RENPAR'14 / ASF / SYMPA
Hamamet, Tunisie, 10 – 13 avril 2002

Serveur NFS distribué pour grappes de PCs*

Pierre Lombard, Yves Denneulin

Laboratoire Informatique et Distribution - IMAG
51 avenue Jean Kuntzmann, 38 330 Montbonnot Saint Martin, France
{pierre.lombard,yves.denneulin}@imag.fr

Résumé
Les nœuds des grappes disposent souvent de disques qui sont principalement utilisés pour l'installation du système de base et pour les fichiers temporaires. L'idée est d'utiliser cet espace inutilisé en offrant une abstraction des moyens de stockage à l'utilisateur, tout en étant distribués sur différentes machines. Cet article présente l'implémentation d'un serveur NFS distribué s'appuyant sur une architecture à base d'un serveur de méta-données et de plusieurs serveurs d'entrées/sorties (`iods`) utilisant du *spoofing* UDP afin de servir directement le client — cette technique consistant à forger des paquets avec une adresse IP différente de l'adresse IP de la machine. Les premières performances du prototype développé à partir du serveur NFS Linux en mode utilisateur montrent des résultats intéressants.

Mots-clés : grappes NFS distribué UDP *spoofing*

1. Introduction

Le parallélisme connaît actuellement un fort développement dans le domaine des grappes de type *Beowulf* (cf. [8]). De nombreux travaux ont été et sont encore effectués en ce qui concerne l'ordonnancement, la répartition de charge, les environnements de programmation et d'exécution ainsi que l'accès à distance à ces ressources.
Le domaine des systèmes de fichiers n'est pas non plus en reste comme nous le montrons dans la partie 2. Cependant, ces solutions ont les défauts de leur qualité : elles offrent de nombreuses fonctionnalités, ce qui se traduit par une installation << intrusive >> et une maintenance non triviale.
Le système que nous présentons dans cet article vise à fournir un serveur NFS purement logiciel qui résout les problèmes suivants :
– possibilité d'exploiter l'espace disque inutilisé de plusieurs machines (nœuds) ;
– offrir une vue unique des différents espaces disque ;
– avoir des performances suffisantes pour saturer les cartes réseau des nœuds ;
– conserver la même configuration des clients (ne pas modifier le protocole NFS).
Pour cela, nous avons développé à partir du serveur NFS Linux en espace utilisateur un serveur NFS distribué composé d'un serveur de méta-données et de serveurs d'entrée/sortie (E/S) assurant le stockage des données. Le code client demeurant du NFS standard, ils voient le serveur de méta-données comme un serveur NFS classique et les réponses aux requêtes d'E/S viennent en fait directement des serveurs d'E/S — grâce à des techniques de *spoofing*[2] UDP.
Après cette introduction, nous présenterons quelques systèmes de fichiers distribués au vu des problèmes qui nous intéressent. La section suivante décrira les principes de fonctionnement de notre solution, `nfsp` et sera suivie d'une discussion sur les problèmes techniques liés à notre implémentation test. Les premiers résultats obtenus seront présentés puis nous conclurons et donnerons des extensions possibles à ce travail.

*Ce travail a été effectué dans le cadre du projet APACHE (CNRS, INPG, INRIA et UJF) et a utilisé les ressources de la grappe *i-cluster* ID/HP (`http://icluster.imag.fr`)
[2]Ce terme désigne la technique par laquelle des paquets (principalement leurs entêtes) peuvent être modifiés ou créés (forgés) afin que le destinataire croie qu'ils viennent d'une source différente de la source physique réelle.

