d'une reprise de service, si une autre node est défaillante.
Nous n'avons pour l'instant, effectué aucun test de simulation de pannes, il n'est donc pas nécessaire de paramétrer cette option.
Il faut définir ensuite les stations membres du cluster.

1.4.3 Ajout d'une node

Le cluster va être équipé de 3 stations membres ou **nodes**.
La configuration d'un membre repose entièrement sur la resolution nom réseau/adresse IP.
Avant de configurer un nouveau membre, il faut s'assurer que celui-ci soit joignable par son nom réseau, dans les paramètres du serveur **DNS** ou bien dans le fichier **/etc/hosts** de chaque node.

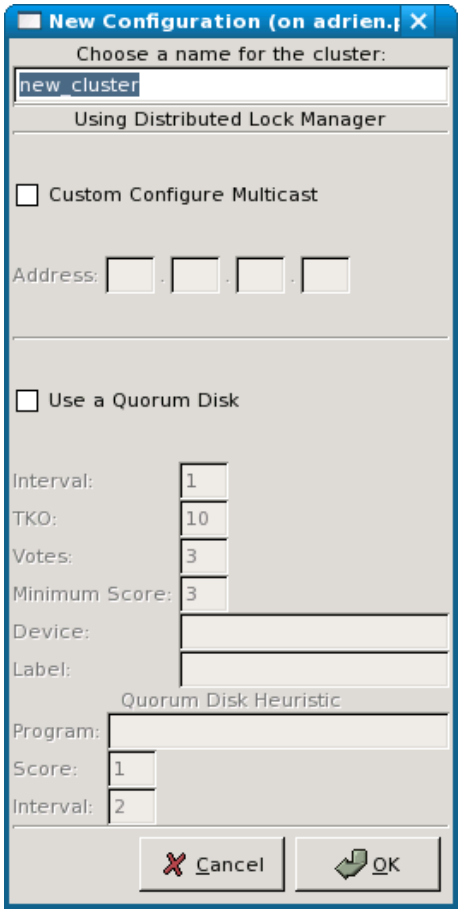

FIG. 1.1: Nouvelle configuration

Pour ajouter un nouveau membre, il suffit de cliquer sur l'onglet **Cluster Nodes** puis sur "Add a Cluster Node".
Une fenêtre apparaît alors, ou l'on indique le nom réseau de la node :

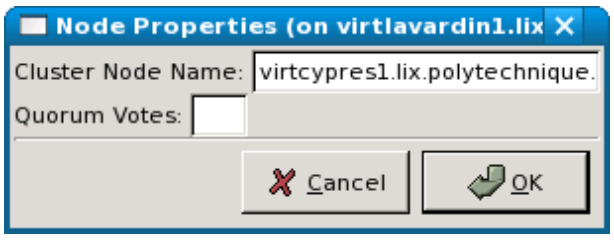

FIG. 1.2: Nouvelle node

Les stations sont ensuites affichées dans l'onglet "Cluster Management" du logiciel :

Le logiciel signale quelles sont les nodes actives ou non avec les paramètres respectifs *Member* et *Not a member*. Les nodes s'interrogent régulièrement entre elles, afin de

