Accès optimisés aux fichiers distants dans les grappes disposant d'un réseau rapide.

Brice Goglin, Olivier Glück, Pascale Vicat-Blanc Primet

Laboratoire de l'Informatique du Parallélisme

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

Accès optimisés aux fichiers distants dans les grappes disposant d'un réseau rapide

Brice Goglin,
Olivier Glück,
Pascale Vicat-Blanc Primet

Décembre 2004

Research Report N° 2004-56

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`

Accès optimisés aux fichiers distants dans les grappes disposant d'un réseau rapide

Brice Goglin, Olivier Glück, Pascale Vicat-Blanc Primet

Décembre 2004

Abstract

Parallel applications running on computation clusters with high-speed interconnects require both efficient communications between processing nodes and fast access to the storage system. However, the software interface of these networks has been largely optimized for inter-nodes communication and thus does not fit the special requirements of distributed file systems.
In this article we propose and study several solutions to make remote access to distant files performant on high-speed networks in clusters. Our ideas are implemented in the GM interface of MYRINET networks and then to their new interface, MX. You present the first performance comparisons between MX and GM and their efficiency in a distributed file system context. Our first results show that it is possible to efficiently interface distributed file systems with these networks.

Keywords: Distributed File Systems, High-Speed Networks, Clusters, Memory Registration, Linux

Résumé

L'exécution performante d'applications parallèles sur grappes de calcul interconnectées par des réseaux hautes performances nécessite des communications efficaces entre les processus de calcul mais aussi des accès rapides au système de stockage. Cependant, comme l'interface logicielle des réseaux rapides a été optimisée plus particulièrement pour les communications interprocessus, elle ne répond pas bien aux besoins particuliers des systèmes de fichiers distribués.
Dans cet article, nous proposons et étudions plusieurs solutions pour améliorer l'accès distants aux fichiers via les réseaux hautes performances de grappe. Ces idées ont été intégrées à l'interface GM des réseaux MYRINET puis à la nouvelle interface MX. Nous réalisons les premières comparaisons entre les performances relatives de MX et GM puis de leur utilisation dans un système de fichiers distribué. Nos premiers résultats montrent qu'il est possible d'interfacer efficacement les systèmes de fichiers distribués avec le réseau rapide des grappes.

Mots-clés: Systèmes de fichiers distribués, réseaux hautes performances, grappes, enregistrement mémoire, Linux

1 Introduction

Depuis l'avènement des applications parallèles, les super-calculateurs ont été remplacés par des grappes de stations plus génériques, extensibles et moins onéreuses, basées sur des réseaux spécifiques à très haut débit et très faible latence tels que MYRINET [1], QUADRICS [11] ou INFINIBAND [12]. La nécessité de fournir aux applications parallèles le maximum de puissance de calcul s'est traduite par l'utilisation de cartes réseaux intelligentes et l'intégration d'optimisations spécifiques dans les interfaces de programmation. Ces optimisations permettent d'une part de recouvrir les phases de communication par des phases de calcul et d'autre part de fortement diminuer le coût des communications.

Pour recouvrir les communications entre les différents nœuds par le calcul, une grande partie du traitement des communications est déportée dans la carte d'interface réseau. Au niveau logiciel, cela se traduit par l'utilisation de primitives asynchrones de communications proposées dans les nouvelles interfaces de programmation spécifiques telles que MPI (*Message Passing Interface* [4]). La réduction du coût des communications dans l'hôte passe par l'absence de copies intermédiaires (zéro-copie), la suppression du système d'exploitation du chemin critique (*OS-bypass*) et le traitement par la carte d'interface des transferts de données entre le réseau et les applications par DMA (*Direct Memory Access*).

En contre-partie, ce modèle de programmation soulève le problème de la traduction des adresses virtuelles manipulées par les applications en adresses physiques manipulées par le matériel . Le gestionnaire de mémoire virtuelle habituellement en charge de cette traduction est situé au cœur du système d'exploitation, n'est plus invoqué de manière transparente. Il faut donc soit traduire dans l'application, soit modifier le système d'exploitation, soit le faire dans la carte d'interface.

Ainsi, beaucoup de travaux ciblés sur les communications entre nœuds d'une application parallèle ont été menés dans le domaine des interfaces logicielles des réseaux rapides pour fournir aux applications des performances élevées. En revanche, peu de travaux se sont focalisés sur l'utilisation de ces réseaux hautes performances dans le cadre du stockage.

Cet article s'intéresse à l'interaction entre les interfaces de programmation très spécifiques des réseaux tels que MYRINET et les couches systèmes d'accès aux fichiers. Nos travaux montrent qu'il peut être difficile de les faire interagir efficacement sans ajouter le support nécessaire dans les couches réseaux et le système d'exploitation.

La partie 2 présente les systèmes de fichiers distribués dans les grappes et les différents problèmes rencontrés lors de leur mise en œuvre. La partie 3 détaille la façon dont nous avons adapté l'interface logicielle GM des réseaux MYRINET afin de bénéficier au mieux des performances du réseau pour accéder aux fichiers distants. Enfin nous proposons dans la partie 4 différentes idées pour améliorer les interfaces de programmation réseau actuelles et les expérimentons au travers de l'implantation du nouveau pilote des réseaux MYRINET, MX.

