Centre de Données astronomiques de Strasbourg

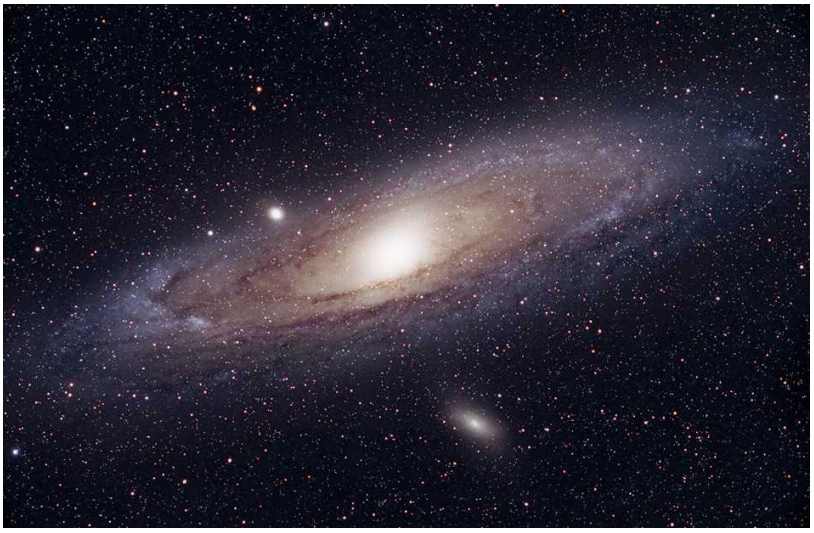
iut Nancy-Charlemagne

Licence **P**rofessionnelle
Administration de **S**ystèmes, **R**éseaux et **A**pplications à base de **L**ogiciels **L**ibres

---

Rapport de Stage — Année universitaire 2008-2009

Stockage hautement-disponible pour l'Observatoire Virtuel



Tuteur du stage :
André Schaaff
Responsable universitaire :
Philippe Dosch
Auteur du rapport :
Vincent Meslard

Table des matières

Remerciements

Je tiens tout d'abord à remercier M. André Schaaff, ingénieur de recherche, pour m'avoir proposé ce stage, accueilli au sein du Centre de Données astronomiques de Strasbourg et encadré tout au long de cette expérience.

Je remercie également tous les membres de l'Observatoire que j'ai pu côtoyer et qui m'ont accueilli avec sympathie, en particulier M. Jean-Yves Hangouet et M. Thomas Keller pour leur patience et leurs conseils qui m'ont permis de mener à bien mon projet.

Enfin, je tiens à remercier Mathias Depretz et Anaïs Oberto, dont j'ai partagé le bureau. Mathias pour s'être accommodé de ma présence et m'avoir fait redécouvrir les bières locales (en dehors des heures de bureau bien sûr). Anaïs pour avoir supporté mes blagues pas toujours bien senties. Je remercie finalement Dominika, William et Benjamin, stagiaires durant la même période que moi, pour leur soutient pendant le déroulement du stage et l'écriture de ce rapport.

Résumé

Ce rapport concerne le stage que j'ai effectué dans le cadre de la licence professionnelle Administration de Systèmes, Réseaux et Applications à base de Logiciels Libres (*ASRALL*), et qui s'est déroulé au Centre de Données astronomiques de Strasbourg, laboratoire de recherche de l'Observatoire Astronomique.

Ce stage a pour buts la modification et la validation d'un prototype de stockage hautement-disponible pour l'Observatoire Virtuel ainsi que la mise en place de procédures d'installation et de configuration des différents éléments.

Nous présentons ici l'ensemble du travail effectué. Après une présentation du laboratoire et du stage en lui-même, nous verrons un court état de l'art sur les solutions libres disponibles à certains niveaux de notre architecture. Nous verrons ensuite le détail des choix de paramètres de configuration des différentes solutions retenues puis la mise en œuvre de celles-ci avec les procédures d'installation et de test. Enfin nous dresserons un bilan autant au niveau technique que personnel et conclurons sur les apports du stage.

Chapitre 1

Présentation de L'Observatoire

1.1 Présentation générale

L'observatoire astronomique est une unité du *CNRS* (Centre National de Recherche Scientifique) et de l'Université de Strasbourg. Il est situé sur le campus de l'Esplanade, non loin du centre ville de Strasbourg. Son personnel se compose d'astronomes, d'informaticiens et de documentalistes. Les enseignements suivants y sont dispensés :

- Master « Analyse et Traitement des Données sur les Milieux Astronomiques »,
- Licence en Sciences et autres licences (enseignements d'ouverture),
- Licence de Géosciences,
- Préparation au Certificat d'Aptitude au Professorat de l'Enseignement du Second degré (*CAPES*) et à l'Agrégation,
- Master Applications des Technologies Spatiales,
- Diffusion de la Culture.

A proximité des trois bâtiments de l'observatoire, se trouve, le Planétarium destiné à l'astronomie grand public, et où les gens peuvent venir s'initier à la découverte de l'univers.

1.2 Les domaines de recherche

Au même titre que l'espace, les domaines de recherche en astronomie sont vastes. C'est pourquoi on y trouve différentes équipes de recherche spécialisées chacune dans un domaine.

Hautes énergies

L'équipe Astrophysique des Hautes énergies a pour thème l'étude des astres et sites de l'univers émetteurs de photons de haute énergie. Cette thématique générale recouvre des aspects variés, comme l'étude des astres compacts en

fin d'évolution, la physique de leur activité, les phénomènes de haute énergie intéressant les étoiles jeunes ou le soleil, ou l'étude de ces phénomènes à l'échelle galactique. Ces recherches se sont largement appuyées sur les données acquises par le satellite *ROSAT* (RÖntgenSATellit, satellite artificiel allemand d'observation des rayons X de 1990 à 1999) et s'appuient désormais sur celles des satellites X de nouvelle génération, tout spécialement *XMM-NEWTON*.

Étoiles et systèmes stellaires

Les recherches menées par l'équipe « Étoiles et systèmes stellaires » recouvrent un domaine étendu, incluant les étoiles, les milieux interstellaires, la galaxie et les galaxies proches. De l'étoile, objet individuel, l'intérêt s'est porté aux groupes d'étoiles, témoins de l'évolution stellaire mais aussi traceurs des grandes structures de la Voie Lactée.

Galaxies

Les activités de l'équipe sont centrées sur les problèmes de la structure du Groupe Local (ensemble d'une petite quarantaine de galaxies auquel appartient la nôtre), de ses populations stellaires et sur la dynamique gravitationnelle. De plus, l'équipe possède un savoir- faire sur les outils statistiques d'analyse de données et sur les méthodes inverses non paramétriques. Un des objectifs consiste à combiner les informations d'évolution des populations stellaires et celles de dynamique afin de reconstituer les événements déterminants liés aux processus de formation et d'évolution galactique.

Le centre de données (CDS)

L'activité de recherche du *CDS* s'est concentrée sur l'étude de la dynamique galactique et des populations d'étoiles binaires, sur une participation importante à la mission *HIPPARCOS2* (HIgh Precision PARallax COllecting Satellite, satellite de mesure de parallaxe à haute précision) de l'Agence Spatiale Européenne (*ESA*, European Space Agency), ainsi que sur le développement de méthodologies nouvelles applicables à l'analyse et au traitement de données astronomiques. Il participe activement à la mise en place de l'Observatoire Virtuel que nous décrirons plus loin.

1.3 Le Centre de Données astronomiques de Strasbourg

Présentation

Le *CDS* joue un rôle dans d'importantes missions astronomiques spatiales : contribuant aux catalogues d'étoiles guides, aidant à identifier les sources observées, organisant l'accès aux archives, etc. Le service a également signé des accords d'échanges internationaux avec les organismes suivants :

- la *NASA* (National Aeronautics and Space Administration),
- le *NAO* (National Astronomical Observatory à Tokyo, Japon),
- l'Académie des Sciences de Russie,
- le réseau PPARC Starlink au Royaume-Uni,
- l'Observatoire de Beijing (Chine),
- l'Université de Porto Allegre au Brésil,
- l'Université de La Plata en Argentine,
- l'InterUniversity Center for Astronomy and Astrophysics (Inde).

Le *CDS* est membre de la Fédération des Services d'Analyse de Données Astrophysiques et Géophysiques.

Le *CDS* coopère aussi avec l'Agence Spatiale Européenne (ASE) via le transfert au *CDS* du service de catalogues du projet *ESIS* (European Space Information System), le projet *VizieR*. Il coopère également avec la *NASA*, en particulier le *CDS* abrite une copie miroir du Système de Données Astrophysiques (*ADS*) et *ADS* abrite une copie miroir de *SIMBAD*. Le *CDS* contribue aussi au projet *NASA AstroBrowse* et abrite aussi les copies miroirs Européennes des journaux de l'American Astronomical Society (*AAS*).

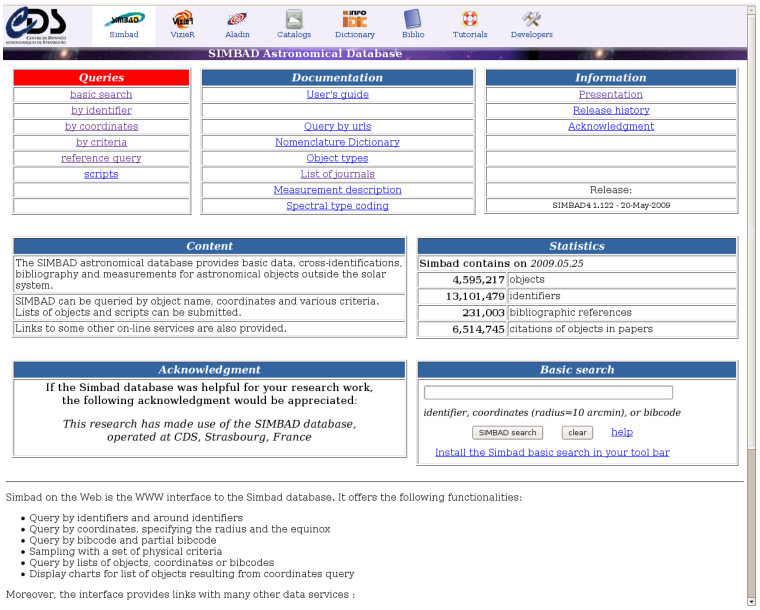
1.4 Les services du CDS

Le *CDS* développe et maintient trois principaux services qui constituent chacun des références dans le milieu de l'astronomie.

SIMBAD

SIMBAD est une base de données d'identificateurs d'objets astronomiques permettant d'associer à un objet tous les identificateurs cités dans la littérature scientifique depuis 1980. Cette base de données contient actuellement 13 000 000 d'identificateurs pour 4 500 000 d'objets et 230 000 articles référencés. Ce service qui traite 100 000 requêtes par jour est aujourd'hui une référence mondiale et ne connaît pas d'équivalent. L'an dernier, 11 000 000 d'identificateurs, 3 000 000 d'objets et 110 000 articles étaient référencés ce qui montre une croissance significative.

FIG. 1.1 – Page d'accueil de SIMBAD



Le service permet de retrouver un objet astronomique à l'aide de son nom s'il est contenu dans les articles de la base de données. Il fournit des informations de base sur l'objet comme sa position, son type, quelques mesures effectuées ainsi que l'ensemble des articles dans lesquels il est cité.

VizieR

VizieR est une base de données regroupant plusieurs milliers de catalogues d'objets astronomiques et permet d'accéder d'une manière uniforme aux données, de les croiser, de les exporter, etc. *VizieR* recherche dans plus de 7 000 catalogues de différentes tailles, certains de petite envergure mais aussi d'autres très importants comme le catalogue *2MASS* comportant près de 500 000 000 d'objets, *GSC2* 950 000 000, *USNO* un milliard ou *NOMAD* contenant également un milliard de lignes. Ce service répond à plus de 100 000 requêtes par jour.

