

Grégoire Bruyère
Jean-Baptiste Renevier
Vincent Bouzeran
Benjamin Cazain
Nicolas Vollmer
Romain Sahut



RAPPORT DE PROJET TUTORE

Sujet : Synchronisation de systèmes de fichiers en temps réel

Sommaire

Sommaire	- 0 -
Introduction	- 2 -
1. Présentation du sujet	- 3 -
1.1. Explication de l'intitulé du sujet	- 3 -
1.2. Buts de ce projet	- 3 -
1.3. Cahier des charges	- 3 -
1.4. Attentes et apports de ce projet	- 4 -
2. Rapport Utilisateur	- 5 -
2.1. Présentation du problème	- 5 -
2.2. Solutions possibles	- 6 -
3. Rapport technique	- 9 -
3.1. Solutions non testées.....	- 9 -
3.2. Solutions testées.....	- 10 -
3.3. Les stratégies alternatives	- 14 -
3.4. Problèmes rencontrés	- 16 -
Conclusion	- 18 -
Annexe 1: code de fam_mirror	- 19 -
Annexe 2: Installation et configuration de SSH	- 22 -
Annexe 3: Installation de NTPD et sudo	- 23 -
Liens Internet et sources	- 24 -
Table des matières	- 25 -

Introduction

Dans le cadre de notre deuxième année en informatique à l'IUT de Belfort, un projet tutoré par un enseignant doit être réalisé.

Le projet, choisi parmi tous ceux proposés, consistait à synchroniser en temps réel le serveur NFS de l'IUT et une machine de sauvegarde. Notre choix s'est porté sur celui-ci car il semblait à même de nous apporter une meilleure connaissance du système d'exploitation, ainsi que d'approfondir nos connaissances dans le domaine des réseaux de données informatiques.

Dans une première partie, nous présenterons notre sujet, ainsi que le contexte de développement d'un tel projet, en particulier quels ont été les buts pédagogiques visés. Nous verrons ensuite le cahier des charges, en précisant les objectifs techniques. On pourra ainsi conclure sur ce que nous a apporté la réalisation d'un tel projet, tant sur le plan technique que sur le plan organisationnel.

La deuxième partie servira de manuel utilisateur. Cette partie a pour but d'exposer au néophyte ce que fait la solution apportée, et d'expliquer les différents concepts nécessaires à la bonne compréhension de la partie technique.

Enfin, Nous reprendrons la partie technique de la première partie que nous développerons plus en détail, surtout pour ce qui concerne les logiciels pouvant apporter une solution au problème abordé, et leur mode de fonctionnement. Nous montrerons également dans quelles proportions et comment nous avons pu résoudre les problèmes cités dans la première partie. Cette partie permettra à d'autres informaticiens souhaitant travailler sur ce sujet d'avoir une base de documentation et ainsi leur éviter de répéter les nombreuses erreurs auxquelles nous avons pu nous heurter.

1. Présentation du sujet

Dans cette partie, nous nous efforcerons de décrire le sujet tout en restant général, c'est-à-dire sans développer les aspects techniques qui constitueront notre troisième partie.

Cette partie a pour but d'expliquer le sujet, les attentes ainsi que les nombreux apports que ce projet nous a apporté.

1.1. Explication de l'intitulé du sujet

Synchronisation de systèmes de fichiers en temps réel

Ce sujet, que l'on aurait pu qualifier, soit de sujet pour petits génies du système Linux, soit de simple installation/utilisation de programmes prévus à cet effet, nous a intéressé pour son application s'appuyant sur des couches de bas niveau du système d'exploitation. Le but de ce sujet était de préparer le travail de l'étudiant qui installera de façon définitive une des solutions proposées lors de son stage de fin de deuxième année.

1.2. Buts de ce projet

Ce projet, et notamment ce rapport, a pour but d'exposer les principales solutions possibles en matière de réplication de données, en temps réel ou non, et ainsi de permettre aux stagiaires dont le sujet de stage sera, entre autres, de mettre en place une solution de synchronisation en temps réel. Notre travail était dans un premier temps surtout un travail de documentation afin de déterminer la solution la plus efficace, puis nous avons essayé de mettre en place diverses solutions afin de mieux juger de leur fonctionnement et de leur complexité d'installation.

1.2.1. Buts pédagogiques

Le but de ce projet était de nous faire apprendre, souvent par l'erreur, les réflexes à avoir et les méthodes de travail à appliquer, dans le cadre de la recherche de solutions, et, de leurs possibilités d'application dans le cadre d'un projet. En effet, ce projet n'est absolument pas un projet de programmation, il s'agissait plus de rechercher la solution la plus adaptée et la plus fonctionnelle ainsi que de vérifier qu'elle est bien applicable dans le cadre de l'IUT.

1.3. Cahier des charges

La technique de réplication qui nous était demandée, en temps réel, devait fonctionner de la façon suivante : si on accède en écriture sur le disque à sauvegarder, le système d'exploitation exécute des appels à des fonction qui lui sont connues afin d'effectuer ces modifications. Lorsqu'il va effectuer celles-ci, ou juste après l'avoir fait, il doit transmettre ces informations à la machine de sauvegarde afin de faire les mêmes modifications aux mêmes endroits.

1.4. Attentes et apports de ce projet

En choisissant ce projet, nous savions tous qu'il s'agissait d'un projet où le travail de recherche était indispensable, et où notre inexpérience dans certains domaines du système d'exploitation risquait de ne pas voir le projet aboutir. Cependant, s'agissant d'un projet tutoré, et étant encadrés par M. Jean Luc Anthoine (LA référence de l'IUT en matière de système d'exploitation), nous savions qu'il pourrait nous expliquer et nous aider en cas de problème auquel aucun d'entre nous n'avait été confronté auparavant, et si celui-ci ne pouvait pas être résolu malgré nos tentatives.

Malgré de très nombreux problèmes, M. Anthoine nous a aidé en de très nombreuses occasions et nous a permis de comprendre nos erreurs afin de ne pas les refaire par la suite, ou tout du moins, de les résoudre plus rapidement.

1.4.1. Organisationnel

Le problème posé par un tel projet est bien entendu la juste répartition des tâches, nous avions à notre disposition deux machines de travail, ainsi que l'ensemble des machines des autres salles pour la recherche d'informations sur l'Internet. Cela impliquait un travail par petits groupes, car tout le monde ne peut pas faire avancer le projet en même temps. Ces groupes furent rapidement formés en fonction de l'intérêt de chacun pour ce sujet.

