Interaction efficace entre les réseaux rapides de stockage distribué dans les grappes de calcul

Brice Goglin, Olivier Glück, Primet, Pascale Vicat-Blanc

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

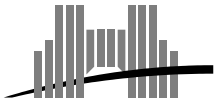
Interaction efficace entre les réseaux rapides et le stockage distribué dans les grappes de calcul

Brice Goglin,
Olivier Glück,
Pascale Vicat-Blanc Primet

Janvier 2006

Research Report N° 2006-04

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`



Interaction efficace entre les réseaux rapides et le stockage distribué dans les grappes de calcul

Brice Goglin, Olivier Glück, Pascale Vicat-Blanc Primet

Janvier 2006

Abstract

Parallel applications running on clusters require both high-performance communications between nodes and efficient access to the storage system. We propose to improve the performance of distributed storage systems in clusters by efficiently using the underlying high-performance network to access distant storage systems. We show that storage requirements are very different from those of parallel computation and that a modification of the network programming interface is required. We detail several propositions for these interfaces which make their utilization easier in the context of distributed storage. Performance evaluations show that their integration makes it easy to use and very efficient in the context of storage.

Keywords: Distributed storage, remote file access, cluster, high-speed network, zero-copy, unexpected messages, event notification.

Résumé

Les applications parallèles s'exécutant sur les grappes nécessitent à la fois des communications performantes entre les différents nœuds et des accès efficaces au système de stockage. Nous proposons dans ce travail d'améliorer les performances du stockage distribué dans les grappes en utilisant au mieux le réseau haute performance sous-jacent pour accéder au stockage distant. Nous montrons que les besoins du stockage sont très différents de ceux du calcul parallèle et proposons différentes solutions pour résoudre, au niveau du système de fichiers, les problèmes liés au contrôle du réseau mais montrons qu'il est nécessaire de modifier l'interface de programmation réseau et le système d'exploitation pour venir à bout des difficultés liées au transfert de données. Des expérimentations montrent qu'elles permettent une utilisation aisée et efficace des réseaux rapides dans le cadre du stockage distribué.

Mots-clés: Stockage distribué, accès aux fichiers distants, grappe de calcul, réseau haute performance, zéro-copie, messages inattendus, notification d'événements.

1 Introduction

Les besoins croissants en puissance de calcul informatique dans des domaines aussi variés que la simulation numérique, la physique des particules, la génomique ou la réalité virtuelle ont nécessité une évolution régulière du matériel. Cela s'est traduit par le développement dans les années 1970 de super-calculateurs parallèles utilisant des technologies réseau dédiées. Le coût prohibitif de ces machines a conduit à leur disparition progressive dans les années 1990 au profit des grappes de stations. Cette solution, même si elle peut être moins puissante, se révèle plus extensible et générique, et par conséquent moins onéreuse. L'obtention de performances élevées lors de l'exécution d'applications parallèles sur des grappes de stations a nécessité de réduire au maximum le coût des communications entre les différents instances du programme.

De nombreuses recherches ont donc été menées sur les systèmes de communication dans les grappes, permettant d'une part des transferts de données très rapides et d'autre part, une gestion autonome des communications par le matériel. Ces travaux ont abouti à des réseaux dédiés aux grappes, présentant une latence très faible (quelques microsecondes) et une bande passante très élevée (plusieurs centaines de gigaoctets par seconde). Les protocoles traditionnels ayant du mal à utiliser efficacement ce matériel [BGM99], des interfaces spécialisées ont été conçues, par exemple BIP [PT97] sur MYRINET. Le modèle du passage de message est désormais le plus souvent utilisé pour la programmation des applications parallèles [BM00], en particulier par l'interface MPI [For94]. Ces implantations logicielles ayant été conçues pour supporter efficacement ces réseaux spécifiques, les communications entre les différentes instances d'une application parallèle peuvent bénéficier aisément de ces performances très élevées.

D'autre part, l'exécution performante des applications parallèles passe également par un accès efficace aux données stockées sur disque. Les nombreux travaux menés dans le domaine des systèmes de fichiers distribués s'étaient tout d'abord traduits par le développement de protocoles complexes permettant au client d'être indépendant du serveur via des stratégies de cache et de maintien de cohérence [NWO88]. La généralisation des grappes de centaines ou milliers de stations a ensuite imposé au serveur de supporter les besoins très importants de nombreux clients. Cela a conduit à des travaux spécialisés pour ce contexte [Sch99], avec tout d'abord des systèmes de stockage partagé sans serveur en utilisant des mécanismes de synchronisation globale tels que GFS [SRO96] ou XFS [ADN+96]. Des modèles orientés clients-serveurs tels que PVFS [LR99] ou LUSTRE [Sch03] ont ensuite été developpés pour répondre aux besoins des grappes de calcul en parallélisant le serveur, c'est-à-dire en répartissant la charge de travail et les données à traiter sur différentes machines. Cependant, les communications entre client et server dans ce cadre restent très simples et profitent peu des performances du matériel sous-jacent. Pourtant les besoins en débit sont similaires voire supérieurs à ceux des communications inter-processus au sein des applications parallèles.

La spécialisation des technologies réseau pour obtenir des communications performantes au sein des applications parallèles les rend difficiles à utiliser dans d'autres contextes où les contraintes peuvent être très différentes. L'adaptation de ces technologies au stockage distribué, en particulier l'intégration du modèle de programmation de ces réseaux dans les différents contextes du stockage doit être étudiée. Cela implique d'une par,t de s'intéresser aux transferts de données entre les différents nœuds, notamment entre un client et un serveur, et d'autre part, au contrôle des communications sur ces nœuds, c'est-à-dire à la gestion des

communications et des événements qu'elles engendrent.

Cet article présente une étude de l'utilisation efficace des réseaux rapides dans le cadre du stockage distribué. L'idée est de proposer des optimisations complémentaires aux stratégies de cache et de parallélisation afin d'améliorer les performances de l'accès au stockage distant en profitant au maximum du matériel réseau disponible dans les grappes. Nous présentons tout d'abord en partie 2 une modélisation de l'accès au stockage distant afin de localiser les optimisations potentielles. Nous étudions ensuite en partie 3 la mise en œuvre des accès distants dans le contexte du noyau. Nous distinguons le plan données, où les contraintes sur la gestion mémoire sont problématiques, et le plan contrôle, où les notifications d'événements et le contrôle de flot sont étudiés. Nous proposons différentes solutions pour adapter les interfaces des réseaux rapides existantes aux besoins du stockage distribué puis évaluons leur performance en partie 4 avant de conclure en partie 5.

2 Analyse des interactions entre le système de stockage et le réseau

Pour mettre en évidence les optimisations possibles lors de l'accès au stockage distant, nous présentons tout d'abord une modélisation de cet accès puis détaillons les différents coûts mis en jeu.

2.1 Modélisation des accès aux fichiers distants

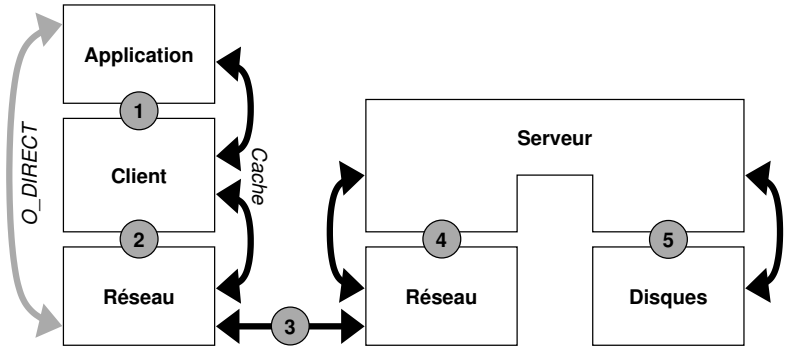


Figure 1: Différentes étapes d'accès aux fichiers distants. Les différentes interactions logicielles sont numérotées de 1 à 5. Les flèches noires désignent le chemin habituel des données. La flèche grise présente un type d'accès particulier au stockage que l'application peut demander.

La figure 1 illustre les différentes étapes d'accès aux fichiers distants. Le délai total T_{Total} d'accès aux fichiers distants se décompose en quatre termes correspondant aux délais client, réseau, serveur et disques (équation 1).

$$T_{\text{Total}} = T_{\text{Client}} + T_{\text{Réseau}} + T_{\text{Serveur}} + T_{\text{Disques}} \quad (1)$$

Pour analyser plus finement ces différents délais, nous introduisons les notations suivantes :

T_{Stockage}	Temps logiciel d'accès au gestionnaire de stockage local
$T_{\text{Données}}$	Temps logiciel de transfert des données lors des communications réseau
$T_{\text{Contrôle}}$	Coût logiciel du contrôle des communications réseau

Les délais disques T_{Disques} et réseau $T_{\text{Réseau}q}$ dépendent du matériel (temps d'accès et débit), et sont aujourd'hui bien maîtrisés. Les coûts client et serveur dépendent par contre des couches logicielles. Du côté client, il s'agira d'accéder depuis l'application au module client gérant le stockage distant depuis l'application (interaction 1 et coût $T_{\text{Stockage1}}$), puis effectuer les communications réseau (interaction 2) ce qui implique de gérer les contraintes liées au contrôle $T_{\text{Contrôle2}}$ et au transfert de données $T_{\text{Données2}}$. Du côté serveur, il faut gérer les communications réseau ($T_{\text{Contrôle4}}$ et $T_{\text{Données4}}$) puis accéder au sous système de stockage (interaction 5 et coût $T_{\text{Stockage5}}$).

$$T_{\text{Client}} = T_{\text{Stockage1}} + T_{\text{Contrôle2}} + T_{\text{Données2}} \quad (2)$$

$$T_{\text{Serveur}} = T_{\text{Contrôle4}} + T_{\text{Données4}} + T_{\text{Stockage5}} \quad (3)$$

Contrairement aux coûts liés au matériel, ces coûts logiciels permettent d'envisager un certain nombre d'optimisations. L'interaction 1 entre l'application et le client d'accès aux fichiers distants a été abondamment étudiée. Des travaux ont mené à l'amélioration de l'interface de programmation pour satisfaire les besoins des applications parallèles, avec notamment des accès parallèles, collectifs, vectoriels et/ou asynchrones dans MPI-IO [BdRC93] ou des fonctionnalités dédiées à une meilleure utilisation du réseau sous-jacent dans DAFS [Mag02a]. Le coût $T_{\text{Stockage1}}$ dans le client reste assez faible dans le cas d'une implantation en espace utilisateur. Par contre, une mise en œuvre dans le noyau impose la traversée des couches supérieures du système d'exploitation et une copie des données dans le cache. Le surcoût atteint alors environ 15 μs pour quelques kilo-octets de données.

