Une nouvelle façon d'envisager les entrées/sorties dans un système d'exploitation pour grappe

Gaël Utard, Christine Morin

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Une nouvelle façon d'envisager les entrées/sorties dans un système d'exploitation pour grappe

Gaël Utard, Christine Morin

N°4948

Octobre 2003

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4948--FR

INRIA
RENNES

Une nouvelle façon d'envisager les entrées/sorties dans un système d'exploitation pour grappe

Gaël Utard*, Christine Morin†

Thème 1 — Réseaux et systèmes
Projet PARIS

Rapport de recherche n° 4948 — Octobre 2003 — 13 pages

Résumé : La majorité des applications pour grappe sont très dépendantes des performances d'entrée/sortie du système, qu'il s'agisse de calcul numérique, de compilation ou encore de services TCP (web, mail...). D'autre part, la simplicité d'utilisation et d'administration d'une grappe est conditionnée par la vision unique et cohérente des données (binaires, fichiers de configuration, données utilisateur) sur l'ensemble de la grappe. Or à ce jour, aucun système de fichier n'est capable d'exploiter efficacement les disques d'une grappe pour répondre de manière simple à la très grande variété des besoins. Au lieu de proposer un nouveau système de fichier en tant que tel, nous avons choisi de revisiter la couche d'entrée/sortie du système d'exploitation afin de la rendre optimale dans un environnement distribué. Nous proposons de partager le cache de données et de méta-données sur l'ensemble de la grappe grâce à des techniques de mémoire partagée répartie. Cette solution de bas niveau offre de nouvelles perspectives liées à l'intégration avec la gestion de la mémoire, sans pour autant fermer la porte à la création d'interfaces de haut niveau, tel MPI-IO.

Mots-clé : Grappe de calculateurs, Entrées/sorties, SGFP, Système d'exploitation distribué

(Abstract: pto)

* Gael.Utard@irisa.fr
† Christine.Morin@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Revisiting Inputs/Outputs in a Cluster Operating System

Abstract: The majority of cluster applications depends on I/O performance, whether they are numerical computations, compilers or mail servers. And their ease of use and administration depends on the single and consistent view of the data (including system binaries, configuration files and home directories). To date, no cluster file system provides both performance and ease of use. Rather than putting an other middleware forward, this paper explores a way to make the operating system naturely capable of distributed I/O. It proposes to manage a clusterwide cache (consisting of data and metadata) through distributed shared memory techniques. Thanks to its perfect integration into the operating system, this low level solution offers interesting prospects such as an efficient support for fault tolerance.

Key-words: Cluster of workstations, Inputs/outputs, Parallel file system, Distributed operating system

1 Introduction

Le champ des applications pour grappes de PC est très varié : calcul numérique, compilation, services TCP (HTTP, FTP, SMTP, IMAP...). Mais pour que les grappes soient réellement intéressantes en terme de rapport performance/coût, il convient de se rapprocher au maximum des performances théoriques offertes par le matériel, ce qui n'a rien d'immédiat. De plus, pour simplifier la programmation et l'administration des grappes, il est nécessaire de fournir un support de haut niveau offrant un certain degré de transparence, autrement dit un système à image unique.

Dans ce contexte, les entrées/sorties sont d'une importance capitale. Elles font l'objet de trois grandes attentes:

1. les *performances*, autant pour les accès séquentiel sur les fichiers de grande taille, que pour les multiples opérations de création/suppression sur les fichiers de petite taille,
2. la *simplicité d'utilisation*, qui passe par la vision unique de l'arborescence via une interface POSIX (ce qui n'exclut pas de proposer d'autres interfaces en parallèle),
3. la *liaison* parfaite avec les autres mécanismes système de la grappe: migration de processus, création de points de reprise, mémoire viruellement partagé.

Le papier traite de cette problématique des entrées/sorties pour grappe. Dans un premier temps, il présente une analyse des entrées/sorties et des possibilités d'optimisation, illustrée par trois des solutions généralement appliquées aux grappes (NFS, PVFS et GFS). Dans un second temps, il énonce quelques principes pour résoudre le problème plus efficacement à moindre coût. Puis il propose une nouvelle solution, basée sur le partage du cache de données et de méta-données selon des techniques dérivée de celles de la mémoire partagée répartie.