2. Contexte

Les systèmes de fichiers distribué tentent de répondre à de nombreux problèmes parmi lesquelles on trouve les performances (utilisation de caches), le passage à l'échelle, la sécurité des données (confidentialité ou intégrité)…
L'article [2] recense plusieurs systèmes de fichiers en réseau ainsi que leur fonctionnement. Nous avons étudié ces systèmes du point de vue de la disponibilité sous Linux (grappes de type Beowulf) ainsi que de la simplicité d'installation et d'administration.
La famille de type AFS (AFS, OpenAFS, CODA) est parmi les plus ancienne et fournit des moyens efficace de partager des données au sein d'un SAN (cache sur disques locaux, support du mode déconnecté pour CODA). En revanche, l'installation et l'administration ne peuvent pas être qualifiées de triviales.
Plusieurs autres systèmes sont intéressants mais malheureusement non disponibles pour Linux ou bien les projets se sont terminés. On citera : xFS du projet NoW de Berkeley [9], Swarm [6, 5], …
Un autre système, PVFS [3], est développé depuis quelques années et vise à offrir un système de fichier distribué pour grappe se décomposant en serveur de méta-données et en serveur de données. Ce systèmes a des atouts remarquables mais l'installation d'un client nécessite le chargement d'un nouveau module noyau afin d'assurer une intégration dans le VFS Linux.
Finalement, le classique serveur NFS est un logiciel simple à installer et bien connu des administrateurs. La cohérence des données peut parfois s'avérer malheureuse mais cela n'empêche pas l'industrie de fournir des serveurs NFS dédiés à des prix élevés (solutions NAS).
Ainsi, aucun des systèmes présentés n'a pu répondre aux objectifs présentés dans la section 1. C'est ce constat qui a motivé le développement d'un serveur NFS avec une architecture inspirée de celle de PVFS.

3. Présentation de `nfsp`

Dans cette section nous présentons notre proposition en partant de la description d'un serveur NFS traditionnel et en mettant en évidence les modifications que nous avons effectuées pour mettre en œuvre notre proposition.

3.1. Fonctionnement d'un serveur NFS

L'approche d'un serveur NFS est une approche client-serveur classique : un serveur puissant équipé de plusieurs disques à haute performance sert de nombreux clients (cf. figure 1).

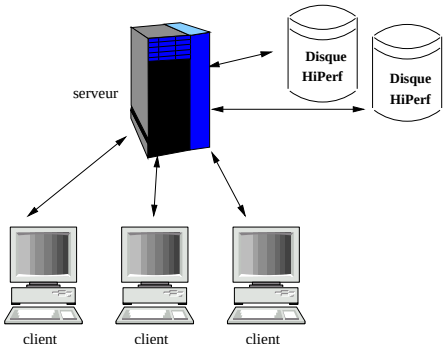

FIG. 1 – Serveur NFS traditionnel

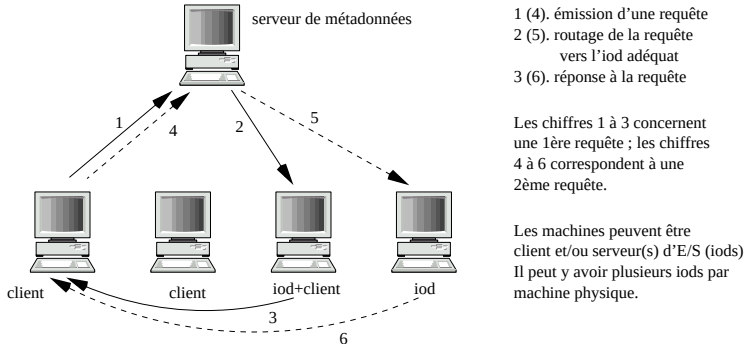

FIG. 2 – Serveur `nfsp`

L'inconvénient de cette approche est son manque d'extensibilité — sauf investissement dans du matériel dédié (baies RAID, cartes Gb) coûtant plusieurs nœuds — et est aussi quelque peu en désaccord avec l'idée du super-calculateur bon marché. De plus, cela ne résout pas le problème de l'espace disque disponible inutilisé (donc perdu) sur les nœuds d'une grappe — cet espace pouvant atteindre plusieurs TB sur les grappes récentes…