Aladin

Aladin est un atlas interactif du ciel ainsi qu'un logiciel de visualisation et de traitement d'images astronomiques. Il permet entre autres de générer des images en couleur à base d'images du ciel prises à différentes longueurs d'ondes en appliquant un filtre RVB (rouge, vert, bleu) de colorisation pour chacune d'elles. Il permet également de superposer sur l'image obtenue, des catalogues

Bienvenue sur le site de l'Observatoire Astronomique de Strasbourg
L'Observatoire astronomique

FIG. 1.2 – Page d'accueil de VizieR

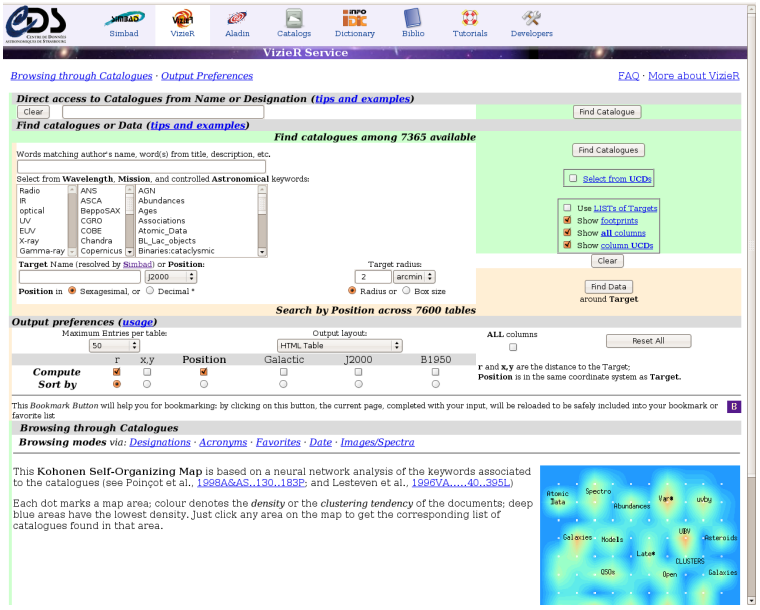
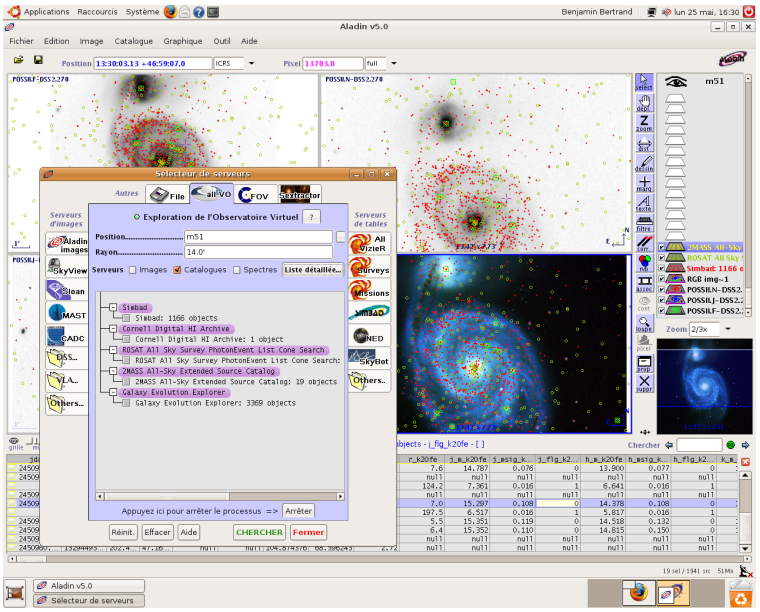


FIG. 1.3 – Exemple d'utilisation d'Aladin



de données à partir de *VizieR* ou *SIMBAD*, qu'il peut interroger par le biais d'une interface simplifiée.

On notera également qu'entre 6 et 10 000 requêtes par jour sont effectuées sur *Aladin* de la part de 400 à 500 utilisateurs différents.

Chapitre 2

Présentation du sujet

2.1 L'existant

2.1.1 Contexte

Que ce soient les chercheurs, les documentalistes ou bien les applications pour leur usage interne, il est nécessaire de pouvoir stocker de façon fiable, temporairement ou pour longtemps des données de tout ordre allant de simples fichiers textes de description aux images haute-résolution. Il fallait ainsi une plateforme adéquate afin de mettre à disposition un espace de stockage hautement-disponible et ce de n'importe où et depuis diverses applications. Le but est donc que les chercheurs, tout comme les applications, puissent partager des données et méta-données entre eux sans se soucier de l'architecture de stockage. Dans le cadre d'un projet européen de l'*Euro-VO* (l'Observatoire Virtuel Européen), un prototype a été développé afin d'être mis en production au *CDS*.

L'*Euro-VO* participant à l'*IVOA* (Alliance Internationale des Observatoires Virtuels), cet espace de stockage se doit de respecter les spécifications de cette dernière. En l'occurrence, il s'agit de la spécification *VOSpace* qui est l'interface pour le stockage distribué de l'*IVOA*. *VOSpace* correspond donc à la représentation externe des données. La façon de stocker les données est donc totalement détachée de cette interface et le choix du *CDS* s'est porté sur *iRODS* qui répond en partie à la problématique exposée dans le premier paragraphe.

2.1.2 Architecture en place

iRODS est développé au *DICE* (Data Intensive Cyber Environments Center) de l'Université de Californie à San Diego. Ses caractéristiques principales sont :

- la création de collections de données partageables à partir de systèmes de fichiers et d'archives sur bande,
- la gestion d'un catalogue permettant la recherche, la gestion, le contrôle et le suivi de l'accès aux données et leurs manipulations,

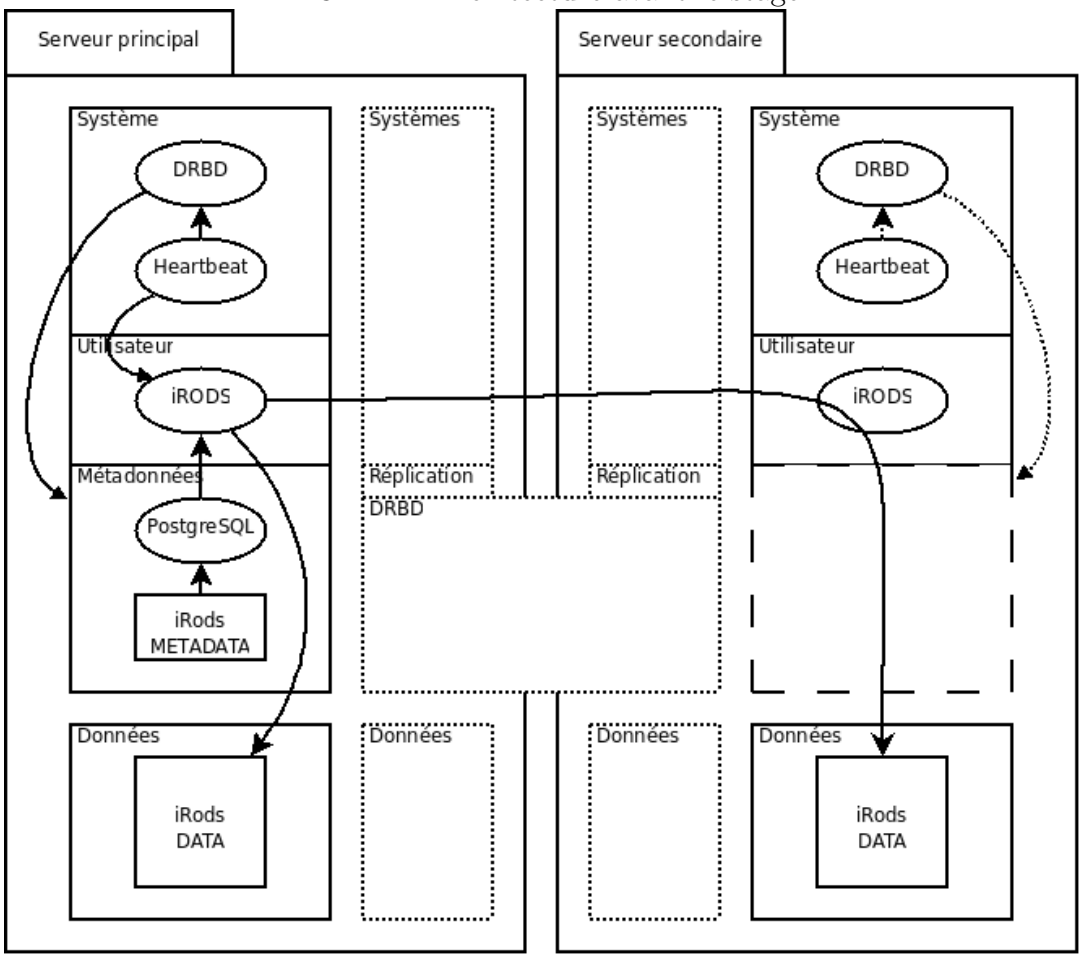
– l'utilisation de règles et politiques sur les données pour permettre par exemple l'automatisation des tâches administratives ou la gestion de collections à grande échelle.

Le prototype a ainsi été développé pour faire reposer *iRODS* sur une architecture fiable, hautement-disponible et maintenable. L'architecture proposée (voir figure 2.1 page 8) est composée de :

– *Heartbeat* pour la haute-disponibilité des données,
– *DRBD* pour la réplication des métadonnées,
– *iRODS* pour la gestion des données ainsi que leur réplication.

Cependant, si le serveur principal est hors-service, le serveur secondaire reprend la main mais de manière « dégradée ». De plus le retour à la normale n'est pas automatique et l'absence de possibilité de synchronisation des données par *iRODS* nous oblige à effectuer cette tâche à la main.

FIG. 2.1 – Architecture avant le stage



2.2 Ce qu'il faut améliorer

En plus de tests en ce qui concerne la fiabilité et la haute-disponibilité des données, l'étude du problème doit amener à affiner les paramètres.

2.2.1 Modifications

Les modifications à effectuer doivent permettre de répondre le plus correctement possible à la problématique et de combler les lacunes du prototype. Les points touchés sont :

- la réplication des données,
- la reprise du service par le matériel de remplacement,
- la simplification de l'architecture pour la maintenance.

2.2.2 Ajouts

Les ajouts nécessaires pour répondre au problème sont :

- la création d'un mode opératoire pour l'installation ou remplacement du matériel et des logiciels,
- la mise en place de procédures de test.

Chapitre 3

Etat de l'art / Tests préalables

3.1 Haute-disponibilité

3.1.1 Les différentes solutions

Heartbeat (v1)

Heartbeat est le programme principal du projet *Linux-HA* [1] qui vise à fournir une solution de haute-disponibilité pour Linux, FreeBSD, OpenBSD, Solaris et Mac OS X.

Heartbeat est un démon qui fournit les services d'appartenance à un cluster et de communication à l'intérieur de celui-ci. Cela permet aux clients de connaître l'état de présence des processus sur les autres machines et d'échanger facilement des messages entre elles.

Afin d'être utile, *Heartbeat* doit être combiné à un manageur de ressources (*CRM* pour *Cluster Resource Manager*) qui a pour tâche de démarrer et arrêter les services que le cluster doit rendre hautement disponible.

Un manageur de ressources simple est fourni avec Heartbeat. Il ne peut gérer que deux nœuds et ne détecte pas les erreurs et pannes au niveau des ressources. Une notification automatique par e-mail peut être mise en place pour signaler le déplacement d'une ressource d'un nœud à un autre.

La documentation est abondante mais pas toujours pertinente. Même sur le site officiel, beaucoup d'explications sont obsolètes, la part des choses n'est pas toujours facile à faire.

Heartbeat (v2) + Pacemaker

Un nouveau manageur de ressources a été créé afin de dépasser les limitations de la version précédente (série des $1.X.X$) et ajouter de nouvelles fonctionnalités. Écrit pour *Heartbeat* 2.0.0, il est d'abord fourni avec celui-ci, avant

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

de devenir en 2007 un projet indépendant sous le nom de *Pacemaker*. Le but de *Pacemaker* étant de rendre la gestion des ressources indépendante de la couche cluster et de pouvoir en supporter d'autres cluster comme *OpenAIS*.

Heartbeat et *Pacemaker* n'ont plus de limitation du nombre de nœuds et la haute-disponibilité est effectuée au niveau des ressources (et plus seulement selon l'état général du nœud). Les principales nouvelles fonctionnalités sont les principes de politiques au niveau du placement, de l'ordre et de la collocation des ressources sur les nœuds.