Du côté du serveur, l'interaction 5 entre le processus serveur et son système de stockage local a été longuement étudiée dans le cadre général des accès aux fichiers haute performance. Le coût $T_{\text{Stockage5}}$ associé varie là aussi considérablement selon la mise en œuvre.

Par contre, le délais liés à l'utilisation du réseau ont été peu étudiés dans le cadre de l'accès aux stockage distant. Nous étudions maintenant les spécificités des réseaux rapides afin de mettre en évidence les contraintes imposées aux systèmes de stockage distribués et les coûts $T_{\text{Contrôle}}$ et $T_{\text{Données}}$ qu'elles impliquent.

2.2 Les spécificités des réseaux rapides

Les réseaux rapides utilisés dans les grappes de calculateurs se caractérisent par des performances très élevées (quelques microsecondes de latence et plusieurs centaines de mega-octets par seconde de débit) mais aussi par des innovations matérielles et logicielles très spécifiques. La nécessité d'obtenir des performances les plus élevées possibles pour les communications en espace utilisateur entre différentes instances d'une application MPI a conduit à un certain nombre de caractéristiques et d'optimisations des interfaces de programmation des réseaux rapides pour ce contexte : court-circuit du système d'exploitation, suppression des copies mémoire, communications asynchrones et contrôle de flot adapté au trafic homogène.

2.2.1 Des communications OS-Bypass

Les applications parallèles s'échangeant beaucoup de messages nécessitent, pour s'exécuter rapidement, une latence minimale. Le chemin critique entre l'application et le réseau doit donc éviter de traverser de nombreuses couches logicielles dans le système d'exploitation. Les réseaux rapides court-circuitent complètement le système lors de la soumission des requêtes puis la notification de leur terminaison (*OS-Bypass*).

Cette optimisation est très efficace puisqu'elle contribue à la possibilité d'atteindre des latences logicielles de quelques microsecondes. Mais elle impose des contraintes fortes lors de la programmation de du réseau puisqu'il a été conçu pour être utilisé en espace utilisateur. Les systèmes de stockage distribué étant souvent mis en œuvre dans le système d'exploitation, cette contrainte peut induire un surcoût important du contrôle des communications $T_{\text{Contrôle}}$.

2.2.2 Des communications zéro-copie

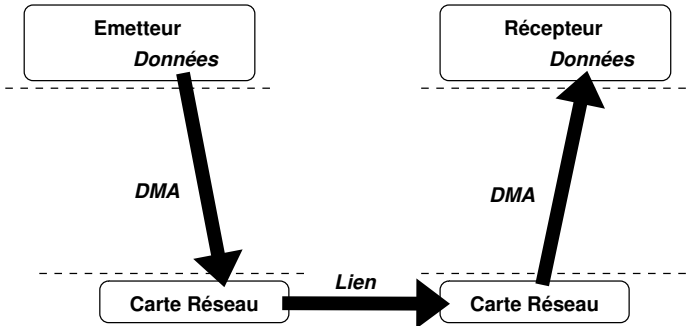


Figure 2: Transfert de données zéro-copie et *OS-bypass* sur un réseau de grappe.

Les applications parallèles s'exécutant sur les grappes de calculateurs souhaitent disposer de toute la puissance de calcul disponible. Il faut donc éviter de consommer des cycles processeur pour effectuer des copies mémoire. Les réseaux rapides proposent donc des communications zéro-copie entre le réseau et l'espace utilisateur de l'application initiées par DMA (*Direct Memory Access*) depuis la carte d'interface réseau (voir la figure 2).

Associées aux stratégies OS-bypass, ces communications zéro-copie posent le problème de la traduction des adresses virtuelles manipulées par l'application en adresses physiques gérées par le matériel. Cette traduction est traditionnellement effectuée par le système d'exploitation, ce qui est impossible dans le cas de l'OS-bypass. Les interfaces de programmation des réseaux rapides utilisent donc souvent une stratégie nommée enregistrement mémoire pour effectuer cette traduction [WBvE97]. Cependant, cet enregistrement impose une gestion avancée de la mémoire par l'application et la préparation des zones mémoire avant les communications, ce qui n'est habituellement pas fait dans les systèmes de stockage distribué. Il aura donc un coût important pour le traitement des données $T_{\text{Données}}$.

2.2.3 Une interface asynchrone de programmation

Les applications parallèles souhaitant disposer au maximum du processeur pour effectuer leurs calculs, elles ne peuvent pas se permettre d'être bloquées pendant le traitement d'une communication. Le traitement des communications dans le réseau rapide est donc déporté dans la carte d'interface afin de laisser le processeur disponible pour les calculs de l'application. Des requêtes asynchrones de communication sont soumises à la carte d'interface qui les traite en arrière plan. L'application recouvre ce traitement des communications par des calculs puis vient vérifier leur terminaison plus tard.

Ce modèle de programmation basé sur les événements est très différent du modèle classique de l'interface bloquante SOCKET et il est important de bien l'utiliser afin que les communications soient bien recouvertes. Sa gestion peut également être une part importante du coût du contrôle des communications $T_{\text{Contrôle}}$.

2.2.4 Un trafic homogène

Les applications parallèles présentent une utilisation particulière du réseau dans la mesure où les communications y sont souvent déterministes (le récepteur sera la plupart du temps conscient de l'arrivée d'un message avant son arrivée effective) et réparties de manière homogène (afin de minimiser les goulets d'étranglement). Par ailleurs, le cœur du réseau est suffisamment dimensionné pour éviter les congestions dans ce type de trafic. Le protocole de contrôle de flot des réseaux rapides est conçu pour cette utilisation, avec des stratégies du type *stop-and-go* ou *backpressure*, sans notion d'équité. Mais il peut se révéler inadapté pour les modèles du type client-serveur que l'on rencontre dans les systèmes de stockage distribué. Il s'agit ici d'un troisième élément du coût de contrôle des communications $T_{\text{Contrôle}}$.

2.3 Analyse des délais d'accès aux fichiers distants

Nous détaillons maintenant l'impact de ces spécificités des réseaux rapides sur les différentes mises en œuvre possibles de l'accès au stockage distant. Nous étudions successivement le cas de l'accès en espace utilisateur, en espace noyau et finalement l'accès bas-niveau par bloc.

2.3.1 Accès en espace utilisateur

Dans le cas d'une mise en œuvre de l'accès aux fichiers distants dans une bibliothèque client en espace utilisateur, le coût $T_{\text{Stockage1}}$ d'accès au module client du système de fichiers distribués est quasi-nul (un simple appel de fonction). Par contre, le processus serveur souffrira d'un grand coût $T_{\text{Stockage5}}$ d'accès à son dispositif de stockage local puisqu'il doit traverser toutes les couches logicielles de gestion du stockage.

Les coûts $T_{\text{Contrôle}}$ et $T_{\text{Données}}$ d'accès au réseau sont similaires à ceux d'une application parallèle puisque la mise en œuvre des communications est faite en espace utilisateur (voir la figure 3). Le coût $T_{\text{Données1}}$ de gestion des transferts de données peut être réduit en modifiant l'interface standard de programmation entre le client et l'application. Les études menées sur DAFS dans [RI04] ont en effet montré que l'intérêt de charger l'application d'adapter son utilisation mémoire en fonction des indications que la couche réseau lui fournit. Le développeur de l'application maîtrise son utilisation de la mémoire et peut alors l'organiser en tenant compte de ces indications. Un gain de 25 % en bande passante a ainsi été observé pour une application (base de données) sur DAFS. Des résultats similaires ont été obtenus

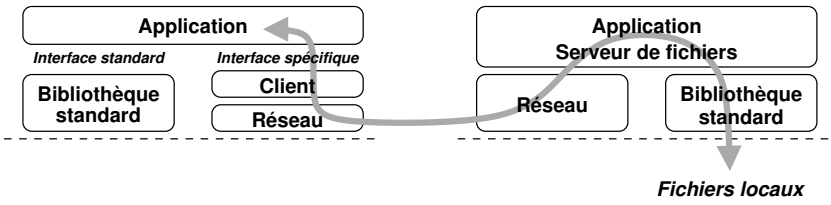


Figure 3: Accès aux fichiers distants en espace utilisateur par une interface de programmation spécifique.

avec MPI-IO sur DAFS [WP02]. De la même façon, l'application peut être chargée de gérer le contrôle des communications pour réduire le coût $T_{\text{Contrôle}}$.

L'utilisation d'une interface spécifique de programmation permet d'optimiser l'accès aux fichiers distants, mais a l'inconvénient d'imposer de ré-écrire les applications pour se conformer à cette interface. Certains projets préfèrent donc *surcharger* l'interface des bibliothèques standard et réaliser l'enregistrement mémoire et le contrôle des communications de manière transparente dans le client, ce qui peut rendre leur coup prohibitif. Nous avons étudié ce type de système en développant un prototype nommé ORFA (voir en partie 4.1).

