Jérôme Delamarche

Cluster haute-disponibilité avec équilibrage de charge

À travers un exemple concret, nous vous proposons de déjouer les pièges de la mise en œuvre d'un cluster haute-disponibilité économique, avec équilibrage de charge et constitué uniquement de deux machines !

1. Introduction

En entreprise, certains services réseau sont considérés comme primordiaux (serveur Intranet, messagerie, annuaire...). En outre, les serveurs qui les hébergent peuvent être fortement chargés, produisant alors des temps de réponse fortement dégradés.

Nous avons donc deux problèmes simultanés à résoudre :

1) une nécessité de haute-disponibilité (*high-availability* ou *H-A*) ;

2) un besoin de performance et de dimensionnement (*scalability*).

Existe-t-il des solutions simples, gratuites et *open source* et qui nécessitent un investissement matériel minimal ?

La réponse est oui, bien sûr ! Dans cet exposé, nous allons vous montrer pas à pas, comment créer un *cluster H-A* avec équilibrage de charge (*load-balacing*) à l'aide de seulement deux serveurs.

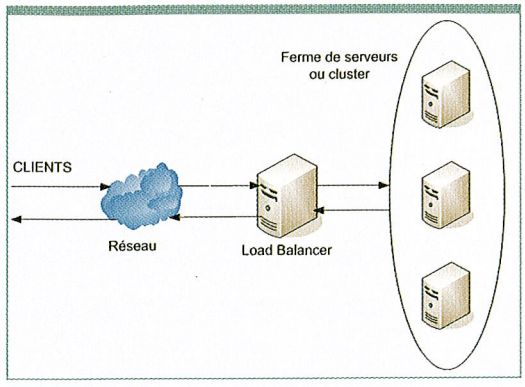
Notez que même si la haute-disponibilité seule vous intéresse, l'ajout de la fonction d'équilibrage de charge vous permettra de vérifier que votre serveur de secours est bien lui aussi toujours fonctionnel.

2. Un peu de théorie – vraiment très peu !

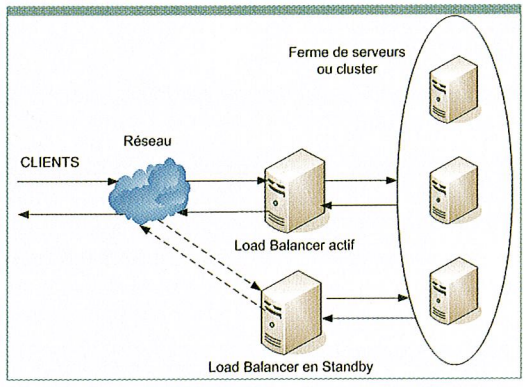
Un cluster (une grappe en français) est un groupe de machines rendant le même service de manière transparente pour les clients. En effet, ceux-ci ne savent pas qu'ils s'adressent à un cluster. En outre, les services activés sur les nœuds du cluster ne doivent pas non plus être « conscients » de leur appartenance à un cluster.

Deux principes sont mis en œuvre pour aboutir à cette transparence :

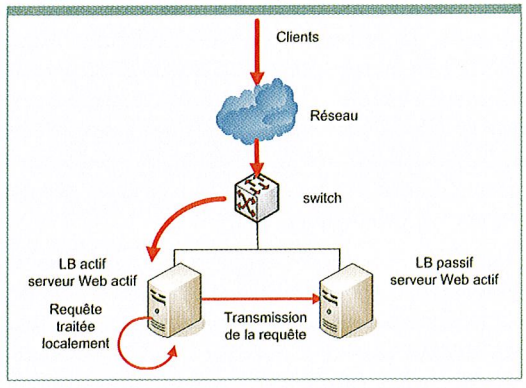
1) Un serveur particulier, appelé « l'équilibreur de charge » (le *load-balancer* aussi appelé le « *director* ») est placé entre les clients et les nœuds du cluster. Son rôle consiste à aiguiller les requêtes du client vers un nœud particulier.



2) Si le load-balancer tombe en panne, le cluster est indisponible, donc ce serveur est redondé. Le schéma de principe devient alors :



Mais rappelez-vous, je vous ai indiqué que nous ferions un cluster H-A avec équilibrage de charge à l'aide de deux machines seulement ! En ajoutant un *switch* réseau, le schéma correspondant à notre plate-forme d'étude devient donc :

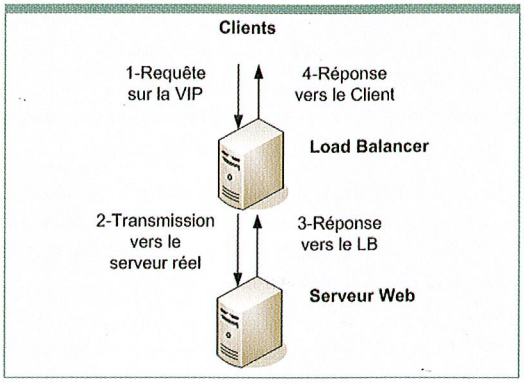


Pour que le load-balancer soit transparent pour les clients, l'astuce consiste à lui faire porter, ce que nous appellerons une « adresse IP virtuelle » (VIP en abrégé). Chaque serveur dispose toujours de son adresse IP réelle (RIP en abrégé).

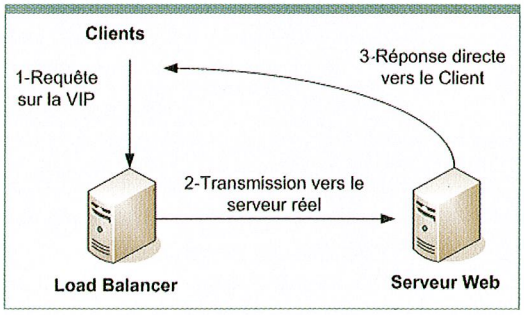
Le client s'adresse alors toujours au load-balancer par sa VIP et celui-ci peut relayer la requête sur un serveur réel. Mais comment le serveur réel renvoie-t-il au client la réponse ?

Plusieurs solutions sont possibles et dépendent de l'architecture réseau :

- Si le load-balancer agit comme une passerelle pour les serveurs réels, il s'agira de mettre en place de la translation d'adresses (NAT).



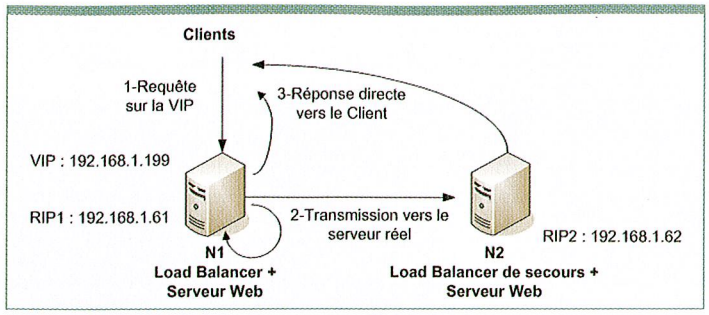
- Si le load-balancer est sur le même sous-réseau que les serveurs réels, la solution la plus propice sera le Retour-Direct (DR ou *Direct Return*). Cette solution est plus performante, car elle décharge le *load-balancer* de la gestion des réponses. Elle est toutefois plus délicate à mettre en œuvre (c'est néanmoins celle qui est proposée dans cet article).



- Il existe une autre solution, moins courante, qui met en œuvre des tunnels quand on veut faire communiquer le load-balancer et les serveurs réels à travers Internet.

3. Préparation de notre plate-forme

Dans le reste de l'article, nous utiliserons deux machines nommées « N1 » et « N2 ». L'adresse réelle de ces machines sera RIP1 et RIP2, leurs valeurs réelles sont indiquées dans le schéma ci-après, mais ce que nous dirons, est, bien sûr, indépendant de ces valeurs. VIP est l'adresse IP virtuelle portée par le load-balancer. Chaque fois que vous verrez les notations `$VIP`, `$RIP1` et `$RIP2`, pensez qu'il faut leur substituer l'adresse IP correspondant à votre architecture.