Au niveau de la documentation, le problème est resté le même et la documentation des dernières versions du *CRM* inclus dans *Heartbeat* se croise souvent avec celle de *Pacemaker* sans précision des versions utilisées, incompatibles entre elles sur certains points.

OpenAIS

OpenAIS[2] apparaît comme une bonne alternative à *Heartbeat*. Ses fonctionnalités sont dans l'ensemble assez équivalentes à celles de *Heartbeat* + *Pacemaker*. Même si *OpenAIS* et *Heartbeat* n'ont pas été implémentées autour des mêmes normes pour ce qui est de la couche cluster elle-même, Pacemaker est également utilisé pour gérer les ressources des clusters sous *OpenAIS*.

Plus récent et en développement actif, le manque de documentation est certain et même si l'information est disponible, elle est difficile à trouver.

3.1.2 Synthèse des fonctionnalités

Tab. 3.1 – Synthèse des fonctionnalités : haute-disponibilité

		Heartbeat (v1)	Heartbeat (v2)	OpenAIS
Documentation	Quantité	⋆⋆⋆⋆	⋆⋆⋆	⋆
	Qualité	⋆⋆⋆	⋆	⋆⋆
Nombre de nœuds		2	∞	∞
Détection des pannes	Nœud	oui	oui	oui
	Service	non	oui	oui
Développement	Création	Nov. 1998	Juillet 2005	Juin 2005
	Der. version	Août 2006	Août 2009	Juillet 2009
	Activité	⋆	⋆⋆⋆	⋆⋆⋆⋆

À la vue des fonctionnalités, *Heartbeat (v2)* et *OpenAIS* semblent être deux solutions aux capacités équivalentes et bien supérieures à celles de *Heartbeat (v1)*. La renommée historique de *Heartbeat* nous a donc poussé, au premier abord, vers le choix de *Heartbeat (v2)*.

[2] http://www.openais.org/

3.1.3 Essai de certaines solutions

Heartbeat (v2)

Présentation Notre choix s'est tout d'abord tourné vers *Heatbeat (v2)* et *Pacemaker*. Successeurs de *Heartbeat*, ils sont censés nous apporter son bon niveau de fiabilité et sa stabilité éprouvée. Tout en apportant des fonctionnalités de haut-niveau tenant la comparaison, sur certains points, avec les logiciels propriétaires.

Installation via compilation Les versions de *Heartbeat* présentes dans les dépôts officiels d'*Ubuntu* (y-compris dans la version instable[3]) ne sont pas compatibles avec *Pacemaker* ou non référencés comme stables par les développeurs de *Heartbeat*. Comme nous le verrons par la suite, nous avons testé plusieurs versions de ce couple de programmes. La méthodologie ainsi que les problèmes ont été les mêmes.

Nous avons donc compilé *Heartbeat* et *Pacemaker* depuis les sources. Le processus de compilation est identique pour ces deux programmes et se fait soit via un script nommé *ConfigureMe* qui est en fait une sur-couche des *./configure* et *make* créés par le couple *autoconf/automake* soit directement en utilisant les commandes habituelles induites par les *autotools*.

Afin de ne pas perturber la gestion des paquets de notre distribution, nous avons généré via le script *ConfigureMe* les paquets pour *Heartbeat* et *Pacemaker* ainsi que des paquets censés combler des dépendances provoquées par le fait de vouloir rester compatible avec les versions antérieures. Cette compilation se passe sans problème mis-à-part que *Heartbeat* doit être compilé et installé avant la compilation de *Pacemaker*.

Il est alors possible d'installer les paquets que nous venons de générer de manière classique. L'installation ne pose alors qu'un seul problème : le script de post-installation tente de lancer le démon *Heartbeat* alors que son fichier de configuration principal n'existe pas encore, ce qui renvoie une erreur, ne permet donc pas de terminer l'installation correctement et bloque par la même occasion la commande de désinstallation. Dans ces conditions de test, la méthode pour contourner ce problème a été de créer à la main ce fichier de configuration vide.

Installation via les paquets officiels Des paquets officiels sont fournis via l'openSUSE Build Service[4]. L'installation se fait de manière classique mais pose des problèmes similaires à ceux que nous avons eu avec les paquets générés par nos soins.

[3] Au moment de l'écriture, *Karmic*, future *Ubuntu* 9.10

[4] http://software.opensuse.org/download/server:/ha-clustering

Utilisation Des procédures d'installation, de configuration et d'utilisation de base ont été créées mais ne sont pas fournies avec ce rapport. Celles-ci se rapportent aux versions de développement disponibles au moment de l'écriture. Le choix de baser ces procédures sur des versions non stables des programmes vient du fait que l'intégration de ces outils dans notre architecture s'est avérée trop instable et anarchique, quelles que soient les versions testées. Selon les documentations suivies, même si celles-ci viennent des sites officiels, les éléments de configuration diffèrent et les interfaces des outils sont différentes. La version de développement s'est avérée la plus proche des documents officiels[5] et ayant les outils dont l'ergonomie est la plus poussée (et donc proche de l'objectif final recherché).

Conclusion L'utilisation d'Heartbeat avec *Pacemaker* n'a donc pas été retenue, et ce pour plusieurs raisons :

- les différentes versions sont parfois considérées soit stables, soit de développement selon la page du site officiel consultée,
- la mise en place parfois hasardeuse (Est-ce que cette option va fonctionner avec cette version ? Ha, oui, mais écrite en minuscule ce coup-ci !. . .),
- l'instabilité obtenue est loin de ce que l'on attend d'un système de haute disponibilité.

Cependant, si son utilisation en production dans notre architecture n'est pas possible pour le moment, les perspectives entrevues par cette solution sont assez importantes pour être surveillées en vue d'un fonctionnement stable.

3.2 Réplication des Données

3.2.1 Les différentes solutions

FAM / Gamin

File Alteration Monitor (*FAM*)[6] est un démon dont le but est de prévenir les applications dès que certains fichiers sont créés, modifiés, supprimés. Son développement s'est arrêté en 2003. *Gamin*[7] est son successeur, il reste totalement compatible avec *FAM* tout en l'améliorant.

Il serait possible d'écrire un script (en *PERL* par exemple) pour l'utiliser conjointement à scp ou rsync pour répliquer les données sur la deuxième machine. Cependant, l'écriture d'un tel script demanderait beaucoup de travail pour le rendre robuste et pour qu'il réponde à tout ce dont nous avons besoin.

[5] http://www.clusterlabs.org/mediawiki/images/f/fb/Configuration_Explained.pdf

[6] http://oss.sgi.com/projects/fam/

[7] http://www.gnome.org/~veillard/gamin/overview.html

rsync

rsync[8] (remote synchronization) est un logiciel libre de synchronisation de fichiers. La synchronisation est unidirectionnelle et asynchrone. Il utilise un algorithme de recherche de somme de contrôle afin de ne transférer que les différences entre la source et la destination. Il ne possède pas de système de notification des modifications sur les fichiers et doit donc être lancé à intervalles réguliers.

DRBD

DRBD[9] peut être considéré comme du *raid-1* sur le réseau. Il fonctionne selon le principe maître/esclave afin de répliquer bit-à-bit les données d'un périphérique block sur un autre. Il peut fonctionner selon différents modes dont un où les données écrites sur le maître ne sont considérées comme effectivement écrites qu'une fois que celui-ci a été notifié de l'écriture sur l'esclave. *DRBD* possède également des fonctionnalités d'authentification, de gestion de la bande passante, de récupération automatique, de vérification des données,etc. ainsi qu'une intégration poussée avec *Heartbeat*.

3.2.2 Synthèse des fonctionnalités

TAB. 3.2 – Synthèse des fonctionnalités : réplication

	FAM/Gamin	scp	rsync	DRBD
Surveillance temps-réel	oui	non	non	oui
Copie asynchrone	-	oui	oui	oui
Copie synchrone	-	non	non	oui
Copie uniquement des modifications	-	non	oui	oui
Synchronisation	-	non	oui	oui

À la vue des fonctionnalités offertes par ces différents logiciels, seul *DRBD* répond à l'ensemble de nos besoins sans avoir à "réinventer la roue". Notre choix s'est donc porté vers celui-ci.

[8] http://samba.anu.edu.au/rsync/
[9] http://www.drbd.org/

Chapitre 4

Solutions choisies

4.1 Heartbeat

4.1.1 Détails de fonctionnement

Fonctionnement général

Heartbeat envoie des paquets « battements de cœur »à travers le réseau (ou des ports séries) aux autres instances de *Heartbeat*. Ces paquets sont en quelque sorte des messages « je suis en vie ». *Heartbeat* lui-même agit de façon similaire à un démon *init* mais à l'échelle du cluster. S'assurant que chacun des services qu'il gère fonctionne à chaque instant en se servant des mécanismes de démarrage et redémarrage de *init*.

Quand les « battements de cœur »ne sont plus reçus, le nœud concerné est considéré comme mort et chaque service (ou ressource) qu'il héberge est basculé (*fail-over*) vers un autre nœud. Cette supposition de la mort d'un nœud peut être assurée d'être vraie en intégrant un mécanisme comme *STONITH* (Shoot The Other Node In The Head) ou un « chien de garde »(Watchdog).

Heartbeat peut également surveiller des routeurs et des switches comme s'ils étaient membres du cluster en utilisant ses directives *ping* ou *ping_group*. En combinant ceci avec le programme *ipfail*, il est possible d'effectuer un basculement quand la connexion au réseau est compromise.

Communication

Heartbeat peut communiquer en utilisant une combinaison de broadcast UDP, de multicast UDP, d'unicast UDP et de communication série. La communication peut être compressée et authentifiée. Cette authentification peut aller de la simple somme de contrôle CRC au chiffrement SHA1. Par contre pour les mécanismes à clé, il n'y a pas de protocole d'échange et celle-ci doit donc être propagée manuellement.

4.1.2 Choix des paramètres

La configuration de *Heartbeat* se fait via trois fichiers :
- `/etc/ha.d/authkeys` pour choisir et gérer le mécanisme d'authentification entre les nœuds,
- `/etc/ha.d/ha.cf` pour la configuration générale,
- `/etc/ha.d/haresources` pour la configuration des ressources.

Paramètres d'authentification

Deux lignes sont nécessaires au minimum. La première contient le mot-clé `auth` suivi d'un numéro d'identification compris entre 1 et 15. Les lignes suivantes correspondent aux méthodes potentiellement employées et sont composées d'un numéro d'identification (identique à celui de la première ligne pour la méthode choisie), du nom de la méthode (md5, sha1 ou crc) et de la clé si la méthode choisie est md5 ou sha1.

Paramètres de Heartbeat

En ce qui concerne la communication, les paramètres qui nous intéressent sont `autojoin` que l'on fixe à `none` afin que seules les machines authentifiées puissent se joindre au cluster et la description des liens réseaux. Afin de ne pas faire « trop de bruit »sur le réseau, nous avons choisi une communication unicast. Il faut donc énumérer chaque interface de la façon suivante : `ucast` *`device @IP`*.

Viennent ensuite les paramètres du « battement de cœur »(ce sont des durées exprimées en seconde) :
- `warntime` *`4`*, temps au bout duquel on considère que le retard d'un battement est trop long,
- `deadtime` *`10`*, délai de non-réception de battement pour considérer un nœud comme mort,
- `initdead` *`60`*, temps avant de considérer un nœud comme mort au démarrage de *Heartbeat*,
- `keepalive` *`2`*, intervalle entre les battements.

Ces temps ont été choisis selon les recommandations des développeurs de *Heartbeat*. `warntime` doit être égal au double de `keepalive` et `deadtime` de deux à quatre fois `warntime`. Si l'on veut exprimer des durées inférieures à une seconde, il est possible de lui adjoindre le suffixe *ms* (ex. : `500ms`).

Et enfin la liste des nœuds du cluster : `node` *`hostname...`*.

