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 18 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

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.

Partie 3

GlusterFs

3.1 Présentation

Partie 4

MooseFS

4.1 Présentation

4.1.1 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.

4.1.2 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.

4.2 Mise en place

4.2.1 Environnement logiciel

Le système utilisé est une Debian Squeeze. MosseFs 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 5

Ceph

5.1 Présentation

Partie 6

Comparaison

6.1 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 7

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`
73  #`ssh root@#{masterServer} < execScript/ex-glusterfs-volgen.sh`
74
75  # d marrage des serveurs
76  puts "D marrage 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
```

```
# 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

# Purpose :

# Creation Date : 11-03-2011

# Last Modified : jeu. 17 mars 2011 14:54:51 CET

# Created By : Helldar

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

# doit concorder avec la commande oarsub

if ARGV[0] != nil
  numberOfServers = ARGV[0].to_i
  puts "Nb serveur : #{numberOfServers}\n"
else
  puts "Veuillez relancer le script avec les bons param tres!\n"
  exit
end

# cr ation d'un fichier contenant la liste des noeuds r serv s
`touch listOfNodes`
File.open("listOfNodes", 'w') do |file|
        file << `cat $OAR_FILE_NODES | sort -u`
end
# cr ation de deux fichiers contenant la liste des serveurs, et des clients

`touch listOfClients listOfServers`
```

```
serverWrited = 0
File.open("listOfNodes", 'r') do |node|
        File.open("listOfServers", 'w') do |server|
                File.open("listOfClients", 'w') do |client|
                        while line = node.gets
                                if serverWrited < numberOfServers
                                        server << line
                                        serverWrited += 1
                                else
                                        client << line
                                end
                        end
                end
        end
end

# 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}  hostname -i`.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.$name"
        if numberOfServers > 3
         1.upto(3) { |i| file << "
[mds#{i - 1}]"
         host = `sed -n #{i + 1}p listOfServers | cut -d '.' -f1`.strip
         file << "
       host #{host}" }
        else
         file << "[mds0]"
         host = `sed -n 2p listOfServers | cut -d '.' -f1`.strip
         file << "
       host #{host}"
        end
        file << "
[osd]
       sudo = true
       osd data = /tmp/partage/osd$id
```

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

B.4 NFS

Fichier deploimentNFS.rb :

```
1  #!/usr/bin/ruby -w
2  # encoding: utf-8
3
4  ###############################################
5
6  # File Name : deploiementNFS.rb
7
8  # Purpose :
9
10 # Creation Date : 17-03-2011
11
12 # Last Modified : jeu. 17 mars 2011 16:29:07 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 #{serveur}:/tmp -t nfs /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 : nombe 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'ecriture 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   crire   les donn es du benchmark
21 #whereToWrite = "/media/glusterfs"
22 whereToWrite = "#{ARGV[3]}"
23
24 # chemin du fichier contenant les r sultats
```

```
25 $outputRes = "#{ARGV[1]}"
26
27 # le client doit avoir dans /home/flevigne :
28 # - linux-2.6.37.tar.bz2 : noyau linux compress
29 # - bigFile : un fichier de 3 Go
30
31 # fichier contenant la liste des clients participant au benchmark
32 `touch clientOfBench`
33 `head -#{$clientsOfBench} #{listOfClients} > clientOfBench`
34
35 # si le fichier $outputRes n'existe pas, on le cr e.
36 if !File.exist?($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   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
```