Laboratoire de l'Informatique du Parallélisme

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

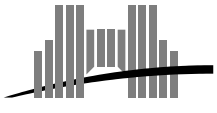
Optimizations of Client's side communications in a Distributed File System within a Myrinet Cluster

Brice Goglin,
Loïc Prylli ,
Olivier Glück

Avril 2004

Research Report N° 2004-24

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



Optimizations of Client's side communications in a Distributed File System within a Myrinet Cluster

Brice Goglin, Loïc Prylli , Olivier Glück

Avril 2004

Abstract

This paper presents a study of the interaction between high-speed interconnects and a distributed file system client. We use our ORFA remote file access protocol and MYRINET network with the GM software layer to show how the highly specific programming model of high-speed interconnects may be used in a high performance distributed file system client.
We present both a user-level and a kernel-level client implementations with either buffered or non-buffered accesses. These implementations show high performances and may be transparently used by applications. Our improvements focus on pin-down cache techniques and memory registration issues.
Our GM modifications show no impact on its performances while the network usage in the distributed file system client is improved.

Keywords: High-Speed Local Network, Distributed File System, Cluster, Memory Registration, Myrinet, Linux.

Résumé

Cet article presente une étude de l'interaction entre les réseaux hautes performances et un client de système de fichiers distribué. Nous utilisons notre protocole d'accès aux fichiers distants ORFA et un réseau MYRINET avec sa couche logicielle GM pour montrer comment le modèle de programmation très spécifique des réseaux rapides peut être utilisé dans un client de système de fichiers distribué performant.
Nous présentons à la fois un client en espace utilisateur et un en espace noyau avec des accès bufferisés ou non. Ces implémentations montrent leur efficacité et peuvent être utilisées de manière transparent par les applications. Nos améliorations s'intéressent aux problèmes liés aux techniques de cache d'enregistrement mémoire.
Nos modifications dans GM ne montrent aucun impact sur ses performances tandis que son utilisation dans le client de système de fichiers distribué est améliorée.

Mots-clés: Réseau local hautes performances, système de fichiers distribué, grappe, enregistrement mémoire, Myrinet, Linux

1 Introduction

Scientific applications now largely use parallel computers to run heavy computations. High-speed interconnects such as MYRINET [BCF+95] or INFINIBAND [Pfi01] are used for about ten years to build powerful clusters of workstations, replacing traditional massively parallel supercomputers. A large amount of interesting work has been done to improve communications between cluster nodes at the application level through the use of the advanced features in the network interface. Meanwhile, storage access needs to reach similar performance to read input data and store output data on a remote node without being the bottleneck.

While usual application communications should obviously occur at user-level, distributed file system were initially implemented in the kernel to supply transparent remote accesses. They were designed for traditional networks and caching was used to compensate the high latency. In a cluster environment, two directions are studied to improve the performance of distributed file systems: distribute the workload across multiple servers or efficiently use the low latency and high bandwidth of the underlying high-speed network.

This paper presents a study of this second direction. We use our ORFA [GP03] user-level and kernel implementations as a distributed file system test platform to improve the usage of high-speed interconnects in this context. Our experiments on a MYRINET network with the GM driver [Myr03] show how such a specific programming model may be used to enhance remote file access performance on the client's side.

Section 2 presents the specificity and features of high-speed interconnects such as MYRINET. Section 3 describes distributed file systems in the context of clusters and our ORFA implementations. Section 4 details how we have enabled non-buffered remote file access in ORFA. Section 5 presents how buffered access may be used by applications in a transparent and efficient way.

2 High-Speed Local Networks

The emergence of high performance parallel applications has raised the need of low latency and high bandwidth communications. Massively parallel supercomputers provided integrated communication hardware to exchange data between the memory of different nodes. They are now often replaced by clusters of workstations which are more generic, more extensive, less expensive and where communications are processed by dedicated network interfaces.

