Conception et mise en oeuvre d'un système d'entrées/sorties performant pour grappe

Gaël Utard, Christine Morin

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Conception et mise en œuvre d'un système d'entrées/sorties performant pour grappe

Gaël Utard and Christine Morin

N°5416

Décembre 2004

Systèmes communicants

ISSN 0249-6399 ISRN INRIA/RR--5416--FR

INRIA
RENNES

Conception et mise en œuvre d'un système d'entrées/sorties performant pour grappe

Gaël Utard * and Christine Morin †

Systèmes communicants
Projet Paris

Rapport de recherche n°5416 — Décembre 2004 — 10 pages

Résumé : Les utilisateurs de grappes sont intéressés par une vue globale et cohérente du système de fichiers. Or les systèmes de gestion de fichiers distribués existants répondent mal aux besoins, tant en terme de fonctionnalités que de performances. Afin de prendre en compte l'aspect distribué de ce type d'architecture, nous proposons d'effectuer quelques ajouts qui s'insèrent au cœur des mécanismes d'entrées/sorties traditionnels du noyau. Nous avons mis en œuvre cette solution sous forme d'un patch pour Linux permettant l'utilisation efficace et cohérente de l'espace de stockage interne d'une grappe sur l'ensemble de ses nœuds.

Mots-clé : Grappe de calculateurs, Entrées/sorties, SGFD, 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

Design and Implementation of an Efficient I/O System for Clusters

Abstract: Cluster users are interested in a global and consistent view of the file system. Existing distributed file systems do not meet their requirements in terms of functionality and performance. In order to take the distributed aspect into acount, we propose to do some additions in the heart of the traditionnal I/O mechanismes of the kernel. We have implemented this solution as a linux patch that permit an efficient and consistent use of the internal storage space on all cluster nodes.

Key-words: Cluster, I/O, Distributed File System, Distributed Operating System

1 Introduction

Le succès des grappes de calculateurs ne se dément pas. L'addition d'un grand nombre de composants banalisés permet en effet d'obtenir un très bon rapport entre performance et coût. Ceci est vrai en ce qui concerne la puissance de calcul, mais l'est également en ce qui concerne le stockage. Cependant, dans ce domaine, les potentialités ont du mal à se concrétiser. Lorsque la réplication (fut-elle simplifiée par des outils d'administration) ou le traditionnel protocole NFS ne suffisent pas, il faut souvent se résoudre à l'achat, coûteux, d'une baie de stockage (NAS) et d'un réseau dédié (SAN). Pourtant, si ce genre de matériel est spécifique, il est paradoxalement constitué des mêmes composants banalisés que notre grappe dont les propre disques sont sous-utilisés. Les utilisateurs de grappes de taille moyenne voulant rentabiliser les disques internes sont relativement démunis. Le support logiciel aux entrées/sorties pour les grappes nécessite un pas en avant. Mais auparavant, nous devons prendre un peu de recul et passer en revue les mécanismes traditionnels d'entrées/sorties afin de mieux appréhender les difficultés induites par l'aspect distribué de l'architecture qui nous intéresse.

2 Les entrées/sorties

2.1 Rôle du système d'exploitation

L'ensemble des applications ne peut pas prendre en considération l'ensemble des périphériques matériels sur le marché. Il faut leur fournir une vision unifiée de l'accès au matériel. De plus, différentes applications doivent coexister sur un même système. Il faut donc partager la ressource physique (le disque) en une multitude de ressources logiques (fichiers) allouées de manière indépendante. Ces deux aspects sont traditionnellement pris en compte par le système d'exploitation. Dans un noyau Unix, on peut identifier trois couches dans le système d'entrées/sorties (figure 1).

2.2 Gestionnaire de périphérique

En bas, on trouve le gestionnaire de périphérique. Il a pour mission de transférer des blocs depuis le disque vers la mémoire et inversement. Chaque bloc est identifié par un doublet (numéro de périphérique, numéro de bloc). Le numéro de bloc correspond à la position physique du bloc sur le périphérique.

