Réplication de données sur plate-forme hébergeant des applications métiers (NSS Gen2)

Rapport de stage de fin d'étude

Ritchen Rémy
2008-10-31

Remerciements

Je tiens tout d'abord à remercier Gregory MAMAN, chef du service Aircraft Information System, responsable de mon stage au sein de Sagem Défense Sécurité, pour m'avoir donné la possibilité d'effectuer ce stage, qui restera ma plus intéressante et enrichissante expérience en entreprise à ce jour.

Je tiens également à remercier Rémi LEFEBRE, ingénieur développement logiciel bas niveau, pour m'avoir plus particulièrement encadré techniquement, pour m'avoir encouragé sans relâche à approfondir mes connaissances.

Quelques mots pour Nicolas FUGIER, avec qui j'ai pu partager mes découvertes et qui n'a jamais hésité à me suggérer des pistes complémentaires, à Jean-François HOLZ, mon voisin de bureau, qui a contribué à renforcer l'atmosphère détendue dans laquelle j'ai eu la chance de travailler.

J'ajouterais un remerciement pour Camille LECOUVEY, aux services Ressources Humaines de Safran, qui fut mon premier interlocuteur et qui m'a aiguillé vers ce stage.

Je finirai cette section avec un remerciement pour Laurent CABARET et Dominique PIEDNOIR qui m'ont suivi au sein de l'Ecole Centrale.

Sommaire

Remerciements 2
Sommaire 4
Introduction 5
Contexte et enjeux pour l'entreprise 6
1. Contexte du stage 6
2. Enjeux pour l'entreprise 7
Organisation du travail, moyens 8
1. Planning, adaptation aux contraintes 8
2. Moyens 8
Etat de l'art 9
1. Architecture matérielle 9
2. Technologie de Réplication de données 11
3. Technologie d'Exportation de disques 16
4. Technologie combinée NET RAID 20
Sélection 21
3.1. Un test de comparaison Gen1 vs Gen2 21
3.2. Choix des dispositifs à évaluer 21
Evaluation des dispositifs 22
1. Moyens 22
2. Procédures 23
3. Test de performances 26
4. Crash et reprise sur incident 38
5. Choix temporaire puis choix final 39
Choix de la solution finale, développement complémentaire 41
1. Base de travail 41
2. Objectifs de la maquette 41
3. Développement complémentaire, cas traités 41
Maquette démonstrative 42
1. Scripts de monitoring SHELL 42
2. Application visuelle, écriture en base de données 44
Livrables remis 45
1. Les rapports 45
2. La maquette et les sources 45
Conclusion 46
Bibliographie 47
Annexe 48
Annexe A : Les différents modes de RAID 48
Annexe B : Fonctionnement d'un support de données 52
Annexe C : Principe de base du système de fichiers 56

Introduction

Le but de ce stage est l'étude, la conception et la réalisation d'une maquette mettant en œuvre un système de duplication de données sur disques durs afin de rendre hautement disponible un support de données. Ce système a pour vocation une inclusion dans le cadre du projet NSS Gen2

Le projet NSS Gen2 consiste en une nouvelle génération du système embarqué NSS présent dans l'A380. Il s'agit d'un système d'information, d'une plate-forme hébergeant des applicatifs métiers. On y trouve notamment des supports de données, des logiciels de monitoring (type nagios), des serveurs web, etc… Ce système se doit d'être robuste aux pannes pouvant survenir en vol (crash total ou partiel d'une machine, crash des moyens de communication). Il doit être performant car supporte une partie des applications métiers.

Sagem Défense Sécurité a déjà une solide expérience dans ce domaine. L'entreprise a conçu et développé le système actuellement utilisé sur l'A380 ; il s'agissait du projet NSS de première génération. C'est donc avec une certaine expérience et dans le cadre d'une amélioration sur reports que doit être conçu Gen2. Un des points à améliorer suite aux retours fournis par Airbus est la duplication des données entre les différents matérielles composant le système.

Ce stage a donc été l'occasion de remettre à plat le système en effectuant :

- Un état de l'art exhaustif des solutions existantes sur le marché pouvant répondre au besoin dans le cadre des contraintes matérielles, logiciels inhérentes à la stratégie développée pour le produit et aux contraintes propres au domaine de l'avionique.
- Le choix d'une solution, le développement complémentaire et la mise en œuvre d'une maquette prouvant la capacité de l'entreprise à réaliser un tel système.

On peut donc distinguer deux activités dans ce stage.

La première consiste en une recherche globale, demandant une forte culture technique, une analyse comparative des modes de fonctionnement des solutions trouvées et la synthèse permettant d'effectuer un bon choix.

La seconde est la mise en œuvre de connaissances concernant les concepts de programmation avancée, leurs éventuels apprentissages, dans le cadre d'un développement logiciel sur une base connue et étudiée auparavant, ayant pour résultat une démonstration.

Contexte et enjeux pour l'entreprise

1. Contexte du stage

1.1. Intitulé, contexte

Le Projet Gen2 consiste en une nouvelle génération du système embarqué NSS présent dans l'A380. Il s'agit d'un système d'information devant être robuste aux pannes pouvant survenir en vol. La première génération du système utilisait un mécanisme à base de réplication par le programme rsync. Il en résultait :

- une synchronisation différée
- une intégrité des données non prouvée

C'est dans le cadre du stage qu'une étude est effectuée sur les systèmes existants de réplication de données : il s'agit de recenser, d'évaluer les solutions existantes et celle en développement, de sélectionner la plus adéquate suivant des critères de performances et de robustesses en prenant en compte les contraintes inhérentes au NSS. Enfin, et si besoin, il s'agit de compléter le système par un développement afin d'apporter toutes les fonctionnalités attendues pour Gen2.

1.2 Domaines techniques abordés

Gen2 impose des contraintes en terme de matériels, l'étude concernant cette partie ayant déjà été effectuée. On peut souligner notamment l'obligation d'utiliser des systèmes de réplication supportant la traversée d'un réseau (dispositif non local).

C'est pourquoi deux grands domaines techniques ont du être étudiés dans le cadre du stage :

- Les dispositifs de réplication (type RAID)
- Le réseau et plus précisément les partages de disques à travers celui-ci

1.3 Planning global, rapports intermédiaires

Dès le début du stage (début mai), un planning global a été établi : les grandes étapes ont été définies et leur durée ont été déterminée à hauteur de:

- Etude, recensement des solutions : 4 semaines
- Première sélection : 1 semaine
- Evaluation des solutions retenues : 7 semaines
- Implémentation : 3 mois

Ce planning a été globalement respecté, bien que légèrement décalé : la fin de la phase d'évaluation des solutions retenues a donné lieu à un premier rapport intermédiaire interne à l'entreprise, qui a servi de base pour la mise au point du présent rapport.

Des rapports intermédiaires mensuels ont été envoyés aux membres du jury représentant l'Ecole, afin que ceux-ci puissent suivre l'avancée du stage, aussi bien en terme d'organisation que du strict point de vue de la technique.

2. Enjeux pour l'entreprise

Après le succès de la première génération du système NSS, il s'agit de pérenniser un système en lui apportant des avancées significatives en terme de performance et de robustesse.

Le succès de la seconde génération ouvrira la porte à l'utilisation de ce système sur les prochaines générations d'avions, tel que l'A350 ou encore l'A400M pour Airbus.

Le stage en lui-même n'aborde qu'un point du système qui est la gestion des données. Le système NSS est beaucoup plus globale. Cependant, c'est sur ce point en particulier qu'une amélioration est attendue par Airbus pour le prochain système. L'enjeu est donc important.

Organisation du travail, moyens

1. Planning, adaptation aux contraintes

De nombreux aléas surviennent lors du développement d'un projet. Le planning initial décidé d'un commun accord avec le responsable permet de se donner un fil conducteur afin de ne pas dériver à l'excès mais le rendre rigide empêcherait toutes adaptations à de nouvelles contraintes.

Le planning global que nous avions fixé a permis de prendre en compte ces aspects. Ainsi, nous gardions un œil sur celui-ci tout en définissant les tâches à accomplir pour les semaines à venir.

Lors de ce stage, j'ai été confronté plusieurs fois à des points de blocage :

- Disponibilité du matériel de test.
- Remise en cause de la roadmap sur un développement libre
- Remise à plat d'un projet

2. Moyens

2.1 Documentation

J'ai été amené découvrir et étudier de nombreux systèmes lors de l'état de l'art. Il a donc fallu que je mette en œuvre de nombreux moyens d'informations utilisé dans le monde libre. On peut donc citer :

- Des articles techniques, sur support papier (Linux Magazine), ou sur support informatiques (documentation en ligne)
- L'utilisation des listes de diffusion, afin de surveiller les évolutions, les reports de bug et les correctifs échangés au sein de la communauté. En particulier, la liste de diffusion principale du noyau linux (LKML)

2.2 Matériels

J'ai eu la chance de travailler sur des maquettes matérielles correspondant aux matériels embarqués dans les boîtiers du NSS. La disposition de ce matériel fût quasi-immédiate mais des problèmes inhérents aux prêts de ce matériel ont provoqué une période d'indisponibilité.

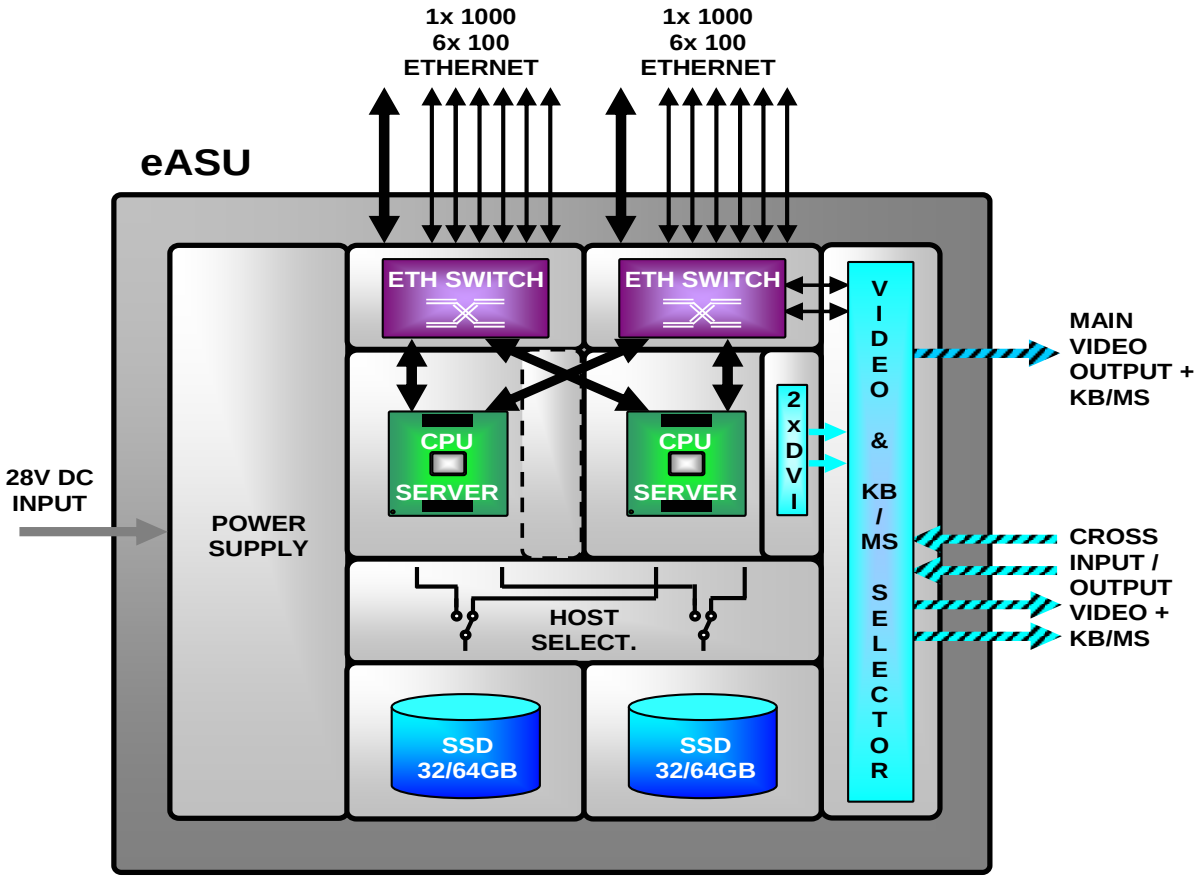
Etat de l'art

1. Architecture matérielle

1.1. Les cartes, le boîtier

Une étude a déjà été effectué par le service s'occupant du hardware, pour prendre en compte les contraintes proposées par Airbus. Cette étude donne donc une base matérielle envisagée pour le NSS de deuxième génération ; celle-ci est composée de deux boîtiers, chacun contenant une carte biprocesseurs, deux switchs, deux disques Flash ainsi qu'une alimentation commune à tous les éléments du boîtier.

Chaque carte du boîtier possède une unité de calcul (CPU) ainsi qu'un switch pour le réseau. Bien que localisé sur une carte, le CPU (ou le switch) de la carte 1 peut communiquer avec le switch (ou le CPU) de la deuxième carte (voir figure ci dessous), ce qui assure une redondance interne. Si un CPU ou un switch tombait en panne, celui de l'autre carte pourrait continuer à fonctionner tout à fait normalement.



1.2. Disques durs, organisation des blocs et système de fichiers

Deux disques durs sont présents dans un boîtier. Il s'agit de disques type Flash Disks. Le système de fichier (ou l'ensemble des fichiers) doit être capable de supporter une panne sans perte de données ou de services. Il est donc nécessaire de prévoir un mirroring des disques (au moins type RAID 1). Deux possibilités ont été étudiées :

- deux grappes de deux disques mis en RAID 1
- les quatre disques (2 disques /boîtier * 2 boîtiers) contiennent des données identiques

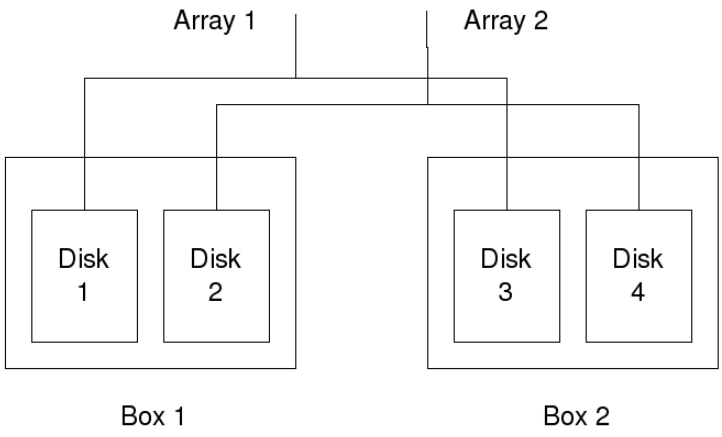
Ce système a l'avantage de garantir une redondance des données très forte : mêmes des algorithmes simple de détection/correction d'erreur type code de Hamming seront efficaces dans une telle configuration. En effet, si on envisage un crash disque, on est encore capable de corriger une erreur.

Cependant, cette solution présente plusieurs inconvénients :

* L'espace disque total disponible est celui d'un seul disque : 64G
* La synchronisation des disques va engendrer un trafic réseau non négligeable

Par rapport à la solution à quatre disques identiques, le deux fois deux grappes perd en redondance mais double l'espace disponible. Les performances globales peuvent être meilleures étant données que le trafic dû à la synchronisation sera moins important.

Il faut noter que la formation des grappes dans le cadre de cette solution ne peut se faire que d'une seule manière : en effet, nous avons vu que le système dans son ensemble doit pouvoir résister à une panne et en particulier celle de l'alimentation d'un des deux boîtiers : c'est alors deux disques qui sont indisponibles : il faut donc que les deux disques d'un même boîtier ne contiennent pas les mêmes données et que la redondance des disques se fasse entre boîtiers (voir schéma ci-dessous)



2. Technologie de Réplication de données

2.1. Les couches logicielles : de l'application au disque

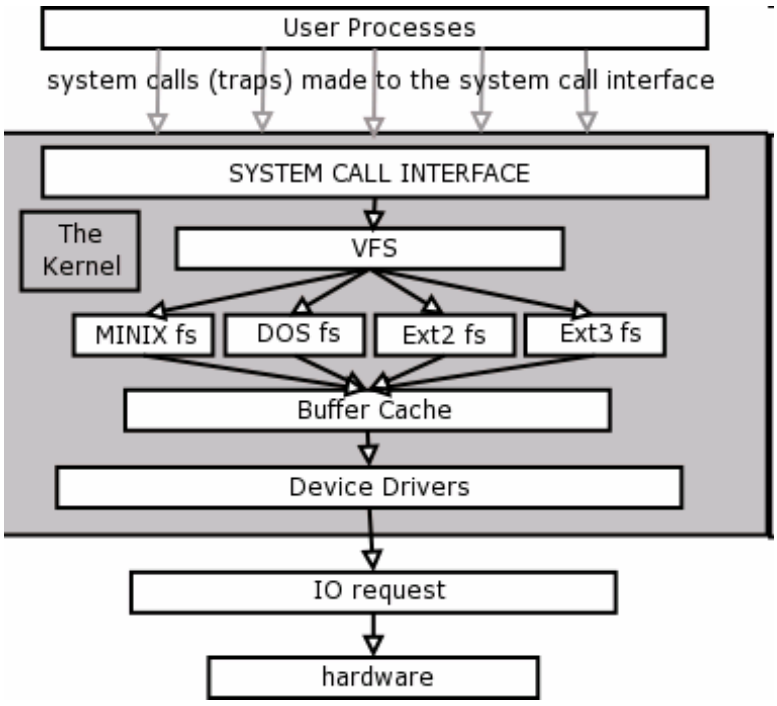
La problématique concerne donc la réplication des disques afin de pouvoir résister à une panne sans perte d'informations. Nous avons présenté brièvement deux organisations de disques possibles et bien que la terminologie classique du RAID (hard ou soft) ait été employé jusqu'ici, il s'agit avant tout de schéma de principe : le but est d'avoir des données répliquées.