2.1 Overview and Key Features

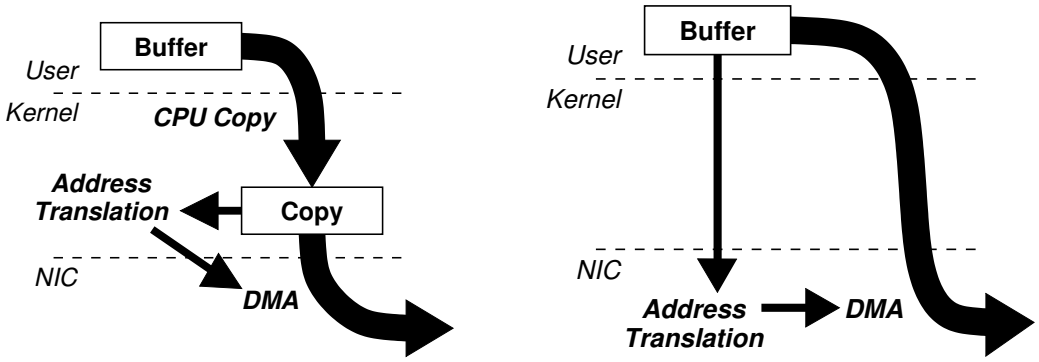
Reducing the critical path between two nodes on the network requires firstly to avoid intermediate buffers and thus copies, secondly to process communications at the user-level and thus bypassing system calls. The usual TCP/IP stack implies to copy the user buffer into the kernel memory where one or more headers are added before the entire message is passed to the network interface (see Figure 1(a)). The U-NET system introduced a *zero-copy* user-level communication model [vEBBV95] allowing buffer transfer to or from the network without useless copies. It reduces the overall latency and host overhead since its CPU is no more involved in any copy.

More recent interfaces such as MYRINET, QUADRICS [PFHC03] and INFINIBAND are based on network processors that are able to significantly assist the local CPU in communication achievements. Active researches on software layer design have enabled fully efficient usage of this hardware. Some software layers such as BIP [PT97] on MYRINET or SISCI on SCI [ISSW97] achieve a one-way latency under 3 μs for several years, and INFINIBAND may now achieve 1 GB/s.

2.2 Programming Model

Most recent communication libraries are based on message passing over DMA. It raises the problem of address translation and completion notification.

This programming model uses send or receive requests submission to the network interface (NIC) which processes them asynchronously. Request completion is then notified to the application through active polling or interrupts. The event-driven API that is actually exported by these software layers is thus highly specific and may still not be easily merged with the traditional POSIX I/O API which is synchronous.



(a) Usual network stack.

(b) High-speed interconnect stack.

Figure 1: Comparison of usual and high-speed network stack. Instead of using an additional buffer in the OS, high-speed networks bypass the OS and directly transfer data between the user application and the link through an address translation table in the NIC.

The requests that are passed to the NIC describe target buffers with their virtual addresses. The DMA engine of the NIC requires these to be translated into physical addresses. However, this translation generally implies to go in the kernel and thus uses a system call. As address translation is a very simple operation, the system call overhead (about 500 ns on modern stations) looks too high (it is non-negligible with regard to network latency). This has led to the idea of totally bypassing the OS and translating addresses in the NIC (see Figure 1(b)).

The most used solution consists of registering communication buffers to the NIC and pining them down in memory to prevent its physical location to change. Several software layers such as BIP provide on-the-flight transparent registration of the buffers involved in the user requests. Some other layers such as GM require the user to explicitly register its buffers. This allows to optimize registration when buffers are contiguous or used several

times.

QUADRICS ELAN uses a variant of this model where no memory pining is required. Intrusive kernel modifications allow to update of the NIC table each time a page mapping is modified.

2.3 GM on MYRINET and Memory Registration

We chose to use MYRINET networks in our work because they are the most commonly used high-speed interconnect and the firmware running in the interface may be easily reprogrammed. While several software layers such as BIP may provide better point-to-point communication performance, especially concerning latency, we use the main software suite GM provided by MYRICOM because of its great reliability and scalability.

