

# Notes de travail Projet 150h

## Plan:

1. Introduction
2. Redondance DHCP/DNS
3. Redondance Samba/Ldap
4. Mise en oeuvre des Machines Virtuelles
5. Installation de Debian Etch sur les NEC
6. ANnexes

## **I/. Introduction**

### Objectifs:

Etude d'une solution de redondance de serveurs de production (SAMBA/LDAP, DNS/DHCP, PF sous FreeBSD) avec virtualisation des backups.

### Axe de travail:

La spécificité d'un serveur SAMBA et sa criticité particulière en raison du stockage des données utilisateurs nous ont porté à nous attacher en premier lieu à la redondance de celui-ci. Nous nous sommes ensuite penchés sur la redondance DNS/DHCP.

### Hôtes utilisés:

- hôte des machines virtuelles: **valinor**
- machines virtuelles:
  - \* DNS/DHCP: **vm-olorin**, IP:10.2.0.210, Cluster: 10.2.0.10
  - \* SAMBA/LDAP: **vm-varda**, IP:10.2.0.202, Cluster: 10.2.0.02

### Logiciels utilisés:

- DRBD v8.2.4

**DRBD** (Distributed Replicated Block Device) est un mécanisme de réplication de données localisées sur deux serveurs distincts par voie réseau. Pour simplifier, il s'apparente à du RAID-1 sur IP. Quand une écriture a lieu sur le disque du serveur maître, l'écriture est simultanément réalisée sur le serveur esclave. La synchronisation est faite au niveau de la partition.

Cependant, il est nécessaire de disposer d'une couche supérieure qui va permettre d'une part de créer un cluster pour ces deux serveurs, et d'autre part de décider lequel des deux sera maître. Pour cela, nous utiliserons Ucarp.

#### **Liens web:**

[DRBD sur wapiti.telecom-lille](#)  
[Site officiel DRBD](#)

- Ucarp v1.4

**UCARP** est un programme de haute-disponibilité pour système d'exploitation Unix et dérivés qui permet le partage par plusieurs hôtes d'une même [adresse IP](#) afin d'assurer la continuité

du service en cas de défaillance d'un hôte. Ce programme implémente en espace utilisateur le protocole [CARP](#) disponible sous [OpenBSD](#) et s'oppose au protocole propriétaire [VRRP](#). Le principal problème d'Ucarp (au même titre qu'Heartbeat) est de ne fonctionner qu'au niveau IP. Afin de provoquer un basculement maître/esclave sur une panne d'un ou plusieurs services, nous avons mis en place un script, appelé **UcManager**. Celui-ci lance périodiquement (via le cron) des tests sur les services pour connaître leurs états et prendre les décisions de gestion ad hoc en conséquence. Nous verrons plus loin les stratégies adoptées pour assurer une disponibilité de service adaptée aux besoins définis en préalable de ce projet.

**Liens web:**

[Doc Ubuntu sur Ucarp](#)

**Remarque:** nous avons initialement testé l'utilitaire de clustering Heartbeat, lequel reprend sensiblement les mêmes fonctionnalités qu'Ucarp. Cependant, plusieurs de ses comportements ne nous ont pas semblés satisfaisants, notamment l'absence de rebasculement maître/esclave après reconnexion suite à une perte de connectivité IP entre les 2 machines du cluster.

- Ifplugd 0.28-2.3

**IFPLUGD** est un utilitaire qui permet de gérer le changement d'état d'interfaces réseaux. Nous en aurons besoin pour stopper le service Ucarp d'une machine qui se trouverait isolée du réseau.

**Liens web:**

[Site officiel ifplugd](#)

- RSync

**RSYNC** (*remote synchronization*, synchronisation distante) est un [logiciel](#) de synchronisation de fichiers, distribué sous [licence GPL](#). La synchronisation est unidirectionnelle, c'est-à-dire qu'elle copie les fichiers de la source en direction de la destination. rsync est donc utilisé pour réaliser des [sauvegardes incrémentales](#) ou pour diffuser le contenu d'un répertoire de référence.

Dans le cadre de ce projet, rsync a été choisi pour réaliser la synchronisation entre les serveurs DHCP/DNS pour maintenir la cohérence des configurations de ces derniers. Dans la mesure où celles-ci ne sont pas modifiées fréquemment, il n'est donc pas nécessaire d'envisager de mettre en oeuvre ici une solution de type DRBD.

Script UcManager:

Ce script répond à nos besoins en terme de supervision des différents services. Historiquement, nous avons retenu l'idée de mettre en oeuvre l'utilitaire Mon, lequel prend en charge cette fonctionnalité. Cependant, au cours de nos tests, nous sommes parvenus à la conclusion que nous avons besoin de prendre des décisions différentes selon plusieurs états du service DRBD, ce que ne permet pas Mon. Nous avons donc développé notre propre script qui est lancé périodiquement par le cron. Ce script utilise lui-même les codes de retour d'autres scripts chargés de tester chacun des services nécessaires.

## **II/. Redondance du Serveur DHCP/DNS**

### **2.1 Architecture de la maquette**

Le but est d'arriver à l'architecture suivante (voir Fig.1) où les services DNS et DHCP sont redondés sur une machine virtuelle :

- *ucarp* (démarré en init) se chargeant de créer un cluster entre les deux serveurs,
- le script de monitoring *ucmanager* (lancé périodiquement par le cron) se chargeant d'arrêter les services et de prévenir les administrateurs en cas d'arrêt d'un ou de plusieurs services sur la même machine.

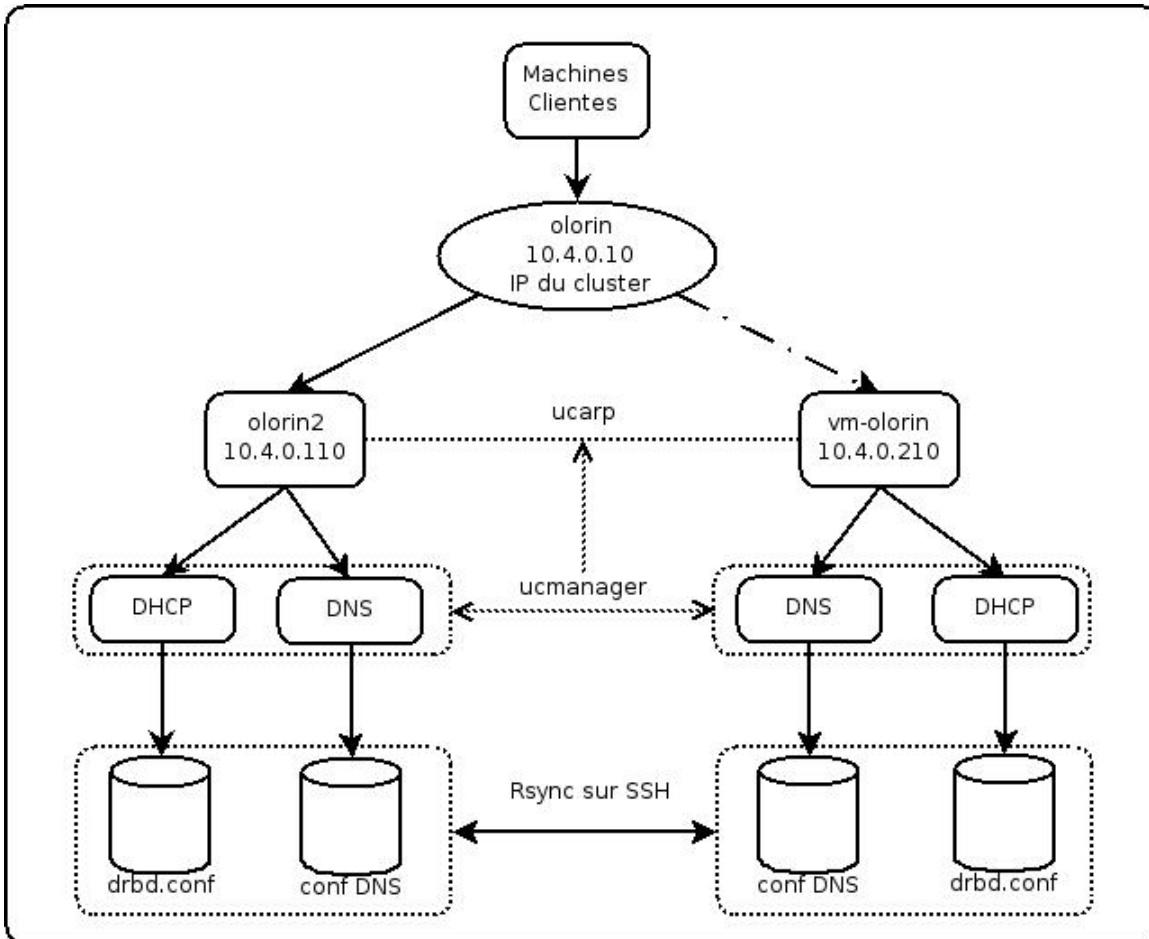


Fig.1

## 2.2 Préparation de l'hôte et de la VM

Il faut avant tout faire quelques petites modifications sur la configuration des deux machines :

