h e p i a

Haute école du paysage, d'ingénierie
et d'architecture de Genève

Hes·so

Haute Ecole Spécialisée
de Suisse occidentale

Projet de semestre 2012

Sheepdog : Distributed Storage Management for qemu/kvm

Étudiant : **GOLLIET Michael**
professeur responsable : **LITZISTORF Gérald**

Rapport sheepdog

Noms **Date** **Signature**

1 Résumé

As part of this project, i had to evaluate **sheepdog**, an **open source** solution developped by **NTT** (Nippon Telegraph and Telephone corporation).

Sheepdog is a **distributed storage system**. It provides highly available **block level storage volumes** that can be attached to QEMU-based virtual machines.

To evaluate sheepdog I did the following tests :

1-I've created a cluster of 3 machines.
2-I've converted a qemu/kvm VM to use with sheepdog and I've observed the data distribution on the cluster.
3-On a 2 nodes cluster, i've tried to add one node to observe the data distribution on the cluster.
4-On a 3 nodes cluster, i've tried to remove one node to observe the data distribution on the cluster.

To understand how sheepdog works, i also had to study others technologies such as **corosync cluster engine** :

Corosync cluster engine is a **group communication system** used by sheepdog as a cluster manager.
The cluster manager is used to manage **cluster membership** and to **guarantee safe ordered messages** between the differents machines of the cluster (via **Totem SRP protocol**).

It provide C programming interface such as **CPG** (closed process group) which is used by sheepdog to communicate with the members of the group communication system (corosync).

Sheepdog is in constant development, i've started with the version 2.3.0-1 (which is implemented in fedora 16), i also tested versions 3.0-0 and 3.0-57 (current version is 4).

Sheepdog is still a 'young' product, it's an innovative and interesting solution but it still need some work to be used in a production environment.

Poblems encountered :

I was able to create a 3 nodes cluster, create a VM or store a converted KVM Virtual machine on it with differents degrees of redundancies without so much troubles...

But i had a LOT of issues, to maintain the cluster membership, indeed after a while, the cluster's integrity is lost.
It seems that some machines left the cluster even if they're still up (they are still members of corosync cluster but not members of sheepdog cluster anymore).
Sometimes i can get them back in the cluster by restarting the sheep daemon on every machines but most of the times i had to manually clean the storage on every machines and format the cluster again (of course all the data are lost...)

The same thing happened when i tried to add or remove a machine of the cluster...

i spent most of my time trying to understand those events without real success....

2 Enoncé

il s'agit d'etudier et de tester la solution sheepdog implémentée dans fedora 16.
Suivre le developpement et comprendre les principaux mécanismes de sheepdog

Description :

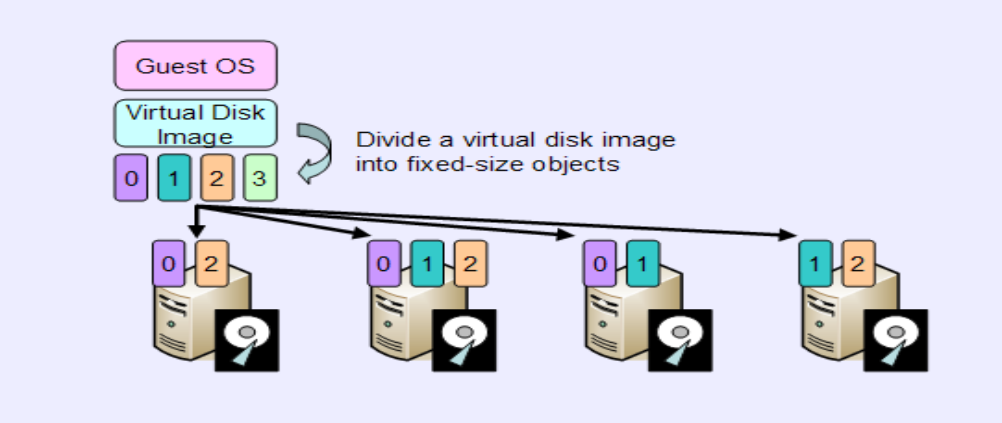
Sheepdog est une solution open source de **stockage distribué** pour machine virtuelle **QEMU/KVM**.

Sheepdog est conçu pour offrir :

-un **stockage de volumes** (**Block level**) qui peut être attachés à des VMs QEMU
-une **redondance** des disques virtuels à partir d'un ensemble de machines (cluster)
-une grande **souplesse** pour ajouter ou supprimer des machines (**scalability**).
-une architecture entièrement **symétrique** (pas besoin de serveur central)
-> il se présente comme un service (**daemon**) présent sur toutes les machines du cluster.

-L'image d'un disque virtuel (de taille variable) est **segmentée** en **objets de taille fixe** par le **block driver de QEMU**.
- sheepdog s'occupe de répartir ces objets sur le cluster.

Dans la figure ci-dessous, trois copies d'un disque virtuel sont réparties sur un cluster de quatre machines :



3 Analyse

Etape et objectifs :

1-comprendre le fonctionnement et tester la **gestion du cluster (avec corosync)**.
2- analyser l'echange de messages entre sheepdog et corosync.
3-étudier et tester la gestion des objets (**stockage,redondance**).

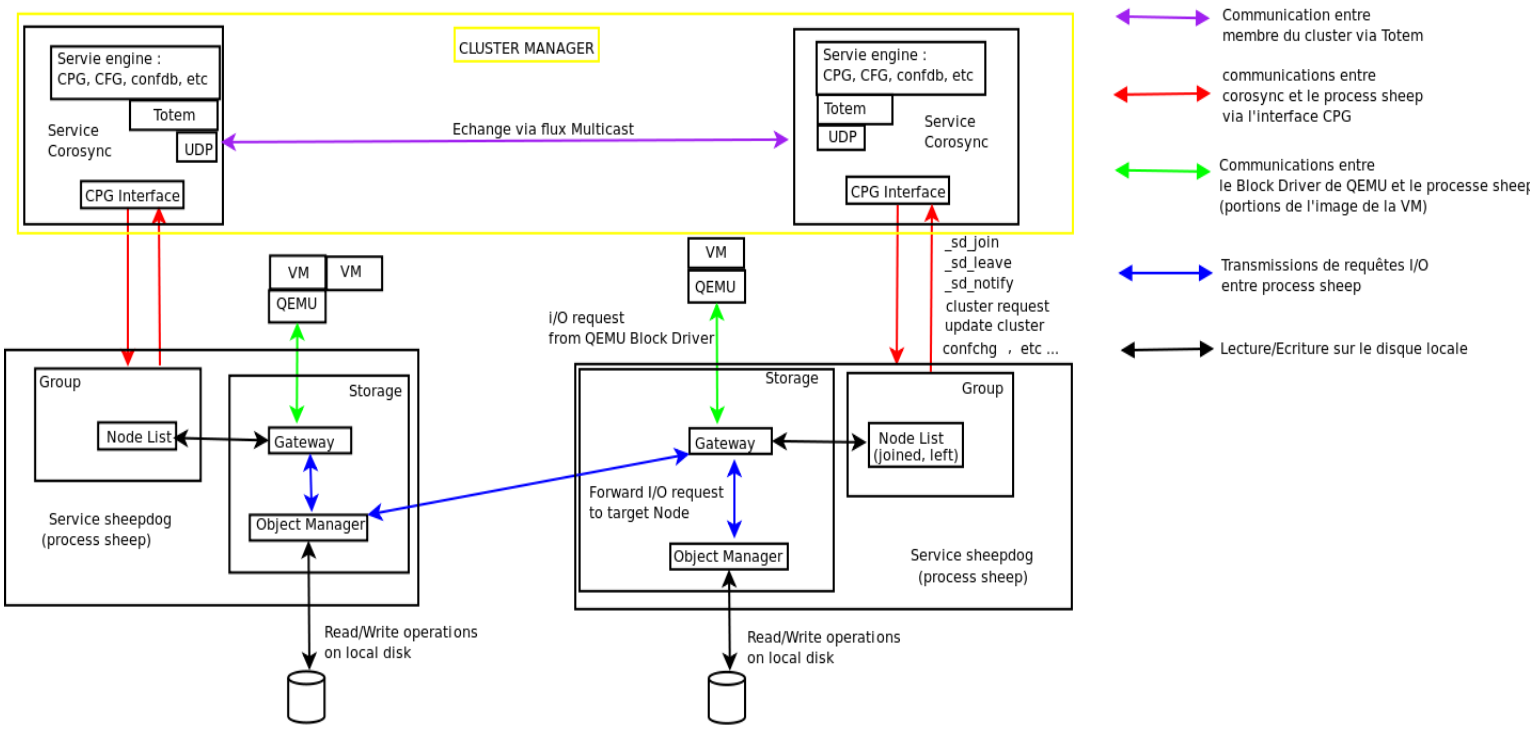
3.1 Architecture de sheepdog :

-Sheepdog est un '**entrepôt d'objet**' distribué sur un ensemble de machine (cluster).
C'est en réalité un ensemble de processus (daemon appelé '**sheep**') gérant chacun une portion de l'espace de stockage global du cluster.

-**Corosync** est un service permettant le '**membership**' et **l'echange de messages** entre membres du cluster.

-Pour le bon fonctionnement d'un cluster sheepdog, on doit avoir démarré sur chaque machines le service corosync et le(s) processus sheep.

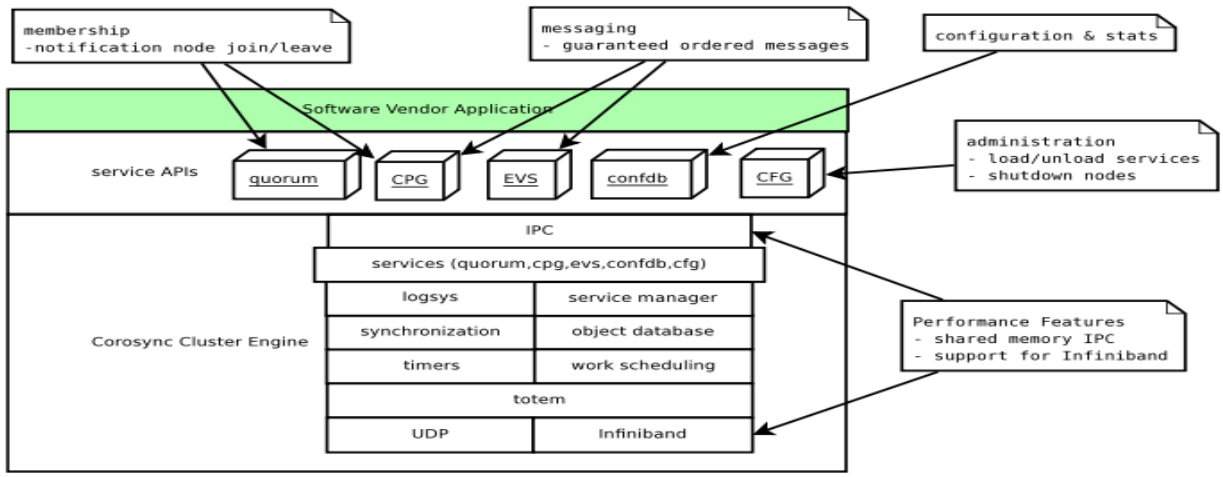
Les communications entre ces processus sont décrites dans la figure ci-dessous, représentant l'architecture de sheepdog :



3-2 Fonctionnement de Corosync (Cluster manager) :

Corsync est un système de communication de groupe, utilisé par sheepdog pour gérer la connexion (membership) au cluster, ainsi que pour permettre et garantir le bon ordonnancement des messages entre les différentes machines du cluster.
Les membres du cluster sont découverts grâce au **multicast** (ce qui permet d' éviter les serveurs de meta-données).