GM was initially designed on the message passing model. The API allows to post multiple receive buffers and submit send requests. Target buffers are identified through an explicit message tagging protocol. As described in the previous section, the GM API exports explicit memory registration routines which use the operating system to pin pages down and register address translation in the NIC.

The table of registered page translation is stored in the NIC. It provides high performance communications. However, we will show in Section 4.1 that the registration overhead is much higher than the 8 μs latency that GM may achieve. This proves that this model has to be used in a clever way to avoid a huge performance penalty. Other software layers that require the user to explicitly register buffers, such as INFINIBAND and VIA [SASB99], also suffer of it, even if this may not be as critical as in GM.

3 Distributed File Systems in Clusters

The emergence of high-speed interconnects have given rise to parallel applications making the most out of the underlying network hardware. MPI [For94] and VIA software layers are now mainly used and have thus influenced the design of recent network access layers. This is the reason why very efficient implementations of MPI and VIA are now available on most interconnects.

Nevertheless, parallel applications require efficient communication between distant application tasks and fast access to remote storage. High performance distributed file systems have special requirements that have not really been considered when designing most underlying network access layers.

3.1 Design choices

Applications running on the client's side may manipulate files through different interfaces. The traditional POSIX API is exported by the GLIBC and describes files as pathname and descriptors that are passed to the kernel through system calls.

3.1.1 Kernel design of the Client

Application requests are handled in the kernel through the VFS (*Virtual File system*) which makes all supported file systems uniform. This is the way most distributed file systems, for instance NFS (*Network File System* [SGK+85]) are implemented. They have to full-fit the VFS

API when providing their specific routines. However, it ensures that they are transparently accessible in any legacy application.

The VFS uses the underlying storage and a metadata cache to quickly convert pathnames and file descriptors into *inodes* (the physical files) and *dentries* (the directory entry names that compose pathnames). The developer may choose to use the underlying *Page-Cache* or not. It allows to cache real data in the kernel and enables *read-ahead* and *delayed write*. Nevertheless, the page granularity access that it implies may lead to an under-utilization of the network bandwidth.

3.1.2 User-level design of the Client

It is possible to modify this traditional behavior by implementing a new user-level library providing its own file access routines. This model enables full configuration, for instance user-level caching or access granularity.

The library may export the traditional POSIX API of the GLIBC, allowing a transparent usage in any legacy application. POSIX I/O have the advantage of being generic and portable. But it is difficult to make the most out of particular contexts such as a cluster environment where the underlying I/O subsystem is fast and allows to overlap I/O with computations.

This is the reason why several specific API have been designed in the context of clusters to serve parallel application file access in a more efficient manner. The MPI-IO standard was introduced to provide several new features such as collective I/O primitives and non-contiguous access patterns. For instance, several routines were added to the PVFS API [CLRT00] to make MPI-IO implementation easy through the ROMIO interface [TGL99]. In addition, DAFS [MAF+02] introduced a highly specific API which is fully asynchronous and provides an efficient event-driven model with completion groups. However, these new API require to rewrite target applications. But such a specific design also enables better interaction with the application and the underlying hardware.

3.2 Research Topics

From our point of view, three main research topics concerning distributed file systems in a cluster environment may be distinguished: client caching, workload and data striping, and network utilization.

In the case of a kernel implementation or a specific caching system in user-level, the developer has to maintain consistency across all client caches and the server. Active researches have been led in this area, from NFS and its relaxed coherency, to INTERMEZZO [Bra99] which allows the survival of clients that are disconnected from the server. In the context of clusters, PVFS and LUSTRE [Clu02] are able to maintain consistency between hundreds of clients and multiple servers.

As performance requirements and the size of parallel applications grow, the I/O throughput of file system servers must increase. The physical bandwidth of network links and I/O workload saturation of servers led to use multiple servers in the same network. As already shown by GPFS [SH02] and PVFS, parallelization of workload and striping of data across different servers and storage systems allow to sustain large cluster and parallel application needs.