Cette réplication peut se faire à différents niveaux que nous allons voir maintenant.

Pour bien comprendre les différents niveaux sur lesquels nous pouvons intervenir, il faut reprendre un schéma simplifié des différentes couches mises en jeu lors de la manipulation de fichiers (voir schéma ci-dessous).

En effet, lorsque l'on manipule un fichier, on traverse un certain nombre d'étape (couche applicative de l'OS, dans le noyau : FS, VFS, drivers des périphériques) avant d'arriver à l'écriture physique sur le disque. Il est possible d'introduire des mécanismes de réplication de données au niveau de chacune de ces couches.

Nous n'allons pas ici citer de façon exhaustive tous les systèmes de réplication existants ; il s'agit avant tout de comprendre à quelles niveaux dans le système (Hard + Noyau + soft) il est possible d'agir.



2.2. Le RAID (Redundant Array of Inexpensive Device)

2.2.1. Introduction au RAID

Le RAID est avant tout une description de la manière dont on peut répartir les données sur les disques. Il en existe plusieurs types, dits hard ou soft. Cette qualification montre le niveau auquel ces algorithmes sont implémentés.

Le RAID hard se fait au niveau du matériel, à l'aide d'une carte contrôleur spécifique, équipée la plupart du temps d'un BIOS : cette carte va présenter au système d'exploitation les arrays enregistrés dans sa mémoire comme étant des disques classiques : toute la charge due aux algorithmes sera donc cachée au CPU de la machine et sera traitée par la carte contrôleur. Ainsi, on a un système totalement indépendant et transparent pour le soft. C'est un système que l'on retrouve très communément dans les entreprises dans les baies des serveurs de fichiers.

Il existe autant d'implémentation du RAID que de constructeurs et bien évidemment, les sources de ces contrôleurs ne sont pas publiées, ce qui rend la compatibilité entre contrôleurs et métadonnées de l'array assez désastreuse.

Le RAID soft utilise les mêmes algorithmes, mais ils sont implantés dans le système d'exploitation. On parlera essentiellement du RAID soft en parlant de celui implanté dans le noyau Linux via le device mapper. Le device mapper est la brique permettant de construire des disques durs virtuels, suivant le même concept que celui utilisé en mémoire : le réadressage. Avoir une implémentation directement dans le système provoque quelques inconvénients ; notamment, cela provoque une charge CPU supplémentaire (overhead).

.

En comparaison avec le raid matériel, le raid logiciel offre plus de souplesse (pas de crash de contrôleur matériel, possibilité de récupérer les disques sur tout système Unix indépendamment du matériel installé) mais amoindri les performances du système - en théorie - puisque des instructions au niveau du CPU sont nécessaires. Cependant, le niveau de performance des cartes RAID n'est pas toujours du premier ordre ce qui explique que pour des cartes RAID matérielles bas de gamme et parfois milieu de gamme, on constate que le RAID soft est aussi performant.

Une description des modes RAID les plus courants peut-être trouvée dans l'annexe A. Parmi ces nombreux modes quelques uns ne sont pas compatibles avec l'architecture envisagée :

- Le RAID 0 est à exclure, étant hors sujet par rapport à la haute disponibilité.
- Le RAID 5 est à exclure pour le moment (et avec lui 3 et 4) en raison de l'alimentation commune à un boîtier. En effet, le système doit être robuste à une alimentation défaillante qui rend indisponible deux disques. Hors, le RAID 5 n'est robuste qu'à la panne d'un disque.

2.2.2. Compatibilité du RAID avec Gen2 :

Il a été impossible de mettre en jeu un niveau de redondance uniquement matériel, en raison de la localisation des disques dans deux boîtiers distincts. Il ne semble pas exister sur le marché des cartes RAID qui présenteraient des connectivités type Ethernet qui permettraient de construire des arrays avec des disques connectés à des cartes distinctes.

De plus, une carte contrôleur RAID dans un boîtier n'apporterait rien en terme de redondance, étant donné que la carte serait un « single point of failure ». Il est possible de mettre deux cartes contrôleurs RAID dans chaque boîtier ; cependant cela complexifie le système, augmente son coût (monétaire, mais également en consommation électrique ou en dégagement thermique) pour obtenir une redondance qu'il est a priori possible d'obtenir à un autre niveau. De plus, une alimentation défaillante provoquerait tjrs la perte du système dans la configuration 2 fois 2 grappes.

Le RAID Soft dans sa déclinaison RAID1 est compatible avec Gen2 : les applicatifs du NSS écrivent leurs données sur des partitions spécifiques, distinctes de celle du système.

2.2.3. Le RAID Soft du noyau

La gestion des périphériques blocs s'est standardisée en respectant un découpage en couches. Le RAID soft intervient dans la couche dit du device mapper à travers des modules noyaux, dont le principal est md-mod. Ce module active la gestion des périphériques RAID. Sa configuration se fait à travers des outils d'administration en espace utilisateur : mdadm. Son développeur, Neil Brown, était à l'époque insatisfait des outils alors disponible (les raidtool). Il est à noter que c'est toujours Neil Brown qui maintient le paquet mdadm et qu'il est extrêmement actif, notamment sur la liste de diffusion linux-raid@kernel.org.

Mdadm est un outil simple d'administration du raid mais on peut également modifier certaines options d'une array directement dans le sysfs[1] (/sys/block/mdXXX/), ce qui s'avère intéressant dans le cadre d'un ensemble de scripts additionnels.

Mdadm permet de créer, manager, supprimer, surveiller des arrays en quelques commandes. Ils supportent de très nombreux mode RAID, dont le RAID1 qui nous intéresse. Une array est un ensemble de disques (réels ou logiques) sur lesquels se trouvent des superblocs qui précisent l'organisation des différents disques physiques. La création d'une array n'est rien d'autre que l'écriture des superblocs.

[1] Voir Annexe C

Pour qu'une array soit accessible, il est nécessaire de l'assembler, c'est-à-dire de regrouper les différents disques en un périphérique spécial mdX. C'est ce périphérique qui servira de point d'accès à l'array

La table de bitmap :

Il s'agit d'une table décrivant les blocs désynchronisés lorsqu'un disque est inactif ou en mode fail. Lorsque le disque revient, seul les blocs taggués dans le bitmap sont synchronisés, ce qui réduit le temps pendant lequel le système est dégradé.

L'utilisation de cette table de bitmap entraîne l'ajout de taches période qui provoque un overhead. Le principal intérêt de cette table est de limiter le temps de resynchronisation après un passage en mode dégradé. Il faudra se poser la question de l'utilité de cette table suivant la fréquence d'évolution des données et la taille des partitions.

On peut également jouer sur le paramètre bitmap-chunk qui définit la taille du bloc surveillé par un bit du bitmap. Bien sur, plus ce paramètre est important, moins la taille de bitmap est grande, moins elle est efficace, mais cela change considérablement l'overhead.

Le write-mostly et le write-behind:

Le write-mostly permet de spécifier à la création ou à l'ajout d'un périphérique dans une array que celui-ci ne doit pas être utilisé pour la lecture si cela est possible. La politique par défaut du RAID 1 est de lire sur le disque dont les têtes de lecture auront à subir le minimum de déplacement. Non seulement ce comportement n'est pas justifié avec des disques Flashs, mais en plus il ne prendrait pas en compte les autres limitations aux performances comme le réseau. Cette option permettra donc de tagguer les disques distants.

Le write-behind permet quand à lui de spécifier un nombre maximum de requêtes sur disques qui pourront être exécuté de façon différée. C'est donc un paramètre qui joue sur le caractère synchrone des écritures sur les disques (qui sont à débit distincts, de part la nature local ou distante d'un disque).

Concernant les synchronisations après panne, on trouve beaucoup d'options intéressantes dans le sysfs :

- possibilité de marquer un disque comme défaillant
- possibilité de borner les débits estimatifs lors des synchronisations
- possibilité de forcer une synchronisation partielle (si bitmap), une reconstruction totale, de l'arrêter.
- possibilité avec un bitmap de changer l'état de synchronisation d'un bloc.

2.3. BTRFS : un système de fichier avec RAID inclus

La couche « Système de fichier » a pour rôle de structurer le disque dur afin d'y « ranger » les fichiers. De très nombreux systèmes de fichiers existent (fat, ntfs pour windows, ext2/3, reiserfs, xfs pour Unix, et bien d'autres…). C'est le système de fichier qui va notamment établir le lien entre l'arborescence classique et la localisation sur le disque des fichiers (voir Annexe B).

C'est très récemment que de nouveaux systèmes de fichiers capables de gérer directement la redondance sont apparus. On peut citer BTRFS (release v0.16 sortie début août !) couplé avec le projet Oracle CRFS (encore en version alpha…).

J'ai procédé régulièrement à des tests de BTRFS, avec les versions v0.14, v0.15, v0.16 ainsi que de nombreuses versions issus des dépôts instable. Je ne présenterai dans la suite aucun résultat lié à BTRFS car aucun n'a été concluant (de nombreux plantages, des kernel panics, quelle que soit la version utilisée). Le projet n'était pas à un stade assez avancé pour entrer dans le cadre de l'étude.

Le projet reste intéressant mais au cours de ces 3 mois de veille sur le projet, j'ai relevé :

- Des changements trop fréquents de structure de l'arbre du FS
- Ce n'est que tout récemment que d'autres développeurs participent au projet
- La roadmap pour la version 1.0 prévu au quatrième trimestre est ambitieuse, peut-être trop au vu de l'avancement du projet ces trois derniers mois.

De plus, rien ne garantit à terme que le RAID sur les données (pas les métadonnées) proposé depuis la version v0.14 soit maintenu : BTRFS a vocation à être intégré au noyau. Or, proposer du RAID au niveau FS remet en cause la structure en couche bien défini au sein du noyau depuis de nombreuses années (drivers/ device mapper (raid soft, lvm) / VFS / FS).

2.4. Les programmes de synchronisation en env. utilisateur

Les applications de sauvegarde sont des mécanismes au dessus de l'OS. C'est ce que l'on trouve couramment pour faire des sauvegardes. Ils ne sont pas destinés à faire de la réplication en temps réel, encore moins pour entrer en jeu dans des dispositifs de haute disponibilité. En effet, le logiciel n'effectue la copie que lorsque vous le lancez et le lancer à chaque modification de fichiers s'avèrerait très gourmand en ressources.

On peut citer rsync, logiciel fonctionnant suivant le schéma client/serveur, qui permet de faire des copies de répertoires, y compris à travers un réseau. On peut également citer dd, un outil Unix qui fait des copies par bloc (de fichier/ partition voire disques durs) en local.

Le niveau applicatif pose un certain nombre de problèmes à priori. Comment avoir une réplication en (quasi) temps réel ? Comment redonder le système de base (si le besoin est exprimé dans la suite du projet) ? L'incompatibilité n'est pas déterminée pour le moment ; mais des solutions communes type rsync, dd, etc… ne permettent pas de répondre à ces problématiques.

De plus, on peut s'attendre à des performances moindres par rapport à des dispositifs plus bas niveau (donc plus proche des périphériques blocs).

3. Technologie d'Exportation de disques

3.1. ATA over Ethernet (AoE)

AoE est un protocole développé par la société Brantley Coile Company. Il consiste à empaqueter des instructions ATA dans l'Ethernet (layer 2). Les spécifications de ce protocole sont extrêmement courtes (une dizaine de pages), ce qui montre sa simplicité.

Il est important de souligner qu'AoE agit uniquement sur la couche Ethernet. Par conséquent, le protocole n'est pas routable, et les annonces ainsi que les découvertes de partage sont diffusées par broadcast. La sécurité d'AoE repose donc sur le sous réseau et éventuellement la création de VLANs. Il n'y a pas besoin de routeurs pour faire fonctionner le protocole, il passe à travers de simples switch.

Cependant, il est souligné dans la spécification que ce protocole ne doit pas être considéré comme fondé sur des connexions : chaque message envoyé est considéré comme étant unique et peu fiable. La spécification définit également 5 types d'erreurs possibles (c'est seulement en cas d'erreur qu'une réponse est envoyée). Il est donc parfaitement possible qu'une trame se perde.

Le module AoE :

Le noyau Linux a inclus un module (aoe) permettant de se connecter à des partages de périphériques bloc par AoE. C'est une solution devenue mature et maintenue par la communauté du Libre.

Des outils de configuration sont disponibles dans le paquet aoetools. Ils permettent de lancer des découvertes de disques manuellement (par broadcast), de lister les partages et leur état, de tester l'accessibilité d'un partage (aoeping), et de repasser un partage en up suite à un crash réseau.

Il faut noter que la connexion à un partage est automatique : on ne contrôle la mise à disposition d'un périphérique bloc que par vblade et une fois ce partage effectué, *toutes* les machines étant sur le même sous réseau avec le module aoe chargé le verront et créeront un chemin dans /dev/etherd/.

Le paquet aoetools est mal conçu dans Ubuntu Hardy : afin de fonctionner correctement, il faut installer manuellement les règles udev nécessaire à la création des périphériques blocs dans /dev. On trouve également des scripts shells à exécuter dans les sources du noyau.

Vblade :

La déclaration de partage se fait à l'aide d'un programme agissait en environnement utilisateur (le programme vblade). Ce programme est bien maintenu (dernière release (la 18) le 24 juillet 2008 disponible sur le site : http://aoetools.sourceforge.net). De plus, il est packagé dans la distribution de Gen2 (Ubuntu Hardy), en version 14 ce qui garantit son maintien pendant les années à venir.

L'exécution de ce programme en environnement utilisateur provoque une perte de performance par rapport à un traitement directement par le noyau. Un module vblade avait été développé pour la version 2.6 du noyau par un développeur libre mais il n'a pas été maintenu sur les noyaux les plus récents.

De même, un projet est hébergé avec les aoetools qui a pour objectif de permettre un multipartage avec un seul thread : kvblade. Ce projet est en version alpha et à l'abandon.

Il faut souligner que vblade permet de nommer soi-même les partages en les identifiant à l'aide de deux chiffres : le shelf (pour le numéro de la machine) et le slot (pour le numéro de partage pour la machine X). Ceci facilite bien évidemment la sélection d'un bloc particulier et évite tous les problèmes relatifs à des changements de noms suite à des crashs ou des reboots.

A l'aide d'outils en environnement utilisateur, on définit les disques à partager (coté serveur, avec vblade) et on réalise la découverte des annonces (coté client, aoe-discover). Cette découverte peut-être automatique, le module aoe envoyant des broadcasts sur le réseau pour cela.

AoE empêche l'utilisation du disque en accès concurrent, par un mécanisme de TAG (busy, etc…). Par contre, il est tout à fait possible via des scripts de déplacer des partages par AoE, voire de récupérer le disque en local.

3.2. ISCSI

ISCSI est un protocole intervenant au même niveau qu'AoE. Il encapsule/décapsule des instructions SCSI dans du TCP/IP au niveau du noyau. Le protocole est donc routable, ce qui permet de l'utiliser sur internet. Par contre, le nombre de couches réseau utilisé est plus important, ce qui provoque un overhead et donc des performances a priori moindre par rapport à AoE. La sécurité se fait par authentification/autorisation de compte.

C'est donc dans le principe une technologie proche de AoE. iSCSI autorise plus de possibilités (notamment en raison du routage possible) mais est d'autant plus lourd : à titre de comparaison, les spécifications de iSCSI font 257 pages alors que celle de AoE n'en font qu'une dizaine.

ISCSI est composé de deux applicatifs : open-iscsi et iscsitarget.

Open-ISCSI :

Il est packagé dans Ubuntu Hardy (version 2.0) et les modules adéquats sont présents dans le noyau de base. Il y a un outil d'administration (iscsiadm) qui permet de lister les partages, de faire les découvertes, de se loguer, ou encore de récupérer l'état d'un partage en cours

ISCSI n'utilise pas de processus de découverte automatique des périphériques mais un système de session. Ainsi, on peut scanner une machine pour connaître les devices disponibles, puis on se logue sur les périphériques qui nous intéressent. Ce mécanisme de session facilite la gestion des partages disque puisque rien n'est fait sans commande de l'utilisateur.

La configuration des partages est aisée et permet de choisir des noms explicites (machineX.appliY par exemple). Cependant, l'ouverture d'une session sur un partage X.Y entraîne la création d'un device SCSI classique ayant pour chemin d'accès un sd[a-z]. Il est important de souligner que ce nom n'est fixe. Ainsi, cela pourrait provoquer des erreurs d'un reboot sur l'autre. Il est possible de définir un id unique par partage, ce qui facilitera la résolution du point précédent.

ISCSITarget :

Il est également packagé dans un des dépôts d'Ubuntu Hardy (universe) dans sa version 0.4.15 (la dernière release disponible est la 0.4.16, sortie en Mars 2008). L'outil se compose d'un module noyau et d'un daemon (ietd). C'est dans la configuration de ce daemon que l'on déclare les partages.

La configuration de ces partages offre des options intéressantes :

- Possibilité de définir des login/mots de passe globaux ou par partage
- Possibilité d'écrire en DIRECT_IO
- Possibilité de limiter la taille des trames
- Possibilité d'inclure des digests pour sécuriser le transfert (en terme d'intégrité)
- Possibilité de définir le nombre de threads kernel pour un partage

3.3. NBD et ses déclinaisons (Network Block Device)

NBD est une technologie très légèrement plus haut niveau qu'AoE ou ISCSI. Elle consiste à partager un périphérique bloc entre deux machines, en créant un protocole de communication spécifique (alors qu'AoE comme ISCSI encapsulent des commandes disques). Il s'agit d'une technologie standard, inclue dans le noyau Linux depuis plusieurs années et déjà éprouvée.

Les composants de NBD sont un serveur (nbd-server), un client (nbd-client) tous les deux disponibles dans Ubuntu Hardy sous forme de paquets, ainsi qu'un module directement inclus dans le noyau.