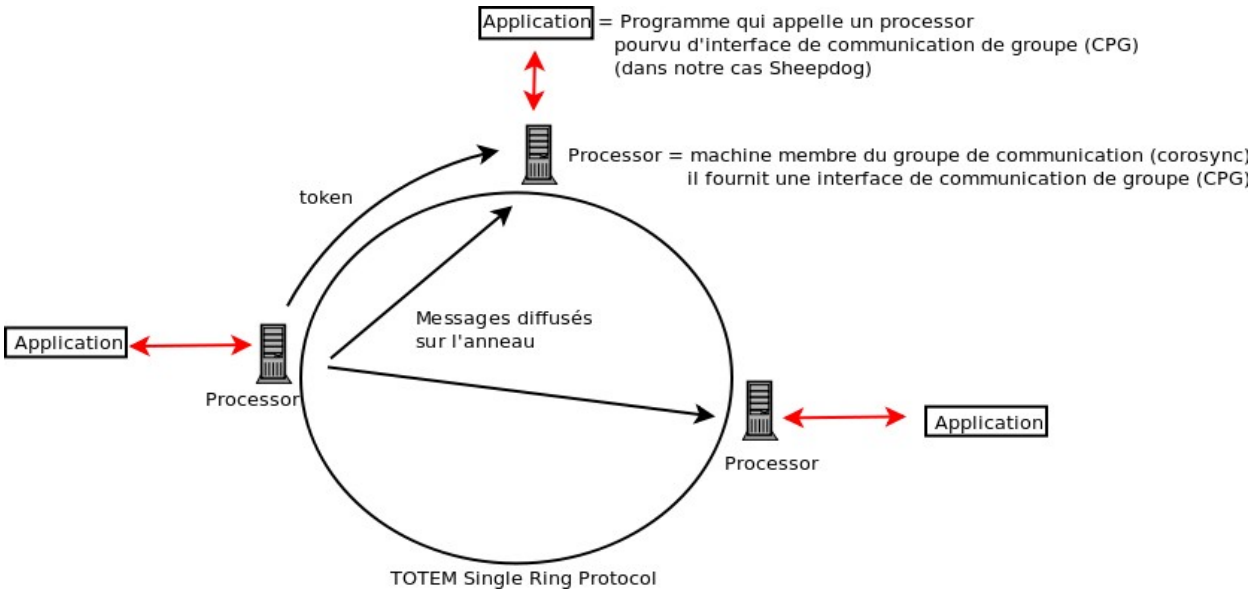
Architecture de corosync :



Corosync Cluster Engine utilise le protocole **TOTEM SRP** (**S**ingle **R**ing **P**rotocol) / **RRP** (**R**edondant **R**ing **P**rotocol) pour établir le membership au cluster et pour garantir l'ordonnancement des messages entre les différentes machines membres du groupe de communication (appelées processors).

chaque machines **transmet périodiquement** (toutes les 300 ms par defaut) un **jeton** (token) a son voisin sur l'anneau. Dès qu'une machine détient le jeton il diffuse un ou plusieurs messages aux autres machines présente sur l'anneau.

Les message échangés entre un process sheep et totem passe par **CPG** (**C**losed **P**rocess **G**roups) un service engine de corosync qui fournit une interface permettant a une machine et a une application de communiquer.



Le fonctionnement du protocole **Totem srp** est décrit plus en details en annexe (section **8.1**).

3.3 Configuration de corosync :

L'adresse et le port multicast par defaut est definie dans **/etc/corosync/corosync.conf :**

```
token {
        interface {
                ringnumber: 0
                bindnetaddr: 192.168.1.0     //reseau, ou plage d'addresse
                mcastaddr: 226.94.1.1        // adresse du process group
                mcastport: 5405              // port pour les communications entre processor
        }
}
```

il est possible de definir plusieurs interfaces, pour cela il faut incrémenter ringnumber (chaque interface aura donc son propre anneau) et choisir un reseau (ou plage d'adresse) a associer a cette interface.

Ex : ringnumber: 1
bindnetaddr: 10.1.2.0

il est aussi possible de définir plusieurs options dans la partie token (voir annexe) , bien que les valeurs par default garantissent un bon fonctionnement dans la majorité des cas.
A noter qu'il est possible de crypter les échanges de messages avec l'option **secauth : on** (au détriment des performances réseau)

corosync tools :

Voici quelque outils utiles, pour vérifier le bon fonctionnement de corosync :

-**corosync-cfgtools** permet de connaître le statut de l'anneau, d'interroger corosync sur la présence d'une machine en précisant son id, il est aussi possible de retirer une machine de l'anneau (shutdown)

```
[root@localhost labotd]# corosync-cfgtool -s
Printing ring status.
Local node ID 1862379712
RING ID 0
        id      = 192.168.1.111
        status  = ring 0 active with no faults
[root@localhost labotd]# corosync-cfgtool -a 1879156928
192.168.1.112
```

-**corosync-cpgtool** permet de connaître les applications qui communiquent avec les machines présentes sur l'anneau (dans ce cas le service sheepdog tourne sur 192.168.1.111 et sur 192.168.1.112)

```
[root@localhost labotd]# corosync-cpgtool
Group Name             PID          Node ID
sheepdog
                      2250      1862379712 (192.168.1.111)
                     11271      1879156928 (192.168.1.112)
```

-**corosync-objtl** permet de connaitre les informations concernant le protocole totem srp, on peut aussi vérifier les machines présentes sur l'anneau (membres du cluster) :

```
runtime.totem.pg.msg_reserved=1
runtime.totem.pg.msg_queue_avail=761
runtime.totem.pg.mrp.srp.orf_token_tx=70
runtime.totem.pg.mrp.srp.orf_token_rx=131697
runtime.totem.pg.mrp.srp.memb_merge_detect_tx=63139
runtime.totem.pg.mrp.srp.memb_merge_detect_rx=63149
runtime.totem.pg.mrp.srp.memb_join_tx=1171
runtime.totem.pg.mrp.srp.memb_join_rx=2140
runtime.totem.pg.mrp.srp.mcast_tx=2104
runtime.totem.pg.mrp.srp.mcast_retx=0
runtime.totem.pg.mrp.srp.mcast_rx=2931
runtime.totem.pg.mrp.srp.memb_commit_token_tx=140
runtime.totem.pg.mrp.srp.memb_commit_token_rx=140
```

```
[root@localhost labotd]# corosync-objctl | grep members
runtime.totem.pg.mrp.srp.members.16777343.ip=r(0) ip(127.0.0.1)
runtime.totem.pg.mrp.srp.members.16777343.join_count=1
runtime.totem.pg.mrp.srp.members.16777343.status=joined
runtime.totem.pg.mrp.srp.members.1879156928.ip=r(0) ip(192.168.1.112)
runtime.totem.pg.mrp.srp.members.1879156928.join_count=18
runtime.totem.pg.mrp.srp.members.1879156928.status=joined
runtime.totem.pg.mrp.srp.members.1912711360.ip=r(0) ip(192.168.1.114)
runtime.totem.pg.mrp.srp.members.1912711360.join_count=6
runtime.totem.pg.mrp.srp.members.1912711360.status=joined
```

-les logs de corosync : **/var/log/cluster/corosync.log**

-pour plus de details sur corosync :

man corosync.conf
man corosync_overview

data sheet officiel de corosync (disponible sur **http://www.corosync.org**)

4 Réalisation

Pour mon étude je vais utiliser trois machines sous fedora 16 pour former un cluster ,
Dans un premier temps il a fallu sur les trois machines :

-lancer corosync au démarrage : **ckkconfig -level 5 corosync on**

-créer d'un volume logique LV pour sheepdog de 100 Go :

lvmcreate –name lv_sheepdog –size 100G vg

-créer d'un système de fichier EXT4 pour le volume logique :

mkfs -t ext4 /dev/vg/lv_sheepdog

-créer d'un point de montage et montage automatique au démarrage:

mkdir /mnt/sheepdog

echo "/dev/vg/lv_sheepdog /mnt/sheepdog ext4 " >> /etc/fstab

mount -a

-créer un volume qui va être utilisé sheepdog :

mkdir /mnt/sheepdog/data/

-Comptes sur chaque machine :

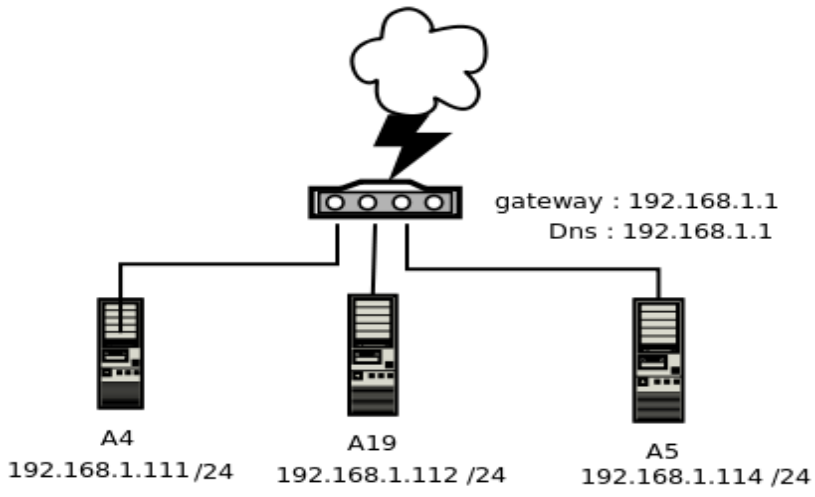
– utilisateur **labotd** (password : **labolabo**) , admin **root** (password : **rootroot)**

4.1 Installation (ou mise à jour) de sheepdog :

- **yum -y install corosynclib-devel git autoconf automake** //dépendances
- **git clone git://github/collie/sheepdog.git** //sources de sheepdog
- **cd sheepdog**
- **./autogen.sh**
- **./configure**
- **make install** //compile et installe les sources

4.2 Configuration :

-configurer les paramètres **TCP/IP** (ip, netmask, gateway et DNS)



-préciser le réseau 192.168.1.0 dans **/etc/corosync/corosync.conf** (Bindnetaddr)
-redémarrer le service network pour prendre en compte les modifications.

