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
ENSIMAG - Antenne de Montbonnot - ZIRST
51 avenue Jean Kuntzmann, 38330 MONTBONNOT SAINT MARTIN, FRANCE
{pierre.lombard,yves.denneulin}@imag.fr

Résumé
Les noeuds 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 stockages 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 métadonnées et de plusieurs serveurs d'entrées/sorties (`iods`) utilisant du *spoofing* UDP afin de servir directement le client. 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 clusters de types *Beowulf* (cf. [9]). De nombreux travaux ont été et sont effectués en ce qui concerne l'ordonnancement, la répartition de charge, les environnements de programmation et d'éxécution 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 noeuds,
– 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 métadonnées 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 métadonnées 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* 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. Contexte

Les systèmes de fichiers distribué tentent de répondre à de nombreux problèmes parmi lesquelles on trouve les performances (utilisation de caches), la scalabilité, la sécurité des données (confidentialité ou intégrité)…
L'article [3] 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 (*clusters* 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 [10, 11], Swarm [7, 6], …
Un autre système, PVFS [4], est developpé depuis quelques années et vise à offrir un système de fichier distribué pour grappe se décomposant en serveur de métadonnées 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 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 << gros >> serveur avec de multiples disques haute-performance qui sert des clients (cf. figure 1).

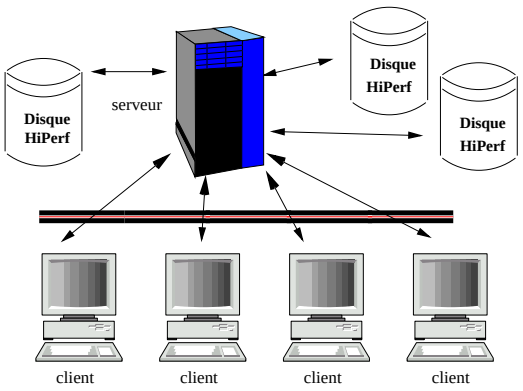

FIG. 1 – Serveur NFS traditionnel

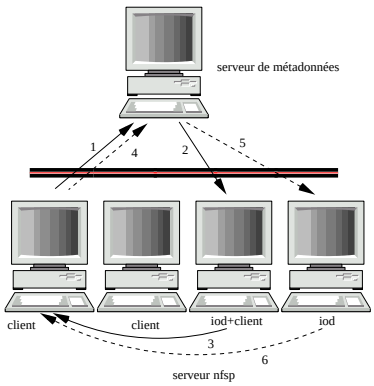

FIG. 2 – Serveur NFSP

L'inconvénient de cette approche est qu'elle est peu scalable - sauf investissement dans du matériel dédié (baies RAID, cartes GB) coûtant plusieurs nœuds - et est aussi quelque peu en désaccord avec l'idée du supercalculateur bon marché.

Cela ne résout pas le problème de l'espace disque inutilisé (donc perdu) et disponible sur les nœuds d'une grappe - espace qui peut atteindre plusieurs tera-octets sur des 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.
Nous avons choisi de rester dans le cadre imposé par le respect du protocole NFS afin de limiter les problèmes liés à l'installation, le client étant très souvent déjà utilisé et disponible sur les systèmes Linux installés. Certains problèmes techniques apparaissent et 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, NFS a été conçu pour être un protocole sans état ce qui se transpose naturellement sur un mode de communication par UDP. Une conséquence de cette conception est que la gestion des << plantages >> des machines (clients ou serveurs) ne pose pas trop de problèmes : une requête n'ayant pu être traitée (parce que perdue par exemple) finit par être réémise par le client. Une fois le volume NFS << monté >> par le client, une pseudo connexion UDP est établie ; le n-uplet (IPclient, Portclient, IPserver, PortServer) définit le montage NFS. Les différentes requêtes sont alors distinguées par un numéro au niveau des requêtes RPC (`xid`) choisi par le client.

3.2. Architecture de NFSP

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

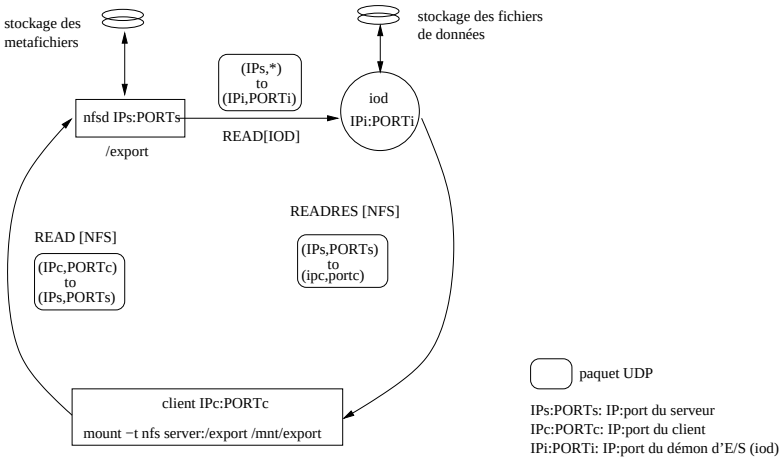

FIG. 3 – Fonctionnement d'un serveur NFSP

Cette architecture est illustrée sur la figure 3. Les problèmes posés par l'utilisation de ce serveur de métadonnées 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 [5] 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.