2 Eléments de performance

Succintement, une entrée/sortie est un *parcours* de données entre le support de stockage physique et l'espace d'adressage d'un processus. Ce parcours est susceptible de comporter des d'étapes de nature variée :

- transfert entre le support magnétique et le contrôleur disque,
- transfert entre le contrôleur et la mémoire centrale,
- appel système,
- copie mémoire de l'espace noyau vers l'espace utilisateur ou inversement,
- traitement en espace noyau par le système de fichier,
- traitement en espace utilisateur par des bibliothèques (ex. libc).

L'environnement *distribué*, ajoute encore quelques possibilités:

- traitement par un démons (ex. serveur NFS),
- passage par une pile réseau,
- transfert entre la mémoire centrale et la carte réseau,
- transfert physiques sur un réseau.

Il faut rappeler qu'en sus des données en tant que tel, des métadonnées, des requêtes et des acquittements sont susceptibles de suivre ces étapes.

2.1 Latence et débit

En terme de performance, chaque étape peut être caractérisée par sa *latence* et son *débit*. Sommairement, la latence globale est la somme des latences partielles, et le débit global est le plus petit des débits partiels. Ce dernier détermine ce que l'on appelle le goulot d'étranglement. Il faut cependant noter que les performances individuelles peuvent être inter-dépendantes en cas de conflit d'utilisation d'une ressource matérielle. Par exemple, comme le montre [2], des transferts mémoire $\leftrightarrow$ disque et mémoire $\leftrightarrow$ carte réseau concurrents peuvent saturer le bus local. Dans le même ordre d'idée, la latence du traitement par un démon est assez problématique car il est difficile de contrôler l'ordonnancement du processus correspondant, en particulier lorsque le nœud sert à la fois pour les entrées/sorties et pour le calcul.

2.2 Caches

Des *caches*, placés en différents endroits, évitent parfois de parcourir un chemin de bout en bout. On en trouve par exemple dans les contrôleurs disque, dans les noyaux, et dans la bibliothèque C. Plus le cache où l'on peut trouver une donnée est proche, plus la latence est réduite. L'efficacité d'un cache est conditionnée par sa taille et par le choix du sous-ensemble des données qu'il contient. Sur ce dernier point, on s'appuie sur le principe de la localité temporelle qui stipule qu'une donnée qui vient d'être accédée à de fortes chance de l'être de nouveau dans un futur proche. On peut aussi s'appuyer sur le principe de localité spatiale en chargeant un sur-ensemble dans le cache et en espérant que les données contiguës seront accédées dans un futur proche. S'ils ne règlent pas tous les problèmes, les cache sont essentiels en terme de performance. Cela est valable tant pour les données que pour les méta-données. La difficulté principale apportée par l'environnement distribué est la gestion de la cohérence des données dupliquées sur plusieurs nœuds.

2.3 Démultiplication du débit

Nous avons vu que le débit peut être limité par un goulot d'étranglement. Par exemple, il peut s'agir du débit physique du disque. Dans ce cas, on peut espérer multiplier le débit par n si on accède aux données sur n disques en parallèle, tant que l'on ne rencontre pas un nouveau goulot d'étranglement. C'est le principe de l'*entrelacement*, où le chemin des données se divise au niveau du contrôleur pour se répartir sur plusieurs disques. D'un autre côté, plusieurs chemins de données peuvent converger vers un même point. C'est le cas par exemple d'un serveur centralisé, qui peut devenir un goulot d'étranglement lorsque le nombre de clients augmente. On dit qu'il ne passe pas à l'échelle.