4.3 Lancement de sheepdog :

Pour faire fonctionner sheepdog il suffit juste de :

1-démarrer corosync : **service corosync start** (si le service n'est pas deja démarré)

```
corosync [MAIN  ] Corosync built-in features: nss dbus rdma snmp
corosync [MAIN  ] Successfully read main configuration file '/etc/corosync/corosync.conf'.
corosync [TOTEM ] Initializing transport (UDP/IP Multicast).
corosync [TOTEM ] Initializing transmit/receive security: libtomcrypt SOBER128/SHA1HMAC (mode 0).
corosync [TOTEM ] The network interface [192.168.1.112] is now up.
corosync [SERV  ] Service engine loaded: corosync extended virtual synchrony service
corosync [SERV  ] Service engine loaded: corosync configuration service
corosync [SERV  ] Service engine loaded: corosync cluster closed process group service v1.01
corosync [SERV  ] Service engine loaded: corosync cluster config database access v1.01
corosync [SERV  ] Service engine loaded: corosync profile loading service
corosync [SERV  ] Service engine loaded: corosync cluster quorum service v0.1
corosync [MAIN  ] Compatibility mode set to whitetank.  Using V1 and V2 of the synchronization engine.
corosync [TOTEM ] A processor joined or left the membership and a new membership was formed.
corosync [CPG   ] chosen downlist: sender r(0) ip(192.168.1.112) ; members(old:0 left:0)
corosync [MAIN  ] Completed service synchronization, ready to provide service.
```

Log au demarrage du service : **/var/log/cluster/corosync/corosync.log**

```
192.168.1.112    226.94.1.1    UDP    126 Source port: hpoms-dps-lstn  Destination port: netsupport
192.168.1.112    226.94.1.1    UDP    126 Source port: hpoms-dps-lstn  Destination port: netsupport
192.168.1.112    226.94.1.1    UDP    126 Source port: hpoms-dps-lstn  Destination port: netsupport
192.168.1.112    226.94.1.1    UDP    126 Source port: hpoms-dps-lstn  Destination port: netsupport
192.168.1.112    226.94.1.1    UDP    126 Source port: hpoms-dps-lstn  Destination port: netsupport
```

port source : 5404 port destination : 5405

remarque : si l'on ne souhaite pas désactiver le firewall (**service iptables stop**), il faut permettre les communications sur les port 5404 et 5405 :

```
iptables -I INPUT -p udp -m state --state NEW -m multiport --dports 5404,5405 -j ACCEPT
```

2-lancer sheepdog : **sheep /mnt/sheepdog/data**

Il est possible de préciser certaines options au démarrage du service :

```
Options:\n\
  -p, --port              specify the TCP port on which to listen\n\
  -f, --foreground        make the program run in the foreground\n\
  -l, --loglevel          specify the level of logging detail\n\
  -d, --debug             include debug messages in the log\n\
  -D, --directio          use direct IO when accessing the object store\n\
  -z, --zone              specify the zone id\n\
  -c, --cluster           specify the cluster driver\n\
  -h, --help              display this help and exit\n\


  Available log levels:\n\
    #    Level           Description\n\
    0    SDOG_EMERG      system has failed and is unusable\n\
    1    SDOG_ALERT      action must be taken immediately\n\
    2    SDOG_CRIT       critical conditions\n\
    3    SDOG_ERR        error conditions\n\
    4    SDOG_WARNING    warning conditions\n\
    5    SDOG_NOTICE     normal but significant conditions\n\
    6    SDOG_INFO       informational notices\n\
    7    SDOG_DEBUG      debugging messages\n");
```

cela crée alors 3 repertoires (**epoch**, **journal** et **obj**) ainsi d'un fichier **log** dans **/mnt/sheepdog/data/**

remarques :

-pour lancer plusieurs sheep sur une machine (par exemple sur plusieurs partitions ou sur plusieurs disques) il faut aussi préciser le port (7000 par default)

ex : **sheep /mnt/sheepdog/data -p 7001**

sheep /mnt/sheepdog/data2 -p 7002

- **service sheepdog start** lance sheepdog par default sur **/var/lib/sheepdog**

dans ce cas il faudrait faire aussi : **#mount -o remount,user_xattr /var/lib/sheepdog**

autrement le service crash au démarrage.

3- lors de la premère utilisation (ou si on a effacé le contenu des entrepôts sur toutes les machines), il faut sur **une seule** machine du cluster lancer : **collie cluster format -c 3**
pour formater le cluster et préciser le degrés de redondance souhaité (nombre de copies)

4-pour terminer le service sheepdog :

-La manière propre est de tuer tout les sheep du cluster d'un coup via la commande :
collie cluster shutdown

-si la commande précédante n'a pas marché (blockage, echec), ou alors si on veut tuer un sheep spécifiquement on utilise la commande **pkill sheep :**

```
[root@localhost labotd]# ps axf | grep sheepdog
17314 pts/0    S+     0:00              |   |   |          \_ grep
--color=auto sheepdog
17029 ?        Ss     0:00 sheep /mnt/sheepdog/data/
17030 ?        Rsl    8:21 sheep /mnt/sheepdog/data/
17215 ?        Ss     0:00 sheep /mnt/sheepdog/data/
[root@localhost labotd]# pkill sheep
[root@localhost labotd]# ps axf | grep sheepdog
17323 pts/0    S+     0:00              |   |   |          \_ grep
--color=auto sheepdog
```

-pour terminer corosync : **service corosync stop**
(si on arrête corosync avant sheepdog le service sheepdog est automatiquement tué)

4.4 Utilisation de sheepdog :

En plus du deamon sheep, les developpeurs de sheepdog on prévu un utilitaire d'administration appelé **collie**, utilisable depuis n'importe quelle machine du cluster

```
Sheepdog administrator utility
Usage: collie <command> <subcommand> [options]

Available commands:
  vdi create              create an image
  vdi snapshot            create a snapshot
  vdi clone               clone an image
  vdi delete              delete an image
  vdi list                list images
  vdi tree                show images in tree view format
  vdi graph               show images in Graphviz dot format
  vdi object              show object information in the image
  vdi setattr             set a VDI attribute
  vdi getattr             get a VDI attribute
  vdi resize              resize an image
  vdi read                read data from an image
  vdi write               write data to an image
  node list               list nodes
  node info               show information about each node
  cluster info            show cluster information
  cluster format          create a Sheepdog store
  cluster shutdown        stop Sheepdog
  cluster recover         manually recover the cluster
  cluster snapshot        snapshot/restore the cluster
```

-pour voir les machines appartenant au cluster : **collie node list**
-pour voir les informations relatives au cluster : **collier cluster info**
-pour lister toute les images des vm : **collie vdi list**
-pour voir l'etat de l'entrepot de tous les sheep : **collie node info**

```
[labotd@localhost sheepdog]$ collie cluster info
Cluster status: running

Cluster created at Fri Mar  2 17:43:09 2012

Epoch Time           Version
2012-03-02 17:45:29      1 [192.168.1.111:7000, 192.168.1.112:7000]
[labotd@localhost sheepdog]$ collie node list
M   Id   Host:Port         V-Nodes       Zone
-    0   192.168.1.111:7000     64 1862379712
-    1   192.168.1.112:7000     64 1879156928
```

-pour crée une VM d' 1 Go : **qemu-img create sheepdog:*vm_name* nbG**

```
[labotd@localhost data]$ qemu-img create sheepdog:vm1 1G
Formatting 'sheepdog:vm1', fmt=raw size=1073741824
[labotd@localhost data]$ qemu-img create sheepdog:vm2 1G
Formatting 'sheepdog:vm2', fmt=raw size=1073741824
```

ou **collie vdi create *vm_name* nbG**

```
[root@localhost labotd]# collie vdi create test 2G
[root@localhost labotd]# collie vdi create test2 2G
```

-pour convertir une VM qemu/KVM afin d'être utilisé par sheepdog :

cd **/var/lib/libvirt/images**
qemu-img convert vm1.img sheepdog:VM1

-pour booter une vm via sheepdog, il faut tout d'abord modifier la balise « disk » du fichier XML de configuration de la VM : **virsh edit vm1**
(**i** pour mode insertion, **esc** pour sortir du mode insertion, **:wq** pour quitter et sauvegarder les modifications)

```
<disk type='network'  device='disk'>
        <driver name='qemu' type='raw' />
        <source protocol='sheepdog' name='VM1' />
        <target dev='vda' bus='virtio' />
```

ensuite, on peut booter la VM de manière graphique via **virtmanager,** ou en ligne de commande via **qemu-system-x86_64 sheepdog:VM1**

```
[root@localhost labotd]# collie vdi list
  Name        Id    Size    Used  Shared    Creation time   VDI id
  VM1          1  8.0 GB  4.9 GB  0.0 MB 2012-04-01 20:55   f77447
[root@localhost labotd]# collie node info
Id      Size    Used    Use%
 0      96 GB   4.9 GB    5%
 1      96 GB   4.9 GB    5%
 2      96 GB   4.9 GB    5%
Total   288 GB  15 GB     5%

Total virtual image size        8.0 GB
```

-les logs de sheepdog : **/mnt/sheepdog/data/sheep.log**

5 Tests

5.1 Créer un cluster de 2, puis de 3 machines avec corosync :

-on lance corosync sur la machine 192.168.112 :

```
labotd]# tail /var/log/cluster/corosync.log
corosync [MAIN  ] Completed service synchronization, ready to provide service.
corosync [TOTEM ] A processor joined or left the membership and a new membership was formed.
corosync [CPG   ] chosen downlist: sender r(0) ip(192.168.1.112) ; members(old:1 left:0)
corosync [MAIN  ] Completed service synchronization, ready to provide service.
corosync [TOTEM ] A processor joined or left the membership and a new membership was formed.
corosync [CPG   ] chosen downlist: sender r(0) ip(192.168.1.112) ; members(old:1 left:0)
corosync [MAIN  ] Completed service synchronization, ready to provide service.
corosync [TOTEM ] A processor joined or left the membership and a new membership was formed.
corosync [CPG   ] chosen downlist: sender r(0) ip(192.168.1.112) ; members(old:1 left:0)
corosync [MAIN  ] Completed service synchronization, ready to provide service.
```

-on peut observer à l'aide de wireshark sur la machine 192.168.1.111 :

192.168.1.112	226.94.1.1	IGMP	46	V2 Membership Report / Join group 226.94.1.1
192.168.1.112	226.94.1.1	UDP	154	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	IGMP	46	V2 Membership Report / Join group 226.94.1.1

-ensuite, on lance corosync sur 192.168.1.111 : une fois que les deux machines on joint le groupe 226.94.1.1 on peut voir qu'elles communiquent entre elles (rotation du token)

192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	124	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.111	226.94.1.1	IGMP	60	V2 Membership Report / Join group 226.94.1.1
192.168.1.111	226.94.1.1	UDP	154	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	198	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	198	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.111	226.94.1.1	UDP	198	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	226.94.1.1	UDP	198	Source port: hpoms-dps-lstn Destination port: netsupport
AsustekC_25:2f:4	Broadcast	ARP	60	Who has 192.168.1.112? Tell 192.168.1.111
AsustekC_25:2a:2	AsustekC_25:2f:44	ARP	42	192.168.1.112 is at e0:cb:4e:25:2a:23
192.168.1.111	192.168.1.112	UDP	268	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	192.168.1.111	UDP	268	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.111	192.168.1.112	UDP	268	Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.112	192.168.1.111	UDP	268	Source port: hpoms-dps-lstn Destination port: netsupport

```
labotd]# tail /var/log/cluster/corosync.log
corosync [TOTEM ] A processor joined or left the membership and a new membership was formed.
corosync [CPG   ] chosen downlist: sender r(0) ip(192.168.1.111) ; members(old:1 left:0)
corosync [MAIN  ] Completed service synchronization, ready to provide service.
corosync [TOTEM ] A processor failed, forming new configuration.
corosync [TOTEM ] A processor joined or left the membership and a new membership was formed.
corosync [CPG   ] chosen downlist: sender r(0) ip(192.168.1.112) ; members(old:2 left:1)
corosync [MAIN  ] Completed service synchronization, ready to provide service.
```

d'apres l'outil corosync-cfgtool 192.168.1.111 connais 192.168.1.112 (id 1879156928) :

```
[root@localhost labotd]# corosync-cfgtool -s
Printing ring status.
Local node ID 1862379712
RING ID 0
        id      = 192.168.1.111
        status  = ring 0 active with no faults
[root@localhost labotd]# corosync-cfgtool -a 1879156928
192.168.1.112
```