4.1. Liens entre fichiers, métafichiers, fichiers de données

Afin de stocker les fichiers du client, nous utilisons des métafichiers et des fichiers de données.
Les métafichiers sont des fichiers sur le `nfspd` , ils stockent les métadonnées[2] du fichier du client. Celles-ci sont contenues dans le fichier (la taille et la version actuelle - cf. 4.2) et dans les métadonnées du métafichier (date de création, permissions, etc. . .). Évidemment, les opérations concernant les métadonnées (appel `stat()`) sont gérées directement par par le seul démon `nfspd` .
Les fichiers de données sont des fichiers sur les `iods` qui correspondent à des portions (*stripes*) du fichier du client. Leur nom est de la forme i<inode>s<cookie>o<offset>[3] 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 métaserveur crée un métafichier de même nom. Ce métafichier (en fait un simple fichier sur le métaserveur) contient alors une partie des métadonnées 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 métaficher a été réallouée après un effacement),
– la taille du fichier de l'utilisateur
Le reste des métadonnées (droits, dates, etc. . .) est directement géré par le système de fichiers du métaserveur.
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 métafichier est vérifié : s'il est supérieur à 1 alors un simple `unlink()` du métafichier est effectué car il reste des données sur les `iods` qui peuvent être jointes par un autre fichier. S'il vaut 1, alors nous devons prendre soin d'envoyer à tous les `iods` une demande d'effacement des données du fichier de l'utilisateur.

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` , la requête de lecture est envoyée au `nfspd` qui va effectuer les deux premières étapes. Le métafichier va ensuite être lu afin de connaître les discrimants de la requête (*inode* , *offset* , *cookie*) puis l'`iod` destination va être trouvé en calculant un *hash* sur ces valeurs.
La requête est transmise à l'`iod` concerné enrichie des métainformations (permissions, heures, ...) nécessaires pour générer la réponse depuis l'`iod` . Celui-ci lit les données correspondantes et génère une réponse NFS à destination du client en se faisant passer pour le serveur, la réponse semblant alors appartenir à la pseudo-connexion UDP établie entre le client et le métaserveur.

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.
Avec `nfsp` , la même phase de recherche de l'`iod` devant stocker les données se produit. Les métainformations sont mises à jour et, une fois le destinataire 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

[2] littéralement : les << données sur les données >> , telles que la taille, la date de dernier accès, les permissions, etc. . .
[3] le << 's' >> vient de *seed*, qui correspond à un germe aléatoire pour choisir un premier hôte pour le *striping*

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*[4]. Celles-ci sont conçues autour d'un P3-733Mhz et équipées 256MiB de RAM avec 300MiB de *swap* sur un disque IDE de 15GB et d'une carte ethernet à 100Mb/s. Au moment des tests, elles étaient installées en Mandrake 7.1 munie d'un noyau 2.4.4. À titre d'information, le serveur NFS de la grappe est un P3-1Hz avec 512MiB de RAM et 1GiB de *swap* sur disques SCSI et dispose d'un 2.4.5-xfs.

5.2. État du prototype

Le prototype actuel s'appuie en grande partie sur le code du serveur NFS Linux en espace utilisateur[5] qui a été implémenté il y a quelques années par Mark Shand, puis amélioré par Donald Becker, Rick Sladkey, Orest Zborowski, Fred van Kempen, et Olaf Kirch. Il a peu évolué depuis 1998 (version 2.2) mais constitue une base très pratique afin d'effectuer des expérimentations (par exemple ClusterNFS [2]) sur un serveur NFS.

Parce qu'exclusivement en espace utilisateur, il est moins performant que le serveur NFS du noyau Linux mais cela évite d'avoir des << plantages >> durs sur la grappe distante. Une autre conséquence appréciable est l'installation simplifiée du serveur qui s'installe comme une simple application.

Le prototype a fonctionné pendant un mois sur l'*i-cluster* sans problème majeur et est disponible à l'URL : `http ://www-id.imag.fr/Laboratoire/Membres/Lombard_Pierre/nfsp/`

5.3. Lectures concurrentes d'un gros fichier séquentiel

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

Afin de lancer les clients rapidement en parallèle nous utilisons `ka-run` le lanceur parallèle des `ka-tools`[6]

Les courbes 4 et 5 illustrent les résultats obtenus.