2 Mise en œuvre d'un système de fichiers distribué dans une grappe

2.1 État de l'art

Les applications parallèles, exécutées sur grappes de calcul, cherchent désormais souvent à optimiser autant l'accès au système de stockage qu'au réseau. Cependant, l'interface stan-

dard d'accès aux fichiers ne propose habituellement aucune primitive asynchrone, vectorielle (utilisant plusieurs zones mémoire à la fois) ou zéro-copie, permettant d'utiliser les optimisations classiques de recouvrement et d'accélération des entrée-sorties. La rigidité de l'interface standard a donc conduit les concepteurs de systèmes de fichiers distribués dans les grappes à proposer une interface spécifique, et à ré-écrire leurs applications pour en bénéficier, tout comme ils l'avaient fait pour tirer le maximum des communications.

L'interface MPI-IO a été conçue pour fournir aux applications parallèles des primitives d'accès aux fichiers sur le même modèle que les primitives de communications de MPI. Sa principale implantation est basée sur ROMIO [15], une couche portable visant à utiliser efficacement différentes machines, réseaux et systèmes de fichiers. DAFS (*Direct Access File System* [8]) a poussé cette idée à l'extrême en proposant une interface logicielle d'accès aux fichiers distants complètement calquée sur les interfaces réseaux de type VIA (*Virtual Interface Architecture* [13]). Ces interfaces fournissent un accès performant aux fichiers distants mais imposent de ré-écrire les applications pour se conformer à leur modèle de programmation très spécifique.

Plus récemment, les systèmes d'exploitation modernes ont commencé à intégrer de nouvelles primitives d'accès aux fichiers pour permettre à n'importe quelle application d'en bénéficier. Après les accès vectoriels, les accès zéro-copie ont été intégrés dans LINUX puis les entrées-sorties asynchrones dans LINUX 2.6 (par LINUX AIO [7]). On observe depuis un retour à l'utilisation de l'interface standard du système d'exploitation plutôt que l'utilisation d'interfaces spécifiques et la ré-écriture des applications. Le système LUSTRE [3] est un des systèmes de fichiers distribués les plus aboutis dans le contexte des grappes. Il propose un système parallèle capable de passer à l'échelle des grandes grappes tout en respectant l'interface habituelle d'accès aux fichiers.

Si l'interface standard d'accès aux fichiers a été améliorée, celle des réseaux hautes performances des grappes reste très spécifique. Le modèle événementiel basé sur des primitives asynchrones et la notification de terminaison ne pose, en général, pas de problème dans le cadre de l'accès aux fichiers distants. Par contre, la gestion de la mémoire virtuelle, et notamment l'enregistrement mémoire, demeure très délicate. Par exemple, LUSTRE utilise des copies mémoire pour s'interfacer avec GM [9], l'interface logicielle des réseaux MYRINET. Ces copies mémoire sont coûteuses.

Dans le paragraphe suivant nous étudions comment optimiser l'interaction entre les couches standard d'accès aux systèmes de fichiers et les interfaces logicielles bas niveau des réseaux rapides utilisés dans les grappes. L'objectif est de faire profiter aux applications qui accèdent à un système de fichiers distant des performances de ces réseaux.

2.2 Le problème de la traduction d'adresses

2.2.1 L'enregistrement mémoire

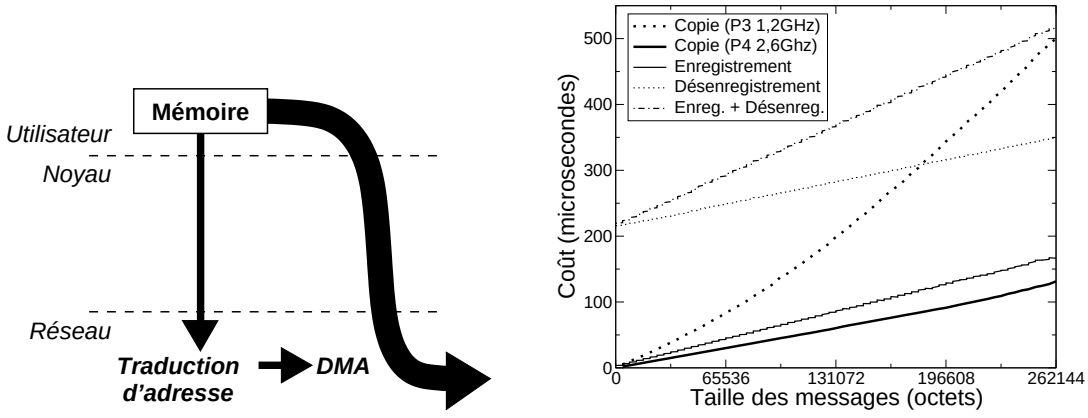
Lorsque le système d'exploitation est court-circuité (*OS-bypass*), la stratégie habituelle pour réaliser la traduction des adresses virtuelles en adresses physiques consiste à demander à l'application de préparer les zones mémoire qu'elle va utiliser pour ses communications. Cette opération, l'*enregistrement mémoire*, utilise un appel système pour verrouiller ces pages puis enregistrer leur traduction d'adresse virtuelle vers physique dans la carte d'interface. Les communications suivantes peuvent donc directement passer les adresses virtuelles à la carte d'interface qui pourra aisément retrouver les adresses physiques correspondantes.