-Membership report (IGMP) des 2 machines a l'adresse multicast 226.94.1.1 :

192.168.1.112	226.94.1.1	IGMP	46 V2 Membership Report / Join group 226.94.1.1
192.168.1.112	226.94.1.1	IGMP	46 V2 Membership Report / Join group 226.94.1.1
192.168.1.112	226.94.1.1	IGMP	46 V2 Membership Report / Join group 226.94.1.1
192.168.1.111	226.94.1.1	IGMP	60 V2 Membership Report / Join group 226.94.1.1
192.168.1.111	226.94.1.1	IGMP	60 V2 Membership Report / Join group 226.94.1.1
192.168.1.111	226.94.1.1	IGMP	60 V2 Membership Report / Join group 226.94.1.1
192.168.1.112	226.94.1.1	IGMP	46 V2 Membership Report / Join group 226.94.1.1
192.168.1.112	226.94.1.1	IGMP	46 V2 Membership Report / Join group 226.94.1.1
192.168.1.111	226.94.1.1	IGMP	60 V2 Membership Report / Join group 226.94.1.1
192.168.1.112	226.94.1.1	IGMP	46 V2 Membership Report / Join group 226.94.1.1
192.168.1.112	226.94.1.1	IGMP	46 V2 Membership Report / Join group 226.94.1.1
192.168.1.111	226.94.1.1	IGMP	60 V2 Membership Report / Join group 226.94.1.1
192.168.1.112	226.94.1.1	IGMP	46 V2 Membership Report / Join group 226.94.1.1
192.168.1.111	226.94.1.1	IGMP	60 V2 Membership Report / Join group 226.94.1.1
192.168.1.111	226.94.1.1	IGMP	60 V2 Membership Report / Join group 226.94.1.1
192.168.1.111	226.94.1.1	IGMP	60 V2 Membership Report / Join group 226.94.1.1
192.168.1.112	226.94.1.1	IGMP	46 V2 Membership Report / Join group 226.94.1.1
192.168.1.112	226.94.1.1	IGMP	46 V2 Membership Report / Join group 226.94.1.1
192.168.1.112	226.94.1.1	IGMP	46 V2 Membership Report / Join group 226.94.1.1
192.168.1.111	226.94.1.1	IGMP	60 V2 Membership Report / Join group 226.94.1.1
192.168.1.112	226.94.1.1	IGMP	46 V2 Membership Report / Join group 226.94.1.1
192.168.1.112	226.94.1.1	IGMP	46 V2 Membership Report / Join group 226.94.1.1
192.168.1.112	226.94.1.1	IGMP	46 V2 Membership Report / Join group 226.94.1.1

-avec trois machines le comportement est identique :

192.168.1.111	226.94.1.1	UDP	124 Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.111	192.168.1.112	UDP	112 Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.114	192.168.1.111	UDP	112 Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.111	192.168.1.112	UDP	112 Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.114	192.168.1.111	UDP	112 Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.111	226.94.1.1	UDP	124 Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.111	192.168.1.112	UDP	112 Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.114	192.168.1.111	UDP	112 Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.111	192.168.1.112	UDP	112 Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.114	192.168.1.111	UDP	112 Source port: hpoms-dps-lstn Destination port: netsupport
192.168.1.111	226.94.1.1	UDP	124 Source port: hpoms-dps-lstn Destination port: netsupport

On peut voir quatre rotations du token sur l'anneau (192.168.1.111 est la première machine sur l'anneau et 192.168.1.114 est la dernière)

Lorsqu'une des machines stop corosync, il envoie un Leave (IGMP), Le membership est alors mis à jour.

5.2 Créer un cluster sheepdog de 2, puis de 3 machines :

en lancant sheepdog sur 192.168.1.111 et sur 192.168.1.112 :
-on peut observer depuis sur les deux machines :

```
[root@localhost labotd]# corosync-cpgtool
Group Name             PID         Node ID
sheepdog
                      2250      1862379712 (192.168.1.111)
                     11271      1879156928 (192.168.1.112)
```

```
cdrv_cpg_deliver(449) 1
sd_join_handler(1223) join ip: 192.168.1.112, port: 7000
sd_join_handler(1225) [0] ip: 192.168.1.111, port: 7000
sd_join_handler(1225) [1] ip: 192.168.1.112, port: 7000
sd_join_handler(1237) allow new confchg 0x13190f0
cpg_event_done(950) 0x13190f0
__sd_join_done(834) l ip: 192.168.1.111:7000, port: 7000
update_cluster_info(530) status = 2, epoch = 0, 0, 1
update_cluster_info(607) l ip: 192.168.1.111:7000, port: 7000
update_cluster_info(607)   ip: 192.168.1.112:7000, port: 7000
cpg event done(969) free 0x13190f0
```

on peut voir, dans sheep.log les messages join envoyés via cpg et la mise a jour des informations du cluster.

```
[labotd@localhost sheepdog]$ collie cluster info
Cluster status: running

Cluster created at Fri Mar  2 17:43:09 2012

Epoch Time           Version
2012-03-02 17:45:29      1 [192.168.1.111:7000, 192.168.1.112:7000]
[labotd@localhost sheepdog]$ collie node list
M   Id   Host:Port         V-Nodes       Zone
-    0   192.168.1.111:7000     64 1862379712
-    1   192.168.1.112:7000     64 1879156928
```

-après un certain temps (variable), on peut constater des pertes d'intégrité du cluster :

```
[labotd@localhost ~]$ collie cluster info
Cluster status: IO has halted as there are too few living nodes

Cluster created at Sun Mar  4 16:44:35 2012

Epoch Time           Version
2012-03-04 18:14:48      7 [192.168.1.111:7000, 192.168.1.112:7000]
2012-03-04 18:14:48      6 [192.168.1.111:7000]
2012-03-04 17:26:06      5 [192.168.1.111:7000, 192.168.1.112:7000]
2012-03-04 17:26:06      4 [192.168.1.111:7000]
2012-03-04 17:00:34      3 [192.168.1.111:7000, 192.168.1.112:7000]
2012-03-04 17:00:34      2 [192.168.1.111:7000]
2012-03-04 16:46:57      1 [192.168.1.111:7000, 192.168.1.112:7000]
```

a chaque fois qu'on observe une perte d'intégrité il suffit généralement de redémarrer le service sheep sur l'un des deux node (n'importe lequel) du cluster pour en restaurer l'integrité (ex : epoch 3,5,7,13,15).

```
 17 []
 16 [192.168.1.112:7000]
 15 [192.168.1.111:7000, 192.168.1.112:7000]
 14 [192.168.1.112:7000]
 13 [192.168.1.111:7000, 192.168.1.112:7000]
```

Si, on obtient une perte d'intégrité grave (epoch 17 [ ]), redémarrer sheep n'a aucun effet, il faut alors effacer le contenu des volumes sheepdog (perte des données !) et reformater le cluster :

-tuer tout les sheep avec pkill sheep sur chaque noeud (collie cluster shutdown bloque dans ce cas...)
-effacer le directory sheep dog : **# rm -rf /mnt/sheepdog/data**
-relancer sheepdog sur les machines
-reformater le cluster depuis une machine : **collie cluster format -c 2**

j'obtiens les mêmes résultats avec un cluster de trois machines.

A chaque fois que j'ai observé une perte d'intégrité du cluster avec l'utilitaire collie, j'ai vérifié également l'intégriter du cluster du coté de corosync (**corosync-objctl | grep members**) et tout était correct...

Apparemment, si **j'utilise collie uniquement** sur la machine sur laquelle j'ai **lancé le premier process sheep** et la commande **collie cluster format,** et que je reste sur la même machine,je ne constate pas de perte d'intégrité du cluster.
Le fait de lancer **collie**, d'acceder au repertoire **/mnt/sheepdog/data** (via des commandes comme df ou même ls) sur les autres machines du cluster, ou même de simplement d'utiliser une autre machine (unlock de la mise en veille) corrompt l'intégrité du cluster en mettant à jour l'epoch (version de membership) de facon érroné.
Il semble donc nécessaire de prévoir une machine dédié a l'administration ou l'on lance, stop et gére le cluster sheepdog.

5-3 création et stockage d'une vm (vdi) avec un cluster de trois machines :

avec collie cluster format **-c 2** :

```
[root@localhost labotd]# collie node info
Id       Size     Used     Use%
 0       96 GB    0.0 MB     0%
 1       96 GB    0.0 MB     0%
 2       96 GB    0.0 MB     0%
Total    288 GB   0.0 MB     0%

Total virtual image size         0.0 MB
```

cd **/var/lib/libvirt/images**
qemu-img convert vm1.img sheepdog:VM1

Cette commande prend un bon moment pour terminer son exécution car le block driver de qemu va segmenter la partie utilisée de l'image de la vm en objet de taille fixe (a peu prés 4mb par default), et sheepdog va répartir les deux copies sur le reseau.