Nous supposons que les deux serveurs N1 et N2 ont été installés avec la même distribution et utilisent un noyau Linux 2.6.

Les serveurs doivent être synchronisés via NTP **[NTP]**. Ce pré-requis est indispensable, mais je suis sûr que tous vos serveurs sont déjà synchronisés sur un serveur de temps. ;-).

Comme service supporté par notre cluster, nous choisissons Apache, car c'est l'un des besoins les plus simples et les plus courants.

Normalement, le contenu délivré par les serveurs Apache doit être le même sur N1 et N2, mais pour tester notre équilibrage de charge, nous allons commencer par créer une page statique « personnalisée » sur chacun des serveurs.

Sous la racine du serveur Apache (sa valeur est donnée par la directive `DocumentRoot` dans le fichier de configuration `httpd.conf`), placez un fichier nommé `testcluster.html` et contenant la chaîne `N1` pour le serveur N1 et `N2` pour le serveur N2.

Rappel pour trouver le fichier de configuration de votre serveur Apache :

Tapez la commande `httpd -V`, puis repérez les valeurs des variables `HTTPD_ROOT` et `SERVER_CONFIG_FILE`. Si `SERVER_CONFIG_FILE` a pour valeur un nom de fichier absolu : bingo ! C'est le bon ! Sinon, le nom complet est donné par la concaténation des deux variables.

Au besoin, démarrez le serveur Apache sur chacun des nœuds, puis vérifier avec un navigateur que vous pouvez obtenir les pages nouvellement créées en naviguant à l'adresse réelle des serveurs.

```
# wget http://$RIP1/testcluster.html
# wget http://$RIP2/testcluster.html
```

ATTENTION

Lorsque, dans les paragraphes suivants, vous ferez des tests avec un navigateur, veillez bien à en désactiver le cache, faute de quoi, vous ne constaterez pas toujours l'effet du load-balancing. En cas de doute, la commande `wget` peut être d'un précieux secours.

4. Mise en place du load-balancer

4.1 Vérification des pré-requis

Pour réaliser cette tâche, nous allons utiliser la fonctionnalité IPVS (*IP Virtual Server*) qui est incluse dans le noyau 2.6, ainsi que dans le noyau 2.4, depuis la version 2.4.26. IPVS est composé d'un ensemble de modules noyau dont le rôle est de rediriger les paquets entrants vers d'autres serveurs en utilisant divers algorithmes de distribution. IPVS fait partie du projet *Linux Virtual Server* **[LVS]**.

Vous pouvez vérifier que vous disposez de ces modules par la commande :

```
# ls /lib/modules/`uname -r`/kernel/net/ipv4/ipvs/
up_vs_dh.ko
ip_vs_ftp.ko
ip_vs.ko
...
```

Pour gérer le moteur IPVS, nous avons besoin d'une commande nommée `ipvsadm` (pour ceux qui connaissent le moteur de filtrage de paquets `netfilter`, on peut dire que `ipvsadm` est à `ipvs` ce que `iptables` est à `netfilter`).

Malheureusement, selon les distributions, la commande `ipvsadm` n'est pas toujours proposée ou installée par défaut. La version la plus récente est la 1.24 dont vous pouvez télécharger les sources à l'adresse suivante : http://www.linuxvirtualserver.org/software/ipvs.html. Vous pouvez aussi trouver des packages pour RedHat et Debian à l'adresse **http://www.ultramonkey.org/download/heartbeat/2.0.4/**.

Pour compiler les sources de `ipvsadm`, vous aurez besoin de l'environnement de compilation pour votre noyau et, en particulier, des fichiers inclus associés. Pour compiler les sources, appliquez les commandes suivantes :

```
# tar xvfz ipvsadm-1.24.tar.gz
# make
# make install
```

4.2 Réalisation manuelle de l'équilibrage de charge

Nous décidons arbitrairement que la machine N1 sera le load-balancer. La machine N1 doit prendre la VIP en plus de son adresse réelle. Nous allons créer pour cela un alias sur la carte `eth0` :

```
N1# ip addr add $VIP netmask 255.255.255.0 dev eth0
```

ATTENTION

Linux fait une différence entre les adresses « aliasées » qui sont supportées par une interface virtuelle et les adresses « secondaires » qui sont portées par l'interface réelle. La commande `ifconfig` permet de définir et afficher les interfaces « aliasées ». La commande `ip` permet de définir les adresses secondaires, mais liste toutes les interfaces et adresses :

```
# ifconfig eth0:1 1.2.3.4
```

définit une interface « aliasée » `eth0:1` ;

```
# ip addr add 5.6.7.8 netmask 255.255.255.0 dev eth0
```

définit une adresse IP secondaire.

La commande `ifconfig` affiche l'interface `eth0:1`, alors que la commande `ip addr show` affiche les deux adresses ! Dans le doute, utilisez donc la deuxième commande pour connaître toutes les adresses IP associées à vos cartes réseau.

On met en place un équilibrage de charge pour les paquets à destination du port 80 sur la VIP. Nous devons choisir un algorithme de répartition. IPVS en propose 10 dont les principaux sont :

- `rr` (*Round-Robin*) : les requêtes sont transmises à tour de rôle sur chaque nœud du cluster.
- `wrr` (*Weighted Round Robin*) : identique au précédent, mais peut gérer une pondération associée à chaque nœud, l'objectif étant de charger les machines proportionnellement à leur puissance.
- `lc` (*Least Connection*) : IPVS transmet la requête au serveur sur lequel il y a le moins de connexions actives.
- `wlc` (*Weighted Least-Connection*) : sa variante pondérée...
- `lblc`, `lblcr`, `dh` et `sh` : déterminent le nœud en fonction de l'adresse IP source ou destination.

Ici l'algorithme de répartition choisi est `rr` (Round-Robin). Il nous permettra, dans un premier temps, de bien vérifier que tous les serveurs sont actifs. La commande de création d'un nouveau service à équilibrer est :

```
N1# ipvsadm -A -t $VIP:80 -s rr
```

À tout moment, la commande `ipvsadm -L` permet d'afficher les règles de load-balancing :

```
N1# ipvsadm -L
IP Virtual Server version 1.2.0 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port      Forward Weight ActiveConn InActConn
TCP  192.168.1.199:http rr
```

Puis, on ajoute la liste des serveurs réels. On indique qu'on souhaite du Retour-Direct avec l'option `-g`. On commence avec le nœud N1 qui est le load-balancer :

```
N1# ipvsadm -a -t $VIP:80 -r $RIP1:80 -g
```

Les règles gérées par IPVS deviennent alors :

```
N1# ipvsadm -L
IP Virtual Server version 1.2.0 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port      Forward Weight ActiveConn InActConn
TCP  192.168.1.199:http rr
  -> N1:http                 Local   1      0          0
```

On peut alors tester que l'on peut obtenir la page `testcluster.html` en naviguant à l'adresse VIP. On ajoute alors le 2ème serveur :

```
N1# ipvsadm -a -t $VIP:80 -r $RIP2:80 -g
```

Puisque l'ordonnanceur sélectionné est le Round-Robin (`-s rr`), les requêtes seront acheminées « à tour de

rôle » sur N1 et N2. Si on rafraîchit la page obtenue avec le navigateur, quelque chose de bizarre doit se produire : pour un rafraîchissement sur deux, la page est obtenue, dans l'autre cas, un *timeout* se produit... !