Cette stratégie présente cependant deux inconvénients. D'une part, le coût de l'enregistrement étant grand, elle ne peut-être efficace que si l'application réutilise plusieurs fois la zone mémoire enregistrée pour ses communications. D'autre part, les applications habituelles n'étant pas prévues pour demander explicitement l'enregistrement des zones mémoire qu'elles utilisent, il faudra soit les modifier, soit utiliser une couche logicielle intermédiaire pour enregistrer à la volée.

Les bibliothèques standard de calcul parallèle telles que MPI ou VIA ont été implantées sur ces interfaces de programmation spécifiques, permettant aux applications de tirer les meilleures performances des réseaux disponibles. Cependant, cette spécificité rend difficile leur utilisation dans un cadre différent, comme celui d'un système de fichiers distribué.

2.2.2 Exemple de GM et des réseaux MYRINET

GM est l'interface de programmation des réseaux MYRINET la plus répandue. Elle est orientée passage de messages et a été conçue pour s'interfacer avec des applications de type MPI. GM nécessite d'enregistrer dans la carte d'interface les traductions en adresses physiques des zones mémoire qui vont être utilisées par l'application (voir la figure 1(a)). Le nombre de pages enregistrées dans la carte étant limité, il faut désenregistrer les pages inutiles. Comme dans toutes les interfaces de communication utilisant ce modèle, les opérations d'enregistrement et désenregistrement sont cependant très coûteuses. Dans GM, l'enregistrement coûte 3 μs par page et le désenregistrement plus de 200 μs (voir la figure 1(b)). Il est donc important d'utiliser l'enregistrement mémoire intelligemment. Il n'est intéressant que si les mêmes zones mémoire sont grandes et utilisées plusieurs fois.



(a) Transfert par DMA entre l'application et le réseau après enregistrement mémoire.

(b) Comparaison entre le coût des copies et celui de l'enregistrement mémoire dans GM.

Figure 1: Modèle et coût de l'enregistrement mémoire dans GM.

Une stratégie consiste à utiliser une copie intermédiaire dans une zone de mémoire statiquement enregistrée pour les petits messages. L'avancée principale a été proposée dans [14], un cache d'enregistrement mémoire (*Pin-Down Cache*). Pour éviter le coût du désenregistrement, on le retarde au maximum (jusqu'à ce qu'aucune page ne puisse plus être enregistrée). Si les pages sont réutilisées entre temps, il sera inutile de les réenregistrer réelle-

ment. L'inconvénient de cette méthode est qu'il faut veiller à ce que les pages restées enregistrées soient encore valides, ce qui impose de bien maîtriser les modifications de l'espace d'adressage de l'application, par exemple en interceptant les appels aux primitives `free`, `munmap`, ...

C'est notamment un problème lorsqu'on souhaite implanter une couche logicielle (par exemple MPI) entre GM et des applications non prévues pour l'enregistrement. Dans ce cas, la couche logicielle intermédiaire doit enregistrer à la volée les zones mémoire que lui passe l'application et maintenir sa table de zones enregistrées en interceptant les appels de l'application aux routines susceptibles de modifier son espace d'adressage. Dans le cas des systèmes de fichiers, il est nécessaire d'implanter un mécanisme d'enregistrement à la volée analogue.

2.3 Mise en œuvre dans les systèmes de fichiers distribués

2.3.1 Interaction avec le cache de pages

Les dispositifs physiques de stockage étant lents, les systèmes d'exploitation modernes optimisent leurs accès. Un cache de page (*page-cache*) permet d'une part, d'éviter des lectures répétitives des mêmes blocs disque et d'autre part, d'écrire les données de manière asynchrone sans bloquer l'application. Les transferts de données vus de l'application se limitent donc en fait à des copies mémoire entre elle et le cache de pages de noyau.

Dans un système de fichiers distribué, il faut mettre en place un protocole pour maintenir ces pages à jour vis-à-vis du serveur de fichiers. Utiliser une interface de programmation du type GM dans ce cadre diffère considérablement de l'utilisation qu'en font les applications parallèles en espace utilisateur. D'une part, ces interfaces logicielles n'ont pas été conçues pour réaliser des communications depuis l'espace noyau. D'autre part, les zones mémoire mises en jeu dans les communications (les pages du cache de pages) ont des caractéristiques très spéciales : ces pages sont déjà verrouillées en mémoire physique et elles ne sont généralement pas mappées en mémoire virtuelle mais leur adresse physique est facilement obtenue puisque ce traitement a lieu dans le contexte du noyau. Les hypothèses de base de l'enregistrement mémoire sont donc complètement modifiées dans ce cadre.