Paramétrage de la ressource

Chaque ligne du fichier `haressources` correspond à une ressource, elle même composée de différents services qui seront démarrés de gauche à droite et arrêtés

de droite à gauche. Chaque description de ressource commence ainsi pas le nom d'hôte de la machine qui doit l'héberger en priorité suivi de la liste (ordonnée, donc) de ses services. Notre ressource sera donc composée comme suit :

1. `drbddisk::r0` → on veut une ressource *DRBD* nommée *r0*.
2. `Filesystem::/dev/drbd0::/data::ext3` → on monte un système de fichier de type *ext3* de */dev/drbd0* sur */data* (voir la configuration de *DRBD* section 4.2.2, page 20).
3. `130.79.129.147` → on associe une adresse ip virtuelle à la machine hébergeant la ressource.
4. `irodsservice` → on lance un service classique, ici *iRODS*, via son script de démarrage contenu dans `/etc/init.d`.

4.2 DRBD

4.2.1 Détails de fonctionnement

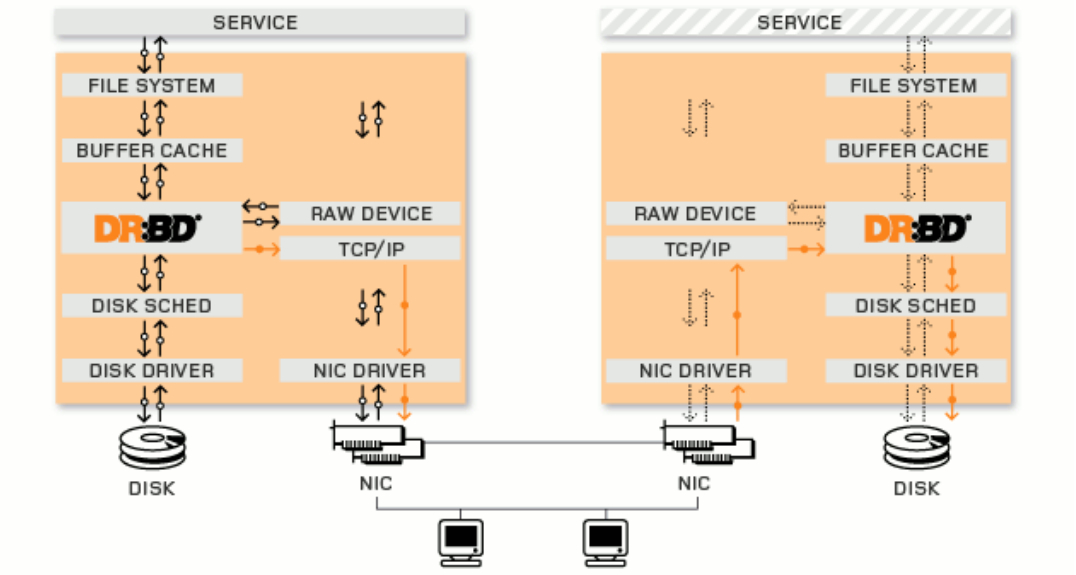


FIG. 4.1 – Schéma de principe de DRBD

Réplication

DRBD fonctionne par dessus des périphériques block, c'est-à-dire des partitions de disques durs ou des volumes logiques LVM. Il réplique chaque block de données écrit sur le disque sur l'autre nœud. DRBD fournit une bonne latitude d'usage, il peut fonctionner de manière totalement synchrone pour une

utilisation en haute disponibilité, de manière asynchrone pour de la réplication à longue distance ou bien encore entre les deux.

Accessibilité des données

Du fait de la conception des systèmes de fichiers, il n'est pas possible d'accéder depuis deux machines à un disque virtuellement partagé. Seule la machine primaire peut accéder à celui-ci. Cependant, si le système de fichiers le permet (GFS, OCFS2), il est possible que deux machines accèdent à celui-ci. On peut alors utiliser le mode primaire-primaire de DRBD.

En cas de panne

Après la panne d'un nœud, DRBD resynchronise le nœud temporairement indisponible en arrière plan, sans interférer avec les services utilisés.

En cas de problème du réseau de réplication, la connexion est rétablie et la resynchronisation se passe de la même manière qu'en cas d'indisponibilité temporaire d'un nœud.

En cas de problème du sous-système de stockage du nœud actif, DRBD peut masquer ces erreurs, le service continuant de fonctionner normalement. Il est également possible de profiter des mécanismes d'échange à chaud (Hot-Swap) si le matériel le permet.

Après une panne de tous les liens réseau, DRBD met en place plusieurs méthodes, automatiques ou manuelles, de résolution des *split brain* (cas où les deux machines veulent être primaire alors qu'elles ne sont plus connectées entre elles).

4.2.2 Choix des paramètres

La configuration de *DRBD* s'effectue via le fichier `/etc/drbd.conf`. Ce fichier se découpe en plusieurs sections :

- les paramètres globaux (directive *global*),
- les paramètres communs aux différentes ressources (directive *common*),
- les paramètres spécifiques à chaque ressource (directives *resource* suivie d'un nom de ressource).

Paramètres globaux

Le seul paramètre qui nous est utile dans cette section est `usage-count` que nous positionnons à `no` afin de ne pas envoyer de statistiques d'utilisation aux développeurs de *DRBD*.

Paramètres communs

C'est ici que nous paramétrons le protocole utilisé : `protocol C` qui correspond au protocole synchrone. Les données en écriture sur le primaire ne sont considérées comme écrites que lorsque l'on a reçu l'acquittement de bonne écriture sur le secondaire.

La sous-directive *syncer* regroupe les paramètres liés à la synchronisation des données. La seule option qui nous intéresse est le réglage de la bande passante allouée à cette synchronisation. Étant donné que le même lien réseau est utilisé pour la communication avec les autres services des serveurs, les développeurs de *DRBD* recommandent de limiter la bande passante allouée à 1/3 de la bande passante totale. Notre réseau est en gigabit donc le tiers d'un gigabit ramené en mégaoctet nous donne 41Mo. Nous réglons donc l'option `rate` à `41M`.

La sous-directive *handlers* permet de modifier ou préciser la commande à exécuter en cas d'appel de ces *handlers*. Nous allons utiliser les *handlers* suivants :

- `outdate-peer "/usr/lib/heartbeat/drbd-peer-outdater -t 5"` afin d'exécuter le script `drbd-peer-outdater` pour signaler que les données d'un pair ne sont plus à jour.
- `pri-lost "echo pri-lost. Have a look at the log files. | mail -s 'DRBD Alert' root"` afin d'envoyer un mail à l'administrateur du serveur si ce dernier perd sa qualité de primaire.

Paramètres par ressource

La sous-directive *startup* contient les options concernant le démarrage de *DRBD*. Les options qui nous intéressent sont `wfc-timeout` et `degr-wfc-timeout`. *wfc* signifie *wait for connection* et *degr*, dégradé. Ces deux options permettent donc de préciser le temps pendant lequel *DRBD* va attendre une connexion de la part d'un paire avant de se considérer comme tout seul (mode *stand-alone*), respectivement s'il était dans un état normal ou anormal avant son arrêt/redémarrage.

La sous-directive *disk* traite des problèmes liés au matériel. L'option `on-io-error detach` demande à *DRBD* de détacher la ressource concernée, c'est-à-dire de ne plus la prendre en compte sur la machine concernée. L'option `fencing dont-care`, où *fencing* signifie les mesures préventives à effectuer pour éviter les situations de *split- brain*, indique à *DRBD* de ne rien faire dans ce cas. Étant donné qu'il n'y a pas d'autre moyen que le principal pour faire communiquer les deux nœuds, si le réseau est hors-service, aucune communication entre les deux machines n'est possible.

La sous-directive *net* permet de régler plus finement *DRBD*. Les options `after-sb-`*`[0-2]`*`pri` indiquent quoi faire après un *split-brain* en commençant par `after-sb-0pri`. La valeur `discard-older-primary` signifie à *DRBD* de se

synchroniser sur le dernier nœud à avoir été primaire et `call-pri-lost-after-sb` demande de toujours exécuter la méthode indiquée par `after-sb-0pri` puis si le secondaire a les bonnes données, appeler le *handler pri-lost-after-sb* sur le primaire courant.

La sous-directive *on* nom-de-machine permet la configuration de la ressource sur la machine donnée. L'option `device /dev/drbd0` indique que la ressource sera accessible comme le périphérique block *drbd0*, `disk /dev/sdb3` indique le périphérique utilisé pour stocker les données répliquées, `address @IP:PORT` indique l'adresse IP et le port utilisés pour la communication entre les nœuds et `flexible-meta-disk internal` indique que les métadonnées de *DRBD* se trouveront au même endroit que les données elles-mêmes.

4.3 Le système de gestion de fichiers

4.3.1 Comparatif de FS

Les comparatifs de performance de Systèmes de Gestion de Fichiers que l'on trouve sur Internet sont peu nombreux, anciens pour la plupart, souvent imprécis sur les conditions des tests et répondent à des problématiques qui ne sont pas les nôtres.

Nous avons donc mis en place une batterie de tests qui répond à nos besoins, c'est-à-dire :

- Nécessité de la journalisation.
- Stabilité et fiabilité → FS éprouvés.
- Taille des fichiers très hétérogène.
- Arborescence peu profonde.

Nous avons donc retenu ext3, xfs et reiserfs[1]. Sur lesquels nous avons effectué les tests suivants :

1. copie depuis un autre disque,
2. copie dans un répertoire différent,
3. parcours de l'arborescence,
4. suppression des fichiers.

Les jeux de test sont :

- un répertoire contenant 3 fichiers de 4Go,
- un répertoire contenant 48 sous-répertoires ayant chacun 64 fichiers de 4Mo,
- un répertoire contenant 128 sous-répertoires ayant chacun 1024 fichiers de 16Ko.

[1] les plus récents systèmes de fichiers que sont *ext4* et *btrfs* n'ont pas pu être testé car la distribution (*Ubuntu 8.04*) utilisée à ce moment là avait un noyau trop ancien.

Les indices de performance retenus pour chaque opération sont :
- le temps total,
- le temps utilisateur et le temps système,
- le taux d'utilisation du processeur.

Nous avons donc écrit trois scripts bash : un premier (voir B.1 page 45) pour la génération des jeux de test, un deuxième (voir B.2 page 46) pour les tests à proprement parler et un dernier (voir B.3 page 47) pour l'analyse et la synthèse des résultats.

4.3.2 Choix du FS

Les résultats synthétiques obtenus (voir C.1 page 51) nous ont permis de tracer des graphes de résultats nous permettant de mieux appréhender ces derniers.

