

IUT Nancy-Charlemagne Université Nancy 2
Licence Pro Asrall

Tuteur : Maître de Conférences : Lucas Nussbaum

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

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

Nancy, le 22 mars 2011

Table des matières

1	Introduction	3
1.1	Contexte	3
1.2	Système de fichiers distribué	3
1.3	Le Grid'5000	3
2	NFS	4
2.1	Présentation	4
3	Fuse	5
4	GlusterFs, véritable serveur d'archivage.	6
4.1	Présentation	6
5	MooseFS	7
5.1	Présentation	7
5.1.1	Présentation générale	7
5.1.2	Aspect technique	7
5.2	Mise en place	8
5.2.1	Environnement logiciel	8
6	Ceph	9
6.1	Présentation	9
7	Comparaison	10
7.1	Test de performances	10
8	Conclusion	11
A	Répartition des tâches	12
A.1	Florent Lévine	12
A.2	Jean-François Garcia	12
B	Scripts	13
B.1	GlusterFs	13
B.2	MooseFs	15
B.3	Ceph	15
B.4	NFS	19
B.5	Benchmark	20

Partie 1

Introduction

1.1 Contexte

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

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

1.2 Système de fichiers distribué

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

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

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

1.3 Le Grid'5000

Le Grid'5000 est une infrastructure distribuée de neuf sites sur la France, dédiée à la recherche, sur des systèmes distribués à grande échelle.

Les ingénieurs assurant le développement et le support de l'infrastructure jour après jour viennent pour la plupart de l'INRIA ¹.

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

1. Institut National de Recherche en Informatique et en Automatique

Chaque site possède un ou plusieurs clusters². Par exemple, le site de Nancy possède deux clusters :

graphene : 144 nœuds contenant :

- 1 CPU Intel@2.53GHz
- 4 cores/CPU
- 16GB RAM
- 278GB DISK

griffon : 92 nœuds contenant :

- 2 CPUs Intel@2.5GHz
- 4 cores/CPU
- 16GB RAM
- 278GB DISK

2. TODO : def d'un cluster

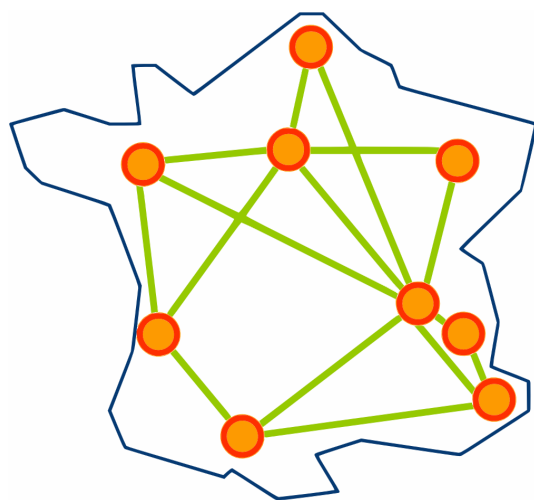


FIGURE 1.1 – Les sites français du Grid'5000

Partie 2

NFS

2.1 Présentation

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

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

On retrouve pour NFS différentes versions :

NFSv1 et v2 définies dans la RFC 1094 prévues pour fonctionner sur UDP.

NFSv3 définie dans la RFC 1813 prenant en charge le TCP.

NFSv4 définie dans la RFC 3010 puis réviser dans la RFC 3530. L'ensemble du protocole est repensé, et les codes sont réécrits. Cette version intègre une meilleur gestion de la sécurité, de la montée en charge, et un systèmes de maintenances simplifiés. NFSv4 supporte les protocoles de transports TCP et RDMA.

2.2 Aspect technique

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

On retrouve pour NFS différentes versions :

NFSv1 et v2 définies dans la RFC 1094 prévues pour fonctionner sur UDP.

NFSv3 définie dans la RFC 1813 prenant en charge le TCP.

NFSv4 définie dans la RFC 3010 puis réviser dans la RFC 3530. L'ensemble du protocole est repensé, et les codes sont réécrits. Cette version intègre une meilleur gestion de la sécurité, de la montée en charge, et un systèmes de maintenances simplifiés. NFSv4 supporte les protocoles de transports TCP (par défaut) et RDMA, ce qui augmente la fiabilité. Nous utiliserons donc cette version NFS dans l'ensemble de nos test.