Aussi, au lieu d'avoir un serveur central très puissant, avons-nous décomposé le serveur NFS en plusieurs sous-serveurs pouvant être distribués sur plusieurs machines (cf. figure 2). Le respect du protocole et de sa sémantique permet de limiter les problèmes liés à l'installation car les machines clientes disposent souvent déjà d'un support NFS. Ceci pose donc quelques problèmes techniques qui seront présentés dans la section 4.
Le protocole NFS utilise des RPC Sun. Ceux-ci peuvent utiliser de l'UDP ou du TCP, mais très souvent c'est l'UDP qui est choisi car le fait que le protocole NFS ait été conçu pour être sans état se transpose naturellement sur UDP. Une autre conséquence de ce choix est la gestion des << plantages >> des machines : une requête n'ayant pu être traitée (parce que perdue par exemple) finit par être ré-émise par le client. Une fois le volume NFS << monté >> par le client, le n-uplet $(IP_{\texttt{client}}, port_{\texttt{client}}, IP_{\texttt{serveur}}, port_{\texttt{serveur}})$ va alors définir le montage NFS. Les différentes requêtes sont alors distinguées par un numéro au niveau des requêtes RPC choisi par le client (`xid`).

3.2. Architecture de `nfsp`

De la même manière que dans PVFS, nous utilisons un serveur de méta-données (nous l'appellerons dorénavant `nfspd`) ainsi que des démons d'Entrées/Sorties (E/S) appelés `iods`.

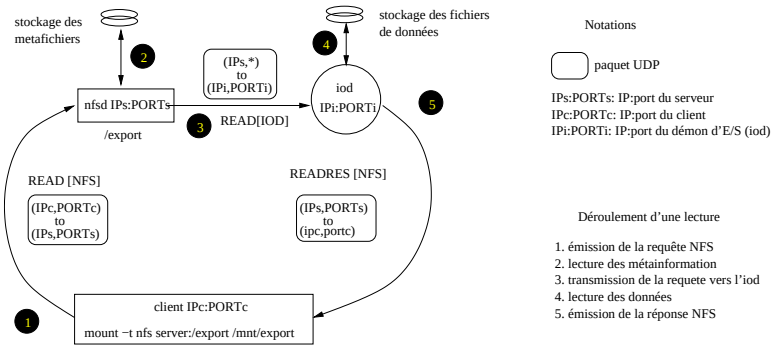

FIG. 3 – Fonctionnement d'un serveur `nfsp`

Cette architecture est illustrée sur la figure 3. Les problèmes posés par l'utilisation de ce serveur de méta-données sont multiples.
Le principal que nous avons eu à résoudre fut celui posé par le chemin de retour des acquittements et des données car à chaque requête de type NFS doit correspondre une réponse. Or le client ne connaît que le `nfspd`, par conséquent la réponse doit (sembler) provenir de la machine qui héberge le `nfspd`.
Partant ce de constat, il semble naturellement plus judicieux de faire répondre les `iods` en utilisant des techniques de *spoofing* UDP/IP. Celles-ci sont souvent utilisées dans dans le cas des attaques de types déni de service (*DoS*). De nombreuses références sur cette technique étant disponibles sur les sites de sécurité réseau, nous mentionnerons simplement [4] à titre d'information. Nous traitons les autres problèmes dans la section suivante qui présente en détails l'implantation réalisée.

4. Notes techniques et implémentation

Cette partie présente les aspects techniques liés à l'implémentation, notamment les fonctionnalités modifiées du démon NFS — les `iods` ayant été réalisés sous forme de serveurs *multithreadés*.

4.1. Liens entre fichiers, méta-fichiers, fichiers de données

Afin de stocker les fichiers du client, nous utilisons des méta-fichiers et des fichiers de données.
Les méta-fichiers sont des fichiers sur le `nfspd`, ils stockent les méta-données[3] du fichier du client. Celles-ci sont contenues dans le fichier (la taille et la version actuelle — cf. 4.2) et dans les méta-données

[3] Littéralement : les << données sur les données >> , telles que la taille, la date de dernier accès, les permissions, etc…

du méta-fichier (date de création, permissions, etc...). Évidemment, les opérations concernant les méta-données (appel `stat()` par exemple) sont gérées directement par le seul démon `nfspd`.
Les fichiers de données sont des fichiers sur les `iods` qui correspondent à des portions (appelées *stripes*) du fichier du client. Leur nom est de la forme `i<inode>s<cookie>o<offset>`[4] et permet à l'`iod` de trouver les données demandées à partir des informations transmises dans la requête envoyée par le `nfspd`.
Nous allons maintenant décrire le déroulement des opérations de base sur les fichiers (création, effacement, lecture, écriture) dans `nfsp`.