- Le package ***dnsutils*** doit être installé sur les deux machines pour avoir accès à la commande ***nslookup*** (pour le script dns.monitor).
- Les services bind9 et dhcp doivent être installés et doivent fonctionner (il faut les tester unitairement).
- Le package ***exim*** (pour le script Mail) doit être installé et configuré (voir annexe exim4 pour l'installation et la configuration).
- Les services DHCP et DNS doivent être supprimés des répertoires ***/etc/rcX.d*** . Ils seront lancés par Ucarp qui lui sera lancé au démarrage.
- De manière à utiliser SSH entre les deux machines, il faut utiliser l'authentification ssh par échange de clés :
  - générer les clefs : ***ssh-keygen -t rsa -b 1024*** ( ***/root/.ssh/id\_rsa*** et passphrase vide)
  - copier la clef publique d'olorin2 sur vm-olorin : ***scp ~/.ssh/id\_rsa.pub root@vm-olorin:/root/***
  - ajouter la clef publique d'olorin2 dans le fichier ***/root/.ssh/authorized\_keys*** de vm-olorin : ***cat ~/id\_rsa.pub >> ~/.ssh/authorized\_keys***
  - effectuer le même processus mais dans le sens vm-olorin > orlorin2

## 2.3 Installation et configuration d'Ucarp

Cette étape a lieu sur les deux machines.

```
apt-get install ucarp
```

Il faut ensuite placer Ucarp au démarrage et à l'arrêt, après avoir créé le script d'init d'Ucarp:

```
In -s /etc/init.d/ucarp /etc/rc2.d/S99ucarp
In -s /etc/init.d/ucarp /etc/rc6.d/K01ucarp
In -s /etc/init.d/ucarp /etc/rc0.d/K01ucarp
```

On crée l'arborescence suivante:

```
`-- etc
  |-- default
  | `-- ucarp --> options de démarrage d'Ucarp
  |-- init.d
  | `-- ucarp --> script de démarrage/arrêt d'Ucarp
  |-- ucarp
  | |-- interface
  | | `-- ucarp0 --> paramètres généraux d'Ucarp
  | |-- scripts
  | | |-- 200 (correspond au vrid du cluster Ucarp pour ces services)
  | | | |-- ucmanager --> script de supervision des services/ucarp
  | | | |-- vip-down.sh --> script lancé par Ucarp au passage en mode MASTER
  | | | `-- vip-up.sh --> script lancé par Ucarp au passage en mode BACKUP
  | |-- resource.d
  | | |-- Mail --> script d'envoi de mail
  | | |-- dhcp.monitor --> script de test du service DHCP
  | | `-- dns.monitor --> script de test du service DNS (résolution de www.google.fr sur le
  | serveur olorin(10.4.0.10))
```

Pour plus d'informations, et notamment les options utilisées pour Ucarp, se reporter aux commentaires de chacun des fichiers. A noter donc qu'Ucarp est configuré ici pour exécuter le script vpi-up.sh lorsqu'il passe en mode MASTER, et vip-down.sh en mode BACKUP.

On crée ensuite un script de supervision des services: UcManager.

## 2.4 Mise en place du monitoring des services : UcManager

Prérequis: installation de l'utilitaire de clustering "Ucarp".

Algorithme du script UcManager:

- sur le serveur primaire (olorin2):

```
TEST est faux.
Si le service DHCP est DOWN 3 fois de suite (avec un intervalle de 1s entre chaque
test), alors TEST est vrai.
Si le service DNS est DOWN 3 fois de suite (avec un intervalle de 1s entre chaque
```

test), alors TEST est vrai.

Si TEST, alors

on arrête UCARP (--> *prise de service par le serveur secondaire*)  
on envoie un mail aux administrateurs  
on passe la périodicité de UcManager dans le cron à 4h

- sur le serveur secondaire (vm-olorin):

TEST est faux.

Si le service DHCP est DOWN 3 fois de suite (avec un intervalle de 1s entre chaque test), alors TEST est vrai.

Si le service DNS est DOWN 3 fois de suite (avec un intervalle de 1s entre chaque test), alors TEST est vrai.

Si TEST, alors

on envoie un mail aux administrateurs  
on passe la périodicité de UcManager dans le cron à 4h

### Observations:

- Le script exécuté lors du passage en mode MASTER (vip-up.sh) inscrit le script UcManager dans le cron avec une périodicité initiale de 1min. Au préalable, il supprime toute entrée relative à ce script (qui serait restée après un arrêt brutal de la machine par exemple).
- Le script exécuté lors du passage en mode BACKUP (vip-down.sh) désinscrit le script UcManager du cron.
- Conséquemment, via ces scripts, le script d'init d'Ucarp (/etc/init.d/ucarp) lance et arrête les services DNS et DHCP, inscrit et désinscrit le script de supervision UcManager du cron.
- Pour pallier une perte de connexion réseau de la machine principale (olorin2) qui ne pourrait donc plus envoyer de mail pour prévenir l'administrateur, le script vip-up.sh de la machine secondaire (vm-olorin) envoie un mail de prise de service.
- Nous loggons l'ensemble des événements Ucarp dans le fichier ***/var/log/ucarp.log***.

## 2.5 Installation et configuration de Rsync

Afin de synchroniser les configurations des différents serveurs, un rsync est effectuée toutes les nuits. Sur vm-olorin, la configuration :

- le fichier /etc/dhcpd.conf
- les fichiers contenus dans /etc/bind/

sont synchronisés depuis olorin. Les deux scripts effectuant ces actions sont :

- /etc/cron.daily/rsync\_dhcp
- /etc/cron.daily/rsync\_dns

La date d'exécution du cron.daily d'origine a été modifié dans le /etc/crontab, pour être repoussé dans le courant de la nuit. Afin que cela puisse se faire automatiquement, la clef publique du root de vm-olorin est autorisée sur olorin. Cette manipulation n'a pas été faite sur vm-varda, puisqu'elle n'est pour le moment pas en production, mais une opération identique sera nécessaire.

## **III/. Redondance du Serveur SAMBA/LDAP**

### **3.1 Architecture de la maquette**

Le but est d'arriver à l'architecture suivante (voir Fig.2) où les services SAMBA et LDAP sont redondés sur une machine virtuelle :

- *ucarp* se charge de créer un cluster entre les deux serveurs,
- le script de monitoring *ucmanager* se charge d'arrêter les services, de gérer *ucarp* en conséquence et de prévenir les administrateurs en cas d'arrêt d'un ou des services sur la même machine,
- DRBD se charge de synchroniser les données du serveur primaire sur le secondaire. (voir <http://wapiti.telecom-lille1.eu/commun/ens/peda/options/ST/RIO/pub/exposes/exposesrio2007/legrand-playez/fonctionnement.htm> pour plus d'informations sur la synchronisation des données).