Réfléchissons : comment N1 effectue-t-il le relayage vers N2 ? Dès réception du paquet à destination de la VIP, le load-balancer va modifier l'adresse MAC destination en y plaçant celle du serveur réel N2. Le serveur N2 va donc recevoir le paquet, mais encore faut-il qu'il l'accepte ! Ce n'est possible que s'il dispose aussi de l'adresse VIP définie sur une interface. Ici, N2 ne possède pas la VIP, il refuse les paquets. D'où notre échec pour une tentative d'affichage sur deux.

Il faut donc que N2 possède **aussi** cette VIP ! Mais, si nous plaçons aussi la VIP comme adresse secondaire (c'est-à-dire sur `eth0` dans notre exemple), N2 va répondre aux requêtes ARP qui cherchent à obtenir l'adresse MAC associée à la VIP ! N2 entre ainsi en conflit avec le load-balancer pour la possession de la VIP.

L'astuce consiste alors à placer cette VIP comme adresse secondaire sur l'interface *loopback*. Nous l'appellerons l'« interface cachée » (*hidden interface*) :

```
N2# ip addr add $VIP netmask 255.255.255.255 dev lo
```

Mais cela ne suffit pas, il faut empêcher N2 de répondre aux requêtes ARP concernant la VIP. Pour cela, nous allons modifier les paramètres `arp_announce` et `arp_ignore` de la pile TCP/IP de N2 (voir encadré). Cette facilité est offerte avec un noyau 2.6 ou bien à partir du noyau 2.4.26. La modification de ces paramètres doit avoir lieu **avant** la déclaration de l'adresse secondaire :

```
N2# ip addr del $VIP netmask 255.255.255.255 dev lo
N2# echo 1 > /proc/sys/net/ipv4/conf/eth0/arp_ignore
N2# echo 2 > /proc/sys/net/ipv4/conf/eth0/arp_announce
N2# ifconfig lo:0 $VIP netmask 255.255.255.255 up
```

Vous constatez que le `netmask` utilisé est 255.255.255.255 (ou /32). En effet, s'il était moins strict (par exemple 255.255.255.0 ou /24), l'interface loopback `lo` accepterait aussi les paquets pour n'importe quelle adresse IP du sous-réseau associé à la VIP ! Ce comportement est particulier à l'interface loopback. Pour vous en convaincre : essayez de « *pinger* » n'importe quelle adresse du réseau 127.0.0.0/8 : ça marche !

Dans de nombreuses documentations, vous trouverez plutôt les commandes suivantes :

```
# echo 1 > /proc/sys/net/ipv4/conf/lo/arp_ignore
# echo 2 > /proc/sys/net/ipv4/conf/lo/arp_announce
# echo 1 > /proc/sys/net/ipv4/conf/all/arp_ignore
# echo 2 > /proc/sys/net/ipv4/conf/all/arp_announce
# route add -host $VIP lo:0
```

Tout d'abord, il faut savoir que, dans cet exemple, la configuration de l'interface `lo` est redondante par rapport à la configuration sur `all`, et, d'autre part, nous ne voulons pas modifier le comportement de l'interface loopback cachée, mais bien le comportement de `eth0` vis-à-vis d'ARP ! Ces commandes fonctionnent donc, mais ne sont pas strictement exactes.

Rappel pour trouver le fichier de configuration de votre serveur Apache :

Explication sur les valeurs des paramètres `arp_ignore` et `arp_announce` :

Le paramètre `arp_announce` détermine l'adresse IP source annoncée dans les requêtes ARP émises. Si le paramètre vaut 1, le noyau essaye d'utiliser une adresse IP source qui corresponde au sous-réseau cible. Ce peut-être une adresse IP primaire (`eth0`) ou secondaire/aliasée (`eth0:1`). Si la valeur du paramètre est 2, seules les adresses IP primaires sont considérées.

Le paramètre `arp_ignore` détermine l'adresse MAC à retourner en réponse à une requête ARP. Avec la valeur 0, ce peut être une adresse MAC quelconque de la machine. La valeur 8 inhibe toute réponse ARP : l'interface devient muette pour ARP, il faut utiliser des caches ARP statiques pour communiquer avec la machine. Ici, nous utilisons la valeur 1, une interface propose son adresse MAC uniquement si elle est configurée avec l'adresse IP cible de la requête. La valeur 2 impose en plus que l'adresse IP source et l'adresse IP cible appartiennent au même réseau.

Enfin, pour un paramètre donné, la plus grande valeur entre `/proc/sys/net/ipv4/conf/lo/parm` et `/proc/sys/net/ipv4/all/parm` est utilisée par le noyau **[ARP, PROC]**.

Les modifications sur les paramètres `arp_ignore` et `arp_anounce` doivent être effectuées AVANT l'activation des interfaces concernées.

4.3 Gestion de la persistance des sessions

Selon l'application servie par le cluster, il faut éventuellement s'assurer qu'un client sera toujours affecté au même nœud du cluster. C'est par exemple le cas si vous utilisez PHP et que les sessions sont gérées via des fichiers locaux.

IPVS sait gérer la persistance des connexions : la première requête est redirigée sur un nœud en fonction de l'algorithme d'ordonnancement sélectionné. Les autres requêtes seront assurées d'aboutir sur le même nœud. Notons que des algorithmes comme le *source hashing* (`sh`) donnent le même résultat.

L'activation de la persistance des sessions est réalisée à l'aide de l'option `-p` qui permet en outre de définir un timeout de persistance en cas d'inactivité du client (`man ipvsadm`).

La commande `ipvsadm -L -c` affiche l'état des connexions en cours.

5. Automatisation de la gestion du load-balancer

5.1 Principe

Tel que nous l'avons mis en œuvre, notre load-balancer fonctionne... tant que tous les nœuds sont fonctionnels ! Je m'explique : que se passe-t-il si le nœud N2 est arrêté ou même si tout simplement le service Apache sur N2 est interrompu ? Le load-balancer va rediriger une requête sur deux vers N2 et retournera donc, pour ces requêtes, une erreur au client.

Pour résoudre ce problème, nous allons mettre en œuvre sur le load-balancer un processus spécialisé, nommé `ldirectord` et que nous appellerons le « directeur », en français. Écrit en Perl, ce programme a pour rôle la surveillance applicative des nœuds et modifie, en temps réel, les règles d'équilibrage de charge, à l'aide de la commande `ipvsadm`.

5.2 Installation des pré-requis

Pour installer `ldirectord`, nous devons préalablement installer `heartbeat` que nous décrirons au chapitre suivant. La procédure d'installation décrite ci-dessous doit être réalisée sur N1 et N2.

5.2.1 Installation à partir des sources

Ils sont disponibles à l'adresse suivante : **http://www.ultramonkey.org/download/heartbeat/2.0.8/**. Téléchargez la dernière version (ici : la 2.0.8), puis compilez-la ainsi :

```
# tar xvfz heartbeat-2.0.8*
# cd heartbeat-2.0.8
# ./configure
# make
# make install
```

5.2.2 Installation à partir des RPM

Vous pouvez préférer l'utilisation de packages « clés en main » disponibles à l'adresse **http://linux-ha.org/download/index.html#2.0.8**.

Commencez par le package `heartbeat-pils-2.0.8`, puis `heartbeat-stonith-2.0.8`, `heartbeat-2.0.8` et enfin `heartbeat-ldirectord-2.0.8`. Si les pré-requis pour `heartbeat-ldirectord` ne sont pas satisfaits, forcez l'installation avec l'option `--nodeps`. Nous installerons les modules Perl manquants après.

```
# rpm -Uvh heartbeat-pils* heartbeat-stonith*
heartbeat-2.0.8*
# rpm -Uvh -nodeps heartbeat-ldirectord*
```

5.2.3 Arborescence des répertoires et fichiers créés

Selon la méthode d'installation choisie, les fichiers de configuration sont situés sous `/etc/ha.d` (cas des RPM) ou `/usr/local/etc/ha.d` (installation à partir des sources). Dans la suite de l'article, nous supposerons que le répertoire de configuration est `/usr/local/etc/ha.d`.