Nous avons tout d'abord recherché des témoignages sur la réplique et étudié le contenu de la page donnée en exemple par M. Anthoine afin de mieux comprendre les attentes et la problématique de la synchronisation (et donc la nécessité que celle-ci soit effectuée en temps réel). Puis nous avons étudié les sites des différents programmes cités en exemple sur les pages trouvées afin d'évaluer leur possible utilisation dans le cadre de notre projet.

1.4.2. Technique

D'un point de vue technique, ce projet nous a permis de vérifier, si besoin était, qu'une bonne documentation est indispensable lors de l'étude de solutions, celle-ci permettant, en général, de comprendre le fonctionnement de chaque solution, les problèmes et les avantages par rapport aux autres solutions découvertes.

Nous avons également acquis une méthode, une organisation permettant de trouver rapidement des solutions à des problèmes simples, mais qui font perdre un temps qui devient de plus en plus précieux au fur et à mesure que la fin du projet se rapproche. Grâce à ces réflexes acquis, notre capacité à travailler s'est accrue et notre rythme de travail s'est accéléré et nous a permis, au terme de la durée impartie pour la réalisation de ce projet, de présenter une technique de synchronisation en temps réel qui était fonctionnelle.

2. Rapport Utilisateur

2.1. Présentation du problème

Pour un fichier quelconque, et sa copie de sauvegarde, si on souhaite mettre à jour la copie, le plus simple est d'écraser le fichier de sauvegarde par la copie du fichier. On obtient alors une nouvelle copie, mise à jour.

Lors de l'application dans le cadre d'un réseau d'ordinateurs, envoyer chaque fichier pour réaliser chaque copie de sauvegarde est inenvisageable pour des raisons d'occupation de la bande passante et d'inutilité de copie de fichier identique à sa sauvegarde.

De plus, lorsque l'on veut synchroniser en temps réel deux fichiers, une donnée peu poser problème : l'heure. Le fait que l'heure de création d'un fichier ne soit pas la même sur les deux machines les fait passer pour différents lors de la comparaison. Pour éviter ce type de problème nous avons mis en place un serveur ntpd qui permet de ralentir ou d'accélérer l'horloge de l'ordinateur afin d'avoir une date système identique. Cette solution existe déjà sur l'ensemble des machines du département informatique de l'IUT.

On pourrait donc souhaiter comparer ces fichiers avant d'en effectuer la copie si cela s'avère nécessaire. Le problème rencontré est alors l'occupation de la bande passante du réseau, la comparaison s'effectuant sur une des deux machines, elle doit dans tous les cas faire transiter le fichier par le biais de celui-ci. Si le fichier n'existe pas en sauvegarde, la copie est indispensable. Même en compressant les données, cette solution n'est pas efficace dans la pratique.

La solution à ce problème est de calculer localement sur chaque machine une valeur en fonction de ses caractéristiques et de son contenu, qui permette, en ne comparant que cette valeur, d'évaluer une éventuelle modification et, le cas échéant, de copier ce fichier à la place du fichier de sauvegarde.

Ce système pourrait toutefois être amélioré en annulant la phase de comparaison en masse et en n'appliquant cette technique qu'aux fichiers ayant été modifiés. Nous avons donc besoin d'être mis au courant lors de la création/modification/suppression d'un fichier afin de la mettre à jour. C'est ce système que nous avons finalement adopté bien que ce n'était pas celui premièrement choisi et qu'il ne réponde pas tout à fait aux objectifs demandés du point de vue du fonctionnement. Ce système permet aux machines d'être synchronisées initialement, et surtout de le rester après chaque modification effectuée sur la zone à synchroniser.

Une dernière méthode de réplication en temps réel existe et permet de retransmettre les requêtes de bas niveau de la machine à sauvegarder à la machine de sauvegarde. C'est cette solution que nous avons abordée dans un premier temps.

2.2. Solutions possibles

2.2.1. Rsync

Rsync est un programme qui synchronise à un instant t le contenu d'une zone choisie avec une autre machine à distance. Ce programme compare les valeurs calculées en fonction des deux enregistrements et met à jour la copie si besoin. Cette solution, appliquée régulièrement, permet d'avoir une copie de sauvegarde récente, ou tout du moins, de ne pas perdre l'intégralité des données lors d'un problème matériel.

Cette solution est néanmoins coûteuse en accès disques en ne garantissant absolument pas la validité des informations à court terme si celles-ci sont enregistrées entre la dernière sauvegarde et la panne. Il est à remarquer que cet outil permet d'avoir la même date/heure sur le fichier et sa sauvegarde, sans avoir besoin d'un serveur ntpd.

Cette solution est relativement efficace, mais dispose de nombreux problèmes dont les plus sérieux apparaissent sur de grosses quantités de données (plus de 200 utilisateurs utilisent régulièrement la machine en question) :

- Le temps nécessaire pour effectuer la synchronisation
- Le fait que celle-ci soit faite à un instant précis, et non en temps réel, ne garantissant pas l'intégrité des données enregistrées juste avant la panne.

2.2.2. imon

Imon (Inode Monitor) est une partie du système d'exploitation qui lui permet d'envoyer des signaux spécifiques lorsqu'un changement est effectué sur une zone choisie d'une unité de stockage. Les programmes, comme fam, peuvent récupérer ces informations, qui, de plus contiennent le type de changement (création, modification, effacement, exécution...).

Par exemple, si on efface un fichier, le système d'exploitation va prévenir les programmes qui sont à son écoute que le fichier a été effacé. C'est simple et permet d'avoir une connaissance en temps réel des modifications effectuées.

Pour cela, le système d'exploitation doit savoir sur quelle zone il doit observer les modifications, il doit donc être amorcé par un programme servant de relais entre les besoins de l'utilisateur et les messages du système d'exploitation.

Enfin, ces fonctions ne sont pas exploitables directement, elles sont par contre utilisables par le biais de programmes qui « écoutent » imon, comme fam, et permettent de définir les zones à observer ainsi que les opérations à effectuer selon les messages émis par imon et reçus par ces programmes.

2.2.3. DNotify

C'est également une partie du système d'exploitation, similaire à imon, mais qui est plus récent. Il peut aussi être utilisé par fam, mais celui-ci devra être modifié pour pouvoir comprendre les messages du système d'exploitation (imon et DNotify ne dialoguent pas de la même façon, pas dans la même langue).