FIG. 4.2 – Résultats de la copie depuis un autre disque

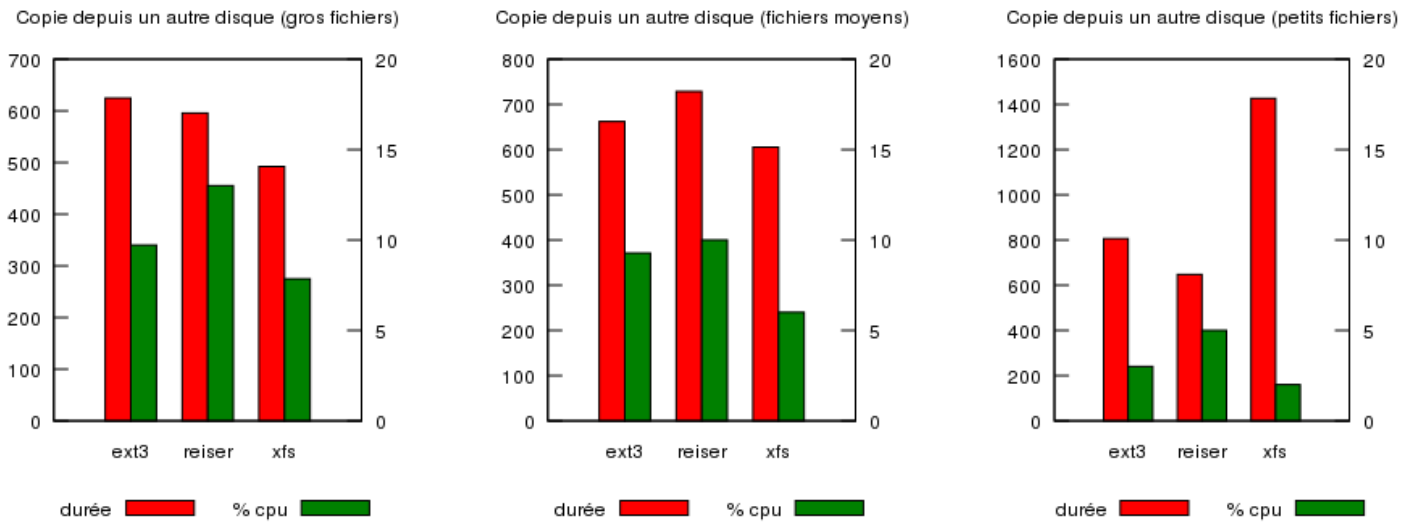


FIG. 4.3 – Résultats de la copie dans un répertoire différent

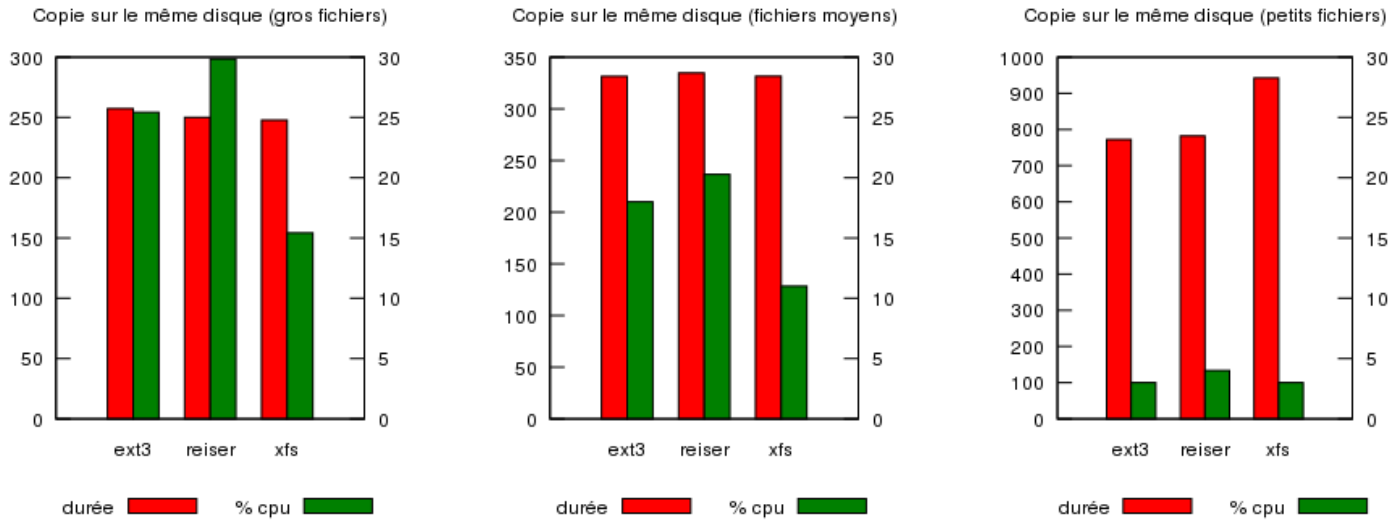


FIG. 4.4 – Résultats du parcours de l'arborescence

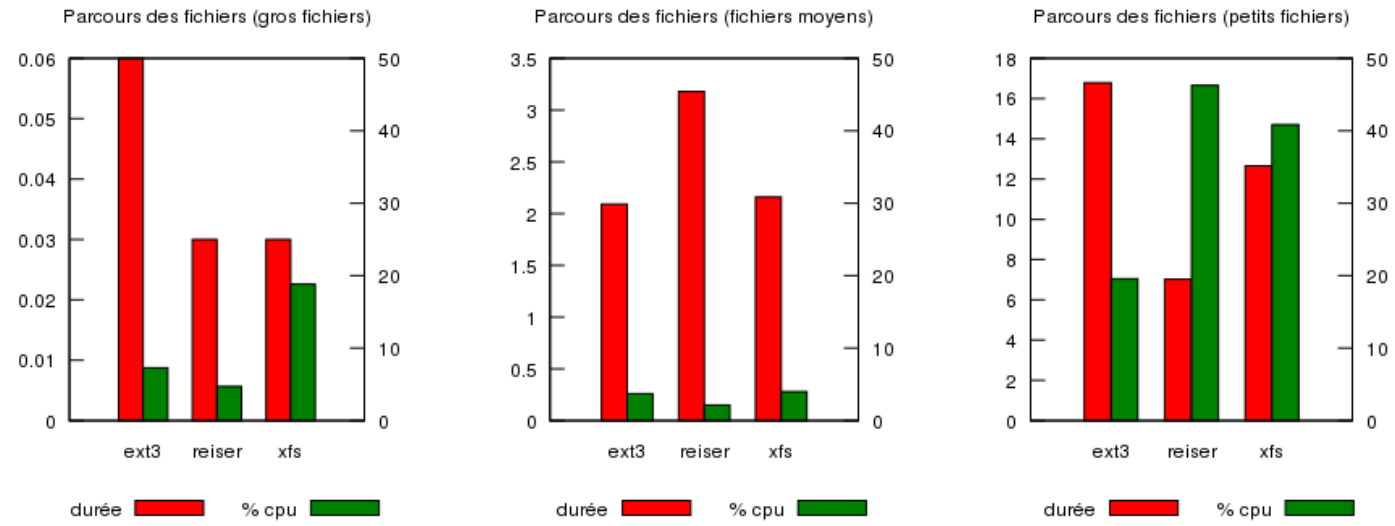
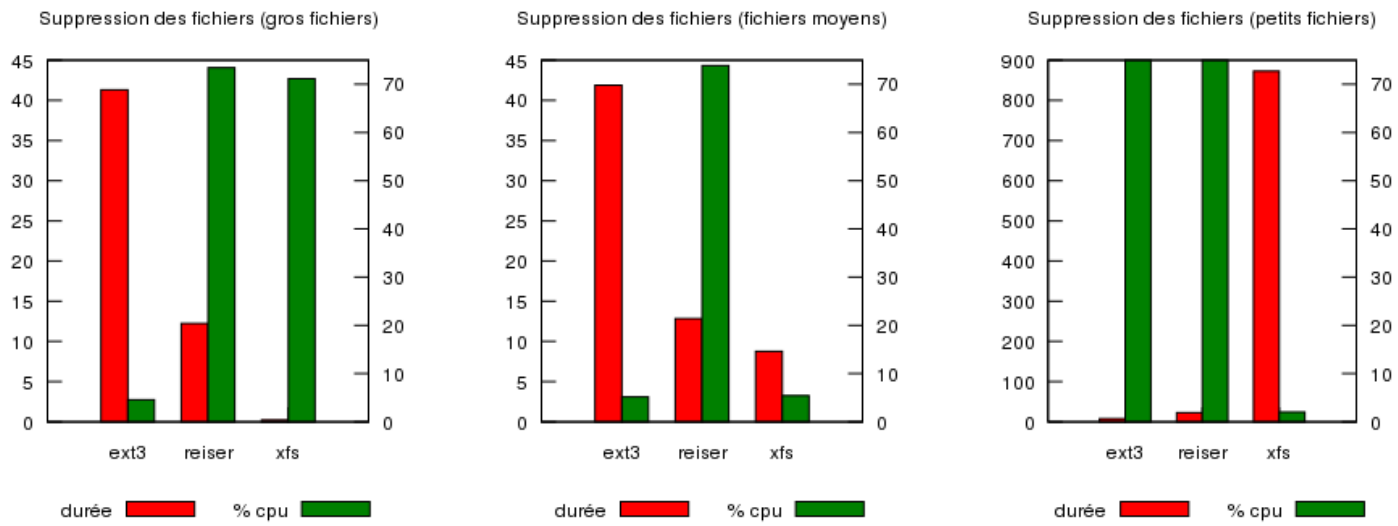


FIG. 4.5 – Résultats de la suppression des fichiers

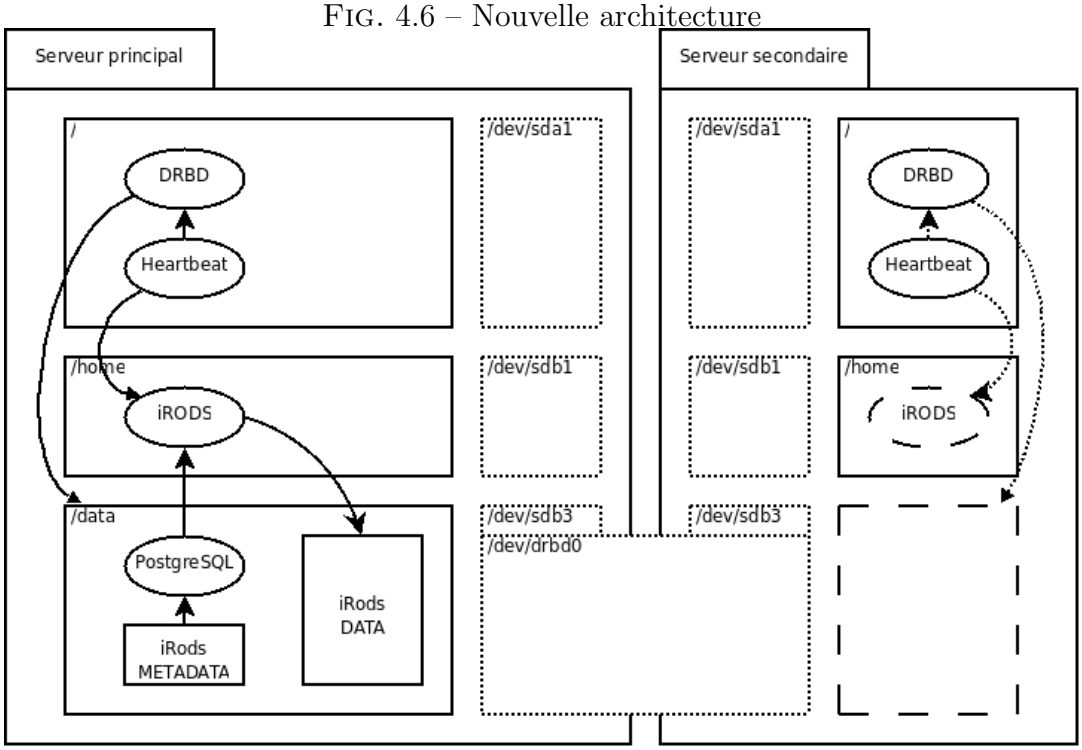


Aucun de ces systèmes de fichier ne semble sortir du lot. Cependant, *ext3* nous apparaît comme étant le plus versatile. Son utilisation par défaut dans les distributions linux ne nous paraît donc pas usurpée et c'est donc le choix que nous faisons pour notre architecture.

4.4 Agencement des applications et données

Dans l'optique de maintenabilité de l'architecture et afin d'être certains de la réplication complète des données, nous avons décidé de modifier cette architecture pour la rendre plus simple. Par simple, nous entendons plus facile à mettre en œuvre et à comprendre afin que les problèmes pouvant survenir soient plus facilement identifiables et réparables ou contournables. Le schéma simplifié de la nouvelle architecture est visible sur la figure 4.6 page 26.

FIG. 4.6 – Nouvelle architecture



Ainsi, contrairement à l'ancienne architecture, toutes les données et métadonnées sont répliquées par *DRBD* ce qui nous assure d'une part la réplication complète de toutes les données et d'autre part, conjointement au reste de l'architecture, le fonctionnement identique quelque soit le serveur.

Chapitre 5

Mise en œuvre

5.1 Architecture mise en place

5.1.1 Architecture matérielle

Deux serveurs sont à notre disposition pour les différents tests (voir tableau 5.1 page 27) ainsi que deux autres pour le prototype (voir tableau 5.2 page 28). Tous les tests que nous avons effectués l'ont été sur les serveurs de test. Les serveurs pour le prototype serviront, en tout cas à notre niveau, à « la mise en production » afin de servir de plateforme de base au reste du projet.