2.3.2 Accès direct aux fichiers distants

Le cache de pages du noyau présente l'inconvénient d'empêcher les applications de maîtriser les accès disque. Des applications très gourmandes en mémoire, telles que les bases de données ou le calcul *out-of-core*, maintiennent souvent leur propre cache et souhaitent que leurs requêtes d'écriture ne soient pas retardées par le système d'exploitation. Il serait notamment dommage que l'application soit évincée (*swappée*) parce que le système a besoin d'allouer des pages pour le cache de pages. Les systèmes UNIX modernes supportent donc des accès zéro-copie qui contournent le cache de pages (en passant le paramètre `O_DIRECT` à l'ouverture d'un fichier). Les transferts de données sont alors directs entre l'application et le support de stockage, c'est-à-dire les disques ou un serveur distant.

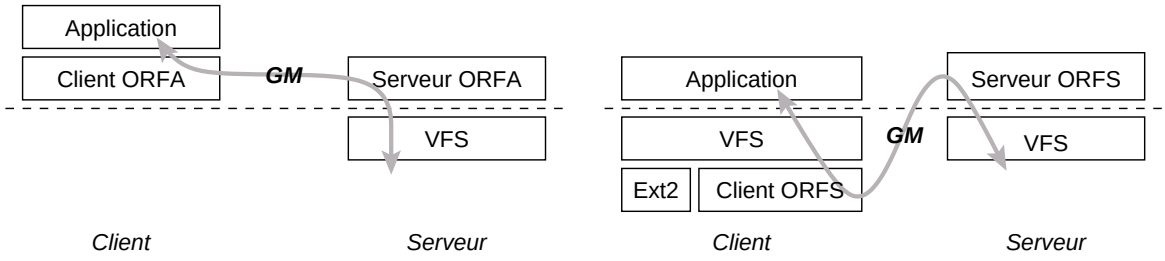
Cette stratégie est similaire aux transferts zéro-copie entre la mémoire des nœuds d'une application parallèle sur un réseau de grappe. Mais, l'implantation logicielle est très différente car le noyau LINUX ne fournit aucun support pour ce type de réseau.

3 Implantation et expérimentation avec GM

3.1 ORFA, un protocole d'accès aux fichiers distants pour réseaux rapides

Nous avons développé une plate-forme d'expérimentation nommée ORFA (*Optimized Remote File-system Access*) destinée à exhiber les limites de l'interaction entre les couches d'accès aux fichiers et les interfaces de programmation des réseaux des grappes. Nos travaux visent donc à optimiser la communication point-à-point entre un client et un serveur de système de fichiers. Nos résultats devraient ensuite pouvoir s'appliquer dans des systèmes de fichiers aboutis, tels que PVFS (*Parallel Virtual File System* [2]) ou LUSTRE, pour améliorer l'usage qu'ils font du réseau. Nous nous intéressons uniquement aux interfaces standard d'accès aux fichiers, telles qu'elles sont présentes dans les noyaux LINUX récents et essayons d'y intégrer finement l'utilisation des pilotes des réseaux MYRINET pour en tirer les meilleures performances.

ORFA a tout d'abord été implanté en espace utilisateur pour étudier l'impact de l'utilisation d'un réseau rapide lors de l'accès aux fichiers distants, sans subir les contraintes d'une implantation dans le noyau. Le client ORFA (voir la figure 2(a)) se présentait sous la forme d'une bibliothèque interceptant de manière transparente tous les accès aux fichiers distants et supportant les appels aux primitives `fork`, `exec`, ... [6].



(a) Modèle ORFA. Une bibliothèque en espace utilisateur intercepte les appels de l'application et les redirige vers un serveur distant.

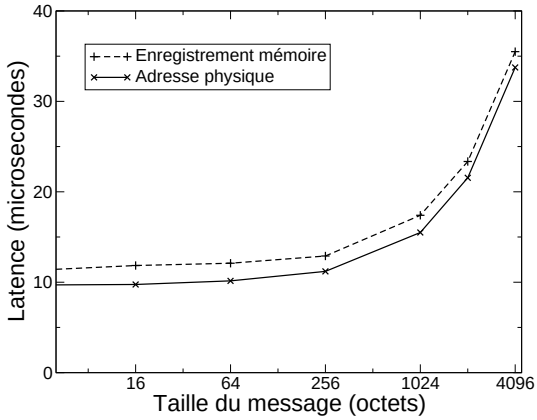
(b) Modèle ORFS. Un système de fichiers dans le noyau transmet à un serveur distant les requêtes que lui passe l'application par l'intermédiaire des couches systèmes.

Figure 2: Modèles de ORFA et ORFS.