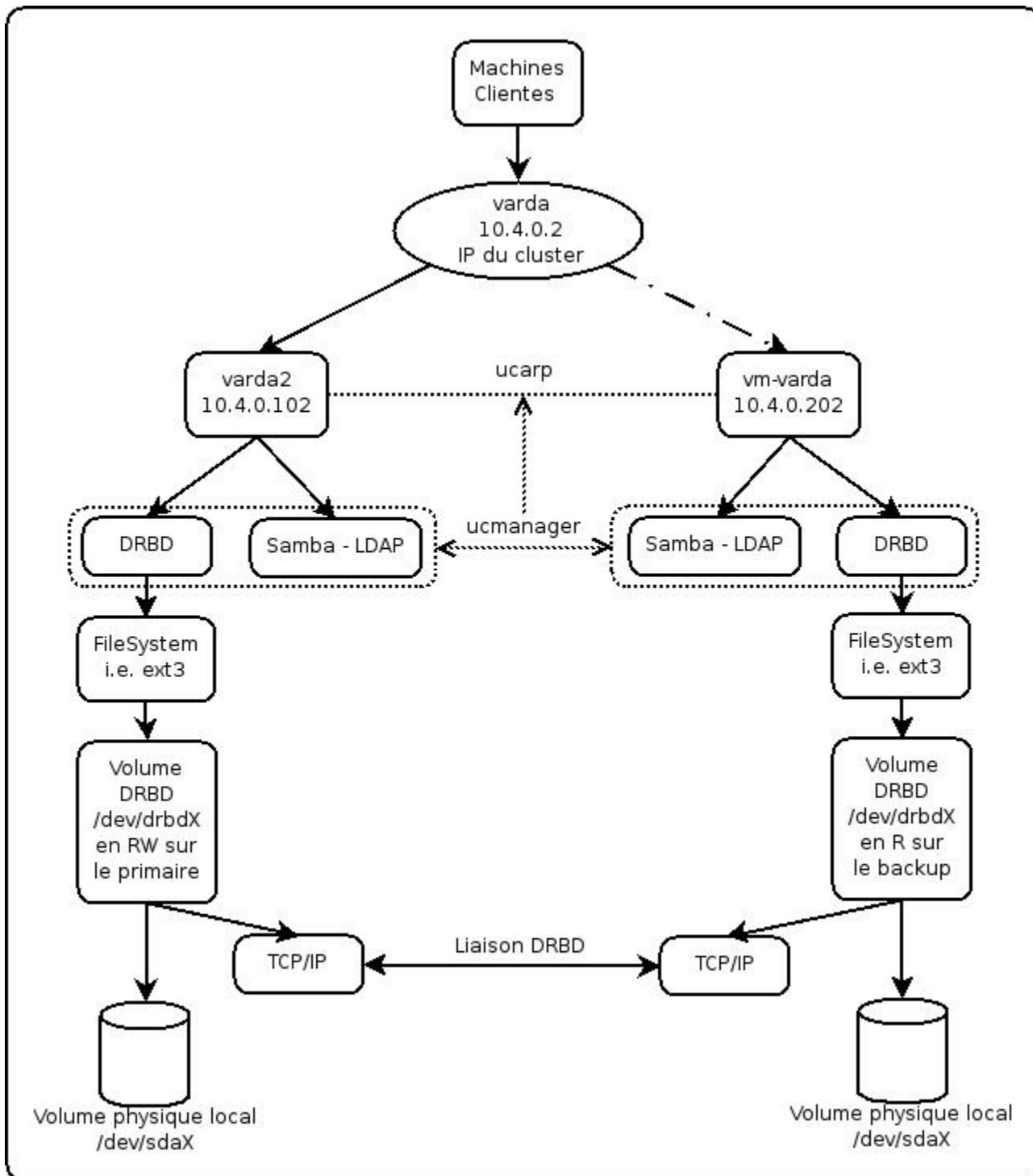


Fig.2

### 3.2 Préparation de l'hôte et de la VM

Il faut avant tout faire quelques petites modifications sur la configuration des deux machines :

- Les services DRBD (comme indiqué ci-après), SAMBA et LDAP, avec leurs fichiers de configuration respectifs, doivent être installés et doivent fonctionner (il faut les tester unitairement).

- Le package `exim` (pour le script Mail) doit être installé et configuré (voir annexe `exim4` pour l'installation et la configuration).
- Les services SAMBA et LDAP doivent être supprimés des répertoires `/etc/rcX.d`. Ils seront lancés par Ucarp qui lui-même sera lancé par le script de supervision `UcManager` (décrit plus bas) à inscrire dans le cron. Seul DRBD sera lancé au démarrage des machines.
- De manière à utiliser SSH entre les deux machines, il faut utiliser l'authentification ssh par échange de clés:
  - générer les clefs : `ssh-keygen -t rsa -b 1024` ( `"/root/.ssh/id_rsa"` et passphrase vide)
  - copier la clef publique de varda sur vm-varada : `scp ~/.ssh/id_rsa.pub root@vm-olorin:/root/`
  - ajouter la clef publique de varda dans le fichier `/root/.ssh/authorized_keys` de vm-varada : `cat ~/id_rsa.pub >> ~/.ssh/authorized_keys`
  - effectuer le même processus mais dans le sens `vm-varada > varda`

### 3.3 Installation et configuration d'Ucarp

Cette étape a lieu sur les deux machines.

```
apt-get install ucarp
```

Il faut ensuite placer Ucarp au démarrage et à l'arrêt, après avoir créé le script d'init d'Ucarp:

```
In -s /etc/init.d/ucarp /etc/rc6.d/K01ucarp
In -s /etc/init.d/ucarp /etc/rc0.d/K01ucarp
```

On crée l'arborescence suivante:

```
`-- etc
   |-- default
   |   `-- ucarp --> options de démarrage d'Ucarp
   &nbsp;|-- init.d
   |   `-- ucarp --> script de démarrage/arrêt d'Ucarp
   |-- ucarp
   |   |-- interface
   |   |   `-- ucarp0 --> paramètres généraux d'Ucarp
   |   |-- scripts
   |   |   |-- 210 (correspond au vrid du cluster Ucarp pour ces services)
   |   |   |   |-- ucmanager --> script de supervision des services/ucarp
   |   |   |   |-- vip-down.sh --> script lancé par Ucarp au passage en mode MASTER
   |   |   |   |-- vip-up.sh --> script lancé par Ucarp au passage en mode BACKUP
   |   |   `-- resource.d
   |   |       |-- Filesystem --> script montant/démontant les partitions physiques sur les
   |   |       |   partitions virtuelles (type /dev/drbdX)
   |   |       |-- Ldap --> script de démarrage/arrêt de ldap
   |   |       |-- Mail --> script d'envoi de mail
   |   |       |-- drbd.monitor --> script de test du service DRBD
   |   |       |-- drbddisk --> script passant le(s) périphérique(s) lié(e) à une ou des ressources
   |   |       |   en mode maître sur la machine courante
   |   |       `-- ldap.monitor --> script de test du service LDAP
```

Pour plus d'informations, et notamment les options utilisées pour Ucarp, se reporter aux commentaires

de chacun des fichiers. A noter donc qu'Ucarp est configuré ici pour exécuter le script vip-up.sh lorsqu'il passe en mode MASTER, et vip-down.sh en mode BACKUP.

### 3.4 Mise en place du monitoring des services : UcManager

Prérequis: installation de l'utilitaire de clustering "Ucarp".

Ici, tous les scripts sont identiques sur les serveurs primaire (varda) et secondaire (vm-varda).

Le script exécuté lors du passage en mode MASTER (vip-up.sh)

```
on configure une nouvelle interface réseau avec l'IP du cluster
on passe en mode primaire les peripheriques DRBD
on monte les partitions
on démarre le service ldap
on démarre le service samba
on envoie un mail pour prévenir du passage en MASTER sur le cluster
```

Le script exécuté lors du passage en mode BACKUP (vip-down.sh)

```
on arrete le service samba
on arrete le service ldap
on passe en mode secondaire les peripheriques DRBD
on demonte les partitions
on supprime la nouvelle interface réseau avec l'IP du cluster
```

Algorithme du script UcManager:

```
Si DRBD est à l'état STANDALONE et si la machine est VARDA, alors
|   on arrête UCARP
|   on arrête DRBD
|   on arrête UCARP sur la machine vm-varda en ssh
|   on redémarre DRBD sur la machine vm-varda en ssh
|   on démarre DRBD
|   on démarre UCARP
|   on envoie un mail de détection de Split-Brain à l'admin
Fin Si

Si DRBD est à l'état STANDALONE et si la machine est VM-VARDA, alors
|   on arrête UCARP (et donc SAMBA et LDAP)
|   on redémarre DRBD
|   on démarre UCARP
Fin Si

Si UCARP n'est pas démarré, alors
|   si DRBD est démarré à l'état UPTODATE (état 0), alors on démarre UCARP
|   si DRBD est HS (état 3), on tente de redémarrer DRBD
|   on ne fait rien dans tous les autres cas
Sinon
    si DRBD est HS (état 3), alors
    |   on arrête UCARP
    |   on tente de redémarrer DRBD
    |   on envoie un mail à l'administrateur DRBD HS
```

```

sinon
|
| si SAMBA est HS, alors on arrête UCARP
|   sinon
|     | si LDAP est HS, alors on arrête UCARP
|     |   fin si
|     fin si
|   fin si
Fin Si

```

*Remarque: tout arrêt/démarrage d'UCARP signifie également un arrêt/démarrage de SAMBA et LDAP.*

#### Observations:

- UcManager nécessite d'être inscrit dans le cron de chaque machine pour qu'il puisse périodiquement tester le service DRBD et prendre toutes les mesures adéquates afin d'assurer au mieux une continuité de service aux utilisateurs. Pour ce faire, il nous faut éditer la table du cron:

```
crontab -e
```

puis ajouter notre script:

```
*/1 * * * * /etc/ucarp/scripts/210/ucmanager
```

qui s'exécutera dès lors toutes les minutes.

- Afin d'empêcher tout basculement maître/esclave lors d'une synchronisation qui pourrait causer une incohérence des données, notre script UcManager ne démarre Ucarp que si DRBD est à l'état UpToDate, i.e. lorsque toute synchronisation est donc terminée. Ceci explique pourquoi Ucarp n'est pas lancé au démarrage de la machine mais par notre script qui aura préalablement testé DRBD.

Cependant, si la machine primaire se retrouve isolée du réseau, l'autre machine va prendre la main, mais Ucarp va continuer à fonctionner sur notre serveur isolé: ainsi, lorsque le lien sera rétabli, celui-ci va reprendre immédiatement son statut de Primaire même si une synchronisation est en cours. Nous devons donc arrêter Ucarp sur une machine qui se retrouve dans une telle situation. Pour ce faire, nous utilisons Ifplugd.