Tab. 5.1 – Matériel de test

	vm-server1	*vm-server2*
Type	Primaire	Secondaire
Processeur	bi-XEON 3 GHz	
RAM	512 Mo	
Partitionnement		
hd1 : /	74 Go	
hd2 : swap	512 Mo	
hd2 : /home	1 Go	
hd2 : /dev/sdb3 (drdb)	232 Go	
Réseau		
Vitesse	1 GB	
IP	130.79.129.145	130.79.129.146
MAC	`00:30:48:2a:4d:00`	`00:30:48:29:83:fc`

5.1.2 Architecture logicielle

Les deux serveurs ont pour système d'exploitation *Ubuntu 9.04*, les paquets installés sont ceux fournis par l'installation de base, ainsi que le serveur *sshd*.

Tab. 5.2 – Matériel du prototype

	Serveur 1	*Serveur 2*
Model	HP Prolian	
Processeur	Intel Quadcore	
RAM	8 Go	
Disques systèmes	6×170 Go (raid0) soit 170 Go	
Disques données	6×1 To (raid5) soit 4,7 To	
Autre	Double alimentation	

5.2 Procédures d'installation

Installation : Heartbeat et DRBD

Attention, toutes les commandes suivantes sont à effectuer en tant qu'administrateur.

5.2.1 Installation de Heartbeat

Installation de *Heartbeat* :

```
apt-get install heartbeat
```

Ceci nous crée également un groupe `haclient` et un utilisateur `hacluster`.

5.2.2 Installation de DRBD

Avant d'installer les outils liés à *DRBD*, il faut d'abord charger le module noyau associé.

```
modprobe drbd
```

Nous nous assurons également que celui-ci sera automatiquement chargé au prochain démarrage.

```
echo "drbd" >> /etc/modules
```

Nous pouvons alors installer les outils `DRBD` :

```
apt-get install drbd8-utils
```

Afin que les outils (`drbdsetup` et `drbdmeta`) fonctionnent correctement avec `Heartbeat`, il faut modifier les attributs de ceux-ci de telle sorte qu'ils appartiennent au groupe `haclient`. Il faut également interdire l'exécution aux utilisateurs autres que le propriétaire ou n'appartenant pas au groupe et permettre l'exécution avec les droits *root* (le propriétaire) :

```
chgrp haclient /sbin/drbdsetup
chmod o-x /sbin/drbdsetup
chmod u+s /sbin/drbdsetup

chgrp haclient /sbin/drbdmeta
chmod o-x /sbin/drbdmeta
chmod u+s /sbin/drbdmeta
```

Démarrage automatique

Afin de s'assurer du démarrage correct (et surtout dans le bon ordre) de *Heartbeat* et *DRBD*, on doit s'assurer que le réseau soit déjà démarré et que *Heartbeat* succède à *DRBD* :

```
update-rc.d -f heartbeat remove
update-rc.d -f drbd remove
update-rc.d heartbeat start 71 2 3 4 5 . stop 71 0 1 6 .
update-rc.d drbd start 70 2 3 4 5 . stop 70 0 1 6 .
```

Configuration

Attention, toutes les commandes suivantes sont à effectuer en tant qu'administrateur.

5.2.3 Configurations préliminaires

- Afin de s'assurer que le réseau démarre bien avec le service *Network Service*, le fichier */etc/network/interfaces* doit être renseigné à propos des interfaces que nous utiliserons par la suite. Par exemple, pour avoir le *loopback* et que l'interface *eth0* obtienne son adresse IP via le service *dhcp* (Voir Annexe A.1 page 41).
- Pour se simplifier la vie (et éviter d'éventuelles erreurs de frappe par la suite), on peut ajouter, sur chaque machine, le nom et l'adresse IP de l'autre dans `/etc/hosts`. Par exemple, sur le serveur principal :
  `echo -e "130.79.129.146\tvm-server2" >> /etc/hosts`
- Il est (plus que) recommandé que les horloges des serveurs soient synchronisées entre elles. Selon la dérive relative des horloges, leur synchronisation avec un serveur de temps doit se faire plus ou moins souvent. Nous allons donc rajouter une tâche périodique :
  `crontab -e`
  Nous pouvons alors ajouter notre tâche à proprement parler, ici à 21h43 tous les jours du mois, tous les mois de l'année et tous les jours de la semaine nous exécuterons la commande `ntpdate` :

```
# m h  dom mon dow   command
43 21   *   *   *    ntpdate ntp.u-strasbg.fr ntp.ubuntu.com
```

5.2.4 Heartbeat

Comme nous l'avons vu au moment du choix des paramètres de *Heartbeat* (voir section 4.1.2 page 18), la configuration se fait via trois fichiers : les ressources (`/etc/ha.d/haresources` voir listing A.2 page 41), les paramètres (`/etc/ha.d/ha.cf` voir listing A.3 page 41) et le fichier `/etc/ha.d/authkeys`. Nous générons ce dernier de façon à ce que `Heartbeat` utilise un chiffrement *sha1* avec une clé humainement introuvable. Sur le serveur principal :

```
echo "auth 1" > /etc/ha.d/authkeys
echo -n "1 sha1 " >> /etc/ha.d/authkeys
dd if=/dev/urandom bs=512 count=1 | openssl sha1 >> /etc/ha.d/authkeys
chmod 600 /etc/ha.d/authkeys
```

Ces trois fichiers doivent être identiques sur les deux machines donc nous copions ces fichiers vers le serveur secondaire :

```
scp /etc/ha.d/authkeys vm-server2:/etc/ha.d/
scp /etc/ha.d/ha.cf vm-server2:/etc/ha.d/
scp /etc/ha.d/haresources vm-server2:/etc/ha.d/
```

5.2.5 DRBD

Configuration

La configuration de *DRBD* se fait via le fichier `/etc/drbd.conf` (voir listing A.4 page 42) dont le choix des paramètres a été effectué section 4.2.2 page 20.

Copions ce fichier vers le serveur secondaire :

```
scp /etc/drbd.conf vm-server2:/etc/drbd.conf
```

Initialisation de DRBD

Attention, ceci est à effectuer sur les deux machines.

Avant d'initialiser notre partition *DRBD*, il faut effacer les éventuelles données qu'elle contient (tout du moins le début) :

```
shred -zvf -n0 -s 1G /dev/sdb3
```

Il est alors possible de générer les méta-données :

```
drbdadm create-md r0
```

On peut alors activer `DRBD` :

```
drbdadm up r0
```

A ce stade, nos deux serveurs sont considérés comme secondaires. Avant de passer à la configuration d'un primaire et donc à la synchronisation initiale, nous passons le paramètre `rate` de la section de configuration `syncer` à 125M (pour une liaison 1 Gigabit) afin d'utiliser l'intégralité de la bande passante. Ne pas oublier de remettre la bonne valeur dès la synchronisation terminée. Pour prendre en compte le changement dans la configuration, il faut lancer la commande suivante :

```
drbdadm adjust r0
```

Configuration du Serveur Principal

Attention, cette étape n'est à effectuer que sur le serveur primaire :

```
drbdadm -- --overwrite-data-of-peer primary r0
```

La synchronisation des deux serveurs se passe en arrière plan. Il est possible d'en suivre le déroulement via la commande :

```
watch -n1 cat /proc/drbd
```

sur l'un des deux serveurs. Une fois la synchronisation terminée, ne pas oublier de rétablir le paramètre `rate` à l'état normal. A partir de maintenant, toute modification sur la partition `/dev/drbd0` sur le serveur principal sera répercutéesur le serveur secondaire.

Nous pouvons maintenant formater notre partition `/dev/drbd0`. Afin d'utiliser tout l'espace disponible, il faut préciser que l'on ne veut pas réserver d'espace disque pour `root`. Par défaut, ce sont 5% qui lui sont réservés et donc sur une partition de 100 Go, après de savants calculs, on peut voir que 5 Go seraient réservés et plus ou moins perdus pour nous. Donc :

```
mkfs.ext3 -m0 /dev/drbd0
```

5.2.6 Installation / Configuration de iRODS

Outils nécessaires

Attention, toutes les commandes suivantes sont à effectuer en tant que *root*. Pour installer et faire fonctionner *iRODS*, nous avons besoin de quelques outils supplémentaires. L'installation se faisant sur le serveur primaire, nous installons sur celui-ci les paquets nécessaires à la compilation de *PostgreSQL* par exemple :

```
apt-get install build-essential
```

Par contre, nous avons besoin sur les deux machines de *Perl* que nous installons donc sur les deux machines :

```
apt-get install perl
```

Préliminaires

Attention, toutes les commandes suivantes sont à effectuer en tant qu'administrateur.

Il faut créer sur chaque machine le répertoire `/data`.

```
mkdir /data
```

Le cluster n'étant pas encore disponible, nous allons effectuer l'installation sur le serveur primaire. Il faut donc monter notre partition et permettre à tout utilisateur d'écrire dans ce répertoire, d'où le *sticky bit*. C'est donc `/dev/drbd0` sur cette machine :

```
mount -t ext3 /dev/drbd0 /data
chmod +t /data
```

Installation

Attention, toutes les commandes suivantes sont à effectuer en tant qu'utilisateur normal. Après avoir obtenu la version souhaitée de *iRODS*, on la place (`cp`, `scp`, `wget`,...) dans le répertoire `/data`. On la décompresse puis on supprime l'archive :

```
cd /data
tar xzf irods2.0.1.tgz
rm irods2.0.1.tgz
```

Nous avons maintenant un répertoire *iRODS*. Plaçons-nous dans celui-ci et démarrons l'installation :

```
cd iRODS
./irodssetup
```

Le script d'installation est plutôt explicite et les paramètres par défaut conviendront généralement. Un point à ne pas manquer (de toute façon, il n'a pas de valeur par défaut) est le choix du répertoire de postgres que nous plaçons dans `/data/postgres`, donc également sur la partition en réplication.

Une fois *iRODS* installé et configuré, on peut en tester le bon fonctionnement, d'abord en le démarrant puis en lui demandant de se tester :

```
./irodsctl start
./irodsctl test
```

Cependant, *iRODS* est pour le moment lié à la machine sur laquelle il a été installé, et plus particulièrement à son *hostname*. Afin qu'il puisse s'exécuter aussi bien sur le serveur primaire que secondaire, il faut modifier sa configuration pour qu'elle se réfère à *localhost* plutôt qu'à *vm-server1* dans notre cas :

- dans le fichier `/data/iRODS/config/irods.config` :
  `$DATABASE_HOST = 'localhost';`
- dans le fichier `/data/iRODS/server/config/server.config` :
  `icatHost localhost`
- dans le fichier `/data/postgres/pgsql/etc/odbc.ini` :
  `Servername=localhost`

Script de démarrage

Afin de pouvoir gérer l'exécution de *iRODS* par *Heartbeat*, nous avons écrit un script de démarrage nommé `irodsservice` à placer dans le répertoire `/etc/init.d/` (voir annexe B.4 page 49) et son fichier de configuration nommé également `irodsservice` mais à placer cette fois-ci dans `/etc/default/` (voir annexe B.5 page 50). Les deux fichiers précédents sont à placer (en tant que root) sur chacune des machines.

Il faut également rendre le script exécutable (en tant que *root*) :

```
chmod +x /etc/init.d/irodsservice
```

5.2.7 Démarrage de iRODS

Pour vérifier que cela fonctionne correctement, nous pouvons démarrer *iRODS* sur le serveur principal. Il suffit d'exécuter la commande suivante :

```
/etc/init.d/irodsservice start
```

5.3 Procédures de test

Les procédures de test que nous avons définies ne sont pas des procédures automatisées mais un ensemble de cas d'utilisation et de simulation de problèmes pendant lesquels nous surveillons la réaction de notre architecture.

5.3.1 Utilisation normale

En utilisation normale, la machine principale écrit sur *iRODS* (en local donc), les données sont répliquées par *DRBD*. Nous avons alors testé les vitesses de lecture et d'écriture sur *iRODS* en utilisant les commandes `iget` et `iput` via un simple script (voir listing B.6 page 50) servant à calculer des valeurs moyennes sur des exécutions consécutives.

Vitesses de lecture et d'écriture

TAB. 5.3 – Vitesses de lecture et d'écriture (avec réplication)

	Lecture	Écriture
fichier de 1 Mo	18 Mo/s	13 Mo/s
fichier de 10 Mo	50 Mo/s	20 Mo/s
fichier de 100 Mo	49 Mo/s	26 Mo/s

5.3.2 Défaillances hors transfert

Par défaillance hors transfert nous entendons que les défaillances que nous simulons ne se passent pas pendant l'écriture de données sur *iRODS*.