Par ailleurs, certains systèmes préfèrent maintenir un cache dans le client de façon à éviter d'avoir à contacter le serveur distant pour chaque requête de l'application. Les transferts de données ne sont plus réalisés directement depuis l'espace mémoire de l'application mais depuis le cache. Dans ce cas, le coût d'accès au client de stockage $T_{\text{Stockage1}}$ contient une copie mémoire pour transférer les données entre l'application et le cache. Par contre, le client maîtrise l'utilisation mémoire du cache et peut donc l'adapter aux contraintes du réseau afin de réduire $T_{\text{Données}}$.

2.3.2 Mise en œuvre du serveur dans le noyau

Le coût d'accès au stockage du côté serveur peut être réduit de manière importante en mettant en place le module serveur dans le système d'exploitation afin d'éviter dela traversée de certaines couches système ainsi qu'une copie mémoire (voir la figure 4).

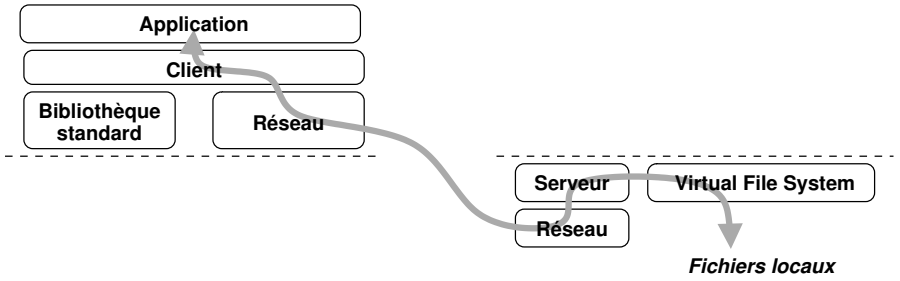


Figure 4: Accès en espace utilisateur vers un serveur noyau.

Cependant, l'utilisation du réseau haute performance dans une application interne au noyau impose la disponibilité d'une interface de programmation de ce réseau dans le noyau, ce qui est rarement le cas. Nous présenterons plus loin la mise en place d'interface noyau

dans le pilote MX des réseaux MYRINET. De plus, cette interface doit être compatible avec l'interface utilisateur si elle doit dialoguer avec un client en espace utilisateur. C'est par exemple impossible avec QUADRICS QSNET [PFHC03].

De plus, les zones mémoire manipulées du côté serveur sont en fait constituées des pages du système où les données sont cachées. Les caractéristiques très spéciales de ces pages (notamment le fait qu'elles n'ont généralement pas d'adresse virtuelle) peuvent les rendre difficile à utiliser avec l'interface de programmation du réseau rapide. Le coût $T_{\text{Données4}}$ peut alors devenir très important.

2.3.3 Accès par un système de fichiers dans le noyau

La stratégie la plus courante pour accéder aux fichiers distants consiste non pas à placer le client en espace utilisateur, mais dans le système d'exploitation. Le coût d'accès au client $T_{\text{Stockage1}}$ devient alors plus important du fait de l'utilisation d'appels-système. Mais il présente l'avantage de fournir un accès transparent à tous les types de fichiers, qu'ils soient locaux ou distants grâce à une couche de virtualisation dans le noyau (le *Virtual File System* dans le cas de LINUX).

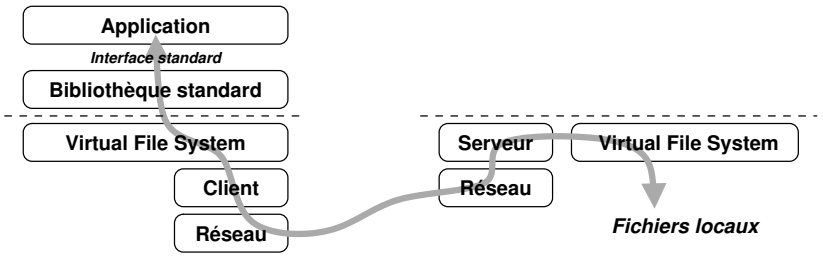


Figure 5: Accès par un système de fichiers implanté dans le noyau du client.

Ce modèle, représenté sur la figure 5, fournit depuis peu des fonctionnalités avancées telles que les accès asynchrones, directs (sans traverser le cache) ou vectoriels aux fichiers. Elles permettent de satisfaire les besoins des applications parallèles sans modifier l'interface standard de programmation par une bibliothèque en espace utilisateur. C'est la raison pour laquelle les systèmes de stockage distribué tels que PVFS et LUSTRE sont maintenant souvent utilisés dans le noyau.

Les communications mises en jeu dans ce modèle vont être initiées depuis le système d'exploitation du client. Il faut là aussi que l'interface de programmation du réseau supporte ce type de communication, ce qui est également très rare. Les échanges de données présentent les mêmes difficultés que du côté serveur puisque les zones de mémoire manipulées lors des communications sont celles du cache du système d'exploitation.

L'application a également la possibilité de demander à ce que les fichiers qu'elle manipule ne soient pas cachés du côté client (grâce au paramètre `O_DIRECT`, représenté par la flèche grise sur la figure 1). Les données doivent alors être échangées directement entre l'espace mémoire de l'application et le serveur distant. Ce type de transfert est conceptuellement très proche des communications zéro-copie traditionnelles sur un réseau haute performance, à ceci près qu'ici ils sont initiés dans le contexte du système d'exploitation et non dans celui de l'application elle-même. C'est une contrainte forte sur l'interface de program-

mation du réseau que de supporter ce type de communication puisque elle n'a généralement pas été prévue pour ce genre de zone mémoire.

Dans ces deux types d'accès (avec ou sans cache), le modèle de programmation classique des transferts des données sur un réseau rapide s'applique difficilement. Le coût $T_{\text{Données1}}$ des transferts de données peut devenir très grand.

2.3.4 Accès par bloc

La dernière méthode d'accès aux fichiers distants (*Network Block Device*) consiste à placer le client au niveau des périphériques bloc et à manipuler une partition ou un disque distant. Les données transférées sont alors exclusivement des blocs. Ce modèle est mis en œuvre dans les systèmes de stockage partagé sans serveur pour grappes tels que GFS [SRO96] ou GPFS [SH02].

Des travaux ont été réalisés du côté serveur de ce modèle, en particulier dans OPIOM [Geo01] et READ^2 [CRU03] mais très peu du côté client. Les communications impliquent des zones mémoire similaires au cas de la section précédente. Les contraintes sur l'interface de programmation du réseau sont les mêmes, elle doit pouvoir manipuler ces zones mémoire spéciales du noyau. Les différents coûts restent similaires.

2.4 Synthèse des interactions entre stockage et réseau

Dans la section précédente nous avons montré que les besoins de l'accès au stockage distant varient beaucoup selon l'implantation du client et du serveur et que l'interaction entre les couches logicielles du stockage et du réseau haute performance présente un certain nombre de difficultés.

Nous avons mesuré des coûts d'accès au module client depuis l'application ($T_{\text{Stockage1}}$) variant de 0 (pour un client en espace utilisateur) à 50 μs pour un client au niveau des blocs. Du côté serveur ($T_{\text{Stockage5}}$), les mêmes ordres de grandeur sont observables selon le type de mise en œuvre.

Le coût de transfert de données du côté client est élevé et peut-être amoindri en utilisant une interface spécifique qui charge l'application de gérer les problèmes d'enregistrement mémoire. La présence d'un cache du côté client permet également de limiter les contraintes de l'enregistrement mémoire aux zones du cache dans le client. Dans le cas d'une implantation dans le noyau, que ce soit du côté client ou serveur, les zones mémoire manipulées sont très différentes des habituels segments de mémoire en espace utilisateur et impose des coûts $T_{\text{Données}}$ importants, jusqu'à 100 μs pour quelques kilo-octets.

Les coûts $T_{\text{Contrôle}}$ d'utilisation de l'interface asynchrone de programmation varient en revanche peu avec le type d'implantation. Ils atteignent en général quelques centaines de nano-secondes ou quelques microsecondes.

La table 1 synthétise les différents coûts logiciels de l'accès aux fichiers distants. Nous avons montré que les coûts d'accès au stockage ou certaines copies mémoire sur le chemin d'accès aux fichiers distants étaient liés à la conception du client ou du serveur, en particulier la présence d'un cache et l'implantation en espace utilisateur ou dans le noyau. Par contre, les coûts d'utilisation du réseau, que ce soit dans le plan données à cause de l'enregistrement mémoire ($T_{\text{Données}}$) ou dans le plan contrôle pour gérer l'interface asynchrone de programmation et le contrôle de flot ($T_{\text{Contrôle}}$) restent présents dans tous les modèles, parfois avec des impacts différents. C'est à ces problèmes que nous nous intéressons.

Coût	Valeur actuelle	Valeur envisageable
T_{Stockage}	Jusqu'à 50 μs, fixé par l'implantation	
$T_{\text{Données}}$	1-100 μs	faible si utilisation efficace
$T_{\text{Contrôle}}$	100 ns-1 μs	très faible si utilisation efficace

Table 1: Synthèse des différents coûts logiciels lors de l'accès aux fichiers distants et gains envisageables.

Cette idée est motivée par l'observation que les principaux systèmes de fichiers distribués existants souffrent de la non-adaptation des interfaces de programmation des réseaux rapides de grappe. Par exemple, LUSTRE n'est disponible que sur très peu de réseaux haute performance. Les premières implantations étaient disponibles sur MYRINET/GM mais utilisaient toujours des copies mémoire car l'interface logicielle ne convenait pas à leurs besoins. Les développeurs de PVFS2 [Lig01] ont quant à eux décidé de ne pas accéder au réseau depuis l'espace noyau, mais au contraire, de remonter en espace utilisateur où un processus dédié se charge des communications. Ce choix est justifié dans la documentation par l'absence potentielle de support suffisant pour les accès au réseau depuis le noyau [PVF03]. Notre objectif est de réduire $T_{\text{Données}}$ et $T_{\text{Contrôle}}$ en fournissant une interface logicielle de programmation du réseau qui soit adaptée aux besoins du stockage distribué.