2.2.4. fam

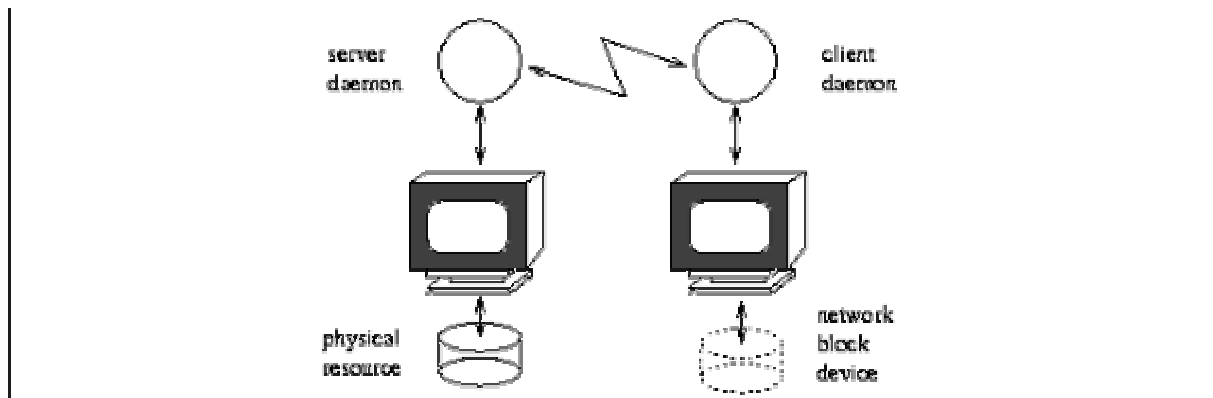
fam (File Alteration Monitor) est le programme qui va fournir à imon les zones à observer pour reporter les modifications. C'est un programme, donc doit être exécuté pour être activé. Il permet d'observer et de déclencher des opérations à effectuer selon le type de modification.

Une fois exécuté, il va commencer par avertir le système d'exploitation que imon doit observer la zone choisie, ce qu'il fera, s'il détecte un changement dans cette zone, il va prévenir fam (ex : « le fichier N°xxxxx a été supprimé »), celui-ci décidera alors de la marche à suivre. Il se place donc en plus et au dessus de imon, bien que pouvant fonctionner sans, mais d'une manière moins efficace. Ce programme peut être facilement modifié afin de pouvoir fonctionner avec DNotify et les systèmes d'exploitation plus récents.

2.2.5. NBD

NBD agit comme de « longs câbles », il permet d'accéder aux fichiers spéciaux de périphériques de type bloc d'un ordinateur distant par l'intermédiaire du réseau. Le disque distant apparaît sur la machine locale comme un périphérique de type bloc.

Il n'est pas nécessaire que ce dernier soit un disque entier ou même une partition, cela peut seulement être un fichier.



Le NBD est à l'origine destiné à l'usage du RAID sur des réseaux, et au partage de système de fichiers. On peut monter une partition NBD sur un système configuré en RAID mirroring (ou RAID1) afin d'effectuer des sauvegardes en temps réel et sécurisées d'une machine distante (serveur) sur une machine locale (client). Pour être plus clair : on lance une connexion nbd vers un serveur nbd distant, celle-ci utilise les ressources physiques de ce serveur (généralement le périphérique /dev/nda) qui devrait normalement être configuré en RAID.

2.2.6. Le RAID

Le RAID permet d'enregistrer simultanément les données en plusieurs exemplaires sur plusieurs disques distincts, en cas de panne il est possible de récupérer les données.

Le RAID 1, ou "Mirroring" : - c'est une option de sécurité, en fait simplement, les données sont enregistrées simultanément et en double sur un deuxième disque. Si l'un des disques tombe en panne, on peut retrouver les données sur le deuxième disque. Sur certains contrôleurs SCSI il est même possible de changer le disque défectueux sans arrêter le système... L'inconvénient de ce mode est qu'il n'est justement pas économique puisque pour faire un volume de 10Go il faut deux disques de... 10Go. L'avantage est que les données sont sécurisées par rapport aux pannes mécaniques/électroniques des disques.

3. Rapport technique

Dans ce projet, nous avons du avoir accès à de nombreux noyaux avec des modules et des fonctionnalités différentes (pour imon et NBD notamment) afin tester les différentes possibilités intéressantes.

Devant le temps pris pour la compilation d'un noyau sur une des machines dont nous disposons, nous avons fait compiler la quasi-intégralité de nos noyaux par M.Anthoine qui disposait dans son bureau d'une machine plus puissante. Nous lui demandions donc un noyau, accompagné de la liste des modules et des fonctionnalités dont nous avons besoin.

3.1. Solutions non testées

3.1.1. Rsync (remote synchronization)

Rsync est un programme qui compare des fichiers au travers d'un réseau de communication par la comparaison de valeurs calculées en fonction du nom, de la date de dernière modification, et du contenu du fichier.

Si rsync détecte un fichier créé, modifié ou effacé, il procède à la copie de ce fichier et donc à sa sauvegarde.

Ce programme est utilisé régulièrement à l'IUT pour faire la sauvegarde du serveur de fichiers NFS. Pour les raisons expliquées précédemment, cette solution est devenue lourde à l'utilisation car les cinq serveurs NFS (demeter, dionysos, heracles, helios, zeus) ont été fusionnés l'an dernier, et, par conséquent l'utilisation du serveur fut multiplié par cinq (A ce jour, le NFS contient 27 giga octets de données pour plus de 200 utilisateurs réguliers). Le fait de vérifier si des modifications ont été effectuées ou non prend un temps considérable et procède à de nombreuses opérations inutiles.

Notre projet avait pour but de le remplacer, cependant, la solution apportée utilise toujours rsync, mais uniquement de manière locale : on utilisera rsync QUE pour mettre à jour un fichier ou un répertoire. On évite ainsi le problème du temps de vérification (il ne procède à la vérification que d'un fichier ou que d'un répertoire).

Ce programme nécessite SSH (qui signifie Secure Shell). C'est un protocole qui permet de faire des connexions sécurisées (=cryptées) entre un serveur et un client SSH. Il est également possible d'utiliser le programme OpenSSH, qui est la version libre du client et du serveur SSH.

Le programme sudo est également nécessaire. Sudo permet à certains processus de s'exécuter en mode super-utilisateur alors que le programme principal tourne en mode utilisateur normal.

La commande utilisée pour la réplication d'un fichier ou d'un utilisée dans le script perl fam_mirror est :

```
rsync -logptz -e 'ssh -l root'
```

Explication de la commande :

l : links = convertit les liens symboliques en leurs chemins respectifs réels

o : owner = conserve le N° du propriétaire du fichier (UID)

p : perms = conserve les permissions du fichier

g : group = conserve le N° du groupe (GID)

z : compress = envoie après compression

t : times = conserve l'heure originale

-e : ce qui suit remplace le rsh normalement effectué

ssh -l root : on préfère faire un ssh à partir d'un processus lancé par le root.

3.1.2. DNotify (directory notification)

DNotify étant une version concurrent et plus récents que imon (et celui-ci étant un module expérimental), il est présent dans les version plus récentes du noyau (2.4.19 ou plus récent) et ne nécessite pas de patcher le noyau.