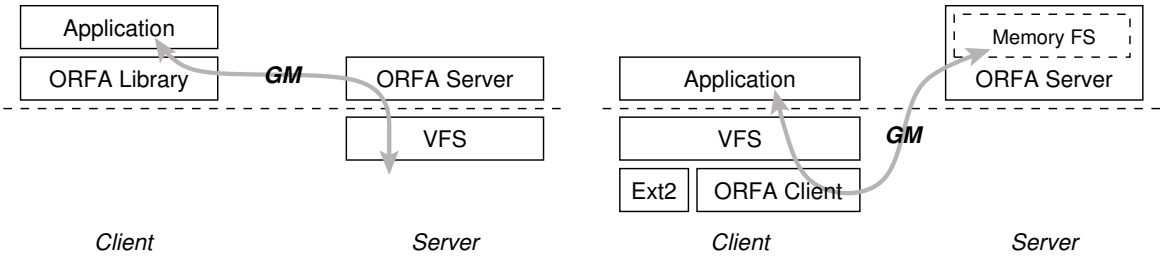
As described in Section 2, high-speed networks allow to transfer data between distant applications without intermediate copy and host overhead. In the context of distributed

file systems, this leads to the idea of connecting the client application buffers to the remote storage server using zero-copy mechanisms. DAFS was designed on this idea [MAFS03]. Several optimizations such as client initiated data transfer with remote exceptions were presented.

In this research area, we study and try to optimize the client's side design of a distributed file system. Several existing projects targeting general purpose remote file access may be improved by efficiently using the underlying network. The amount of copy may often be reduced while memory registration should be used in a more optimized way. In this context, we developed ORFA (*Optimized Remote File system Access*) to study the impact of an efficient usage of the underlying network on general purpose remote file access.

3.3 Overview of ORFA

Most file system API were designed on application requirements while most network access software layers target communications between user applications. This makes the interaction between network and file system non trivial.



(a) ORFA User model accessing usual file systems in the server.

(b) ORFA Kernel model with a custom memory file system.

Figure 2: ORFA user and kernel model.

Our first idea was to study the fastest way to transfer data from client application buffers to remote storage subsystem. High-speed interconnects now enable very low latency (less than 10 μs) which makes network transfer much faster than disk access and approaching local file access hitting the VFS cache. We thus assume that the hypotheses on network performance that led to the design of traditional network file systems are no more valid in a cluster environment. Therefore, we began our study by removing any intermediate layer on the critical path, especially any cache or intelligence on the client's side.

This first implementation [GP03] was a user-level library client accessing a user-level server through GM, BIP or TCP. Performance evaluation [GP04a] led us to the conclusion that remove caching does not suit a specific class of applications. We thus decided to port the ORFA client into the LINUX kernel. It supplies to any legacy applications not only a transparent access to the remote file system but also a transparent use of the metadata VFS cache. The ORFA server may access its local file system through usual system calls (see Figure 2(a)) or a custom memory file system in user-space for our experiment needs (see Figure 2(b)).

3.3.1 User-level Implementation

The aim of the ORFA user-level client implementation was to achieve efficient zero-copy data transfer between client and server side. The main difficulty was to enable a transparent remote file access for any legacy application. This transparent behavior is achieved through a shared library which exports a traditional POSIX file access API. This library is pre-loaded through the `LD_PRELOAD` environment variable and thus overrides usual GLIBC routines.

Each application I/O call is intercepted by the library which converts it either into a network request to the remote server, or into a usual call to local file systems. This lightweight client only handles virtual file descriptor semantics. For instance, it ensures that remote files emulate usual file behavior under `fork` or `exec`. This work has been presented in [GP04b] with details on the preservation of local file access compatibility.

Efficient data transfers from the client application buffers to the network required to deal with memory registration. We will describe how we solve this issue in Section 4.1.

3.3.2 Kernel Implementation

Several performance results have been presented in [GP04a]. On one hand, these results showed that the absence of file attribute caching on the client's side may not be compensated by the very low latency of high-speed interconnects. On the other hand, the absence of data caching allows to benefit from the network bandwidth by avoiding the page granularity during large requests.