## 3.5 Installation et configuration de Ifplugd 0.28-2.3

Nous l'installons à partir des paquets Debian:

```
apt-get install ifplugd
```

Nous créons les liens symboliques pour le démarrage et l'arrêt du service avec la machine:

```
ln -s /etc/init.d/ifplugd /etc/rc2.d/S99ifplugd
ln -s /etc/init.d/ifplugd /etc/rc6.d/K01ifplugd
ln -s /etc/init.d/ifplugd /etc/rc0.d/K01ifplugd
```

Nous définissons les paramètres de démarrage de Ifplugd dans le fichier ***/etc/default/ifplugd***. On y définit notamment l'interface scrutée (-i), le temps d'attente avant exécution du script (-u et -d) et le script qui sera exécuté (-r).

Nous créons enfin le script qui sera lancé par Ifplugd lors d'un changement d'état de l'interface scrutée: ***/etc/ifplugd/net\_supervisor***.

## 3.6 Installation et configuration de DRBD 8.2.4

Cette étape a lieu sur les deux machines.

Téléchargement du package dans le répertoire `/usr/src`: il faut impérativement le placer à cet endroit car il contiendra les sources du module DRBD qu'il faudra intégrer au kernel courant:

```
cd /usr/src
wget http://oss.linbit.com/drbd/8.2/drbd-8.2.4.tar.gz
tar zxvf drbd-8.2.4.tar.gz
```

Ensuite téléchargeons les headers du kernel pour l'intégration du module, ainsi que quelques packages utiles à la compilation:

```
apt-get install linux-headers-`uname -r` build-essential gcc-3.4 flex
```

Une fois le tout installé, créons le module

```
cd /usr/src/drbd-8.2.4
make module
```

Faisons de même pour les binaires de DRBD qui vont nous servir à son administration:

```
cd /usr/src/drbd-8.2.4/
make tools
make all install
```

Ceci aura pour effet de compiler le module et de l'installer, on peut maintenant l'utiliser:

```
modprobe drbd
```

DRBD est installé.

Voici ensuite le fichier de configuration de DRBD (***/etc/drbd.conf***) à utiliser dans le cas de l'IUT, il doit être identique sur les deux machines.

**Remarque** : dans ce fichier se trouve le secret partagé entre les deux noeuds DRBD

```
# Fichier de configuration de DRBD
# Celui-ci doit etre strictement identique sur les machines du cluster
#
# parametre global
global { usage-count yes; }

# parametres commun a toutes les ressources
```

```

common {
    # on place le débit a 100Mb/s
    syncer {      rate 100M; }

    # utilisation du protocole le plus sécurisé
    protocol C;

    # temps d'attente de l'autre noeud au démarrage
    startup {    wfc-timeout 10; }

    net {
        # SPLIT BRAIN : primaire : pour la synchronisation apres split-brain, le noeud
        # ayant le plus de blocs
        # modifies est considere UpToDate
        after-sb-0pri discard-least-changes;

        #SPLIT BRAIN : secondaire : on prend la même décision que le primaire
        after-sb-2pri violently-as0p;

        # authentification sha1 entre les deux noeuds
        cram-hmac-alg sha1;

        # secret partage
        shared-secret "tpri";
    }
}

# Declaration des ressources prises en compte par DRBD
# Une ressource par partition
# /iut/admin
resource "admin" {
    on varda2 {
        device /dev/drbd0;
        disk /dev/mapper/vserver-Vadmin;
        address 10.4.0.102:7789;
        meta-disk internal;
    }
    on vm-varda {
        device /dev/drbd0;
        disk /dev/hdb5;
        address 10.4.0.202:7789;
        meta-disk internal;
    }
}

# /iut/yext
resource "yext" {
    on varda2 {
        device /dev/drbd1;
        disk /dev/mapper/vserver-Vext;
        address 10.4.0.102:7790;
        meta-disk internal;
    }
    on vm-varda {
        device /dev/drbd1;
        disk /dev/hdb6;
        address 10.4.0.202:7790;
    }
}

```

```

        meta-disk internal;
    }
}

# /iut/profiles
resource "profiles" {
    on varda2 {
        device /dev/drbd2;
        disk /dev/mapper/vserver-Vprofiles;
        address 10.4.0.102:7791;
        meta-disk internal;
    }
    on vm-vara {
        device /dev/drbd2;
        disk /dev/hdb7;
        address 10.4.0.202:7791;
        meta-disk internal;
    }
}

# /iut/yprofs
resource "yprofs" {
    on varda2 {
        device /dev/drbd3;
        disk /dev/mapper/vserver-Vprofs;
        address 10.4.0.102:7792;
        meta-disk internal;
    }
    on vm-vara {
        device /dev/drbd3;
        disk /dev/hdb8;
        address 10.4.0.202:7792;
        meta-disk internal;
    }
}

# /iut/recup
resource "recup" {
    on varda2 {
        device /dev/drbd4;
        disk /dev/mapper/vserver-Vrecup;
        address 10.4.0.102:7793;
        meta-disk internal;
    }
    on vm-vara {
        &nbsp; device /dev/drbd4;
        disk /dev/hdb9;
        address 10.4.0.202:7793;
        meta-disk internal;
    }
}

# /iut/yusr
resource "yusr" {
    on varda2 {
        device /dev/drbd5;
        disk /dev/mapper/vserver-Vusr;
        address 10.4.0.102:7794;
    }
}

```

```

        meta-disk internal;
    }
    on vm-vara {
        device /dev/drbd5;
        disk /dev/hdb10;
        address 10.4.0.202:7794;
        meta-disk internal;
    }
}

```

Maintenant, pour lancer DRBD au démarrage, il reste une chose à faire: modifier le script `/etc/init.d/drbd` pour qu'il n'y ait pas de problème au lancement. Il peut arriver que l'interface réseau et le commutateur mettent du temps à communiquer, auquel cas l'interface réseau met plus de temps que prévu à s'attribuer une adresse IP. Il faut donc bloquer le démarrage de DRBD pour qu'il attende cette attribution: voici les modifications à apporter au script d'init de DRBD:

Tout d'abord, il faut récupérer l'IP locale qu'utilise DRBD pour communiquer avec l'autre noeud, étant donné qu'il est possible que DRBD ait un lien dédié pour transférer les données d'un noeud à l'autre. Il faut donc récupérer cette adresse dans le fichier de configuration de DRBD. Ajoutons dans le script `/etc/init.d/drbd` la variable :

```

ADD_MOD_PARAM=""

sed_expr1="/`uname -n`[[ :space:]]*{/,/[[:space:]]}/p"
IP=""`sed -n -e "$sed_expr1" /etc/drbd.conf | grep address | cut -d':' -f1 | awk '{ print $2 }'
| uniq ` "
[ -z "$IP" ] && IP="localhost"

```

**LIGNE EXISTANTE**

**NOUVELLE LIGNE**

**Remarques :**

`sed_expr1` : récupération des blocs `<hostname> { ..... }`

Ensuite, il faut attendre au maximum 60 secondes, dans le cas d'un start de DRBD, qu'il y ait une interface configurée avec cette adresse:

```

start)
    echo "Starting DRBD resources:"

    echo -n "Waiting for ip address allocation (max 60sec): "
    if=""`/sbin/ifconfig | grep $IP` "
    i=0
    while [ -z "$if" ] && [ "$i" != "60" ]
    do
        echo -n .
        sleep 1
        if=""`/sbin/ifconfig | grep $IP` "
        i=`expr $i + 1`
    done
    echo "IP $IP Ok !"

    assure_module_is_loaded

```

## LIGNE EXISTANTE

## NOUVELLE LIGNE

### Remarques :

Ce bout de script permet de vérifier si "ifconfig" nous rend une ligne avec cette adresse, ceci avec une attente d'une seconde par vérification.

Mettre ce script au démarrage (par défaut, Debian démarre en init 2):

```
In -s /etc/init.d/drbd /etc/rc2.d/S98drbd
In -s /etc/init.d/drbd /etc/rc6.d/K02drbd
In -s /etc/init.d/drbd /etc/rc0.d/K02drbd
```

Au premier lancement de DRBD, il faut:

- gérer l'authentification ssh entre le primaire et le secondaire par échange de clefs:

#### **sur le primaire**

- Générer le couple de clef privée/publique :

1. `varda# ssh-keygen -t rsa -b 1024`  
- fichier à utiliser pour sauver les clefs : `/root/.ssh/id_rsa`  
- mettre une pass-phrase vide pour la clef privée afin d'éviter les demandes de mot de passe
2. Mettre la clef publique dans le fichier `~/.ssh/authorized_key`  
`varda# scp ~/.ssh/id_rsa.pub root@balrog`  
`vm-varada# cat id_rsa.pub >> ~/.ssh/authorized_keys`
3. il faut maintenant se logger une fois en ssh du primaire sur le secondaire pour récupérer le fingerprint de la clef publique du secondaire:  
`varda:~/.ssh# ssh balrog`  
The authenticity of host varda (10.4.0.102)' can't be established.  
RSA key fingerprint is a6:cc:b8:64:0a:ce:ae:33:3c:0b:42:c5:c0:1d:95:76.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added 'vm-varada,10.4.0.202' (RSA) to the list of known hosts.  
Last login: Wed Feb 20 16:20:43 2008 from varda  
Linux vm-varada 2.6.23 #5 SMP Tue Feb 19 18:42:46 CET 2008 i686  
vm-varada:~#

- OPTIONNEL : préparer les disques réels (créer les partitions, les points de montage et les systèmes de fichiers) si ce n'est pas le cas.
- Créer les meta-datas qui serviront à la synchronisation des deux disques; il faut lancer la commande suivante sur les deux noeuds du cluster:

```
drbdadm create-md <nom de la ressource>
```

- Sur la machine principale uniquement, il faut dire à DRBD que ses ressources sont UpToDate afin que Ucarp puisse passer ces ressources en "Primary". Pour ce faire, on utilise la commande sur chacun des périphériques virtuels de DRBD:

```
drbdsetup /dev/drbdX primary -o
```

- Il faut enfin placer le script de supervision des services UcManager dans le cron, avec la périodicité désirée:

Ce script, décrit au point 2.4, va se charger périodiquement de tester les services, et notamment de démarrer/arrêter DRBD en mode Primary et de monter les partitions en conséquence.

