Solution Haute Disponibilité pour Linux : Un cluster avec Heartbeat et Drbd

Nicolas Schmitz
Centre de Ressources Informatiques
Ecole Centrale de Nantes, 44000 Nantes
Nicolas.Schmitz@ec-nantes.fr

Résumé

Pour assurer la haute disponibilité d'un service, on utilisera ici deux serveurs identiques, disposant chacun de ressources en disque suffisantes pour assurer le service. Ces deux serveurs seront reliés par une interface réseau gigabit dédiée et un lien série. Grâce à ces liaisons le logiciel Heartbeat pourra surveiller l'état des deux serveurs, et lancer les procédures de bascule d'un serveur à l'autre en cas de panne. C'est le logiciel DRBD qui garantira la disponibilité des données lors des bascules, grâce à son implémentation très performante de raid1 sur ip. Grâce au travail qu'accomplit DRBD au niveau block device, le couple Heartbeat/DRBD permet de rendre hautement disponible à peu près n'importe quel service.

Les avantages de cette solution sont sa robustesse, son excellent support communautaire, l'économie de matériels couteux de type SAN , et son caractère 100% libre.

Mots clefs

Haute disponibilité, Linux, HA, DRBD, Heartbeat, cluster, Debian, raid1

1 Problématique

Assurer la haute disponibilité d'un service signifie notamment être capable d'assurer la continuité du service malgré une panne du serveur sur lequel il est situé. Il s'agit donc, en général, de doubler un maximum d'éléments matériels du système (habituellement, on double le serveur) et de prévoir des mécanismes de bascule d'exploitation du matériel en panne vers celui de réserve. Nous partons du principe que le service assuré est critique et que des procédures de bascules automatiques sont nécessaires : la bascule doit être déclenchée immédiatement après la détection de la panne.

Le principal défi dans le cas de services réseaux impliquant la manipulation intensive de données (par exemple un serveur imap, une base sql) est donc de s'assurer que les données qui étaient présentées à l'utilisateur avant la panne soient toujours disponibles, et intègres, lorsque le service sera de nouveau rendu (normalement quelques secondes plus tard).

Une solution couramment utilisée est de placer les données sur un équipement SAN accessible depuis les deux serveurs : en cas de panne du serveur actif, le serveur de secours retrouvera les données à jour sur le SAN. Cependant cette solution présente plusieurs inconvénients :

- Elle est chère pour une structure comme la nôtre (Ecole d'Ingénieur, 2000 comptes)
- Elle peut représenter un SPOF (Single Point Of Failure) si elle vient à tomber en panne. Redonder complétement une baie SAN est possible, jusqu'à la double connexion au serveur, mais alors les prix s'envolent de manière complètement démesurée, et on a vu des cas où même avec tout cet équipement, une panne survenait...

Une autre solution est de disposer de deux serveurs avec leurs propres espaces disques, et de procéder à des synchronisations régulières (type rsync) entre les deux machines, dans le sens serveur actif=> serveur passif. En cas de crash, le service pourra démarrer, mais avec des données datant de la dernière synchronisation.

La solution que nous allons étudier ici répond à deux impératifs : ne pas avoir de SPOF dans le cluster, et avoir des données parfaitement à jour en cas de bascule du service vers la deuxième machine.

2 Description générale de la solution

La configuration matérielle de notre solution est la suivante : deux serveurs identiques, disposant chacun de ressources en disque suffisantes pour assurer le service.

En temps normal, un seul de ces deux serveurs (serv-a) rend effectivement le service : il dispose de l'adresse IP sur laquelle est disponible le service, le système de fichier contenant les données est monté, et les différents services réseaux sont lancés. L'autre machine (serv-b) au contraire se contente d'attendre. Les deux machines s'informent mutuellement de leur fonctionnement par un système de « battement de coeur » implémenté par le logiciel « Heartbeat ».

Lorsqu'une panne survient sur A, la machine B détecte l'arrêt des battements de coeur et lance une procédure de bascule : B va acquérir l'adresse IP du service, monter le système de fichier, et lancer les services réseaux rendus par le cluster. La configuration de Heartbeat est simple : ce mécanisme de bascule est donc facile à implémenter, à une exception près : le système de fichier que l'on monte sur B

doit contenir exactement les mêmes données que celui de A au moment du crash. C'est là que DRBD joue son rôle.

DRBD fonctionne comme une sorte de raid1 sur IP. Le mécanisme est le suivant : le block device monté sur A n'est pas /dev/sda1, par exemple, mais /dev/drbd1. De fait, quand le serveur maître écrit sur le système de fichier « drbdisé », il écrit à la fois sur son propre /dev/sda1, mais également sur le /dev/sda1 de l'esclave. L'esclave dispose ainsi en temps réel d'une copie des données.