Le script Perl `ldirectord` est situé alors sous `/usr/local/sbin/ldirectord`.

5.2.4 Test préalable et installation des modules Perl manquants

Nous vérifions que le directeur est fonctionnel en l'exécutant manuellement :

```
# cd /usr/local
# sbin/ldirectord -d etc/ha.d/ldirectord.cf start
```

S'il se plaint avec un message du type :

```
Can't locate Mail/Send.pm in @INC (@INC contains:
..... ) at ldirectord line 3417.
BEGIN failed--compilation aborted at ldirectord line 3417
```

alors, il manque des modules Perl. Dans cet exemple, il manque le module `Mail::Send`. C'est probablement un des cas les plus fréquents. Pour installer ce module, nous utilisons CPAN :

```
# perl -MCPAN -e 'install Mail::Send'
```

Notez que si vous n'avez jamais utilisé CPAN auparavant, vous devrez répondre à des questions pour configurer CPAN **[CPAN]**.

5.3 Mise en œuvre

`ldirectord` est configuré via un fichier nommé par `ldirectord.cf` et situé dans notre cas sous `/usr/local/etc/ha.d`. Voici un exemple commenté du fichier qui correspond à notre cluster de deux nœuds. N'oubliez pas de remplacer `$VIP`, `$RIP1` et `$RIP2` par leurs valeurs réelles...

```
checktimeout=10       ; timeout de 10s avant de déclarer un serveur
"mort"
                      ; en cas d'échec de l'établissement de la connexion réseau
checkinterval=10      ; effectue un test de "vie" toutes les 10s

emailalert="root"     ; adresse mail de l'Administrateur
emailalertfreq=0      ; une seule alerte par problème
emailalertstatus=stopping       ; seules les situations d'arrêt nous intéressent

virtual=$VIP:80
    # Pour vérifier le bon fonctionnement des noeuds du cluster, on
    # peut définir avec les paramètres "request" et "receive", l'URL
    # à obtenir et le contenu retourné.
    # Nous préférons définir ce test serveur par serveur, ce qui permet
    # de mieux affiner le test.
    # Dans notre cas, ldirectord demande à chaque noeud la page
    # "testcluster.html" et vérifie que la réponse est bien "N1" ou
    # "N2" selon les noeuds.
    # (le mot-clé "gate" rappelle que nous faisons du Retour-Direct)
          checktype=negotiate             ; test par question / réponse
real=$IP1:80 gate 1 "testcluster.html", "N1"
          real=$IP2:80 gate 1 "testcluster.html", "N2"

service=http
          checkport=80
          protocol=rr          ; toujours le Round-Robin
```

Si vous devez aussi supporter le protocole HTTPS, définissez simplement une autre rubrique `virtual` sur le port 443.

Nous allons tester cette configuration sur le nœud N1. À ce stade, normalement, le service Apache est actif sur N1 et N2... Tout d'abord, nettoyons les règles IPVS éventuellement résiduelles :

```
N1# ipvsadm -C
```

Puis, lançons le directeur en mode *debug* pour tester la configuration :

```
N1# cd /usr/local
N1# sbin/ldirectord -d etc/ha.d/ldirectord.cf start

DEBUG2: Running exec(/usr/local/sbin/ldirectord -d ldirectord.cf
start)
Running exec(/usr/local/sbin/ldirectord -d ldirectord.cf start)
...
DEBUG2: Running system(/sbin/ipvsadm -A -t 192.168.1.199:80 -s rr )
```

```
Running system(/sbin/ipvsadm -A -t 192.168.1.199:80 -s rr )
DEBUG2: Added virtual server: 192.168.1.199:80
Added virtual server: 192.168.1.199:80
DEBUG2: Disabled server=192.168.1.61
DEBUG2: Disabled server=192.168.1.62
DEBUG2: Checking negotiate: real server=negotiate:http:tcp:192.168.1.61:
80::80:1:testcluster\.html:node (virtual=tcp:192.168.1.199:80)
DEBUG2: check_http: url="http://192.168.1.61:80/index.html" virtualhost
="192.168.1.61"
LWP::UserAgent::new: ()
LWP::UserAgent::request: ()
LWP::UserAgent::send_request: GET http://192.168.1.61:80/testcluster.
html
LWP::UserAgent::_need_proxy: Not proxied
LWP::Protocol::http::request: ()
LWP::Protocol::collect: read 11 bytes
LWP::UserAgent::request: Simple response: OK
DEBUG2: Running system(/sbin/ipvsadm -a -t 192.168.1.199:80 -r
192.168.1.61:80 -g -w 1)
Running system(/sbin/ipvsadm -a -t 192.168.1.199:80 -r 192.168.1.61:80 -g -w 1)
DEBUG2: Added real server: 192.168.1.61:80 (192.168.1.199:80) (Weight set to 1)
...
```

Vous constaterez la mise en place des règles pour IPVS, puis les tests périodiques des serveurs réels (toutes les `checkinterval` secondes). Dans un autre terminal, sur N1, la commande `ipvsadm` permet de vérifier ces règles :

```
N1 # ipvsadm
IP Virtual Server version 1.2.0 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port      Forward Weight ActiveConn InActConn
TCP  192.168.1.199:http rr
  -> N2:http                 Route   1      0          0
  -> N1:http                 Local   1      0          0
```

Puis, arrêtez le service `httpd` sur le nœud N2 :

```
N2# /etc/init.d/httpd stop
```

Lors du prochain test fait par `ldirectord`, l'affichage montre les lignes suivantes :

```
DEBUG2: Checking negotiate: real server =negotiate:http:tcp:192.168.
1.62:80::80:1: testcluster\.html:node (virtual=tcp:192.168.1.199:80)
DEBUG2: check_http: url=http://192.168.1.62:80/testcluster.html
virtualhost="192.168.1.62"
LWP::UserAgent::new: ()
LWP::UserAgent::request: ()
LWP::UserAgent::send_request: GET http://192.168.1.62:80/
testcluster.html
LWP::UserAgent::_need_proxy: Not proxied
LWP::Protocol:http::request: ()
LWP::UserAgent::request: Simple response: Internal Server Error
DEBUG2: Running system(/sbin/ipvsadm -d -t 192.168.1.199:80 -r
192.168.1.62:80)
```

La commande `ipvsadm` nous montre bien que le nœud N2 a été supprimé :

```
N1 # ipvsadm -L
IP Virtual Server version 1.2.0 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port      Forward Weight ActiveConn InActConn
TCP  192.168.1.199:http rr
  -> N1:http                 Local   1      0          0
```

À tout moment, vous pouvez vérifier le bon fonctionnement d'IPVS en affichant les compteurs de répartition des paquets par la commande :

```
N1# ipvsadm -l --stats
```

5.4 Réglages de la gestion des logs d'Apache

Nous allons peaufiner notre configuration en évitant de polluer les « *access logs* » d'Apache avec les requêtes émises par le directeur. Sur tous les nœuds du cluster (ici N1 et N2), il faut éditer la configuration du serveur Apache et y ajouter la ligne suivante :

```
SetEnvIf Request_URI ^/testcluster\.html$ nolog
```

Puis, il faut modifier la ligne `CustomLog` pour qu'elles ressemblent à :

```
CustomLog logs/access_log combined env=!nolog
```

Ces directives indiquent qu'il faut tout d'abord définir une variable d'environnement pour chaque requête de la forme `/testcluster.html`. Les logs d'accès ne seront générés que si cette variable d'environnement n'est pas positionnée.

6. Mise en place de la haute-disponibilité

6.1 Principe

À ce stade, nous disposons d'un cluster de deux nœuds offrant le service Apache avec, sur l'un des nœuds, un directeur capable d'assurer l'équilibrage de charge.