Les tentatives de superposition NBD + RAID furent toutes des échecs : la machine plantait à la création d'une array. Les quelques recherches que j'ai pu faire pour expliquer cela montrent que d'autres personnes ont rencontré ce problème. De plus, il semblerait que NBD n'autorise pas la reconnexion d'un device présent sur du RAID lors d'un crash réseau. C'est ce qui a motivé une des déclinaisons de NBD.

Un développeur indépendant a fait une version dérivée de NBD nommé ENBD (enhanced network block device) avec des fonctionnalités le rendant plus aptes au RAID, tel la reconnexion automatique du device à la reprise du réseau. La dernière version stable 2.4.34 (mars 2008) n'était pas compatible avec des noyaux récents. Une nouvelle release 2.4.35 vient de sortir, que j'ai très rapidement testé : les performances sont inférieures à ISCSI, aussi bien en débit max qu'en ratio charge/débit. Considérant que le maintien de cette version n'est pas assuré, le caractère non standard de la technologie, et ces performances, il n'était pas nécessaire d'aller plus loin sur ce programme d'export de périphérique bloc.

D'autres versions de NBD existent, non adapté au RAID : ANBD, DNBD, GNBD (très orienté cluster)…

3.4 Le nouveau DST

Fin août, un nouveau produit a fait son apparition (voir l'ancienne description du produit au 4.2). Le développeur de DST a décidé de revoir les objectifs de son produit (voir l'ancien au 4.2). Il s'agit d'une réorientation de DST vers un système de partage de périphérique par bloc, type NBD, AoE, ISCSI… Les possibilités de duplication de données qu'il comportait ont été arrêtées pour laisser place aux dispositifs déjà existants dans le noyau (ie, le device mapper et mdadm).

Le projet restait intéressant puisqu'il s'agissait d'exporter les périphériques en utilisant un mode par transactions avec des barrières. Un transfert par transactions consiste à mettre à jour les métas donnés lorsque l'écriture des données est terminée. Cela revient à réadresser l'endroit où se trouve tel fichier sur le disque. Ainsi, en cas de coupure brutal, on conserve des données intègres. Les barrières permettent de taguer une série de blocs pour décrire l'endroit jusqu'où on est sur que les données sont à jour.

Avec ces deux supports, DST s'avérait être un dispositif d'export de disque très intéressant sur le papier. C'est ce qui a justifié une ré-organisation du planning afin de pouvoir l'évaluer.

4. Technologie combinée NET RAID

4.1. DRBD (Distributed Replicated block device)

D'autres systèmes de réplication au niveau bloc apparaissent, comme DRDB. Ces systèmes sont en général libre (licence GPL ou assimilé) et sont donc potentiellement intéressants puisqu'ils peuvent être complétés si nécessaire par un développement annexe.

DRBD se compose d'un ensemble d'outil d'administration (drbdadm), d'un daemon et d'un module noyau. Il est un équivalent tout intégré de ISCSI+mdadm : il effectue du RAID 1 par réseau via TCP/IP pour échanger des messages entre les deux périphériques. Il est possible comme pour mdadm de limiter le débit du lien, mais la précision de ce dispositif est bien moindre.

DRBD permet également d'exploiter une même array sur deux machines distinctes en même temps (mode primay-primary) : il intègre des dispositifs d'accès concurrent pour peu que le FS soit adapté (type GFS ou OCFS). Cependant, dans la prochaine sous version annoncée, on peut voir : « Introducing mechanisms to better deal with temporary network failures for devices in primary-primary mode » On peut comprendre aisément que les merges sont difficiles a effectuer lors que les deux systèmes ne peuvent communiquer.

La version stable de DRBD est la 8. C'est celle-ci qui est packagée dans Ubuntu Hardy. L'équipe de développement de DRBD n'annonçait jusqu'à très récemment aucune roadmap pour les versions suivantes : il considérait qu'il devait avant tout stabiliser la version 8.

La version 9 apportera la possibilité de gérer dans une array plus de deux disques, ainsi qu'une meilleure intégration avec le device mapper (avec à terme la possibilité que DRBD soit inclus dans le noyau ?).

4.2. L'ancien DST

DST est un système développé par Evgeniy Polyakov. Il n'est pas du tout inclus dans le noyau, n'est maintenu que par son unique créateur. Il avait l'avantage, dans sa version v0.14 -présenté comme stable et fonctionnel- de proposer un système de tout inclus, type DRBD. Il se trouve que les tests effectués en patchant un noyau 2.6.22 ne se sont pas avérés concluant.

De plus, le développeur a annoncé mi-août une remise à plat de son travail et une réorientation du produit. Il servira à l'export de périphériques block uniquement. C'est pourquoi je ne présenterais pas de résultats liés à cette version dans la suite du rapport. Il est cependant intéressant de signaler que cette décision du développeur a eu des répercutions importantes sur mon travail et mon planning.

Sélection

1. Un test de comparaison Gen1 vs Gen2

Un premier test basique a été fourni à Airbus pour prouver la plus grande rapidité d'un système à base de RAID (Gen2) par rapport à celui de Gen1 (rsync). Il s'agissait de mesurer le temps d'exécution pour créer un fichier identique sur deux machines.

Gen1 : création d'un fichier de 6Go à l'aide de dd depuis /dev/zero par bloc de 1M, puis synchronisation sur un disque distant à l'aide de rsync.

Gen2 : création d 'une array RAID1 composée d'un périphérique local et d'un périphérique distant, puis on mesure le temps nécessaire à la création sur l'array d'un fichier de 6Go par bloc de 1M.

Résultat : 13min47s pour Gen2 (soit 7,4MB/s) contre 15min (6,8MB/s) pour Gen1. De fait, ce test a prouvé le gain en performance brute d'un système à base de RAID. De plus, il faut souligner que rsync utilise la compression de données. Hors un fichier généré à partir de /dev/zero ne contient que des 0 et est donc extrêmement compressible : les performances réalisés dans ce test par Gen1 correspondent donc au meilleur cas.

Ce test nous a amené à ne pas évaluer de façon poussée les solutions de réplication en temps différé, en environnement utilisateur (type rsync).

2. Choix des dispositifs à évaluer

J'ai effectué de premier test rapides pendant l'état de l'art afin de ma familiariser avec les outils, de mettre au point des procédures et des scripts de tests. J'ai également pu commencer à éliminer des solutions sur la base de tests en débit maximum.

Ainsi, il était prévu d'évaluer de façon plus poussée :

- AoE +mdadm
- ISCSI + mdadm
- DRBD, bien que semblant déjà en retrait
- DST dans sa nouvelle version

Les annonces du développeur de DST ont mis en attente les tests concernant ce projet. J'ai également inclus un test avec ENBD, suite à une première série de test décevant en multi arrays. Il est probable qu'une mauvaise configuration du réseau ait provoqué ces résultats décevant car la deuxième série fût bien meilleure.

Evaluation des dispositifs

1. Moyens

1.1. CPUStat

CPUStat est un petit programme en C permettant de calculer les charges processeurs globales à partir du fichier /proc/stat. On trouve dans ce fichier des données sur la répartition des taches processeurs entre différentes catégories, notamment :

- user : les processus s'exécutant en environnement utilisateur sans privilège
- sys : les processus kernel ainsi que les processus s'exécutant en tant que root
- iowait : les processus mis en attente d'une ressource (proc, ram, disque, réseau)
- idle : la mesure des processus qui pourraient encore être rajouté à l'exécution sans saturer les ressources

CPUstat récupère ces données à intervalle régulier et calcule alors les pourcentages d'occupation. Ce programme a été développé en interne et j'ai été amené à le modifier afin de l'adapter à mes besoins ; j'ai augmenté la finesse de l'échantillonnage en effectuant des mesures toutes les demi secondes. J'ai également inclus les mesures d'iowait.

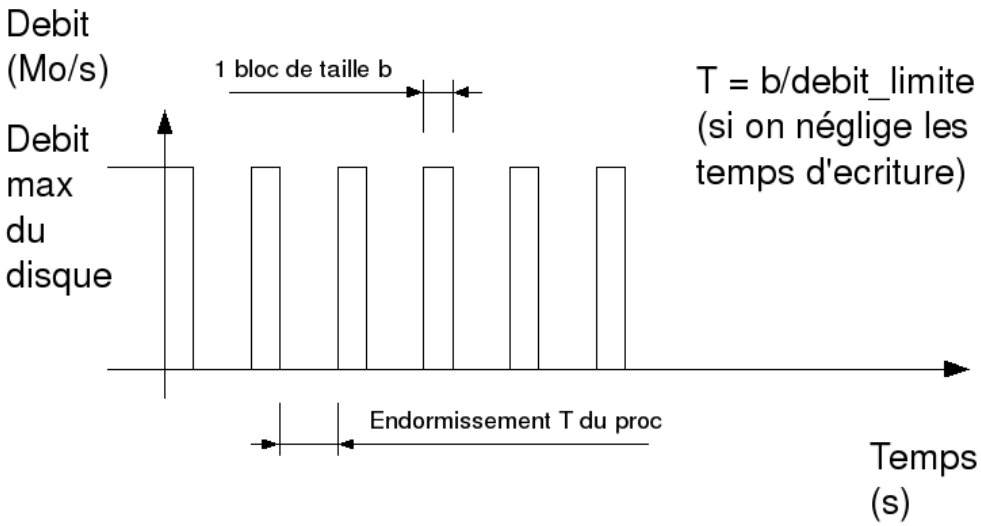
1.2. DD et DD Customisé

DD est un outil d'écriture par bloc. Il a été principalement utilisé pour calculer les débits à saturation – à condition que ce ne soit pas lui-même qui sature le processeur… Afin de limiter autant que possible cela, il écrivait depuis /dev/zero des blocks de données de 100K ou de 1M (bs). Afin que le calcul du débit soit précis, le count (nombre de blocs écrit) était réglé afin que la mesure dure plusieurs minutes : ainsi une erreur de quelques secondes n'a qu'un faible impact sur le calcul du débit.

DD Customisé est un outil que j'ai écrit en C afin de combler un manque de DD : le contrôle du débit. En effet, j'avais besoin de tracer des courbes de charges CPU en fonction des débits d'écriture. Or je n'ai pas trouvé de programme permettant d'écrire avec des débits contrôlés (comme un logiciel de gravure par exemple). J'ai donc écrit ce programme sur le modèle de DD avec une option de débit maximal.

Lorsque que cette option est active, le programme calcule un temps d'endormissement, va alterner l'écriture linéaire de quelques blocs et un endormissement par usleep du processus. On obtient une écriture par créneau, ce qui entraîne des saturations ponctuelles en écriture (donc en ressource), lesquelles vont être moyenné sur l'échelle de temps correspondant à mes mesures.

Modèle :



1.3. IOStat

IOStat permet de mesurer le nombre de requêtes, leur taille ainsi que d'autres éléments sur des périphériques bloc. Il a servi à vérifier que les requêtes était bien équivalentes à l'entrée sur une array (/dev/mdX) et à l'entrée sur les devices composant cette array (/dev/sda[1-2]).

On a ainsi eu la confirmation que tout se passait comme attendu, à savoir qu'AoE et ISCSI procédaient bien à une simple encapsulation.

2. Procédures

2.1. Environnement de tests

Les tests ont été effectué sur une Ubuntu Hardy, avec des noyaux de distribution (2.6.24-[16-19]-generic) dans la mesure du possible. Les deux couples de deux machines ont pu être utilisés :

- 2 desktop classiques, doté de P4 2,6GHz, 1G RAM et 2GHz 500Mo
- 2 maquettes, doté de centrino à deux cores 1,5GHz, 2G mémoire.

Les périphériques bloc utilisés pour construire les arrays sont des volumes logiques de LVM de 500Mo. Cela introduit une légère dégradation des performances obtenues et il convient donc de considérer ces résultats comme étant non optimisés.

Pour se donner une idée de la dégradation des performances avec LVM (test hors maquette) :

* 35,5 Mo/s en écriture direct sur sda9 (moyenne sur 5 séries)
* 31,5 Mo/s en écriture sur un LV contenu dans un VG occupant tout sda9 (donc même endroit sur le disque)

Soit une perte de performance d'environ 11%. Par la suite, j'ai donc enlevé la couche LVM lors des tests sur maquette afin de m'approcher de conditions réelles.

Le système de fichier utilisé lors des tests est ext3 (en dehors des tests sur BTRFS). Le système de fichier possède quelques paramètres[2] dont la taille d'un bloc. Par défaut, la taille d'un bloc est de 4ko mais l'outil mkfs peut - suivant la taille du périphérique sur lequel il écrit – modifier cette taille par défaut ce qui m'a causé quelques difficultés pour interpréter des résultats. La solution à ce problème consista à forcer la taille du paramètre.

Les charges CPU relevées dans le cadre des tests avec maquette correspondent à une moyenne entre les deux processeurs.

Le Réseau :

- Les tests hors maquettes ont été effectués sur des cartes en 100Mbits avec un MTU à 1500, en liaison direct.
- Les tests maquettes ont été effectués sur des cartes 1GBits, avec un MTU à 9216, soit en connexion directe, soit à travers un switch Gigabit standard.
- Le test à 4 machines s'est fait à travers le switch, avec MTU standard à 1500. Je n'ai pu tester des jumbo-frames car une des cartes réseau ne le supportait pas.

Attention :

Il conviendra de faire attention dans les résultats exposés : il ne faudrait pas comparer 2 graphes effectués sur des couples de machines distincts.

De plus, les résultats obtenus hors maquette permettent de voir des tendances évolutives mais ne doivent en aucun cas être pris comme des valeurs absolues.

[2] Voir Annexe

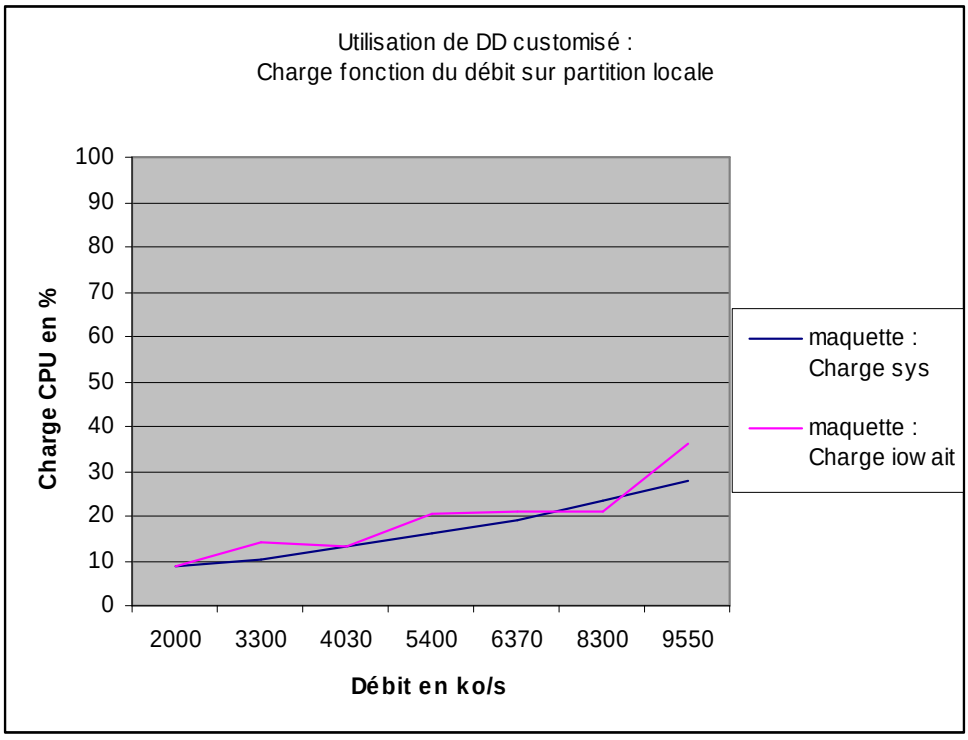
2.2. Le test par écriture direct

Après avoir écrit le dd customisé, j'ai procédé à quelques tests à vide afin de vérifier que ce dernier n'interférerait pas ou peu dans les mesures de charge que j'allais effectuer.

Le graphe ci-dessous montre que sur des débits très petits, le DD customisé va fausser les mesures de charge. Il m'a cependant permis de faire quelques tests avec des profils de charges. En effet, lors d'une utilisation réelle, un grand nombre d'applications sont amenés à utiliser le disque, mais suivant des débits différents.

J'en ai tiré la conclusion que l'on atteignait des débits maximum proches mais légèrement en deçà des débits maximums présentés par la suite dans les cas de resynchronisation : les performances sont moindres de quelques points.

Par la suite, j'ai donc fait essentiellement des tests par resynchronisation.



2.3. Le test par resynchronisation

Le test par resynchronisation consiste à exploiter les possibilités offertes par les outils s'occupant de la partie RAID (mdadm et drbdadm). Ces deux outils permettent de déclarer des périphériques comme étant inaccessibles pour simuler des pannes et limiter grossièrement les débits de synchronisation. J'ai donc exploité ces possibilités pour contourner le problème de perturbation des mesures de charge engendré par le dd customisé. La procédure consistait à :

- déclarer le périphérique distant comme étant inaccessible
- écrire sur l'array (donc sur le disque local) une quantité X de donnée
- ré inclure le disque distant dans l'array, ce qui lance une synchronisation.