4.2. Création d'un fichier

Lorsqu'un fichier normal est créé par le client, le méta-serveur crée un méta-fichier de même nom. Ce méta-fichier (en fait un simple fichier sur le méta-serveur) contient alors une partie des méta-données du fichier de l'utilisateur :
– un *cookie* qui sert à avoir une version du fichier (nécessaire pour éviter que les `iods` ne servent des données non à jour si l'*inode* du méta-fichier a été ré-allouée après un effacement) ;
– la taille du fichier de l'utilisateur ;
– (ultérieurement) des informations sur la répartition des fichiers de données ;
Le reste des méta-données (droits, dates, etc...) est directement géré par le système de fichiers du méta-serveur. Un fichier est alors caractérisé par son numéro d'*inode* et par le *cookie* qui lui a été attribué.

4.3. Effacement d'un fichier

Dans le cas d'un serveur NFS classique, le serveur a simplement à appeler l'appel système `unlink()` et à renvoyer l'acquittement de l'effacement au client.
Avec `nfsp`, cette étape est un peu plus délicate car les données stockées sur les `iods` doivent aussi être effacées sous peine de fuites d'espace disque. Lorsqu'un fichier doit être effacé, le nombre de liens durs sur le méta-fichier est vérifié : s'il est strictement supérieur à 1 alors le méta-fichier est simplement effacé car il reste un moyen d'accéder au données sur les `iods` (en passant par les autres liens durs sur le méta-fichier). S'il vaut 1, alors une demande d'effacement des fichiers de données est envoyée à tous les `iods`.

4.4. Lecture

Sur un serveur classique, la lecture s'effectue en plusieurs phases :
– retrouver le fichier local correspondant au *handle* NFS ;
– vérifier les permissions ;
– lire les données (*offset*, taille) ;
– renvoyer ces données au client.
En utilisant `nfsp` (voir figure 3), la requête de lecture est envoyée au `nfspd` qui va effectuer les deux premières étapes. Le méta-fichier va ensuite être lu afin de connaître les discriminants de la requête (*inode*, *offset*, *cookie*) puis l'`iod` final va être trouvé en calculant un *hash* sur ces valeurs[5] La requête NFS est alors transformé en requête pour l'`iod` après avoir été enrichie des diverses informations nécessaires pour que la réponse NFS soit générée et envoyée au client. L'`iod` va alors se faire passer pour le serveur NFS en *spoofant* l'adresse de ce dernier afin que la réponse semble appartenir au n-uplet $(IP_{\texttt{client}}, port_{\texttt{client}}, IP_{\texttt{serveur}}, port_{\texttt{serveur}})$ définissant le montage NFS.
Les requêtes continuent donc à passer par le méta-serveur qui les oriente vers l'`iod` contenant les données : le gain de performances que l'on va observer vient du fait que plusieurs `iods` vont effectuer en parallèle les E/S lourdes (au lieu d'un seul serveur pour un serveur NFS standard). La charge du méta-serveur sera réduite à des E/S légères, voire inexistantes si les méta-fichiers (quelques dizaines d'octets) sont cachés par le système.

4.5. Écriture

L'écriture utilise les mêmes deux premières phases que celles de la lecture, la lecture étant (évidemment) remplacée par une écriture, et le retour des données au client par un acquittement.

[4] Le `'s'` vient de *seed* car c'est en fait germe aléatoire utilisé pour choisir l'`iod` sur lequel va aller le premier *stripe*.
[5] Un système de routage par la source permet de gérer le cas ou les données seraient à cheval sur deux `iods`.

Avec `nfsp`, la même phase de recherche de l'`iod` devant stocker les données se produit. Les méta-informations sont mises à jour et une fois l'`iod` destination trouvé, elles lui sont transmises accompagnées des données à écrire. L'`iod` va alors effectuer l'écriture puis renvoyer l'acquittement au client en se faisant passer pour le serveur.