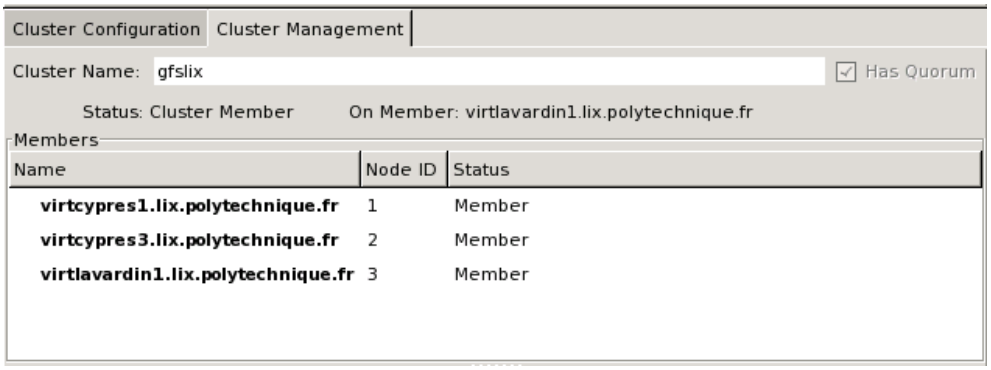

FIG. 1.3: Nodes configurées

vérifier l'état et la disponibilité de chacune.
Pour que cette communication puisse avoir lieu, il faut configurer un domaine **Failover**.

1.4.4 Ajout d'un domaine Failover

Le domaine **Failover** est juste défini par son nom dans le fichier **cluster.conf**.
Pour le paramétrer il faut cliquer sur l'onglet "Failover Domains" présent dans "Managed Ressources", puis sur "Create a Failover Domain".
La fenêtre de configuration apparaît alors, ou l'on peut entrer le nom du domaine :

FIG. 1.4: Nouveau domaine Failover

Il faut ensuite définir les stations membres du domaine parmis les nodes, mais également leur ordre de priorité pour la reprise de service en cas de défaillance :

Il est préférable de définir un ordre de priorité entre les nodes, pour éviter les conflits dus aux reprises de service.

Disposant de trois nodes, un unique domaine **Failover** est suffisant.
Il pourrait être interessant d'en définir plusieurs, pour un cluster possédant d'avantage de membres, ou le partage des services serait réparti en plusieurs groupes de serveurs.
Lorsque les nodes sont membres d'un domaine **Failover** elles peuvent avoir accès aux mêmes services sur la même ressource.

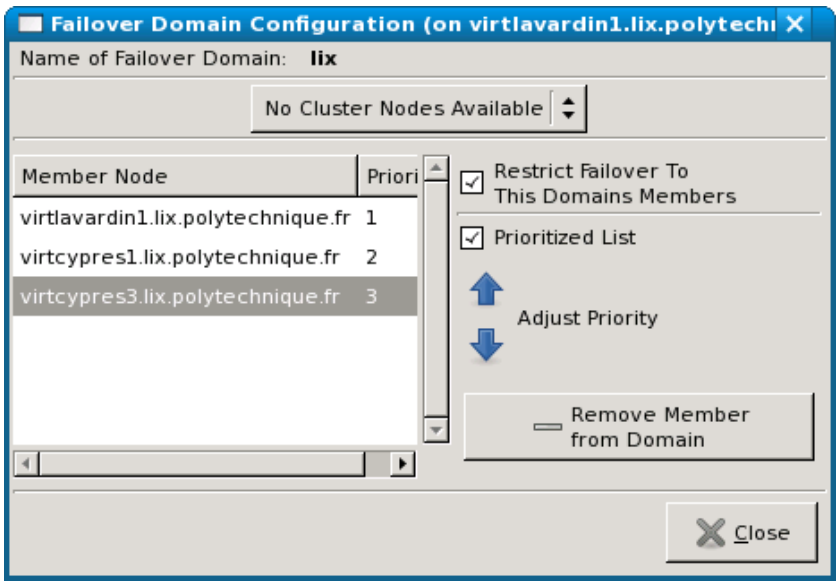

FIG. 1.5: Membres du Failover

1.4.5 Ajout d'une ressource

La ressource correspond au disque ou au fichier partagé sur le cluster par les nodes. Dans le cas présent il s'agit du volume "export" situé sur */dev/VolGroup01/export*.
Ce volume accueillera ultérieurement le système de fichiers GFS dans le répertoire **/gfs-export** que l'on aura créé.
Pour créer une nouvelle ressource, il faut sélectionner l'onglet "Ressources" dans "Managed Ressources" puis cliquer sur "Create a Ressource".
Une fenêtre de configuration apparaît dans laquelle il est possible de choisir différents types de ressources :

Dans notre cas, on sélectionne le type "GFS" dans lequel on entre les paramètres suivants :

- **Name** : GFS
- **Mount Point** : /gfs-export
- **Device** : /Dev/VolGroup01/export

La ressource est dès à présent configurée pour accueillir de nouveaux services.