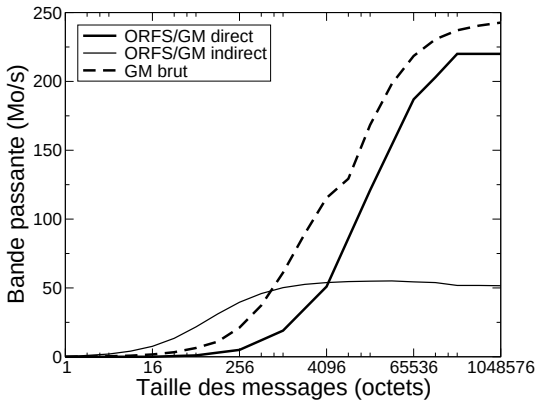
Nous avons montré dans [5] que le réseau MYRINET pouvait être utilisé très efficacement dans ce cadre pour transférer des quantités importantes de données. Par contre, l'accès aux métadonnées (les attributs des fichiers), ne bénéficie pas suffisamment de la faible latence du réseau. Nous avons ensuite travaillé sur ORFS (*Optimized Remote File System*), le portage du client ORFA dans le noyau LINUX (voir la figure 2(b)). Cela a permis d'une part de bénéficier des caches du VFS (*Virtual File System*) pour améliorer l'accès aux métadonnées, et d'autre part d'envisager une validation beaucoup plus large de nos travaux (puisque les systèmes de fichiers dans les grappes sont pour la plupart implantés dans le noyau).

Dans nos travaux, nous avons utilisé GM 2.0.12 sur un noyau LINUX 2.4.26. Les machines utilisées sont des bi-Xeon 2.6 GHz munis de 2 Go de RAM et de cartes MYRINET PCI64C et PCIXD. Ce réseau est capable de soutenir 250 Mo/s bidirectionnels.

3.2 Utilisation du cache page avec GM



(a) Latence des communications avec enregistrement mémoire ou passage direct de l'adresse physique.



(b) Performances comparées des accès directs (zéro-copie) et indirects (par le *page-cache* et en utilisant l'adresse physique) dans ORFS.

Figure 3: Utilisation de l'adresse physique pour les communications depuis le cache de pages lors d'accès indirects aux fichiers distants.

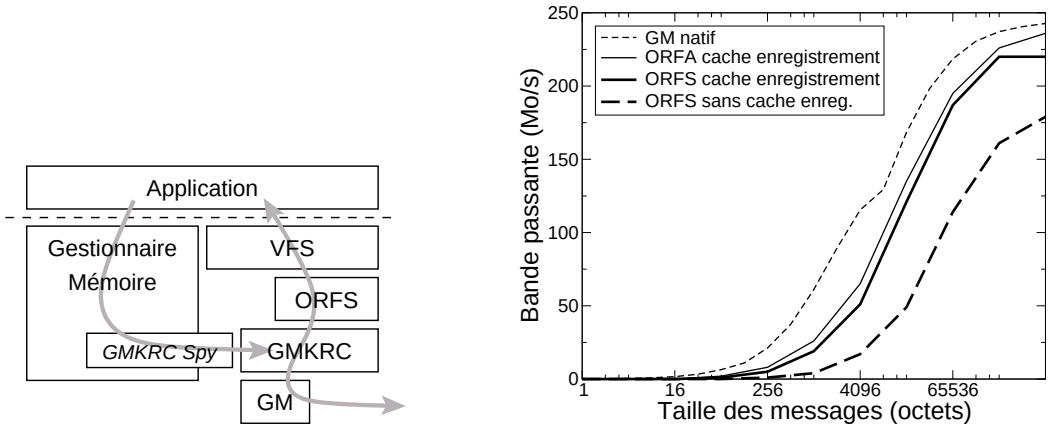
Nous avons ajouté à l'interface de GM des primitives de communication basées sur les adresses physiques et le support nécessaire dans le MCP (*Myrinet Control Program*, le programme qui tourne dans la carte d'interface). Les accès aux fichiers distants via le *page-cache* dans ORFS reposent donc sur le passage de l'adresse physique des pages mises en jeu à l'interface réseau. Cette stratégie permet par ailleurs de gagner en latence puisque la carte d'interface n'a plus à chercher dans sa table l'adresse physique correspondant à l'adresse virtuelle qu'on lui passe. Nous avons mesuré un gain de 0.8 μs du côté émetteur et du côté récepteur (figure 3(a)), ce qui représente un gain de 15 % en latence pour des petits messages.

L'impact de l'utilisation des adresses physiques dans ORFS pour transmettre les données entre le cache de pages et un serveur distant est détaillée sur la figure 3(b). Nous mesurons le débit vu de l'application. Le remplissage progressif du cache de pages du noyau au fur et à mesure des besoins de l'application présente l'inconvénient de n'utiliser que des transferts de la taille d'une page (4 ko sur nos machines), ce qui réduit la bande passante observée. On constate d'ailleurs que les performances observées sont légèrement supérieure au cas direct pour cette taille de message, et ce malgré la copie nécessaire pour transférer les données entre le cache de page et l'application (2 μs sur nos machines, soit environ 5 % de la latence GM pour ces messages). Cela traduit la plus grande efficacité des accès par adresse physique.

En revanche, lorsque l'application demande des accès plus volumineux, les performances sont bien inférieures au cas direct puisque le système d'exploitation est incapable de regrouper les différents transferts de pages sur le réseau. Ce problème devrait disparaître avec les noyaux LINUX 2.6 qui permettent ce genre d'optimisation. Cela nécessite par contre des accès vectoriels au réseau, ce que GM ne supporte pas (voir la partie 4.1).