5. Premiers résultats

5.1. Matériel et logiciels utilisés

Les tests présentés ont été effectués sur des machines de la grappe ID/HP *i-cluster*. Celles-ci sont des P3-733 équipées de 256MB de RAM, 300MB de *swap* sur disque IDE de 15GB et de cartes Ethernet 3c905C-TX à 100Mb/s. Lors des tests, elles utilisaient une Mandrake 7.1 munie d'un noyau 2.4.4. À titre d'information, le serveur NFS de la grappe est un P3-1GHz avec 512MB de RAM et 1GB de *swap* sur disques SCSI sous noyau 2.4.5-xfs. Afin de lancer les clients rapidement en parallèle nous utilisons *ka-run*, le lanceur parallèle des *ka-tools* (`http://ka-tools.sf.net`).

5.2. État du prototype

Le prototype actuel utilise une version du serveur NFS Linux en espace utilisateur (ou *Universal NFS Daemon* — UNFSD) développée il y a quelques années mais qui a peu évolué depuis 1998 (version 2.2) mais est encore actuellement utilisé dans divers cadres d'utilisation (voir le projet *ClusterNFS* par exemple). Parce qu'exclusivement en espace utilisateur, ce serveur est moins performant que le serveur NFS fourni avec le noyau Linux mais cela simplifie grandement le développement.
Une autre conséquence appréciable est que ce serveur s'installe alors comme une simple application : édition des fichiers `exports` (format classique) et `iods.conf` (liste `host:port` des `iods`), création du répertoire contenant les méta-fichiers et des répertoires de stockages sur les `iods`, démarrage des divers démons (`iods`, `nfspd`, `mountd`). Le système de fichier est dès lors disponible sur les clients avec la commande `mount`. De plus amples informations ainsi que les sources du prototype sont disponibles en ligne à l'URL : `http://www-id.imag.fr/~plombard/nfsp/`.

5.3. Lectures concurrentes d'un gros fichier séquentiel

Le fichier fait ici 1 GB et a été créé dans le cas de 16 `iods` en 110s soit 9,3MB/s, ce qui est inférieur au maximum théorique (un peu plus de 11MB/s) mais néanmoins satisfaisant puisque pour écrire un tel fichier sur le serveur NFS de la grappe 10 secondes de plus sont nécessaires (soit 8,5MB/s).
Les courbes 4 et 5 illustrent les résultats obtenus.

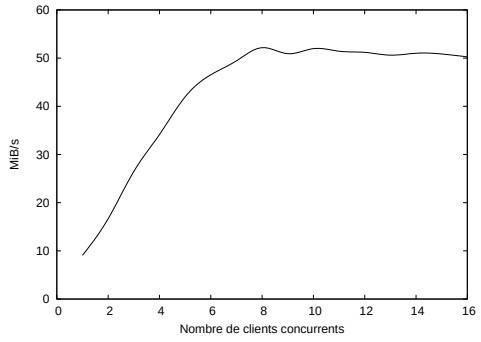

FIG. 4 – Bande passante cumulée (MB/s) : 16 `iods` sur 16 nœuds, 1 à 16 clients sur 16 nœuds

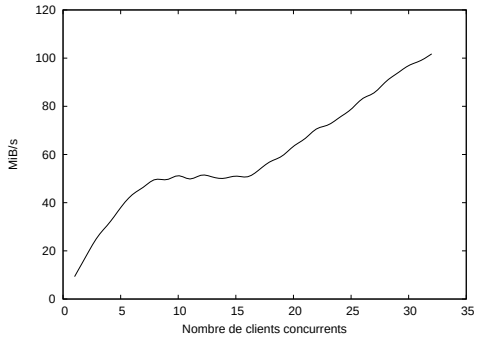

FIG. 5 – Bande passante cumulée (MB/s) : 8 `iods` sur 8 nœuds, 1 à 32 clients sur 16 nœuds

