Université de La Rochelle

Rapport de projet

Dossier d'Architecture Technique

Auteurs :
Joris BERTHELOT
Laurent LE MOINE

Superviseurs :
Philippe HARRAND
Bertrand VACHON

Master ICONE 2011-2012

Table des matières

Introduction

Dans le cadre de notre formation Master Ingéniérie Informatique et de son Unité d'Enseignement Architecture : Conception et Gestion, nous avons réalisé un projet d'architecture réseau HA (High Availability) en 3 jours seulement.

Ce projet nous a permis de mettre en exergue nos connaissances récemment acquises lors des cours respectifs de la même UE mais aussi de reprendre et appliquer les concepts vus en TP la semaine auparavant.

Assigné comme table nř5, nous avons utilisé les postes suivants :

Joris Berthelot (sera la machine « JB » dans le reste du rapport)

- Adresse IP : 10.192.10.23
- Host : mamba13

Laurent Le Moine (sera la machine « LLM » dans le reste du rapport)

- Adresse IP : 10.192.10.24
- Host : mamba14

Etant donné que ce projet fut réalisé en équipe, le code source des différents scripts et fichiers de configuration sont disponible sur Google Code + Subversion. Ainsi, vous pouvez à tout moment récupérer notre travail (ainsi que le code LaTeX du document) comme ceci :

```
svn export http://ulr-acg.googlecode.com/svn/trunk/ ulr-acg-src
```

Première partie

Infrastructure logicielle

Avant toute choses, vous devez savoir que l'ensemble des opérations décrites dans cette section sont a réaliser avec l'utilisateur root. Si vous lancez les scripts livrés avec le rapport sans être root, vous aurez droit à un gentil message d'erreur.

Nous avons aussi par ailleurs vidé et désactivé les tables de pare-feu afin de laisse toute nos applications travailler sans avoir de gêne dans un premier temps :

```
# Flushes iptables
/sbin/chkconfig --del iptables
# Disables firewall service
service iptables stop
# Enables Network Time Protocol to sync time between machines
service ntpd start
```

0.1 Installation des paquets

Avant de commencer à configurer et déployer les services, nous aurons besoin d'installer un certain nombre de paquets afin de pouvoir parvenir à nos fins. Aussi divers que variés, nous avons scripté cette installation afin de faciliter la tâche.

0.1.1 Configuration du proxy

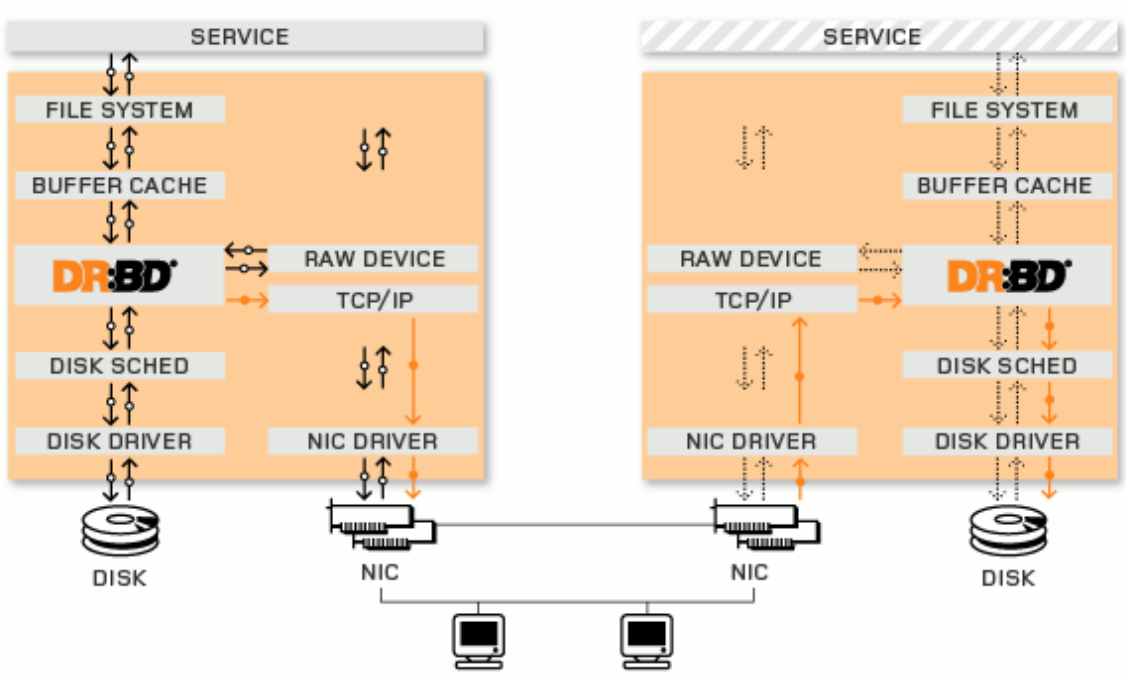
Il faudra auparavant configurer manuellement les paramètres du proxy si besoin afin de ne pas rendre le script d'installation inopérationnel. Pour se faire, veillez à bien changer les paramètres dans les Serveur mandataires (Système > Configuration > Serveurs mandataires) ainsi que rajouter les bons paramètres à Yum (proxy) :

```
# Configuration de Yum
vim /etc/yum.conf
```