Nous avons vu que le directeur est capable de déterminer le bon ou mauvais fonctionnement des services Apache et sait adapter sa configuration en conséquence. Oui, mais, que se passe-t-il si le directeur s'arrête ou si le nœud qui l'héberge devient indisponible ?

Le directeur est un SPOF (*Single Point of Failure*) (toute relation avec une personne existante ou ayant existé sera fortuite ;-)), il faut donc prévoir un directeur de secours. C'est là qu'intervient le dernier composant de notre architecture : `heartbeat` **[HRBT]**.

`heartbeat` est un système de surveillance de machines associé à un gestionnaire de ressources. On définit, au démarrage, quel nœud doit activer quelle ressource et, en cas de défaillance de celui-ci, `heartbeat` migrera la ressource sur un nœud sain.

Notons qu'il existe deux versions de `heartbeat` :

- Avec la version 1, `heartbeat` ne gère que les clusters de deux nœuds.
- Avec la version 2, `heartbeat` s'est étoffé, son architecture interne a été repensée et il sait maintenant gérer les clusters de seize nœuds. La configuration de `heartbeat` utilise maintenant un fichier XML qui est automatiquement répliqué sur les nœuds. Mais, dans cet article, nous continuerons à employer la syntaxe de la version 1 qui est compatible avec la version 2.

6.2 Installation des pré-requis

L'installation de `heartbeat` a été réalisée au §5.2 lors de l'installation du directeur !

6.3 Mise en œuvre

Cet article ne prétend pas couvrir toutes les fonctionnalités de heartbeat. Nous nous concentrons sur notre problématique de gestion du directeur. Pour en savoir plus, vous pouvez consulter les sites indiqués en référence ainsi que le numéro hors-série n°18 de votre revue préférée **[HS18]**.

Comme indiqué au §5.2.3, le répertoire racine d'installation de heartbeat peut être / ou /usr/local. Ainsi, les fichiers de configuration seront situés sous /etc/ha.d/ ou /usr/local/etc/ha.d/. Nous nommerons HA_DIR ce répertoire.

heartbeat utilise trois fichiers de configuration **[HS18]** :

- ha.cf qui définit le cluster et les méthodes de surveillance des nœuds ;
- haresources qui liste les ressources à gérer (migrer) ;
- authkeys qui indique la méthode d'authentification/chiffrement à utiliser pour la communication entre les nœuds.

6.3.1 Fichier ha.cf

Ce fichier décrit les paramètres de la « mécanique » heartbeat : les valeurs des différents timeouts, les méthodes de surveillance de l'état des nœuds, les informations de *login*, la liste des nœuds du cluster.

Voici un exemple de fichier de configuration commenté qui correspond à notre cluster :

```
# Par quels liens, établit-on la surveillance: eth0, ttyS0 ?
# et quels types de paquets doit-on émettre: bcast, mcast, ucast ?
# Nous choisissons le Multicast sur eth0 et le port série
# il est possible de spécifier un groupe pour le Multicast:
# syntaxe: mcast [dev] [mcast group] [port] [ttl] [loop]
# si le loop est "1" (au lieu de "0") le paquet est aussi reçu en loopback

mcast         eth0 225.0.0.7 694 1 0
baud          19200
serial        /dev/ttyS0

debugfile     /var/log/ha.debug
logfile       /var/log/ha.log
logfacility   local0

# temps entre 2 battements (2 signifie 2s, sinon 2000ms)
keepalive     2
# temps nécessaire avant de déclarer un noeud comme mort
deadtime      10
# temps avant d'avertir dans le log
warntime      6
# valeur utilisée pour le démarrage (au moins 2 fois le deadtime)
initdead      60

# le nom des noeuds DOIT être celui retourné par 'uname -n'
node          N1
node          N2

# "off" indique que le primaire ne retrouve pas son rôle quand il ressuscite
# (c'est plus sûr de contrôler manuellement ce retour "à la normale")
auto_failback off
```

Dans cet exemple, les machines N1 et N2 sont connectées par un câble série croisé (connexion « *back-to-back* » avec un câble NULL-Modem), ce qui permet une surveillance des plus sûres, car ne faisant pas intervenir de composants tiers comme un switch Ethernet par exemple.

La valeur du timeout initdead doit être suffisamment élevée pour que, au démarrage des deux machines, la plus rapide ne considère pas de manière anticipée que l'autre est hors service alors qu'elle n'a pas achevé sa phase d'initialisation.

Enfin, il est primordial que les valeurs des paramètres node soient exactement identiques au résultat de la commande uname -n, exécutée sur les nœuds. Ces noms seront à nouveau utilisés pour créer le contenu du fichier haresources (voir §6.3.3).

6.3.2 Fichier authkeys

Ce fichier décrit le mécanisme d'authentification utilisé par les nœuds du cluster. Plusieurs méthodes sont disponibles :

- le calcul d'un crc, idéal pour ses performances en cas de liaison série ;
- le chiffrement avec MD5 ou SHA-1, à l'aide d'un secret partagé.

Le fichier indique plusieurs méthodes, et, parmi celles-ci, la méthode à utiliser :

```
auth 1
1 md5 "secret"
2 crc
3 sha1 "autre secret"
```

Ici, nous utilisons la méthode « 1 », soit MD5.

6.3.3 Fichier haresources

Notre fichier haresources contient une seule ligne :

```
N1 IPaddr2::$VIP ldirectord
```

(N'oubliez pas de remplacer $VIP par l'adresse virtuelle de votre cluster !)

Cette ligne indique qu'au démarrage la machine N1 doit acquérir les ressources IPaddr2 et ldirectord. Ces « ressources » sont en fait des scripts situés sous $HA_DIR/resource.d et ressemblent à des scripts d'initialisation « System V », c'est-à-dire qu'ils acceptent des paramètres, ainsi que les arguments start, stop, status, restart. Certains de ces scripts peuvent en outre être conformes à la norme OCF (*Open Cluster Framework*) qui définit les paramètres en entrée, les comportements attendus (codes retours) et des règles de nommage de variables pour échanger des informations **[OCF]**.

Dans notre exemple, le script IPaddr2 reçoit un paramètre supplémentaire : la VIP. Ce script fait partie de la distribution de heartbeat. Il gère une adresse virtuelle passée en paramètre. Il existe aussi un script nommé IPaddr qui fait la même chose, à ceci près que, dans notre cas, IPaddr2 supporte la gestion des interfaces cachées (voir encadré ci-après).

Avantage du script IPaddr2 :

Lorsqu'un nœud (ici N2) acquiert une ressource, le script IPaddr2 teste si la VIP était activée sur une interface cachée. Dans ce cas, l'interface cachée est supprimée et la VIP attribuée comme adresse secondaire de l'interface eth0. Mais le script mémorise le fait qu'il avait une interface cachée en créant un fichier nommé $VIP sous le répertoire /var/run/heartbeat/rsctmp/IPaddr. Ce fichier contient une ligne de la forme :

```
$VIP 32 255.255.255.255 lo
```

Ainsi, si le nœud N2 doit rendre la ressource, le script IPaddr2 supprime l'adresse secondaire sur eth0 et réattribue la VIP sur l'interface cachée lo. C'est magique !

6.3.4 Configuration du nœud N2

Les fichiers ha.cf, haresources et authkeys doivent ensuite être recopiés sur N2.