3 Mise en œuvre des accès distants sur les réseaux Myrinet

Nous présentons maintenant une étude des problèmes rencontrés lors de la mise en œuvre des systèmes de fichiers sur un réseau rapide. L'objectif est de proposer des modifications des systèmes existants, que ce soit le réseau ou le système d'exploitation, afin d'améliorer leur réponse aux besoins du stockage distribué dans les grappes de calculateurs. Il s'agit donc de réduire $T_{\text{Données}}$ et $T_{\text{Contrôle}}$.

3.1 Hypothèses de travail

Nous avons montré en partie 2.3.1 que les coûts liés à l'utilisation du réseau rapide pouvaient être réduits en laissant à l'application la responsabilité de gérer les problèmes liés au transfert de données et au contrôle des communications. Cette idée, qui a notamment été étudiée dans DAFS, consiste à modifier l'interface de programmation des accès aux fichiers qui est proposée à l'application. Mais elle présente l'inconvénient d'imposer une réécriture de l'application pour tenir compte des contraintes du réseau. Nous avons choisi de nous restreindre à l'**interface standard de programmation des accès aux fichiers**, ce qui permet à notre étude de s'appliquer à toutes les applications existantes.

Les systèmes de stockage distribué auxquels nous nous intéressons sont du type **client-serveur(s)**, en particulier PVFS et LUSTRE qui sont les répandus dans le monde académique du calcul haute performance. Ces systèmes respectent l'interface standard de programmation des accès aux fichiers et sont mis en œuvre en espace utilisateur ou dans le système d'exploitation. Leur étude nous permet donc de nous intéresser aux principaux types de mises en œuvre que nous avons détaillés en partie 2.3.

Nous utilisons le réseau MYRINET de MYRICOM [BCF+95] qui est le plus répandu dans le monde du calcul haute performance et dont les pilotes logiciels sont facilement reprogrammables, ce qui nous permet de mettre en œuvre assez facilement des optimisations logicielles. Les travaux ont tout d'abord été menés dans le pilote GM, dont l'interface de programmation présente des spécificités telles que celles que nous avons présentées en partie 2.2 et qui est assez représentative des interfaces actuelles des réseaux rapides. Nous proposons ensuite des solutions et les appliquons dans le nouveau pilote MX des réseaux MYRINET qui était en cours de développement au moment de notre étude.

3.2 Transferts de données

Nous avons expliqué en partie 2.2.2 que les communications zéro-copie OS-bypass permettaient d'obtenir des performances élevées pour les communications dans les applications parallèles. Mais les zones mémoire manipulées dans ces communications doivent être *préparées*, par un mécanisme très coûteux nommé enregistrement mémoire. Il est donc nécessaire d'utiliser intelligemment cette stratégie.

3.2.1 L'enregistrement mémoire

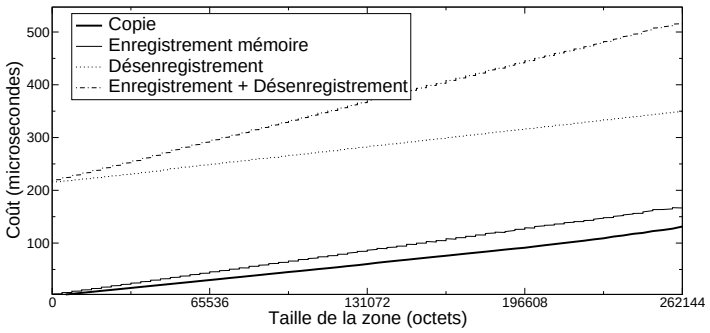


Figure 6: Comparaison des coûts de l'enregistrement mémoire dans MYRINET-GM et d'une copie mémoire. Les mesures ont été réalisées sur une machine équipée de deux processeurs PENTIUM 4 XEON cadencés à 2,6 GHz.

La figure 6 présente les coûts comparés de l'enregistrement mémoire et des copies mémoire. Ces mesures montrent qu'il peut être avantageux de remplacer l'enregistrement mémoire par une copie dans des zones statiquement pré-enregistrées. Cependant cette stratégie est consommatrice de cycles processeur et donc peu satisfaisante dans le contexte des applications parallèles. Une optimisation plus intéressante consiste à retarder le désenregistrement le plus longtemps possible afin d'éviter son coût. Il s'agit donc de maintenir un cache d'enregistrement mémoire (*Pin-down Cache* [TOHI98]).

Cette stratégie est connue pour être relativement facile à mettre en œuvre en espace utilisateur dans les applications parallèles. Elle s'applique en fait de la même façon aux systèmes de fichiers distribués mis en œuvre en espace utilisateur puisque les contraintes y sont très similaires [GPG04].

Nous étudions maintenant le cas des systèmes mis en œuvre dans le noyau, en particulier LUSTRE. Nous nous concentrons sur le côté client des implantations car c'est celui qui

impose le plus de contraintes. En effet, notre choix de nous conformer à l'interface standard d'accès aux fichiers limite notre marge de manœuvre puisque nous ne pouvons pas remonter jusqu'à l'application les contraintes liées à l'interface de programmation du réseau rapide sous-jacent. Nous détaillons les deux types d'accès que l'application peut demander : accès par le cache et accès directs.

3.2.2 Accès par le cache du système d'exploitation

Les accès traditionnels aux fichiers passent par le cache du système d'exploitation. C'est-à-dire que l'application accède aux données du cache, tandis que le système maintient le cache à jour en contactant le serveur distant en arrière plan (voir la figure 7). Les communications mises en jeu concernent donc des zones mémoire du système d'exploitation au lieu des zones de l'espace utilisateur de l'application.

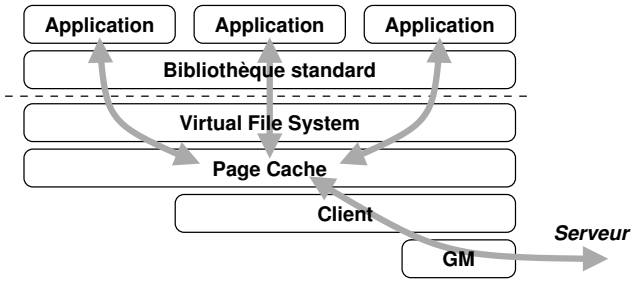


Figure 7: Accès aux fichiers distants par un client mis en place dans le système d'exploitation et à travers le cache.

Des travaux ont été menés afin de définir une sémantique claire de partage et de copie des pages mémoire dans les caches du type cache de fichiers ou les couches protocolaires d'accès au réseau [PDZ00]. Ils ont abouti à des propositions de fusion du cache de fichiers du système d'exploitation et des zones mémoire intermédiaires utilisées par le réseau. Cela peut permettre de transmettre directement des données entre le réseau et le cache de fichiers [WWPR04]. Mais ces modèles restent basés sur l'enregistrement mémoire, ce qui nous semble beaucoup trop complexe compte tenu du contexte d'exécution dans le noyau et de la spécificité des pages mises en jeu. Le cache de fichiers du système d'exploitation est constitué de zones mémoire dont les caractéristiques sont très différentes des zones de l'espace utilisateur. Nous avons montré que la stratégie de l'enregistrement mémoire n'y est pas du tout adaptée [GPG04].

Plutôt que d'adapter l'enregistrement mémoire à ces pages spéciales, nous avons proposé de modifier l'interface de programmation du réseau pour utiliser les adresses physiques associées aux zones mémoire du cache. Cette stratégie à l'avantage d'être très simple à utiliser puisqu'aucune préparation des zones mémoire n'est nécessaire avant les communications : les pages sont déjà verrouillées et leur adresse physique est connue. Nous l'avons mise en place dans l'interface GM des réseaux MYRINET, ce qui a imposé la modification du microprogramme tournant dans la carte d'interface.

3.2.3 Accès directs depuis le système d'exploitation

Les accès par le cache du système d'exploitation présentent l'inconvénient d'imposer une copie entre l'application et le cache, ce qui consomme des cycles processeur. Par ailleurs, les applications parallèles souhaitent souvent maîtriser leurs écritures, c'est-à-dire savoir qu'une écriture sera effectivement envoyée sur le serveur distant et non pas conservée dans le cache du client pendant un délai indéterminé. Les systèmes d'exploitation modernes proposent donc des accès directs qui évitent le cache du système d'exploitation.

Nous avons également étudié ce type d'accès. Contrairement aux accès par le cache où l'enregistrement mémoire n'est pas adapté, le cas des accès directs reste similaire au cas des communications dans les applications parallèles. En effet, les zones mémoire mises en jeu sont situées en espace utilisateur. Mais les communications sont initiées depuis le système d'exploitation. Il faut donc que l'interface de programmation du réseau soit capable de manipuler simultanément des adresses virtuelles utilisateur de différents processus depuis le noyau. Il s'agit d'une contrainte forte puisque les pilotes des réseaux rapides ont l'habitude d'associer chaque canal de communication à un seul processus. Nous avons de nouveau modifié le microprogramme de la carte d'interface MYRINET pour ce faire en utilisant des pointeurs d'adresse de taille supérieure afin de stocker un identifiant de l'espace d'adressage en plus de l'adresse virtuelle elle-même [GPG04].

Ensuite, il faut là aussi utiliser intelligemment l'enregistrement mémoire afin de réduire son coût apparent. La stratégie du cache d'enregistrement (voir en partie 3.2.1) est tout à fait adaptée. Cependant, elle impose de maintenir à jour le cache d'enregistrement vis-à-vis des éventuelles modifications de l'espace d'adressage de l'application. En effet, l'application n'étant pas consciente que certaines de ses pages ont été enregistrées à la volée par les couches logicielles sous-jacentes, elle peut modifier son adressage sans prévenir le gestionnaire du cache d'enregistrement. Si le cache n'est pas invalidé dans ce cas, la carte d'interface réseau risque d'utiliser des traductions d'adresses virtuelles invalides.