3.3 Accès direct aux fichiers distants sur GM

L'implantation des accès directs dans ORFS réalise des communications à partir de la mémoire utilisateur d'une ou plusieurs applications vers un port de communication du réseau MYRINET ouvert dans le noyau. L'enregistrement paraît être la plus simple solution dans ce cas, sous réserve d'implanter un cache d'enregistrement (comme dans le client ORFA). Cela impose de connaître les modifications de l'espace d'adressage des applications. Contrairement à l'espace utilisateur où on peut intercepter les appels de l'application pour connaître ces modifications, le noyau LINUX ne fournit aucun support pour ce faire. Nous lui avons donc intégré une interface générique nommée VMA SPY permettant à n'importe quel module externe d'être notifié des modifications de l'espace d'adressage (changement de mapping ou de protection, `fork`, ...).



(a) Accès directs dans ORFS avec GMKRC (cache d'enregistrement) et VMA SPY (notification des modifications d'adressage).

(b) Performance des accès directs dans ORFS avec et sans cache d'enregistrement. Comparaison avec GM natif et ORFA.

Figure 4: Accès directs avec le client ORFS sur GM.

Nous avons ensuite implanté un cache d'enregistrement générique dans le noyau nommé GMKRC (*GM Kernel Registration Cache*) qui utilise VMA SPY pour se maintenir à jour. (voir la figure 4(a)) GMKRC est de plus chargé de résoudre les collisions entre les espaces d'adressage des différents processus accédant à notre système de fichier. En effet, GM considère qu'un *port* ne peut pas être utilisé par différents processus. La carte d'interface est donc incapable de savoir dans quel espace d'adressage une adresse virtuelle se trouve. Nous avons résolu ce problème en recompilant le pilote de la carte en 64 bits et en stockant un descripteur d'espace d'adressage dans les bits de poids forts. Cette stratégie est masquée dans GMKRC, qui expose une interface traditionnelle en 32 bits.

Ces travaux permettent d'obtenir de très bonnes performances sur GM dans le noyau (voir la figure 4(b)). La comparaison entre ORFS avec et sans cache d'enregistrement rappelle que les performances sont tributaires de la réutilisation des mêmes zones de mémoire par l'application. Les performances d'ORFS restent en deçà de celles de ORFA à cause des appels systèmes et la traversée du VFS, mais elle permet de bénéficier du cache de pages si l'application ne demande pas des accès directs.

4 Propositions pour l'évolution des interfaces de programmation des réseaux des grappes

La nécessité de modifier GM et le noyau LINUX pour pouvoir utiliser efficacement le réseau dans ORFS illustre les difficultés qu'on rencontre actuellement lorsque l'on tente de bénéficier des réseaux hautes performances des grappes dans un contexte différent des applications parallèles.

Les problèmes étudiés dans la partie précédente ne sont pas inhérents à GM. On les rencontre en fait dans la plupart des interfaces logicielles basées sur l'enregistrement mémoire. Nous proposons quelques pistes pour faire évoluer les interfaces de programmation des réseaux rapides vers un usage plus aisé et efficace dans des contextes particuliers.

4.1 Adresse physique et communications vectorielles

Nous avons montré à la partie 2.3.1 que l'enregistrement ne convenait pas aux communications avec le cache de pages du noyau. L'utilisation de primitives basées sur les adresses physiques se révèle en fait intéressante pour tout type de communication réalisée à partir d'un contexte noyau. Il y est en effet possible d'obtenir facilement, l'adresse physique des pages du cache de pages, mais aussi de n'importe quelle zone mappée dans l'espace noyau (qui est toujours verrouillée) ou dans l'espace utilisateur d'une application (qu'on devra verrouiller nous-même). La mémoire noyau peut notamment servir pour les messages de contrôle tandis que la mémoire utilisateur servira aux transferts zéro-copie entre l'application et le réseau comme on l'a vu en 2.3.2. Dans les deux cas, on échappe au très coûteux enregistrement mémoire et on gagne en latence, ce qui peut être important dans un protocole utilisant de nombreux messages de contrôle (messages de petite taille). L'interface réseau doit donc proposer un accès mémoire spécifié directement avec des adresses physiques.

Cependant, des zones mémoire virtuellement contiguës ne le sont généralement pas physiquement. C'est notamment le cas des zones en espace utilisateur. Tout transfert de plus d'une page (4 ko sur les architectures IA32) entre l'espace utilisateur et le réseau sera donc découpé en plusieurs segments. De la même façon, le traitement simultané de plusieurs pages du cache de pages qui correspondent à des parties successives d'un fichier conduit au même problème (voir la partie 2.3.1).