1.4.6 Démarrage du cluster

Pour activer le cluster on démarre les services **cman** et **clvmd** :

```
[root@virtlavardin1 ~]# service clvmd start
Starting clvmd:                                            [  OK  ]
Activating VGs:   4 logical volume(s) in volume group "VolGroup00" now active
```

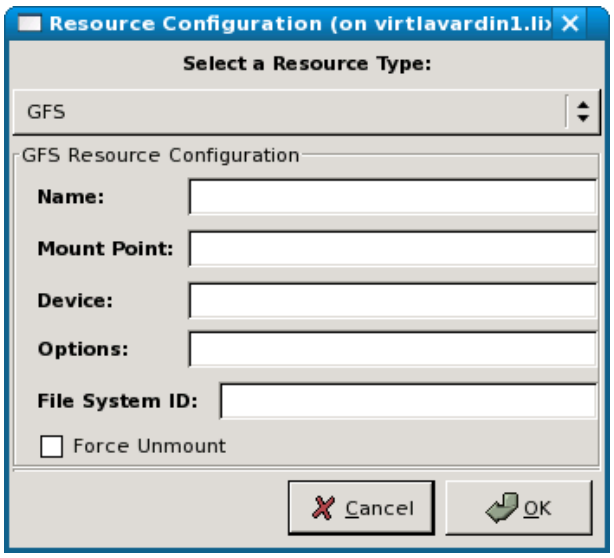

FIG. 1.6: Nouvelle Ressource

```
1 logical volume(s) in volume group "VolGroup01" now active
                                                          [  OK  ]
```

Le cluster est actif, il est possible d'installer le système de fichier GFS sur chaque node.

1.4.7 Implémentation du système de fichiers GFS

L'implémentation de GFS s'effectue lorsque le cluster est démarré, afin que GFS puisse rapatrier l'ensemble de ses informations.
On installe donc en premier lieu sur chaque node, le **package gfs-utils**, qui permet de monter et d'installer GFS.
L'implémentation de GFS se déroule en deux étapes :

1. installation du système de fichiers sur la partition
2. montage de la partition

Avant de procéder à ces opérations il faut vérifier si les volumes groupes ont été correctement activé, via la commande **vgscan** :

```
[root@virtlavardin1 ~]# vgscan
  Reading all physical volumes.  This may take a while...
  Found volume group "VolGroup00" using metadata type lvm2
  Found volume group "VolGroup01" using metadata type lvm2
```

Pour installer le système de fichier sur le volume "export" on utilise la commande **gfs_mkfs**, sa syntaxe est la suivante :
gfs_mkfs -j [Nombre de nodes] -p lock_dlm -t [nom du Cluster] :[nom du système de fichiers] -O [chemin de la ressource]
Dans le cas présent la commande sera la suivante :

```
[root@virtlavardin1 ~]# gfs_mkfs -j 8 -p lock_dlm -t gfslix:GFS -O /dev/VolGroup01/export
Device:                    /dev/VolGroup01/export
Blocksize:                 4096
Filesystem Size:           2358024
Journals:                  8
Resource Groups:           36
Locking Protocol:          lock_dlm
Lock Table:                gfslix:GFS
```

Ici on a volontairement paramétré huit nodes pour GFS, afin d'éviter une réinstallation du système de fichiers si l'on souhaite intégrer ultérieurement une nouvelle node dans le cluster.
Le rapport nous indique que l'installation s'est bien déroulée, on peut donc monter la partition dans le répertoire souhaité (ici /**gfs-export**.
Le montage du volume s'effectue avec la commande **mount** dont la syntaxe est :

```
mount -t [nom du système de fichiers] [chemin de la ressource] [point de montage]
```