2.3 Système de fichiers

Au milieu, on trouve le système de fichiers. Il a pour mission de transférer, non plus des blocs, mais des pages entre le disque et la mémoire. Chaque page est identifiée par un triplet (numéro de périphérique, numéro de fichier, numéro de page). Le numéro de fichier est un identifiant unique pour le périphérique donné. Le numéro de page correspond à la

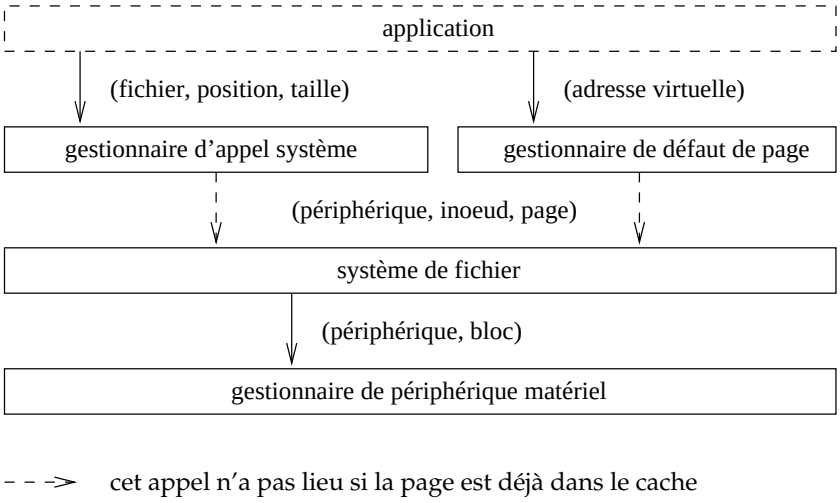

FIG. 1 – *Mécanismes d'entrées/sorties*

position de la page dans le fichier. Le système de fichier gère la placement des données et permet ainsi le multiplexage d'une ressource physique unique (le disque) en une multitude de ressources logiques (les fichiers). Il fait appel au gestionnaire de périphérique pour transférer l'ensemble des blocs formant une page.

2.4 Gestionnaires d'appel système et de défaut de page

En haut, on trouve les gestionnaires d'appel système (`read()` ou `write()`). Ils ont pour mission de copier les données entre l'espace mémoire utilisateur et l'espace mémoire noyau. La zone concernée est spécifiée par un décalage dans le fichier (le pointeur de fichier) et par une taille (en octets). Les gestionnaires d'appel système font appel au système de fichier pour ramener en mémoire physique l'ensemble des pages contenant partiellement ou totalement cette zone. Au même niveau, on trouve également un gestionnaire de défaut de page. Lorsqu'un fichier est lu à travers une projection en mémoire virtuelle et qu'il se produit un défaut de page, le gestionnaire fait appel au système de fichiers pour amener cette page en mémoire physique.

2.5 Cache des pages

La présence d'une page en mémoire physique n'est indispensable que lors d'une opération de copie depuis ou vers l'espace utilisateur ou lors d'un accès à travers le mécanisme de mémoire vituelle. Cependant, comme les entrées/sorties sont relativement lentes

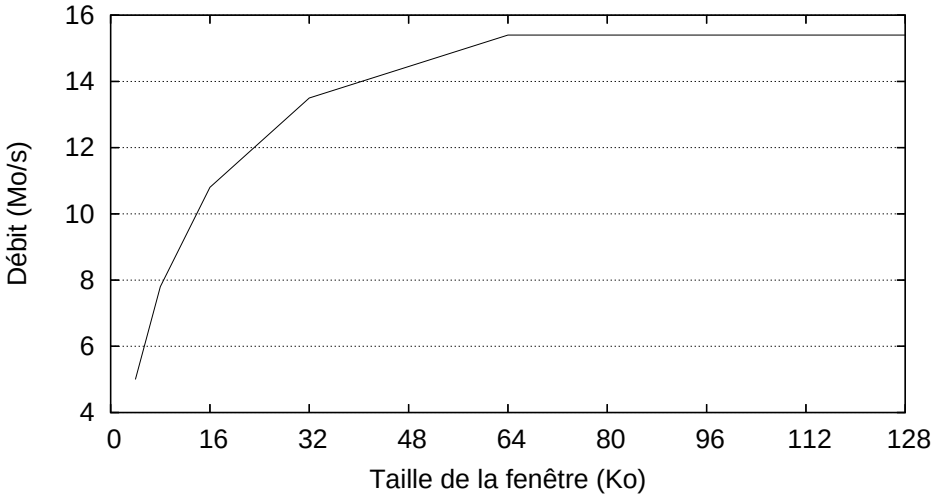

FIG. 2 – *Lecture en avance*

et qu'elles sont sujettes à une bonne localité temporelle, il est intéressant de garder les pages le plus longtemps possible en mémoire dans l'éventualité d'une réutilisation. L'ensemble des page dédiées aux entrées/sortie forme ainsi ce que l'on appelle le cache des pages.