Cependant, il a autant besoin de fam que imon, pour lui annoncer ce qu'il doit faire (les zones à surveiller) et ce qui doit être fait.

Or, imon envoie des requêtes spécifiques qui sont différentes (d'un point de vue sémantique) des requêtes émises par DNotify. Il faut donc patcher fam avant sa compilation afin qu'il comprenne les messages de DNotify.

DNotify devrait présenter de nombreux avantages par rapport à imon, notamment au niveau de la correction des bugs.

3.2. Solutions testées

Nous avons débuté ce projet par des recherches sur Internet. Concernant ce sujet de duplication en temps réel, nous nous sommes tout d'abord orienté vers une technique utilisant NBD et le RAID logiciel. Nous expliquerons plus tard pourquoi cette méthode n'a pas pu être finalisée.

3.2.1. NBD (Network Block Device)

Nbd est un module du noyau linux et un tandem client/serveur servant à exporter des ressources de type bloc, sous forme de ressources de type bloc par le réseau. Afin de fonctionner correctement, NDB requiert que le noyau soit SMP (pour machines multiprocesseurs) et inclus les modules RAID et NBD, ainsi que SSH (Secure Shell) et sudo.

- Il faut tout d'abord s'assurer que le noyau 2.4.18 est présent sur le serveur comme sur le client, sinon le compiler puis l'installer avec la commande :
dpkg -i version_debian
- Se procurer l'archive **kbd-2.4-current.tgz**

- Extraire cette archive avec la commande : **tar xzvf nbd-2.4-current.tgz**
- Entrer dans le répertoire ainsi créé : **cd nbd-2.4.29**

A partir d'ici nous expliquons les modalités d'installation et de compilation du module NBD, sans passer par une recompilation du noyau.

Attention, avant tout, vérifiez bien que le lien linux pointe vers la version de votre noyau, nous avons, par manque d'expérience, compilé avec succès plusieurs modules pour noyau 2.2.20 qui refusaient de fonctionner (nous étions sur un noyau 2.4.18). Après modification, le module ne se compilait plus, à cause de problèmes indépendants de notre compétence.

- Il devrait être suffisant de taper « **make** » dans ce répertoire. Cela va créer *nbd.o*, *nbd-server* et *nbd-client* dans */tmp*.
- Lancer « **make config all** » pour être vraiment sûr que tout a été installé bien que « **make** » normalement suffit.
- Editer le « **Makefile** » et remplacer SERVER et CLIENT par le nom respectif de ces machines.
- S'assurer que *sudo* et *ssh* sont bien installés sur les 2 machines SERVER et CLIENT et qu'en tant qu'utilisateur vous avez les droits *root*.
- Lancer « **make test** ». Pour que cela fonctionne, cela dépend de la présence de *sudo* et *ssh*. S'assurer que le module *nbd.o* est chargé. Pour cela utiliser */sbin/lsmmod* ainsi que le nbd serveur et le nbd client sont en cours d'exécution. Pour vérifier, utiliser **ps tree**.
- Ensuite il faut vérifier qu'il existe bien une connexion entre le serveur et le client (utiliser de nouveau **ps tree**). Vérifier aussi que l'état du périphérique est bon en faisant un **cat /proc/nbdinfo**. S'assurer que les informations de */dev/nda* qui correspondent aux connexions sont correctes. Si en dépit de toutes ces vérifications quelque chose se passe mal, il est toujours possible de consulter les fichiers logs systèmes pour vérifier les messages d'erreurs.
- Choisir un fichier ou une partition sur le serveur ainsi que quelques ports afin de communiquer avec le client puis lancer le serveur :
- **nbd-server 1001 1002 1003 1004 /dev/sda1**
- Sur le client, charger le module NBD avec : **insmod nbd.o**
- Lancer le client : **nbd-client votre_serveur 1001 1002 1003 1004 /dev/nda**

Remarque :

nbd.o est un module du noyau de Linux, il est par conséquent compilable lors de la compilation du noyau. Nous nous sommes acharné sur la compilation du module du noyau durant plusieurs semaines à cause d'incompatibilités de version (seule le module pour noyau 2.2.20 était compilable, les autres produisaient des messages d'erreurs), alors qu'il nous aura suffi de l'inclure dans les modules choisis dans la liste de ceux disponibles lors de la compilation du noyau.

NBD – Fonctionnement du client :

Le programme *nbd-client* est un démon client NBD qui manipule les requêtes du noyau envoyées depuis un périphérique de type bloc local et qui les transmet à travers le réseau à un démon serveur distant. Il en résulte que la ressource physique du serveur distant apparaît localement. Le NBD est alors traité comme n'importe quel autre périphérique de type bloc. Le *nbd-client* opère comme un processus normal de

niveau utilisateur mais communique avec le noyau via des appels **ioctl()**. Pour cela le module *nbd.o* doit être présent dans le noyau pour que ça marche.

NBD – Fonctionnement du serveur :

Le programme *nbd-serveur* est un démon serveur nbd qui manipule les requêtes du client distant comme une ressource locale. Cette ressource local peut être un fichier (ou des fichiers) et/ou un périphérique de type bloc. S'il y a plus d'une ressource locale, ils doivent être regroupés en utilisant des protocoles de type RAID. Seuls les RAID de niveau 1 et 0 sont supportés.

Ces fichiers regroupés ne doivent pas excéder la taille maximum de 2 Giga-octets. Le *nbd-serveur* opère comme un processus normal de niveau utilisateur et ne fait pas d'appel particulier à **ioctl()**. Cette implémentation permet au client la lecture seule ou la lecture/écriture de ces ressources. Le démon lance un processus à chaque connexion et en relance un s'il meurt.

Le client et le serveur utilisent les procédures d'authentification et de cryptage *Secure Socket Layer (SSL)*.

Cette solution fut abandonnée par manque de renseignements clairs sur la possibilité et la méthode d'installer une solution de RAID logiciel.

3.2.2. imon (inode Monitor) et fam (File Alteration Monitor)

Devant le fait que nous avons réussi, avec beaucoup de mal et d'aide à installer un serveur NBD ainsi qu'un client NBD, et que cette solution nécessitait une solution RAID logiciel que nous ne savions pas implanter. Nous nous sommes redirigés vers la solution initialement proposée par M. Anthoine, à savoir fam et imon.

imon, le Moniteur Inode, est la partie du noyau qui informe fam lorsque des fichiers ont été modifiés. Quand des applications informent fam qu'elles sont intéressées par des fichiers ou des répertoires, fam le signale à imon. Lors de modifications des fichiers surveillés par imon, le noyau en informe imon, qui informe fam et qui remonte à son tour l'information aux applications qui lui en ont fait la demande. imon a été développé pour le noyau d'IRIX en 1989 par Wiltse Carpenter; le portage Linux a été réalisé par Roger Chickering.

