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

Partie 1

Introduction

1.1 Contexte

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

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

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 périphérique comme un disque dur, un CD-ROM, une clé USB, etc. Il existe de nombreux systèmes de fichiers (certains ayant des avantages sur d'autres), dont entre autre l'ext (Extented FS), le NTFS (New Technology FileSystem), ZFS (Zettabyte FS), FAT (File Allocation Table).

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

Un système de fichier distribué est donc utilisé par plusieurs machines en même temps (les machines peuvent ainsi avoir accès a des fichiers distants, l'espace de noms est mis en commun). Un tel système permet donc de partager des données entre plusieurs client, 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, dédié à la recherche, sur des systèmes distribuées 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 répartit sur onze sites, dont neuf en France.

1. Institut National de Recherche en Informatique et en Automatique

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

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

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

2. TODO : def d'un cluster

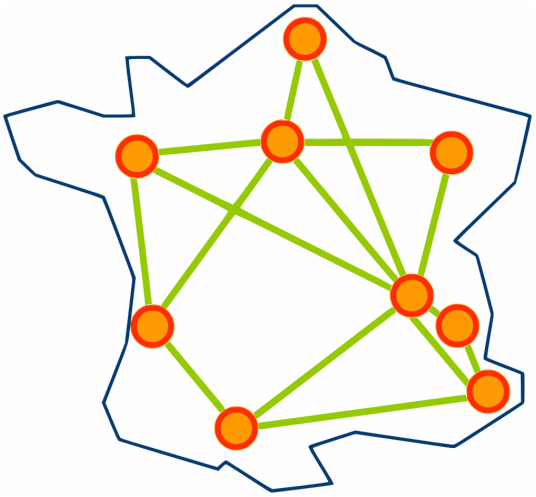

FIGURE 1.1 – Les sites français du Grid'5000

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 Microsystem, permettant de partager des données par le réseau. NFS est aujourd'hui la méthode standard de partage de disques entre machines Unix. C'est une solution simple et pratique, quoique peu sécurisée.

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

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'indisponibilitéé 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 réintégrée 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, répartit 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 clair comment mettre en place une architecture distribuées 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 montage du système de fichiers par les clients se fait à l'aide de FUSE.
MooseFS est constitué de trois types de serveurs :

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

Le point le plus important étant de bien dimensionner le serveur Master (qui stocke les métadonnées) afin de ne pas être limité par le suite. 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 Go [1].

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 taches

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 deploimentGluster.rb :

```
1  #!/usr/bin/ruby -w
2  # encoding: utf-8
3
4  # r servation des noeuds (a lancer manuellement)
5  # oarsub -I -t deploy -l nodes=8,walltime=2
6  # oarsub -I -t deploy -l nodes=8,walltime=2 -p "cluster='graphene'"
7
8  if ARGV[0] == nil
9   puts "doit prendre en parametre le nombre de serveurs"
10  exit(1)
11 end
12
13
14 # doit concorder avec la commande oarsub
15 numberOfClients = 5 # inutile : prend les machines dispo restantes
      comme clients
16 numberOfServers = "#{ARGV[0]}".to_i
17
18 infiniband = 1 # 1 : activ , 0 : non activ  (ne change rien pour
      l'instant)
19
20 # cr ation d'un fichier contenant la liste des noeuds r serv s
21 `touch listOfNodes`
22 File.open("listOfNodes", 'w') do |file|
23  file << `cat $OAR_FILE_NODES | sort -u`
24 end
25
26 # cr ation de deux fichiers contenant la liste des serveurs, et
      des clients
27 `touch listOfClients listOfServers`
28 serverWrited = 0
29 File.open("listOfNodes", 'r') do |node|
```

```
30  File.open("listOfServers", 'w') do |server|
31   File.open("listOfClients", 'w') do |client|
32    while line = node.gets
33     if serverWrited < numberOfServers
34      server << line
35      serverWrited += 1
36     else
37      client << line
38     end
39    end
40   end
41  end
42 end
43
44 # d ploiement des machines
45 puts "Machines en cours de d ploiement..."
46 `kadeploy3 -k -e squeeze-collective -u flevigne -f listOfNodes` #
     image collective
47
48
49 # Envoie d'un script de cr ation d'un r pertoire dans /tmp/
     sharedspace sur les serveurs
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 # Envoie d'un script de cr ation d'un r pertoire dans /media/
     glusterfs sur les clients
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 ration des fichiers de conf, et envoie des fichiers de conf
     aux machines (serveurs et clients)
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`
```

```
#`ssh root@#{masterServer} < execScript/ex-glusterfs-volgen.sh`

# d marrage des serveurs
puts "D marrage des serveurs..."
File.open("listOfServers", 'r') do |file|
 while line = file.gets
  machine = line.split.join("\n")
  `ssh root@#{machine} < startGluster.sh`
 end
end

# montage du r pertoire par les clients
puts "Montage du r pertoire par les clients..."
File.open("listOfClients", 'r') do |file|
 while line = file.gets
  machine = line.split.join("\n")
  `ssh root@#{machine} < mountFs.sh`
 end
end

# r sum  des machines
puts "GlusterFS op rationnel"
puts "\nMachines clients :"
puts `cat listOfClients`

puts "\nMachines serveurs :"
puts `cat listOfServers`

puts "\nServeur maitre : #{masterServer}"

# nettoyage
#`rm listOfNodes listOfClients listOfServers`
```