Improving metadata access performance through client's side caching requires to reproduce VFS caches and pathnames and descriptor conversion. Instead of re-implementing these in user-level, we chose to move the ORFA client in the kernel and automatically take advantage of the VFS caches. Moreover, it enables to propose optimizations of the network usage in kernel file systems and thus increases the amount of potential target projects which may benefit of our work.

In our implementation, we use a simple protocol to maintain a relaxed coherency as in NFS and we keep the possibility for the application to choose to bypass the VFS page-cache or not. Thus, buffered and direct data accesses are supported but they raise their own problems when dealing with memory registration since network layers such as GM were not designed to achieve communications from/to the kernel space.

4 Direct File Transfer in a Cluster Environment

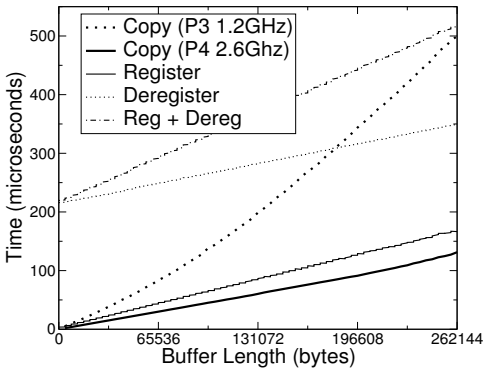
The user-level implementation of ORFA uses non-buffered data transfer. It corresponds to the `O_DIRECT` flag that recent operating systems allow to pass when opening a file. This kind of access is for instance useful for large applications managing their own data cache. It avoids read-ahead and delayed write which may consume physical pages, but enables direct data transfers between user application buffers and the storage system. Such accesses may easily saturate the network link since data transfers may be very large. This flag is supported by our kernel implementation of the ORFA client. The next section describes how both user and kernel implementations may interact in an efficient way with the GM software layer of a MYRINET networks in case of non-buffered data access.

4.1 User-Level Implementation

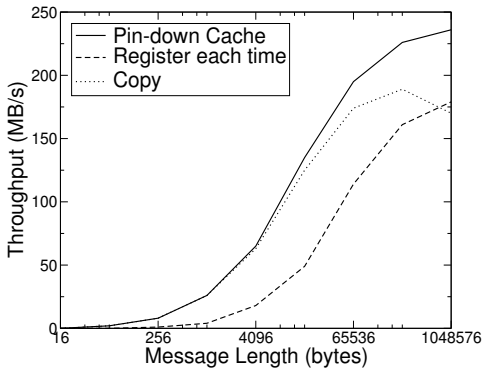
Implementing a shared library which acts on the application buffers raises the problem of their unpredictable locations. The user-level ORFA library intercepts I/O requests from the application and has to pass corresponding buffers to the GM software layer to send or get data from the remote server directly.

GM requires the upper layer to call the explicit memory registration routines before using communication primitives. However, the application was compiled for traditional I/O and thus does not call any registration routine. This problem might be avoided if we were not targeting legacy applications since a specific API may for instance ask the application to use pre-registered memory allocation routines.

Thus, our library has to transparently register and deregister the application buffers on-the-flight. Actually, the same problem appears when implementing a MPI layer because the first MPI standard [For94] was designed before the emergence of recent high-speed interconnects and thus does not care about memory registration issues.



(a) Comparison between the overhead of copy and registration primitives.



(b) Comparison of remote read performance with ORFA user-level implementation when using a pin-down cache, basic registration or a copy.

Figure 3: Impact of copy and memory registration on communications. LINUX 2.4.24 was used with GM 2.0.8 on LANAI 9 MYRINET interfaces.

Figure 3(a) shows the high overhead of memory registration and especially deregistration. Registration processing implies the traversal of the whole target buffer to make each page present in physical memory and then stores their physical address in the NIC. Deregistration removes these addresses from the NIC, it is very expensive because the GM model requires to interrupt the NIC to ensure that its translation table is updated.