6.3.5 Mise en route

Il faut arrêter l'éventuel processus ldirectord lancé dans les étapes précédentes et retirer la VIP sur le nœud N1 en lançant la commande suivante :

```
N1# ip addr del $VIP dev eth0
```

Puis, on arrête le service Apache sur le nœud N2 qui ne fait pas encore partie du cluster.

```
N2# /etc/init.d/httpd stop
```

Sur N1 et N2, on surveille, dans un terminal, les logs de heartbeat par la commande :

```
N?# tail -f /var/log/ha.log
```

Puis, on lance heartbeat sur les deux machines :

```
N1# /etc/init.d/heartbeat start
```

Voici un extrait des logs produits par heartbeat sur N1 :

```
...
heartbeat[3856]: 2007/06/26_09:38:27 info: Configuration validated.
Starting heartbeat 2.0.8
...
heartbeat[3857]: 2007/06/26_09:38:28 info: glib: UDP multicast heartbeat
started for group 225.0.0.7 port 694 interface eth0 (ttl=1 loop=1)
...
heartbeat[3857]: 2007/06/26_09:38:28 info: Local status now set to: 'up'
heartbeat[3857]: 2007/06/26_09:38:29 info: Link N1:eth0 up.
heartbeat[3857]: 2007/06/26_09:39:28 WARN: node N2: is dead
...
ResourceManager[3986]:          2007/06/26_09:39:31 info: Acquiring
resource group: N1 IPaddr2::$VIP ldirectord
IPaddr2[4010]:        2007/06/26_09:39:31 INFO:  Resource is stopped
ResourceManager[3986]:          2007/06/26_09:39:31 info: Running
/usr/local/etc/ha.d/resource.d/IPaddr2 $VIP start
IPaddr2[4078]:        2007/06/26_09:39:32 INFO: /sbin/ip -f inet addr add
$VIP/24 brd 192.168.1.255 dev eth0
...
IPaddr2[4069]:        2007/06/26_09:39:32 INFO:  Success
ResourceManager[3986]:          2007/06/26_09:39:34 info: Running
/usr/local/etc/ha.d/resource.d/ldirectord  start
heartbeat[3857]: 2007/06/26_09:39:40 info: Local Resource acquisition
completed. (none)
heartbeat[3857]: 2007/06/26_09:39:40 info: local resource transition completed.
```

Quand le message local resource transition completed apparaît, le nœud N1 a pris la VIP et le directeur est en place. Vous pouvez accéder au serveur Web par la VIP.

Vérifiez que N2 a bien conservé la VIP sur son interface cachée et que les paramètres ARP sont toujours corrects (voir §4.2). Maintenant, nous activons le service Apache et heartbeat sur N2 :

```
N2# /etc/init.d/heartbeat start
N2# /etc/init.d/httpd start
```

Voici un extrait des logs produits par heartbeat sur N2 :

```
heartbeat[10331]: 2007/06/26_10:01:13 info: Configuration validated. Starting
heartbeat 2.0.8
...
heartbeat[10332]: 2007/06/26_10:01:13 info: glib: UDP multicast heartbeat started
for group 225.0.0.7 port 694 interface eth0 (ttl=1 loop=1)
...
heartbeat[10332]: 2007/06/26_10:01:13 info: Local status now set to: 'up'
heartbeat[10332]: 2007/06/26_10:01:14 info: Link N1:eth0 up.
...
heartbeat[10332]: 2007/06/26_10:01:14 info: Link N2:eth0 up.
...
heartbeat[10332]: 2007/06/26_10:01:15 info: remote resource transition completed.
heartbeat[10332]: 2007/06/26_10:01:15 info: Local Resource acquisition completed.
(none)
heartbeat[10332]: 2007/06/26_10:01:15 info: Initial resource acquisition complete
(T_RESOURCES(them))
```

On constate que N2 voit bien le nœud N1 comme actif et qu'il considère que les ressources sont déjà prises par N1 (remote resource transition completed).

Parallèlement, les logs de N1 montrent que heartbeat sur N1 a bien détecté la présence de N2, et, sur N1, on peut vérifier avec ipvsadm que les 2 serveurs sont actifs.

On obtient un cluster où le load-balancer fonctionne et détecte quand un service Apache est indisponible. La reconfiguration est alors immédiate.

6.3.6 Tests de bon fonctionnement

Test 1 : arrêt du service Apache sur N2

```
N2# /etc/init.d/httpd stop
```

Pas de surprise, comme nous l'avons déjà montré à l'étape 1, le directeur, situé sur N1, supprime N2 du cluster (vérifiez avec la commande ipvsadm sur N1).

Test 2 : déconnexion de N2 du réseau

Pour gagner du temps, on triche en stoppant heartbeat sur N2 :

```
N2# /etc/init.d/heartbeat stop
```

N1 détecte que N2 « disparaît », car, dans les logs, le message suivant s'affiche :

```
heartbeat: WARN: node N2: is dead
heartbeat: info: Dead node N2 gave up resources.
heartbeat: info: Link N2:eth0 dead
```

Test 3 : déconnexion de N1 du réseau

Avant toute chose, on relance heartbeat et Apache sur N2 :

```
N2# /etc/init.d/heartbeat start
N2# /etc/init.d/httpd start
```

N'oubliez pas de tester que le cluster est opérationnel, puis nous allons simuler une panne de N1 :

```
N1# /etc/init.d/httpd/stop
N1# /etc/init.d/heartbeat stop
```

Les logs sur N2 montrent que `heartbeat` a détecté la disparition de N1 et s'est attribué la VIP sur `eth0` :

```
heartbeat: info: Received shutdown notice from 'N1'
...
heartbeat: INFO: Removing conflicting loopback lo
heartbeat: INFO: /sbin/ip -f inet addr delete $VIP dev lo
heartbeat: INFO: /sbin/ip route delete $VIP dev lo
heartbeat: INFO: /sbin/ip -f inet addr add $VIP/24 dev eth0
...
heartbeat: info: all HA resource acquisition completed
(standby).
```

La commande `ip addr show` montre que N2 a pris la VIP sur l'interface `eth0` et non plus sur `lo:0`.

Le directeur est aussi opérationnel sur N2, mais un seul serveur Web est utilisable : N2 !

```
N2# ipvsadm
IP Virtual Server version 1.2.0 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port      Forward Weight ActiveConn InActConn
TCP  192.168.1.199:http rr
  -> node2.zereso.org:http   Local   1      0          0
```

Test 4 : N1 est à nouveau opérationnel

Comme souvent avec les clusters, si la bascule en cas de panne du service redondé (ici le directeur) se passe correctement, qu'advient-il lorsque celui-ci est rendu à nouveau opérationnel ? Souvent cette opération est manuelle, ce que traduit l'option `auto_failback=off` du fichier `ha.cf`.

Relançons les services `httpd` et `heartbeat` sur N1 :

```
N1# /etc/init.d/heartbeat start
N1# /etc/init.d/httpd start
```

N2 est toujours le directeur et il a détecté la présence du service Apache sur N1, ce que confirme la commande `ipvsadm` sur N2 :

```
N2# ipvsadm
IP Virtual Server version 1.2.0 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port      Forward Weight ActiveConn InActConn
TCP  192.168.1.199:http rr
  -> node1.zereso.org:http   Route   1      0          0
  -> node2.zereso.org:http   Local   1      0          0
```

Mais un test à partir d'un navigateur montre qu'un affichage sur deux « tombe » en timeout : c'est normal, N1 n'a pas pris la VIP sur son interface cachée et sa configuration ARP est incorrecte. Nous reproduisons les commandes effectuées sur N2 au §4.2 :

```
N1# ifconfig lo:0 down
N1# echo 1 > /proc/sys/net/ipv4/conf/eth0/arp_ignore
N1# echo 2 > /proc/sys/net/ipv4/conf/eth0/arp_announce
N1# ifconfig lo:0 $VIP netmask 255.255.255.255 up
```

Le cluster est alors complètement fonctionnel. Notons que si maintenant nous arrêtons `heartbeat` sur N2, puis si nous le relançons, le script `IPaddr2` fait en sorte que N2 se réattribue la VIP sur son interface cachée. N1 redevient le directeur.

6.3.7 Finalisation du démarrage

Notre configuration est presque au point. Un seul petit problème persiste : le test 4 précédent montre qu'au démarrage d'un nœud du cluster, la configuration ARP n'est pas faite et la VIP n'est pas affectée à l'interface cachée.