On entre donc la commande suivante :

```
[root@virtlavardin1 ~]# mount -t gfs /dev/VolGroup01/export /gfs-export
[root@virtlavardin1 ~]#
```

La dernière opération consiste à démarrer le service gfs via **service gfs start**.
Le système de fichiers est désormais installé et actif dans le Cluster, pour le vérifier il suffit de visualiser l'état du service gfs :

```
[root@virtlavardin1 ~]# service gfs status
Configured GFS mountpoints:
/gfs-export
Active GFS mountpoints:
/gfs-export
```

1.5 Tests effectués

1.5.1 Tests mis en oeuvre

1.6 Evaluation du projet

Chapitre 2

Mise en place de GlusterFS

2.1 Principe

2.1.1 Principe de FUSE

Filesystem in Userpace (FUSE) est un logiciel implémenté dans le noyau Linux, permettant aux utilisateurs non-privilégiés de créer leur propre système de fichiers, sans modifier les sources **kernel** du noyau.
FUSE établit un "pont" entre le système de fichiers virtuel créé sur l'espace utilisateur et le module **kernel**.
Contrairement aux systèmes de fichiers traditionnels qui conservent essentiellement les données sur des volumes physiques, les systèmes de fichiers virtuels ne stockent pas les données eux-mêmes.
Ils peuvent être assimillés à une vue ou une "translation" des données présentes sur un périphérique physique.
En d'autres termes, les fichiers manipulés sur un système de fichier virtuel ne sont autres que des copies virtuelles de données conservées sur une ressource.

(schéma fonctionnement fuse)

Ceci offre certains avantages, comme la limitation de risque en cas de modifications de données qui perturberaient le système.
Toute opération effectuée sur un système de fichiers virtuel est indépendante du système hôte de la machine, il n'est donc pas possible d'altérer son fonctionnement.
Par ailleurs, la manipulation de données virtuelles offrent de meilleures performances de temps et de charge **CPU**, par rapport aux données physiques.
FUSE rend également l'utilisateur entièrement responsable de son espace, il est donc possible de paramétrer minucieusement le système de fichiers créé afin qu'il corresponde au mieux à ses attentes.
De plus en plus utilisé pour les avantages cités, la communauté du logiciel libre voit de nombreux projets apparaîtrent, fonctionnant avec l'environnement FUSE, tels que :

- **GlusterFS** : partage de données type "Cluster"
- **SSHFS** : prise en main à distance sécurisée

- **GmailFS** : gestion de courrier électronique
- **EncFS** : cryptage de données
- **NTFS-3G** : utilisation du système de fichier **NTFS** sur un espace

2.1.2 Principe de GlusterFS

GlusterFS est un système de fichiers permettant le partage de ressources sur un réseau.
Son fonctionnement est similaire à la plupart des protocoles de partage :

- **Serveur** : exporte la ressource
- **Client** : importe la ressource

La caractéristique de GlusterFS, est de permettre, grâce au module FUSE, une plus grande souplesse dans la gestion des données.
L'utilisation des **translators** permet de configurer un grand nombre de paramètres pour la lecture, l'écriture, la modifiction, ou encore la suppression et la restauration des données...

(schéma de GlusterFS)

2.2 Objectifs du projet

La mise en place d'un cluster sous GlusterFS s'avère être interessante pour les points suivants :

- Export/montage de ressources partagées
- Unification de plusieurs volumes
- Gestion des paramètres de lecture et d'écriture

Afin de tester l'efficacité de cette solution, on réalise le schéma suivant :

(schéma de la maquette)

On dispose ici de quatre stations virtualisées ; deux serveurs (**virtadrien1**,**virtlavardin2**) et deux clients (**virtbarracuda2**,**virtadrien2**).
Chacun des serveurs exportera sa ressource propre.
Ces deux partitions seront importées sur chaque client puis concaténées en un seul volume.
Ceci permet l'accès à deux ressources différentes depuis une seule partition.
On souhaite également mettre en place les principes de lecture/écriture en mémoire cache sur les clients afin d'optimiser la gestion des ressources.
On ici choisit d'utiliser plusieurs clients afin d'observer l'accès simultanné à la ressource commune ainsi que les transferts de fichiers entre-eux.