0.2 Réplication bas niveau

La réplication bas niveau permet de créer non pas une redondance applicative mais directement sur le support des données applicatives (système de fichier). L'intêret à cela est d'éviter de configurer chaque service pour sa propre réplication (si existant) et d'aller droit au but en répliquant directement le volume sur lequel repose les données.

Voici un petit schéma (issu du site de DRBD) afin d'imager le concept :



0.2.1 Installation

Pour la réplication bas niveau (système de fichiers), nous avons utilisé DRBD : un logiciel permettant de faire de la réplication de données au sein d'une architecture en grappe. DRBD est assez complexe et fastidieux à mettre en place car il demande quelques notions assez poussées sur les volumes, leur synchronisations, etc.

Ce logiciel ne fonctionnait autrefois qu'en mode maître/esclave mais depuis les dernières versions, on peut partir sur une configuration maître/maître afin que les données soient bien synchronisées de manière bidirectionnelle.

```
# Installation de drbd
yum install -y drbd drbd-pacemaker drbd-udev
```

0.2.2 Configuration

La configuration de DRBD peu s'avérer très simple mais permet un certain degré de complexité en fonction des architectures. La syntaxe est claire et s'apparente à celle du serveur de nom.
Voici la configuration que nous avons utilisé :

```
global {
    # Enables statistics usage
    usage-count yes;
}
common {
    # Maner to sync data, it's flagged as completed when both disks has written
    protocol C;
}
resource ulr-data {
    # Stores data meta-data in the volume
    meta-disk internal;
    # DRDB device
    device  /dev/drbd1;
    # Volume to work on
    disk    /dev/mapper/ulr--acg-ulr--data;
    syncer {
        # Hash method to check data integrity
        verify-alg sha1;
        # Network sync speed
        rate 100M;
    }
    # Allows both machine are primary
    net {
        allow-two-primaries;
    }
    # JB's machine
    on mamba13 {
        address     10.0.0.23:7789;
    }
    # LLM's machine
    on mamba14 {
        address     10.0.0.24:7789;
    }
}
```

DRBD implique qu'un lien réseau doit être établi en supplément des liens existants. Il est important de comprendre que le DRBD utilise un réseau à lui propre afin d'y transférer les données.

Pour vérifier que DRBD fonctionne bien, il suffit de voir son état en faisant la commande suivante :

```
cat /proc/drbd
```

0.3 Déploiement des services

Dans cette partie, il est important de comprendre que lors de la mise en place d'une architecture en grappe avec des noeuds répliqués, il faut toujours un noeud de référence, surtout dans une architecture active/passive comme celle que nous allons mettre en place. Suivant la machine sur laquelle vous allez lancer les scripts, il faudra ou non déployer les données.

0.3.1 Apache

0.3.1.1 Installation

Pour installer Apache, il vous faudra très simplement lancer son script d'installation dans le répertoire `scripts/apache.sh`. Ce script va essayer de stopper Apache, vérifier l'intégrité de son fichier de configuration et si il ne concorde pas avec le notre, il va le remplacer. Ensuite, selon si vous êtes le premier noeud de la grappe.