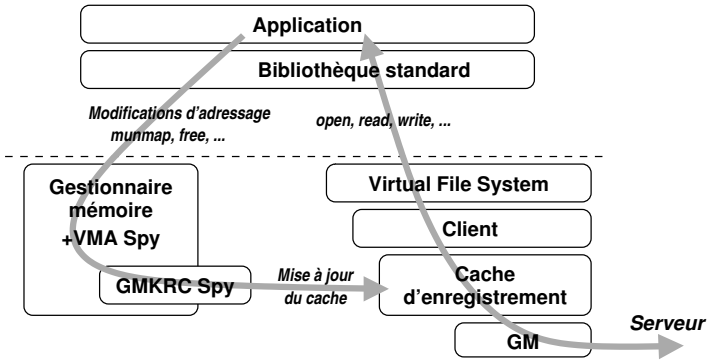


Figure 8: Gestion des accès directs depuis le système d'exploitation avec un cache d'enregistrement mémoire maintenu à jour par l'infrastructure VMA SPY.

Maintenir ce cache à jour dans une implantation en espace utilisateur est une technique bien connue. Par contre, nous avons montré que c'était impossible dans une implantation dans le système d'exploitation en raison d'une absence de support logiciel. Et nous avons ensuite proposé des modifications au noyau LINUX pour résoudre ce problème et ainsi pouvoir mettre en place un cache d'enregistrement mémoire dans le noyau. Ces modifications,

nommées VMA SPY, nous ont permis de mettre en place un cache d'enregistrement mémoire entre GM et le client d'accès aux fichiers distants [GPG04] (voir la figure 8).

3.2.4 Adressage mémoire souple

Nous nous sommes concentrés sur le côté client et avons insisté sur les problèmes d'enregistrement mémoire puisque c'est la stratégie généralement utilisée pour mettre en place des communications zéro-copie. Les différentes modifications que nous avons dû apporter au pilote GM pour mettre en place les différents types d'accès aux fichiers distants montrent comment les multiples besoins du stockage distribué ne sont pas satisfaits par les interfaces réseaux existantes. Il est en effet difficile de faire interagir les couches système d'accès aux fichiers et les interfaces de programmation des réseaux des grappes qui ont été conçues et optimisées pour les communications en espace utilisateur dans les applications parallèles. Nous présenterons une évaluation de cette implantation en partie 4.

En dehors des modifications du système d'exploitation pour maintenir le cache d'enregistrement à jour, nous avons identifié les besoins suivants :

- Manipuler de la mémoire utilisateur lors de communications initiées depuis un contexte noyau ;
- Partager un même canal de communication entre les espaces d'adressage de différents processus accédant aux fichiers distants ;
- Pouvoir éviter l'enregistrement de certaines zones mémoires (par exemple celles du cache de fichiers).

Nous avons donc proposé dans [GGP05] une interface de programmation satisfaisant tous ces besoins. Elle se présente sous la forme de primitives de communications permettant à l'utilisateur de préciser le type d'adressage mémoire utilisé pour les zones mises en jeu. Ainsi il est possible de manipuler de manière uniforme les données situées dans l'espace utilisateur des processus ou dans le cache du système d'exploitation.

Nous avons mis en œuvre cette interface de programmation dans le nouveau pilote MX des réseaux MYRINET. Ce travail ayant été effectué pendant le développement de MX, nous l'y avons intégré et fait en sorte que son implantation interne ne soit pas trop orientée vers les communications en espace utilisateur. Nous présenterons une évaluation de ses performances en partie 4.

3.3 Contrôle de flot

Nous venons de présenter notre étude des problèmes liés au transfert des données sur les réseaux rapides dans les systèmes de stockage distribué. Nous nous intéressons maintenant aux problèmes liés au contrôle, en commençant par le contrôle de flot.

Les réseaux et protocoles traditionnels ont été conçus pour permettre à n'importe quelle machine de la planète de communiquer, quelle que soit la topologie du réseau. Cela permet une très grande flexibilité mais implique un très grand coût puisqu'il faut être capable d'une part, de faire inter-opérer un très grand nombre de machines, et d'autre part, de pouvoir réagir aux phénomènes de congestion dans les goulets d'étranglement. Dans les grappes de calculateurs, la topologie est régulière et statique, et le cœur de réseau y est suffisamment dimensionné pour qu'un trafic régulier entre toutes les extrémités du réseau ne subisse aucune

congestion. Le protocole de transport dans les couches bas niveau de gestion du réseau se base sur l'hypothèse d'un canal fiable, tandis que la reprise sur erreur est un cas particulier peu optimisé.

Les applications parallèles ont généralement un trafic assez uniforme à travers la grappe. En effet, une bonne parallélisation d'un problème nécessite d'une part de répartir des charges de calcul similaires sur les machines, mais aussi de faire en sorte que le trafic réseau soit bien réparti. Ainsi, les goulets d'étranglement sont évités, que ce soit en terme de puissance de calcul ou en terme de capacité des liens.

L'utilisation de ces réseaux dans un modèle de stockage distribué ne génère pas le même trafic. En effet, les modèles client-serveur ont intrinsèquement un trafic centralisé autour du serveur, ce qui crée un goulet d'étranglement. L'utilisation de système de fichiers parallèles a été initialement proposée pour répartir la charge de traitement sur différents serveurs. Cette idée a également l'avantage de diffuser la charge réseau sur les liens des différents serveurs. Le problème consiste à dimensionner correctement le système puisqu'un trop grand nombre de clients forme là encore un goulet au niveau des serveurs.

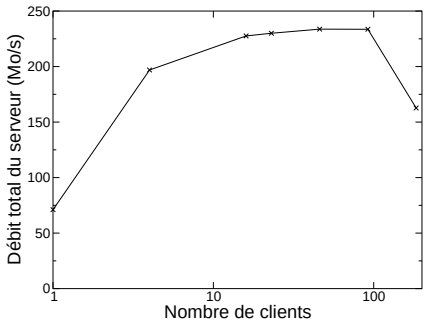


Figure 9: Impact de l'augmentation du nombre de clients sur la bande passante utile d'un serveur de fichiers. Les clients soumettent simultanément des requêtes d'écriture de 64 ko.

La figure 9 présente la bande passante traitée par un prototype de serveur de fichiers lorsque les clients lui envoient simultanément, par l'interface GM des réseaux MYRINET, des requêtes d'écriture de 64 ko. Avec 92 clients, 233 des 250 Mo/s disponibles sur le réseau MYRINET sont utilisés. Par contre, avec 184 clients, seuls 163 Mo/s sont apparemment traités.

Le phénomène qui se produit ici est un gaspillage de la bande passante lié à la réémission des messages non-reçus. Dans le modèle du *Rendez-vous*, celui proposé par l'interface GM et souvent utilisé dans les réseaux rapides, le serveur doit préparer à l'avance des zones de réception, leur nombre étant limité par la mémoire disponible dans la carte d'interface (125 dans notre cas). Si le nombre de clients est trop grand, certaines requêtes ne pourront pas être reçues, leur transfert aura donc utilisé une partie de la bande passante réseau avant que les données soient jetées par le récepteur. Le débit utile du serveur diminue alors avec le nombre de clients. C'est ce que nous constatons sur la Figure 9.

Ce problème des messages inattendus (*Unexpected messages*) est assez courant dans le domaine des applications parallèles. Il a par exemple été montré que la puissance des cartes d'interface réseau existantes ne suffisait pas toujours à traiter efficacement les messages inattendus dans MPI [BU04]. L'idée serait de ne pas laisser l'émetteur gaspiller inutilement la

bande passante utile en lui faisant attendre l'accord du récepteur avant d'envoyer. Ainsi, cela supprimerait l'arrivée de messages inattendus sur le récepteur. Pour les petits messages, il est également envisageable de les déposer dans un espace temporaire sur le récepteur en attendant que l'application fournisse enfin la zone de réception correspondante. Ce type de protocole est désormais souvent mis en place dans MPI mais les interfaces natives de programmation du réseau ne le proposent généralement pas.

La mise en place d'un tel protocole dans GM demandant d'importantes modifications, nous nous sommes donc intéressés à une autre solution. À défaut de support satisfaisant dans l'interface native du réseau, nous avons proposé dans [Gog05] d'adapter l'application pour qu'elle traite elle-même le problème des messages inattendus. Il s'agit donc de mettre en place au-dessus d'une interface limitée un protocole de gestion efficace des messages inattendus.

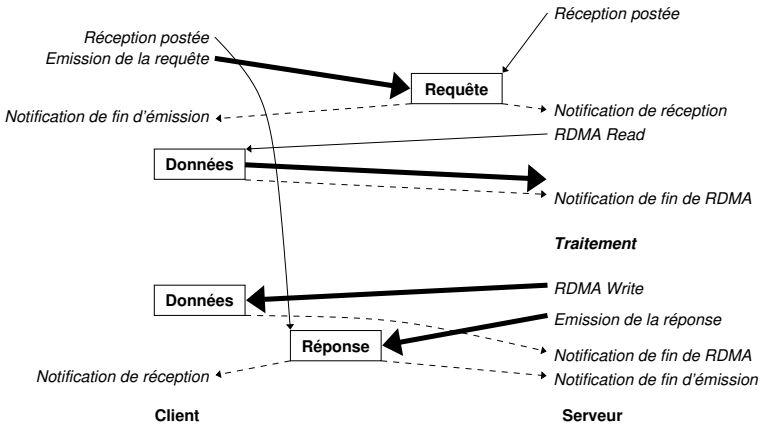


Figure 10: Protocole client-serveur d'accès aux fichiers distants avec contrôle du côté serveur. Le client émet une requête puis le serveur se charge de lire ou écrire les données dans la mémoire du client.