2.4 Masquage de la latence

L'importance de la latence s'efface devant celle du débit lorsque la quantité de données à transférer sur une même requête est importante. Mais pour les applications qui réalisent beaucoup de petites opérations sur une multitude de fichiers, cette condition n'est jamais réalisée. La solution est alors de lancer des requêtes successives sans attendre le résultat des premières, selon le principe du *pipelining*. Celui-ci peut-être déclenché à plusieurs niveaux :

- au niveau d'une bibliothèque d'entrée/sortie spécialisée dotée d'une interface permettant de spécifier des transferts complexes en une seule opération (ex. MPI-IO [7]),
- au niveau de l'application, en déclenchant des opérations asynchrones,
- au niveau du noyau en déclenchant des préchargements de cache (ex. lecture en avant)

2.5 Haute disponibilité et tolérance aux fautes

La haute disponibilité suppose qu'il existe des chemins de données alternatifs. En effet, en cas de défaillance d'un nœud, les chemins qui passaient par ce nœud doivent pouvoir être reportés sur d'autres nœuds. Par exemple, on peut mettre en place une chaîne SCSI partagée par deux contrôleurs situés dans deux nœuds différents, ce qui garantit l'accès aux disques en cas de défaillance de l'un d'entre eux. La tolérance aux fautes suppose la duplication des données sur plusieurs disques, et donc l'existence d'autant de chemins.

3 Exemples d'accès parallèles et distribués pour grappe

Dans un système UNIX, le système de fichier est découpé en trois couches [4] :

- le système de fichier *logique* (Virtual File System), commun à tous les systèmes de fichiers, qui répond aux appels systèmes à partir des informations contenues dans le cache.
- le *cache* qui contient les données (sous forme de pages), et les métadonnées (sous forme d'inœuds et d'entrées de répertoire) récemment utilisées,
- le système de fichier *physique* qui gère l'organisation des informations sur disque.

Voici en guise d'illustration trois systèmes qui permettent à tous les nœuds de calcul d'un grappe d'accéder à des données partagées, tout en multipliant la bande passante des disque grâce à des accès en parallèle.

3.1 RAID exporté par NFS

NFS [10] (figure 3.1) est largement répandu, mais il présente des inconvénients majeurs. Le cache local est utilisé, mais invalidé au bout d'un court intervalle de temps (généralement 30 secondes). C'est un compromis entre performance et respect de la sémantique de partage UNIX. Au final, on a ni l'un, ni l'autre. De plus, il s'agit d'un serveur centralisé qui passe difficilement à l'échelle sans investir dans un serveur coûteux. Enfin, le parcours des données n'est pas du tout optimisé, car il passe deux fois dans les couches du système de fichier, fait

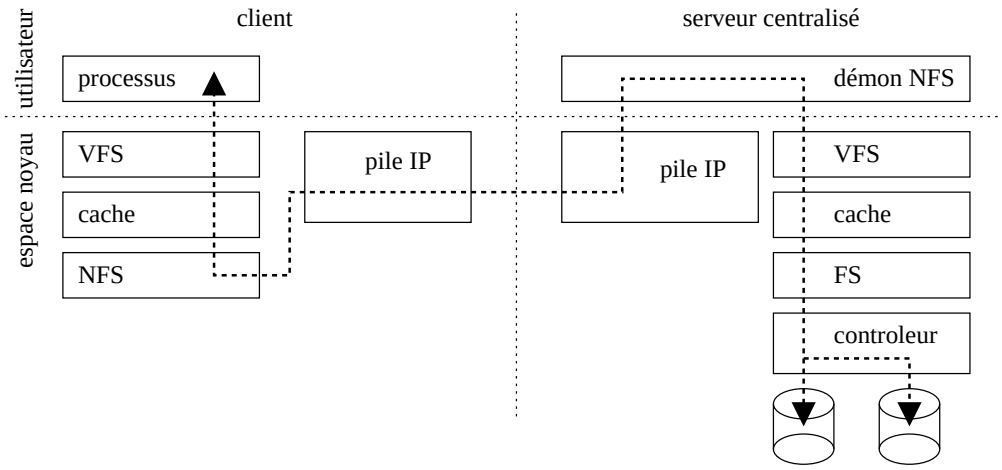

FIG. 1 – *RAID exporté par NFS*

intervenir un démon et traverse trois fois la barrière entre espace utilisateur et espace noyau. Certaines implémentations sont cependant un peu plus performantes car elles intègrent le démon NFS dans le noyau.