Sheep.log pendant l'exécution de la commande :

```
do_io_request(865) 3, f7744700000617 , 1
forward_write_obj_req(349) f7744700000617
get_sheep_fd(848) 1, 17
get_sheep_fd(851) using the cached fd 17
do_local_io(748) 3, f7744700000617 , 1
forward_write_obj_req(465) f7744700000617 0
queue_request(213) 3
```

lorsque la commande a terminé son exécution :

```
[root@localhost labotd]# collie vdi list
  Name        Id    Size    Used  Shared    Creation time   VDI id
  VM1          1  8.0 GB  4.2 GB  0.0 MB 2012-04-01 15:57   f77447
[root@localhost labotd]# collie node info
Id       Size     Used     Use%
 0       96 GB    2.8 GB     2%
 1       96 GB    2.9 GB     2%
 2       96 GB    2.7 GB     2%
Total    288 GB   8.4 GB     2%

Total virtual image size         8.0 GB
```

sheepdog a répartit 4.2 G * 2 sur l'ensemble du cluster, si on regarde le repertoire **/mnt/sheepdog/data/obj/00000001**, on a 720 * 4.19 mb a peu prés 3 Gb

```
-rw-r----- 1 root root 4194304  1 avril 17:16 00f7744700000557
-rw-r----- 1 root root 4194304  1 avril 17:16 00f7744700000558
-rw-r----- 1 root root 4194304  1 avril 17:16 00f7744700000559
-rw-r----- 1 root root 4194304  1 avril 17:16 00f774470000055a
-rw-r----- 1 root root 4194304  1 avril 17:16 00f774470000055b
-rw-r----- 1 root root 4194304  1 avril 17:16 00f774470000055c
-rw-r----- 1 root root 4194304  1 avril 17:16 00f774470000060d
-rw-r----- 1 root root 4194304  1 avril 17:17 00f774470000060f
-rw-r----- 1 root root 4194304  1 avril 17:17 00f7744700000610
-rw-r----- 1 root root 4194304  1 avril 17:17 00f7744700000611
-rw-r----- 1 root root 4194304  1 avril 17:17 00f7744700000612
-rw-r----- 1 root root 4194304  1 avril 17:17 00f7744700000614
-rw-r----- 1 root root 4194304  1 avril 17:17 00f7744700000616
-rw-r----- 1 root root 4194304  1 avril 17:17 00f7744700000617
-rw-r----- 1 root root 4198968  1 avril 17:17 80f7744700000000
[root@localhost labotd]# ls -la /mnt/sheepdog/data/obj/00000001/ | wc -l
720
```

chaque objet est codé sur **64 bit** :

0 (data object) **0** (reserved) **f77447** (vdi id) les **32 derniers bits** représentent l'index du disque virtuel.
8 (vdi object) **0** (reserved) **f77447** (vdi id) les **32 derniers bits a 0** pour un objet vdi (meta-donnée de la VM).

avec collie cluster format **-c 3** :

```
[root@localhost labotd]# collie vdi list
  Name        Id    Size    Used  Shared    Creation time   VDI id
  VM1          1  8.0 GB  4.9 GB  0.0 MB 2012-04-01 20:55   f77447
[root@localhost labotd]# collie node info
Id      Size    Used    Use%
 0      96 GB   4.9 GB    5%
 1      96 GB   4.9 GB    5%
 2      96 GB   4.9 GB    5%
Total   288 GB  15 GB     5%

Total virtual image size        8.0 GB
```

sheepdog a bien répartit, les 4.9 Gb utilisé sur les trois machines du cluster

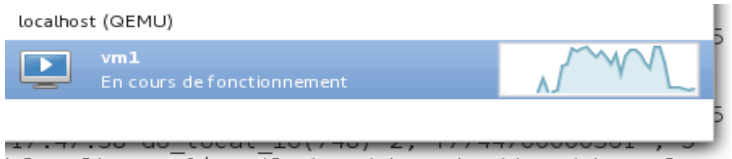
sur une des machines on peut observer :

```
[root@localhost labotd]# ls -la /mnt/sheepdog/data/obj/00000001/ | wc -l
1268
 [root@localhost labotd]# du -h /mnt/sheepdog/data/
 5,0G    /mnt/sheepdog/data/obj/00000001
 5,0G    /mnt/sheepdog/data/obj
 4,0K    /mnt/sheepdog/data/journal
 8,0K    /mnt/sheepdog/data/epoch
 4,0K    /mnt/sheepdog/data/cache
 5,4G    /mnt/sheepdog/data/
```

1268 * 4 Mb = 5 Gb

```
[root@localhost labotd]# collie vdi object VM1
Looking for the inode object 0xf77447 with 3 nodes

192.168.1.111:7000 has the object (should be 3 copies)
192.168.1.112:7000 has the object (should be 3 copies)
192.168.1.114:7000 has the object (should be 3 copies)
```

En démarrant VM1 sur 192.168.1.111 on peut voir dans sheep.log les requêtes i/o sur les objets constituant la VM :




```
@localhost ~]$ tail /mnt/sheepdog/data/sheep.log
17:47:42 do_local_io(748) 2, f77447000002d7 , 5
17:47:42 queue_request(213) 2
17:47:42 do_io_request(865) 2, f77447000002cd , 5
17:47:42 do_local_io(748) 2, f77447000002cd , 5
17:47:42 queue_request(213) 2
17:47:42 do_io_request(865) 2, f77447000002cd , 5
17:47:42 do_local_io(748) 2, f77447000002cd , 5
17:47:42 queue_request(213) 2
17:47:42 do_io_request(865) 2, f77447000002d7 , 5
17:47:42 do_local_io(748) 2, f77447000002d7 , 5
```

5-4 observer la redondance, après l'ajout d'une 3ème machines (cluster de 2 machines, 1 vm) :

sur 192.168.1.111 : collie cluster format -c 2

```
[root@localhost labotd]# cd /var/lib/libvirt/images/
[root@localhost images]# time qemu-img convert vm1.img sheepdog:VM1

real    85m58.308s
user    0m1.296s
sys     0m10.628s
[root@localhost images]#
```

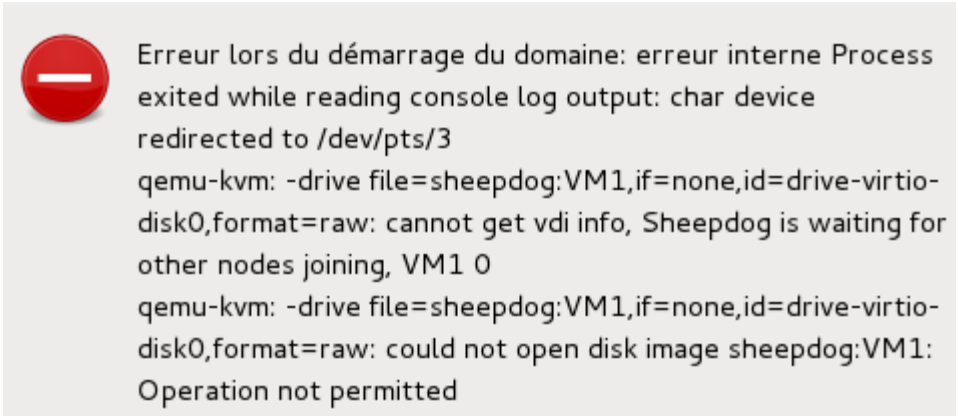
Le simple fait, de switcher sur 192.168.1.114 fait totalement perdre l'intégrité du cluster (192.168.1.112 quitte subitement le cluster suivi de près par 192.168.1.111)

```
[root@localhost labotd]# collie node info
There are no active sheep daemons
[root@localhost labotd]# corosync-cpgtool
Group Name             PID         Node ID

[root@localhost labotd]# collie cluster info
Cluster status: Waiting for other nodes to join cluster

3 []
2 [192.168.1.111:7000]
1 [192.168.1.111:7000, 192.168.1.112:7000]
```

on ne peut plus du tout accéder a notre VM :

```
[root@localhost labotd]# collie vdi list
  Name        Id    Size    Used  Shared    Creation time   VDI id
Failed to read object 80f7744700000000 Waiting for other nodes to join cluster
Failed to read inode header
[root@localhost labotd]# collie node info
Id      Size    Used    Use%
Cannot get information from any nodes
```

quand on essaye de la demarrer, on a l'erreur suivante :



pourtant les daemons sheep tournent sur toutes les machines du cluster....
et les machines sont toujours membres de corosync :

```
runtime.totem.pg.mrp.srp.members.16777343.ip=r(0) ip(127.0.0.1)
runtime.totem.pg.mrp.srp.members.16777343.join_count=1
runtime.totem.pg.mrp.srp.members.16777343.status=joined
runtime.totem.pg.mrp.srp.members.1879156928.ip=r(0) ip(192.168.1.112)
runtime.totem.pg.mrp.srp.members.1879156928.join_count=18
runtime.totem.pg.mrp.srp.members.1879156928.status=joined
runtime.totem.pg.mrp.srp.members.1912711360.ip=r(0) ip(192.168.1.114)
runtime.totem.pg.mrp.srp.members.1912711360.join_count=17
runtime.totem.pg.mrp.srp.members.1912711360.status=joined
```

J'ai recommencé l'opération plusieurs fois sans succés.... le fait de switcher sur 192.168.1.114 provoque un sd_leave de 192.168.1.112 ou de 192.168.1.111 (voir les deux), sheepdog essaye alors de faire un recovery des données mais échoue (failed to read epoch ou invalid epoch)

5-5 observer la redondance, apres la perte d'une machine (cluster de 3 machines, 1 vm) :

même scénario que pour l'ajout d'une machine, perte soudaine d'intégrité du cluster, entrainant la corruption des données stockées....

```
23:00:09      4 []
22:58:37      3 [192.168.1.111:7000]
22:58:37      2 [192.168.1.111:7000, 192.168.1.114:7000]
20:28:52      1 [192.168.1.111:7000, 192.168.1.112:7000, 192.168.1.114:7000]
```

6 Difficultés rencontrées

-problème lors des installations de fedora 16, j'ai du relancer pour certaine plusieur fois l'install (ca bloquait en cours...), une des 4 machines a d'ailleurs apparament rendu l'ame pendant l'une d'entre elle...

J'ai aussi passé pas mal de temps a me documenter sur sheepdog et corosync, a analyser un peu le code source des versions 2.3.0-1 et 3.0-0, ainsi que les différent feedback des utilisateur/testeur (bug report ou les patch proposé via http://lists.wpkg.org/pipermail/sheepdog/)

problème récurent :

-réponse de **collie node list** ou/et de **collie cluster info** fausse,vide, ou erreur :
('**there are no active sheep daemons**' '**waiting other nodes to join cluster**')

-**erreur** dans sheep.log: '**failed to read epoch**' ou '**joining node have invalid epoch**'

-**conséquence :** perte d'intégrité du cluster

Sur toute les versions testées (2.3.0-1, 3.0-0 et 3.0-57):

-essayer de redemarrer sheepdog sur un noeud (voir tous si le problème est toujours présent) :
pkill sheep (si collie cluster shutdown ne marche pas) et **sheep /mnt/sheepdog/data**

si le problème persiste , il faut alors nettoyer le cluster sheepdog 'a la main' (et donc perdre toutes les données):

-tuer tout les sheep avec **pkill sheep** (**collie cluster shutdown** ne marche pas dans ce cas...)
-effacer le directory sheep dog : **# rm -rf /mnt/sheepdog/data/***
-relancer sheepdog sur les machines via **sheep /mnt/sheepdog/data**
-reformater le cluster depuis une machine : **collie cluster format -c 2**

j'ai passé la plus grande partie de mon temps à investiguer sur ce problème de perte d'intégrité présent sur les toute versions testées (pratiquement systématique), sans grand succés, hormis la solution ci dessus.

Apparemment, si **j'utilise collie uniquement** sur le noeud sur lequel j'ai **lancé le service sheep en premier** et sur lequel j'ai lancé la commande **collie cluster format -c 2 (master node ?)**, je ne constate plus de perte d'intégrité du cluster.
Le fait de lancer collie sur les autres noeud (ou même d'acceder au repertoire /mnt/sheepdog/data), corrompt l'intégrité du cluster en mettant à jour l'epoch (version de membership).

7 Conclusion

Dans le cadre de ce projet de semestre, j'ai pu étudier différentes technologies intérréssantes comme corosync pour la gestion d'un groupe de machine, ou comme QEMU/KVM pour la virtualisation sous linux.

J'ai pu aussi comprendre et tester la partie concernant le stockage distribué des disques virtuels sur un cluster de quelques machines.