2.6 Lecture en avance

Lorsqu'une lecture porte sur une zone assez grande, il se produit une alternance entre le transfert disque et la copie noyau/utilisateur. Cela ne permet pas d'exploiter la bande passante maximale du disque. Afin d'effectuer un recouvrement de ces deux opérations, il faut commander le transfert de la page suivante avant d'avoir fini la copie de la page en cours. Cela nécessite de connaitre à l'avance les pages que l'on va lire, ce qui est possible dans un cas très spécifique, mais très fréquent: la lecture séquentielle. Dans le cas d'une projection de fichier en mémoire virtuelle, on essaie que les pages soient présentes en mémoire physique avant qu'il n'y ait de défaut sur ces pages. Cela permet de recouvrir le calcul et les entrées/sorties. La figure 2 montre l'impact de la lecture en avance sur le débit en fonction de la taille de la fenêtre.

2.7 Sémantique de partage

Des applications utilisant des entités logiques distinctes peuvent partager une entité physique commune sans avoir à ce soucier des problèmes de cohérence. Qu'en est-il lorsque deux processus partagent un même fichier ? Cela dépend de la sémantique que l'on donne à ce partage. Deux sémantiques sont courament utilisées. La première, appelée sémantique Unix spécifie que toute écriture sur un fichier ouvert par un utilisateur est immédiatement visible par les autres utilisateurs possédant ce fichier ouvert en même temps. La seconde, appelée sémantique de session spécifie que les écritures restent invisibles pour les autres utilisateurs possédant ce fichier ouvert en même temps. Par contre, lorsque le fichier est fermé par le processus qui y a écrit, les modifications deviennent visibles par tous les processus qui ouvrent le fichier à partir de cet instant.

En environnement centralisé, les données sont physiquement partagées. C'est donc naturellement la sémantique Unix qui est utilisée. En environnement distribué, cette facilité n'existe pas.

3 Les systèmes de fichier distribués

3.1 NFS

NFS [5,] est un système de fichier distribué largement répandu. Son principe de fonctionnement est assez simple. Les requêtes faites au système de fichier NFS sur les différents clients sont redirigées vers un démon présent sur le serveur. Ce démon est un processus comme un autre et il interagit avec le noyau à travers les appels système standards. Comme le noyau voit provenir l'ensemble des opérations depuis cet unique processus, il ne peut pas assurer une sémantique de partage. Et le procole NFS ne prévoit pas d'assurer par lui-même une sémantique particulière, pas plus qu'il ne prévoit de gérer la cohérence des caches entre ses différents clients. Comme désactiver le cache des pages sur les clients ferait chuter les performances, NFS se contente de fixer un certain délai au bout duquel une écriture est forcément répercutée sur le serveur et un autre délai au bout duquel une donnée en cache est forcément revalidée auprès du serveur. En conclusion, NFS assure seulement qu'une modification sera vue par les autres clients après un temps fini pouvant atteindre une minute. Ce temps peut être réduit, mais au détriment des performances.

3.2 Coda

Coda [6,] met en œuvre une sémantique de session. Lors de l'ouverture, le fichier est entièrement répliqué sur le disque local. L'ensemble des opérations de lecture/écriture sur ce fichier s'effectuent alors sans communication avec le serveur. Lors de la fermeture, les modifications sont renvoyées au serveur. Cette solution supporte les déconnexions et offre une sémantique bien définie. Elle est bien adaptée à un réseau de stations de travail. Par

contre elle pose des problèmes sur une grappe où les applications ont besoin de la sémantique Unix.