Sur la courbe 4, `nfsp` est installé sur 16 `iods`. La courbe croit fortement jusque vers 5-6 clients en parallèle puis stagne aux alentours de 50MB/s — à ce moment le serveur de méta-données est saturé (CPU). Des mesures plus fines de *profiling* vont être nécessaires afin de trouver le goulot d'étranglement

dans le `nfspd` sachant que nous avons déjà éliminé plusieurs causes : gestion des méta-fichiers (moins de 10% de CPU), saturation réseau (moins de 2 MB/s en entrée — idem en sortie).
La figure 5 illustre principalement les effets du cache NFS au niveau des clients dans le cas où plusieurs processus accèdent au même fichier. Dans ce test, 8 `iods` sur 8 machines sont utilisés, 16 machines clientes montent la partition exportée et le nombre total de processus clients varie entre 1 et 32 (donc de 0 à 2 processus clients par nœud). Cette courbe présente les mêmes tendances que la précédente (plateau à partir de 5–6 clients) mais recommence à croître une fois passés les 16 clients (i.e. 1 processus client par nœud) car le fichier est déjà caché et donc accessible immédiatement.

6. Conclusion et travaux futurs

Nous avons présenté dans cet article une modification d'un serveur NFS existant dans le but de le répartir sur plusieurs machines. Cette approche permet en effet de pouvoir envisager un passage à l'échelle du serveur de manière plus aisée. Le principe exposé consiste en la séparation du serveur NFS en serveur de méta-données et en serveurs de données. Pour ce faire nous utilisons du *spoofing* UDP afin d'éviter aux réponses de transiter via le serveur de méta-données. `nfsp` vise avant tout à être simple à utiliser et à installer tout en essayant de fournir de bonnes performances en distribuant la charge des E/S sur plusieurs nœuds.
Une évaluation plus poussée du `nfspd` s'avère nécessaire mais l'implantation actuelle gagnerait à mettre en place les techniques suivantes :
– *multithreading* ou E/S asynchrones ou autres architectures du serveur (cf [1]) ;
– passage à NFSv3 qui résoudrait plusieurs limites de NFSv2 [7] : fichiers de plus de 2GB, plus d'asynchronisme (bien que cela pose le problème de la gestion des `COMMIT` dans un environnement distribué…) ;
– passer certaines parties en espace noyau afin de minimiser le nombre de recopies mémoire (*zero-copy*) ;
– remplacement du *hash* et du *striping* simple par un système plus souple afin de prendre en compte plusieurs facteurs (performances du matériel, espace disponible, …) et de permettre l'ajout de nœuds – le retrait risquant de s'avérer plus délicat.
D'autres axes de recherches concernent l'utilisation de techniques *multicast* UDP car les *switchs* récents gèrent cela de manière matérielle.
La gestion des fautes au niveau des `iods` et du méta-serveur ainsi que les problèmes liées à la réplication des données constituent aussi des pistes pour des travaux futurs.

Bibliographie

1. The C10K problem (site web). `http://www.kegel.com/c10k.html`.
2. Peter J. Braam. File systems for clusters from a protocol perspective. In *Extreme Linux Workshop #2, USENIX Technical Conference*. USENIX, Juin 1999.
3. Philip H. Carns, Walter B. Ligon III, Robert B. Bross, and Rajeev Thakur. PVFS : A parallel file system for linux clusters.
4. CERT. CA-1996-21 : TCP SYN Flooding and IP Spoofing Attacks. Disponible à `http://www.cert.org/advisories/CA-1996-21.html`, Septembre 1996.
5. John H. Hartman, Ian Murdock, and Tammo Spalink. The swarm scalable storage system. In IEEE, editor, *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*. IEEE, 1999.
6. Ian Murdock and John H. Hartman. Swarm : A log-structured storage system for linux. In *Proceedings of FREENIX Track : 2000 USENIX Annual Technical Conference*, Juin 2000.
7. B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz. NFS version 3, design and implementation. In *Proceedings of the USENIX Summer 1994 Conference*, pages 65–79, Juin 1994.
8. T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer. BEOWULF : A parallel workstation for scientific computation. In *Proceedings of the 24th International Conference on Parallel Processing*, pages I :11–14, Oconomowoc, WI, 1995.
9. Randolph Y. Wang and Thomas E. Anderson. xFS : A wide area mass storage file system. In *Proceedings of the 4th Workshop on Workstation Operating System*, 1993.