Nous avons proposé l'utilisation combinée du modèle du *Rendez-vous* et des *Accès mémoire à distance* (RDMA). Le déport de l'initiative des transferts de données sur le serveur lui permet de maîtriser le flux entrant et donc d'éviter les goulets d'étranglement. Ainsi, il se charge d'aller lire ou écrire les données dans la mémoire du client quand il a les ressources pour le faire. La figure 10 décrit le diagramme temporel du protocole client-serveur d'accès aux fichiers distants dans ce cas. Ce modèle est cependant limité aux interfaces de programmation proposant à la fois des communications de type RDMA et *Rendez-vous*, ce qui n'est par exemple pas le cas des VERBS sur INFINIBAND.

Des travaux intéressants ont par ailleurs été menés dans DAFS pour permettre l'utilisation de RDMA initiés par le client [Mag02b]. Le principe est que le client tente de lire ou d'écrire directement dans le cache du serveur en espérant que les zones cibles soient toujours exportées. Si c'est le cas, le client n'a plus besoin de l'intervention du serveur pour accéder aux données. Sinon, une exception distante est provoquée et le client se rabat sur un protocole guidé par le serveur, similaire à notre proposition.

3.4 Interface de programmation asynchrone

Nous avons vu en partie 2.2.3 que les réseaux haute performance utilisaient un modèle de programmation très particulier. Contrairement à la traditionnelle interface SOCKET qui propose par défaut des primitives de communication bloquantes, les interfaces de programmation des réseaux rapides sont non-bloquantes et asynchrones. L'application soumet des requêtes qui sont traitées en arrière-plan par la carte d'interface pendant que l'application reprend ses calculs. Elle viendra plus tard tester la terminaison d'une requête et éventuellement bloquer en l'attendant.

L'utilisation de ce modèle de programmation rend difficile l'interaction entre le réseau et le stockage car les besoins du système de stockage sont très variables. En effet, nous avons montré dans [Gog05] qu'il était nécessaire de pouvoir attendre la terminaison de n'importe quelle communication du côté serveur tandis que le client souhaite attendre la terminaison d'une requête précise.

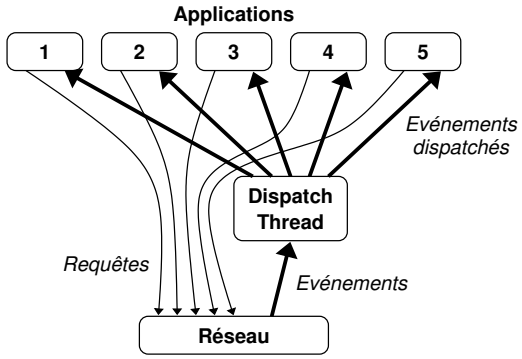


Figure 11: Répartition des événements réseau par un thread *Dispatcher*.

Ces deux stratégies d'attente d'événement sont très souvent utilisées dans MPI mais les interfaces de programmation du réseau ne les proposent pas toujours en natif. Lorsque seule la première est disponible, comme cela est le cas dans l'interface GM des réseaux MYRINET ou les VERBS de INFINIBAND, il est possible d'implanter la seconde en utilisant la première. Un thread est alors dédié à la récupération de tous les événements dans une unique file et de les donner (*Dispatch*) à la file d'exécution concernée (voir la figure 11). Mais ceci augmente la latence puisque les changements de contexte seront plus nombreux. Nous avons mesuré un surcoût de plusieurs microsecondes quand cela est mis en œuvre.

Lorsque seule la seconde stratégie est disponible, comme cela est le cas dans l'interface ELANLIB des réseaux QUADRICS ou SISCI de SCI, l'implantation de la première stratégie par la seconde nécessite des attentes actives qui gaspillent le temps processeur.

L'obtention d'une interaction efficace entre le système de stockage distribué et les réseaux rapides nécessite que ces derniers supportent les deux types de notification. Ce besoin est en fait également rencontré dans les applications parallèles programmées avec MPI.

Par ailleurs, l'exploitation des machines multiprocesseur impose de répartir la charge sur les différents processeurs. On peut donc imaginer de répartir la récupération des événements entre différents threads. Une solution consiste à répartir les événements réseau dans des files distinctes (*Completion Group*) et à confier une file à chaque thread. C'est un autre type de notification dont nous avons montré l'utilité pour le stockage distribué. Ces dif-

férentes stratégies de notification ont été mises en place dans le nouveau pilote MX des réseaux MYRINET.

3.5 Synthèse des besoins et propositions

Nous avons étudié dans cette partie les différents besoins correspondants aux coûts $T_{\text{Données}}$ et $T_{\text{Contrôle}}$ de l'utilisation des réseaux rapides pour le stockage distribué. Dans le plan données, les contraintes sur la gestion mémoire sont très importantes et nécessitent une grande flexibilité de l'interface de programmation du réseau. Nous avons montré comment contourner les limites des interfaces existantes puis proposé une interface adaptée que nous avons intégrée dans le nouveau pilote MX des réseaux MYRINET.

Dans le plan contrôle, nous nous sommes tout d'abord intéressés au problème du contrôle de flot dans le modèle client-serveur du stockage distribué et avons montré qu'il était nécessaire de gérer efficacement les messages inattendus, soit dans l'interface réseau, soit au niveau de l'application. Nous avons ensuite étudié les besoins en notification d'événements dans les clients et les serveurs de stockage distribués et avons montré qu'il était intéressant de bénéficier de différentes stratégies de notification dans l'interface de programmation du réseau.

Ces besoins en contrôle sont en fait très similaires à ceux des applications parallèles programmés avec MPI. Il est donc intéressant d'y répondre dans l'interface de programmation des réseaux rapides. C'est ce que propose le nouveau pilote MX des réseaux MYRINET. Par contre, avec les interfaces plus classiques telles que GM, il faudra recourir à des protocoles plus complexes et moins efficaces au niveau de l'application.

4 Évaluation des performances

Nous nous sommes attachés à mesurer l'impact de nos propositions sur la facilité d'utilisation et l'efficacité des réseaux rapides pour le stockage distribué. Concernant les problèmes liés au contrôle, nos propositions ont été intégrées dans la nouvelle interface de programmation MX des réseaux MYRINET. Nous avons donc comparé l'efficacité de MX pour le stockage distribué par rapport à une interface traditionnelle des réseaux rapides, GM, où les notifications d'événements sont limitées et le contrôle de flot peu adapté aux messages inattendus.

En ce qui concerne le plan données, nous avons intégré nos propositions dans MX. Ceci a tout d'abord nécessité de rendre accessible l'interface de programmation de MX dans le noyau puis d'adapter sa mise en œuvre interne pour supporter efficacement les différents types de transfert et d'adressage mémoire que nous avons décrits précédemment. Pour comparer l'efficacité des transferts de données dans ce cadre avec celle des interfaces traditionnelles, en particulier GM, il a été nécessaire de mettre en place les modifications de l'interface de programmation et du système d'exploitation que nous avons présentées en parties 3.2.3 et 3.2.2 pour pouvoir satisfaire les besoins du stockage distribué.

4.1 Plate-forme d'expérimentation

Les mesures de performance dans les systèmes de fichiers distribués sont très complexes car des fonctionnalités avancées telles que le cache ou la parallélisation sont souvent utilisées. Pour isoler les aspects qui nous intéressent, c'est-à-dire l'utilisation efficace du réseau

rapide sous-jacent, nous avons développé un prototype proposant des accès aux fichiers distants sans fonctionnalités avancées. Il s'agit donc de supprimer toutes les étapes complexes et non-indispensables afin d'obtenir le chemin le plus rapide entre l'espace mémoire de l'application et le serveur.

Notre système a été développé à la fois en espace utilisateur (ORFA, *Optimized Remote File-system Access* [GP04]) et dans le noyau (ORFS, *Optimized Remote File System* [GPG04]) afin de pouvoir étudier les différents types d'accès existants. ORFA est mis en place sous forme d'une bibliothèque partagée exposant l'interface standard d'accès aux fichiers et tranferrant les requêtes de l'application au serveur distant via le réseau MYRINET. Nous avons montré dans [GPG04] que le cache d'enregistrement mémoire était une stratégie intéressante pour les accès aux fichiers distants dans ce cas. Nous présentons ici une étude des performances dans le noyau, avec ORFS, ce qui permet donc d'envisager ensuite une application de nos travaux à PVFS ou LUSTRE. ORFS se présente comme un module noyau ajoutant un nouveau type de système de fichiers dans LINUX.

Nous avons mené nos expériences entre deux machines équipées de deux processeurs PENTIUM 4 XEON cadencés à 2,6 GHz et de 2 Go de mémoire vive. Elles sont reliées par un réseau MYRINET 2000 qui utilise des cartes PCID. Les interfaces GM 2.0.13 et MX 0.8.8 ont été utilisées.

Pour illustrer les performances de l'accès distant sur MX et GM, nous avons publié dans [GGP05] une comparaison des performances brutes de ces deux interfaces de programmation. La débit offert par MX est similaire à celui de GM, tandis que sa latence est bien inférieure (4 contre 8 μs). Nous avons également constaté que notre mise en œuvre des communications noyau dans MX a été intégrée suffisamment finement pour être aussi performante qu'en espace utilisateur. C'est un point intéressant pour notre contexte d'étude car la conception traditionnelle des interfaces de programmation des réseaux rapides, notamment GM, SISCI ou ELANLIB, est souvent trop orientée vers les communications en espace utilisateur. Ainsi, l'interface de programmation noyau est souvent moins efficace voire inexistante. A titre d'exemple, la latence de GM dans le noyau est 30% supérieure à celle en espace utilisateur car GM a été conçu pour ce dernier contexte.