Ce raid1 sur IP s'accompagne d'une gestion intelligente des synchronisations : quand un serveur est temporairement retiré puis ré-attaché au cluster, seules les données modifiées entre temps sont synchronisées. Cela nous garantit des temps de synchronisation faibles, même pour des volumes importants (à peine quelques minutes pour un volume de 1Tera)

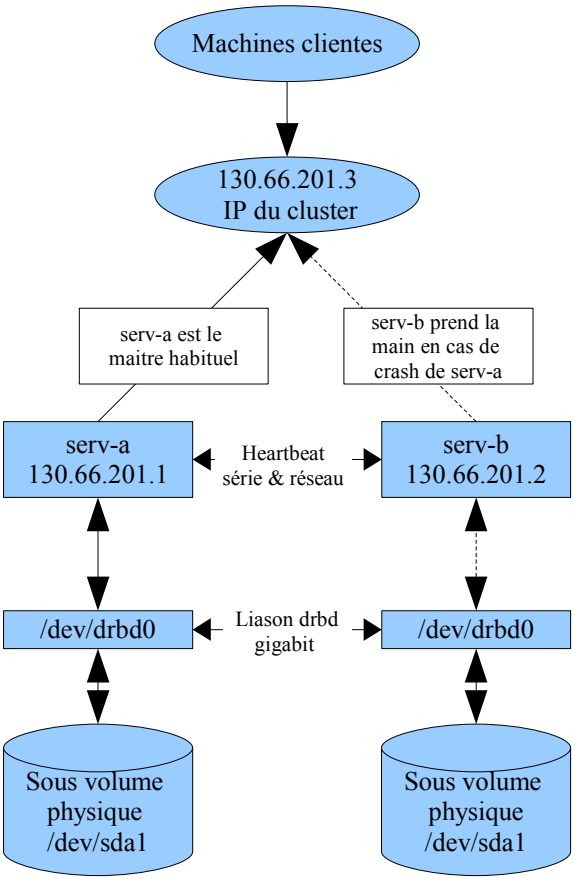

Figure 1 – Schéma général de notre cluster.

3 Installation et configuration de Heartbeat

Heartbeat est le logiciel « gestionnaire » du cluster : c'est lui qui décide des actions nécessaires à la survie du service.

Il gére une adresse IP « virtuelle » : en plus des adresses IP des membres du cluster, cette troisième adresse est attachée au membre maître. C'est sur cette adresse IP qu' il faudra faire pointer les clients. Lors d'une bascule de cette adresse IP d'un membre à l'autre, Heartbeat met à jour les caches arp des éléments actifs du réseau par l'envoi de paquets arp gratuits (émission d'un « flood » de « gratuitous arp »).

Pour prendre ses décisions, Heartbeat se base notament sur un échange permanent de battements de cœur entre les deux machines. La première chose à faire est donc de fiabiliser cet échange : on va avoir besoin de deux liens physiques entre les deux machines pour faire passer ces battements de cœur. En pratique, une liaison Ethernet classique (pas forcément utilisée que pour Heartbeat) et une liaison série font l'affaire.

Assurez-vous que le lien série entre les machine A et B fonctionne :

Sur la machine A :

```
cat < /dev/ttyS0
```

Sur machine B :

```
echo toto > /dev/ttyS0
```

« toto » doit apparaître sur l'écran de A.

3.1 Installation :

Sous Debian, la commande `apt-get install heartbeat` fera l'affaire.

3.2 Edition de /etc/ha.d/ha.cf

Vous pouvez maintenant éditer le principal fichier de configuration, /etc/ha.d/ha.cf. Voici un exemple de configuration commenté :

```
logfacility        local0
```

« facility » utilisée pour syslog

```
keepalive          2
```

2 secondes entre chaque battement de cœur

```
deadtime           10
```

au bout de 10s sans battement de cœur, une machine est déclarée « morte » : si c'est le maître, les services doivent basculer vers l'esclave.

```
warntime           6
```

au bout de 6s sans battement de cœur, une alerte sera générée dans les logs.

```
initdead        60
```
si l'on est en phase de démarrage de la machine, le délai normal (deadtime) est augmenté à 60s pour prendre en compte les éventuelles lenteurs de démarrage de l'autre nœud ou du réseau.

```
udpport         694
```
port udp qui sera utilisé par les battements de cœur sur ethernet (694 par défaut)

```
bcast   eth0
```
lien réseau utilisé pour les battements de cœur.

```
serial  /dev/ttyS0
```
lien série utilisé pour les battements de cœur.

```
auto_failback   on
```
en cas de crash du maître, les ressources basculent vers l'esclave. Avec ce paramètre, on demande à Heartbeat de rebasculer les ressources sur le maître quand celui-ci réapparaît. Avec la valeur par défaut (off), les ressources resteraient sur l'esclave sans une intervention manuelle.

```
node    serv-a
node    serv-b
```
les noms (obtenus avec uname -n) des deux nœuds de notre cluster

Voilà, c'est tout pour ce fichier de configuration de 11 lignes. Ce fichier peut différer selon le noeud du cluster, mais dans notre exemple il est identique sur les deux noeuds.

3.3 Edition de /etc/ha.d/haresources

Ce fichier indique quels sont les services à rendre hautement disponibles, quelle adresse IP acquérir, et le maître habituel. La plupart du temps il ne contient qu'une seule ligne. Il a une importance capitale, puisqu'il est la description exacte des services rendus par le cluster.