0.3.1.2 Configuration

La configuration d'Apache est très légèrement spécifique dû à notre application Web qui utilise Silex mais sinon rien de bien particulier ormis la configuration de l'URL `/server-status` qui doit être disponible pour Pacemaker.

Il faudra donc bien décommenter le bout de code suivant à la fin du fichier :

```
<Location /server-status>
    SetHandler server-status
    Order deny,allow
    Deny from all
    # Only from localhost where Pacemaker runs
    Allow from 127.0.0.1
</Location>
```

Enfin, spécifier le `DocumentRoot` à `/var/cluster/www/org/tp/g1b5/web` et y ajouter les règles de réécritures suivantes afin de faire fonctionner notre application :

```
<IfModule mod_rewrite.c>
    Options -MultiViews
    RewriteEngine On
    # All requests which are not a file
    RewriteCond %{REQUEST_FILENAME} !-f
    # Except this request
    RewriteCond %{REQUEST_URI} !=/server-status
    RewriteRule ^ index.php [L]
</IfModule>
```

0.3.2 MySQL

0.3.2.1 Installation

L'installation de MySQL se fait très facilement grâce au script d'installation `scripts/mysql.sh`. Le script arrête le serveur MySQL, charge le fichier de configuration `confs/mysql/my.cnf` à la place de l'ancien. Il démarre ensuite le service mysqld, puis crée les utilisateurs et peuple la base si nous sommes sur le premier noeud de la grappe.

0.3.2.2 Configuration

Le fichier de configuration de MySQL est court, mais il est important dans notre cas de bien préciser le chemin du répertoire ou seront stockées les bases, à savoir dans notre cluster, dans le répertoire `/var/cluster/mysql`. De même, il est aussi important de préciser que l'adresse utilisées

par le serveur est celle de l'adresse du HAProxy : 10.192.10.50.

```
[mysqld]
# On définit le répertoire ou seront stockées les données sur le cluster
datadir=/var/cluster/mysql
# On définit l'adresse IP utilisée par le serveur MySQL
bind-address=10.192.10.50
# Chemin du fichier de socket pour les connexions locales
socket=/var/lib/mysql/mysql.sock
user=mysql
# On désactive les liens symboliques pour améliorer la sécurité du serveur
symbolic-links=0

[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

Nous devons aussi définir le mot de passe de l'utilisateur "root", qui est le même sur les deux machines. Cela ce fait simplement grâce à la commande :

```
mysqladmin password <mot_de_passe>
```

Il faut ensuite créer un utilisateur "tpuser", et lui allouer des droits de sélection sur tout les machines du réseau 10.192.10.0, afin de permettre l'accès à la base "projet_hd" depuis une machine distante.

```
CREATE USER 'tpuser'
IDENTIFIED BY 'tpuser';

GRANT ALL PRIVILEGES
ON *.* TO 'root'@'%';

CREATE DATABASE projet_hd;