Vous m'objecterez qu'avec un cluster de deux nœuds seulement, on peut réaliser manuellement ces opérations, mais rien ne vaut quand même une petite automatisation.

Pour cela, je vous propose le script d'initialisation suivant, qui doit être copié et activé au démarrage, avant `heartbeat`, sur tous les nœuds du cluster, qu'ils soient le futur directeur ou non. Ce script examine le contenu du fichier `haresources` et en extrait la liste de toutes les VIP associées aux ressources `IPaddr2` et `ldirectord`. Puis, il associe chaque VIP sur l'interface loopback et règle les paramètres ARP des interfaces Ethernet.

```
#! /bin/sh
#
# lvsdr    Start/Stop network configuration for LVS with Direct-Return
#
# chkconfig: 345 74 06
# description: just set some network parameters in case this host belongs to
# a LVS cluster. We assume the cluster is managed by heartbeat and this
# host may also run heartbeat. This script must be launched before heartbeat.
#

# Source function library.
. /etc/init.d/functions

# For RedHat like distros:
# Get config.
if [ -f /etc/sysconfig/network ]; then
    . /etc/sysconfig/network
else
    echo $"Networking not configured - exiting"
    exit 1
fi

# Check that networking is up.
if [ "$NETWORKING" = "no" ]; then
          exit 0
fi

# Our stuff starts here:
prog="lvsdr"

# You can change here the Default Interface if it is not specified by IPaddr2:
DFLT_INTERFACE=eth0

declare -a VIPS INTERFS

# Try to determine the VIP and the INTERFACE if they are not set:
if [ "$VIPS" = "" ]; then
          HA_INIT=/etc/init.d/heartbeat
          if [ -f $HA_INIT ]; then
               HA_DIR=`grep "^HA_DIR=" $HA_INIT 2>/dev/null`
```

```
            HA_DIR=`echo $HA_DIR | cut -d';' -f1 | cut -d'=' -f2`
            LINES=(`sed -r -n \
            "/IPaddr2::.*ldirectord/s/.*IPaddr2::([^[:space:]]*).*/\1/p" \
            $HA_DIR/haresources | tr -s '\t ' `)

            IND=0
            while [ $IND -lt ${#LINES[@]} ]; do
                  #echo ${LINES[$IND]}
                  VIPS[$IND]=`echo ${LINES[$IND]} | cut -d/ -f1`
                  INT=`echo ${LINES[$IND]} | cut -d/ -f3`
                  if [ "$INT" != ${VIPS[$IND]} ]; then
                        INTERFS[$IND]=$INT
                  fi
                  IND=$[ $IND + 1 ]
            done
      fi
fi

if [ ${#VIPS[@]} -eq 0 ]; then
    echo "Warning: could not determine VIP to handle in file $HA_DIR/
haresources"
    exit 0
fi

RETVAL=0

check_interface() {
      if [ -d /proc/sys/net/ipv4/conf/$INTERFACE ]; then
            return 0
      fi

      echo "Interface $INTERFACE not present"
      return 1
}

start() {
      echo -n $"Starting $prog: "
        IND=0
        while [ $IND -lt ${#VIPS[@]} ]; do
            INTERFACE=${INTERFS[$IND]:-eth0}
            if check_interface $INTERFACE; then
                  ip addr add ${VIPS[$IND]}/32 \
                  brd 255.255.255.255 dev lo 2>/dev/null
                  echo 1 > /proc/sys/net/ipv4/conf/
                        $INTERFACE/arp_ignore
                  echo 2 > /proc/sys/net/ipv4/conf/
$INTERFACE/arp_announce
            fi
            IND=$[ $IND + 1 ]
      done
      success
      echo
      return $RETVAL
}


stop() {
      echo -n $"Stopping $prog: "
        IND=0

        while [ $IND -lt ${#VIPS[@]} ]; do
            INTERFACE=${INTERFS[$IND]:-eth0}
            if check_interface $INTERFACE; then
                  ip addr del ${VIPS[$IND]}/32 \
                  brd 255.255.255.255 dev lo 2>/dev/null
                  echo 0 > /proc/sys/net/ipv4/conf/
                        $INTERFACE/arp_ignore
                  echo 0 > /proc/sys/net/ipv4/conf/
                        $INTERFACE/arp_announce
            fi
            IND=$[ $IND + 1 ]
      done
      success
      echo
      return $RETVAL
}

status() {
      ip addr show | grep "${VIPS[0]}.*lo$" >/dev/null 2>&1
      if [ $? -eq 0 ]; then
            echo "$prog is running"
      else
            echo "$prog is not running"
      fi

      return 0
}

restart() {
      stop
      start
}

# See how we were called.
case "$1" in
  start)
      start
      ;;
  stop)
      stop
      ;;
  status)
      status
      ;;
  restart|reload)
      restart
      ;;
  *)
      echo $"Usage: $0 {start|stop|status|restart|reload}"
      exit 1
esac

exit $?
```

Le script est disponible à l'adresse : **http://www.maje.biz/downloads/lvsdr**.

6.3.8 Gestion de la persistance des sessions (bis)

Au chapitre §4.3, nous avons déjà évoqué ce besoin. Dans notre nouvelle architecture, le directeur est redondé, mais qu'advient-il des sessions en cours si N1 est indisponible ? Le directeur est activé sur N2 qui démarre donc avec une table de sessions vide.

Pour éviter cela, il faut lancer les commandes suivantes :

```
N1# ipvsadm --start-daemon master
N2# ipvsadm --start-daemon backup
```

Comment faire en sorte que ces commandes soient lancées correctement au démarrage ? Vous pouvez ajouter la commande suivante dans la fonction `start` du script `lvsdr` précédent :

```
ipvsadm --start-daemon backup
```

Puis, on modifiera le script `$HA_DIR/resource.d/ldirectord` pour que le `case` principal devienne :

```
case "$1" in
  start)
    ipvsadm --stop-daemon backup
    ipvsadm --start-daemon master
    action "Starting ldirectord" $DAEMON $2 start
    ;;
  stop)
    ipvsadm --stop-daemon master
    ipvsadm --start-daemon backup
    action "Stopping ldirectord" $DAEMON $2 stop
    ;;
...
esac
```

7. Outils complémentaires

7.1 Supervision du cluster avec Ganglia

Une fois que le cluster est en production, vous pouvez mettre en œuvre l'outil Ganglia **[GANG]** pour visualiser l'état global du cluster, ainsi que la distribution de la charge. Ganglia charge très peu les nœuds et est hautement dimensionnable (dans notre cas, avec deux nœuds, ces qualités ne seront pas vraiment mises en évidence !).

7.1.1 Architecture

Ganglia est constitué de trois composants :

- `gmond` : installé sur chaque nœud, il collecte les métriques de performances (charge CPU, occupation mémoire...)
- `gmetad` : installé uniquement sur un nœud particulier (ou n'importe quelle machine) qu'on appellera le « *cluster manager* », il collecte les indicateurs mis à disposition par les démons `gmond` (par *multicast* sur le port 8649/tcp). Notons, qu'avec une architecture plus complexe, les démons `gmetad` peuvent communiquer entre eux pour consolider les indicateurs en créant une architecture arborescente.
- Une interface de consultation Web, écrite en PHP. Cette interface est à l'écoute du port 8651/tcp et reçoit les données provenant du démon local `gmetad`.

7.1.2 Installation

Sur le cluster manager, installez PHP, Apache et RRDTool **[RRDT]**.

Puis, téléchargez l'archive `ganglia-3.0.4.tar.gz` et exécutez les commandes suivantes :

```
# tar xvfz ganglia-3.0.4.tar.gz
# cd ganglia-3.0.4
# ./configure --with-gmetad
# make
# make install
```

Sur les autres nœuds du cluster, installez uniquement `gmond` :

```
# tar xvfz ganglia-3.0.4.tar.gz
# cd ganglia-3.0.4
# ./configure
# make
# make install
```

(Un petit partage réseau s'impose, si vous avez beaucoup de nœuds à installer !)