L'implémentation Linux dans le correctif du noyau pour imon, est semblable à celle d'IRIX sauf pour ce qui concerne la manière dont elle s'intègre dans le code du système de fichiers.

Il s'agit d'un patch pour le noyau (ou kernel) dont la dernière version s'applique à la version 2.4.17 du noyau. Nous avons utilisé un noyau 2.4.18, et le patch s'est bien effectué à l'exception d'une insertion qui n'a pas pu s'effectuer. M. Anthoine nous a donc patché cette ligne à la main, puis a compilé le noyau. Le problème venait du test de positionnement de cette insertion qui vérifiait les 3 lignes suivant et précédent le patch. Or, il ne reconnaissait pas, dans le cas présent, la troisième ligne suivant le patch à appliquer.

Principe de fonctionnement :

Lorsqu'un programme prévient le noyau (patché pour inclure imon) qu'il souhaite avoir des informations sur les actions effectuées sur un répertoire ou un fichier, imon récupère son n° d'inode (le numéro d'identification du fichier en quelques sortes) et le stocke. Toute action sur un fichier géré par ce noyau lancera un appel à une macro de imon, qui, si les actions effectuées sur ce fichier sont observées (si son n° d'inode est dans la liste des inodes stockés), déclenchera une fonction de imon afin de prévenir le programme de plus haut niveau (ici fam) que le fichier a été accédé, et le type d'accès effectué (création, suppression, modification, exécution, fermeture).

Les événements sont gérés en file de type FIFO (First In First Out) ce qui s'inscrit parfaitement dans la logique de la correcte réplication des données, à savoir que même en cas de surcharge, les versions concurrentes d'un même fichier arriveront dans le bon ordre.

fam, le Moniteur d'Altération de Fichier, fournit une API (Les « Application Program Interface » donnent accès à des informations ou des fonctions directement intégrées au système d'exploitation) que les applications peuvent utiliser pour être informées de chaque modification de fichiers ou de répertoires. Il est le lien entre le développeur d'applications et imon, et permet de dialoguer plus simplement avec imon). Ce programme fonctionne également sur les noyaux sans moniteur d'activité (comme imon et DNotify), mais les temps de réponse sont plus élevés et cela ne permet plus d'observer les débuts et fins de lectures de fichiers (notre problème est ici le temps de réponse). Un autre problème de fam est le fait qu'il ne gère qu'un niveau de répertoire, il n'est absolument pas récursif.

fam se décompose en deux parties: fam, le démon qui est à l'écoute des requêtes et fournit les informations, et libfam, une bibliothèque que les applications clientes utilisent pour dialoguer avec fam.

fam peut également informer ses clients du début et de la fin de l'exécution d'un fichier, cette fonctionnalité ne nous concerne pas pour effectuer la synchronisation.

fam a été développé à l'origine pour IRIX en 1989 par Bruce Karsh, et a été réécrit en 1995 par Bob Miller.

SGL::FAM est le module perl qui permet de communiquer avec fam encore plus simplement ! les scripts en perl sont vraiment très simples à modifier (et on le verra avec le script fam_mirror) et par conséquent, l'installation du module SGL::FAM requiert d'avoir perl installé sur la machine (installation pour la Debian grâce à dselect, nous utilisons la version 5.6.1).

Pour l'installation, on peut l'effectuer en grande partie via la commande dselect, puis en exécutant les commandes suivantes :

```
$> perl -MCPAN -e shell
cpan> install SGL::FAM
```

Ces commandes vont vous ouvrir la console perl dans laquelle vous demanderez d'installer SGL::FAM, il se peut qu'il vous propose d'installer quelques

autres modules, installez les tous. Même après avoir installé tous les modules requis, il nous manquait encore le module Test::Helper. Nous l'avons donc installé à la main à partir de la console perl.

fam_mirror est un script shell qui permet d'appliquer récursivement fam à une série de répertoires lors de l'initialisation. Pour être fonctionnel, il suffit de modifier la valeur de \$ReplicaHosts, et de lui assigner les noms des machines qui serviront de sauvegardes (ici, une seule machine). On lance le script avec comme argument le répertoire ou le fichier à observer. Si il n'affiche pas de message d'erreur, alors les dépendances de SGI::FAM devraient être respectées. Il place alors dans chaque répertoire un moniteur d'activité. Il dispose également de trois variables d'état pouvant être déclarées ou silencieuses. Il s'agit de DEBUG (pour afficher ce que fait fam_mirror et ainsi bien comprendre comment il fonctionne), INIT_MIRROR (pour synchroniser les machines la première fois) et CREATE_DIRS (pour créer une arborescence similaire, mais vide... cette dernière option semble inutile dans notre projet). Ces trois variables doivent valoir 1 pour être activées.

Exemple : `$>DEBUG=1 INIT_MIRROR=1 ./fam_mirror /users/info`

Synchronisera, puis agira en temps réel sur ce qui se passe sur /users/info

Cependant, le script était « programmé avec une hache » (citation de M. Antoine) ce qui a nécessité une mise au point, chose relativement facile car le script est très simple. Les erreurs présentes dans ce script vont du test sur le répertoire qui vient d'être effacé (donc bug...) au fait qu'il était récursif, mais sans observer l'activité des nouveaux répertoires. Nous avons donc corrigé les non-sens, séparés certain cas afin de rajouter des moniteurs d'activités dans chaque nouveau répertoire.

Cette solution présente le problème d'un ajout sur un gros fichier, même si ne modifie que la date du fichier, celui-ci sera synchronisé et donc copié.

Exemple : `$> touch grosFichier` entraînera la copie du fichier par le réseau.

Le script original, associé aux modifications est en annexe 1.

3.3. Les stratégies alternatives

Dans cette partie nous vous exposerons des stratégies de synchronisation en temps réel non documentées. Celles-ci ne marcheront sûrement pas toutes, mais sont là pour permettre de réfléchir aux orientations à prendre en cas de volonté de d'organiser différemment la répartition des tâches client/serveur :

3.3.1. imon + fam + NBD

Dans la partie technique de imon, nous avons vu qu'il était inapplicable sur un montage NFS, le problème apparaît il avec NBD ? Logiquement, non ! NBD, contrairement à NFS, permet de partager des ressources et de les exporter sous la forme d'un périphérique de type bloc. On pourrait ainsi effectuer le monitoring depuis le client.