3.3 GFS

L'objectif de GFS [4,] est de partager une baie de stockage externe (NAS) reliée à différents clients par un réseau dédié (SAN). Ce partage s'effectue donc au niveau des blocs et non au niveau des fichiers. GFS garantit une sémantique Unix en protégeant toutes les opérations par des verrous sur les fichiers. Ces verrous sont relativement sophistiqués et permettent de garder les données en cache dans un certain nombre de situations. Cependant, comme la granularité du verrou est le fichier, l'écriture d'un seul octet provoque l'invalidation de l'ensemble du fichier sur l'ensemble de la grappe. De plus, GFS ne sait pas tranférer des données directement d'un nœud à un autre. Pour qu'une modification soit vue par un autre nœud, elle doit préalablement être écrite sur le disque externe puis relue à partir de celui-ci.

3.4 Lustre

Lustre vise les très grandes échelles, avec plusieurs milliers de clients et une masses de données à stocker très importante. Il nécessite de dédier un nœud en tant que serveur de métadonnées et un certain nombre d'autres nœuds en tant que serveurs de données. Les clients s'enregistrent auprès du serveur de métadonnées et, une fois qu'ils ont obtenu les verrous nécessaires, ils interagissent directement avec les serveurs de données.

4 Conception d'un cache des pages distribué

En résumé, la difficulté se situe essentiellement au niveau de la cohérence des données présentes sur les différents nœuds. Cette difficulté concerne en réalité bien plus le cache des pages que le système de fichier proprement dit. Nous nous sommes donc attachés à rendre le cache des pages global et cohérent.

4.1 Nommage

Le premier problème qui se pose est de nommer les pages. Habituellement, une page est identifiée à l'aide du triplet (numéro de périphérique, numéro de nœud, numéro de page). Le numéro de périphérique se décompose lui même traditionnellement en un numéro majeur (qui identifie le type de périphérique) et un numéro mineur (qui distingue les périphériques de même type). Or les périphériques matériels sont attachés à un nœud donné. Nous avons donc décidé d'ajouter une troisième composante au numéro de périphérique: le numéro du nœud auquel il est attaché. Ainsi, des pages issues de périphériques semblables, mais provenant de deux nœuds différents peuvent coexister sur un même nœud être confondues.

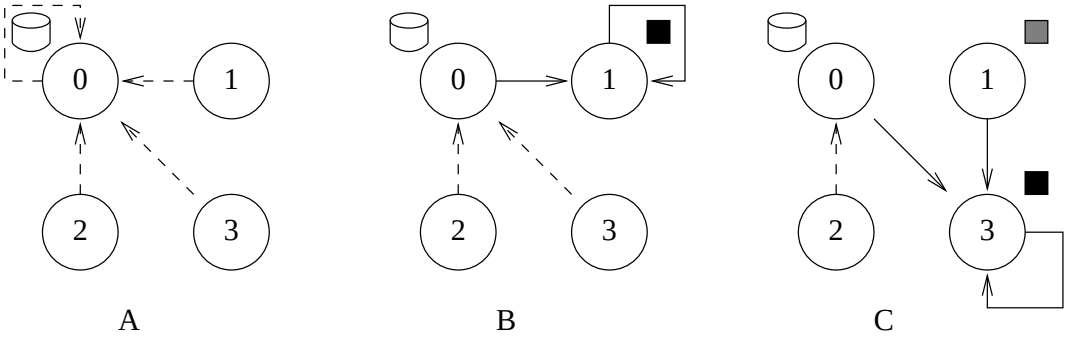

FIG. 3 – *Graphe des propriétaires probables pour une page*

4.2 Localisation

Maintenant qu'une page peut être nommée de manière globale, il faut pouvoir la localiser et la rapatrier sur le nœud qui en a besoin. Pour cela, on maintient sur chaque nœud une table de hachage qui indique quel est le propriétaire probable de chacune des pages. En réalité, pour minimiser la taille des tables, l'absence d'entrée dénote que le propriétaire probable est le nœud d'origine de la page (c'est à dire le nœud auquel est attaché le périphérique dont provient la page en question). Lorsqu'un nœud reçoit une requête et qu'il n'est pas propriétaire, il transmet la requête au propriétaire probable, puis met à jour le propriétaire probable comme étant le nœud d'origine de la requête. Les différentes informations de propriété d'une page donnée forment un graphe orienté non cyclique permettant de retrouver le propriétaire réel. Une démonstration à ce sujet peut être trouvée dans [3,].