Le premier élément de la ligne est le nom (obtenu avec uname -n) du maître habituel des ressources qui suivent. Tous les autres éléments sont des appels à des scripts présents dans /etc/ha.d/resource.d ou /etc/init.d/. En cas de bascule, ces scripts seront appelés :

- dans l'ordre où ils sont énumérés et avec l'argument « start » sur le noeud qui prend la main.
- dans l'ordre inverse avec l'argument « stop » sur le noeud qui rend la main.

Example d'un fichier haresource:

```
serv-a  drbddisk ::user
Filesystem ::/dev/drbd0 ::/user ::ext3
IPaddr2 ::130.66.201.3/24/eth0:0 cyrus mysql-debian
MailTo::root
```

(malgré les apparences, ces éléments tiennent en fait en une seule ligne)

Détaillons les éléments de cette ligne :

`serv-a` : le nom de la machine qui est habituellement maître des ressources qui suivent.

`drbddisk ::user` : Instruction au script drbddisk de rendre disponible la ressource DRBD « user » (nous verrons ce point en partie 4.2).

`Filesystem ::/dev/drbd0 ::/user ::ext3` : Instruction au script « Filesystem » de monter la partition /dev/drbd0 dans /user avec le système de fichier ext3.

`IPaddr2 ::130.66.201.3/24/eth0:0` : Instruction au script IPaddr2 d'acquérir l'adresse IP 130.66.201.3 avec un masque /24 sur l'interface eth0. Cette adresse IP va venir s'ajouter à celle déjà définie pour eth0.

`cyrus` : lancement du serveur cyrus (avec le script /etc/init.d/cyrus).

`mysql-debian` : lancement du serveur mysql (avec le script /etc/ha.d/resource.d/mysql-debian).

`MailTo ::root` : Lancement du script MailTo qui permet d'envoyer un mail indiquant le changement d'état du cluster à root.

Le fichier doit être exactement le même sur les deux nœuds du cluster.

3.4 Edition de /etc/ha.d/authkeys

Troisième et dernier fichier de configuration, le fichier /etc/ha.d/authkeys sert à « authentifier » les battements de coeur que les deux noeuds s'envoient. 3 types d'authentification sont possibles : sha1, crc et md5. Dans notre cas le réseau est sûr, on utilise donc simplement crc :

```
auth    1
1       crc
```

Il faut aussi protéger ce fichier :

```
chmod 600 /etc/ha.d/authkeys
```

Le fichier doit être exactement le même sur les deux nœuds du cluster.

3.5 Tests de bon fonctionnement :

Maintenant que les fichiers de configuration sont prêts, on peut faire quelques essais ; cependant notre fichier haresources n'est pas adapté car nous ne disposons pas

encore des outils DRBD. On va donc provisoirement remplacer le contenu de haresources par :

```
serv-a  IPaddr2 ::130.66.201.3/24/eth0:0 apache2
```

=> On demande simplement à Heartbeat de rendre hautement disponible apache2 sur l'adresse 130.66.201.3

Comme toujours veiller à maintenir haresources identique sur les deux noeuds du cluster.

Les deux machines peuvent être différentes au niveau matériel, mais il est vraiment recommandé que la distribution et la version des logiciels soient identiques.

On peut maintenant éteindre apache2 sur nos deux noeuds :

```
/etc/init.d/apache2 stop
```

et demander aux systèmes des deux noeuds de ne plus gérer le démarrage/l'extinction de apache2, puisque c'est Heartbeat qui s'en chargera :

```
serv-a:/etc/ha.d#update-rc.d -f apache2 remove
update-rc.d: /etc/init.d/apache2 exists during rc.d purge (continuing)
 Removing any system startup links for /etc/init.d/apache2 ...
   /etc/rc0.d/K91apache2
   /etc/rc1.d/K91apache2
   /etc/rc2.d/S91apache2
   /etc/rc3.d/S91apache2
   /etc/rc4.d/S91apache2
   /etc/rc5.d/S91apache2
   /etc/rc6.d/K91apache2
```

Démarrons Heartbeat sur nos deux noeuds serv-a et serv-b. Si les liaisons série et ethernet fonctionnent, ils doivent se « reconnaître », on peut le voir dans le fichier /var/log/syslog :

sur serv-a :

```
heartbeat[21823]: info: Link serv-a:eth0 up.
heartbeat[21823]: info: Link serv-b:/dev/ttyS0 up.
heartbeat[21823]: info: Status update for node serv-b: status up
```

sur serv-b :

```
heartbeat[7499]: info: Link serv-b:eth0 up.
heartbeat[7499]: info: Link serv-a:/dev/ttyS0 up.
heartbeat[7499]: info: Status update for node serv-a: status up
```

Rapidement après, serv-a acquiert l'adresse IP 130.66.201.3 et lance apache2 :

```
heartbeat: info: Acquiring resource group: serv-a IPaddr2::130.66.201.3/24/eth0:0 apache2

heartbeat: info: Running /etc/ha.d/resource.d/IPaddr2 130.66.201.3/24/eth0:0 start

heartbeat: info: /sbin/ip -f inet addr add 130.66.201.3/24 brd 130.66.201.255 dev eth0 label eth0:0

heartbeat: info: /sbin/ip link set eth0 up

heartbeat: /usr/lib/heartbeat/send_arp -i 200 -r 5 -p /var/lib/heartbeat/rsctmp/send_arp/send_arp-130.66.201.3 eth0 130.66.201.3 auto 130.66.201.3 ffffffffffff

heartbeat: info: Running /etc/init.d/apache2  start
```

A ce stade, on peut vérifier avec un naviguateur sur un poste client que apache2 écoute bien sur l'adresse IP 130.66.201.3.