3.3.2. DNotify + fam + NFS

Les renseignements concernant les fonctionnalités de DNotify sont très peu nombreux, on ne peut donc pas savoir si DNotify permet de faire fonctionner fam avec le NFS. On pourrait ainsi effectuer le monitoring depuis le client. Cependant, comme nous l'avons expliqué précédemment, DNotify nécessite de patcher fam et d'avoir un noyau 2.4.19 ou plus récent.

3.3.3. NBD + RAID logiciel

Malgré notre échec dans la partie RAID logiciel de cette solution, il semblerait qu'il soit possible d'effectuer cette partie d'une manière assez simple. Ce que l'on veut ici, c'est du RAID1, c'est-à-dire un simple mirroring.

Principe et configuration du RAID1 :

On dispose de deux disques de taille sensiblement égale que l'on souhaite mettre en miroirs. On peut avoir des disques supplémentaires que l'on gardera en attente comme disques de secours et qui prendront automatiquement place dans la matrice si un disque actif tombe en panne. Ici, un disque dur fixe, et un disque de type NBD répondant aux critères.

Voici le fichier `/etc/raidtab` typique :

```
raiddev                /dev/md0
raid-level              1
nr-raid-disks          2
nr-spare-disks         0
chunk-size             4
persistent-superblock  1
device                 /dev/nda      #Ici le disque en lecture, NDB
raid-disk              0
device                 /dev/sdc5     #ici le disque de sauvegarde
raid-disk              1
```

Pour éventuellement prendre en compte des disques de secours :

```
device                 /dev/sdd5
spare-disk             0
```

N'oubliez pas d'ajuster la variable `nr-spare-disks` en conséquence.

A présent, on peut initialiser la matrice RAID. Son contenu doit être construit et les contenus des deux disques (sans importance pour l'instant) synchronisés.

Exécutez :

```
$> mkraid /dev/md0
```

L'initialisation de la matrice démarrera.

Examinez le fichier `/proc/mdstat`. On doit y lire que `/dev/md0` a été démarré, que le miroir est en cours de reconstruction et y trouver une estimation de la durée de reconstruction.

La reconstruction a lieu durant les périodes d'inactivité au niveau des entrées/sorties. L'interactivité du système ne devrait donc pas en souffrir.

3.4. Problèmes rencontrés

3.4.1. Niveau technique

Nous avons rencontré différents problèmes lors de l'installation de nbd.

- Problème de recompilation

Le `make clean` n'efface pas tous les fichiers dans le répertoire `/tmp` et ceci génère des erreurs si on veut réaliser une recompilation après une modification. Il suffit d'aller supprimer les fichiers.

- Nécessité des sources du kernel pour la compilation de nbd en tant que module du system d'exploitation

Ce problème a été résolu par une compilation d'un kernel avec nbd déjà inclus en module à la compilation, on peut ainsi faire un `modprobe nbd` pour charger le module.

- Problème de compilation car il nous manquait des primitives

Ce problème a été résolu par une recompilation d'un noyau avec SMP (multiprocesseur même si nous en disposons que d'un) et RAID.

- Création des devices nbd

Ce problème a été résolu par un simple `mknod` pour créer les devices dans `/dev` comme la documentation nous l'invitait à faire.

Le fait de charger le serveur NFS de moniteurs d'activités, d'une couche de communication par-dessus les moniteurs d'activités, puis encore une couche de communication simplifiée en perl peut sembler brutale pour un serveur déjà chargé, et devant être assigné à d'autres charges par la suite, nous aurions souhaité décharger en partie le serveur, et gérer les moniteurs sur le montage du NFS sur le client. Ceci est impossible à cause de bugs dans `imon`, le NFS lui font générer des événements qui n'existent pas.

Notre serveur, à deux reprises, a détruit les informations contenues sur le disque dur, et le disque dur, pour des raisons matérielles, ce qui a induit un retard dans les tests, au moment où l'échéance de la fin des projets tutorés se rapprochait de manière inquiétante. Nous avons pu toutefois faire marcher la réplication de façon tout à fait acceptable ; Seule subsistait une erreur de `rsync` dû à une version trop vieille. L'utilisation de la ligne de commande telle que décrite dans la partie technique consacrée à `rsync`, ainsi que la dernière version de `rsync` devrait permettre une synchronisation en temps réel sans que cela ne vienne perturber les accès des utilisateurs.

3.4.2. Niveau organisationnel

Un des problèmes organisationnel majeur rencontré lors de ce travail en groupe était la volonté et la présence de quelques éléments du groupe. Cependant, le nombre de machines mises à notre disposition ne nous permettait pas de tous travailler activement au même moment.

Ce projet nous aura permis d'acquérir des méthodes ainsi que des réflexes essentiels dans la recherche et la mise en place de solutions adaptées.

Grâce à l'aspect tutoré de ce projet, M. Antoine nous a également aidé à ne pas perdre de vue l'objectif fixé, et nous à rappelé de bien lire les nombreuses documentations disponibles sur l'Internet, bien que, hélas, elles ne soient pas toujours claires et exhaustives, ainsi que TOUS les manuels auxquels nous pouvions avoir accès et étant susceptible de contenir des solutions.

Notre projet tutoré aura donc été très enrichissant du point de vue organisationnel, et cet apprentissage par l'erreur nous aura été bénéfique, car nous aura fait prendre conscience de notre manque d'expérience dans ce domaine.

L'expérience acquise nous permettra donc d'être plus efficace dans la recherche de solutions et d'organisation de la tâche à effectuer, d'abord lors du stage en entreprise de deuxième année, puis dans notre vie professionnelle future.

Conclusion

Bien que nous ne soyons pas parvenu, à remplir totalement les objectifs qui étaient de faire du RAID via le réseau, c'est-à-dire reproduire les requêtes d'écriture bas niveau d'un disque dur et les effectuer sur une machine distante, nous avons pu mettre en place une solution qui répond aux attentes.

Nous sommes malgré tout parvenus à nous organiser, pour mettre en place une solution pleinement fonctionnelle. L'apprentissage du travail en groupe aura été enrichissant pour chacun d'entre nous, ce qui paraît essentiel avant de découvrir le monde de l'entreprise. A ce jour, la solution fam accompagnée de imon et du script perl fam_mirror modifié réalise une synchronisation en temps réel d'un système de fichiers par le réseau.

Nous avons également acquis une plus grande connaissance du système d'exploitation Linux, et plus particulièrement de ses accès et requêtes bas niveau.