Cette procédure provoque la lecture des blocs modifiés (grâce au bitmap) sur le disque local, l'écriture sur le périphérique par le réseau. Par rapport à une écriture directe, on remplace une écriture sur le disque local par une lecture, on ne passe pas par les couches hautes du RAID ou équivalent mais on passe par la couche synchronisation et utilisation du bitmap. En terme de charge processeurs, cette procédure est légèrement moins coûteuse en sur la machine hébergeant le RAID et équivalente sur la machine fournissant les partages. Cependant, un rapide test d'écriture sur une array composée de deux périphériques locaux montre que la charge imputable au RAID est négligeable

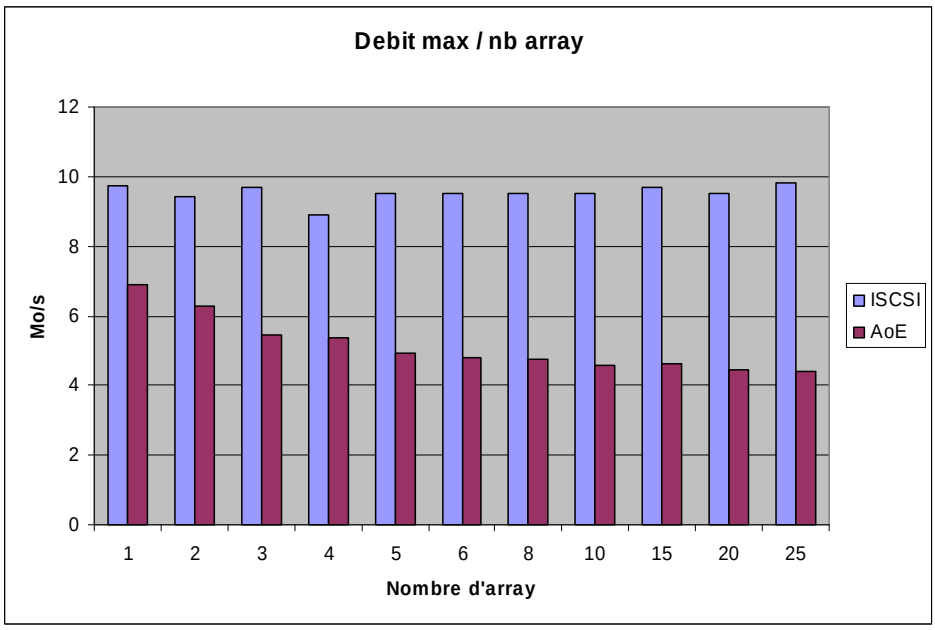
3. Test de performances

3.1. Test hors maquette

3.1.1 Influence du nombre d'array RAID sur le débit maximum en resynchronisation

Dans ce test, X arrays ont été créé, à partir d'une partition locale et d'une partition partagée par le réseau via AoE ou ISCSI. L'objectif était de voir l'impact sur les performances maximal et la gestion par le noyau de la présence d'X threads pour le RAID, et pour les clients/serveurs des partages.

On constate dans ce test qu'ISCSI résiste mieux qu'AoE à la montée en charge – en terme d'arrays – et conserve un débit proche du goulot d'étranglement qu'est le réseau en 100Mbits (~ 12Mo/s).

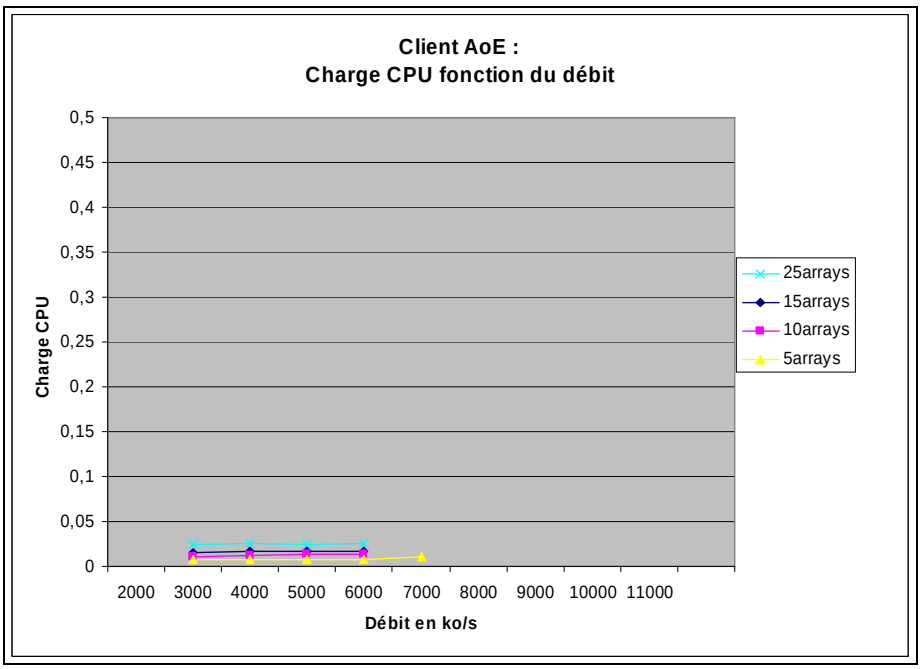


3.1.2 Les charges CPU fonction du nombre d'arrays et du débit

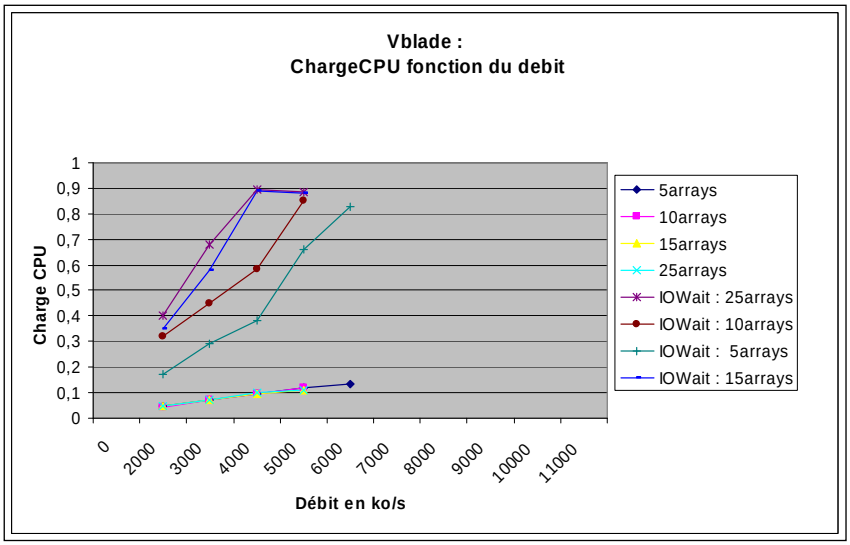
ATA over Ethernet :

Afin de comprendre la perte de débit maximal que l'on voyait sur les tests précédents, j'ai procédé à une multiplication des tests avec AoE, en limitant le débit grâce aux variables du RAID, et en faisant varier le nombre d'arrays.

Si dessous, que la partie client d'AoE ne consomme que peu de charge CPU système de façon indépendant du débit ET du nombre d'array. Bien que non représenté sur le graphique, l'IOWait coté client était négligeable.

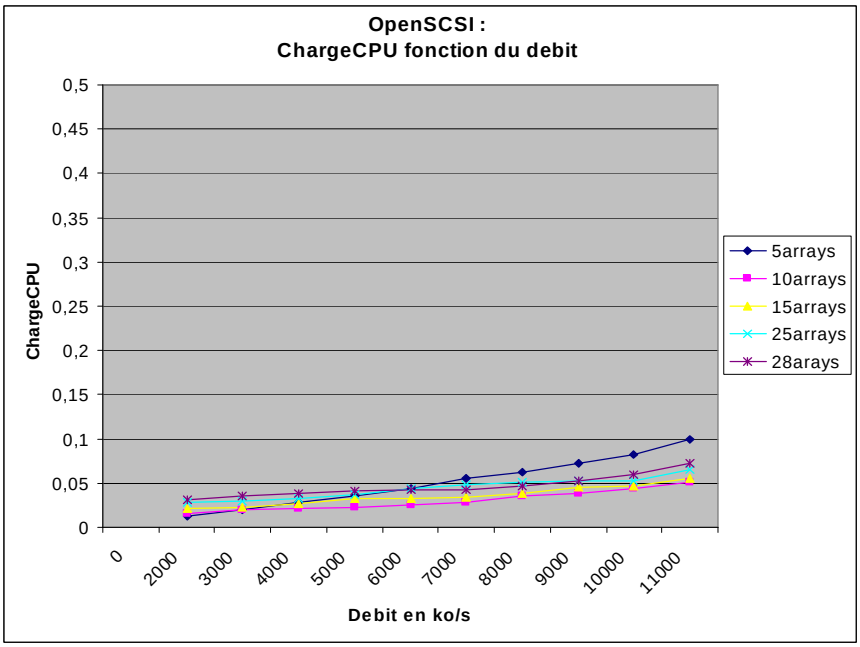


Sur la partie serveur (vblade), on constate que la charge système augmente quasi linéairement fonction du débit et est peu influencée par le nombre d'arrays. Elle reste de plus raisonnable (< 20%). Par contre, l'IOWait croit de façon très inquiétante en fonction du débit et provoque une saturation (sys+iowait) du CPU très rapidement. C'est cette croissance de l'IOWait qui explique la saturation en débit maximal plus basse que le goulet du réseau.

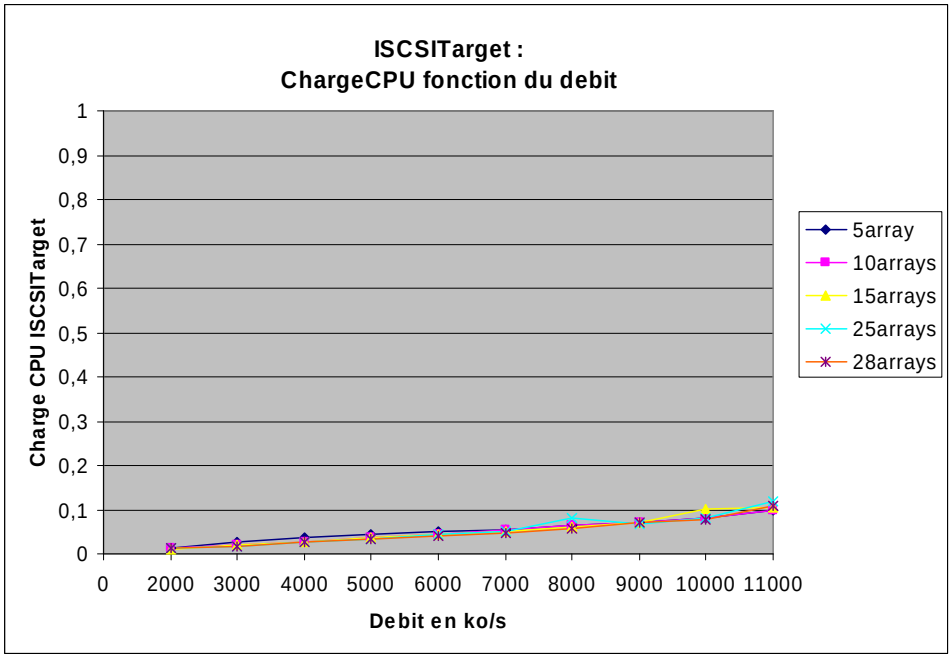


ISCSI :

J'ai procédé au même test avec ISCSI afin de voir son comportement en terme de charge. On peut voir que la partie client d'ISCSI se comporte linéairement par rapport au débit. La charge reste raisonnable (< 10%), le nombre d'array a peu d'influence. L'IOWAIT est négligeable



Sur la partie serveur, on fait le même constat : influence quasi-linéaire du débit sur la charge, et influence négligeable du nombre d'array sur les résultats.



Premières conclusion :

De ces premiers tests hors maquette, on serait tenté de conclure à une meilleure gestion d'ISCSI par rapport au nombre d'array. Des recherches rapides sur Internet montrent que de nombreuses personnes ont rencontré un problème de performances lors de multiples partages avec AoE. L'explication semble être la mauvaise gestion lors de la présence de nombreux vblade en exécution.

Cependant, ces résultat doivent être confirmés sur maquette (l'architecture processeur n'étant pas la même). De plus, on ne prévoit pas ici le comportement sur un réseau Gigabit, ce dont pourrait disposer Gen2.

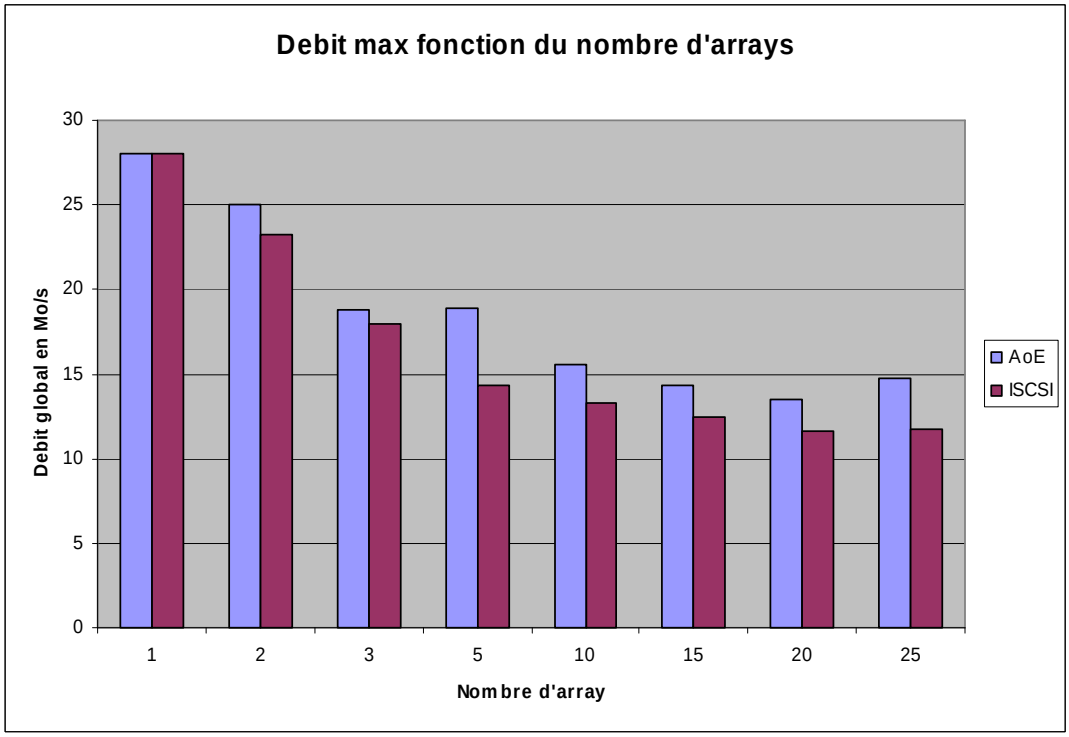
3.2. Test sur maquette (Gigabit, jumboframes)

Dès que la maquette fût à nouveau disponible, j'ai procédé à la même série de test que précédemment, cette fois sur maquette. Les résultats obtenus m'ont amené à réinclure des tests rapides sur DRBD et ENBD.

3.2.1 Influence du nombre d'array RAID sur le débit maximum en resynchronisation

Même procédure que précedemment. Les débits sont plus importants (en raison du Giga, le réseau ne limite plus les performances), tout près des performances du disque pour une array. On découvre une influence du nombre d'array sur les performances max. AoE se révèle plus performant qu'ISCSI ici, avec un débit max quasi-constant sur une plage d'array entre 10 et 25. Dans cette plage d'array, on atteint des débits max entre 12 et 15 Mo/s, probablement grâce aux jumbo-frames.

Une première série de ce test avait donné un résultat inverse (ISCSI > AoE), avec des débits max bien moindres. La configuration entre ces deux séries n'a pas changé (les mêmes scripts ont été utilisés), en dehors du MTU qui pouvait être mal réglé. J'ai donc décidé d'éliminer la première série, moins fiable.



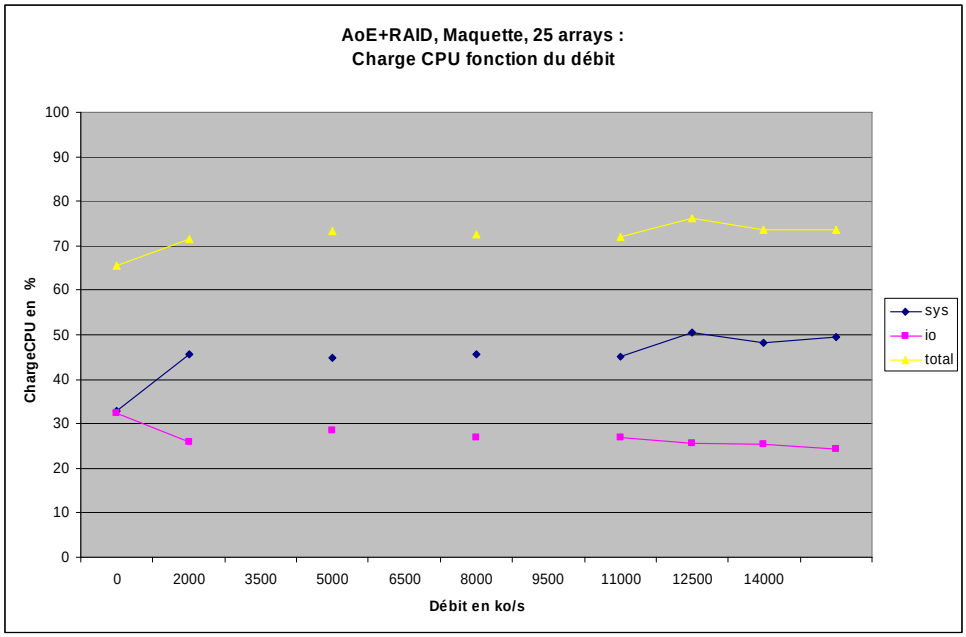
3.2.2 Influence du débit sur la charge CPU (en version 25 arrays)

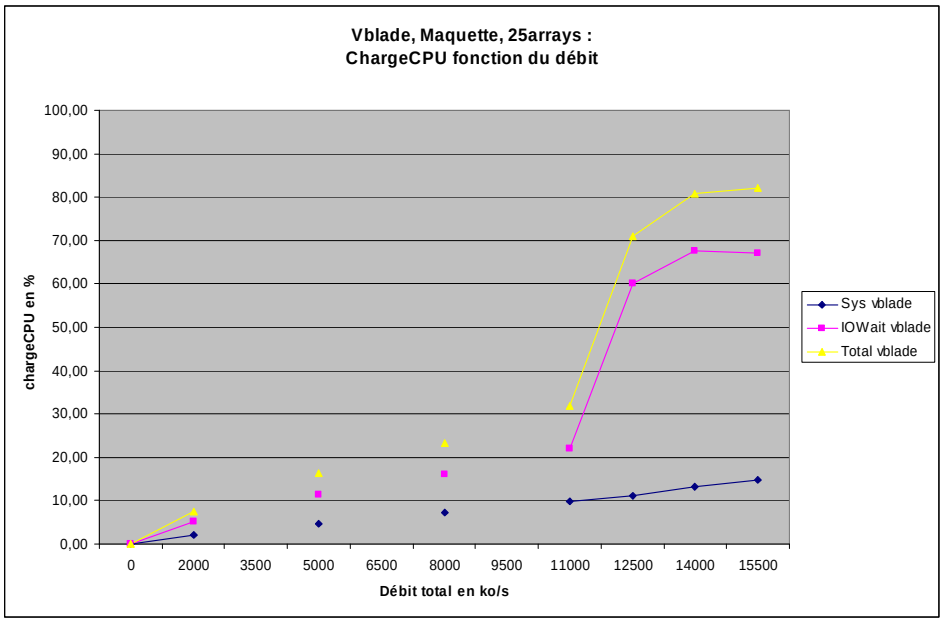
De la même façon, j'ai procédé à une autre série, soupçonnant le MTU d'avoir été mal réglé. Afin d'aller plus vite, je n'ai pris que peu de mesures et uniquement sur le pire des cas, à savoir 25 arrays.