2.3 Mise en place

Dans cette configuration nous partageons un répertoire `/tmp` sur un serveur à plusieurs clients. Utilisant une image de Debian Squeeze, les paquets `nfs-common` et `nfs-kernel-server` sont directement disponible dans les dépôts. Il faut ensuite ajouter la ligne suivante dans le fichier `/etc/exports` :

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

Dans cette ligne le répertoire est partagé pour le domaine `nancy.grid5000.fr`. Cette manipulation n'est pas dutout sécurisé ! Puisque toutes les machines déployées sur la plateforme de Nancy peuvent accéder au répertoire.

rw : autorise l'accès en lecture et écriture ;

fsid=0 : cette option est propre à NFSv4. Elle indique le point de partage racine. Ce qui permet de monter des partages en fonction d'une racine virtuelle.

insecure : l'option `insecure` permet l'accès aux clients dont l'implémentation NFS n'utilisent pas un port réservé.

subtree_check : permet de limiter l'accès exclusivement au répertoire partager.

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

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

Afin d'utiliser la version 4 de NFS, il est nécessaire de le spécifier dans le type lors du montage.

Partie 3

Fuse

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

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

Pour pouvoir monter un système de fichier, il faut être administrateur à moins que les informations de montage est été renseignées le fichier `/etc/fstab`.

FUSE permet à un utilisateur de monter lui-même un système de fichier. Pour profiter de FUSE, il faut des programmes qui exploitent sa bibliothèque et ces programmes sont nombreux.

[http ://fuse.sourceforge.net](http://fuse.sourceforge.net)

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

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

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

```
1 modprobe fuse
```

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

```
1 nano /etc/modules
```


Partie 4

GlusterFs, véritable serveur d'archivage.

4.1 Présentation

GlusterFS est un système de fichiers distribués libre (gpl v.3), à la fois très simple à mettre en œuvre et dont les capacités de stockage peuvent monter jusqu'à plusieurs petabytes (1,000 milliards octets).

GlusterFS est composé de deux éléments logiciels : un serveur et un client.
GlusterFS supporte plusieurs protocoles de communication tels TCP/IP, InfiniBand RDMA.

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

Le nombre de serveur et le nombre de clients ne sont pas limité.

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

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

Partie 5

MooseFS

5.1 Présentation

5.1.1 Présentation générale

MooseFS (Moose File System) est un système de fichiers répartis à tolérance de panne, développé par Gemius SA. Le code préalablement propriétaire a été libéré et mis à disposition publiquement le 5 mai 2008. Il permet de déployer assez facilement un espace de stockage réseau, réparti sur plusieurs serveurs.

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

En effet le tutoriel de prise en main, disponible sur le site du projet, explique de manière claire comment mettre en place une architecture distribuée en quelques heures. Concernant les utilisations, elles sont multiples et surtout, après la phase de configuration, l'évolution du système est très simple. L'ajout de serveurs, d'espace disque peuvent être gérés très facilement.

5.1.2 Aspect technique

Les montages du système de fichiers par les clients se font à l'aide de FUSE.

MooseFS est constitué de trois types de serveurs :

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

Le point le plus important étant de bien dimensionner le serveur Master (qui stocke les métadonnées) afin de ne pas être limité par la suite. Donc pour ceux qui ne peuvent pas mettre en place des systèmes de stockage réseaux propriétaires assez coûteux, je vous conseille d'étudier cette possibilité. Elle vous permettra de partager des données sur plusieurs machines, de manière rapide, fiable, sécurisée et surtout peu coûteuse.

5.2 Mise en place

5.2.1 Environnement logiciel

Le système utilisé est une Debian Squeeze. MooseFs ne faisant pas partie des dépôts de la distribution, nous avons compilé le paquet à partir des sources dans leurs dernières version (1.6.20).

Partie 6

Ceph

6.1 Présentation

Partie 7

Comparaison

7.1 Test de performances

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

Ce cluster est composé de 144 noeuds, avec pour caractéristique :

- 1 CPU Intel de quatre cœurs cadencé à 2.53 GHz
- 16 Go de RAM
- 278 Go d'espace disque

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

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

Écriture de gros fichiers : écriture d'un fichier de 3 Go¹.

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

Lecture de gros fichiers : lecture du fichier de 3 Go. Opération réalisé en faisant un "cat" du fichier, et en redirigeant la sortie vers /dev/nul afin de ne pas "polluer" le terminal.