3.2 Parallel Virtual File System (PVFS)

PVFS [1] (figure 3.2) présente deux améliorations majeures par rapport à NFS. D'abord, les données sont entrelacées non plus sur les différents disques d'un même serveur, mais sur les disques de différents serveurs. Ensuite, il est possible de lier l'application avec une bibliothèque spécifique (dotée d'une interface spécifique) qui communique directement avec les serveurs. Cependant, il n'y a pas de cache côté client, donc toute opération implique une communication avec le serveur, et pour autant la sémantique UNIX n'est pas respectée. D'autre part, les méta-données sont gérées par un unique serveur centralisé, donc PVFS n'est adapté qu'aux applications faisant très peu d'opérations d'ouverture/fermeture par rapport aux opérations de lecture/écriture. Rajoutons également que même si ce n'est pas obligatoire, pour des performances optimales, il convient de distinguer les nœuds de calcul et les nœuds d'entrée/sortie.

3.3 Global File System (GFS)

GFS [9] (figure 3.3) est fondamentalement différent des systèmes précédents, car ici les échanges s'opèrent au niveau des secteurs disque grâce à des commandes SCSI sur un lien FiberChannel. Ces échanges ont lieu entre les clients et une baie de stockage RAID directement attaché au réseau. Un système de verrous fournis par le contrôleur RAID garantit la

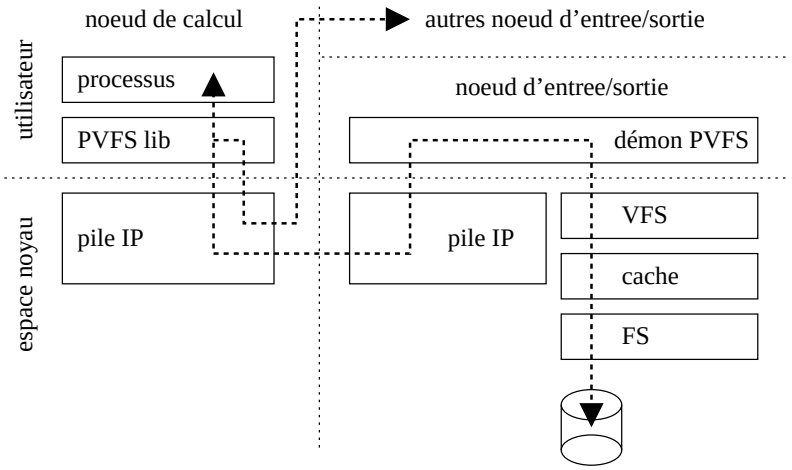

FIG. 2 – *Parallel Virtual File System (PVFS)*

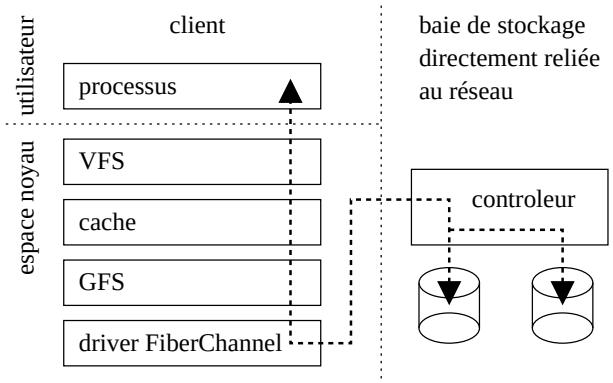

FIG. 3 – *Global File System (GFS)*

sémantique UNIX. Outre qu'il nécessite l'achat d'un matériel très spécifique (et coûteux), sa gestion de cache n'est pas optimale. Aucun échange d'informations ne peut avoir lieu directement entre deux nœuds. Un inœud ou une page modifiés doivent repasser par le contrôleur RAID avant d'être utilisée sur un autre nœud.

4 Quelques principes pour une solution optimale

4.1 Messages actifs

Certaines solutions supposent l'existence d'un démon. Une des conséquences de cela est que la barrière utilisateur/noyau est traversée plusieurs fois. Or chaque passage de l'espace utilisateur à l'espace noyau ou inversement suppose (1) un coût lié aux mécanismes de protection (recopie, changement de contexte du processeur) et (2) une latence due au mécanismes d'ordonnancement. Le coût de la protection peut être supprimé en plaçant le démon dans le noyau, mais les problèmes d'ordonnancement restent entiers. De ce point de vue, la solution optimale est de remplacer le démon d'entrée/sortie par un système à base de messages actifs [11].