Annexe 1: code de fam_mirror

```
#####
#!/usr/bin/perl
# Author: Atif Ghaffar <atif@developer.ch>
# version 0.1
# You may find the later versions of this program at
# http://atif.developer.ch
#####

# Version améliorée par et pour l'IUT de Belfort Montbéliard

use strict;
BEGIN {
    my $usage=qq|

Normal USAGE:
$0 directory [directory2] [directory3] ... [directoryN]

To create a set of identical directories on the remote host if they dont exists
CREATE_DIRS=1 $0 directory [directory2] [directory3] ... [directoryN]

To mirror all files to the remote host. This can be done as the initial setup.
INIT_MIRROR=1 $0 directory [directory2] [directory3] ... [directoryN]

To have VERBOSE messages about what this script is doing
DEBUG=1 [INIT_MIRROR=1] [CREATE_DIRS=1] $0 directory [directory2] [directory3] ... [directoryN]
    \n|;

    if (@ARGV){
        for (@ARGV){
            unless (-d $_ && -e $_){ # check if the argument is a directory
                print "$_ is not a directory\n";
                print $usage;
                exit;
            }
        }
    } else {
        # show the usage unless a directory is specified
        print $usage;
        exit;
    }
}
use vars qw($directory $cmd $event $dir $file $filepath $dirname);

#load some modules.
use File::PathConvert qw(realpath);
use File::Basename;
use File::Find;
use SGI::FAM;

#start a fam object
my $fam=new SGI::FAM;
my $event;

#####
#     #define the rsh command. This could be rsh, ssh or whatever
#     my $rsh="ssh -l root ";
#     #define the rsync command with the flags that you want
#     my $rsync="rsync -rlogptC --delete -e 'ssh -l root' ";
#     #define replica hosts separated by space
#     my @replicaHosts=qw(host1 host2 host3);
#####
```

Synchronisation de systèmes de fichiers en temps réel

```
#define the rsh command. This could be rsh, ssh or whatever
my $rsh="ssh -l root ";

#define the rsync command with the flags that you want
my $rsync="rsync -logzt -e 'ssh -l root' "; #non testé mais plus adapté
# my $rsync="rsync -rlogztC -e 'ssh -l root' "; #testé, fonctionne très bien mais risque de mouliner
#define replica hosts separated by space
my @replicaHosts=qw(projet1);

#fill up the @directories list
my @directories;
find(sub { -d && -e && push @directories, $File::Find::name; }, @ARGV);

for (@directories){
    #convert symlinks to realpath
    $directory=realpath($_);
    #get some stats about this directory
    my ($dev,$ino,$mode,$nlink,$uid,$gid) = stat($directory);
    $mode=sprintf "%04o", $mode & 07777;

    #create identical directories on replica hosts if environment variable CREATE_DIRS is set.
    if ($ENV{'CREATE_DIRS'} || $ENV{'INIT_MIRROR'}){
        for my $host(@replicaHosts){
            $cmd="$rsh $host 'mkdir -p $directory; chmod $mode $directory; chown $uid.$gid
$directory";
            print "$cmd\n" if $ENV{'DEBUG'};
            system ("$cmd 2>/dev/null");
        }
    }

    print "setting monitor on $directory\n" if $ENV{'DEBUG'};
    $fam->monitor(realpath($_));
}
# if there is a request to initiate a mirror then lets do it.
# directories must already have had been created above
if ($ENV{'INIT_MIRROR'}){
    for (@ARGV){
        $directory=realpath($_);
        for my $host(@replicaHosts){
            $cmd="$rsync $directory $host:$directory";
            system ("$cmd 2>/dev/null");
            print "$cmd\n" if $ENV{'DEBUG'};
        }
    }
}
# now running the main loop which will recieve events from fam
# this should actually be forked into a background process.
# for the timebeing you can run it with &
# perhaps I will use POE at somepoint with this
while (1) {
    do {
        $event=$fam->next_event;
        $dir=$fam->which($event);
        $file=$event->filename;
        #dont copy swap files
        next if $file=~/(\.swp|\.~)$/;
        if ($dir eq $file){
            $file="";
        }

        #set correct filename. dir/file
        my $filepath="$dir/$file";

        #remove multiple / from filepath
        $filepath=~s/\+/\/g;
    }
}
```

Synchronisation de systèmes de fichiers en temps réel

```
#set dirname
$dirname=dirname($filepath);

#####
#           # if there a change or create event then
#           # rsync the file to the replica hosts
#           if ($event->type =~/^(change|create)$/){
#               for my $host(@replicaHosts){
#                   $cmd="$rsync $filepath $host:$dirname/";
#                   print "$cmd\n" if $ENV{'DEBUG'};
#                   system ("$cmd 2>&1 >/dev/null");
#               }
#           }
#           # if the file or directory is deleted
#           # then delete it on the server too
#           # This needs some testing
#           if ($event->type =~/^(delete)$/){
#               for my $host(@replicaHosts){
#                   if (-d $filepath){
#                       $cmd="$rsh $host 'rm -rf $filepath'";
#                   } else {
#                       $cmd="$rsh $host 'rm $filepath'";
#                   }
#                   print "$cmd\n" if $ENV{'DEBUG'};
#                   system ("$cmd 2>&1 >/dev/null");
#               }
#           }
#       }
#####
# Si on récupère l'événement "fichier modifié"
# on fait la sauvegarde
#           if ($event->type =~/^(change)$/){
#               for my $host(@replicaHosts){
#                   $cmd="$rsync $filepath $host:$dirname/";
#                   print "$cmd\n" if $ENV{'DEBUG'};
#                   system ("$cmd 2>&1 >/dev/null");
#               }
#           }
# Si on récupère l'événement "fichier créé"
# on fait la sauvegarde ou on crée le répertoire et on
# y met un moniteur d'activité
#           if ($event->type =~/^(create)$/){
#               for my $host(@replicaHosts){
#                   if (-d $filepath){
#                       $directory=$filepath;
#                       $sch="setting monitor on $directory\n";
#                       print "$sch\n" if $ENV{'DEBUG'};
#                       $fam->monitor($filepath);
#                   }
#                   $cmd="$rsync $filepath $host:$dirname/";
#                   print "$cmd\n" if $ENV{'DEBUG'};
#                   system ("$cmd 2>&1 >/dev/null");
#               }
#           }
# Si on récupère l'événement "fichier effacé"
# on efface sur les clients de sauvegarde
#           if ($event->type =~/^(delete)$/){
#               for my $host(@replicaHosts){
#                   $cmd="$rsh $host 'rm -rf $filepath'";
#                   print "$cmd\n" if $ENV{'DEBUG'};
#                   system ("$cmd 2>&1 >/dev/null");
#               }
#           }
} while $fam->pending;
}
```

Annexe 2: Installation et configuration de SSH

- installation d'un serveur sshd :

Le serveur sshd peut s'installer rapidement depuis une debian à travers un « apt-get install ssh » ou bien directement depuis l'utilitaire dselect de debian.