4.2 Accès directs aux fichiers distants

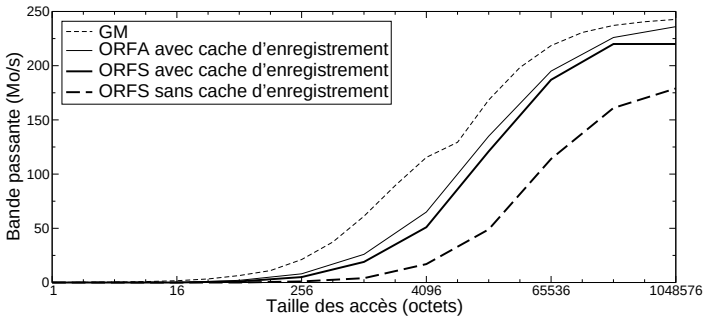
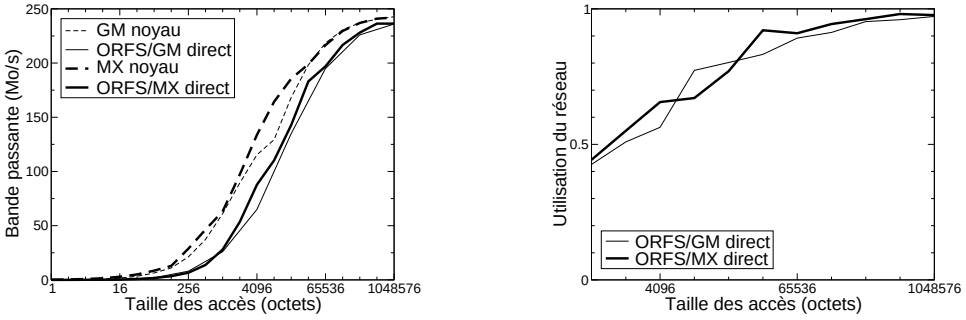


Figure 12: Performance comparée des accès directs aux fichiers distants en espace utilisateur (avec ORFA) et dans le noyau (ORFS) sur GM.

La figure 12 présente une étude des transferts directs depuis le noyau avec ORFS et GM. La figure confirme tout d'abord l'intérêt du cache d'enregistrement dans le noyau. On constate que les modifications que nous avons apportées à GM et notre infrastructure VMA SPY (voir en partie 3.2.3) permettent d'obtenir de bonnes performances lorsqu'un cache d'enregistrement est mis en place pour des communications depuis le noyau. Ces résultats montrent que la quantité de travail qui a été nécessaire pour supporter les accès directs depuis le noyau sur GM permet heureusement d'atteindre des performances proches de celles en espace utilisateur. Ces performances restent toute fois légèrement inférieures puisque le client doit ici traverser les couches système entre l'application et ORFS ($T_{\text{Stockage1}}$ plus élevé).



(a) Bande passante.

(b) Bande passante relative.

Figure 13: Performance comparée des accès directs aux fichiers distants depuis le noyau avec ORFS sur MX et GM.

La figure 13 compare ensuite les accès directs depuis le noyau avec ORFS sur GM et MX. La légère supériorité des performances brutes de MX sur celles de GM se reflète dans les performances de ORFS. Mais ORFS utilise également légèrement mieux MX que GM puisque sa bande passante relative est la plupart du temps supérieure. De plus, il faut noter que les performances sur GM sont tributaires de la réutilisation des mêmes zones mémoire dans l'application afin de maximiser l'efficacité du cache d'enregistrement, alors que celles de MX n'en dépendent pas puisqu'aucun enregistrement mémoire explicite n'est demandé à l'application utilisant MX.

Mais il faut surtout insister ici sur la facilité d'utilisation de l'interface noyau de MX. En effet, MX est directement utilisable par ORFS alors que l'utilisation de GM depuis le noyau a nécessité de mettre en place un cache d'enregistrement, de modifier le noyau pour mettre à jour ce cache (VMA SPY) et de modifier son interface de programmation ainsi que le microprogramme dans la carte MYRINET (voir en 3.2.3).

4.3 Accès aux fichiers distants à travers le cache

La figure 14 présente ensuite les performances des accès à travers le cache (voir en 3.2.2). La traversée du cache impose un découpage naturel des requêtes de l'application en requête de la taille d'une page (4 kilo-octets ici). Les noyaux LINUX récents permettent des accès par pages multiples mais ils rends plus difficile l'interprétation des résultats.

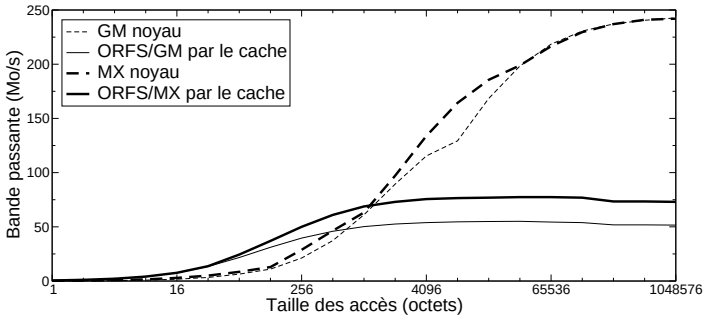


Figure 14: Performance des accès distants avec ORFS sur MX et GM à travers le cache.

Les performances des accès ORFS par le cache sur MX présentent une bande passante 40 % supérieure à celle sur GM, ce qui correspond à des temps de traitement de 54 et 76 μs respectivement. Une mesure plus précise des différents coûts liés au réseau lui-même nous permet de montrer que 13 μs de cette différence sont dues aux performances brutes de MX qui sont supérieures à celles de GM grâce à notre mise en œuvre efficace des communications dans le noyau. Le gain additionnel de 9 μs est lié à l'élimination (grâce à l'interface de MX) de la gestion complexe du contrôle des communications que ORFS doit faire avec GM, en particulier les stratégies de notification d'événements qui nécessitent un thread *Dispatcher* (voir en partie 3.4).

L'influence sur les performances des fonctionnalités proposées par MX pour gérer les messages inattendus et les notifications d'événements reste difficile à chiffrer car elles influent essentiellement sur la latence vue par l'application. Or, l'effet de la latence est difficile à observer dans les systèmes de fichiers distribués car les requêtes de petites tailles sont contrôlées par le cache des attributs des fichiers. Nous avons cependant montré dans [GGP05] que nos travaux permettent d'abaisser notablement la latence d'une application ayant les mêmes contraintes de communication que ORFS, un protocole de SOCKET zéro-copie.

4.4 Accès au niveau bloc

Les besoins de l'accès distant au niveau des blocs (*Network Block Device*, voir en partie 2.3.4) sont très proches de ceux des accès à travers le cache que nous venons de présenter. Nous nous attendions donc à ce que MX soit également particulièrement adapté à cette application. Nous avons donc mis en œuvre un *Network Block Device* utilisant notre interface de programmation de MX dans le noyau.

La figure 15 présente une comparaison des performances des accès aux fichiers distants à travers le cache, en effectuant les transferts réseau au niveau bloc (MXBD) ou au niveau système de fichiers (ORFS). La différence entre ces deux stratégies est que la seconde transfère des attributs ou des pages de contenu de fichiers tandis que la première ne transfère que des blocs dont le contenu n'est pas structuré. Les contraintes sur l'interface de programmation du réseau sont assez similaires. Mais l'avantage du niveau bloc est qu'il a été conçu pour les contrôleurs de disques et utilise donc des requêtes asynchrones. Il tire donc beaucoup mieux parti des communications asynchrones proposées par le réseau rapide, cela se traduisant notamment par des communications pipelinées. Ceci explique pourquoi les ac-

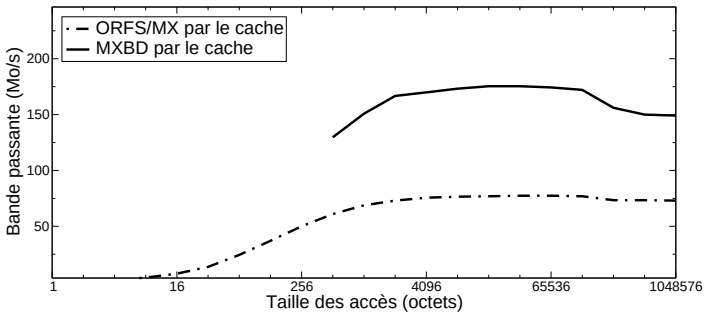


Figure 15: Performances des accès à travers le cache au niveau bloc (MXBD) et au niveau système de fichiers (ORFS) sur MX.

cès au niveau bloc atteignent un débit double. Si de plus on combine ces stratégies avec des communications vectorielles permettant de gérer des segments mémoire discontigus, il devient possible d'utiliser le maximum de la capacité du lien d'accès.

5 Conclusions

Cet article analyse l'interaction entre les couches système d'accès au stockage et les réseaux rapides des grappes de calculateurs. Ces réseaux et leurs interfaces de programmation ont été conçus et optimisés pour les communications en espace utilisateur dans les applications parallèles. Si les performances obtenues pour ces communications sont très élevées, celles des accès au stockage distribué utilisés dans ces calculateurs ne disposent pas des mêmes performances.

Les travaux sur le stockage distribué ont principalement ciblé les stratégies du type cache et parallélisation. Nous proposons dans cet article une approche complémentaire visant à améliorer les performances des systèmes de stockage en profitant du réseau rapide disponible dans les grappes. La spécificité de ces réseaux a cependant rendu difficile leur utilisation dans un autre contexte que celui des applications parallèles.

Nous avons présenté une analyse détaillée des différents modéles d'accès aux fichier distants puis identifié les limites des interfaces de programmation des réseaux rapides dans ces modèles. Nous avons mis en œuvre différentes modifications dans l'interface de programmation GM des réseaux MYRINET ainsi que dans le noyau LINUX pour obtenir une interaction efficace dans le plan données. Nous avons par ailleurs montré que les besoins inhérents au contrôle des communications pour le stockage distribué étaient assez importants mais finalement assez proches de ceux présents dans les implantations de MPI.