GRANT SELECT PRIVILEGES
ON projet_hd.* TP 'tpuser'@'10.192.10.%';
```

Il faut aussi créer puis initialiser la table "product", qui contient les données sur notre stock. Elle contient les champs "id", "name", "price" et "quantity".

0.3.3 DNS

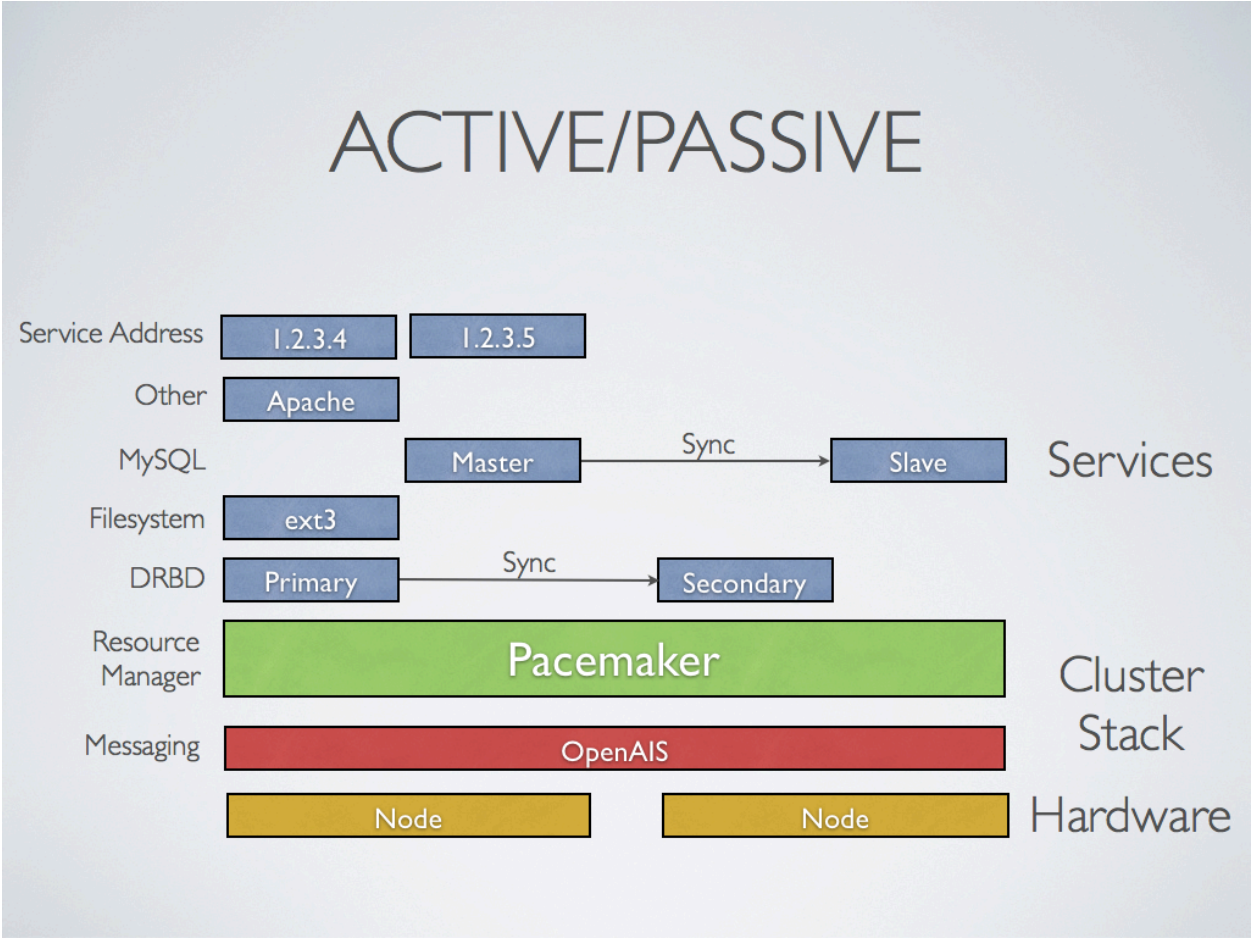
0.3.4 LDAP

0.4 Mise en place de Pacemaker et Corosync

Dans une architecture HA, le système doit pouvoir constamment suivre son état et celui de son entourage afin de décider si il doit basculer (= "failover") ou non vers un noeud de secours. Pour cela, avec Fedora, nous avons choisi d'utiliser le service Pacemaker couplé à Corosync car ils nous ont semblé très complets et très professionnels.

0.4.1 Présentation

Pour notre projet, nous avons choisi de mettre en place une architechure en grappes avec un système actif/passif. Ci-dessous un schéma de l'architechure :



Les « node » correspondent à nos machines (JB et LLM), la couche de messagerie OpenAIS correspondra au service Corosync qui est un projet dérivé d'OpenAIS. Pour les couches supérieures à Pacemaker, nous les avons vu précédemment.

Corosync est un moteur de clustering mettant en oeuvre une messagerie de service sur réseau en utilisant une adresse et un port multicast.

0.4.2 Installation