Même s'il reste possible d'utiliser plusieurs communications pour transférer ces différentes zones, cela rend l'utilisation du réseau beaucoup plus complexe, notamment si l'interface de programmation impose de contrôler le nombre de requêtes soumises simultanément. La solution la plus simple ici est d'utiliser des communications vectorielles, c'est-à-dire une seule primitive transférant plusieurs zones mémoire discontiguës. Certaines interfaces (mais pas GM) fournissent d'ores et déjà de telles primitives pour implanter efficacement MPI. Fournir dans le noyau une variante de ces primitives basée sur les adresses physiques semble donc être une solution intéressante.

4.2 Application à Myrinet eXpress, MX

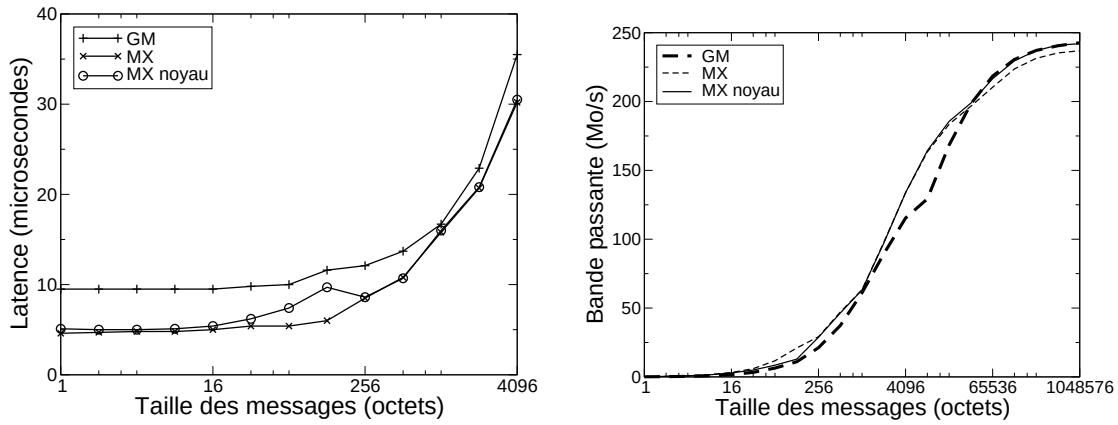
MX (*Myrinet eXpress* [10]) est le nouveau pilote logiciel des réseaux MYRINET. Il a pour principale caractéristique de proposer une interface native très proche de MPI (qui reste la principale application), notamment les communications vectorielles et l'utilisation d'adresses physiques dans la carte d'interface (aucun enregistrement mémoire explicite).

Nous avons travaillé en collaboration avec MYRICOM pour d'une part descendre et exposer l'interface de programmation dans le noyau, et d'autre part l'étendre en y intégrant nos idées pour ainsi améliorer l'interaction entre les systèmes de fichiers distribués et les réseaux MYRINET. L'interface noyau de MX propose désormais nativement le support optimisé des différents types d'adressage. L'utilisateur doit préciser explicitement ce type :
Virtuel utilisateur : MX se charge de verrouiller et traduire en adresse physique.
Virtuel noyau : Ces zones sont déjà verrouillées. MX se contente de traduire les adresses.
Physique : L'application verrouille elle-même si nécessaire.

La distinction entre les deux premiers types permet un support plus générique de la différence entre espace d'adressage noyau et utilisateur. Les noyaux LINUX standard sont en fait un cas particulier de système d'exploitation où il est possible de deviner à quelle espace une zone mémoire appartient en observant uniquement son adresse virtuelle. Comme ce n'est pas le cas dans tous les systèmes, cette distinction est nécessaire dans l'interface de MX.

Cette interface a été intégrée dans MX mais certaines optimisations restent nécessaires.

4.3 Évaluation des performances sur MX



(a) Latence des petits messages.

(b) Bande passante.

Figure 5: Performances comparées de MX et GM.

On peut constater sur la figure 5(a) qu'en terme de latence, MX est bien meilleur que GM puisqu'il atteint 5 μs contre 9. L'interface noyau de MX présente une latence légèrement supérieure à celle en espace utilisateur. Cela est dû, d'une part, aux seuils des différents modes de fonctionnement qui n'ont pas encore été adaptés au noyau, et d'autre part aux copies intermédiaires qui n'ont pas encore été supprimées pour profiter des particularités du contexte noyau.

Comme on l'a vu à la figure 3(a), dans GM l'utilisation d'adresses physiques réduit la latence. Le traitement est alors plus rapide dans la carte d'interface. Par contre, dans MX, on n'observe pas de différence puisque la carte d'interface ne manipule déjà que des adresses physiques et que le coût de traduction dans l'hôte est négligeable.

La latence n'a une influence que sur les performances des petits messages, c'est-à-dire essentiellement des messages de contrôle ou de métadonnées dans un système de fichiers distribué. La bande passante a un impact beaucoup plus important sur l'accès réel aux données des fichiers distants. L'absence d'enregistrement mémoire explicite dans MX permet d'obtenir un bande passante brute similaire à GM lors que l'utilisation de son cache d'enregistrement est optimale (voir la figure 5(b)). On remarque par ailleurs que l'interface noyau de MX a une bande passante légèrement supérieure pour les messages larges. La raison est que l'enregistrement mémoire utilisé en interne dans ce cas est plus simple lorsqu'on est déjà dans le contexte du noyau.