2.3 Installation

Serveurs et clients utilisent les mêmes applications, seule leur configuration variera, par conséquant leur procédure d'installation est identique.
Avant d'installer les modules FUSE et GlusterFS, il faut s'assurer que l'on dispose des **packages** suivants :

- libtool
- gcc
- flex
- bison
- byacc
- kernel-devel

2.3.1 Installation de FUSE

L'utilisation de GlusterFS nécessite en premier lieu l'installation du module FUSE. Celui-ci est disponible à l'adresse *http ://ftp.zresearch.com/pub/gluster/glusterfs/fuse/*. Il faut veiller à utiliser la dernière version (actuellement 2.7.3glfs9).
Une fois l'archive récupérée, on peut procéder à l'installation du module :

```
$ tar -xzf fuse-2.7.2glfs9.tar.gz
$ cd fuse-2.7.2glfs9
$ ./configure  --prefix=/usr  --enable-kernel-module
$ make install
```

FUSE est à présent installé sur le noyau UNIX, et peut être utilisé avec l'ensemble des applications le nécessitant.

2.3.2 Installation de GlusterFS

On récupère tout d'abord l'archive à l'adresse *http ://ftp.zresearch.com/pub/gluster/glusterfs/1.3/*. Comme pour FUSE on utilise la dernière version en date (à ce jour la 1.3.9).
On décompresse ensuite l'archive pour éxécuter le script d'installation :

```
$ tar -xzf glusterfs-1.3.9.tar.gz
$ cd glusterfs-1.3.9
$ ./configure  --prefix=
$ make install
```

L'installation de GlusterFS a créé le répertoire **/etc/glusterfs** qui contient les fichiers de configuration pour le serveur et le client :

- **glusterfs-client.vol.sample** : configuration du client
- **glusterfs-server.vol.sample** : configuration du serveur

L'installation terminée, il est désormais possible de configurer GlusterFS pour le partage de ressources.

2.4 Partage de volumes

On désire dans un premier temps exporter deux volumes depuis chaque serveur et les monter séparemment sur chaque client.
On effectue ceci afin de vérifier le fonctionnement du partage de ressource individuel avant la concaténation des volumes.
On peut schématiser la consigne de la manière suivante :

(schéma)

2.4.1 Configuration du serveur

On détaille dans cette partie, la configuration du serveur " " pour l'export des ressources.
On édite le fichier "/etc/glusterfs/glusterfs-server.vol.sample" dans lequel on entre les commandes suivantes :

```
 ### Export volume "brick" with the contents of "/home/export" directory.
volume brick1
  type storage/posix                   # POSIX FS translator
  option directory /home/export1        # Export this directory
end-volume
```

On configure un volume par ressource, les paramètres suivants doivent être configurés :

- volume : nom du volume partagé
- type : Nom du "translator" utilisé pour le partage ("posix" par défaut)
- option directory : chemin de la ressource à exporter

Dans les commandes ci-dessus on peut paramétrer soi-même le nom du volume ainsi que le chemin de la ressource à exporter.
En revanche il est déconseillé de changer le "translator" (posix) utilisé pour le partage de la partition, celui-ci permettant de conserver le système de fichier de la partition au montage.

```
### Add network serving capability to above brick.
volume server
  type protocol/server
```

```
  option transport-type tcp/server     # For TCP/IP transport
  subvolumes brick1
  option auth.ip.brick1.allow 192.168.113.0 # Allow access to "brick" volume
end-volume
```

La partie "volume server" sert à définir les paramètres de connexion pour l'export des données.
Ce volume doit encadrer le volume "brick" créé précédemment à l'aide la ligne "subvolumes brick" pour identifier la ressource à partager.
On peut également paramétrer des règles de sécurité dans cette partie grâce à la commande "option auth.ip.brick.allow + paramètre" qui permet de définir les machines autorisées à accéder au partage.
Ici "paramètre" a été remplacé par une adresse réseau (192.168.113.0) afin d'authoriser l'accès aux nachines uniquement situées sur ce brin.