```
# Installation de pacemaker et de corosync
yum install -y pacemaker corosync
```

0.4.3 Configuration

0.4.3.1 Corosync

Avant toute chose, voici les adresses IP nécessaires :

Adresse IP multicast 239.0.0.1

Port multicast 6800

Réseau de la grappe 10.192.10.0

Adresse IP de la grappe 10.192.10.50

On va commencer par la configuration de Corosync qui est simpliste. Pour cela, on prends le fichier de configuration par défaut et on va simplement y changer les adresses IP comme ceci :

```
CONF="/etc/corosync/corosync.conf"
cp -f $CONF.example $CONF
sed -i.bak "s/.*mcastaddr:.*/mcastaddr:\ 239.0.0.1/g" $CONF
sed -i.bak "s/.*mcastport:.*/mcastport:\ 6800/g" $CONF
sed -i.bak "s/.*bindnetaddr:.*/bindnetaddr:\ 10.192.10.0/g" \$CONF
```

Ensuite, on va signaler à Corosync de charger le plugin Pacemaker afin qu'il puisse "travailler" avec :

```
cat << EOT > /etc/corosync/service.d/pcmk
service {
        # Load the Pacemaker Cluster Resource Manager
        name: pacemaker
        ver:  1
}
EOT
```

Une fois configuré, nous pouvons lancer Corosync :

```
/etc/init.d/corosync start
```

0.4.3.2 Pacemaker

Pacemaker offre une invite de commande propre à lui-même (comme les switch Cisco) que nous pouvons lancer en tapant « crm ». Il est intéressant de savoir qu'en interne, la configuration de Pacemaker est formattée en XML car suite à quelques erreurs de notre part, nous avons pu constater des erreurs de validation d'entrée contre des Schémas XML.

Afin de vous épargner la configuration à la main de Pacemaker, nous pouvons exporter et injecter directement une configuration donnée :

```
crm configure load replace /path/to/conf/file
```

Mais avant cela, vous devez démarrer Pacemaker et vider sa configuration actuelle (si existante) :

```
/etc/init.d/pacemaker start
cibsadmin -E --force
```

Voici donc notre fichier de configuration pour Pacemaker (la version initiale qui marchait correctement) :

```
primitive ClusterIP ocf:heartbeat:IPaddr2 params ip="10.192.10.50" \
    cidr_netmask="24" op monitor interval="5s"
primitive FS ocf:heartbeat:Filesystem params \
    device="/dev/drbd/by-res/ulr-data" \
    directory="/var/cluster" fstype="ext4"
primitive UlrData ocf:linbit:drbd params drbd_resource="ulr-data" \
    op monitor interval="30s"
primitive WebApp ocf:heartbeat:apache params \
    configfile="/etc/httpd/conf/httpd.conf" op monitor interval="10s"
primitive MySQL lsb:mysqld
primitive LDAP lsb:slapd
ms UlrDataClone UlrData meta master-max="1" master-node-max="1" \
    clone-max="2" clone-node-max="1" notify="true"
colocation WebApp-with-FS inf: WebApp FS
colocation FS-with-UlrDataClone inf: FS UlrDataClone:Master
colocation WebApp-with-ClusterIP inf: WebApp ClusterIP
colocation MySQL-with-ClusterIP inf: MySQL ClusterIP
colocation LDAP-with-ClusterIP inf: LDAP ClusterIP
order MySQL-after-FS inf: FS MySQL
order LDAP-after-FS inf: FS LDAP
order WebApp-after-FS inf: FS WebApp
order WebApp-after-ClusterIP inf: ClusterIP WebApp
order FS-after-UlrDataClone inf: UlrDataClone:promote FS:start
property $id="cib-bootstrap-options" \
    dc-version="1.1.6-1.fc14-b379478e0a66af52708f56d0302f50b6f13322bd" \
    cluster-infrastructure="openais" \
    expected-quorum-votes="2" \
    stonith-enabled="false" \
    no-quorum-policy="ignore"
```

Dans ce fichier, on peut y entrevoir des déclarations de ressources comme une IP "flottante" (`10.192.10.50`) sur laquelle le monde extérieur va se connecter au services que contient la grappe, des ressources de battement de coeurs pour un système de fichier, les serveurs Apache, MySQL, LDAP et enfin la ressource de réplication bas niveau DRBD.

Mais ce n'est que la déclaration des ressources, ensuite viennnent les dépendances de "colocation" qui obligent deux ressources données à toujours être de paire et enfin les ordres de démarrage des ressources en fonction des dépendances. Par exemple, Apache ne peut pas démarrer avant que le système de fichier soit opérationnel.

0.4.4 Utilisation

Pour ~~admirer~~ surveiller le bon fonctionnement de Pacemaker, il y a la commande `crm_mon`.

En ce qui concerne la gestion des noeuds ou des ressources, il faut utiliser la commande `crm`. Voici quelques exemples :