7.1.3 Configuration

Le démon `gmetad` est configuré via le fichier `/etc/gmetad.conf` qui contient au minimum une ligne qui donne un nom au cluster et liste les nœuds :

```
data_source "Mon Cluster" N1 N2
```

Les démons `gmond` sont configurés par les fichiers `/etc/gmond.conf` : ce fichier est plus « copieux » que `gmetad.conf`. La commande `gmond -m` affiche les métriques qui peuvent être remontées par `gmond`. `gmond -t` affiche la configuration à l'écran. La configuration par défaut devrait vous satisfaire dans un premier temps.

Si `gmetad` et `gmond` doivent être activés au démarrage de la machine, recopiez les scripts d'initialisation :

```
# cp gmetad/gmetad.init /etc/init.d/
# cp gmond/gmond.init /etc/init.d/
```

puis, créez les liens symboliques adéquats avec les commandes de gestion des services de votre distribution.

7.1.4 L'interface Web

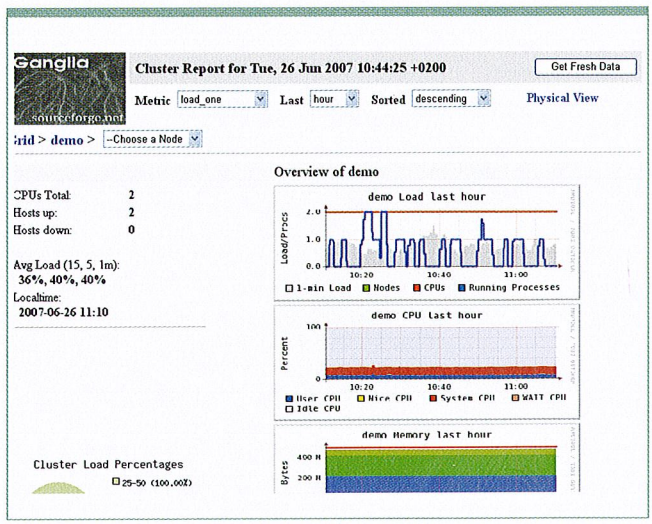
Passez en revue le contenu du fichier `web/conf.php`, puis ajoutez un `Alias` à votre configuration d'Apache :

```
Alias /ganglia/ /chemin/vers/le/répertoire/web/
```

Redémarrez Apache :

```
# /etc/init.d/httpd reload
```

Vous pouvez alors accéder à la page d'accueil qui donne l'état du cluster. Vous pouvez sélectionner une période de temps spécifique ou bien obtenir les détails sur les performances de chacun des nœuds.



7.2 Exécuter des commandes sur tous les nœuds avec DSH

Nous venons de mettre en place un cluster constitué de deux nœuds, mais l'envie va vous gagner de réaliser des clusters plus ambitieux.

Dans ce cas, il est fort probable que vous voudrez exécuter des commandes sur tous les nœuds du cluster. Une boucle en `bash` fera certainement l'affaire, mais vous pouvez aussi utiliser la commande `dsh` (*Distributed Shell*) **[DSH]**.

Ce petit utilitaire permet d'exécuter, éventuellement en parallèle, des commandes sur des machines pré-listées.

Par exemple, si le fichier `/etc/dsh/machines.list` contient N1 et N2, la commande suivante :

```
# dsh -a -r ssh date
```

exécute la commande `date` via un `ssh` sur N1 et N2. Il est aussi possible de définir des groupes de machines ou bien d'utiliser les groupes NIS.

L'installation requiert la bibliothèque `libdshconfig` disponible sur le même site. La compilation des deux packages s'effectue par l'exécution de la « sainte » trilogie :

```
# ./configure
# make
# make install
```

8. Pour aller plus loin

Dans cet article, nous avons surtout insisté sur les problématiques réseau et, en particulier, sur les détails concernant la paramétrisation des piles TCP/IP des nœuds du cluster. Le cluster ainsi construit est tout à fait opérationnel, mais vous pouvez renforcer encore sa robustesse avec les techniques suivantes :

- L'utilisation du *plug-in* `ipfail` : permet de basculer les ressources, non pas quand un nœud est détecté indisponible, mais quand sa connectivité avec des machines distantes (des *ping nodes*) n'est plus assurée **[IPFAIL]**
- La mise en œuvre de liens réseau redondants (*bonding*) : avec un seul lien réseau, la carte Ethernet et le port du switch auquel elle est connectée sont des SPOF ! Il faudrait doubler les ports ou les cartes Ethernet et connecter chaque lien à un switch différent. Le module `bonding`, fourni en standard avec le noyau Linux permet de réaliser cette opération simplement, en configurant les liens en mode actif/passif ou actif/actif **[HS18]**.
- La mise en œuvre de technique de « *fencing* » (que je traduirais par « extinction préventive ») : le pire cas pour un cluster de deux nœuds se produit quand chacun des nœuds considère que l'autre est indisponible. Ils décident alors d'acquérir la ressource simultanément. Imaginez les dégâts si la ressource est un système de fichiers non distribué comme ext3... Pour éviter cela, on met en place des équipements gérables à distance (*power switch*, cartes d'administration iLO, Drac...) qui permettent au premier nœud qui considère que l'autre est « mort » de l'éteindre électriquement. Ce procédé est appelé « STONITH » (pour *Shoot The other One In The Head*) avec `heartbeat` et nécessite des directives supplémentaires dans le fichier `ha.cf`.

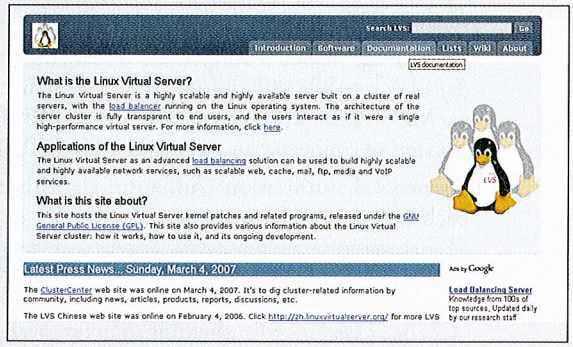
Il y aurait en outre beaucoup à rajouter sur `heartbeat`, sa configuration et les commandes de gestion complémentaires. Un article ultérieur pourra montrer la nouvelle architecture de la version 2 et ses spécificités. En attendant, il ne tient plus qu'à vous d'assurer la redondance de vos services réseau !

Jérôme Delamarche,

jdelamarche@maje.biz

RÉFÉRENCES

- **[NTP]** http://tldp.org/HOWTO/TimePrecision-HOWTO/ntp.html
- **[LVS]** http://www.linuxvirtualserver.org



- **[OCF]** http://opencf.org
- **[ARP]** http://kb.linuxvirtualserver.org/wiki/Using_arp_announce/arp_ignore_to_disable_ARP
- **[PROC]** DAUDEL (Olivier), */proc et /sys*, O'Reilly.
- **[LINCLU]** KOPPER (Karl), *Linux Enterprise Cluster*, No Starch Press.
- **[HS18]** *GNU/Linux Magazine France*, hors-série n°18.
- **[CPAN]** http://www.perl.com/doc/manual/html/lib/CPAN.html
- **[HRBT]** http://www.linux-ha.org
- **[GANG]** http:/ganglia.info
- **[RRDT]** http://oss.oetiker.ch/rrdtool/download.en.html
- **[DSH]** http://www.netfort.gr.jp/~dancer/software/downloads/
- **[IPFAIL]** http://pheared.net/devel/c/ipfail/, http://www.ultramonkey.org/3/ipfail.html