Par contre et au vue des problèmes rencontrés j'ai du mal à comprendre les causes des fréquentes pertes d'intégrité expérimentées lors de mes tests...

Sheepdog est en définitive une solution innovante, intérréssante à étudier mais encore jeune , en plein developpement.Je ne conseillerai en aucun cas son utilisation dans un environnement en production.

De plus d'aprés mes tests l'ajout ou la perte d'une machine au sein du cluster provoque une perte d'intégrité entraînant la corruption des données stockées.
En effet, en cas de perte d'intégrité du cluster, toutes les images de disques virtuel ne sont plus utilisable (on ne peut plus démarrer les machines virtuelles) ce qui représente un « single point of failure », ce que les developpeurs de sheepdog veulent justement éviter.

Par contre, le developpement de sheepdog est à suivre de près, je pense que dans quelque années (voir moins) cette solution sera suffisament mature pour envisager son utilisation en production...

8 Annexes

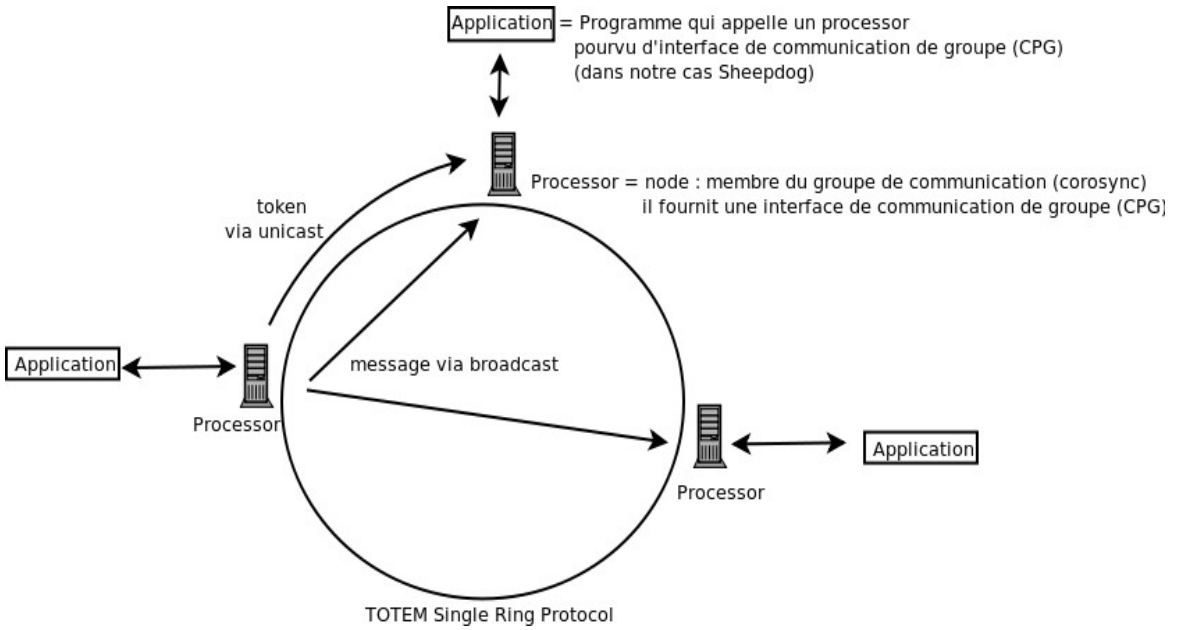
8.1 Fonctionnement de Totem SRP :

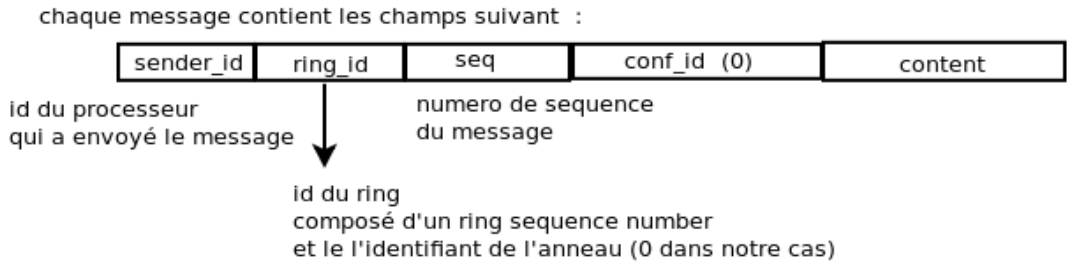
Totem SRP/RRP est en fait, un groupe de protocole constitué de :

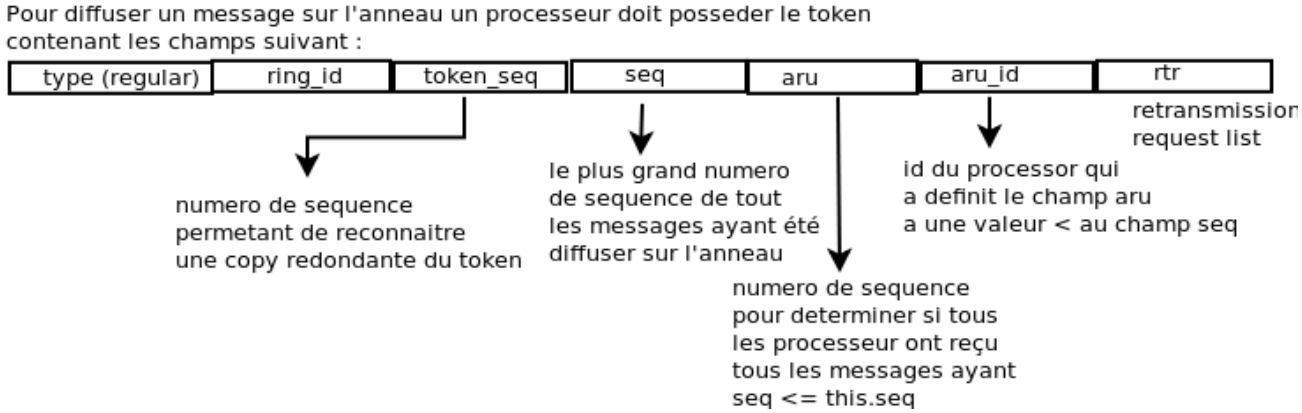
-**Totem Ordering Protocol** (**OP**) pour l'ordonnancement et le bon acheminement de messages
-**Membership Protocol** (**MP**) pour la gestion dynamique de l'appartenence au cluster
- **Recovery Protocol** (**RP**)

8.1.1 Totem Ordering Protocol :

Ce protocole est une implémentation logique du protocole Token ring, chaque noeud transmet périodiquement (toute les 300 millisecondes par defaut) un jeton (regular token) a son voisin sur l'anneau. Dès qu'un noeud posséde le token il va alors transmettre un ou plusieurs messages (Broadcast) au autres noeuds de l'anneau