Une fois le fichier configurée on peut dès lors activer le partage de la ressource, en utilisant la commande **glusterfsd** sur chaque serveur :

```
$ glusterfsd -f /etc/glusterfs/glusterfs-server.vol.sample
```

Les partitions "/home/export1" et "/home/export2" sont à présent partagées sur le réseau.
On procède de la même facon pour partager les répertoires "/home/export3" et "/home/export4" sur le serveur " ".
On peut maintenant configurer l'import sur les clients.

2.4.2 Configuration du client

Il faut maintenant paramétrer chaque client pour importer la ressource dans un répertoire différent, par exemple ici le client " ".
Pour configurer l'accès à la ressource exportée, on édite le fichier "/etc/glusterfs/glusterfs-client.vol.sample" pour y entrer les lignes ci-dessous :

```
volume client1
  type protocol/client
  option transport-type tcp/client     # for TCP/IP transport
  option remote-host 192.168.113.209         # IP address of the remote brick
  option remote-subvolume brick1        # name of the remote volume
end-volume
```

Similairement au fichier de configuration du serveur, il faut donner un nom au volume qui sera cette fois-ci importé sur la machine.

Par défaut on donne la valeur "client" comme nom de volume.
On paramètre ensuite le protocole utilisé pour l'import de la ressource (tcp) ainsi que l'adresse IP de la station serveur et le nom de volume serveur qui correspond à la ressource exportée ("brick").
On procède de la même manière pour la configuration du lcient " ".
Le client est à présent configuré pour importer la ressource avec la commande **glusterfs** :

```
$ glusterfs -f /etc/glusterfs/glusterfs-client.vol.sample /mnt/import
```

Contrairement au serveur, on précise sur le client, le répertoire qui sert de point de montage, dans la commande d'activation.
Pour vérifier que l'export et l'import de la partition se sont bien déroulés, il suffit de visualiser le contenu du répertoire "/mnt/import" sur le client.
Par exemple sur le client **virtbarracuda2** on obtient :

```
$ ls -a /mnt/import
$ virtadrien1
```

On retrouve bien le dossier "virtadrien1" créé sur le serveur "" avant l'export de la partition.
L'export et le montage d'une partition entre un serveur et un client se déroulent correctement, on propose alors d'exporter deux ressources pour les concaténer en un seul volume.

2.5 Unification des ressources

L'intérêt de GlusterFs réside dans les multiples opérations disponibles sur le montage de ressource, notemment l'addition de deux volumes en un.
Dans le cadre de ce projet on souhaite tester cette fonctionnalité, on réalise donc l'architecture suivante :

(schéma)
Une telle somme de ressources est possible grâce au translator **Unify**.

2.5.1 principe de Unify

On peut illustrer le fonctionnement de **Unify** par le schéma suivant :

(schéma de Unify)
Le client importe les volumes "child" exportés par les serveurs, et les unifient dans le nouveau volume "UNIFY".
Afin de réaliser cette opération, **Unify** nécessite qu'un volume supplémentaire "namespace" soit exporté sur au moins un des serveurs.

Le **namespace** correspond au volume sur lequel sera stocké les informations permettant l'unification des volumes.
Il permet de controler l'ensemble des données conservées sur le volume "UNIFY" en définissant des règles d'écriture et de lecture.
Le **namespace** gère aussi la rélication des données d'un volume à un autre, permettant l'accès à l'ensemble des informations depuis n'importe quel client.