- permettre une connexion sans demande de mot de passe :

Une connexion de base en ssh nécessite l'utilisation d'un mot de passe

```
$> ssh -l test projet1
```

```
The authenticity of host 'projet1.iut-bm.univ-fcomte.fr (193.52.61.47)' can't be established.
```

```
DSA key fingerprint is 66:3d:bf:60:42:bf:97:0c:b4:80:af:de:38:86:f8:27.
```

```
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added 'projet1.iut-bm.univ-fcomte.fr, 193.52.61.47' (DSA) to the list of known hosts.
```

```
test@projet1.iut-bm.univ-fcomte.fr's password:
```

```
Last login: Sat Mar 22 05:08:09 2003 from ...
```

Or, il est impensable de saisir le mot de passe à chaque fois, surtout lorsqu'il s'agit d'un programme qui souhaite se connecter de manière automatique, donc nous allons créer une clef publique rsa afin de ne pas avoir à saisir ce mot de passe à chaque connexion.

Configuration de la machine client depuis laquelle on se connectera sur une machine serveur :

```
$> ssh-keygen -t rsa
```

```
Generating public/private rsa key pair.
```

```
Enter file in which to save the key (/home/test/.ssh/id_rsa):
```

```
Enter passphrase (empty for no passphrase): ← vide
```

```
Enter same passphrase again: ← vide
```

```
Your identification has been saved in /home/test/.ssh/id_rsa.
```

```
Your public key has been saved in /home/test/.ssh/id_rsa.pub.
```

```
The key fingerprint is:
```

```
9a:65:9a:dc:f3:b0:23:54:40:95:ff:c4:d5:04:ac:3d test@client
```

Il suffit alors de copier le contenu de notre fichier /home/test/.ssh/id_rsa.pub sur la machine serveur sshd dans le fichier ~/.ssh/authorized_key avec un simple cat par exemple :

```
$>cat id_rsa.pub >> ~/.ssh/authorized_key
```

On pourra se connecter sur la machine serveur depuis la machine client par un simple ssh projet2, ou ssh -l test projet2 ou même exécuter une commande sur la machine distance à l'instar de rsh : ssh projet2 date pour obtenir la date de la machine distance.

Annexe 3: Installation de NTPD et sudo

NTPD

- Installation du serveur de temps ntpd

Depuis une debian **apt-get install ntpd** ou bien même depuis l'utilitaire **dselect** ou en téléchargement les sources et en compilant.

- Configuration

Le fichier de configuration est /etc/ntp.conf

Une ligne importante l'adresse des serveur sur le ou lesquels on va se connecter, l'IUT utilise chronos en serveur de temps.

echo "server chronos.iut-bm.univ-fcomte.fr" >> /etc/ntp.conf

- Lancement de ntpd et test

Lancement :

\$> ntpd

Vous pourrez bien sur par la suite le lancer automatiquement, lors du boot de l'ordinateur.

Test de la configuration :

\$>ntpd

ntpd> peers

<i>remote</i>	<i>local</i>	<i>st</i>	<i>poll</i>	<i>reach</i>	<i>delay</i>	<i>offset</i>	<i>disp</i>
=====							
*chronos.iut-bm.	193.52.61.106	2	64	376	0.00037	0.237784	0.00310

ntpd> quit
\$>

sudo

- installation de sudo

On le télécharge avec un apt-get install ou à travers un dselect.

- configuration

Le fichier est /etc/sudoers ou /usr/local/etc/sudoers suivant le déroulement de l'installation.

On nous recommande l'utilisation de visudo qui permet de vérifier la syntaxe du fichier pour son édition mais un simple vi fera bien sur l'affaire.

test ALL = (ALL) ALL

ou un

test ALL = NOPASSWD: ALL ← pour ne pas avoir besoin de saisir le mot de passe

Ainsi, l'utilisateur a tous les droits de root, il peut même faire un « sudo su » afin d'avoir directement une console root.

Liens Internet et sources

rsync

<http://samba.anu.edu.au/rsync/>
<http://sunsite.dk/info/guides/rsync/rsync-mirroring.html>

fam

<http://oss.sgi.com/projects/fam/>
voir en priorité les pages de manuel de fam et imon, la FAQ

imon

<http://oss.sgi.com/projects/fam/imon.txt>

Réplication en temps réel avec fam et imon

<http://www.tldp.org/linuxfocus/Francais/March2001/article199.shtml>

Documentation sur DNotify

<http://nst.dsi.a-star.edu.sg/mcsa/lxr/linux/source/Documentation/dnotify.txt>

NBD

<http://nbd.sourceforge.net/>
<http://www.xss.co.at/linux/NBD/>
<http://www2.linuxjournal.com/lj-issues/issue73/3778.html>
<http://www.it.uc3m.es/~ptb/nbd/>

La technologie RAID

<http://linas.org/linux/raid.html>

Table des matières

Sommaire	- 0 -
Introduction	- 2 -
1. Présentation du sujet	- 3 -
1.1. Explication de l'intitulé du sujet	- 3 -
1.2. Buts de ce projet	- 3 -
1.2.1. Buts pédagogiques	- 3 -
1.3. Cahier des charges.....	- 3 -
1.4. Attentes et apports de ce projet	- 4 -
1.4.1. Organisationnel.....	- 4 -
1.4.2. Technique	- 4 -
2. Rapport Utilisateur	- 5 -
2.1. Présentation du problème	- 5 -
2.2. Solutions possibles	- 6 -
2.2.1. Rsync.....	- 6 -
2.2.2. imon	- 6 -
2.2.3. DNotify	- 6 -
2.2.4. fam.....	- 7 -
2.2.5. NBD	- 7 -
2.2.6. Le RAID	- 8 -
3. Rapport technique	- 9 -
3.1. Solutions non testées.....	- 9 -
3.1.1. Rsync (remote synchronization)	- 9 -
3.1.2. DNotify (directory notification).....	- 10 -
3.2. Solutions testées.....	- 10 -
3.2.1. NBD (Network Block Device).....	- 10 -
3.2.2. imon (inode Monitor) et fam (File Alteration Monitor).....	- 12 -
3.3. Les stratégies alternatives	- 14 -
3.3.1. imon + fam + NBD	- 14 -
3.3.2. DNotify + fam + NFS.....	- 15 -
3.3.3. NBD + RAID logiciel.....	- 15 -
3.4. Problèmes rencontrés	- 16 -
3.4.1. Niveau technique	- 16 -
3.4.2. Niveau organisationnel	- 17 -
Conclusion	- 18 -
Annexe 1: code de fam_mirror	- 19 -
Annexe 2: Installation et configuration de SSH	- 22 -
Annexe 3: Installation de NTPD et sudo	- 23 -
Liens Internet et sources	- 24 -
Table des matières	- 25 -