La figure 3 A montre une situation initiale avec quatre nœuds. La page n'est pas présente en cache. Elle est présent sur le disque du nœud 0. Les tables sont vides. La figure 3 B montre la même situation après une requête du nœud 1 vers le nœud 0. Le nœud 0 a lu la page et l'a envoyée au nœud 1. Les tables des deux nœuds concernés spécifient que le propriétaire probable est le nœud 1. La figure 3 C montre toujours la même situation mais après une requête du nœud 2 vers le nœud 0 (qu'il pense être propriétaire). Le nœud 0 a transmis la requête au nœud 1 qui a répondu directement au nœud 2. Le nœud 2 est le nouveau propriétaire et possède la copie principale. Le nœud 1 possède une copie secondaire. Les trois nœuds concerné ont mis à jour leur propriétaire probable.

4.3 Cohérence

Afin de garantir la sémantique Unix, nous devons assurer une cohérence séquentielle pour chacune des pages. Nous avons choisi d'invalider toutes les copies d'une page avant de d'y procéder à une écriture. A cet effet, le propriétaire de la page connait l'ensemble des

nœuds en possédant une copie. En fait, la gestion du cache global s'inspire fortement des techniques de mémoire partagée répartie [3,].

4.4 Métadonnées

Pour réaliser le multiplexage d'un disque physique en une multitude de fichiers, le système de fichier a besoin de métadonnées. Il y a en particulier celles qui décrivent les fichiers. Ce sont les inœuds. Chaque inœud contient des informations telle que la taille, le propriétaire, les permissions d'accès, mais il décrit également le placement physique du contenu du fichier sur le disque. Chaque opération sur un fichier nécessite que son inœud soit présent en mémoire. Le noyau maintient un cache des inœuds au même titre que le cache des pages. Nous assurons donc la cohérence du cache des inœud selon les même principes que le cache des pages.

Les répertoires ne nécessites pas de traitement spécifique car ce sont de simples fichiers. Les champs de bits servant à repérer les blocs libres peuvent également prendre place dans le cache des pages. Cependant il est préférable de les traiter de manière spécifique car ils forment des points de contention. Ces champs de bits se prêtent particulièrement bien à une cohérence relachée. Un bloc libéré par un nœud peut ne pas être immédiatement disponible pour les autres nœud. Par contre allouer un bloc dans un champ de bits nécessite d'y avoir un accès exclusif.

4.5 Synchronisation

Après une écriture, les pages du cache qui ont été modifiées nécessitent d'être synchronisées sur le disque. Une page modifiée peut avoir des copies, mais seule la copie principale sera synchronisée. L'opération de synchronisation s'effectue directement vers le périphérique concerné sans amener une copie de la page dans le cache des pages du nœud d'origine. Comme le périphérique matériel peut être situé sur un nœud distant, un gestionnaire de périphérique virtuel effectue les redirection nécessaires à travers le réseau.

5 Mise en œuvre

Nous avons mis en œuvre cette solution sous Linux 2.6.9. Il s'agit d'un patch d'environ 5.000 lignes appelé `linux-cluster`. Seules 10% de ces lignes sont des ajouts ou des modifications dans les fichiers existant. Le reste est créé dans un répertoire spécifique. L'installation ne nécessite que de patcher et de recompiler le noyau. Il n'y a pas de fichier de configuration. Au démarrage, `linux-cluster` détermine son numéro de nœud en fonction de son adresse IP. Les périphériques bloc de l'ensemble des nœuds de la grappe apparaissent automatiquement sous `/dev/<numero_de_noeud>/`. Il est alors possible de monter une partition au format `ext2` [1,] en parallèle sur plusieurs nœuds. Il suffit pour cela d'utiliser la commande `mount` standard sans option particulière.