On a modern workstation, the host may copy 1 GB/s. This means that copying the application data in a pre-registered buffer is generally faster than registering and deregistering this buffer. On the other hand, memory copy consumes CPU cycles, that are useful for computation especially in scientific applications. In addition, multiple communications involving the same buffer require only one registration.

These points should be considered when designing a registration strategy. Replacing registration by copy may only be encouraged for small buffers, about a few pages at most.

On-the-flight registration may be used for all other cases. However, the developer should try to register contiguous buffers at the same time, re-use same buffers and avoid deregistration which is the main part of the overhead. The ORFA registration strategy thus uses a *Pin-Down Cache* [TOHI98] implemented on top of GM in the ORFA network access layer. The idea is to delay deregistration until really required (when no more other pages may be registered).

This model has the drawback of requiring to trace address space modifications since the NIC register cache have to be updated when a page mapping changes. This is especially the case for large memory allocation through anonymous file mapping. This issue was solve in ORFA library by intercepting memory management routines and then updating the pin-down cache.

As usual parallel applications try to avoid page swapping because of its high overhead, they generally use less memory than physically available. This implies that potential I/O buffers may fit in physical memory. As GM may register a large part of the available physical memory, we observe very few pin-down cache misses. Thus, almost no deregistration is required.

Figure 3(b) shows the performance of remote reads with ORFA. Similar performance is observed when writing. A custom file system implemented in user-space (like in Figure 2(b)) ensures that the server is not the bottleneck. The pin-down cache increases the throughput by more than 25 % over basic registration implementation (register/deregister each time a send or receive occurs). Copying data in a pre-registered buffer allows to reach same performance for small requests but wastes too much CPU cycles for big requests. The ORFA user-level implementation therefore shows a very good utilization of MYRINET network performance.

4.2 Non-buffered File Access in the Kernel

The ORFA kernel implementation is based on a LINUX kernel module which registers a new file system type and provides several routines to handle VFS objects through requests to the remote server. Mounting an ORFA file system then implies the opening of a GM port in the kernel (the entry point that GM provides to the application to access the network). For each mount point, one GM port is used.

When the user application opens an ORFA file, the `O_DIRECT` flag is checked to know whether the user asks for non-buffered file I/O or not. Non-buffered file access with ORFA in the kernel basically means that data should be transfered directly between the user application buffers and the network while staying in the kernel context.

Our idea is to use the same model as in the ORFA user implementation: a memory registration strategy with a pin-down cache. Registering user memory to a GM kernel port raises the problem of sharing this port between multiple address spaces. This issue is more precisely detailed in Section 4.3.

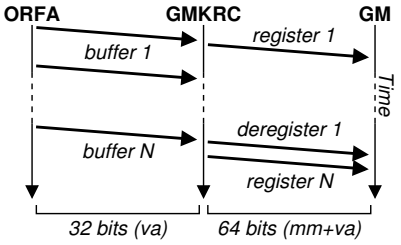
A pin-down cache is maintained in the kernel for each mount point and thus for each GM port. Keeping it up-to-date requires to be able to flush it when buffers are no more used or valid. It is possible to intercept memory management routines in user-space as in ORFA user-level implementation. But updating the kernel registration cache would then involve system calls. Thus, we proposed a more efficient method to update this cache without intercepting the user routines. However, our kernel module is only aware of file access and thus does not know when a buffer is unmapped or when an application is closed. We will describe in Section 4.4 how the pin-down cache may be updated in this context.

4.3 Sharing a Kernel Port between several User Processes

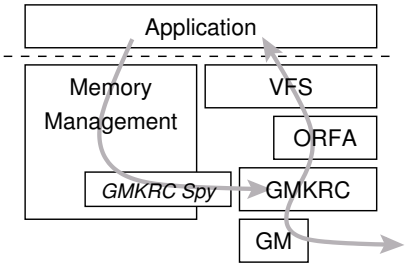
Registering a buffer with GM stores in the registration table a structure composed of the corresponding port, the virtual and physical addresses. Sharing the GM kernel port between several processes implies that this structure may be the same for two pages of different processes, making the NIC use the wrong target buffer. An unique virtual address is needed for each page before its registration.