ATA over Ethernet :

La machine qui héberge AoE et le RAID est très chargée, Cependant, on constate qu'à l'état idle (débit=0), la charge est déjà importante, en raison des tâches cycliques du RAID (bitmap). La synchronisation en elle-même ne provoque pas de charge indécente.

Coté vblade, on constate une charge raisonnable jusqu'à 11Mo/s de débit cumulé. Au delà, vblade IOWait énormément pour une raison qu'il reste à découvrir.



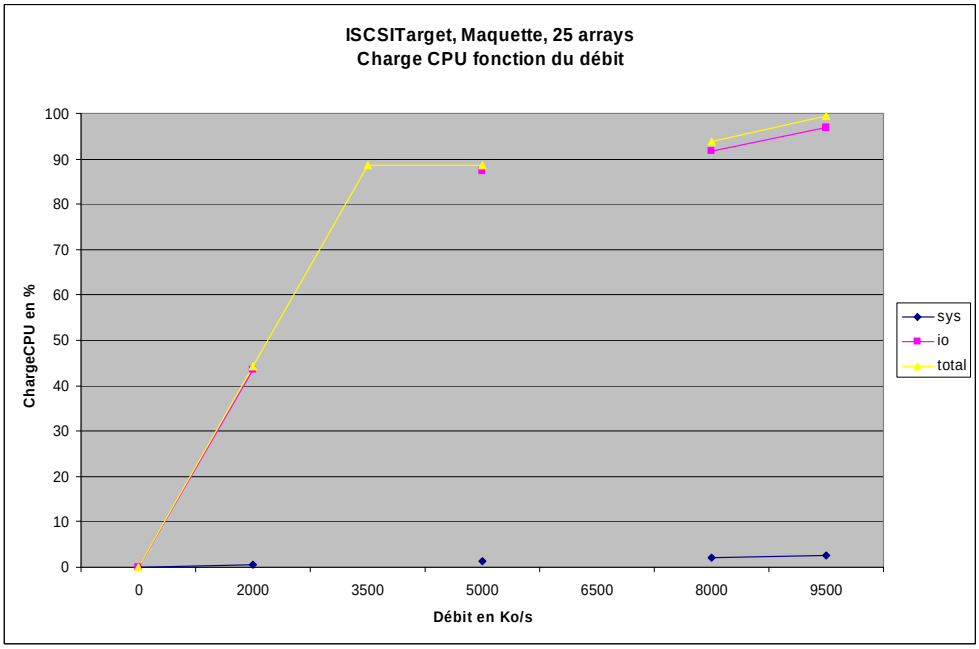


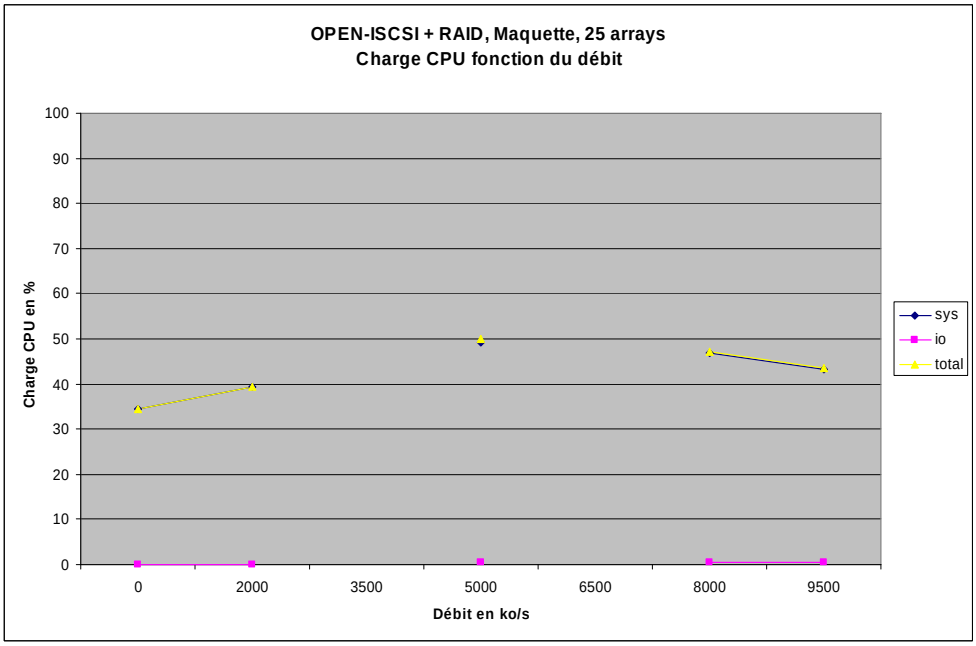
ISCSI :

Le même test que pour AoE (maquette, 25 arrays, même limitation en resynchro) donne :

Une charge système raisonnable coté ISCSITarget mais un IOWait qui explose dès que le débit global atteint quelques Mo/s. Cela fait saturer le CPU et limite les performances

Une charge importante coté RAID+Open-ISCSI, due pour l'essentiel aux taches cycliques du RAID. En effet, la charge n'évolue que très peu avec le débit

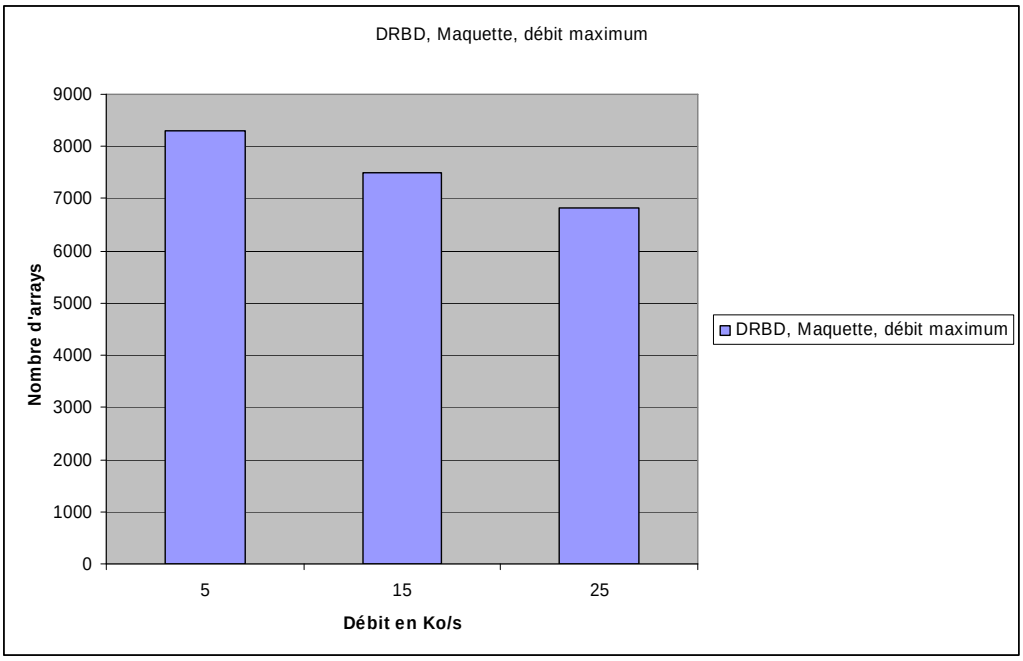




DRBD

Je n'ai pris que quelques mesures, étant donné que les performances étaient très en dessous des autres systèmes :

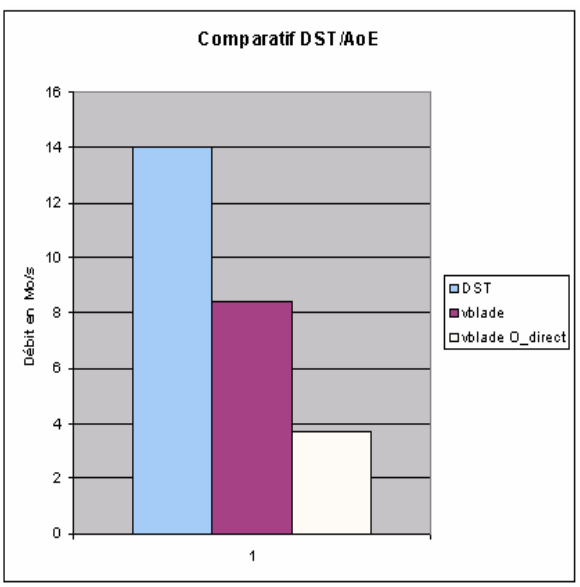
- Les débits maximaux ne dépassent pas 10Mo/s malgré le Gigabit et les jumbo-frames
- Ces débits sont limités par l'explosion des charges (sys et io). En effet, dès 5 arrays DRBD, on a déjà à 8Mo/s une charge sys à 30% et un iowait à 65,5.

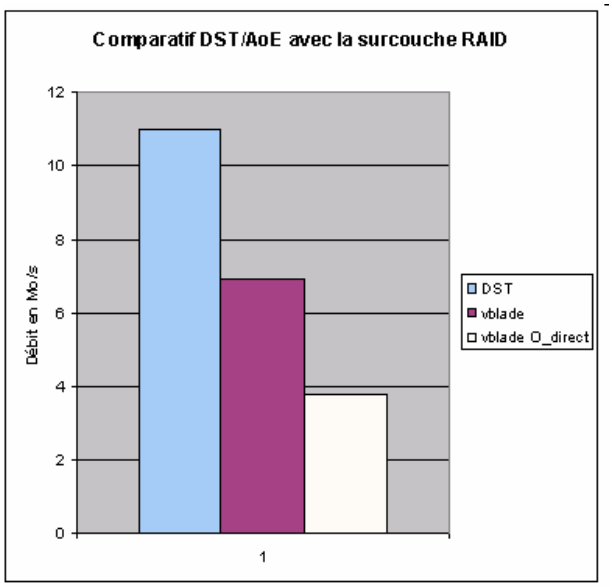


Le nouveau DST :

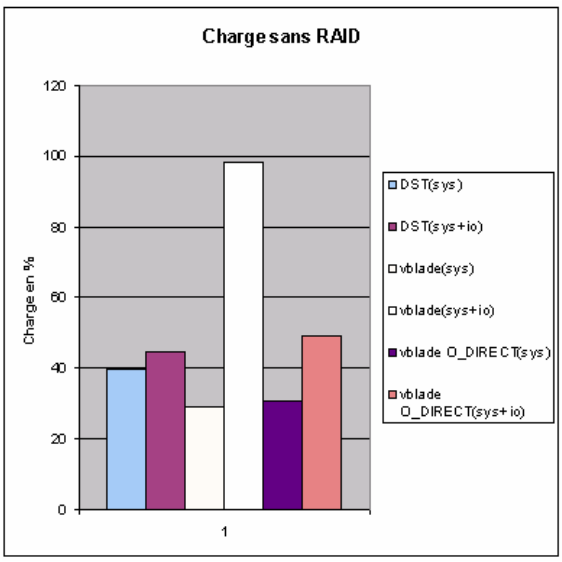
Ces tests ont été les derniers que j'ai effectué, dès la sortie du produit. Il s'agit d'un comparatif avec AoE (que j'avais pré-sélectionné) sur le débit maximum, ainsi que sur les charges systèmes à ces débit max

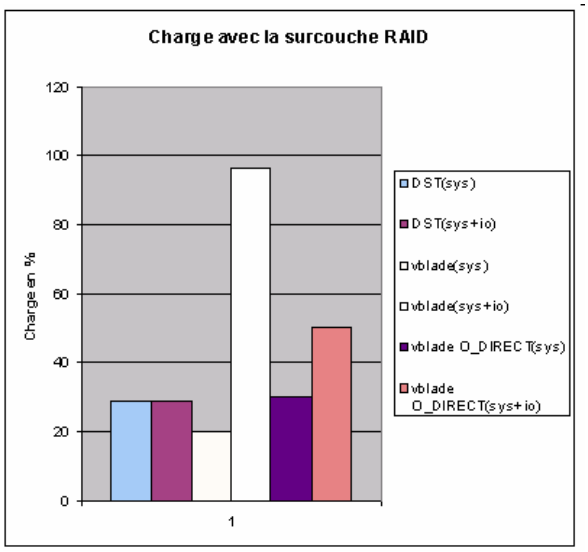
Test en débit maximum :





Charge correspondante :



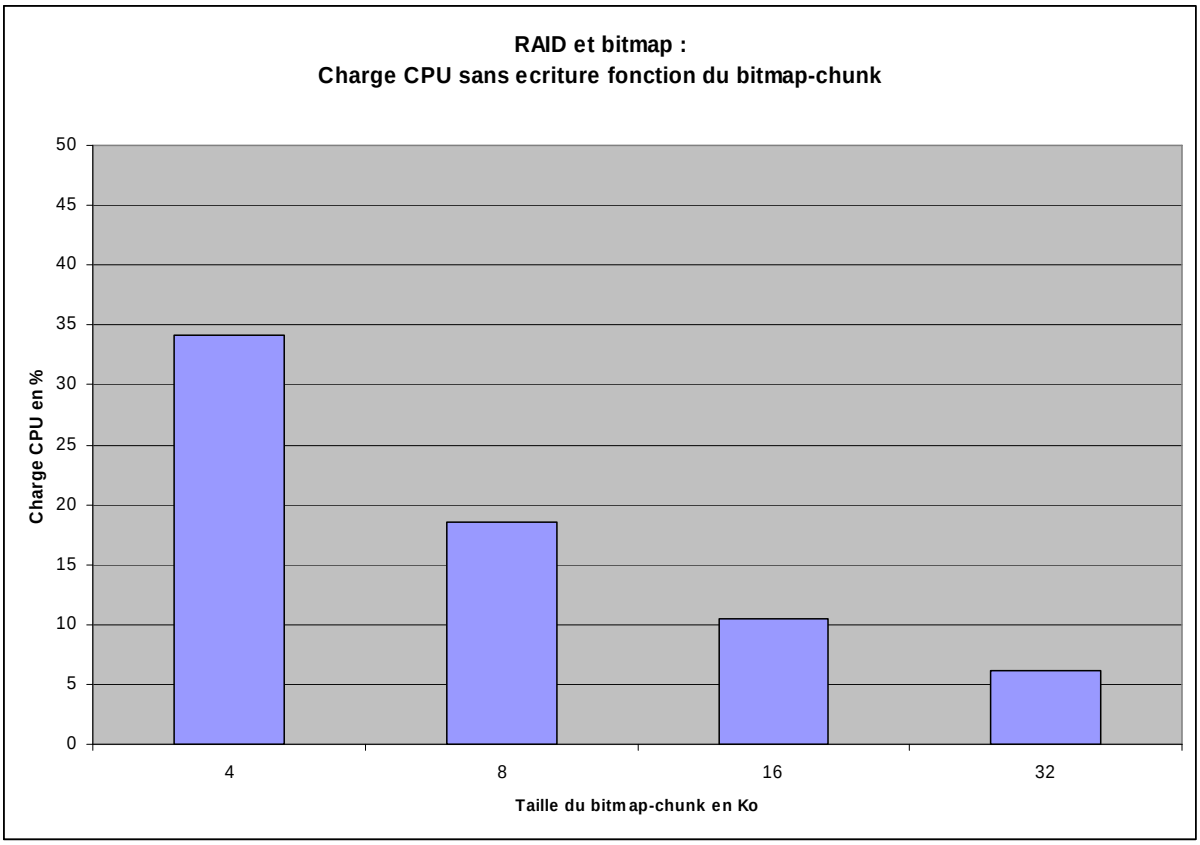


Les résultats ont été sans appel. DST est non seulement beaucoup plus performant, mais en plus consomme moins de ressources.

3.2.3 Influence de la taille du bitmap

Pour ce test, j'ai utilisé CPUstat pour mesurer des charges au repos sur un montage ISCSI + RAID1. Le paramètre est la taille du bitmap (--bitmap-chunk). La valeur par défaut de ce paramètre est de 4Ko. (un bloc de 4Ko modifié => 1 bit modifié dans la table de bitmap). Le bitmap-chunk doit être un multiple de 2.

L'influence de ce paramètre n'est pas négligeable.



Les développements libres connaissent des mises à jour et des changements assez fréquents. Après recherche pour comprendre le phénomène ci dessus, j'ai constaté qu'il s'agissait de tâches cycliques provoquées par une routine du bitmap. Une modification du fonctionnement de cette routine a été incluse dans le noyau 2.6.25. Ainsi, cette routine n'est appelée que lorsque le bitmap signale un bloc désynchronisé. Au repos, on a donc avec des noyaux récents un overhead nul.