Un `ps aux | grep apache2` nous montre aussi que apache2 est lancé sur serv-a et non pas sur serv-b.

Eteignons maintenant brutalement serv-a (via le câble d'alimentation par exemple). serv-b s'en aperçoit et lance l'acquisition de l'adresse IP et de apache2 :

```
heartbeat[7499]: info: Link serv-a:/dev/ttyS0 dead.
heartbeat[7499]: info: Link serv-a:eth0 dead.
heartbeat[7499]: WARN: node serv-a: is dead
heartbeat[7499]: info: Resources being acquired from serv-a.

heartbeat: info: Taking over resource group Ipaddr2::130.66.201.3/24/eth0:0

heartbeat: info: Acquiring resource group: serv-a IPaddr2::130.66.201.3/24/eth0:0 apache2

heartbeat: info: Running /etc/ha.d/resource.d/IPaddr2 130.66.201.3/24/eth0:0 start

heartbeat: info: /sbin/ip -f inet addr add 130.66.201.3/24 brd 130.66.201.255 dev eth0 label eth0:0

heartbeat: info: /sbin/ip link set eth0 up

heartbeat: /usr/lib/heartbeat/send_arp -i 200 -r 5 -p /var/lib/heartbeat/rsctmp/send_arp/send_arp-130.66.201.3 eth0 130.66.201.3 auto 130.66.201.3 ffffffffffff

heartbeat: info: Running /etc/init.d/apache2  start

heartbeat: info: mach_down takeover complete for node serv-a.
```

Et c'est la partie « spectaculaire » de la démonstration : vous pouvez rafraîchir la page dans votre naviguateur et constater qu'apache2 répond toujours sur 130.66.201.3. Des éventuels utilisateurs ne se seraient probablement rendus compte de rien !

On rallume serv-a. Grâce à notre paramètre auto_failback à on, il reprend automatiquement la main :

```
info: Initial resource acquisition complete (auto_failback)
```

serv-b a, quant à lui, « rendu » les services.

Pour identifier rapidement qui est le « maître » dans un cluster Heartbeat, un moyen simple est d'utiliser ifconfig et de regarder si l'adresse IP du cluster est présente ou pas.

3.6 Conclusion :

On voit que l'installation et la configuration de Heartbeat sont vraiment faciles. Il faudra simplement faire attention à toujours maintenir identiques les fichiers haresources et authkeys entre les deux noeuds du cluster.

Dans bien des cas, on pourra l'utiliser sans DRBD, pour assurer la haute disponibilité d'un service avec lequel la réplication des données n'est pas un problème. Par exemple à Centrale Nantes, nous l'utilisons sans DRBD pour assurer la haute disponibilité d'un serveur ftp (en lecture seule...), d'un réplica ldap, d'un webmail, d'un proxy imaps, d'une passerelle smtps, d'un serveur de nom, d'un serveur freeradius... C'est vraiment ce qu'il y a de plus simple !

Cette étude de Heartbeat est loin d'être exhaustive, il existe d'autres options très utiles dans différents contextes, par exemple la fonction ipfail permet de basculer d'un noeud à un autre en ne se basant pas seulement sur un battement de coeur mais aussi sur la disponibilité d'un lien réseau.

Dans sa version 2 (stable depuis aout 2005), Heartbeat intègre [1] deux nouveautés majeures :

- Il permet d'intégrer plus de deux noeuds.
- Il intègre un système de contrôle du bon fonctionnement des services (contrôle applicatif).

On trouvera beaucoup d'autres informations, en anglais, sur le site du projet Heartbeat [2]. A lire également un dossier complet dans un numéro Hors Série de Linux Mag [3].

4 Installation et configuration de DRBD

Une de nos principales contraintes pour ce cluster haute disponibilité est de ne perdre aucune donnée lors de la bascule entre nos deux machines. C'est DRBD qui va répondre à cette contrainte, en répliquant toutes les écritures sur le disque de la machine maître vers le disque de la machine esclave. Pour cela, DRBD se place entre le sous volume physique et le système. Il présente au système un « block device » (du type /dev/drbd0, /dev/drbd1..) que l'on va pouvoir formater, puis monter comme n'importe quel volume.

C'est une des forces de DRBD : une fois que l'on dispose d'un volume DRBD, on peut virtuellement rendre hautement disponible n'importe quel service, sans dépendre des capacités du service à travailler en cluster ou non.

4.1 Installation :

DRBD se compose d'une partie noyau et d'outils de gestion.