2.5.2 Configuration du serveur

La configuration du serveur ne concerne que l'export du volume **namespace**, on crée donc un volume "brick-ns" sur l'un des serveur (ici **virtadrien1**) dans "/etc/glusterfs/glusterfs-server.vol.sample" qui exportera la ressource **/home/export-ns** :

```
### Volume de stockage pour unify
volume brick-ns
  type storage/posix
  option directory /home/export-ns
end-volume
```

On configure ensuite les paramètres d'export de "brick-ns" dans le volume "server" :

```
### Add network serving capability to above brick.
volume server
  type protocol/server
  option transport-type tcp/server     # For TCP/IP transport
  option auth.ip.brick1.allow * # Allow access to "brick" volume
  option auth.ip.brick2.allow *
  option auth.ip.brick-ns.allow *
end-volume
```

Le serveur est alors configuré pour autoriser l'unification des volumes avec le répertoire **/etc/export-ns** comme volume **namespace**.
On peut activer l'export des ressource avec la commandes **glusterfsd** utilisée précedemment.

2.5.3 Configuration du client

On s'interesse ici à la configuration du translator **unify** sur le client.
On détaille dans cette partie l'unification des volumes "brick1" et "brick2" du serveur **virtadrien1** sur le client **virtadrien2**.
Chacun de ces volumes a une taille actuelle de 8.0 Kiloctets.
Dans le fichier "/etc/glusterfs/glusterfs-client.vol.sample" on crée un nouveau volume

"client-ns" qui va permettre d'identifier le volume **namespace** exporté par le serveur :

```
 volume client-ns
  type protocol/client
  option transport-type tcp/client
  option remote-host 192.168.113.209
  option remote-subvolume brick-ns
end-volume
```

Comme pour le volume "client" on indique le protocole utilisé, ainsi que l'adresse IP du serveur et le volume crée pour l'export du **namespace** sur le serveur (ici "brick-ns").
Il faut ensuite paramétrer le translator **unify** pour qu'il additionne les volumes **brick1** et **brick2**.
On crée le volume "unify" :

```
volume unify
  type cluster/unify
  option scheduler rr
  option namespace client-ns
  subvolumes client1 client2
end-volume
```

La ligne "type cluster/unify" permet de déclarer le translator **unify**.
L'option "sheduler" spécifie quel outil est utilisé pour l'addition des volumes, il en existe plusieurs parmis lesquels **ALU**, **NUFA**, **Random Sheduler** et **Round-Robin Scheduler** sont couramment utilisés.
Dans ce cas, on choisit d'utiliser le **Round-Robin Scheduler** (rr) qui offre de meilleures performances pour des fichiers nombreux et similaires en taille (idéal pour des tests).
La commande "namespace" permet de définir le nom du volume utilisé pour identifier le volume **namespace** exporté par le serveur.
Enfin il faut spécifier quelles sont les ressources à concaténer avec la ligne "subvolumes".
Le client est désormais configuré pour recevoir les deux volumes et les unifier en un nouveau volume "unify".
On lance alors l'import des ressources avec **glusterfs** :

```
 $ glusterfs -f /etc/glusterfs/glusterfs-client.vol.sample /mnt/import
```

Le répertoire "mnt/import" contient à présent les données des volumes "brick1" et "brick2" exportés, et possède une taille égale à la somme de ces deux volumes (soit 16 Kiloctets) :

```
$ du -h /mnt/glusterfs
16.0K    /mnt/glusterfs
```

La procédure sera identique pour le client **virtbarracuda2** souhaitant unifier les volumes **brick3** et **brick4**.
On obtient donc deux volumes "unify" ayant en commun le **namespace** "/home/export-ns" exporté par le serveur **virtadrien1**.
Comme le montre le graphique ci-dessous, en l'absence de translators pour paramétrer les opérations de lecture et d'écriture, les performances de GlusterFS sont moindres que celles d'un autre protocole de partage (exemple NFS) :

(benchmark)
On va donc configurer GlusterFS de manière à optimiser ses performances.