### Remarques :

Lors de cette synchronisation, les données du secondaire sont alignées sur celles du primaire nouvellement déclaré, i.e. les données de ce secondaire sont perdues. Au contraire, les données sur les partitions du primaire sont conservées en l'état.

## 3.5 Administration

### Fichiers de log

Il y a plusieurs manières de voir l'état de DRBD:

- le fichier de status du module drbd : **watch -n 1 /cat/proc/drbd** (on peut aussi lancer le script /root/test.drbd)  
exemple lors d'une synchronisation:

```
cs:SyncSource st:Primary/Secondary ds:UpToDate/Inconsistent C r---
ns:3570248 nr:0 dw:0 dr:3570944 al:0 bm:217 lo:0 pe:6 ua:22 ap:0
[=====>..] sync'ed: 91.6% (333524/3903604)K
finish: 0:00:30 speed: 10,952 (10,256) K/sec
resync: used:1/31 hits:222939 misses:218 starving:0 dirty:0 changed:218
act_log: used:0/127 hits:0 misses:0 starving:0 dirty:0 changed:0
```

- Sinon, les informations se trouvent dans /var/log/messages

### Etats de DRBD

Champ cs: (Connection state)

- **Unconfigured** : Device waits for configuration.
- **StandAlone** : Not trying to connect to peer, IO requests are only passed on locally.
- **Unconnected** : Transitory state, while bind() blocks.
- **WFConnection** : Device waits for configuration of other side.
- **SyncingAll** : All blocks of the primary node are being copied to the secondary node.

Champ st: (local/remote State)

- **Primary** : the active node; may access the device.
- **Secondary** : the passive node; **must not access the device**; expects mirrored writes from the other node.
- **Unconfigured** : this is not a role, obviously.

Champ ds: (Data State)

- **Diskless** : No storage attached, or storage had IO errors previously and got detached.
- **Attaching** : in the process of attaching the local storage.
- **Failed** : storage had io errors.
- **Inconsistent** : storage is Inconsistent (e.g. half way during bitmap based resync)
- **Outdated** : storage is consistent, but not UpToDate
- **Consistant** : storage is consistent, not yet decided whether it is UpToDate or Outdated

- **UpToDate** : storage is good

Pour voir la totalité des état il faut se rferencer à la page suivante : <http://www.linux-ha.org/DRBD/FAQ#head-988086376cbd00bdcc9c9d199317af5f7550c5c1>

## Changement du hostname d'une des deux machines

En cas de changement de hostname sur un des deux noeud DRBD il faut recompiler le module noyau de DRBD car il contient en "dur" le hostname local.

Voici les manipulations à effectuer :

```
cd /usr/src/drbd-8.2.4
modprobe -r drbd
make uninstall
make module
make tools
make install all
modprobe drbd
```

## Procédure manuelle en cas de Split brain:

### *Split brain* :

Etat de DRBD où les deux noeuds sont primaires sur les ressources DRBD et les deux ont accès en écriture sur la partition partagée: la connexion DRBD entre les deux est à l'état "StandAlone".

**Remarque:** cette procédure est effectuée par le script ucmanager mais en cas de problème il se peut qu'il faille le faire à la main.

### Cause:

les deux machines ont été déconnectées du réseau ou arrêtées/redémarrées dans un ordre spécifique. Le primaire ET le secondaire se sont arrêté avec DRBD en mode primaire.

Ce cas peut se présenter lors de l'arrêt de tous les serveurs, par exemple on arrete le primaire puis le secondaire 30 secondes après (le temps que le secondaire détecte que le primaire se soit arrêté et qu'il passe primaire). Et que l'on redemarre le primaire avant le secondaire.

Voici l'état que l'on retrouve sur la maquette après avoir redémarré:

- sur le serveur virtualisé:

```
vm-vara:# cat /proc/drbd
version: 8.2.4 (api:88/proto:86-88)
GIT-hash: fc00c6e00a1b6039bfcebe37afa3e7e28dbd92fa build by root@vm-vara,
2008-02-06 15:07:24
0: cs:StandAlone st:Primary/Unknown ds:UpToDate/DUnknown  r---
   ns:0 nr:0 dw:4 dr:21 al:1 bm:0 lo:0 pe:0 ua:0 ap:0
   resync: used:0/31 hits:0 misses:0 starving:0 dirty:0 changed:0
   act_log: used:0/127 hits:0 misses:1 starving:0 dirty:0 changed:1
```

- sur le vrai serveur:

```
varda2:~# cat /proc/drbd
version: 8.2.4 (api:88/proto:86-88)
GIT-hash: fc00c6e00a1b6039bfcebe37afa3e7e28dbd92fa build by root@varda2, 2008-02-06
16:05:20
0: cs:StandAlone st:Secondary/Unknown ds:UpToDate/DUnknown r---
   ns:0 nr:0 dw:8 dr:21 al:0 bm:35 lo:0 pe:0 ua:0 ap:0
   resync: used:0/31 hits:0 misses:0 starving:0 dirty:0 changed:0
   act_log: used:0/127 hits:2 misses:0 starving:0 dirty:0 changed:0
```

Commandes à faire pour relancer le service DRBD sur les deux serveurs DANS CET ORDRE:

```
# Arrêter les services sur varda2
varda2:~# /etc/init.d/ucarp stop
varda2:~# /etc/init.d/drbd stop
```

```
# Redemarrer les services sur vm-varada (effectué par ssh dans ucmanager)
vm-varada:~# /etc/init.d/ucarp stop
vm-varada:~# /etc/init.d/drbd restart
vm-varada:~# /etc/init.d/ucarp start
```

```
# Redemarrer les services sur varda2
varda2:~# /etc/init.d/ucarp stop
varda2:~# /etc/init.d/drbd stop
```

## **IV/. Virtualisation**

### **4.1 Solutions disponibles**

Les principales solutions disponibles pour la virtualisations sous linux sont:

- linux vserver (impossible de faire du \*BSD)
- openVZ (impossible de faire du \*BSD également)
- vmware (architecture lourde, pas libre et pas gratuit en environnement professionnel)
- xen (architecture légère, outils d'administration faciles et bien faits, compatibilité plus limitée du fait de son architecture)
- kvm : qemu avec fonctions hv cpu (architecture entre vmware et xen, outils d'administration plus limités, intégré au noyau linux depuis la v2.6.20)

### **4.2 Choix retenus**

Logiquement nous avons retenus *xen* et *kvm* avec une préférence pour *xen* à priori pour sa légèreté au niveau du système hôte (dom0) et sa réputation pour ce qui est des performances. Xen

### **4.3 Xen**

## **a) Test de Xen 3.0.3 fournit par la Debian stable**

Dans un premier temps nous avons d'abord testé les solutions offertes par le système de paquets de la Debian stable. Elle fournit un Xen version 3.0.3, pour les tools, l'hyperviseur un noyau 2.6.18 patché pour Xen. L'installation est relativement simple, et ce passe sans problème. Le problème se situe au niveau de la version de Xen, qui est la 3.0.3, alors que la version actuelle est la 3.2. De ce fait, l'installation d'un Linux dans le Xen de la machine hôte ne pose aucun problème, mais il nous a été par contre impossible d'installer une quelconque BSD. Lors de son lancement Xen ne reconnaissait tout simplement pas le noyau DomU, en répondant par un "invalid argument".

## **b) Test du noyau Xen 2.6.24 de la Sid**

Les noyaux BSD ne fonctionnant pas, nous avons testé le noyau Xen et l'hyperviseur fournis par la Sid, mais avec les tools 3.0.3 de la stable. Le noyau n'a pas réussi à démarrer. Il ne nous a pas été possible de tester avec le noyau de testing, car elle ne propose pas de noyau Xen, elle ne fournit actuellement que les tools en version 3.1.