Partie noyau :

On peut choisir entre un module ou une inclusion directe dans le noyau. Je ne vais couvrir ici que l'inclusion directe dans le noyau. Voilà comment faire si vous disposez des sources du noyau (noyau de kernel.org) dans /usr/src/linux-2.6.10 :

```
cd /usr/src
wget http://oss.linbit.com/drbd/0.7/drbd-0.7.7.tar.gz
tar xzvf drbd-0.7.7.tar.gz
cd /usr/src/drbd-0.7.7
make KDIR=/usr/src/linux-2.6.10 kernel-patch
cd /usr/src/linux-2.6.10/
patch -p1 < /usr/src/drbd-0.7.7/patch-linux-2.6.10-drbd-0.7.7
```

Il ne reste plus qu'à recompiler le noyau, avec l'option :

```
CONFIG_BLK_DEV_DRBD=y
```

Outils de gestion :

Concernant les outils de gestion de DRBD, on peut utiliser le package pour Debian : `apt-get install drbd0.7-utils`

(Attention, le paquet drbd0.7 a évolué, et il faut que sa version soit la même que celle de la partie noyau. Par exemple en septembre 2005, c'est la version 0.7.10-3, il faut donc installer drbd-07.10 dans le noyau).

4.2 Configuration :

Avant de commencer la configuration, il convient de s'assurer que l'on dispose, sur **chacune** des deux machines :

- Du support DRBD dans le noyau (ou chargé par module)
- Des outils de gestion drbd-utils installés.
- D'une partition prête à être utilisée, non montée (tout son contenu sera ecrasé).

– D'un lien réseau avec l'autre machine, giga et dédié de préférence.

A noter que les disques physiques ne doivent pas forcément être les mêmes sur les deux machines, la règle du plus petit dénominateur commun s'appliquera pour la taille de la partition, mais également pour la vitesse si l'on choisit le protocole C (voir ci-dessous).

On peut maintenant éditer le fichier /etc/drbd.conf, voici mon propre fichier, largement commenté :

```
resource user {
# On commence la déclaration de notre ressource « user ».
# Dans notre cluster, nous n'avons qu'une partition à rendre
# hautement disponible, mais on peut parfaitement avoir
# plusieurs partitions, et dans ce cas plusieurs déclarations
# de type « ressource » dans /etc/drbd.conf

protocol C;
# Trois protocoles sont disponibles :

# En protocole A, l'acquittement d'écriture (sur le maître)
# est envoyé dès que les données ont été transmises au
# sous volume physique et envoyé à l'esclave.

# En protocole B, l'acquittement d'écriture (sur le maître)
# est envoyé dès que les données ont été transmises au
# sous volume physique et reçues par l'esclave.

# En protocole C, l'acquittement d'écriture (sur le maître)
# est envoyé dès que les données ont été transmises au
# sous volume physique ET au sous volume physique de
# l'esclave.

# Le C est donc le plus sûr ; vu la qualité de notre liaison
#(lien giga dédié) on peut l'utiliser sans problème.


    incon-degr-cmd "/sbin/halt -f";
# Si le noeud démarre à la fois « inconsistant » (pas de
# bonnes données en local) et « dégradé » (pas de
# liaison avec le noeud distant), on éteint le noeud.

    startup     {
      wfc-timeout 240;
# Au démarrage du noeud, on attend (cela bloque le
# démarrage) le noeud distant pendant 4 minutes.

      degr-wfc-timeout 240;
                }
# Au démarrage d'un noeud précédemment dégradé on
# attend (cela bloque le démarrage) le noeud
# distant pendant 4 minutes.

    disk        {
      on-io-error   panic;
                }
# Si une erreur d'entrée sortie est rencontrée avec le sous
# volume physique, on freeze la machine et
# Heartbeat basculera l'exploitation sur l'autre machine.

    syncer      {
      rate 700000K;
# On ne limite pas la vitesse de synchronisation.

      al-extents 257;
                }
# al-extent définit la taille de la « hot-area » : DRBD
# stocke en permanence dans les meta-data les zones
# actives du volume physique. En cas de crash ces zones
# seront ainsi resynchronisées dès le retour du
# noeud. Chaque extent définit une zone de 4Mo. Plus al-
# extents est grand, plus la resynchronisation après un
# crash sera longue, mais il y aura moins d'écritures de
# meta-data.
# Avec 257 on a donc 1giga de « hot aera ».

    on serv-a   {
                device    /dev/drbd0;
                disk      /dev/sda1;
# volume physique utilisé, ici sda1.

        Address   130.66.201.1:7788;
# adresse IP avec port TCP sur lequel DRBD va attendre
# les connexions de l'autre noeud. Choisissez ici votre
# interface giga dédiée !

        meta-disk  internal;
                }

# emplacement des meta-data. Si le sous volume physique
# est composé d'un seul disque, mieux vaut placer les
# meta-data sur un autre support. (sinon les écritures de
# meta-data ralentiront tous les accès au disque). Ici le sous
# volume physique est un raid5, pas de problème donc
# avec internal.

    on serv-b   {
      device    /dev/drbd0;
      disk      /dev/sda1;
      address   130.66.201.2:7788;
      meta-disk internal;
              }
    }
```

Le fichier de configuration est identique sur les deux noeuds.