Nous avons ensuite proposé différentes idées afin de faciliter l'utilisation des futures interfaces des réseaux rapides pour le stockage distribué. D'une part, les besoins communs aux applications MPI et au stockage distribué devraient être résolus dans l'interface réseau en proposant **différentes stratégies de notification d'événements** et un **contrôle efficace des messages inattendus**. D'autre part, les interfaces de programmation dans le noyau doivent être suffisamment souples pour pouvoir manipuler les **différents types d'adressage mémoire**. Nous avons mis en place ces idées dans le nouveau pilote MX des réseaux MYRINET. Nos travaux sont désormais intégrés à la distribution officielle de MX.

	Limites de GM	Apports de MX
Latence noyau brute	8 μs (6 en espace utilisateur)	4 μs (4 en espace utilisateur)
Accès aux fichiers distants depuis le noyau (par le cache)	Utilisation adresse physique Recompilation du microprogramme Thread *Dispatcher*	40 % de gain par rapport à GM
Accès aux fichiers distants depuis le noyau (direct)	Enregistrement mémoire complexe Modification du noyau Recompilation du microprogramme Thread *Dispatcher*	Au moins aussi bon que GM Facilité d'utilisation

Table 2: Résumé des limites de GM et des apports de MX dans le cadre des applications de stockage distribué depuis le noyau.

Nous avons enfin présenté des évaluations de performances avec un protocole optimisé d'accès aux données distantes afin de mettre en évidence l'efficacité de l'interaction entre réseau et stockage. La table 2 résume ces résultats. Nos propositions intégrées à MX se révèlent plus efficaces qu'avec les interfaces traditionnelles. Par ailleurs, leur utilisation est beaucoup plus simple. La quantité de modifications nécessaires pour utiliser une interface traditionnelle dans ce contexte montre en effet leur inadaptation aux contextes autres que celui des communications en espace utilisateur. Nos travaux dans MX s'appliquent également très bien à l'accès au stockage distant au niveau bloc, ainsi qu'aux protocoles SOCKET zéro-copie.

Les modifications du noyau que nous avons proposées pour maintenir le cache d'enregistrement mémoire à jour avec GM dans le noyau ne sont plus nécessaires avec MX. Cependant, elles pourraient permettre d'en améliorer les performances une fois de plus. Aucun support pour les réseaux rapides n'est disponible dans le noyau LINUX à ce jour. Nous avons mené des discussions à ce propos avec les développeurs du noyau et ceux des réseaux QUADRICS et INFINIBAND. Aucun consensus n'a pu être trouvé jusqu'à présent mais il est certain qu'un support du type VMA SPY (voir en partie 3.2.3) devra être inclu dans LINUX à terme. En effet, la généralisation des interfaces ETHERNET 10 gigabit soulève des problèmes similaires à ceux que nous avons rencontrés dans les réseaux rapides, problème que nous souhaitons étudier dans nos travaux futurs.

References

[ADN+96] Thomas E. Anderson, Michael D. Dahlin, Jeanna M. Neefe, David A. Patterson, Drew S. Roselli, and Randolph Y. Wang. Serverless network file systems. *ACM Trans. Comput. Syst.*, 14(1):41–79, 1996.

[BCF+95] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, 1995.

[BdRC93] R. Bordawekar, J. M. del Rosario, and A. Choudhary. Design and Evaluation of primitives for Parallel I/O. In *Supercomputing '93: Proceedings of the 1993*

ACM/IEEE conference on Supercomputing, pages 452–461, New York, NY, USA, 1993. ACM Press.

[BGM99] Amnon Barak, Ilia Gilderman, and Igor Metrik. Performance of the Communication Layers of TCP/IP with the Myrinet Gigabit LAN, 1999.

[BM00] R. Brightwell and A. Maccabe. Scalability limitations of VIA-based technologies in supporting MPI. In *Proceedings of the Fourth MPI Developer's and User's Conference*, March 2000.

[BU04] Ron Brightwell and Keith D. Underwood. An Analysis of NIC Resource Usage for Offloading MPI. In *Proceedings of the 2004 Workshop on Communication Architecture for Clusters*, Santa Fe, New Mexico, April 2004.

[CRU03] Olivier Cozette, Cyril Randriamaro, and Gil Utard. READ^2: Put disks at network level. In *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, Tokyo, Japan, May 2003. IEEE Computer Society.

[For94] Message Passing Interface Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230, 1994.

[Geo01] P. Geoffray. OPIOM: Off-Processor I/O with Myrinet. *PPL - Parallel Processing Letters*, 11(2-3):237–250, 2001.

[GGP05] Brice Goglin, Olivier Glück, and Pascale Vicat-Blanc Primet. An Efficient Network API for in-Kernel Applications in Clusters. In *Proceedings of the IEEE International Conference on Cluster Computing*, Boston, Massachussets, September 2005. IEEE Computer Society Press.

[Gog05] Brice Goglin. *Réseaux rapides et stockage distribué dans les grappes de calculateurs : propositions pour une interaction efficace*. PhD thesis, École normale supérieure de Lyon, 46, allée d'Italie, 69364 Lyon cedex 07, France, October 2005. 194 pages.

[GP04] Brice Goglin and Loïc Prylli. Performance Analysis of Remote File System Access over a High-Speed Local Network. In *Proceedings of the Workshop on Communication Architecture for Clusters (CAC'04), held in conjunction with the 18th IEEE IPDPS Conference*, page 185, Santa Fe, New Mexico, April 2004. IEEE Computer Society Press.

[GPG04] Brice Goglin, Loïc Prylli, and Olivier Glück. Optimizations of Client's side communications in a Distributed File System within a Myrinet Cluster. In *Proceedings of the IEEE Workshop on High-Speed Local Networks (HSLN), held in conjunction with the 29th IEEE LCN Conference*, pages 726–733, Tampa, Florida, November 2004. IEEE Computer Society Press.

[Lig01] Walt Ligon. Next Generaton Parallel Virtual File System. In *Proceedings of the 2001 IEEE International Conference on Cluster Computing*, Newport Beach, CA, October 2001.

[LR99] W. Ligon and R. Ross. An Overview of the Parallel Virtual File System. In *Proceedings of the 1999 Extreme Linux Workshop*, June 1999.

[Mag02a] K. Magoutis. Design and Implementation of a Direct Access File system (DAFS) Kernel Server for FreeBSD. In *Proceedings of USENIX BSDCon 2002 Conference*, San Francisco, CA, February 2002.

[Mag02b] K. Magoutis. The Optimistic Direct Access File System: Design and Network Interface Support. In *Proceedings of Workshop Novel Uses of System Area Network 2002*, Cambridge, MA, February 2002.

[NWO88] M. N. Nelson, B. B. Welch, and J. K. Ousterhout. Caching in the Sprite Network File System. *ACM Transactions on Computer Systems*, 6(1), 1988.

[PDZ00] Vivek S. Pai, Peter Druschel, and Willy Zwaenepoel. IO-Lite: a unified I/O buffering and caching system. *ACM Transactions on Computer Systems*, 18(1):37–66, 2000.

[PFHC03] Fabrizio Petrini, Eitan Frachtenberg, Adolfy Hoisie, and Salvador Coll. Performance Evaluation of the Quadrics Interconnection Network. *Journal of Cluster Computing*, 6(2):125–142, April 2003.

[PT97] Loï Prylli and Bernard Tourancheau. Protocol Design for High Performance Networking: a Myrinet Experience. Technical Report 97-22, LIP-ENS Lyon, 69364 Lyon, France, 1997.

[PVF03] PVFS2 Development Team. *Parallel Virtual File System, Version 2*, September 2003. `http://www.pvfs.org/pvfs2/pvfs2-guide.html`.

[RI04] Murali Rangarajan and Liviu Iftode. Building a User-level Direct Access File System over Infiniband. In *3rd Workshop on Novel Uses of System Area Networks (SAN-3)*, February 2004.

[Sch99] Peter Braam School. File Systems for Clusters from a Protocol Perspective. In *Second Extreme Linux Topics Workshop*, June 1999.

[Sch03] Philip Schwan. Lustre: Building a File System for 1,000 Node Clusters. In *Proceedings of 2003 Linux Symposium*, pages 401–408, Ottawa, Canada, July 2003.

[SH02] Frank Schmuck and Roger Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In *Proceedings of the Conference on File and Storage Technologies (FAST'02)*, pages 231–244, Monterey, CA, January 2002. USENIX, Berkeley, CA.

[SRO96] Steven R. Soltis, Thomas M. Ruwart, and Matthew T. O'Keefe. The Global File System. In *Proceedings of the Fifth NASA Goddard Conference on Mass Storage Systems*, pages 319–342, College Park, MD, 1996. IEEE Computer Society Press.

[TOHI98] H. Tezuka, F. O'Carroll, A. Hori, and Y. Ishikawa. Pin-down cache: A Virtual Memory Management Technique for Zero-copy Communication. In *12th International Parallel Processing Symposium*, pages 308–315, April 1998.

[WBvE97] Matt Welsh, Anindya Basu, and Thorsten von Eicken. Incorporating Memory Management into User-Level Network Interfaces. In *Proceedings of Hot Interconnects V*, Stanford, August 1997.

[WP02] Jiesheng Wu and Dhabaleswar K. Panda. MPI-IO over DAFS over VIA: Implementation and Performance Evaluation. In *Workshop on Communication Architecture for Clusters (in conjunction with IPDPS)*, April 2002.

[WWPR04] Jiesheng Wu, Pete Wyckoff, Dhabaleswar Panda, and Rob Ross. Unifier: Unifying Cache Management and Communication Buffer Management for PVFS over InfiniBand. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid2004)*, April 2004.