1. Fichier créé avec la commande : `dd if=/dev/zero of=/lieu/voulu bs=1G count=3`

Partie 8

Conclusion

Partie A

Répartition des tâches

A.1 Florent Lévigne

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

A.2 Jean-François Garcia

- Étude sur la mise en place de GlusterFS
- Étude sur la mise en place de NFS
- Réalisation d'un script de déploiement de NFS
- Étude sur la mise en place de CephFS
- Réalisation d'un script de déploiement de CephFS

Partie B

Scripts

B.1 GlusterFs

Fichier `deploiementGluster.rb` :

```
1 #!/usr/bin/ruby -w
2 # encoding: utf-8
3
4 # r s e r v a t i o n  d e s  n o e u d s  ( a  l a n c e r  m a n u e l l e m e n t )
5 # o a r s u b  -I  -t  d e p l o y  -l  n o d e s = 8 , w a l l t i m e = 2
6 # o a r s u b  -I  -t  d e p l o y  -l  n o d e s = 8 , w a l l t i m e = 2  -p  " c l u s t e r = ' g r a p h e n e ' "
7
8 if ARGV[0] == nil
9   puts "doit prendre en parametre le nombre de serveurs"
10  exit(1)
11 end
12
13
14 # d o i t  c o n c o r d e r  a v e c  l a  c o m m a n d e  o a r s u b
15 numberOfClients = 5 # inutile : prend les machines dispo restantes
   comme clients
16 numberOfServers = "#{ARGV[0]}.to_i
17
18 infiniband = 1 # 1 : a c t i v , 0 : n o n  a c t i v   ( n e  c h a n g e  r i e n  p o u r
   l ' i n s t a n t )
19
20 # c r a t i o n  d ' u n  f i c h i e r  c o n t e n a n t  l a  l i s t e  d e s  n o e u d s  r  s e r v  s
21 'touch listOfNodes'
22 File.open("listOfNodes", 'w') do |file|
23   file << 'cat $OAR_FILE_NODES | sort -u'
24 end
25
26 # c r a t i o n  d e  d e u x  f i c h i e r s  c o n t e n a n t  l a  l i s t e  d e s  s e r v e u r s ,  e t
   d e s  c l i e n t s
27 'touch listOfClients listOfServers'
28 serverWrited = 0
29 File.open("listOfNodes", 'r') do |node|
```



```

30 File.open("listOfServers", 'w') do |server|
31   File.open("listOfClients", 'w') do |client|
32     while line = node.gets
33       if serverWrited < numberOfServers
34         server << line
35         serverWrited += 1
36       else
37         client << line
38       end
39     end
40   end
41 end
42 end
43
44 # d p l o i e m e n t   d e s   m a c h i n e s
45 puts "Machines en cours de d p l o i e m e n t..."
46 'kadeploy3 -k -e squeeze-collective -u flevigne -f listOfNodes' #
   image collective
47
48
49 # E n v o i e   d ' u n   s c r i p t   d e   c r a t i o n   d ' u n   r p e r t o i r e   d a n s   / t m p /
   s h a r e d s p a c e   s u r   l e s   s e r v e u r s
50 File.open("listOfServers", 'r') do |file|
51   while line = file.gets
52     machine = line.split.join("\n")
53     'ssh root@#{machine} < createFolders.sh'
54   end
55 end
56
57 # E n v o i e   d ' u n   s c r i p t   d e   c r a t i o n   d ' u n   r p e r t o i r e   d a n s   / m e d i a /
   g l u s t e r f s   s u r   l e s   c l i e n t s
58 File.open("listOfClients", 'r') do |file|
59   while line = file.gets
60     machine = line.split.join("\n")
61     'ssh root@#{machine} < createMountDirectory.sh'
62   end
63 end
64
65 masterServer = 'head -n 1 listOfServers'.split.join("\n")
66
67 # g n r a t i o n   d e s   f i c h i e r s   d e   c o n f ,   e t   e n v o i e   d e s   f i c h i e r s   d e   c o n f
   a u x   m a c h i n e s   ( s e r v e u r s   e t   c l i e n t s )
68 puts "Configuration des serveurs et des clients..."
69 'scp listOfServers root@#{masterServer}:'
70 'scp listOfClients root@#{masterServer}:'
71 'scp glusterfs-volgen.rb root@#{masterServer}:'
72 'ssh root@#{masterServer} ./glusterfs-volgen.rb'