4.3 Tests de bon fonctionnement :

Maintenant que le fichier de config est prêt, on peut démarrer DRBD sur nos deux noeuds :

```
/etc/init.d/drbd start
```

DRBD ne sait pas quel noeud doit devenir maître, les deux noeuds vont donc passer en « secondary » :

```
serv-b:~# cat /proc/drbd
version: 0.7.7 (api:77/proto:74)
SVN Revision: 1680 build by root@serv-a, 2005-09-16
19:28:18
 0: cs:Connected st:Secondary/Secondary ld:Inconsistent
```

```
   ns:0 nr:0 dw:0 dr:0 al:0 bm:118 lo:0 pe:0 ua:0 ap:0
```

et cela va être à nous d'indiquer lequel doit devenir maître, à l'aide de la commande :

```
serv-a:~# drbdsetup /dev/drbd0  primary --do-what-I-say
```

(à exécuter sur le maître, donc ici serv-a)

Une première synchronisation est donc effectuée (dans le sens maître=>esclave) :

```
serv-a:~# cat /proc/drbd
version: 0.7.7 (api:77/proto:74)
SVN Revision: 1680 build by root@serv-a, 2005-09-16 19:28:18
 0: cs:SyncSource st:Primary/Secondary ld:Consistent
    ns:479744 nr:0 dw:0 dr:487936 al:0 bm:565 lo:0 pe:38 ua:2048 ap:0
       [>.....................] sync'ed:  4.7% (9617/10085)M
       finish: 0:11:43 speed: 13,856 (11,416) K/sec
```

Malheureusement, les machines de démonstration ne disposent pas d'un lien giga dédié, ce qui explique qu'ici la vitesse de transfert plafonne à la limite du réseau, c'est à dire 10Mo. Avec nos serveurs réels, on atteint les 40Mo en vitesse de synchronisation.

Une fois la synchronisation effectuée, voilà ce que renvoie /proc/drbd sur le maître :

```
 serv-a:~# cat /proc/drbd
version: 0.7.7 (api:77/proto:74)
SVN Revision: 1680 build by root@serv-a, 2005-09-16 19:28:18
 0: cs:Connected st:Primary/Secondary ld:Consistent
   ns:8776936 nr:0 dw:0 dr:8776936 al:0 bm:1072 lo:0 pe:0 ua:0 ap:0
```

cs: Connected nous indique que DRBD a trouvé le noeud distant.

St:Primary/Secondary nous indique que le noeud est maître, et que le noeud distant est esclave.

Ld:Consistent nous indique que les données sont à jour sur ce noeud.

Pour les autres informations que renvoie /proc/drbd, voir le wiki de linux-ha.org [4].

On peut maintenant formater notre block device (attention, lancez le formattage depuis le maître !) :

```
serv-a:~# mkfs.ext3 /dev/drbd0
```

Le choix du filesystem est libre, cependant il est primordial de prendre un filesystem journalisé (il semble aussi que XFS soit à éviter, même s'il est supporté depuis la version 0.7).

Détail qui a son importance : vous ne pouvez monter /dev/drbd0 que sur le noeud maître. Ne tentez pas d'y accéder depuis le noeud esclave.

Ne tentez pas non plus d'accéder directement au sous volume physique, que ce soit sur le maître ou sur l'esclave.

Si les précédentes manipulations de ce chapitre se sont bien déroulées, voici une série d'opérations que vous pouvez effectuer pour tester votre installation DRBD :

Sur le maître (serv-a):

```
mount /dev/drbd0 /mnt/test
cp /usr/src/linux-2.6.10.tar.gz /mnt/test
md5sum /mnt/test/linux-2.6.10.tar.gz > /root/md5testserv-a
umount /mnt/test
drbdsetup /dev/drbd0 secondary
```

Sur l'esclave (serv-b):

```
drbdsetup /dev/drbd0 primary
mount /dev/drbd0 /mnt/test
ll /mnt/test/
```

=> Le fichier linux-2.6.10.tar.gz doit être présent

```
md5sum /mnt/test/linux-2.6.10.tar.gz > /root/md5testserv-b
```

=> Les contenus des deux fichiers md5test doivent être identiques.

Toujours sur l'esclave, vous pouvez maintenant écrire un fichier dans /dev/drbd0, puis vérifier qu'il est bien propagé sur le maître lorsque celui-ci reprend la main :

Sur l'esclave (serv-b) :

```
cp /boot/vmlinuz-2.6.10 /mnt/test/
md5sum /mnt/test/vmlinuz-2.6.10  > /root/md5test2serv-b
umount /mnt/test
drbdsetup /dev/drbd0 secondary
```

Sur le maître (serv-a) :

```
drbdsetup /dev/drbd0 primary
mount /dev/drbd0 /mnt/test
md5sum /mnt/test/vmlinuz-2.6.10 > /root/md5test2serv-a
```

=> Les contenus des deux fichiers md5test2 doivent être identiques.

4.4 Conclusion :

Lors des nombreux tests effectués, il a été constaté une grande robustesse du système, même dans des situations de stress maximum. Que ce soit pour une base SQL, pour un serveur de fichier, ou encore pour un serveur IMAP, on constate que les services ont toujours pu redémarrer, malgré de multiples « reset » intempestifs en pleine opération d'écriture.

D'un point de vue performance, les tests n'ont pas montré d'écart significatif entre l'écriture sur une partition purement locale et l'écriture sur la partition « drbdisée ». Sur les listes de discussion de DRBD [5], les différents échanges confirment que les baisses de performance sont en général minimes (moins de 10%).