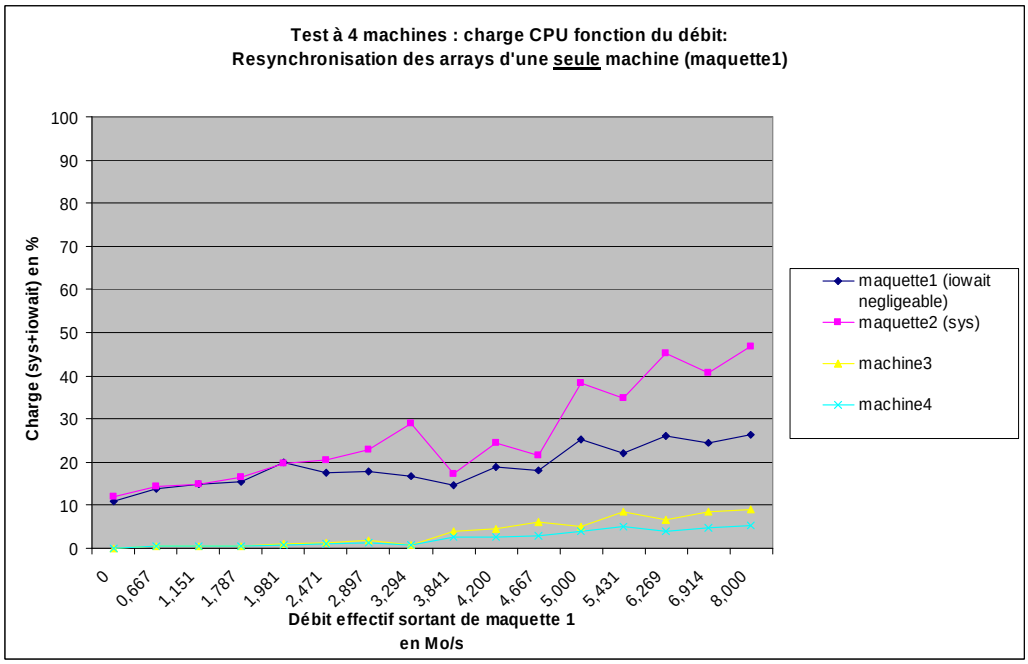
3.3. Test sur 4 machines

Après commande et récupération de cartes Gigabits, des tests sur 4 machines ont été effectués. Chaque machine possédait 28 partitions : 7 étaient 'locales' (ie utilisées directement par le RAID), les 21 autres étant partagés via ISCSI. (le choix d'ISCSI faisait suite à la première série de tests sur la maquette, où le MTU était mal réglé).

Le test a été fait en utilisant la resynchronisation, avec limite de débit. Cependant, Je n'ai pas trouvé de correspondance entre les débits limités par array et les débits que j'ai réellement relevé. Les débits présentés sur les graphes sont ceux que j'ai relevé, montre en main.

J'ai effectué 2 types de tests :

- Un test minimal où les arrays d'une machine – la maquette 1 - (soit 7 arrays) ont été désynchronisés. Ce cas là n'est que peu probable : en effet, un crash machine ou une coupure réseau sur une seule machine provoquera la désynchronisation de 28 partitions alors que dans ce test il n'y en a que 21.
  En terme de débit : c'est le trafic SORTANT sur la machine 1 soit 7 arrays* 3 disques/array * Quantité de donnée / temps de synchro total qui est présenté sur le graphique. Les machines 2 à 4 reçoivent un trafic de 7 * Q / T, soit un débit trois fois moindre



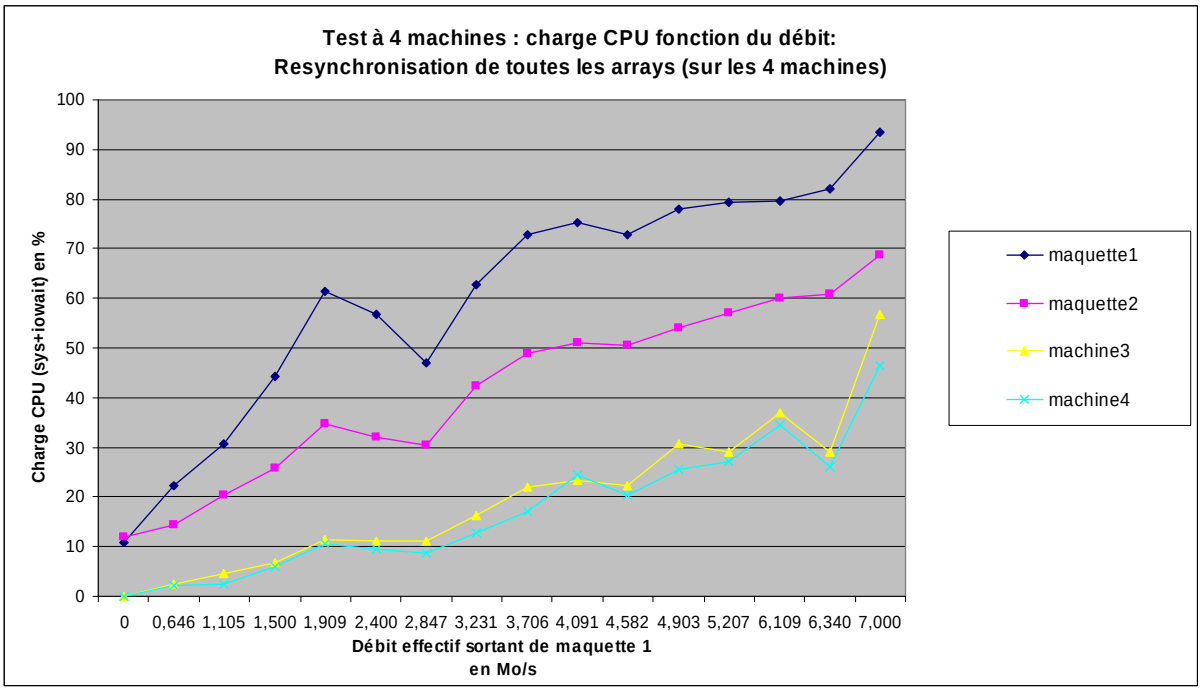
Interprétation des résultats :

Les charges notées sur le graphique correspondent à l'addition des charges sys et des charges IOWait.

La maquette 1 a une charge bien plus importante que les autres machines, ce qui est tout à fait normal puisqu'elle génère le trafic le plus important. On note également que les maquettes supportent moins bien que les autres machines la charge créé par le RAID + les partages : il est possible que le noyau gère mal les cores duo, la fréquence deux fois moindre du core duo expliquant l'écart significatif entre la maquette2 et les deux autres machines.

En terme de valeur absolue, les charges relevées sont déjà importantes : près de 45% pour un débit sortant à 8 Mo/s. On peut s'attendre à une très forte charge dans le cas du test complet.

- Un test complet ou tout le monde est désynchronisé (soit 4 * 21 partitions à resynchroniser). Dans ce test, chacune des machines émet et reçoit un trafic égal à 7 arrays * 3 disques/arrays * Q / T



Interprétation :

A nouveau, on différence les résultats des maquettes et des autres machines. On note un décalage non négligeable entre la maquette 1 et la 2 : je n'ai pas d'explication plausible à cela, les paramètres de haut niveau des maquettes étant exactement les mêmes. La seule différence notable qui existe entre les deux maquettes est le nombre de bits d'adressage mémoire : la maquette 1 est en 32bits tandis que la maquette 2 est en 64bits.

Les charges relevées sont dans tous les cas très importantes pour des débits sortant des machines à 5-7 Mo/s

Conclusion des tests à quatre machines :

Les résultats sont plutôt décevant. Les charges relevées sont beaucoup trop élevées pour les débits testés. Bien que la configuration en 4 machines soit plus avantageuse en terme de répartition de charges et de sécurité (4 disques identiques), les charges engendrées en cas de crash sont trop importantes pour retenir cette solution.

De plus, on perd beaucoup d'espace disque et on génère un trafic réseau très important.

4. Crash et reprise sur incident

4.1 Le RAID mdadm

La synchronisation n'est pas automatique au retour d'un disque. Mdadm exige que l'on enlève puis remette le disque concerné dans l'array par des opérations d'administrations (2 commandes). Cela a pour conséquence le besoin d'une application de monitoring du raid qui effectuera ces opérations d'administrations.

La resynchronisation des disques se fait à débit maximal, ces débits pouvant être bornés en écrivant dans des fichiers du sysfs. Il n'y a pas de gestion de priorité à proprement dite dans le cas où plusieurs arrays sont à synchroniser. On note cependant que :

- Les débits sont distribués équitablement entre les arrays dans les cas de non saturation
- En cas de saturation, on se rapproche d'un mode « FIFO »
- Il est possible de créer facilement un gestionnaire de priorité en utilisant la possibilité de borner les débits : il suffit alors de définir des poids.

4.2 ATA over Ethernet

En cas de crash vblade, il faut prévoir une application capable de relancer les partages. En cas de crash réseau, le disque réseau (/dev/etherd/eX.Y) passe en down au bout d'un certain timeout. Lorsque le réseau revient, il semble que le disque ne repasse pas automatiquement en up si une couche l'utilise au dessus. Par exemple, sa présence dans un array RAID bloque ce retour. Il est nécessaire de sortir le disque de l'array pour que celui-ci repasse en up, ce qu'il faudra prévoir dans le monitoring.

4.3 ISCSI

En cas de cash de ISCSI, il faut également une application capable de relancer la machine.

En cas de cash réseau, aucune opération n'est requise sur la couche ISCSI : le test effectué fût une coupure réseau. J'ai alors écrit sur les arrays, ce qui a provoqué des tentatives d'écriture sur les partages –> les disques passent en état défaillant dans les arrays. J'ai alors rebranché le réseau : il suffit de sortir/remettre les disques dans les arrays (opération au niveau du RAID donc) pour que la synchronisation se lance automatiquement.

4.4 Le nouveau DST

Une routine est déclenchée lors de la détection d'une perte de connectivité qui va se charger de procéder à la reconnexion automatique dès le lien rétabli. Aucun plantage n'a été à déplorer lors de mes derniers tests, ce qui en fait la solution la plus stable et efficace.

5. Choix temporaire puis choix final

Les différentes parties de ce rapport très technique jusqu'à maintenant vous ont présenté une synthèse des différents solutions envisagés et tests que j'ai effectué lors du stage, cependant, il convient ici de souligner que le choix de la solution s'est fait en deux temps.

J'ai, à plusieurs reprises, signalé le fait que la disponibilité de DST dans sa nouvelle version avait provoqué des changements de planning. En réalité, d'après le planning initial, le choix aurait du être effectué juste après le moment où le nouveau DST a été annoncé par son développeur. Au vu des fonctionnalités annoncées par celui-ci, il a donc été décidé d'effectuer un choix temporaire et de préparer les scripts de maquette indépendamment du choix de la technologie d'exportation de disque.

J’ai donc commencé la mise en place de la maquette en gardant trois éléments à l’esprit :

- Il faut que l’implémentation de l’automatisation du système soit générique
- AoE est la technologie préchoisie pour son potentiel d’évolution
- Il faut effectuer les tests dès la sortie du nouveau DST

Deux semaines plus tard, le produit est sorti, j’ai effectué les premiers tests que vous avez pu lire précédemment qui se sont révélés concluant. Le choix final pour la maquette s’est donc arrêté sur DST combinée au RAID Soft du noyau Linux.

Choix de la solution finale, développement complémentaire

Suite aux choix de technologie

1. Base de travail

- Exportation des périphériques blocs par DST
- Gestion de la réplication par RAID logiciel
- Surveillance de l'état du RAID logiciel par le daemon mdadm monitor

2. Objectifs de la maquette

La maquette doit permettre de faire une démonstration d'un système entièrement automatisé dans ses opérations de reprises. Les seules interventions humaines lors de cette démonstration consisteront en la simulation de panne.

NSS Gen2 est une plateforme d'hébergement qui utilise des bases de données. Il est probable que le gestionnaire de bases de données MySQL soit utilisé lors de ce projet. C'est pourquoi, il a été décidé de mettre en œuvre ce logiciel lors de la maquette

La maquette doit être résistante aux crashs d'une machine. Le système doit donc être muni d'un middleware détectant l'extinction d'une machine : HeartBeat a été choisi pour sa facilité d'utilisation. Le but n'est pas de le tester, il s'agit simplement d'un moyen de déclencher les scripts de gestion et d'opérer les bascules.

3. Développement complémentaire, cas traités

Il était nécessaire de compléter la base de travail afin d'obtenir un système entièrement automatisé. Il s'agit essentiellement de développer un ensemble de script automatisant les opérations d'administration nécessaires suite à une panne. Les fonctionnalités requises sont :

- Démarrage automatique du système, suivant toutes conditions (démarrage simultané des machines ou différé)
- Bascule du daemon MySQL automatiquement en cas de crash de la machine primaire.
- Bascule en cas de crash disque sur la machine primaire
- Ré-inclusion automatique des disques distants (coupure réseau, crash des parties DST)

Maquette démonstrative

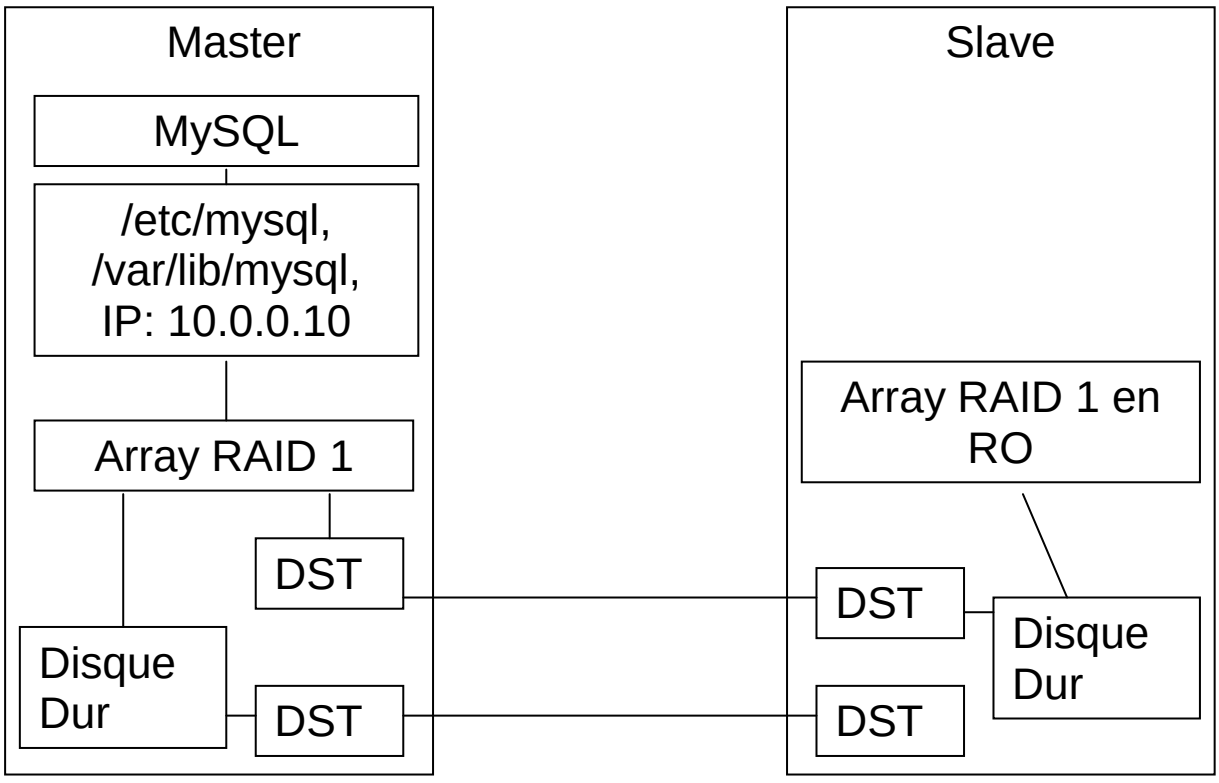
1. Scripts de monitoring SHELL

1.1. Séquence d'initialisation

La séquence d'initialisation consiste à la succession d'événement lors du boot du système. Il est nécessaire de :

- Démarrer les partages de disques à travers le réseau
- Démarrer le système RAID
- Démarrer les outils de monitoring et l'application
- Pré-démarrer la machine esclave pour accélérer les bascules.

L'état après boot de la machine est le suivant :



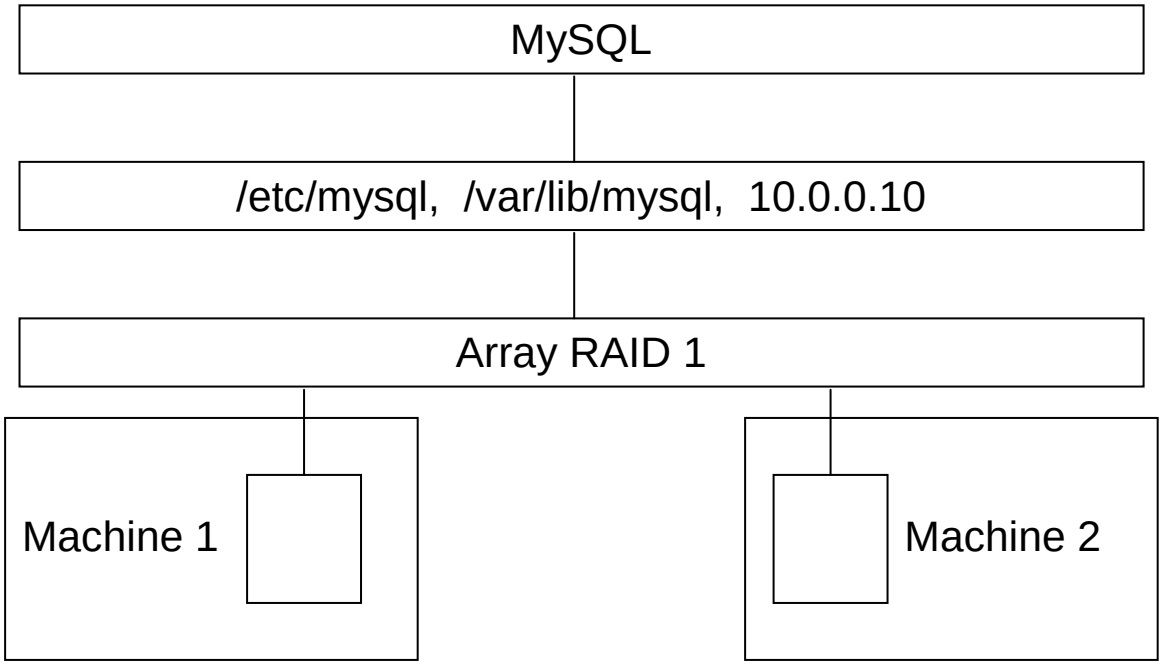
2.2. Principe de la bascule

La bascule consiste en la migration des éléments spécifiques au daemon MySQL d'une machine à une autre, à savoir :

- L'adresse IP de contact de ce daemon (10.0.0.1)
- Les fichiers de configuration (/etc/mysql)
- Les fichiers contenant les données (/var/lib/mysql)

Dans le détail, cela consiste à effectuer sur le master l'arrêt de MySQL, un umount sur la partition qui lui est spécifique, à enlever de l'array les partitions distantes et à passer l'array en readonly.