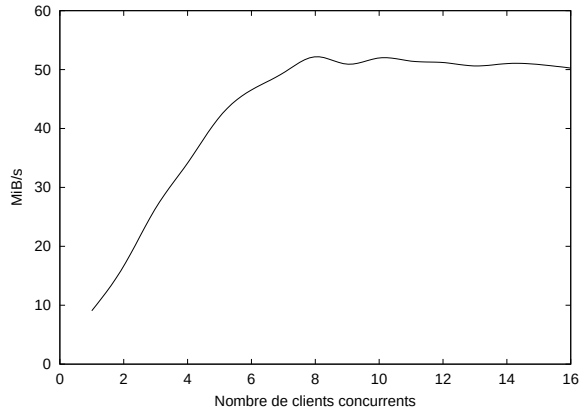

FIG. 4 – Bande passante cumulée (MiB/s) : 16 `iods` sur 16 nœuds, de 1 à 16 clients sur 16 nœuds

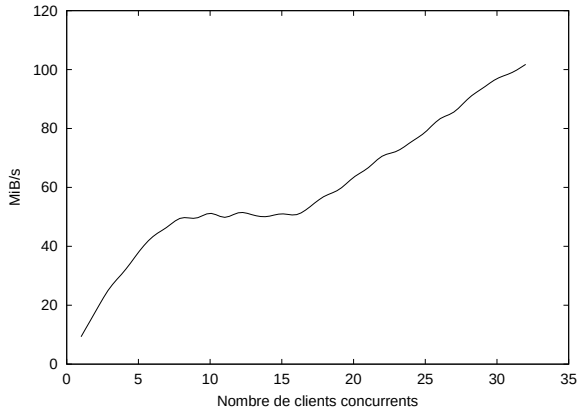

FIG. 5 – Bande passante cumulée (MiB/s) : 8 `iods` sur 8 nœuds, de 1 à 32 clients sur 16 nœuds

Sur la courbe 4, `nfsp` est installé sur 16 `iods` . La courbe croit fortement jusque vers 5-6 clients en pa-

[4] Voir `http ://icluster.imag.fr`
[5] ou Universal NFS Daemon - UNFSD
[6] `http ://ka-tools.sf.net`

rallèle puis stagne aux alentours de 50MiB/s - à ce moment le serveur de métadonnées est saturé (CPU). Nous n'avons pas encore effectué de mesures de *profiling* plus fines afin de savoir quelle part du `nfspd` consomme le plus de ressources mais nous avons éliminé plusieurs causes : gestion des métafichiers (moins de 10% de CPU), saturation réseau (moins de 2 MiB/s en entrée - idem en sortie).
La figure 5 illustre les effets du cache 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 précedente (plateau à partir de 5-6 clients) mais recommence à croître une fois passé 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 afin d'augmenter sa scalabilité. La proposition que nous avons présentée consiste en la séparation du serveur NFS en serveur de métadonnées et de serveurs annexes de stockage. Pour ce faire nous utilisons du *spoofing* UDP afin d'éviter aux réponses de transiter via le serveur de métadonnées. 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 noeuds.
Une évaluation plus poussée est nécessaire mais l'implantation actuelle gagnerait à mettre en place les techniques suivantes :
– utilisations de techniques plus avancées pour les performances [1]
– passage à NFSv3 qui résout plusieurs limites de NFSv2 [8] : fichiers de plus de 2GiB, 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*)
D'autres axes de recherches concernent l'utilisation de techniques *multicast* en UDP car ces paquets sont gérées de manière *hardware* par les *switchs* récents. L'ajout d'un support de reconfiguration à la volée des `iods` afin de gérer l'arrivée ou le départ de nœuds (un peu comme dans les systèmes *peer-to-peer*) afin d'augmenter la capacité de stockage globale serait aussi un axe à étudier.

Bibliographie

1. The C10K problem (site web). http ://www.kegel.com/c10k.html.
2. ClusterNFS (site web). http ://clusternfs.sf.net.
3. Peter J. Braam. File systems for clusters from a protocol perspective. In *Extreme Linux Workshop #2, USENIX Technical Conference*. USENIX, June 1999.
4. Philip H. Carns, Walter B. Ligon III, Robert B. Bross, and Rajeev Thakur. PVFS : A parallel file system for linux clusters.
5. CERT. CA-1996-21 : TCP SYN flooding and IP spoofing attacks. http ://www.cert.org/advisories/CA-1996-21.html, september 1996.
6. 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.
7. Ian Murdock and John H. Hartman. Swarm : A log-structured storage system for linux. In *Proceedings of FREENIX Track : 2000 USENIX Annual Technical Conference*, June 2000.
8. 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, June 1994.
9. 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.
10. 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.
11. Randolph Y. Wang, Thomas E. Anderson, and Michael D. Dahlin. Experience with a distributed file system implementation. Technical Report CSD-98-986, January 1997.