Un des principaux intérêts de DRBD0.7 est sa gestion intelligente des synchronisations. Une fois la synchronisation initiale effectuée, le cluster ne connaîtra plus que des synchronisations limitées aux blocs modifiés, donc très courtes. C'est très appréciable, surtout lorsque l'on a connu DRBD0.6 ou les solutions md+(e)nbd.

Dans sa future version 0.8 [6], DRBD devrait autoriser les écritures simultanées par les deux membres du cluster, à charge pour le Filesystem de le gérer également (GFS...). On devrait également pouvoir utiliser des partitions de plus de 4 Tera.

5 Clusterisation des services :

Une fois que l'on dispose du socle Heartbeat + DRBD, on va pouvoir commencer à y « empiler » des services réseaux. Pour rendre hautement disponible un service, il va falloir effectuer 5 opérations :

1) S'assurer que l'on possède un script d'init pour ce service implémentant la commande « status », soit dans /etc/init.d, soit dans /etc/ha.d/resource.d/. Sur une Debian où les scripts d'init ne possèdent pas cette commande, on peut par exemple créer un script dans /etc/ha.d/resource.d qui gère le « status », et qui renvoie vers le script par défaut pour les autres opérations. Voir un exemple d'un tel script en annexe.

2) Bien entendu, il faut indiquer au service de stocker ses données sur la partition rendue hautement disponible par DRBD. Pour Mysql par exemple, on édite la ligne datadir dans /etc/mysql/my.cnf (ne pas oublier ensuite de copier les données du service sur la partition !).

3) Demander à Heartbeat de gérer le service, en ajoutant le nom du script à la fin de notre ligne dans haresource :

```
serv-a drbddisk::user Filesystem::/dev/drbd0::/user::ext3 Ipaddr2::130.66.201.3/24/eth0:0 mysql-debian
```

4) Indiquer au système de ne plus gérer le démarrage ou l'extinction du service, en supprimant les liens dans /etc/rc2.d ou /etc/rc5.d. Sur Debian, la commande update-rc.d est disponible pour cette opération.

5) S'assurer que le service redémarre correctement après un crash. Imaginons que le crash survienne pendant un INSERT sur une table mysql : celle-ci sera corrompue, y compris sur le noeud survivant (exactement de la même manière que sur une machine isolée). Mysql est la seule application qui m'a posé ce problème, mais il y en a surement d'autres. On peut le résoudre en demandant à mysql, non seulement de vérifier les tables lors de son lancement, mais aussi de les réparer automatiquement (il suffit de rajouter un « -r » dans le script de lancement /etc/mysql/debian-start). Pour le reste des applications que j'utilise, ext3 fait bien son travail et le noeud survivant peut directement lancer l'exploitation.

6 Conseils et astuces :

6.1 Gestion des pannes de courant :

Un point important à traiter dans ce dispositif de haute disponibilité est celui de la gestion des coupures de courant. La configuration des onduleurs et des différents modes de rallumage automatique des machines devra se faire avec la plus grande attention, afin d'éviter par exemple qu'une machine ne soit allumée intempestivement et commence à rendre le service avec des données fausses. La configuration que nous avons choisie est la suivante :

le maître possède son propre onduleur, qu'il gère via l'outil apcupsd. En cas de panne de courant, le maître éteint assez rapidement l'esclave via une commande shutdown en ssh. L'esclave n'est pas capable de repartir de lui même, c'est le maître qui lancera une commande wake-on-lan quand l'onduleur aura atteint un niveau de chargement suffisant. Le maître quant à lui s'éteint lorsqu'il ne reste que 5% de batterie, et est redémarré automatiquement lorsque le courant revient.

6.2 Astuce pour éviter les fsck :

Nos clusters disposent de larges capacités en disque (1Tera). Un fsck sur une partition de cette taille peut être long. Pour éviter que ce fsck ne se déclenche au mauvais moment et n'entraine un long moment d'indisponibilité de service, on peut utiliser tune2fs pour désactiver les fsck automatiques:

```
tune2fs -c 0 /dev/drbd0
tune2fs -i 0 /dev/drbd0
```

A charge pour l'administrateur de lancer des fsck manuels lorsque c'est possible.

6.3 Cluster actif/actif :

Ce point n'est pas traité ici, mais l'on peut aussi mettre en place un cluster de type actif/actif avec Heartbeat et DRBD. Il faut simplement pouvoir diviser les données en deux partitions, pour monter chaque partition sur une machine différente. On aura donc deux lignes dans /etc/ha.d/haresources, et en cas de panne d'un des deux noeuds, toute la charge sera assurée par le noeud survivant.

6.4 Autres mesures simples :

On peut noter que cette duplication des données sur le réseau ne doit pas empêcher de prendre des mesures simples sur chacune des machines : raid5 matériel pour les disques de données, alimentation redondante, doublage des liens réseaux avec du « channel bonding », sécurisation de l'alimentation électrique avec un onduleur, etc...

6.5 Serveurs identiques... ou non :

Autre chose à signaler : il n'est pas nécessaire d'avoir deux machines exactement équivalentes pour mettre en oeuvre ce cluster, on peut parfaitement utiliser un serveur dimensionné normalement, et un autre moins puissant qui ne servira que pendant les quelques moments où le maître n'est pas disponible. Cependant ce type de configuration est probablement moins facile à mettre en place, les phases de debug étant plus difficiles.