## **c) Test du noyau Xen 2.6.24 et des tools de la Sid**

A la suite de cela, nous avons installé les tools de la Sid pour que la version soit en adéquation avec la version du noyau. Nous avons donc les tools, l'hyperviseur et le noyau Xen de la Sid, tout en ayant le reste de la distribution en stable. Le système à cette fois réussi à démarrer correctement. Comme avec la version 3.0.3, nous avons réussi à démarrer un linux dans Xen, mais toujours pas de BSD. On voit que cette fois le noyau domU BSD est mieux reconnu par Xen, le message d'erreur n'étant plus un simple "invalid argument". Apparemment le noyau domU ne serait pas compilé pour la version de Xen que nous utilisons.

## **d) Test du noyau de la Debian stable (2.6.18-xen) avec les tools de la testing (3.1)**

Nous avons également testé les tools fournis par testing avec le noyau patché pour xen de la stable, mais le résultat fut le même qu'avec les tools de la stable, Xen nous a répondu avec un "Invalid argument" lors du démarrage d'un noyau BSD.

## **e) Recherche de noyau DomU BSD**

Le problème venant apparemment des noyaux domU que nous utilisons, nous en avons recherché et testé d'autres. Le souci est qu'il en existe très peu de disponibles. Les domU ne sont pas forcément fournis par la distribution elle-même, et se trouvent disséminés sur différents sites. Résultat aucun noyau ne fonctionne et pas moyen de savoir pour quelle version de Xen ils sont prévus. Une solution serait de compiler soit même le noyau BSD patché pour Xen, mais nous ne sommes pas lancés dans cette procédure.

## f) Installation de Xen 3.2 depuis les sources

Nos précédents essais n'ayant pas permis de démarrer une FreeBSD nous avons décidé de tester la toute dernière release disponible sur xen.org. Le package fournit les sources pour compiler un noyau et les tools de la dernière version. C'est un noyau 2.6.18 patché qui est récupéré via mercurial (un système de gestion de version style svn). Il faut procéder de cette façon car depuis la version 3.1, xen ne fournit plus directement les patch applicables à un noyau personnalisé.

Procédure d'installation:

Il y a visiblement (sous debian du moins) un problème avec la procédure d'installation de xen. Il n'arrive pas à récupérer tout seul les sources du noyau avec mercurial, il faut donc préparer le terrain.

```
% mkdir build
% wget http://bits.xen-source.com/oss-xen/release/3.2.0/xen-3.2.0.tar.gz
% tar zxvf xen-3.2.0.tar.gz
% # xen-3.2.0/ et linux-2.6.18-xen.hg/ doivent être dans le même répertoire, au même
niveau.
% hg clone http://xenbits.xen-source.com/linux-2.6.18-xen.hg
% cd xen-3.2.0
% make KERNELS="linux-2.6-xen0 linux-2.6-xenU" linux-2.6-xen-config
CONFIGMODE=menuconfig% cp /boot/config-2.6.18-my build-linux-2.6.18-xen_x86_32/
.config
% make linux-2.6-xen-build
% make linux-2.6-xen-install
% make install
```

A partir de ce moment, les xen-tools sont installés et les images des noyaux xen, dom0 et domU sont présents dans /boot. Il suffit alors de générer les initrd pour chaque noyau, mettre à jour grub et rebooter sur le nouveau dom0.

Malheureusement le résultat est toujours le même malgré l'utilisation de la dernière version. Impossible de démarrer les noyaux FreeBSD pour xen que nous avons trouvés.

L'ultime solution était de tester le support hvm (fonctions de virtualisation intégrées au cpu) pour installer depuis un iso les os (Linux et plus particulièrement FreeBSD) contrairement au fonctionnement classique de xen.

## g) Test de Xen 3.2 avec HVM

FreeBSD posant problème dans tous nos précédents tests de xen, c'est donc le premier test que nous avons effectué. Il n'a pas été concluant, le boot de l'installation s'interrompt avec l'erreur :

```
int=0000000d err=00000000 efl=00010046 eip=000090db
eax=00050033 ebx=00002820 ecx=00000000 edx=0000a000
esi=00009701 edi=00051f98 ebp=00007bea esp=00001800
cs=0008 ds=0000 es=0000 fs=0000 gs=0000 ss=0010
cs:eip=0f 01 15 d0 96 00 00 66-ea e8 90 18 00 b1 20 8e
d1 8e d9 8e c1 8e e1 8e-e9 48 0f 22 c0 ea fd 90
ss:esp=0a 69 6e 74 3d 30 30 30-30 30 30 30 64 20 20 65
72 72 3d 30 30 30 30 30-30 30 30 20 20 65 66 6c
BTX halted
```

Les expérimentations se sont arrêtées là puisqu'après quelques recherches sur internet, nous avons constaté que de nombreux messages décrivent cette erreur sur la mailing list xen-sources et

visiblement c'est systématique avec FreeBSD lorsqu'on active les options HVM dans xen.

## 4.4 KVM

### a) Installation de KVM

Les tests avec Xen pour faire fonctionner une BSD s'étant avéré infructueux, nous avons décidé de tester KVM, un système basé sur QEMU, mais utilisant les instructions HVM des nouveaux processeurs, le rendant ainsi beaucoup plus rapide. L'installation est relativement simple, bien que les tools ne soit pas fournis avec la Debian stable. Nous avons donc téléchargé la dernière version (kvm-60), compilé et installé.

Afin de rester avec un système propre au maximum, nous avons créé un paquet Debian via checkinstall. Le paquet intègre les tools mais aussi les deux modules (kvm et kvm\_intel) permettant d'utiliser KVM. Les tests furent concluants que ce soit pour des systèmes Linux ou BSD. Voici donc les détails de cette solution, qui est celle que nous avons finalement retenue.

### b) Installation des VM

Pour installer une nouvelle VM, il faut commencer par lui créer son fichier image, nommé .qcow pour qemu. Pour cela on utilise la commande suivante :

```
qemu-img create -f qcow2 win.qcow 8G
```

Cela aura pour effet d'allouer les 8Go directement sur le disque, que l'espace soit utilisé par la VM ou non. Il est également possible de ne pas allouer tout l'espace dès le départ via cette commande :

```
qemu-img create disk.cow 8G
```

La différence est que si l'on utilise la seconde solution il n'est a priori pas possible de monter l'image via mount, alors qu'avec la première cela reste possible via :

```
mount -o loop,offset=32256 image mountpoint
```

Les VM se lancent et s'installent ensuite par la commande `qemu-system-x86_64`. Pour effectuer une installation depuis un fichier iso, il suffit de préciser à sur quoi l'on veut démarrer, et spécifier l'image que l'on souhaite utiliser pour l'installation :

```
qemu-system-x86_64 -hda disk.cow -cdrom debian-40r1-i386-businesscard.iso -boot d -m 512
```

On voit que le fichier image "disk" sera utilisé comme hda dans la VM et que l'on précise l'iso à utiliser pour le cdrom. L'option "d" de "boot" permet de préciser que l'on souhaite démarrer sur le cdrom, ce qui n'est pas le cas par défaut. L'option "m" permet quant à elle de spécifier la quantité de RAM que l'on souhaite allouée.

On notera au passage que cette quantité de RAM est directement consommée au lancement de la VM, et qu'il n'est pas possible de la changer une fois démarrée, contrairement à Xen qui le permettait.

## c) Gestion des VM

### \* **Introduction**

Un des défauts majeur de KVM, certainement du au fait qu'il soit encore jeune, est qu'il ne possède pas d'outil permettant de gérer un ensemble de VM tel que le permettait la commande "xm" de Xen. De ce fait nous en avons créé un simpliste, permettant de démarrer, arrêter et lister les VM. Il s'agit d'un script shell au même titre que ceux déjà présent dans le init.d. Ce script est d'ailleurs lancé au démarrage pour lancer automatiquement certaines VM.

### \* **Fonctionnement**

Le script, /etc/init.d/kvm/, utilise le répertoire de configuration /etc/kvm dans lequel se trouve un répertoire pour la configuration de chaque VM. Ainsi la configuration de la VM vm-olorin se trouve dans /etc/kvm/vm-olorin. Dans chacun de ces répertoires se situe quatre fichiers :

- start : appelé lors du démarrage de la VM
- stop : appelé lors de l'arrêt de la VM
- launcher.sh : appelé par start (permet de démarrer la VM)
- config : utilisé par les trois scripts précédant et contenant la configuration de la VM

Le fichier de configuration contient notamment, l'image à utiliser, la quantité de RAM allouée, ou encore si il faut démarrer la VM au lancement de la machine via le paramètre "autostart". Pour le détails du contenu de la configuration des VM, se reporter au fichier de configuration eux-mêmes, qui sont commentés.