Test 1 - Performances

Le serveur secondaire est arrêté. *iRODS* se trouve sur le serveur principal et il n'y a donc pas de réplication même s'il est en attente du retour du serveur secondaire. Nous utilisons le même script et les mêmes fichiers que pour le test d'utilisation normale.

TAB. 5.4 – Vitesses de lecture et d'écriture (sans réplication)

	Lecture	Écriture
fichier de 1 Mo	18 Mo/s	23 Mo/s
fichier de 10 Mo	50 Mo/s	32 Mo/s
fichier de 100 Mo	49 Mo/s	27 Mo/s

Test 2 - Temps de synchronisation

Le serveur secondaire est arrêté. *iRODS* se trouve sur le serveur principal et il n'y a donc pas de réplication. Le but ici est de voir la relation entre la quantité de données écrite en l'absence du serveur secondaire et le temps de synchronisation à son retour. Dans le tableau 5.5 page 35, nous indiquons également la quantité de données signalée par *DRBD* comme étant synchronisée.

La différence au niveau de la taille des fichiers envoyés et la quantité de données à synchroniser est due à l'espace pris par *iRODS* pour ses métadonnées et l'écriture des fichiers de *log*.

A cause des arrondis à la seconde que nous avons dans les fichiers de *log*, les taux de transferts ne sont pas vraiment exacts. Cependant, pour les fichiers de 50 Mo et surtout ceux de 100 et 500 Mo, on voit bien que la limite de la bande passante fixée à 41 Mo/s est bien respectée et également bien exploitée.

Tab. 5.5 – Temps de synchronisation en fonction du volume de données

	Taille des données transférées				
	10 Mo	20 Mo	50 Mo	100 Mo	500 Mo
Qté (Ko)	10 420	21 000	51 524	102 752	514 036
Durée (s)	<1	<1	<2	<3	12

Test 3 - Temps de reprise après arrêt

Contrairement aux deux tests précédents, nous arrêtons le serveur principal. D'après les fichiers de `log` et des mesures manuelles nous avons pu nous assurer des timings prévus par la configuration. Quelques secondes supplémentaires sont nécessaires, d'une part pour l'arrêt des services sur la machine primaire puis le démarrage de ceux-ci sur la machine secondaire. De même, au redémarrage du serveur primaire, les services sont immédiatement repris par celui-ci.

Test 4 - Temps de reprise après coupure réseau

Test identique au précédent sauf qu'au lieu d'éteindre le serveur principal, nous débranchons son câble réseau. Le timer pour considérer la machine comme indisponible démarre immédiatement et la machine secondaire reprend la main aussitôt. Le retour à la normal est quasi simultané au retour du réseau modulo le temps de synchronisation vu au test 2.

5.3.3 Défaillances durant transfert

En cas de défaillance pendant le transfert, celui-ci est tout de même interrompu. Cependant, les données déjà transférées ont bien été écrites et la reprise du transfert est possible.

5.4 Résultats des tests / Conclusion

5.5 Procédures d'utilisation

Chapitre 6

Bilan

6.1 Bilan du stage et perspectives

La plupart des problématiques du stage ont été traitées avec succès depuis la remise en question de l'architecture à la mise en place des différentes procédures.

Même si certaines ambitions ont du être revues à la baisse après nous avoir fait perdre un temps certain, la modification de l'architecture a été finalement suffisamment concluante en termes de stabilité et de performance.

Deux perspectives sont donc en vue. À très court terme, la mise en production de l'ensemble des solutions que nous avons choisies. Dans un futur pas forcément très lointain, le remplacement de **Heartbeat** par sa dernière version, une fois celle-ci réellement aboutie. Notre architecture est prête à la recevoir et la procédure d'installation et de configuration de **Heartbeat v2** que nous avons essayée ne devrait pas changer fondamentalement d'ici là.

6.2 Bilan technique

D'un point de vue technique, ce stage m'aura beaucoup apporté. En effet, même si un prototype existait déjà, j'ai été amené à repenser les choses depuis le début. Même si certaines choses ont assez peu évolué, j'ai du me « mettre dedans »afin d'avoir en ma possession la majorité des tenants et aboutissants.

Le sujet en lui-même était également très intéressante car il m'a permis d'avoir une vision assez globale de la notion de haute-disponibilité dont on entendra toujours autant parler dans le futur (avec ou sans sa virtualisation).

6.3 Bilan personnel et professionnel

Sous tous les points de vue, cette expérience à l'Observatoire Astronomique de Strasbourg m'aura été agréable et profitable. Côté personnel, l'accueil et l'ambiance au sein du laboratoire étaient plus que parfaits tout comme le cadre de travail avec ses bâtiments situés entre un jardin botanique et le parc de l'observatoire, qui se situe pourtant en plein cœur de Strasbourg.

Côté professionnel, j'ai appris à mes dépens que même une bonne idée, si elle n'est pas réalisable, doit être abandonnée. Je me suis également rendu compte de la place de l'anglais dans les milieux professionnels. En effet, j'ai pu assister à quelques conférences de l'IVOA pendant lesquelles les intervenants avaient pour seule langue commune l'anglais. De même, l'anglais était le seul moyen de communication avec certains collègues et ce ne sont pas les cours promulgués pendant les études qui nous préparent vraiment à cela (merci aux séries américaines en VO).

Chapitre 7

Conclusion

Ce stage m'a beaucoup apporté, tant sur le plan professionnel que personnel. Il m'a permis de mettre en pratique un certain nombre de connaissances acquises tout au long de mon cursus ainsi que d'appliquer les méthodes de travail et de réflexion assimilées avec la licence professionnelle.

Les problèmes que j'ai pu rencontrer m'ont permis de progresser en me forçant à réfléchir différemment et en me poussant à ne pas agir de façon scolaire. En outre, j'ai particulièrement apprécié le fait d'avoir un vrai projet à traiter dans sa globalité, plutôt qu'une multitude de petites tâches sans réelle cohérence d'autant plus que ce projet à pour fin d'être mis en production.

Enfin, j'ai été très heureux de conclure la dernière ligne droite de mes études dans un endroit aussi sympathique que les personnes y sont intéressantes. J'en garderai de ce fait un excellent souvenir.

Annexe A

Fichiers de configuration

A.1 Configuration générale

Listing A.1 – /etc/network/interfaces

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet dhcp
```

A.2 Heartbeat

Listing A.2 – /etc/ha.d/haresources

```
vm-server1 drbddisk::r0 Filesystem::/dev/drbd0::/data::ext3 \
    130.79.129.147 irodsservice
```

Listing A.3 – /etc/ha.d/ha.cf

```
autojoin none
ucast eth0 130.79.129.145
ucast eth0 130.79.129.146

warntime 5
deadtime 15
initdead 60
keepalive 2

node vm-server1 vm-server2
```

A.3 DRBD

Listing A.4 – /etc/drbd.conf

```
global {
    # Interdire l'envoi de statistiques au serveur http://usage.__
        drbd.org
    usage-count no;
}

common {
  protocol C;

  syncer {
    # Recommandation des developpeurs de DRBD, regler a 1/3 de __
        la bande
    # passante. Ici 1GB/3 => 1000MB/3 => 125M/3 => 41M.
    rate 41M;
  }

  handlers {
    outdate-peer "/usr/lib/heartbeat/drbd-peer-outdater -t 5";
    pri-lost "echo pri-lost. Have a look at the log files. | __
        mail -s 'DRBD Alert' root";
  }
}

resource r0 {

  startup {
    wfc-timeout  60;
    degr-wfc-timeout 30;
  }

  disk {
    on-io-error   detach;
    fencing resource-only;
   }

  net {
    after-sb-0pri discard-older-primary;
    after-sb-1pri call-pri-lost-after-sb;
    after-sb-2pri call-pri-lost-after-sb;
  }

  on vm-server1 {
    device     /dev/drbd0;
    disk       /dev/sdb3;
    address    130.79.129.145:7788;
    flexible-meta-disk  internal;
  }
```

```
  on vm-server2 {
    device      /dev/drbd0;
    disk        /dev/sdb3;
    address     130.79.129.146:7788;
    flexible-meta-disk  internal;
  }
}
```

Annexe B

Scripts

B.1 Test de systèmes de fichiers

Listing B.1 – Génération des jeux de test

```
#!/bin/bash
# Genere les arborescences de test

ROOT=`pwd`

###########
# Test 1 #
#########################################################################
# 3 fichiers de 4Go.                                                    #
#########################################################################

TEST=$ROOT/test1

mkdir $TEST
for (( i=1 ; i<=3 ; i=$i+1 )) ; do
  dd if=/dev/urandom of=$TEST/fichier$i bs=1K count=4M 2> /dev/__
      null
done

###########
# Test 2 #
#########################################################################
# 48 repertoires de 64 fichiers de 4Mo.                                 #
#########################################################################

TEST=$ROOT/test2

mkdir $TEST
for (( i=1 ; i<=48 ; i=$i+1 )) ; do
  DIR=$TEST/dir$i
  echo "mkdir $DIR"
```

```
  mkdir $DIR
  for (( j=1 ; j<=64 ; j=$j+1 )) ; do
    dd if=/dev/urandom of=$DIR/fichier$j bs=1K count=4K 2> /dev/__
        null
  done
done

##########
# Test 3 #
#########################################################################
# 128 repertoires de 1024 fichiers de 16Ko.                             #
#########################################################################

TEST=$ROOT/test3

mkdir $TEST
for (( i=1 ; i<=128 ; i=$i+1 )) ; do
  DIR=$TEST/dir$i
  echo "mkdir $DIR"
  mkdir $DIR
  for (( j=1 ; j<=1024 ; j=$j+1 )) ; do
    dd if=/dev/urandom of=$DIR/fichier$j bs=1K count=16 2> /dev/__
        null
  done
done
```

Listing B.2 – Exécution des tests

```
#!/bin/bash
# Effectue les tests

ROOT=`pwd`

# parametres
DEV=/dev/sdb3
MNT=/media/test
FS="ext3 xfs reiserfs"
TESTS="test1 test2 test3"

for t in $TESTS ; do
  echo "****************"
  echo "* TEST : $t *"
  echo "****************"
  for fs in $FS ; do
    echo -e "\n    ***   $fs   ***\n"
    echo -ne "[$t] Creation du systeme de fichier $fs..."
    if [ "$fs" = "ext3" ] ; then
      /usr/bin/time -a -o $ROOT/bench.$fs.txt -f "%C :\t%e real,__
          %U user, %S sys, %P cpu" mkfs.ext3 $DEV > /dev/null
    else
```

```
      /usr/bin/time -a -o $ROOT/bench.$fs.txt -f "%C :\t%e real,__
          %U user, %S sys, %P cpu" mkfs -t $fs -f $DEV > /dev/__
          null
    fi
    echo -ne "OK\n"

    echo -ne "[$t] Montage du systeme de fichier $fs..."
    /usr/bin/time -a -o $ROOT/bench.$fs.txt -f "%C :\t%e real, %__
       U user, %S sys, %P cpu" mount $DEV $MNT -t $fs > /dev/__
       null
    echo -ne "OK\n"

    echo -ne "[$t] Copie vers $MNT..."
    /usr/bin/time -a -o $ROOT/bench.$fs.txt -f "%C :\t%e real, %__
       U user, %S sys, %P cpu" cp -r $ROOT/$t $MNT > /dev/null
    echo -ne "OK\n"

    mkdir $MNT/new
    echo -ne "[$t] Copie de $MNT vers $MNT/new..."
    /usr/bin/time -a -o $ROOT/bench.$fs.txt -f "%C :\t%e real, %__
       U user, %S sys, %P cpu" cp -r $MNT/$t $MNT/new/$t > /dev/__
       null
    echo -ne "OK"

    echo -ne "[$t] Parcours des donnees..."
    /usr/bin/time -a -o $ROOT/bench.$fs.txt -f "%C :\t%e real, %__
       U user, %S sys, %P cpu" du -sh $MNT/* > /dev/null
    echo -ne "OK\n"

    echo -ne "[$t] Suppression des donnees..."
    /usr/bin/time -a -o $ROOT/bench.$fs.txt -f "%C :\t%e real, %__
       U user, %S sys, %P cpu" rm -rf $MNT/* > /dev/null
    echo -ne "OK\n"

    echo -ne "[$t] Demontage du systeme de fichier $fs..."
    /usr/bin/time -a -o $ROOT/bench.$fs.txt -f "%C :\t%e real, %__
       U user, %S sys, %P cpu" umount $DEV
    echo -ne "OK\n"
  done
done
```

Listing B.3 – Analyse partielle des résultats