4.2 Accès distant direct

En théorie, il semble intéressant d'utiliser les disques présents dans les nœuds de calcul de la grappe, en vertu du principe de *recouvrement* des calculs et des communications. En pratique, cela est difficile à réaliser, car servir une donnée suppose (1) de la transférer du disque à la mémoire centrale puis (2) de la mémoire à la carte réseau. Et comme le calcul implique également des transferts importants entre mémoire et processeur, le bus local est potentiellement saturé. En permettant un transfert des données par DMA entre contrôleur disque et contrôleur réseau, l'accès distant direct [2] permet de maximiser le débit d'entrée/sortie d'un nœud sans nuire à son calcul. Cela suppose cependant de disposer de contrôleurs réseau adaptés (ex. carte Myrinet).

4.3 Cache coopératif

Le cache est un élément cruxial pour les performance d'une système d'entrée/sortie. Cependant, dans un système où les communications sont au niveau utilisateur (ex. PVFS), le système de cache du système local est court-circuité, ce qui oblige à en réimplémenter un autre. Et dans un système où les communication sont à très bas niveau (ex. GFS), il n'est pas possible d'échanger directement des pages ou des méta-donnés entre deux nœuds. De plus, en environnement distribué, il est difficile de maintenir la cohérence d'un cache. Dans la plupart des systèmes de fichier distribués, le problème est contourné, soit en assouplissant la sémantique du partage, soit en désactivant au moins partiellement le cache. Nous pensons qu'aucune de ces solutions n'est acceptable, car pour être réellement utilisable, l'application devient obligée de palier aux défaillances du système à travers l'utilisation de son propre système de cache. De plus, il ne faut pas oublier que le cache de méta-données, souvent négligé, est aussi important que le cache de données, voir plus pour certaines applications. Nous pensons que la solution pour obtenir une gestion de cache optimale est d'organiser l'échange des informations directement entre les caches des différents nœuds (figure 4.3).