Les opérations inversent sont exécutées sur l'ancien slave, jusqu'à revenir dans la position du schéma ci-dessus, dans la mesure du possible, avec les rôles master/salve inversés.



2. Application visuelle, écriture en base de données

2.1. Approvisionnement de la base de données

Afin de faire une démonstration réaliste, il était nécessaire de mettre en place une application générant un flux de données vers la base MySQL. Le plantage du système engendre une coupure du flux de données et donc des pertes dans l'enregistrement attendu. Il fallait que l'on puisse facilement repérer celle-ci, de façon percutante, visuelle.

J'ai donc décidé de mettre en œuvre une application enregistrant une vidéo en base de données. L'œil humain est très sensible aux pertes d'images ou aux images dégradés. Une telle application permet donc de mettre en valeur le système.

2.2. Développement sur VLC : enregistrement et lecture à partir d'une base de données

J'ai donc procédé au développement de module pour le logiciel VLC qui permette de réaliser cette ambition. Le principe de fonctionnement consiste à enregistrement image par image et à recomposer à la lecture. Il a donc fallu développer 3 modules :

- Module database, assurant l'interface avec l'API C MySQL
- Module d'enregistrement, sur la base du module scène existant
- Module de lecture, par recomposition des images.

Ce développement m'a amené à entrer en contact avec les développeurs de VLC et à acquérir des connaissances sur les domaines de l'audio/vidéo. J'ai désormais une connaissance très précise sur la manière donc fonctionne l'encodage de la vidéo et de la musique, ainsi que sur la manière qui permet de multiplexer ses deux composantes afin de former les fichiers que nous connaissons dans leur format usuelle (.avi, .mkv, …).

Je soulignerais également que cela m'a permis de voir tous les avantages de la programmation modulaire. J'ai pu très facilement « entrer » dans le code - alors que celui-ci est très complexe – et développer des modules sans connaître le fonctionnement complet du logiciel. Cette structure en brique nécessite probablement plus de réflexion lors des étapes de conception mais procure une souplesse qui, je pense, apporte beaucoup lorsqu'il s'agit de répondre au besoin d'un client, à son attente spécifique. Je garde donc à l'esprit cette façon d'organiser le code et j'essayerais de la réutiliser lors des différents projets que je pourrais développer tout au long de ma carrière.

Livrables remis

1. Les rapports

En plus des nombreux échanges par mail avec mon suivi technique, Sagem a en sa possession :

- Un rapport intermédiaire concluant sur l'état de l'art écrit mi-août
- Une présentation Powerpoint qui a servi pour information interne
- Le présent rapport

2. La maquette et les sources

Sagem a en sa possession une archive contenant les sources de tous les développement effectués, ainsi que deux portables sur lesquels sont installés la maquette. Ainsi, l'entreprise peut à tout moment faire une démonstration de sa capacité à produire un système de réplication de données à travers un réseau, résistant aux pannes.

Conclusion

Le stage a donc permis la réalisation d'un système de réplication bas niveau sur le NSS, qui sera un atout non négligeable pour Sagem Défense Sécurité et lui permettra de répondre à un besoin exprimé par son client.

Ce stage s'est révélé très enrichissant. Techniquement tout d'abord, en me permettant de mettre en œuvre les connaissances acquises au long de mes dernières années d'études. Il m'a permis de les renforcer fortement dans les domaines de la haute disponibilité, de la programmation noyau et de la programmation d'application bas niveau. L'état de l'art effectué me permet d'avoir une vue globale sur ce domaine de compétence en pleine explosion. A l'heure du déploiement de systèmes embarqués dans tous les secteurs, la résistance aux pannes sera sans aucun doute l'objectif primordial des constructeurs.

Humainement, ce stage m'a permis d'évoluer dans une atmosphère très accueillante ; j'ai pu échanger avec des spécialistes et c'est avec grande satisfaction que nous avons mutuellement enrichi nos connaissances. Leurs expériences, leurs avis critiques m'ont poussé à approfondir mes recherches et j'espère qu'eux-mêmes ont pu découvrir de nouveaux produits. J'ajouterais que des petits plus tel que le partage d'un croissant pour fêter un événement facilite les échanges et permet de se découvrir mutuellement. J'ai été étonné par cela et je me permets de le souligner dans ce rapport car je suis convaincu que ces petites gestes ont un impact énorme sur la capacité des équipes à travailler ensemble.

Enfin, je terminerais sur la satisfaction que je ressens à l'issue de ce stage. Demain, l'entreprise pourra faire preuve de sa capacité à répondre à un des besoins exprimés par le client, démonstration à l'appui. Savoir que son travail sera utile et utilisé est la plus grande satisfaction que l'on peut ressentir. L'intérêt que l'équipe a pu porter à mon travail lors de la présentation et de la démonstration que j'ai faîte en fin de stage devant eux suffit à m'en convaincre.

Bibliographie

Ata over Ethernet (AoE) :

[1] Site Internet
http://support.coraid.com/

ISCSI :

[2] RFC 3720, 3721, 3722

DST :

[3] Site Internet :
http://www.ioremap.net/projects/dst

Raid soft, MDADM :

[4] Site Internet
http://tldp.org/HOWTO/Software-RAID-0.4x-HOWTO.html
[5] Site Internet
http://neil.brown.name/blog/mdadm

BTRFS :

[6] Site Internet
http://btrfs.wiki.kernel.org/index.php/Main_Page

Documentation noyau, linux :

[7] Linux Network Administrator's Guide, 2nd Edition
Olaf Kirch & Terry Dawson, O'Reilly

[8] Understanding the Linux Kernel
Daniel P. Bovet & Marco Cesati, O'Reilly

Programmation C:

[9] Le langage C
Peter Aitken & Bradley L. Jones, Campus Press

Annexe

Annexe A : Les différents modes de RAID

A.1. Introduction au RAID

Le RAID (Redundant Array of Inexpensive Disks) est un principe de fonctionnement pour les supports de données. Il consiste à créer une unité de stockage de masse à partir de plusieurs disques durs physiques (le mot physique est ici à entendre au niveau de la couche RAID : il peut s'agit de disques matériels (le plus courant), mais aussi de partition ou de disques logiques). Suivant les algorithmes utilisés pour mettre en commun ces disques, on va obtenir un système offrant une performance globale plus importante, ou bien une redondance pour assurer la sécurité des données.

Cette technologie a été créé en 1987 et depuis 1992 une fondation, le RAID Advisory Board, gère ces spécifications, ce qui permet d'uniformiser les principes des différents niveaux. Ces niveaux représentent les différents algorithmes qui sont utilisés, privilégiant soit les performances, soit la redondance, soit le coût. A ce jour, les niveaux existant les plus courants sont le RAID 0, 1, 10 (ou 0+1) qui diffèrent légèrement), le RAID 5, le RAID 6. Il existe également les niveaux 2, 3, 4, 7, ainsi que des combinaisons de ces RAID. Je ne les détaillerais pas en raison de leur utilisation marginale, ou bien en raison d'un besoin en disques supérieurs à 4 (qui est une contrainte de notre architecture matérielle).

A.2. Les différents niveaux de RAID

A.2.1. Le RAID 0

Le RAID 0, appelé Stripping (traduisez 'entrelacement') est un mode RAID utilisant deux disques durs au minimum. L'algorithme consiste à écrire les données d'un même fichier sur plusieurs disques physiques en même temps. Dans la pratique, on se donne une taille de bande, par exemple 4Ko, et on coupe le fichier en morceaux de cette taille. Le premier morceau sera écrit sur le premier disque dur, le second morceau sur le second disque, etc…

Ce mode offre des performances élevées (en terme de bande passante ainsi que de temps d'accès) ainsi que la possibilité d'obtenir un gros disque à partir de deux plus petits. Cependant, la perte d'un des deux disques entraînerait la perte de toutes les données. C'est donc un mode qui n'est pas du tout adapté pour faire de la haute disponibilité.

A.2.2. Le RAID 1

Le RAID 1, Mirroring, est un mode RAID axé sur la redondance. Il utilise deux (ou plus) disques physiques et dupliquent les données sur ces disques. Ce sont donc des disques miroirs qui sont parfaitement (en théorie) identiques (voir schéma).

Ce mode permet donc de faire de la haute disponibilité puisqu'il est robuste à la perte d'un disque. Il offre des performances en lecture intéressante mais l'écriture est ralentie. De plus, il a un coût important étant donné qu'avec deux disques de 40Go, vous n'obtenez que 40Go d'espace exploitable. Avec le RAID 0, il fait partie des premiers modes RAID qui sont apparus.

A.2.3. Le RAID 10 (0+1)

Le RAID 10 a pour but de récupérer les avantages des modes précédents en les combinant. Il utilise au minimum 4 disques ce qui le rend plus coûteux que les deux modes précédents. Cependant il en récupère les deux avantages (rapidité et sécurité).

Il permet donc de combiner les deux modes. Une subtile différence est à noter dans la combinaison des deux modes ce qui donne lieu en fait à deux niveaux de RAID possibles : 01 ou 10. Cela peut avoir des conséquences sur des architectures matérielles un peu spécifiques (ie, autre chose que 4 disques dans une unité), ce qui est le cas de notre étude. En réalité, ce mode n'est pas utilisable dans notre cas.

A.2.4. Le RAID 5.

Le RAID 5 est la solution la plus courante appliquée à des baies de disques dans les entreprises. Ce mode combine quasiment tous les avantages du RAID 0 et 1 et cela, en limitant le coût en terme d'espace disque.

Il utilise au minimum trois disques. Le principe consiste à écrire des données redondantes, des données de parité. On pourrait écrire les informations de parité sur le même disque (c'est ce que fait le RAID 3 (parité par octet) et le 4 (parité par bloc) mais dans ce cas, on aurait un disque qui travaille plus que les autres (c'est en fait un disque qui limiterait les temps d'accès de l'ensemble). Afin de pallier à ce problème, le RAID 5 consiste à répartir les informations de parités sur chacun des disques de la grappe.

Ainsi le système est résistant à une panne de disque, il offre des performances plus élevées que des disques fonctionnant séparément (aussi bien en lecture qu'en écriture). Son coût se calcule suivant une formule bien connu : (N-1) * Capacité d'un disque où N est le nombre de disque dans la grappe et sous l'hypothèse que chacun des disque soit de taille identique.

A.2.5. Le RAID 6

Le RAID 6 est une déclinaison du RAID 5 qui permet de renforcer la redondance du système. Pour cela, il utilise deux informations de parité. Le système demandera donc au moins 4 disques durs, il sera résistant à 2 pannes de disques et sa capacité sera de (N-2) * Capacité d'un disque. Les données de parité sont toujours réparties sur les disques.

.

A.3. Du RAID hard (matériel) au RAID soft (logiciel)

Nous l'avons vu en introduction cette annexe, le RAID est avant tout un concept de répartition en tel, ce qui permet une implantation de différentes manières. Les premières implantations du concept ont été matérielles à travers les fameux contrôleurs RAID. Avec les derniers développements noyaux des OS, le RAID logiciel est apparu et le développement grand public du RAID a donné lieu à l'apparition du RAID pseudo matériel.

A.3.1. Le RAID matériel

C'est donc la première implantation du RAID qui consiste à piloter le système par une carte dédiée appelée carte RAID. Cette implantation est 'constructeur' et donc le code est à la fois propriétaire et non divulgué. Les principales caractéristiques de cette implantation sont :

- L'OS est déchargé de la gestion du RAID. Il ne voit donc qu'un disque qui correspondant à la grappe. La création de l'array, le rajout de disques, etc… sont gérés à l'aide du BIOS de la carte RAID ou bien via un liveCD fourni par le constructeur. Les performances des disques sont assujetties à la performance de la carte contrôleur RAID
- La carte RAID devient un point critique (single point of failure), puisqu'elle contrôle les disques. Il faut noter que les méta données de l'array (notamment l'organisation des disques matériels) sont présentes (en général) à la fois dans une mémoire de la carte contrôleur et sur les disques durs ce qui permet, en théorie de remplacer une carte grillée. Cependant, il faut avoir la même carte, avec le même bios (par sécurité) en réserve, ce qui oblige presque à acheter le matériel en double. De plus, les métadonnées sur les disques sont parfois corrompues par la carte défectueuse, ce qui enlève tout espoir de récupération.

C’est pourquoi il faut faire très attention avec un système RAID matériel et ne pas négliger des backups réguliers. Les grands systèmes d’entreprise qui ne peuvent tolérer une panne de carte contrôleur possèdent des boîtiers contenant deux cartes qui gèrent les mêmes disques, avec deux interfaces de sortie pour supprimer le single point of failure.

A.3.2. Le RAID pseudo matériel

C’est un RAID qui vise le grand public. Il s’agit des puces RAID qui sont vendus sur les cartes mères depuis quelques années. Ces puces sont en fait des contrôleurs disques avancées qui possèdent quelques routines propres à un système RAID. En réalité, il ne s’agit que d’un RAID logiciel camouflé, qui cumule les inconvénients en terme de performance, de fiabilité (difficile de remplacer une telle puce).

A.3.3. Le RAID logiciel.

Comme son nom l’indique, il s’agit d’une implantation du RAID géré par les couches bases de l’OS (en fait, dans le noyau). De nombreux systèmes d’exploitations permettent de gérer certains modes de RAID aujourd’hui. Par exemple, Windows XP et Mac OS X gèrent le RAID 0, 1. Le support le plus complet est celui proposé par les systèmes à noyau Linux (depuis le noyau 2.6). Le RAID soft a pour caractéristique :

- Une grande souplesse d’administration (via le logiciel)
- Un système libre (pour linux), un développement/support optimisé
- Pas de single point of failure
-

Cependant, il présente aussi des inconvénients :

- Overhead (le noyau gère le système)
- Parfois impossibilité de l’utiliser sur le disque système (ou sur une partie, le /boot par exemple).
- Système logiciel donc potentiellement buggué
- Difficulté de changement à chaud des disques ou du repérage de disques défectueux.

Annexe B : Fonctionnement d'un support de données

B.1. Les supports de données

Cette partie a pour principal but de donner un support de base pour parler des systèmes de fichiers. Bien qu'il existe de très nombreux supports de données (disquette, CDROM, DVDROM, mémoire flash, disques durs, bande magnétique, etc…), je ne détaillerais que le plus utilisé (le disque dur classique) ainsi que les disques SSD envisagés pour le projet NSS Gen2. On va voir notamment que la structure matérielle de ces disques impose des conditions aux logiciels qui les gèrent (et aux systèmes de fichiers) si on veut optimiser leur utilisation.

B.1.1. Le disque dur classique

Le disque dur utilisé couramment dans les PCs est un support de masse magnétique. C'est un ensemble de disques (les plateaux) superposés. Chaque face de plateaux possède une tête de lecture/écriture qui lui est propre et est décomposée en pistes circulaires, la superposition des pistes de chacun des plateaux forme un cylindre. Les pistes sont décomposé en secteurs de disques : ce sont les blocs. Ces blocs ne font pas la même taille physique suivant la piste mais contiennent la même quantité de données : 512 octets. Nous verrons que cette unité a de l'importance dans le système de fichiers et qu'elle constitue la plus petite unité de disque dur.

Ainsi, lorsque que l'on veut retrouver une donnée, par exemple un fichier de quelques octets, on a besoin du numéro de la tête, du numéro de la piste et du numéro du secteur sur la piste. Ces informations seront stockées dans le système de fichier qui établira la correspondance entre l'arborescence et ces unités « géographiques » sur le disque.

B.1.2. Les disques SSD (mémoire flash)

Les disques SSD (Solid State Drive) sont conçus à partir de mémoire flash (pour la plupart), ils n'ont pas d'éléments mécaniques, ce qui les rend plus robuste aux chocs. Leur développement a commencé avec les baladeurs type mp3 et les performances qu'ils ont montrées en font des successeurs très sérieux pour remplacer à terme les disques durs classiques. Les flashs qu'ils contiennent sont de deux types: NAND et NOR.