```
# Vérifier le status d'une ressource
crm resource status Apache
# Redémarrer une resource
crm resource restart MySQL
# Déplacer une ressource sur un autre noeud
crm ressource move FS mamba14
# Voir le status d'un noeud
crm node status mamba13
# Mettre un noeud en standby (simulation du failover)
crm node standby mamb14
# Et le rendre disponible à nouveau
crm node online mamba14
```

Deuxième partie

Tests

0.5 Serveur de noms

0.6 Serveur Apache

0.7 Serveur MySQL

0.8 Serveur LDAP

Troisième partie

Conclusion

0.9 Difficultés rencontrés

Durant ce projet, la contrainte majeur était de réaliser une telle architechure en 3 jours seulement sans avoir aucune expérience dans le domaine. Nous avons souhaité nous orienter vers des choix fonctionnels et matures plutôt qu'universitaires et peu professionnels.

Le premier facteur de difficulté aura été l'abscence d'indications concernant le choix des technologies car nous avons réellement été soumis à un environnement pragmatique où nous jouons le rôle des responsables SI vers qui on vient se conseiller suite une problématique bien présente. Le choix et la décision des technologies est une chose mais ce qui est le plus chronophage est de regarder les produits et les solutions existantes sur le marché.

Une fois les choix réalisés, il a fallu se documenter pour savoir implémenter au mieux possible l'infrastructure logicielle. Nous ne vous cacherons pas que nous avons lu 90% de documentation en anglais durant ces 3 jours. De l'anglais pas toujours facile à comprendre de part ses termes techniques mais surtout de part la notion nouvelle que nous découvrions sur le tas.

La difficulté suivante fut liée aux machines elles-même : impossible de prévoir que tout va fonctionner du premier coup, le fait que les machines aient besoin d'un serveur mandataire pour accèder au Web rajoute quelques contraintes sur les configurations mais surtout le manque de maîtrise complet sur les distributions Fedora nous aura fait perdre un peu de temps non négligeable.

Malgré tout cela, nous sommes parvenus automatiser de manière relativement complète l'installation, la configuration et le déploiement de notre projet à l'aide de Subversion et de scripts Bash.

0.10 Retours sur échec

Après 3 jours de recherche, de développement et de tests acharnés sur les différentes briques de notre architechure, nous avons échoué lors de la présentation de notre projet. En voici les raisons majeures :

1. Manque d'expérience avec les technologies utilisées
2. Manque d'organisation et de rigueur dans les procédures
3. Manque de temps (implique du stress)
4. Analyses sur tests défaillants pas assez complètes

D'une certaine manière, tous ces facteurs sont liés car ils s'entre-croisent et s'encouragent et nous le savions mais pas c'est pas toujours évident d'avoir le temps de prendre le recul nécessaire afin de mieux repartir.

Après des heures intensives de travail, nous avions réussi à faire tourner notre architechure active/passive en simulant des failovers et des failback avec succès mais sans tester tous les cas possibles et surtout les cas d'évaluation. Suite à ce succès précipité, nous avons voulu optimiser la configuration de Pacemaker afin de la simplifier mais nous nous sommes rendu compte après analyse que cette nouvelle configuration ne répondait plus au contraintes d'ordre de basculement des services lors du failover.

Dans le stress et la précipitation, nous avons voulu essayer une architechure active/active car nous pensions que le type d'architechure active/passive ne répondait pas à la demande de l'évaluation (analyse d'échec pas assez poussée) mais près quelques heures d'installation et de configuration, cette nouvelle monture n'a pas été un succès à cause d'un soucis de synchronisation de volume avec DRBD bien que la nouvelle configuration active/active de Pacemaker était déjà prète.

0.11 Retours personnels