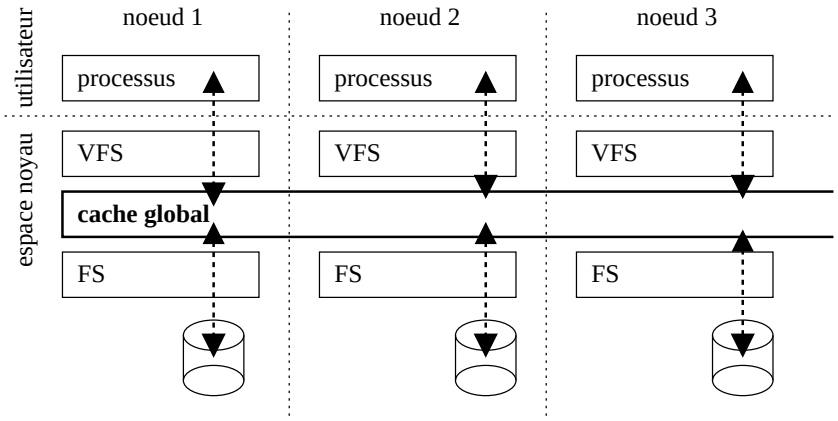

FIG. 4 – *Cache coopératif global*

5 La solution proposée

5.1 Cache des pages global

Notre proposition s'articule essentiellement autour d'un cache de données global à l'ensemble de la grappe. Il s'agit du classique cache des page du noyau auquel on adjoint des mécanismes permettant l'échange et la duplication des pages entre les nœuds tout en garantissant une cohérence séquentielle. Ces mécanismes sont dérivés des techniques de mémoire partagée répartie [5]. Tout d'abord, il s'agit d'adresser les pages non plus sur un nœud, mais sur l'ensemble de la grappe. D'habitude, on adresse une page avec le triplet (périphérique, numéro d'inœud, décalage dans le fichier). Dans notre cas, il suffit d'intégrer un numéro de nœud dans l'identification du périphérique en sus des numéros majeurs et mineurs. Ensuite, sur un nœud donné, il faut connaître l'état (dans le protocole de cohérence) de chaque page physique. Cela est réalisé à moindre coût en utilisant quelques bits réservé dans la structure déjà existante du noyau qui décrit chaque page physique. Enfin, il faut maintenir de manière distribuée des tables permettant de retrouver les différentes copies d'une page en cache. Nous proposons de mettre en œuvre un protocole du type gestionnaire statique distribué [5]. Un nœud est dit gestionnaire d'une page lorsque cette page à une image sur son disque. Parmi les nœuds qui possèdent une copie d'une page dans leur cache local, un et un seul d'entre eux est identifié comme étant propriétaire de la page. Contrairement au gestionnaire, le propriétaire d'une page est dynamique. Le gestionnaire d'une page connaît le nœud qui en est propriétaire à cet instant. Le propriétaire d'une page en connaît toutes les copies à cet instant. Ainsi un nœud A a besoin d'une page en lecture, il va d'abord chercher dans son cache local. En cas de défaut, il contacte le gestionnaire G de cette page. Si G n'a pas d'information de propriété, c'est que la page n'est pas en cache global et il répond

à la requête à partir de son disque. Sinon, il transmet la requête au propriétaire P qui en envoie une copie à A. Si A a besoin du droit en écriture alors P doit envoyer des requêtes d'invalidations à tous les nœuds disposant d'une copie. Puis il transfère la propriété de la page à A en même temps que son contenu. Enfin il invalide sa propre copie.

5.2 Optimisations du protocole

De manière individuelle, sur chacun des nœuds, et pour chaque fichier ouvert, il est possible de maintenir un cache, contenant les propriétaires des dernières pages accédées. Cela permet de connaître de manière probable (mais pas certaine) le propriétaire de certaines pages, et de court-circuiter le gestionnaire. On peut aussi extrapoler l'information pour une page à partir des informations pour les autres pages du même fichier. Statistiquement, un grand nombre de requêtes sont ainsi réduites à un saut au lieu de deux. En contrepartie, les autres auront à faire deux sauts et parfois plus en suivant la chaîne des propriétaires probables, voire en repassant par le gestionnaire. Par exemple, lors d'un accès séquentiel, si un nœud A sait que les quatre première pages d'un fichier sont dans le cache d'un nœud B, mais ne dispose pas d'information concernant la cinquième page, il supposera qu'elle se trouve également sur le nœud B. Le cas échéant, B transmettra la requête en fonction de ses propres informations. Si aucune information n'est disponible, la requête est dirigée vers le gestionnaire. En effet, le gestionnaire doit garder une information certaine. Ceci impose de ne pas le court-circuiter pour les requêtes qui supposent un changement de propriétaire, c'est à dire les demandes de page en écriture. Cela limite la portée de l'optimisation aux seules requêtes en lecture.

5.3 Cache des métadonnées

En parallèle du cache global des pages, un cache global des blocs disque de métadonnées (bitmaps d'allocation, contenu des répertoires, blocs d'indirection) fonctionne selon le même principe. Ainsi, la couche physique du système de fichier (i.e. l'organisation des données sur disque) reste au choix de l'administrateur du système et sa mise en œuvre est inchangée. De petites adaptation des systèmes de fichier existants sont toutefois à envisager en ce qui concerne une éventuelle contention au niveau des bitmaps d'allocation. Quant aux inodes, il sont gérés par un troisième et dernier cache global.

5.4 Interaction avec le système de mémoire virtuelle

Pour que le système de cache global soit optimal, il s'appuie sur une politique de remplacement de page globale [3]. Des informations statistiques échangées à intervalles de temps réguliers permettent de déterminer s'il faut invalider tel duplicat, injecter telle page sur tel nœud, ou synchroniser telle autre sur disque. Un avantage est qu'il devient possible pour des processus distribuées sur la grappe de travailler en mémoire partagée en projetant simplement un fichier commun dans leur espace d'adressage. Il devient inutile d'utiliser une

bibliothèque de mémoire virtuellement partagée. La politique de remplacement globale permet de plus d'effectuer des calculs *out-of-core* par projection d'un fichier de données de grande taille. Un autre avantage est la possibilité de créer des points de reprise de manière efficace en réutilisant le code qui gère l'écriture dans la zone d'échange.