Quelques ordres de grandeur vis-à-vis de leur performance :

* temps d'accès ~0,1ms contre 10 pour un DD
* consommation faible, deux fois moindre qu'un DD (-> portable et système embarqué)
* silence et meilleure robustesse étant donné qu'il n'y a pas d'éléments mécaniques

B.2. Type de mémoire Flash

B.2.1. La flash NAND

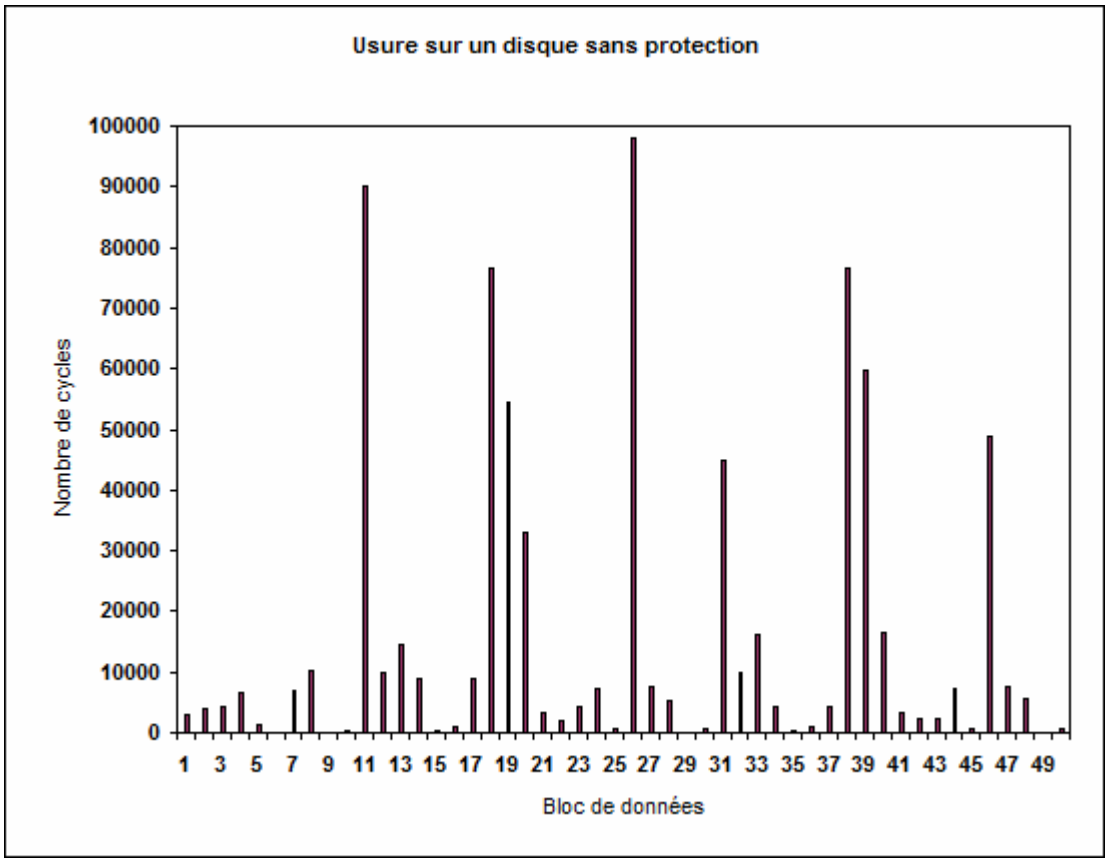
Elle est composée de transistors à grille. L'état du transistor peut être modifié à l'aide d'une tension électrique de l'ordre de 7 à 12 Volts. La lecture se fait avec une tension plus faible (5V) et une mesure de résistivité. Ils existent plusieurs type de NAND, on parle de mémoire SLC (Single Layer Cell) ou MLC (Multi Layer Cell) : la différence consiste en le nombre de bit qui est sauvegardé dans un transistor. Cela a des conséquences sur les performances (je ne détaille pas ici les processus physiques de lecture/écriture en MLC) : la SLC contient moins de données mais est sensiblement plus performante (-20% pour la MLC) ce qui explique que ce soit la SLC qui soit utilisé dans les SSD.

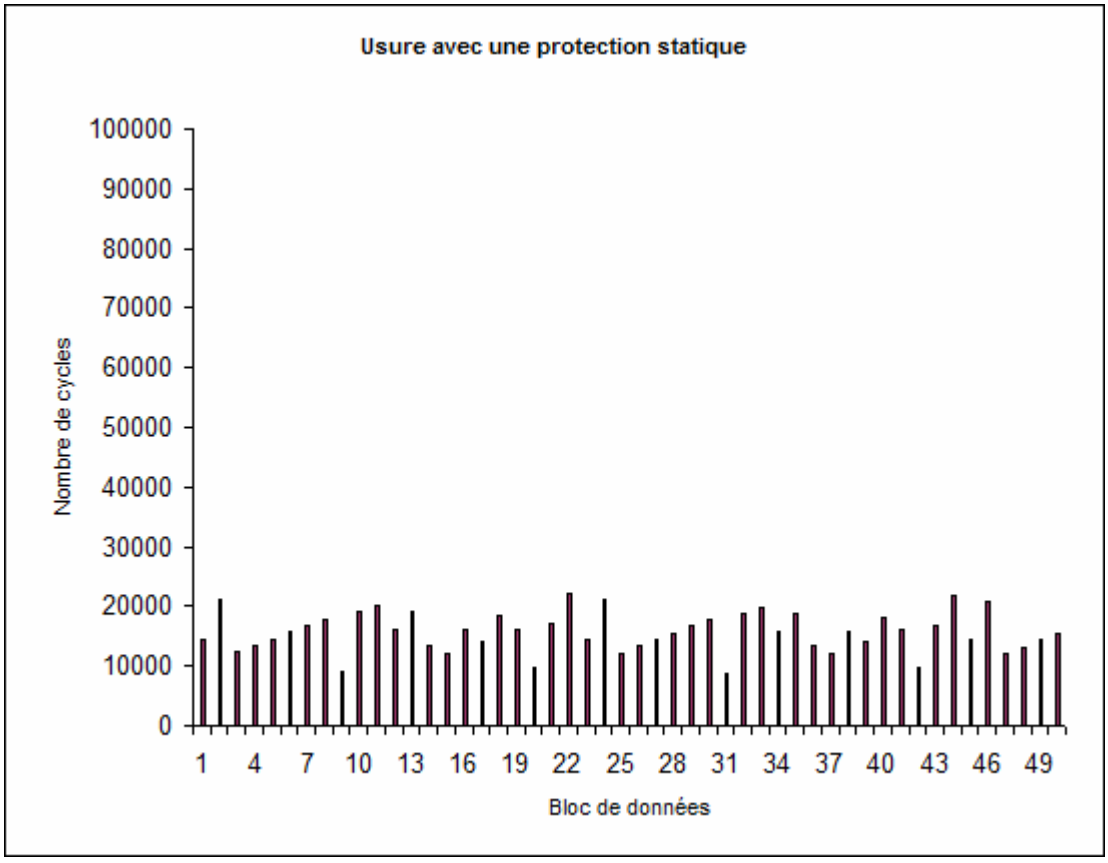
Contrairement à la mémoire NOR, il n'est pas possible d'accéder à un bit particulier directement. Pour accéder à un bit, il faut charger un ensemble de données en mémoire cache (dans la RAM) et on va y lire le bit recherché. Les unités de base du système sont le bloc et la page. Dans les systèmes actuelles (en mai 2007 !) , un bloc a une taille de 128koet est composé de 64 pages. Une page fait donc ~2ko (2048 octets + des bits de contrôle). La prochaine génération devrait travailler avec des pages de Ko. Il est important de signaler que le système fonctionne en lecture par page et en écriture par bloc ! Donc, si l'on veut écrire un seul bit sur le disque, il faut écraser 128ko ce qui explique une certaine lenteur pour les petits fichiers. Il faut cependant relativiser cette lenteur de fonctionnement en écriture un bit : on écrit rarement de si petits fichiers d'une part (cas extrême) et le temps d'accès très rapide (pas de positionnement d'une tête de lecture) compense largement ce désavantage.

Le (faux) problème de la durée de vie : la mémoire SLC, de part sa technologie, a une durée de vie limitée. Les estimations donnent en moyenne une réécriture d'un transistor possible 100000 fois avant qu'il soit détruit. Or le fonctionnement en écriture par bloc intensifie ce phénomène puisque l'on doit écrire un bloc entier lorsque l'on change un seul bit.

Afin de gérer ce problème, les constructeurs ont mis en place un certain nombre de techniques de gestion de l'usure (le Wear Leveling) :

- L'Erase pool : cette technique consiste à réserver des blocs de mémoire pour remplacer les défectueux. L'usure des blocs ne change pas, mais elle est ralentie du point de vue de l'utilisateur.
- Le Static Wear Leveling : cette technique consiste à sauvegarder le nombre d'écriture qu'un bloc a déjà subi. Ainsi, lorsque vous voulez écrire un bloc, le contrôleur interne du disque va déterminer un bloc peu utilisé pour y mettre les données. Plus évolué : le système va déterminer s'il s'agit d'un fichier souvent écrit (log vs fichier de configuration par exemple), et va déplacer les fichiers quasi-statique sur les blocs les plus écrits. Conséquence, l'usure moyenne du disque est plus importante, il n'y a plus de pics d'usure sur certains blocs et le disque est préserve sa capacité totale (voir les schémas comparatifs ci-dessous, tiré d'un article de présence-pc sur les SSD).


Usure sur un disque sans protection
Nombre de cycles
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
Bloc de données


Usure avec une protection statique
Nombre de cycles
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49
Bloc de données

Pour conclure sur ce problème de durée de vie, l'usage des techniques précédentes et la réalisation de benchmark semblent montrer que les SSD ont une durée de vie équivalente, si ce n'est plus important que les disques durs classiques. Pour des disques de 64Go, un calcul théorique prenant pour hypothèse l'écriture continue à vitesse maximum (selon les données constructeurs) donnent une durée de vie de l'ordre de 3-4 ans suivant les disques. Il s'agit du pire cas et dans la réalité, on doit obtenir une durée de vie double ou triple.

On retiendra de cette section le fonctionnement en lecture par page et écriture par bloc qui doit avoir des conséquences sur le système de fichier utilisé.

B.2.2. La flash NOR

Il existe également une flash de type NOR, dont le coût est plus important que la NAND et qui n'est à ce jour que très peu utilisé dans les disques durs. Son usage est adapté à l'enregistrement de données directement exécutées à partir de la mémoire. Elle est donc utilisée pour les OS des appareils électroniques mais peu pour le stockage.

Annexe C : Principe de base du système de fichiers

C.1. Pourquoi un système de fichiers ?

Un système de fichier a pour rôle de créer une relation entre une représentation des fichiers pour l'utilisateur (typiquement, une arborescence) et la façon dont les données et métadonnées des fichiers vont être sauvegardé sur le support matériel. C'est cette architecture qui va permettre de proposer à l'utilisateur des fonctions de base, comme la création, la suppression, la copie en cache de fichiers, indépendamment du support de données utilisées (disque, clé usb, cdrom).

C.2. Structure de base d'un système de fichiers.

C.2.1. Les métadonnées

Les métadonnées sont des informations relatives à un certain fichier. On peut y accéder sous Unix à l'aide de la commande stat. On retrouve notamment dans la structure inode :

- le nom du propriétaire et du groupe (en fait les uid)
- Les permissions (lecture, écriture, exécution pour le propriétaire, le groupe, les autres)
- Les dates de création/dernière modification
- L'id de l'inode.

Ces métadonnées sont relativement normalisées par la norme POSIX. En fait, la norme POSIX dérive du système de fichiers des Unix et exige que les fichiers dit réguliers (ce que l'utilisateur appelle fichier) contiennent les éléments précédemment cités ainsi que leur taille, et un compteur sur le nombre de lien physique pointant sur le fichier. Les métadonnées correspondent donc aux éléments décrivant un fichier. Celle-ci sont stockées dans la brique de base du FS : l'inode

C.2.2. La structure de base : l'inode

L'inode est une structure (en C pour Unix) qui contient notamment des liens vers les métadonnées. Lorsque l'utilisateur demande à accéder à un fichier, il consulte un index qui, en fonction de l'arborescence du fichier, va renvoyer un lien vers le périphérique où le fichier est stocké et le numéro d'inode (unique !) du fichier dans celui-ci. Le numéro d'inode permet alors d'accéder aux données du fichier.

Si on prend l'exemple du système de fichier ext2, avec une taille de bloc de 1ko (l'unité de base du FS) et un adressage en 32bits, voila ce que l'on obtient :

- le bloc de l'inode est composé de 64 champs.
- 13 de ces champs contiennent les données du fichier ou des adresses vers des blocs qui contiennent des données.
- Les 10 premiers champs parmi ces treize contiennent des données (1024 * 10 = 10ko).

- Si jamais la taille du fichier dépasse10ko, le 1eme champ va pointer vers un bloc qui va contenir des adresses vers le fichier. On a donc au total (adressage en 32bits) 256 adresses dans ce bloc d'adresse. Soit un taille de fichier de 1ko * (10 + 256) = 266ko
- Si le fichier est d'une taille supérieur, on utilise le 1ème champ avec une profondeur de 2 : le $12^{ème}$ champ contient une adresse pointant vers un bloc de 256 adresses dont chacune pointent vers un de 256 adresses qui pointent vers un bloc de données. Soit 1ko*(10+256+256^2) = ~64Mo
- Etc avec une profondeur de 3 pour le dernier champ : soit un fichier de ~16Go.

Cet exemple est important car :

- il explique la taille maximale d'un fichier dans un FS
- il montre l'utilité de la décomposition par bloc (32bits utilisés pour en référencés 8 * 1024)
- il montre l'importance de la taille d'adressage (les FS les plus récents sont en 64bits, comme xfs) et l'importance de la taille d'un bloc.

C.3. Prise en compte de l'architecture matérielle dans le FS.

Le rôle du FS est de créer l'organisation des blocs sur le disque dur. A ce titre, il prend en compte les contraintes matérielles comme les mouvements nécessaires de la tête de lecture pour les disques durs classiques. Comme les disques SSD ne possèdent pas de tête mécanique, les optimisations faites pour les disques durs peuvent être un handicap pour les SSD. C'est pourquoi des FS spécifiques aux SDD sont en développement.

Il faudra probablement suivant les solutions retenues pour évaluation effectuant des benchmarks entre les FS classiques et ceux développés pour les SSD : citons JFFS2, YAFFS2, logFS et UBIFS. Cette évaluation n'avait pas sa place dans le cadre du stage.

C.4. Les différents types de système de fichiers.

C.4.1. Système de fichiers non journalisés

Un système de fichier non journalisé est un système de fichier qui ne loggue pas ces modifications. C'est le cas d'ext2 et de la plupart des vieux systèmes de fichiers. Ainsi, si le système plante en pleine écriture, on ne peut pas retrouver les blocs où l'on écrivait les données (qui peuvent être corrompues). Pour assurer l'intégrité des données, il faut donc parcourir le FS en entier à l'aide d'outil tel fsck. Avec la taille des disques durs actuels, c'est une opération excessivement longue.

C.4.2. Système de fichiers journalisé

Un système de fichier journalisé tient un log (un journal) de ces modifications avant toute écriture sur le disque. Ainsi, si jamais une coupure brutale a lieu sans synchronisation du cache avec les disques (un disque qui crash, une coupure de courant), il suffit à fsck de parcourir les dernières entrées du journal et de 'rejouer' le log pour trouver les endroits où les données sont potentiellement corrompues. Tous les FS développés maintenant sont du type journalisés. Pour les plus courants, on peut citer : ext3, reiserfs, xfs. Et en développement : ext4 et reiser4

C.4.3. Système de fichier par transaction

Un système de fichier par transaction consiste à effectuer la mise à jour des métadonnées d'un fichier (et notamment les blocs où il est écrit) après la finalisation de l'écriture des données proprement dite. Bien évidemment, l'écriture ne doit pas modifier les blocs qui contiennent les anciennes données et doit donc se faire dans des blocs libres. Ainsi, à la fin d'une transaction, il suffit de changer le pointeur dans l'inode.

Ce système veut donc dire qu'à tout moment, l'inode d'un fichier pointe sur des données cohérentes. Si un plantage brutal a lieu pendant une écriture, on retrouve au redémarrage les anciennes données, puisque les métadonnées ne sont pas à jour. On garde donc l'intégrité des données.

C.4.4. Système de fichiers distribués

Il s'agit de systèmes de fichiers destinés aux partages de ressources de stockage à travers le réseau. Cette couche permet en quelque sorte d'émuler un disque local pour les applications et l'utilisateur, alors que le disque est en fait distant. Dans les technologies courantes, on retrouve : NFS, AFS (et ces dérivées, OpenAFS, Coda), GFS, Lustre, CIFS (remplaçant du proto samba) notamment. Certains de ses systèmes gèrent déjà l'accès concurrent aux données. Le développement des nouvelles versions le ferait pour ceux dont ce n'est pas encore le cas. Ces systèmes de fichiers sont de plus en plus orientés vers du clustering.

C.5. Système de fichiers sous linux

Le noyau linux fournit aux Unix un environnement multi-fs et ceci en deux étapes. Tout d'abord, il existe dans le noyau une couche d'abstraction du système de fichiers : il s'agit de VFS. D'autre part, linux inclut dans son noyau un ensemble de système de fichiers « réels ». En plus de cela, des outils de gestion des FS sont standardisés dans les unix.

C.4.1. La couche VFS (Virtual File System)

Son rôle est donc de fournir une couche d'abstraction entre les applications et les systèmes de fichiers. Ainsi, lorsqu'une application demande le droit de lire le fichier /mnt/toto, elle n'a pas à se préoccuper du système de fichier sur lequel est le fichier toto. C'est le VFS qui va reconnaître que /mnt est le point de montage de la partition /dev/hd*, qui est formaté en ext3. VFS va alors appeler la fonction de lecture adaptée à l'ext3. Cela vous permet donc aux développeurs d'application de ne pas se préoccuper du système de fichier et cela permet également de faire cohabiter différents FS sous un même système.

C.4.2. Les outils classiques de gestion de FS.

Les outils de gestion de partition/ accès aux données sont standardisés (en partie grâce à VFS). Ainsi toutes les partitions sont formatées grâce à l'outil mkfs (qui va appeler le mkfs.* correspondant au FS souhaité), la cohérence des données peut être vérifiée ou refaite à l'aide de fsck. Les partitions sont montées à l'aide de mount.

C.4.3. Le Sysfs

Le sysfs est le système de fichier virtuel du noyau linux 2.6, il permet d'exporter les propriétés d'un certain nombre d'objets noyaux pour les rendre lisibles en espace utilisateurs, voire pour pouvoir les modifier. Dans notre cas, on peut lire l'état dans lequel se trouve une array RAID, ou encore, l'état du partage DST, simplement en affichant le contenu d'un fichier.