6.6 Support

DRBD et Heartbeat ont une communauté très active autour d'eux. Les listes de diffusion linux-ha et drbd-user sont très bavardes, et les développeurs assurent eux-mêmes le support, en général dans la journée. On peut noter que DRBD a été créé et est développé par les membres d'une SSII autrichienne [7]. Cette société offre des contrats commerciaux de support sur DRBD. Le développeur principal de Heartbeat est quant à lui embauché par IBM en tant que leader de ce projet.

6.7 Gestion des fichiers de configuration :

Une des principales difficultés dans l'administration quotidienne d'un cluster est de toujours faire attention à synchroniser les fichiers de configuration. Par exemple, si l'on modifie un réglage dans un fichier de configuration d'un service sur le maître, il faut penser à recopier ce fichier sur l'esclave ; sans quoi le jour où le service basculera sur l'esclave, ce dernier n'aura pas les bons paramètres de fonctionnement... La commande « scp » et l'authentification par clef publique/clef privée deviennent donc très vite les meilleurs amis de l'administrateur de cluster. Si cela ne suffit pas, il existe des outils comme csync2 [8] ou cfengine [9] qui peuvent aider à maintenir les fichiers de configuration synchronisés.

6.8 Phases de tests :

Avant de mettre en production le cluster, la phase de test doit être la plus complète possible. Pour le premier cluster que j'ai installé, j'ai passé une semaine complète à le tester (reset désordonnés, simulations de pannes de courant...), à faire des benchmarks... La réalisation d'une machine à état est également d'un grand secours.

7 Conclusion :

L'installation d'un cluster Heartbeat/DRBD n'est pas une mince affaire ; même si les grands concepts sont assez facilement assimilables, il y a beaucoup de petits détails à gérer. Cela prend donc du temps, et la phase de tests ne doit surtout pas être négligée.

Après plus d'un an d'exploitation, le bilan est largement positif, à un regret près : la version d'Heartbeat (1.2.3) que nous avons déployée sur nos clusters n'intègre pas de contrôle du bon fonctionnement des services. Nous nous sommes donc retrouvés plusieurs fois avec un noeud dont un service critique était planté mais comme le système d'exploitation fonctionnait correctement, aucune bascule n'était initiée... On peut résoudre ce problème en ajoutant à notre solution un troisième produit : Mon [10]. Mon va surveiller les services sur le noeud actif, et déclencher la bascule vers l'autre noeud en cas de panne du service.

Malgré ce problème, on dispose d'une solution vraiment complète, fiable, facile à administrer, et n'impliquant que des matériels standards. Le fait que la solution soit 100% logiciel libre est également un avantage.

Annexe :

Script permettant de gérer la commande status avec mysql sous Debian :

```
#!/bin/sh
#
#       High-Availability MySQL-Debian control script
#
# Description:  start/stop MySQL server using Debian
scripts.
#
# Author:       Sebastien Bechet
# Support:      sebastien.bechet@osinix.com
# License:      GNU Lesser General Public License (LGPL)
# Copyright:    (C) 2004 OSINIX
# Depends:      apt-get install mysql-client
#
# An example usage in /etc/ha.d/haresources:
#       node1  10.0.0.170 mysql-debian
#

NAME=mysql
INITDFILE=/etc/init.d/$NAME
PIDFILE=/var/run/mysqld/${NAME}d.pid
MYADMIN="/usr/bin/mysqladmin --defaults-extra-
file=/etc/mysql/debian.cnf"
CMD="$1"

trap "" 1

case "$CMD" in
   start)
      $INITDFILE start
   ;;
   stop)
      $INITDFILE stop
   ;;
   status)
      if [ -f $PIDFILE ]; then
         ping_output=`$MYADMIN ping 2>&1`;
ping_alive=$(( ! $? ))
         ps_alive=`ps cax | grep -c ' mysqld$'`
         if [ $ping_alive == 1 ] ||
           [ $ps_alive != 0 ]; then
            echo "running"
         else
            echo "stopped"
         fi
      else
         echo "stopped"
      fi
   ;;
   *)
      echo "Usage: mysql-debian {start|stop|status}"
      exit 1
   ;;
esac
exit 0
```

Bibliographie :

[1] http://linux-ha.org/NewHeartbeatDesign

[2] http://linux-ha.org Voir particuliérement http://linux-ha.org/GettingStarted

[3] Denis Bodor, Utilisez Heartbeat pour assurer un service. *Linux Magazine*, Hors Série n°18, Février 2004

[4] http://wiki.linux-ha.org/DRBD_2fFAQ#head-854cf16e5a3dbf7376642a4173c9c38e0b11a8e7

[5] Liste drbd-user : http://lists.linbit.com/mailman/listinfo/drbd-user

[6] http://svn.drbd.org/drbd/trunk/ROADMAP

[7] SSII Linbit : http://www.linbit.com/

[8] Logiciel csync2 : http://oss.linbit.com/csync2/

[9] Logiciel cfengine : http://www.gnu.org/software/cfengine/cfengine.html

[10] Logiciel Mon : http://www.kernel.org/software/mon/