5.5 RAID logiciel

La mise en œuvre logicielle des différents niveaux de RAID [8] est compatible avec le cache global, en particulier le *stripping* (pour les performances), le *mirroring* (pour la tolérance aux fautes) et la combinaison des deux. L'administrateur peut choisir la solution qui lui apportera le niveau de performance et de tolérance aux fautes souhaité. Le principe est qu'à partir d'un offset et un numéro de périphérique virtuels on dérive un offset et un numéro de périphérique réel. Ce sont ces derniers qui servent à adresser la page dans le cache global. La correspondance virtuel/réel est statique et connue par tous les nœuds qui utilisent la matrice RAID.

6 Conclusion

Dans ce papier, nous avons présenté une nouvelle façon d'envisager les entrées/sorties dans un système d'exploitation pour grappe. Elle combine différentes techniques (messages actifs, accès distant direct, RAID logiciel, mémoire partagée répartie) qui permettent d'exploiter le matériel de manière symétrique (même nœuds pour le calcul et les communications) au mieux de ses possibilités et donc d'obtenir un rapport performance/prix optimal. Le fait de gérer les méta-données avec autant de soin que les données, la rend utilisable pour tout type d'application et d'utilisation (exécutables, données des applications, répertoires de travail des utilisateurs). Elle offre une image unique et rend donc l'utilisation et l'administration de la grappe aussi simple qu'en environnement centralisé. Enfin elle offre des perspective d'intégration entre mémoire virtuellement partagée et entrées/sorties parallèles, aussi bien que des perspectives de mise en œuvre d'une interface MPI-IO. Nous développons activement un prototype dans le cadre de notre système d'exploitation pour grappe Kerrighed [6].

Références

[1] Carns (Philip H.), Ligon III (Walter B.), Ross (Robert B.) et Thakur (Rajeev). – PVFS: A parallel file system for linux clusters. *In: Proceedings of the 4th Annual Linux Showcase and Conference*. pp. 317–327. – Atlanta, GA, 2000.

[2] Cozette (Olivier). – Maximisation du débit d'entrées/sorties des grappes de stations par l'utilisation de l'accès distant direct. *In: 14èmes Rencontres Francophones du Parallélisme*, pp. 25–32.

[3] Feeley (M.), Morgan (W.), Pighin (F.), Karlin (A.), Levy (H.) et Thekkath (C.). – Implementing global memory management in a workstation cluster. *In : SOSP*, pp. 201–212.

[4] Kleiman (Steve R.). – Vnodes: An architecture for multiple file system types in sun UNIX. *In : USENIX Summer*, pp. 238–247.

[5] Li (Kai) et Hudak (Paul). – Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems (TOCS)*, vol. 7, n4, 1989, pp. 321–359.

[6] Morin (Christine), Gallard (Pascal), Lottiaux (Renaud) et Vallée (Geoffroy). – Towards an efficient single system image cluster operating system. *In : Proc. of International Conference on Architecture and Algorithms for Parallel Processing (ICA3PP 2002)*. pp. 370–377. – Pékin, Chine, octobre 2002.

[7] MPIF (Message Passing Interface Forum). – MPI-2: Extensions to the Message-Passing Interface. – Technical Report, University of Tennessee, Knoxville, 1996.

[8] Patterson (D. A.), Chen (P.), Gibson (G.) et Katz (R. H.). – Introduction to redundant arrays of inexpensive disks (RAID). *In : Digest of Papers for 34th IEEE Computer Society International Conference (COMPCON Spring '89)*, pp. 112–117. – San Francisco, CA, 1989.

[9] Preslan (Kenneth W.). – A 64-bit, shared disk file system for linux, 1999.

[10] Sandberg (R.), Goldberg (D.), Kleiman (S.), Walsh (D.) et Lyon (B.). – Design and implementation of the sun network filesystem. *In : Proceedings of the Summer*.

[11] von Eicken (Thorsten), Culler (David E.), Goldstein (Seth Copen) et Schauser (Klaus Erik). – Active messages: A mechanism for integrated communication and computation. *In : 19th International Symposium on Computer Architecture*, pp. 256–266. – Gold Coast, Australia, 1992.

Table des matières

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399