Ces scripts sont utilisés par l'init script /etc/init.d/kvm pour démarrer, arrêter et voir l'état des VM. Les commandes s'utilisent de la façon suivant pour gérer la machine vm-olorin par exemple:

- démarrage : /etc/init.d/kvm start vm-olorin
- arrêt : /etc/init.d/kvm stop vm-olorin
- état : /etc/init.d/kvm status

L'affichage de l'état retourne des informations comme ça:

```
root@valinor /etc/kvm/sambatest# /etc/init.d/kvm status
Liste des vm demarrees :
- vm-olorin.qcow (pid 2850) :1
```

Dans cet exemple, le :1 correspond au numéro de display à utiliser pour se connecter à la VM avec VNC.

Les fonctions *start* et *stop* peuvent être exécutées sans préciser un nom de VM. Dans ce cas, *start* démarrera toutes les VM configurées en "autostart" et *stop* arrêtera toutes les VM démarrées.

La fonction *stop* est un peu particulière, elle corrige un comportement parfois capricieux de KVM qui parfois, ne s'arrête pas totalement alors que la VM est bien arrêtée. Pour corriger ce problème la fonction *stop* tue le process de la VM après un certain temps. Ce délais est de 30 secondes par défaut et est paramétrable au début du fichier */etc/init.d/kvm*.

## \* Accès aux VM

Une fois les VM lancées, il est nécessaire de pouvoir y accéder quelque soit la situation. La plupart du temps cela se fera en SSH, mais en cas de problème un accès VNC est possible. En effet une option est passé au lancement de chaque VM de façon à ce qu'elles écoutent sur un display différents. Cet accès est indépendant de l'état réseau de la VM car il se fait directement sur l'hôte, on y accède de la sorte pour la VM étant sur le display 2 :

```
vncviewer valinor:2
```

Via ce procédé il est également possible d'accéder au grub des VM.

## d) Moniteur

Qemu fournit pour chaque VM un moniteur permettant de suspendre leur exécution, les stopper, ou enregistrer leur état. Le problème est que ce moniteur n'est pas accessible facilement, et ne peut être redirigé que vers des TTY ou vers */dev/pts/x*. Si l'on souhaite pouvoir accéder à cette console, il est nécessaire d'utiliser le */dev/pts/x* utiliser par le terminal courant. Ceci est fournit par la commande "tty".

Afin de pouvoir tout de même accéder à cette console, *screen* est utilisé. Ainsi les moniteurs sont accessibles depuis des screens. Pour cela, le script "start" exécute dans un *screen* le script *launcher.sh*. Le script "launcher.sh" utilise alors la configuration (*config*) pour lancer la commande "qemu-system-x86\_64" elle même dans un *screen*, car en plus de lancer la VM, cette commande donnera accès à la console KVM de la VM.

On a donc un *screen* par VM exécutée qui porte le nom du fichier image de la VM, et qui permet d'accéder à la console QEMU de la VM. Attention, l'arrêt de ce *screen* signifie l'arrêt de la VM.

On peut lister les moniteurs accessibles par la commande */etc/init.d/kvm monitors* et accéder à un moniteur en faisant un *screen -r nom\_vm.qcow*. Attention a bien détacher le *screen* du moniteur pour en sortir pour ne pas stopper la VM associée.

En cas d'arrêt d'une VM sans utiliser les scripts (un *halt* directement sur la VM) le *screen* du moniteur reste présent. Il sera terminé lors de redémarrage par le script */etc/init.d/kvm*. Il est donc préférable de stopper les VM via */etc/init.d/kvm stop nom\_vm*, plutôt que par un "halt".

## e) Réseau

Chaque interface réseau physique d'une VM est associée à une interface TAP sur l'hôte (valinor). Si on veut que cette interface TAP puisse accéder au reste du réseau et pas juste à l'hôte, il faut créer un

bridge entre le tap et une interface ethernet physique de l'hôte.  
Concrètement cela se résume par :

```
brctl addbr br0  
tunctl -t tap0  
brctl addif br0 tap0  
ifconfig tap0 up
```

Cela dit l'ensemble de la configuration des bridges et des TAP se trouve dans le /etc/network/interfaces.  
On peut voir la configuration courante via un :

```
brctl show
```

## f) Qemuctl

Malgré l'absence de script fournit pour contrôler les VM, nous avons trouver durant nos recherches un outil nommé qemuctl. Celui-ci ne permet cela de ne gérer qu'une VM via une interface graphique. Cet outil mérite cela dit d'être souligné, même si nous ne l'utilisons pas, car il permet d'accéder aux fonctionnalités principales de QEMU.

## g) Le cas des \*BSD

L'intérêt majeur de KVM est qu'il permet de virtualiser des OS BSD, et notamment FreeBSD utilisé comme routeur de l'IUT. Il y a donc une FreeBSD 6.3 d'installée mais non prête à être utilisé pour le moment comme backup, car sa configuration n'est pour le moment pas la même que celle du véritable routeur. Avant de pouvoir s'en servir, il faut donc modifier sa configuration, à commencer par ses IP qui ne sont pas les bonnes actuellement. Pour cela, on peut choisir de monter l'image qcow.

```
mount -r -t ufs -o ufstype=ufs2,loop,offset=32256 /home/vm/vm-bombadil2.qcow /mnt
```

Cela a pour effet de ne monter que le premier slice, mais permet d'accéder au rc.conf. Cette VM est celle qui servait de backup lors des tests, ce que l'on peut remarquer dans la configuration de carp.

### Configuration de carp

dans le rc.conf :

```
cloned_interfaces="carp0 carp1"  
ifconfig_carp0="vhid 1 advskew 100 pass toto 10.7.1.123"  
ifconfig_carp1="vhid 2 advskew 100 pass titi 10.7.0.123"
```

ou directement via ifconfig :

```
ifconfig carp0 vhid 1 advskew 100 pass toto 10.7.1.123  
ifconfig carp1 vhid 2 advskew 100 pass titi 10.7.0.123
```

Ceci a pour effet d'activer deux interfaces carp, une sur chaque interface réseaux de la VM, chacune étant dans un vhid différent. Dans l'exemple deux IP virtuelles sont donc créées 10.7.1.123 et 10.7.0.123 utilisées comme IP publique et comme IP du routeur par défaut.

Afin que le master reprenne la main lorsqu'il repasse UP, il est nécessaire d'activer la preemption. Pour cela il faut ajouter dans /etc/sysctl.conf :

```
net.inet.carp.preempt=1
```

ou directement

```
sysctl -w net.inet.carp.preempt=1
```

### Configuration de pfsync

dans le rc.conf :

```
pfsync_enable="YES"  
pfsync_syncdev="re1"  
pfsync_syncpeer="10.7.0.104"
```

ou directement via ifconfig :

```
ifconfig pfsync0 syncdev re1 syncpeer 10.7.0.104  
ifconfig pfsync0 up
```

De cette façon la VM synchronise sa table des connexions avec le peer 10.7.0.104. Par défaut les tables sont échangées en multicast, mais pour plus de sécurité il est possible d'utiliser de l'unicast en précisant le peer avec lequel se synchroniser. Une commande symétrique est bien entendu saisie sur la machine

d'en face.

## V/. Installation de la Debian Etch sur les NEC

Comme les PC NEC qu'on a utilisé n'arrivent pas à booter les CD debian, la solution la plus simple est d'utiliser une clé usb pour faire une netinstall. Malheureusement, comme la carte ethernet non plus n'est pas reconnue par défaut sous Debian, il faut aussi en ajouter une autre.

Procédure pour faire une installation sur clé USB :

Télécharger <http://ftp.debian.org/debian/dists/sarge/main/installer-i386/current/images/hd-media/boot.img.gz> et s'assurer d'avoir syslinux installé. Cette image va permettre de créer un système d'installation de 128 Mb.

```
zcat boot.img.gz > /dev/sda
```

Montez ensuite la clé et copier une image ISO de type *netinst* ou *businesscard*. Le nom de ce fichier doit se terminer par `.iso`.