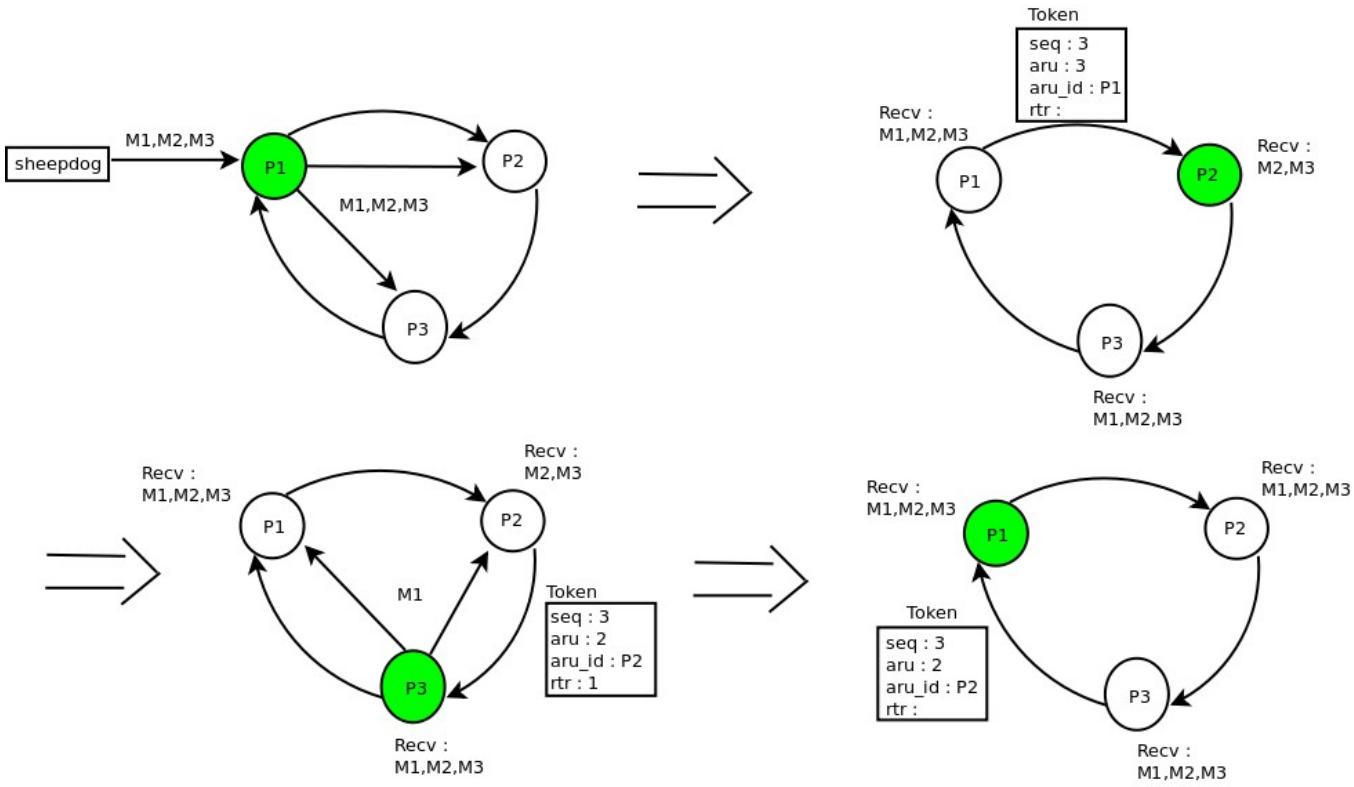






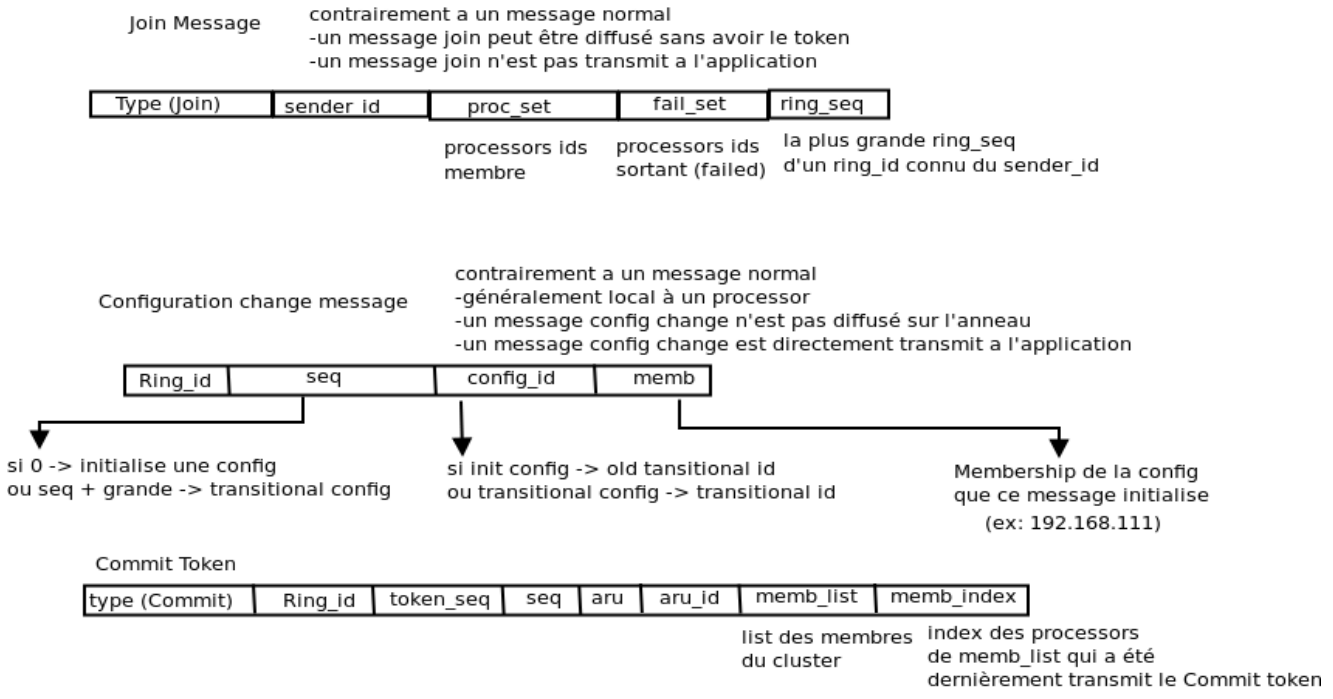
Quand sheepdog désire envoyer un message a une machine du cluster, il l'envoie a totem (via cpg) qui stocke le message dans sa send_queue (variable locale a un processor).
La prochaine fois que le processor reçoit le token, il enlève les messages de sa queue et les diffuse dans le bon ordre. Totem inclus dans chaque message un numero de sequence (seq), le token s'occupe du numero de séquence du dernier message diffusé sur l'anneau. Pour chaque message diffusé le processor (sender_id) incrémente le champ seq du message. Après avoir diffusé l'intégralité de ses messages, le processor copie seq dans le token et le passe a son voisin. Ce numero de sequence, inclus dans chaque message permet de garantir la transmission de ceux-ci dans le bon ordre.

Exemple ci-dessous de la retransmission d'un message :

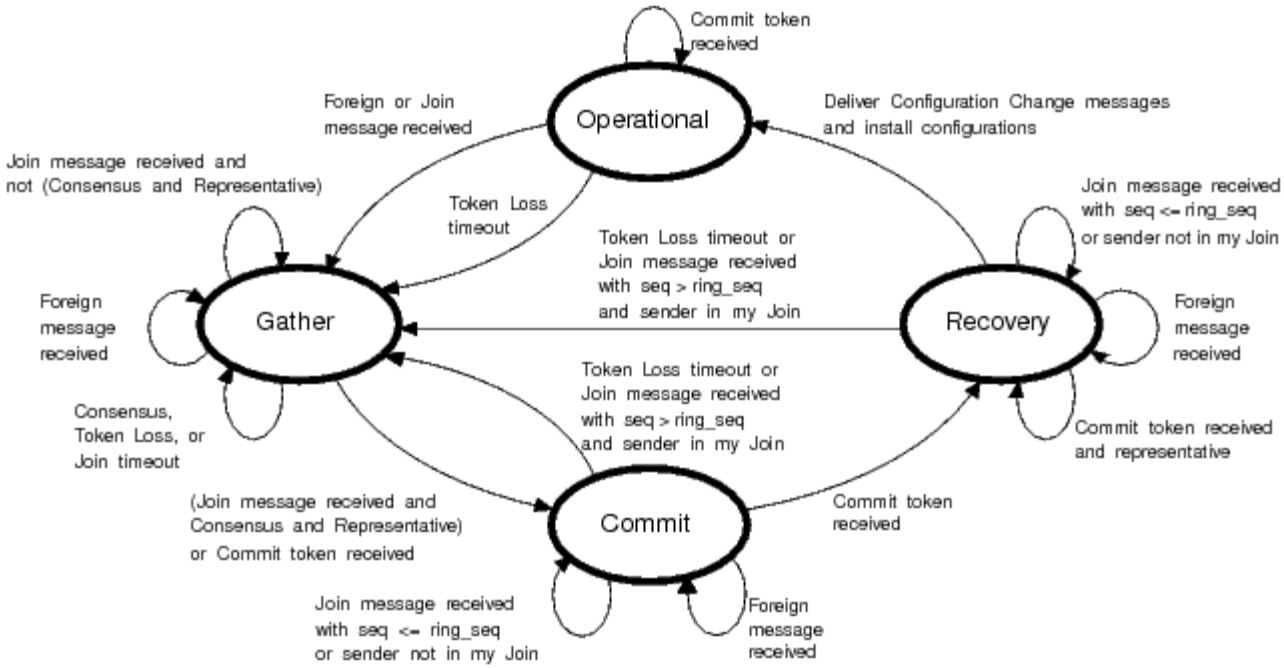


The TOTEM ORDERING PROTOCOL (OP)

8.1.2 Membership Protocol :



Le Membership protocole peut être représenté par l'automate finis suivant :



Etat Operational : les messages sont diffusés et délivrés dans le bon ordre via Totem Ordering Protocol si on recoit un message join ou Foreign (message qui a été diffusé par un processor qui n'est pas membre du cluster) on passe a l'etat gather (activation du membership Protocol).

Etat Gather : Dans cet état un processor rassemble les informations des processors membres et des failed processor (ceux qui ne sont plus membre) et diffuse ces informations sous la forme de messages join.

Etat Commit : le Commit Token fait une premier tour sur le nouvel anneau proposé et confirme que tous les membres présent dans memb_list sont bien membre du cluster. Il collecte aussi les informations utiles pour gérer correctement les messages de l'ancien anneau qui doivent être encore être retransmit quand le membership protocol a été activé.Quand le processor reçoit pour la deuxième fois le Commit token il passe dans l'etat recovery.
Lors de son deuxième passage sur l'anneau le Commit tokens transmet les informations collectées lors de son premier passage.

Etat Recovery : Quand le processor (Representative) reçoit pour la troisième fois le Commit token (après sa deuxième rotation), il convertit le Commit token en un regular token pour le nouvel anneau (en remplaçant les champs memb_list et memb_index par rtr). A cet instant, le nouvel anneau est formé mais pas encore installé et l'exécution du recovery protocol commence.

```
runtime.totem.pg.mrp.srp.memb_commit_token_tx=68
runtime.totem.pg.mrp.srp.memb_commit_token_rx=68
runtime.totem.pg.mrp.srp.token_hold_cancel_tx=10
runtime.totem.pg.mrp.srp.token_hold_cancel_rx=16
runtime.totem.pg.mrp.srp.operational_entered=34
runtime.totem.pg.mrp.srp.operational_token_lost=13
runtime.totem.pg.mrp.srp.gather_entered=75
runtime.totem.pg.mrp.srp.gather_token_lost=0
runtime.totem.pg.mrp.srp.commit_entered=34
runtime.totem.pg.mrp.srp.commit_token_lost=0
runtime.totem.pg.mrp.srp.recovery_entered=34
runtime.totem.pg.mrp.srp.recovery_token_lost=0
runtime.totem.pg.mrp.srp.consensus_timeouts=25
```

Pour des explications plus détaillées sur le protocole totem SRP/RRP voir les documents en annexe (**tocs.ps** et **Totem_protocol.pdf**)

options de configuration de totem :

Ci-dessous, quelque options pour configurer la partie totem dans corosync.conf :

token This timeout specifies in milliseconds until a token loss is declared after not receiving a token. This is the time spent detecting a failure of a processor in the current configuration. Reforming a new configuration takes about 50 milliseconds in addition to this timeout.

The default is 1000 milliseconds.

token_retransmit
This timeout specifies in milliseconds after how long before receiving a token the token is retransmitted. This will be automatically calculated if token is modified. It is not recommended to alter this value without guidance from the corosync community.

The default is 238 milliseconds.

hold This timeout specifies in milliseconds how long the token should be held by the representative when the protocol is under low utilization. It is not recommended to alter this value without guidance from the corosync community.

The default is 180 milliseconds.

token_retransmits_before_loss_const
This value identifies how many token retransmits should be attempted before forming a new configuration. If this value is set, retransmit and hold will be automatically calculated from retransmits_before_loss and token.

The default is 4 retransmissions.

join This timeout specifies in milliseconds how long to wait for join messages in the membership protocol.

The default is 50 milliseconds.

send_join
This timeout specifies in milliseconds an upper range between 0 and send_join to wait before sending a join message. For configurations with less then 32 nodes, this parameter is not necessary. For larger rings, this parameter is necessary to ensure the NIC is not overflowed with join messages on formation of a new ring. A reasonable value for large rings (128 nodes) would be 80msec. Other timer values must also change if this value is changed. Seek advice from the corosync mailing list if trying to run larger configurations.

The default is 0 milliseconds.

consensus
This timeout specifies in milliseconds how long to wait for consensus to be achieved before starting a new round of membership configuration. The minimum value for consensus must be 1.2 * token. This value will be automatically calculated at 1.2 * token if the user doesn't specify a consensus value.

The default is 1200 milliseconds.

merge This timeout specifies in milliseconds how long to wait before checking for a partition when no multicast traffic is being sent. If multicast traffic is being sent, the merge detection happens automatically as a function of the protocol.

The default is 200 milliseconds.

downcheck
This timeout specifies in milliseconds how long to wait before checking that a network interface is back up after it has been downed.

The default is 1000 millseconds.

fail_to_recv_const
This constant specifies how many rotations of the token without receiving any of the messages when messages should be received may occur before a new configuration is formed.

The default is 50 failures to receive a message.

seqno_unchanged_const
This constant specifies how many rotations of the token without any multicast traffic should occur before the merge detection timeout is started.

The default is 30 rotations.

heartbeat_failures_allowed
[HeartBeating mechanism] Configures the optional HeartBeating mechanism for faster failure detection. Keep in mind that engaging this mechanism in lossy networks could cause faulty loss declaration as the mechanism relies on the network for heartbeating.

So as a rule of thumb use this mechanism if you require improved failure in low to medium utilized networks.

This constant specifies the number of heartbeat failures the system should tolerate before declaring heartbeat failure e.g 3. Also if this value is not set or is 0 then the heartbeat mechanism is not engaged in the system and token rotation is the method of failure detection

The default is 0 (disabled).

max_network_delay
[HeartBeating mechanism] This constant specifies in milliseconds the approximate delay that your network takes to transport one packet from one machine to another. This value is to be set by system engineers and please dont change if not sure as this effects the failure detection mechanism using heartbeat.