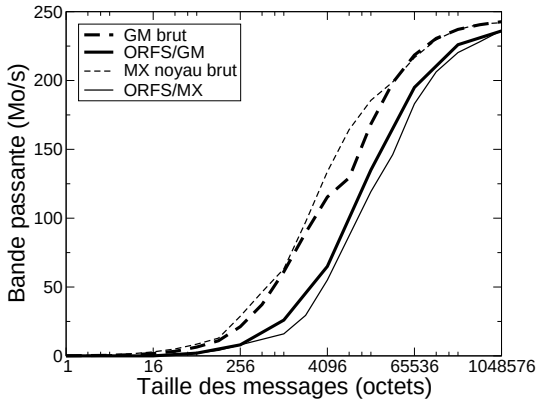


Figure 6: Efficacité de l'utilisation de MX et GM lors d'accès directs dans ORFS.

Par contre, dans une utilisation réelle telle que l'accès direct aux fichiers à partir du noyau, les performances sur MX restent légèrement en deçà de GM (voir la figure 6). Il faut bien voir que les performances de ORFS sur MX sont ici indépendantes des zones mémoire utilisées par l'application. Par contre, sur GM, une faible réutilisation des zones mémoire pourra réduire l'apport du cache d'enregistrement.

Par ailleurs, nous sommes en train d'optimiser l'implantation noyau de MX en régime intermédiaire (entre quelques ko et quelques dizaines). En effet, ce régime de fonctionnement utilise une copie pour éviter d'avoir à entrer dans le noyau pour traduire les adresses virtuelles. L'optimisation dans le noyau pour ce type de messages éviterait cette copie en profitant du fait que l'adresse physique est disponible.

5 Conclusion

Cet article étudie les problèmes rencontrés lors de l'utilisation des interfaces de programmation des réseaux de grappe dans un contexte différent des applications parallèles. La spécificité de ces interfaces logicielles rend difficile leur interaction avec un contexte tel que les systèmes de fichiers distribués.

Nous avons tout d'abord présenté des modifications de l'interface GM des réseaux MYRINET. D'une part, l'utilisation des adresses physiques se révèle très utile pour les communications avec des zones mémoire spécifiques telles que le cache de pages du noyau. D'autre part, l'implantation du cache d'enregistrement GMKRC et du mécanisme VMA SPY pour notifier les modifications d'espace d'adressage permet de supporter efficacement les accès directs

aux fichiers distants. Les performances obtenues dans le cadre de notre système de fichiers distribué ORFS montrent l'importance d'une bonne interaction entre l'interface réseau et les couches système d'accès aux fichiers.

Nous avons ensuite proposé différentes idées pour les futures interfaces de programmation des réseaux des grappes. L'utilisation de l'adresse physique et la généralisation des communications vectorielles se révèlent importantes pour une utilisation efficace du réseau lors des accès aux fichiers distants, mais en fait aussi pour d'autres types d'applications dans le noyau. Nous les avons appliquées dans la nouvelle interface MX des réseaux MYRINET qui est en cours de développement. Les premières évaluations de performance sont intéressantes même si certaines optimisations sont encore nécessaires dans le cadre spécifique du noyau.

References

[1] 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.

[2] Philip H. Carns, Walter B. Ligon III, Robert B. Ross, and Rajeev Thakur. PVFS: A Parallel File System for Linux Clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, 2000. USENIX Association.

[3] Cluster File Systems, Inc. Lustre: A Scalable, High Performance File System, November 2002. http://www.lustre.org.

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

[5] 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*, Santa Fe, New Mexico, April 2004. IEEE Computer Society Press.

[6] Brice Goglin and Loïc Prylli. Transparent Remote File Access through a Shared Library Client. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'04)*, volume 3, pages 1131–1137, Las Vegas, Nevada, June 2004. CSREA Press.

[7] B. Lahaise. Design Notes on Asynchronous I/O (AIO) for Linux, 2002. http://lse.sourceforge.net/io/aionotes.txt.

[8] K. Magoutis, S. Addetia, A. Fedorova, and M. I. Seltzer. Making the Most out of Direct-Access Network Attached Storage. In *Proceedings of USENIX Conference on File and Storage Technologies 2003*, San Francisco, CA, March 2003.

[9] Myricom, Inc. GM: A message-passing system for Myrinet networks, 2003. http://www.myri.com/scs/GM-2/doc/html/.

[10] Myricom, Inc. Myrinet Express (MX): A High Performance, Low-level, Message-Passing Interface for Myrinet, 2003. http://www.myri.com/scs/.

[11] 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.

[12] Gregory F. Pfister. Aspects of the InfiniBand Architecture. In *Proceedings of the 2001 IEEE International Conference on Cluster Computing*, pages 369–371, Newport Beach, CA, October 2001.

[13] Evan Speight, Hazim Abdel-Shafi, and John K. Bennett. Realizing the Performance Potential of the Virtual Interface Architecture. In *International Conference on Supercomputing*, pages 184–192, 1999.

[14] 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.

[15] 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.