```

```

73 # 'ssh root@#{masterServer} < execScript/ex-glusterfs-volgen.sh'
74
75 # d marriage des serveurs
76 puts "D marriage des serveurs..."
77 File.open("listOfServers", 'r') do |file|
78   while line = file.gets
79     machine = line.split.join("\n")
80     'ssh root@#{machine} < startGluster.sh'
81   end
82 end
83
84 # montage du r pertoire par les clients
85 puts "Montage du r pertoire par les clients..."
86 File.open("listOfClients", 'r') do |file|
87   while line = file.gets
88     machine = line.split.join("\n")
89     'ssh root@#{machine} < mountFs.sh'
90   end
91 end
92
93 # r sum des machines
94 puts "GlusterFS op rationnel"
95 puts "\nMachines clients :"
96 puts 'cat listOfClients'
97
98 puts "\nMachines serveurs :"
99 puts 'cat listOfServers'
100
101 puts "\nServeur maitre : #{masterServer}"
102
103 # nettoyage
104 # 'rm listOfNodes listOfClients listOfServers'

```

B.2 MooseFs

B.3 Ceph

Fichier deploymentCeph.rb :

```

1 #!/usr/bin/ruby -w
2 # encoding: utf-8
3
4 #####
5
6 # File Name : deploymentCeph.rb
7

```

```

8 # Purpose :
9
10 # Creation Date : 11-03-2011
11
12 # Last Modified : jeu. 17 mars 2011 14:54:51 CET
13
14 # Created By : Helldar
15
16 #####
17
18 # doit concorder avec la commande oarsub
19
20 if ARGV[0] != nil
21   numberOfServers = ARGV[0].to_i
22   puts "Nb serveur : #{numberOfServers}\n"
23 else
24   puts "Veuillez relancer le script avec les bons param tres!\n
      nUsage : <nombre de serveur>"
25   exit
26 end
27
28 # cr ation d'un fichier contenant la liste des noeuds r serv s
29 'touch listOfNodes'
30 File.open("listOfNodes", 'w') do |file|
31   file << 'cat $OAR_FILE_NODES | sort -u'
32 end
33 # cr ation de deux fichiers contenant la liste des serveurs, et
   des clients
34
35 'touch listOfClients listOfServers'
36 serverWrited = 0
37 File.open("listOfNodes", 'r') do |node|
38   File.open("listOfServers", 'w') do |server|
39     File.open("listOfClients", 'w') do |client|
40       while line = node.gets
41         if serverWrited < numberOfServers
42           server << line
43           serverWrited += 1
44         else
45           client << line
46         end
47       end
48     end
49   end
50 end
51
52 # d ploiment des machines

```

```

53 puts "Machines en cour de d ploiement..."
54 'kadeploy3 -k -e squeeze-collective -u flevigne -f listOfNodes' #
    image collective
55
56 # configuration du serveur
57 serveur_1 = 'head -1 listOfServers | cut -d "." -f1'.strip
58 ip_serveur = 'ssh root@#{serveur_1} ifconfig eth0 |grep inet\ |
    cut -d ":" -f2 |cut -d ' ' -f1'.strip
59
60 # g n ration du fichier de ceph.conf
61
62 'touch ceph.conf'
63 File.open("ceph.conf", 'w') do |file|
64     file << "[global]
65         pid file = /var/run/ceph/$name.pid
66         debug ms = 1
67         keyring = /etc/ceph/keyring.bin
68 [mon]
69         mon data = /tmp/partage/mon$id
70 [mon0]
71         host = #{serveur_1}
72         mon addr = #{ip_serveur}:6789
73 [mds]
74         debug mds = 1
75         keyring = /etc/ceph/keyring.$name"
76     if numberOfServers > 3
77         1.upto(3) { |i|
78             file << "
79 [mds#{i - 1}]"
80             host = 'sed -n #{i + 1}p listOfServers | cut -d '.' -f1'.
                strip
81             puts "host #{i} : #{host}\n"
82             file << "
83             #{host}"
84             }
85         else
86             file << "[mds0]"
87             host = 'sed -n 2p listOfServers | cut -d '.' -f1'.strip
88             file << "
89             #{host}"
90         end
91         file << "
92 [osd]
93         sudo = true
94         osd data = /tmp/partage/osd$id
95         keyring = /etc/ceph/keyring.$name
96         debug osd = 1