The default is 50 milliseconds.

8.2 CPG service API :

cpg (Closed Process Group), est l'interface C qui a été choisit par les developpeurs de sheepdog pour communiquer avec corosync

exemple :

```
cs_error_t cpg_mcast_joined ( cpg_handle_t         handle,
                              cpg_guarantee_t      guarantee,
                              const struct iovec * iovec,
                              unsigned int         iov_len
                            )
```

Multicast to groups joined with cpg_join.

Parameters:

handle	
guarantee	
iovec	This iovec will be multicasted to all groups joined with the cpg_join interface for handle.
iov_len	

Definition at line 771 of file cpg.c.

References coroipcc_msg_send_reply_receive(), CS_OK, guarantee, and MESSAGE_REQ_CPG_MCAST.

```c
static int send_message(cpg_handle_t handle, struct message_header *msg)
{
    struct iovec iov;
    int ret;

    iov.iov_base = msg;
    iov.iov_len = msg->msg_length;
retry:
    ret = cpg_mcast_joined(handle, CPG_TYPE_AGREED, &iov, 1);
    switch (ret) {
    case CS_OK:
        break;
    case CS_ERR_TRY_AGAIN:
        dprintf("failed to send message. try again\n");
        sleep(1);
        goto retry;
    default:
        eprintf("failed to send message, %d\n", ret);
        return -1;
    }
    return 0;
```

The closed process group API

Functions

cs_error_t	**cpg_initialize** (cpg_handle_t *handle, cpg_callbacks_t *callbacks) Create a new cpg connection.
cs_error_t	**cpg_model_initialize** (cpg_handle_t *handle, cpg_model_t model, cpg_model_data_t *model_data, void *context) Create a new cpg connection, initialize with model.
cs_error_t	**cpg_finalize** (cpg_handle_t handle) Close the cpg handle.
cs_error_t	**cpg_fd_get** (cpg_handle_t handle, int *fd) Get a file descriptor on which to poll.
cs_error_t	**cpg_context_get** (cpg_handle_t handle, void **context) Get contexts for a CPG handle.
cs_error_t	**cpg_context_set** (cpg_handle_t handle, void *context) Set contexts for a CPG handle.
cs_error_t	**cpg_dispatch** (cpg_handle_t handle, cs_dispatch_flags_t dispatch_types) Dispatch messages and configuration changes.
cs_error_t	**cpg_join** (cpg_handle_t handle, const struct cpg_name *group) Join one or more groups.
cs_error_t	**cpg_leave** (cpg_handle_t handle, const struct cpg_name *group) Leave one or more groups.
cs_error_t	**cpg_membership_get** (cpg_handle_t handle, struct cpg_name *group_name, struct cpg_address *member_list, int *member_list_entries) Get membership information from cpg.
cs_error_t	**cpg_local_get** (cpg_handle_t handle, unsigned int *local_nodeid)
cs_error_t	**cpg_flow_control_state_get** (cpg_handle_t handle, cpg_flow_control_state_t *flow_control_state)
cs_error_t	**cpg_zcb_alloc** (cpg_handle_t handle, size_t size, void **buffer)
cs_error_t	**cpg_zcb_free** (cpg_handle_t handle, void *buffer)
cs_error_t	**cpg_zcb_mcast_joined** (cpg_handle_t handle, cpg_guarantee_t guarantee, void *msg, size_t msg_len)
cs_error_t	**cpg_mcast_joined** (cpg_handle_t handle, cpg_guarantee_t guarantee, const struct iovec *iovec, unsigned int iov_len) Multicast to groups joined with cpg_join.
cs_error_t	**cpg_iteration_initialize** (cpg_handle_t handle, cpg_iteration_type_t iteration_type, const struct cpg_name *group, cpg_iteration_handle_t *cpg_iteration_handle) Iteration.
cs_error_t	**cpg_iteration_next** (cpg_iteration_handle_t handle, struct cpg_iteration_description_t *description)
cs_error_t	**cpg_iteration_finalize** (cpg_iteration_handle_t handle)

Class CPG (en Pearl) :

```
def fileno(self):
    """Return a file descriptor that can be selected on."""
    if self.m_handle is None:
        raise Error, 'Service not started.'
    return _cpg.fd_get(self.m_handle)

def _dispatch(self, type=DISPATCH_ALL):
    """Dispatch events."""
    self.m_dispatch = True
    _cpg.dispatch(self.m_handle, _cpg.DISPATCH_ALL)
    self.m_dispatch = False
    if self.m_stop:
        self.stop()

def members(self):
    """Return a list of group members.

    The return value is a list of 3-tuples (nodeid, pid, reason).
    """
    if self.m_handle is None:
        raise Error, 'Service not started.'
    name, members = _cpg.membership_get(self.m_handle)
    return members

def send_message(self, message, guarantee=TYPE_AGREED):
    """Send a message to all group members."""
    if self.m_handle is None:
        raise Error, 'Service not started.'
    _cpg.mcast_joined(self.m_handle, guarantee, message)

def _deliver_fn(self, name, addr, message):
    """INTERNAL: message delivery callback."""
    self.message_delivered(addr, message)

def _confchg_fn(self, name, members, left, joined):
    """INTERNAL: configuration change callback."""
    self.configuration_changed(members, left, joined)

def message_delivered(self, addr, message):
    """Callback that is raised when a message is delivered."""

def configuration_changed(self, members, left, joined):
    """Callback that is raised when a configuration change happens."""
```

8.3 Stockage des objets (object storage) :

Chaque objet est codé sur 64 bits.

type d'objets :

-1/ data object (id : 0):

-Les volumes (images de VMs) sont segmentés par le QEMU block driver en objets de taille fixe (4 Mb par default)

-Une table d'allocation est stocké comme VDI object

-2/vdi object (id : 8):

Contient les meta données des volumes (nom, taille, date de création, ID du data object appartenant au vdi)

Ou sont stockés les objets :

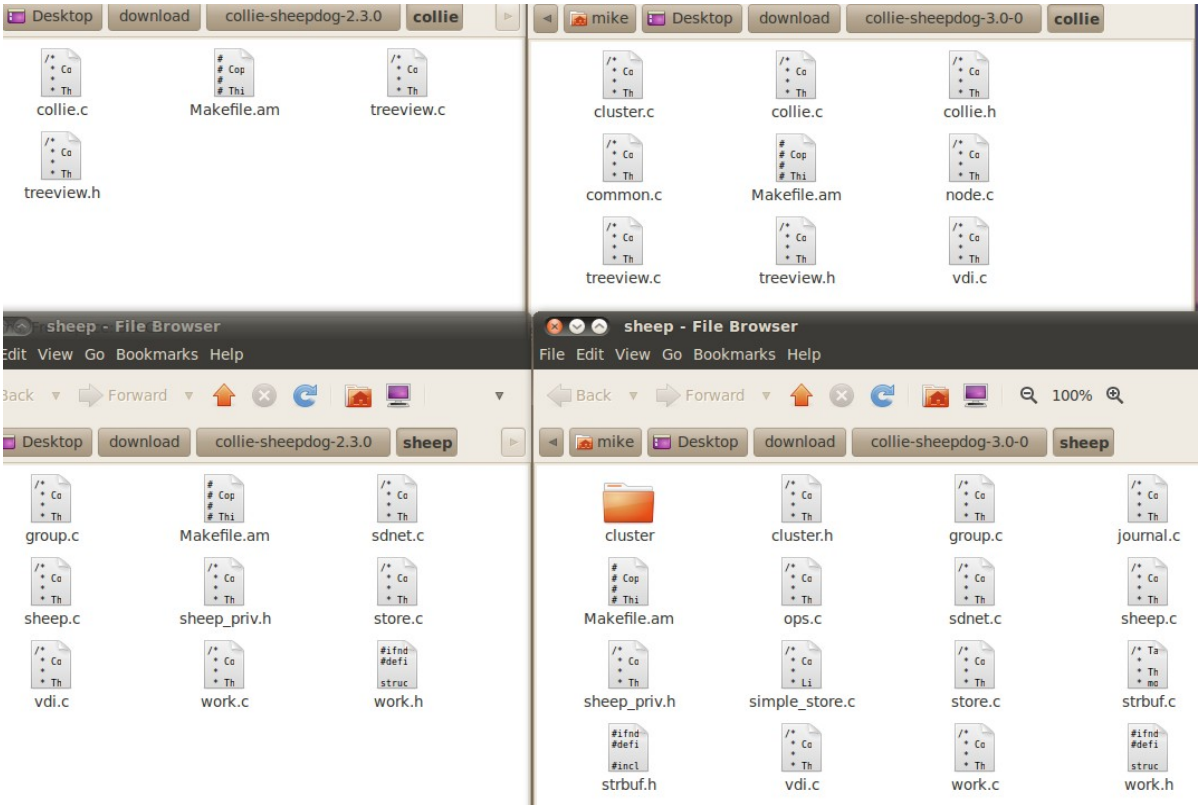
Le c**onsistent hashing**" est utilisé pour déterminer ou (quel node) va être stocké les objets .

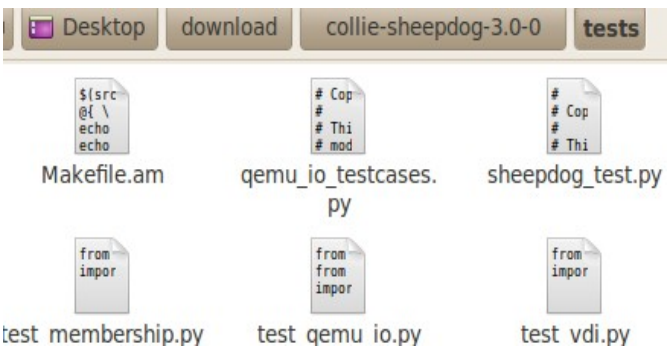
l'**ajout** ou la **suppression de noeuds ne change pas le mapping des objet**, contrairement au hachage classique (key mod N).

http://en.wikipedia.org/wiki/Consistent_hashing

http://www.tomkleinpeter.com/2008/03/17/programmers-toolbox-part-3-consistent-hashing/

8.4 Différences entre sheepdog 2.3.0-1 et 3.0-0 :





-mise a jour de l'outil collie (nouvelle commandes comme vdi create, cluster recover, etc...).
-support de **accord** et **zookeeper** comme système de communication de groupe (alternative a corosync).
-granularité au niveau du contenu des logs (option -l 'loglevel1..7').

archive concernant les mise a jour : **http://lists.wpkg.org/pipermail/sheepdog/**

9. Liens & références :

pour comprendre le fonctionnement de sheepdog :

https://github.com/collie/sheepdog/wiki/Sheepdog-Design

pour suivre le developpement de sheepdog (mise à jour, patchs, feedbacks, etc...):

http://lists.wpkg.org/pipermail/sheepdog/

pour la prise en main de sheepdog :

https://github.com/collie/sheepdog/wiki/Getting-Started

http://berrange.com/posts/2011/10/11/setting-up-a-sheepdog-cluster-and-exporting-a-volume-to-a-kvm-guest/

http://www.russellbryant.net/corosync/doxygen/files.html //corsync source code