## VI/. Annexes

### 6.1 Redondance et monitoring

#### Récupérer les scripts de monitoring des services

Ces scripts permettent de donner vérifier l'état du service monitoré  
Avant tout du fait de l'utilisation de perl dans les scripts de monitoring, il faut installer les core files de perl.

```
apt-get install perl
```

Récupérons ensuite les scripts de monitoring (sur le site de nagios et sur le cvs de mon)

Monitor DRBD (version utilisée : 0.5.1) :

[http://www.nagiosexchange.org/Linux.55.0.html?&tx\\_netnagext\\_pi1\[p\\_view\]=296](http://www.nagiosexchange.org/Linux.55.0.html?&tx_netnagext_pi1[p_view]=296)

Monitor LDAP (version utilisée : 1.1.1.1):

ce script perl est disponible dans les sources de mon v1.2.0, il s'appelle `ldap.monitor`

Pour monitorer le serveur samba nous n'utiliserons pas de script mais simplement la commande `smbclient` qui permettra de savoir s'il est possible de monter un répertoire via samba. pour ce faire nous utiliserons un compte client sans

## Installation et configuration de Exim4

Pour permettre l'envoi de courriels d'alerte sur panne de service, nous devons configurer un utilitaire de messagerie. Nous installons ici exim:

```
apt-get install exim4
dpkg-reconfigure exim4-config
Dans les menus, faire:
- ne pas séparer la configuration dans plusieurs fichiers
- Distribution directe par SMTP
- nom de courriel de système : legolas.iutlp.u-strasbg.fr
- adresse d'écoute d'exim: 127.0.0.1
- autres destinations dont le courriel doit être accepté: legolas.iutlp.u-strasbg.fr
- domaine à relayer: mailhost.u-strasbg.fr
- machines à relayer: laisser vide
- ne pas minimiser les requetes DNS
- format mbox
```

## 6.2 Virtualisation

### Configurations de vm

Exemple de configuration d'une machine virtuelle xen (avec hvm dans ce cas)

```
# -*- mode: python; -*-
#=====
# Python configuration setup for 'xm create'.
# This script sets the parameters used when a domain is created using 'xm create'.
# You use a separate script for each domain you want to create, or
# you can set the parameters for the domain on the xm command line.
#=====

import os, re
arch = os.uname()[4]
if re.search('64', arch):
    arch_libdir = 'lib64'
else:
    arch_libdir = 'lib'

#-----
# Kernel image file.
kernel = "/usr/lib/xen/boot/hvmlloader"

# The domain build function. HVM domain uses 'hvm'.
builder='hvm'

# Initial memory allocation (in megabytes) for the new domain.
#
# WARNING: Creating a domain with insufficient memory may cause out of
```

```

#      memory errors. The domain needs enough memory to boot kernel
#      and modules. Allocating less than 32MBs is not recommended.
memory = 256

# Shadow pagetable memory for the domain, in MB.
# If not explicitly set, xend will pick an appropriate value.
# Should be at least 2KB per MB of domain memory, plus a few MB per vcpu.
# shadow_memory = 8

# A name for your domain. All domains must have different names.
name = "Samba"

# 128-bit UUID for the domain. The default behavior is to generate a new UUID
# on each call to 'xm create'.
#uuid = "06ed00fe-1162-4fc4-b5d8-11993ee4a8b9"

#-----
# The number of cpus guest platform has, default=1
vcpus=2

# Enable/disable HVM guest PAE, default=1 (enabled)
pae=1

# Enable/disable HVM guest ACPI, default=1 (enabled)
acpi=1

# Enable/disable HVM APIC mode, default=1 (enabled)
# Note that this option is ignored if vcpus > 1
#apic=1

# List of which CPUS this domain is allowed to use, default Xen picks
#cpus = ""      # leave to Xen to pick
#cpus = "0"     # all vcpus run on CPU0
#cpus = "0-3,5,^1" # run on cpus 0,2,3,5

# Optionally define mac and/or bridge for the network interfaces.
# Random MACs are assigned if not given.
#vif = [ 'type=ioemu, mac=00:16:3e:00:00:11, bridge=xenbr0, model=ne2k_pci' ]
# type=ioemu specify the NIC is an ioemu device not netfront
vif = [ 'type=ioemu, bridge=tap0' ]

#-----
# Define the disk devices you want the domain to have access to, and
# what you want them accessible as.
# Each disk entry is of the form phy:UNAME,DEV,MODE
# where UNAME is the device, DEV is the device name the domain will see,
# and MODE is r for read-only, w for read-write.

#disk = [ 'phy:hda1,hda1,r' ]
disk = [ 'file:/home/vm/linux-samba.img,hda,w', ',hdc:cdrom,r' ]

#-----
# Configure the behaviour when a domain exits. There are three 'reasons'
# for a domain to stop: poweroff, reboot, and crash. For each of these you
# may specify:
#
# "destroy",      meaning that the domain is cleaned up as normal;

```

```

# "restart",      meaning that a new domain is started in place of the old
#                  one;
# "preserve",    meaning that no clean-up is done until the domain is
#                  manually destroyed (using xm destroy, for example); or
# "rename-restart", meaning that the old domain is not cleaned up, but is
#                  renamed and a new domain started in its place.
#
# The default is
#
# on_poweroff = 'destroy'
# on_reboot   = 'restart'
# on_crash    = 'restart'
#
# For backwards compatibility we also support the deprecated option restart
#
# restart = 'onreboot' means on_poweroff = 'destroy'
#                  on_reboot   = 'restart'
#                  on_crash    = 'destroy'
#
# restart = 'always'  means on_poweroff = 'restart'
#                  on_reboot   = 'restart'
#                  on_crash    = 'restart'
#
# restart = 'never'   means on_poweroff = 'destroy'
#                  on_reboot   = 'destroy'
#                  on_crash    = 'destroy'

#on_poweroff = 'destroy'
#on_reboot   = 'restart'
#on_crash    = 'restart'

#=====

# New stuff
device_model = '/usr/' + arch_libdir + '/xen/bin/qemu-dm'

#-----
# boot on floppy (a), hard disk (c), Network (n) or CD-ROM (d)
# default: hard disk, cd-rom, floppy
boot="cda"

#-----
# write to temporary files instead of disk image files
#snapshot=1

#-----
# enable SDL library for graphics, default = 0
sdl=1

#-----
# enable VNC library for graphics, default = 1
vnc=1

#-----
# address that should be listened on for the VNC server if vnc is set.
# default is to use 'vnc-listen' setting from /etc/xen/xend-config.sxp
#vnclisten="127.0.0.1"

```

```

#-----
# set VNC display number, default = domid
#vncdisplay=1

#-----
# try to find an unused port for the VNC server, default = 1
#vncunused=1

#-----
# enable spawning vncviewer for domain's console
# (only valid when vnc=1), default = 0
#vncconsole=0

#-----
# set password for domain's VNC console
# default is depends on vncpasswd in xend-config.sxp
vncpasswd='test'

#-----
# no graphics, use serial port
#nographic=0

#-----
# enable stdvga, default = 0 (use cirrus logic device model)
stdvga=0

#-----
# serial port re-direct to pty device, /dev/pts/n
# then xm console or minicom can connect
serial='pty'

#-----
# Qemu Monitor, default is disable
# Use ctrl-alt-2 to connect
#monitor=1

#-----
# enable sound card support, [sb16|es1370|all|...], default none
#soundhw='sb16'

#-----
# set the real time clock to local time [default=0 i.e. set to utc]
localtime=1

#-----
# set the real time clock offset in seconds [default=0 i.e. same as dom0]
#rtc_timeoffset=3600

#-----
# start in full screen
#full-screen=1

```

```

#-----
# Enable USB support (specific devices specified at runtime through the
# monitor window)
#usb=1

# Enable USB mouse support (only enable one of the following, `mouse' for
# PS/2 protocol relative mouse, `tablet' for
# absolute mouse)
#usbdevice='mouse'
#usbdevice='tablet'

#-----
# Set keyboard layout, default is en-us keyboard.
keymap='fr'

```

## Notes diverses

Configuration de ldap+samba, après importation de l'annuaire ldap et la configuration de samba, il faut ajouter à samba le compte/mot de passe (admin dans ce cas) de l'annuaire ldap (informations définies dans le *slapd.conf* pour le mot de passe, l'utilisateur étant défini dans le *smb.conf*)

```
smbpasswd -w admin
```

Pour changer le mot de passe de uid=root,o=iut,o=ulp, qui permet l'insertion des machines windows dans le domaine, il faut utiliser la commande suivante :

```
smpasswd root
```

tests:

changement de serveur avec heartbeat :

- perte de la connexion avec l'explorateur, mais recup en en lançant un autre
- enregistrement du profile
- doc oo : enregistrer -> erreur i/o  
enregistrer sous ->ok

Dans la configuration d'un domaine samba, le contrôleur de domaine possède un identifiant samba local et pour le domaine

```

~# net getlocalsid
SID for domain PDC is: S-1-5-21-3604470699-2325248622-2544762165

~# net getdomainsid
SID for domain PDC is: S-1-5-21-3604470699-2325248622-2544762165
SID for domain ARDA is: S-1-5-21-3604470699-2325248622-2544762165

```

Il faut absolument que le contrôleur de domaine de backup ait les mêmes id que le principal et il faut que le local id soit le même que le domain id. Ces identifiants sont modifiables via, par exemple :

```
~# net setlocalsid S-1-5-21-3604470699-2325248622-2544762165
```

## 6.3 Sources

XEN

<http://www.unixgarden.com/index.php/administration-systeme/faire-fonctionner-les-bsd-dans-xen>  
[http://www.freebsd.org/doc/fr\\_FR.ISO8859-1/books/handbook/virtualization-guest.html](http://www.freebsd.org/doc/fr_FR.ISO8859-1/books/handbook/virtualization-guest.html) : section 21.2.2

KVM

<http://www.lefinnois.net/wp/index.php/2007/10/13/debian-et-machine-virtuelle-kvm/>  
<http://www.lefinnois.net/wp/index.php/2007/10/14/kvm-dans-un-screen-encore-avec-le-monitor-kvm/>

CARP / PFSYNC

Le hors série numéro 29 : BSD acte 1