B.2 MooseFs

B.3 Ceph

Fichier deploimentCeph.rb :

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

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

# File Name : deploiementCeph.rb

```

```
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!\
       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 ploiement des machines
```

```
puts "Machines en cour de d ploiement..."
`kadeploy3 -k -e squeeze-collective -u flevigne -f listOfNodes` # image collective

# configuration du serveur
serveur_1 = `head -1 listOfServers | cut -d "." -f1`.strip
ip_serveur = `ssh root@#{serveur_1}  ifconfig eth0 |grep inet\  | cut -d ":" -f2 |cut -d ' ' -f1`.strip

# g n ration du fichier de ceph.conf

`touch ceph.conf`
File.open("ceph.conf", 'w') do |file|
     file << "[global]
             pid file = /var/run/ceph/$name.pid
             debug ms = 1
             keyring = /etc/ceph/keyring.bin
[mon]
             mon data = /tmp/partage/mon$id
[mon0]
             host = #{serveur_1}
             mon addr = #{ip_serveur}:6789
[mds]
             debug mds = 1
             keyring = /etc/ceph/keyring.$name"
  if numberOfServers > 3
    1.upto(3) { |i|
      file << "
[mds#{i - 1}]"
      host = `sed -n #{i + 1}p listOfServers | cut -d '.' -f1`.strip
      puts "host #{i} : #{host}\n"
      file << "
    #{host}"
    }
  else
    file << "[mds0]"
    host = `sed -n 2p listOfServers | cut -d '.' -f1`.strip
    file << "
    #{host}"
  end
  file << "
[osd]
    sudo = true
    osd data = /tmp/partage/osd$id
    keyring = /etc/ceph/keyring.$name
    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 marrage 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 deploimentNFS.rb :

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

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

# File Name : deploiementNFS.rb

# Purpose :

# Creation Date : 17-03-2011

# Last Modified : lun. 21 mars 2011 12:33:55 CET

# Created By : Helldar

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

`cat $OAR_FILE_NODES | sort -u > listOfNodes`

# D ploiement des machines
#puts "Machines en cour de d ploiement...\n"
#`kadeploy3 -k -e squeeze-collective -u flevigne -f listOfNodes # image collective`

serveur = `head -1 listOfNodes`.strip
puts "Le serveur : #{serveur}!\n"
# Suppression du serveur de la liste
`sed -i 1d listOfNodes`

puts "Configuration du serveur...\n"
`scp exports root@#{serveur}:/etc/`
`ssh root@#{serveur} /etc/init.d/nfs-kernel-server restart`

puts "Configuration des clients...\n"
line = `wc -l listOfNodes | cut -d ' ' -f1`.strip.to_i
puts "Il y a #{line} nodes"
1.upto(line) { |i| clients = `sed -n #{i}p listOfNodes | cut -d "." -f1`.strip
 `ssh root@#{clients} mkdir /tmp/partage`
 `ssh root@#{clients} mount -t nfs4 #{serveur}:/ /tmp/partage` }
```

B.5 Benchmark

Fichier benchmark.rb :

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


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

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

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

# chemin ou  crire  les donn es du benchmark
#whereToWrite = "/media/glusterfs"
whereToWrite = "#{ARGV[3]}"

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

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

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

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

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

$numberOfClients = open("clientOfBench").read.count("\n").to_i

```

```
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  crire  les donn es 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 * $clientsOfBench.to_i
57  workFinished = 0
58  startOfBench = Time.now
59
60  # execution du sript pour tous les clients
61  File.open("clientOfBench", 'r') do |file|
62   while line = file.gets
63    fork do
64     machine = line.split.join("\n")
65     `scp #{work} root@#{machine}:/root`
66     `ssh root@#{machine} ./#{work} #{whereToWrite}`
67     exit(0)
68    end
69   end
70  end
71
72  # on attend que tous les clients aient fini leur travail
73  1.upto($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("ecriture de petits fichiers", "writingSmallFiles.sh",
      whereToWrite, 479)
92 startBench("ecriture 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 syst me de fichier distribue (necessaire pour
      enchainer 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/$nameOfMachine"
9
10 # decompression dans ce repertoire
11 cd "$whereToWrite/$nameOfMachine"
12 tar -xf /home/flevigne/linux-2.6.37.tar.bz2
```

Fichier writingBigFiles.sh :

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