```

```

97     debug filstore = 1
98     osd journal = /tmp/partage/osd$id/journal
99     osd journal size = 1000"
100 1.upto(numberOfServers) { |i|
101     file << "
102 [osd#{i - 1}]"
103     host = `sed -n #{i}p listOfServers | cut -d ' ' -f1`.strip
104     file << "
105     #{host}"
106 }
107 end
108
109 # copie du fichier ceph.conf vers le serveur
110 `scp ceph.conf root@#{serveur_1}:/etc/ceph`
111 puts "Envoy !"
112
113 # g n ration du fichier keyring.bin
114 `ssh root@#{serveur_1} cauthtool --create-keyring -n client.admin
    --gen-key keyring.bin`
115 `ssh root@#{serveur_1} cauthtool -n client.admin --cap mds 'allow'
    --cap osd 'allow *' --cap mon 'allow rwx' keyring.bin`
116 `ssh root@#{serveur_1} mv keyring.bin /etc/ceph/`
117 puts "Keyring g n r !"
118
119 # montage
120 `ssh root@#{serveur_1} mount -o remount,user_xattr /tmp`
121 1.upto(numberOfServers - 1) { |i|
122     serveurs = `sed -n #{i + 1}p listOfServers | cut -d " " -f1`.
        strip
123     `ssh root@#{serveurs} mount -o remount,user_xattr /tmp`
124 }
125 puts "Montage fait!"
126
127 # d mariage du serveur
128 `ssh root@#{serveur_1} mkcephfs -c /etc/ceph/ceph.conf --allhosts -
    v -k /etc/ceph/keyring.bin`
129 `ssh root@#{serveur_1} /etc/init.d/ceph -a start`
130 puts "Serveur ceph d marr !"
131
132 # configuration des clients
133 1.upto(`wc -l listOfClients`.to_i) { |i|
134     clients = `sed -n #{i}p listOfClients | cut -d " " -f1`.strip
135     `ssh root@#{clients} mkdir /ceph`
136     `ssh root@#{clients} cfuse -m #{ip_serveur} /ceph`
137 }
138 puts "Clients mont s!"

```

B.4 NFS

Fichier `deploiementNFS.rb` :

```
1  #!/usr/bin/ruby -w
2  # encoding: utf-8
3
4  #####
5
6  # File Name : deploiementNFS.rb
7
8  # Purpose :
9
10 # Creation Date : 17-03-2011
11
12 # Last Modified : lun. 21 mars 2011 12:33:55 CET
13
14 # Created By : Helldar
15
16 #####
17
18 'cat $OAR_FILE_NODES | sort -u > listOfNodes'
19
20 # D ploiement des machines
21 #puts "Machines en cour de d ploiement...\n"
22 #'kadeploy3 -k -e squeeze-collective -u flevigne -f listOfNodes #
   image collective'
23
24 serveur = 'head -1 listOfNodes'.strip
25 puts "Le serveur : #{serveur}!\n"
26 # Suppression du serveur de la liste
27 'sed -i 1d listOfNodes'
28
29 puts "Configuration du serveur...\n"
30 'scp exports root@#{serveur}:/etc/'
31 'ssh root@#{serveur} /etc/init.d/nfs-kernel-server restart'
32
33 puts "Configuration des clients...\n"
34 line = 'wc -l listOfNodes | cut -d ' ' -f1'.strip.to_i
35 puts "Il y a #{line} nodes"
36 1.upto(line) { |i| clients = 'sed -n #{i}p listOfNodes | cut -d "."
   -f1'.strip
37 'ssh root@#{clients} mkdir /tmp/partage'
38 'ssh root@#{clients} mount -t nfs4 #{serveur}:/ /tmp/partage' }
```