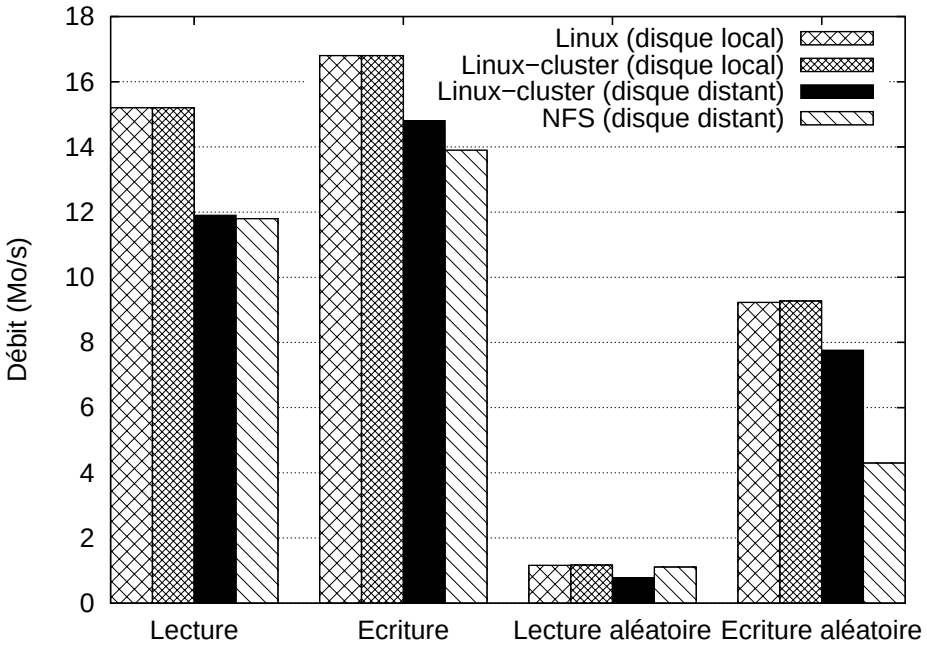

FIG. 4 – *Performances*

6 Evaluation

Les mesures de performance (figure 4) ont été réalisées à l'aide du benchmark `iozone` [2,] sur un fichier de 100 Mo et en prenant en compte le temps de synchronisation sur disque. Nous avons mesuré le débit en lecture, en écriture, en lecture aléatoire et en écriture aléatoire. On constate que le patch n'introduit aucun surcoût pour les accès au disque local. Lors des accès au disque distant, linux-cluster se comporte mieux que NFS sauf pour la lecture aléatoire. Les différences entre les accès locaux et les accès distants s'expliquent par un mauvais recouvrement entre les communications et les entrées/sorties. La lecture en avance provoque des requêtes pour des groupes de 32 pages. Ces requêtes sont transmises groupées sur le réseau mais à la réception, elles sont traitées et transmises page par page au système de fichier. Un traitement plus efficace des requêtes améliorerait les performances.

7 Conclusion

Nous avons conçu une adaptation des couches d'entrées/sorties du noyau à même d'assurer les fonctionalités (utilisation des disques internes, sémantique Unix) et les performances attendues sur une grappe. La mise en œuvre est fonctionnelle. Elle prouve la faisabilité et la simplicité d'utilisation d'une telle approche. Il convient cependant d'en optimiser le performances et de d'en évaluer la passage à l'échelle. Il convient également de s'intéresser à la consolidation de l'ensemble de disque de la grappe en un disque virtuel unique afin d'améliorer les performance, la capacité et la répartition de charge. Cela peut être réalisé en utilisant les techniques de RAID logiciel et de gestion de volumes logique que linux met déjà en œuvre. Enfin, le support d'opérations collectives mériterait d'être étudié. Il permettrait de fournir un environnement de choix pour l'exécution d'applications MPI-IO [7,].

Références

[1] R. Card, T. Ts'o, and S. Tweedie. Design and implementation of the second extended filesy system. In *Proceedings of the First Dutch International Symposium on Linux*, December 1994.

[2] http://www.iozone.org.

[3] Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems (TOCS)*, 7(4):321–359, 1989.

[4] Kenneth W. Preslan. A 64-bit, shared disk file system for linux. In *6th IEEE Mass Storage Systems Symposium*, 1999.

[5] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the sun network filesystem. In *Proceedings of the Summer USENIX Conference*, 1985.

[6] Mahadev Satyanarayanan. Scalable, secure, and highly available distributed file access. *Computer*, 23(5):9–18, 20–21, 1990.

[7] Rajeev Thakur, William Gropp, and Ewing Lusk. On implementing MPI-IO portably and with high performance. In *Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems*, pages 23–32, 1999.

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