```
#!/bin/bash
# analyse les resultats

ROOT=`pwd`
RES=$(/bin/ls bench.*.txt)
oldIFS=$IFS
IFS=$'\n'
```

```
i=1
tot_real=0
tot_user=0
tot_sys=0
tot_cpu=0
for t in $RES ; do
  echo -ne "\n=> RES : $t\n"
  l1=""
  for l in $(cat $t | sort | grep -v non-zero) ; do
    if [ ! "$l1" = "$(echo $l | cut -d':' -f1)" ] ; then
      if [ -n "$l1" ] ; then
        tot_real=$(echo "scale=2;$tot_real/$i" | bc)
   tot_user=$(echo "scale=2;$tot_user/$i" | bc)
   tot_sys=$(echo "scale=2;$tot_sys/$i" | bc)
   tot_cpu=$(echo "scale=2;$tot_cpu/$i" | bc)
   echo "==> $tot_real real, $tot_user user, $tot_sys sys, \
      $tot_cpu cpu"
      fi
      l1=$(echo $l | cut -d':' -f1)
      echo $l1
      i=1
      tot_real=0
      tot_user=0
      tot_sys=0
      tot_cpu=0
    else
      i=$(( i + 1 ))
    fi
    tmp_l=$(echo $l | cut -d':' -f2 | tr -d '\t')
    tmp=$(echo $tmp_l | cut -d' ' -f1)
    tot_real=$(echo "$tot_real + $tmp" | bc)
    tmp=$(echo $tmp_l | cut -d' ' -f3)
    tot_user=$(echo "$tot_user + $tmp" | bc)
    tmp=$(echo $tmp_l | cut -d' ' -f5)
    tot_sys=$(echo "$tot_sys + $tmp" | bc)
    tmp=$(echo $tmp_l | cut -d' ' -f7 | sed "s/\%//")
    tot_cpu=$(echo "$tot_cpu + $tmp" | bc)
  done
  tot_real=$(echo "scale=2;$tot_real/$i" | bc)
  tot_user=$(echo "scale=2;$tot_user/$i" | bc)
  tot_sys=$(echo "scale=2;$tot_sys/$i" | bc)
  tot_cpu=$(echo "scale=2;$tot_cpu/$i" | bc)
  echo "==> $tot_real real, $tot_user user, $tot_sys sys, \
     $tot_cpu cpu"
done

IFS=$oldIFS
```

B.2 iRODS

Listing B.4 – /etc/init.d/irodsservice

```
#!/bin/bash
# /etc/init.d/irodsservice: start and stop the irods

set -e

DEFAULT=/etc/default/irodsservice
SU=/bin/su
IRODSCTL='irodsctl'
IRODSERR='/var/log/irodsservice.err'

not_start() {
  echo "## WARNING ##"
  echo "The irods service won't be started/stopped unless it is __
      configured"

  if [ "$1" != "stop" ] ; then
    echo ""
    echo "Please configure it and then edit /etc/default/__
        irodsservice."
  fi

  echo ""
  exit 0
}

if [ -f $DEFAULT ] ; then
  . $DEFAULT
  if [ "$start" -ne "1" ] ; then
    not_start
  fi
else
  not_start
fi

if [ $User ] ; then
  uid=`(grep $User /etc/passwd | cut -f 3 -d :)`
fi

case "$1" in
  start)
    $SU $User $irods/irodsctl start
  ;;
  stop)
    $SU $User $irods/irodsctl stop
  ;;
  restart)
    $0 stop
```

```
    $0 start
  ;;
  *)
    echo "Usage: $0 {start|stop|restart}"
    exit 1
esac

exit 0
```

Listing B.5 – /etc/default/irodsservice

```
# Login de l'utilisateur (non root) pour demarrer iRODS
User=vincent

# Repertoire contenant iRODS
irods=/data/iRODS

# Doit-on demarrer iRODS (1 pour oui)
start=1
```

Listing B.6 – Lectures/Écritures sur iRODS

```
#!/bin/bash
# params : nb_iter fichier

tot_ecrit=0
tot_lect=0
for (( i = 0; i < $1; i++ )) ; do
  vit_ecrit=$(iput -v $2 | cut -d'|' -f4 | sed 's/ //g' | sed 's__
     /[^0-9.]//g')
  rm $2
  vit_lect=$(iget -v $2 | cut -d'|' -f4 | sed 's/ //g' | sed 's__
     /[^0-9.]//g')
  irm $2
  tot_ecrit=$(echo "$tot_ecrit + $vit_ecrit" | bc )
  tot_lect=$(echo "$tot_lect + $vit_lect" | bc )
done

moy_ecrit=$(echo "$tot_ecrit / $1" | bc)
moy_lect=$(echo "$tot_lect / $1" | bc)
echo "$2 (ecrit $1 fois) : $moy_ecrit"
echo "$2 (lut $1 fois) : $moy_lect"
```

Annexe C

Autres

Listing C.1 – Résultats synthétiques des tests

```
=> RES : bench.ext3.txt
cp -r /media/test/test1 /media/test/new/test1
==> 624.69 real, .56 user, 63.54 sys, 9.71 cpu
cp -r /media/test/test2 /media/test/new/test2
==> 662.29 real, .54 user, 62.73 sys, 9.28 cpu
cp -r /media/test/test3 /media/test/new/test3
==> 805.84 real, .81 user, 25.64 sys, 3.00 cpu
cp -r /var/local/test/test1 /media/test
==> 257.25 real, .52 user, 66.24 sys, 25.42 cpu
cp -r /var/local/test/test2 /media/test
==> 331.44 real, .59 user, 60.83 sys, 18.00 cpu
cp -r /var/local/test/test3 /media/test
==> 772.43 real, .83 user, 25.53 sys, 3.00 cpu
du -sh /media/test/lost+found /media/test/new /media/test/test1
==> .06 real, 0 user, 0 sys, 7.28 cpu
du -sh /media/test/lost+found /media/test/new /media/test/test2
==> 2.09 real, 0 user, .07 sys, 3.71 cpu
du -sh /media/test/lost+found /media/test/new /media/test/test3
==> 16.79 real, .19 user, 3.08 sys, 19.57 cpu
mkfs.ext3 /dev/sdb3
==> 63.35 real, .12 user, 11.01 sys, 17.00 cpu
mount /dev/sdb3 /media/test -t ext3
==> .14 real, 0 user, 0 sys, 3.28 cpu
rm -rf /media/test/lost+found /media/test/new /media/test/test1
==> 41.30 real, 0 user, 2.03 sys, 4.57 cpu
rm -rf /media/test/lost+found /media/test/new /media/test/test2
==> 41.89 real, 0 user, 2.33 sys, 5.14 cpu
rm -rf /media/test/lost+found /media/test/new /media/test/test3
==> 7.68 real, .12 user, 6.50 sys, 85.57 cpu
umount /dev/sdb3
==> 1.04 real, 0 user, .03 sys, 3.33 cpu

=> RES : bench.reiserfs.txt
cp -r /media/test/test1 /media/test/new/test1
```

```
==> 595.66 real, 1.18 user, 78.78 sys, 13.00 cpu
cp -r /media/test/test2 /media/test/new/test2
==> 728.50 real, 1.19 user, 76.17 sys, 10.00 cpu
cp -r /media/test/test3 /media/test/new/test3
==> 646.94 real, 1.15 user, 33.19 sys, 5.00 cpu
cp -r /var/local/test/test1 /media/test
==> 250.05 real, 1.08 user, 74.47 sys, 29.85 cpu
cp -r /var/local/test/test2 /media/test
==> 334.74 real, 1.16 user, 68.90 sys, 20.28 cpu
cp -r /var/local/test/test3 /media/test
==> 782.04 real, 1.12 user, 33.23 sys, 4.00 cpu
du -sh /media/test/new /media/test/test1
==> .03 real, 0 user, 0 sys, 4.71 cpu
du -sh /media/test/new /media/test/test2
==> 3.18 real, 0 user, .08 sys, 2.14 cpu
du -sh /media/test/new /media/test/test3
==> 7.02 real, .24 user, 3.03 sys, 46.28 cpu
mkfs -t reiserfs -f /dev/sdb3
==> 11.03 real, .03 user, .27 sys, 2.19 cpu
mount /dev/sdb3 /media/test -t reiserfs
==> .63 real, 0 user, .04 sys, 6.14 cpu
rm -rf /media/test/new /media/test/test1
==> 12.23 real, 0 user, 9.03 sys, 73.42 cpu
rm -rf /media/test/new /media/test/test2
==> 12.83 real, 0 user, 9.51 sys, 73.85 cpu
rm -rf /media/test/new /media/test/test3
==> 22.37 real, .16 user, 19.72 sys, 88.42 cpu
umount /dev/sdb3
==> .56 real, 0 user, .03 sys, 11.90 cpu

=> RES : bench.xfs.txt
cp -r /media/test/test1 /media/test/new/test1
==> 492.04 real, .51 user, 40.57 sys, 7.85 cpu
cp -r /media/test/test2 /media/test/new/test2
==> 605.28 real, .61 user, 40.40 sys, 6.00 cpu
cp -r /media/test/test3 /media/test/new/test3
==> 1426.44 real, 1.27 user, 30.76 sys, 2.00 cpu
cp -r /var/local/test/test1 /media/test
==> 247.76 real, .53 user, 39.22 sys, 15.42 cpu
cp -r /var/local/test/test2 /media/test
==> 331.53 real, .56 user, 37.92 sys, 11.00 cpu
cp -r /var/local/test/test3 /media/test
==> 942.50 real, 1.30 user, 31.02 sys, 3.00 cpu
du -sh /media/test/new /media/test/test1
==> .03 real, 0 user, 0 sys, 18.85 cpu
du -sh /media/test/new /media/test/test2
==> 2.16 real, 0 user, .09 sys, 4.00 cpu
du -sh /media/test/new /media/test/test3
==> 12.66 real, .33 user, 4.89 sys, 40.85 cpu
mkfs -t xfs -f /dev/sdb3
==> 2.79 real, 0 user, .06 sys, 2.04 cpu
```

```
mount /dev/sdb3 /media/test -t xfs
==> .93 real, 0 user, 0 sys, .33 cpu
rm -rf /media/test/new /media/test/test1
==> .22 real, 0 user, .16 sys, 71.14 cpu
rm -rf /media/test/new /media/test/test2
==> 8.76 real, .01 user, .51 sys, 5.42 cpu
rm -rf /media/test/new /media/test/test3
==> 872.82 real, .22 user, 19.23 sys, 2.00 cpu
umount /dev/sdb3
==> 9.36 real, 0 user, .02 sys, 1.23 cpu
```

Annexe D

Listes et tables

Table des figures

Liste des tableaux

Annexe E

Glossaire

CDS
: Centre de Données astronomiques de Strasbourg

CRM
: Cluster Resource Manager, Gestionnaire de ressources du cluster

Cluster
: Un cluster est une grappe de serveurs (ou ferme de calcul) de deux serveurs (nœuds) au minimum.

Démon
: Un démon ou daemon est un programme (un processus) qui s'exécute en arrière-plan.

ESA
: European Space Agency, Agence Spatiale Européenne

Euro-VO
: European Virtual Observatory, Observatoire Virtuel Européen

IVOA
: International Virtual Observatory Alliance, Alliance Internationale des Observatoires Virtuels

SIMBAD
: Set of Identification, Measurement and Bibliography for Astronomical Data

VOSpace
: VOSpace est l'interface définie par l'IVOA pour le stockage distribué

Watchdog
: Un watchdog est une protection destinée généralement à redémarrer le système, si une action définie n'est pas exécutée dans un délai imparti.

iRODS
: Integrated Rule-Oriented Data System