2.6 Gestion lecture/écriture

Afin d'améliorer le traitement des opérations de lecture et d'écriture, GlusterFS dispose de certains translators, durant ce projet ceux qui furent testés sont les suivants :

- **io-cache** : lecture/écriture en mémoire cache
- **write-behind** : écriture en arrière-plan
- **red-ahead** : lecture en arrière-plan

2.6.1 Mémoire cache

Le cache est une mémoire relativement petite et rapide qui stocke les informations les plus utilisées d'une autre mémoire plus grande et plus lente.
Le processus fonctionne ainsi :

(schéma de fonctionnement cache)

1. L'élement demandeur (client) demande une information
2. Le cache vérifie s'il possède ou non cette information. Si il la possède il la transmet directement à l'élément demandeur, sinon il la demande à l'élément fournisseur (mémoire principale).
3. Dans le cas ou il ne possède pas l'information, il la conserve après l'avoir récupéré auprès de l'élément fournisseur, et la conserve pour utilisation ultérieure.

Cette procédure améliore les temps de traitements des requêtes ; un transfert de fichiers entre un client et une petite mémoire (cache) est plus rapide qu'un transfert avec une mémoire plus importante (plus encombrée).
Ce principe s'applique aussi bien pour des opérations de lecture que d'écriture.
Le translator **io-cache** permet de mettre en place une mémoire cache sous GlusterFS, on décide donc de l'utiliser sur chacun de des clients.
Comme précédemment, il faut créer un nouveau volume à chaque utilisation d'un nouveau translator, ici on crée le volume "iocache" dans "glusterfs-client.vol.sample" :

```
### Add IO-Cache feature
volume iocache
  type performance/io-cache
  option page-size 64MB
  subvolumes unify
end-volume
```

La commande "option page-size" permet de définir la taille de la mémoire cache.
Il faut optimiser la taille du cache, si elle est trop petite elle ne conviendra pas aux fichiers importants, et réciproquement pour des fichiers légers avec un cache trop élevé.
On fixe ici le cache à 64 mégaoctets, valeur moyenne de la taille des fichiers qui seront utilisés ultérieurement pour les tests de lecture/écriture.
On applique ce cache sur le volume "unify" grâce à la commande "subvolumes".

2.6.2 Opérations en arrière-plan

Les opérations d'"arrière-plan" consistent à regrouper des procédures de lecture ou d'écriture ensemble.
Cela permet de libérer la mémoire utilisée pour l'éxécution des multiples petites opérations afin de l'attribuer à des opérations moins nombreuses et plus larges.
On utilise alors respectivement les translators **write-behind** et **read-ahead** pour les opérations d'écriture et de lecture en arrière-plan .

Ecriture en arrière-plan

On configure le translator **write-behind** dans "glusterfs-client.conf" :

```
### Add writeback feature
volume writebehind
  type performance/write-behind
  option aggregate-size 1MB
  option flush-behind off
  subvolumes unify
end-volume
```

La ligne "option aggregate-size" permet de définir la taille des opérations d'écritures en arrière-plan. On fixe ici la taille a 1 mégaoctet, les procédures d'écritures seront donc regroupées dans des opérations de 1 mégaoctet.

Lecture en arrière-plan

La lecture en arrière-plan nécessite la configuration du translator **read-ahead** :

```
### Add readahead feature
volume readahead
 type performance/read-ahead
  option page-size 1MB     # unit in bytes
  option page-count 4       # cache per file  = (page-count x page-size)
  subvolumes unify
end-volume
```

Comme pour la procédure d'écriture en arrière-plan, on définie avec la commande "option page-size" la taille des plages de lecture.

2.7 Tests réalisés

2.8 Evaluation du projet