B.5 Benchmark

Fichier benchmark.rb :

```
1 #!/usr/bin/ruby -w
2 # encoding: utf-8
3
4
5 if ARGV[0] == nil || ARGV[1] == nil || ARGV[2] == nil
6   puts "Usage correcte :"
7   puts "param1 : nombre de clients participant au bench"
8   puts "param2 : fichier de sortie"
9   puts "param3 : url de la liste des clients"
10  puts "param4 : lieu d'écriture du bench"
11  exit(1)
12 end
13
14 $clientsOfBench = "#{ARGV[0]}"
15
16 # chemin du fichier contenant la liste des clients
17 #listOfClients = "/home/flevigne/glusterFs/listOfClients"
18 listOfClients = "#{ARGV[2]}"
19
20 # chemin ou écrire les données du benchmark
21 #whereToWrite = "/media/glusterfs"
22 whereToWrite = "#{ARGV[3]}"
23
24 # chemin du fichier contenant les résultats
25 $outputRes = "#{ARGV[1]}"
26
27 # le client doit avoir dans /home/flevigne :
28 # - linux-2.6.37.tar.bz2 : noyau linux compressé
29 # - bigFile : un fichier de 3 Go
30
31 # fichier contenant la liste des clients participant au benchmark
32 'touch clientOfBench'
33 'head -#{ $clientsOfBench } #{listOfClients} > clientOfBench'
34
35 # si le fichier $outputRes n'existe pas, on le crée.
36 if !File.exist?($outputRes)
37   'touch #{ $outputRes }'
38 end
39
40 'echo "\nBenchmark sur #{ $clientsOfBench } clients" >> #{ $outputRes }'
41
42 $numberOfClients = open("clientOfBench").read.count("\n").to_i
43
```

```

44 puts "Lancement du benchmarck sur #{numberOfClients} clients."
45
46
47 # lance un travail
48 # parametres :
49 # - name : nom du travail (str)
50 # - work : chemin du script de travail (str)
51 # - whereToWrite : chemin ou ecrire les donnees du benchmark (str
    )
52 # - size : taille (en Mo) du/des fichier(s) a ecrire/lire (float)
53 def startBench(name, work, whereToWrite, size)
54   puts "bench : #{name} en cours..."
55
56   totalSize = size.to_i * $clientsOfBench.to_i
57   workFinished = 0
58   startOfBench = Time.now
59
60   # execution du script pour tous les clients
61   File.open("clientOfBench", 'r') do |file|
62     while line = file.gets
63       fork do
64         machine = line.split.join("\n")
65         'scp #{work} root@#{machine}:/root'
66         'ssh root@#{machine} ./#{work} #{whereToWrite}'
67         exit(0)
68       end
69     end
70   end
71
72   # on attend que tous les clients aient fini leur travail
73   1.upto(numberOfClients) do
74     pid = Process.wait
75     workFinished += 1
76     puts "Machine(s) ayant termin leur travail : #{workFinished}"
77   end
78
79   endOfBench = Time.now
80   duration = endOfBench - startOfBench
81
82   puts "Toute les machines ont termin leur travail."
83
84   puts " --> Le benchmark \"#{name}\" a dur  #{duration} secondes.
      (debit : #{totalSize / duration} Mo/s)"
85
86   'echo "#{name} : #{duration} sec : #{totalSize / duration} Mo/s"
      >> #{ $outputRes }'
87 end

```



```

88
89
90 # lancement du benchmark
91 startBench("écriture de petits fichiers", "writingSmallFiles.sh",
    whereToWrite, 479)
92 startBench("écriture de gros fichiers", "writingBigFiles.sh",
    whereToWrite, 3076)
93 startBench("lecture de petits fichiers", "readingSmallFiles.sh",
    whereToWrite, 479)
94 startBench("lecture de gros fichiers", "readingBigFile.sh",
    whereToWrite, 3076)
95
96 # nettoyage du système de fichier distribue (nécessaire pour
    enchaîner les benchmark)
97 puts "Nettoyage de l'espace de travail..."
98 oneClient = 'head -1 clientOfBench'.strip
99 'ssh root@#{oneClient} rm -r #{whereToWrite}/*'
100
101 puts "\nBenchmark termine"

```

Fichier writingSmallFiles.sh :

```

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

```

Fichier writingBigFiles.sh :

```

1 #!/bin/bash
2
3 whereToWrite=$1
4
5 nameOfMachine='uname -n'
6
7 # on copie le gros fichier au lieu voulu
8 cp /home/flevigne/bigFile "$whereToWrite/$nameOfMachine"

```

Fichier readingSmallFiles.sh :

```

1 #!/bin/bash
2

```

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

Fichier readingBigFile.sh :

```
1 #!/bin/bash
2
3 whereToWrite=$1
4
5 nameOfMachine='uname -n'
6
7 cd "$whereToWrite/$nameOfMachine"
8
9 # lecture du gros fichier
10 cat bigFile > /dev/nul
```