A safe idea consists of re-mapping user pages in the kernel space where the address space is common to all processes. But in LINUX, remapping a page in the kernel is either very limited or has an high overhead because of the TLB (*Translation Look-aside Buffer*) flush in the processors (about 100 μs). The ORFA kernel implementation thus keeps user buffer mapped in user space and we modified the GM implementation to prevent collisions between same virtual zones of different address spaces. The idea is to get a unique identifier for these zones. Basically, this requires to get an address space identifier and to store it in the structure that describes a registered page in the NIC. For instance, this identifier may be the *Process IDentifier*.

The GM programming model is not based on memory handles as VIA and INFINIBAND. We are currently working on a specific model where each registered buffer is associated to an unique imaginary zone in a fake GM space through a traditional memory allocation algorithm. This model raises several problems that we plan to study.



(a) GMKRC principles. Deregistration is delayed. The virtual address (*va*) is prefixed with the the address space pointer (*mm*).



(b) Schematic view of user-memory registration from the kernel with GMKRC and VMA SPY.

Figure 4: Non-buffered access in ORFA kernel implementation.

We started with a more simple solution based on recompiling the GM firmware with 64 bits host addresses on a 32 bits architecture. The least significant 32 bits still describe the virtual address, while the most significant bits contain the address space identifier. This technique is hidden by a pin-down cache called GMKRC (*GM Kernel Registration Cache*) which exports a traditional 32 bits GM register primitive (see Figure 4(a)). This method is really simple to use and has very small impact on performance.

4.4 Tracing User Space Modifications from the Kernel

Maintaining the pin-down cache up-to-date requires that the memory management system of the kernel notifies it when a registered page is unmapped. However, the kernel does not

provide anything but the ability to notify the creator of pages (for instance, a file or a device that is mapped in memory).

The kernel divides the user address space into several VMA (*Virtual Memory Area*) where all pages map the same object with the same properties. We have implemented in the LINUX kernel a generic infrastructure called VMA SPY to help external objects to be notified of VMA modification (size or property changes and `fork`). It adds the ability to store in each VMA a list of spies containing several call-backs. When a VMA is modified, the corresponding call-backs of all its spies are launched.

Finally, each mount point creates a spy type associated with its GMKRC. Each time a page is registered, this spy is added to the containing VMA (if not already). When the memory management system modifies the VMA, GMKRC is notified through its spy call-backs. This method solves the address space modification tracing problem. But it requires to slightly patch the kernel. However, we took care of designing a generic enough infrastructure that may be reused in other circumstances.

This model may be compared to the *VM Coproc Hooks* in QUADRICS QSNET driver which introduces a way to update the copy of the host MMU in the NIC. This implementation adds a hook in each kernel path that modifies some page table entries. Our implementation has the advantage of adding hooks only for VMA which contain registered pages, and we know that very few VMA are generally concerned.

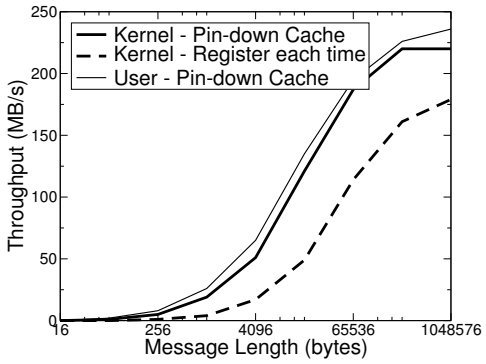


Figure 5: Performance of non-buffered access in ORFA kernel and user implementations. Pin-down cache and basic registration are compared.

The VMA SPY infrastructure shows full efficiency in notifying the GMKRC layer without adding any significant overhead. This model is summarized by Figure 4(b). As described in Figure 5, it achieves very good performance. It remains under the user-level implementation because of system call and VFS overheads. The pin-down cache has the same impact as in the user-level implementation.

5 Buffered Access in the Kernel

The previous section describes how the GMKRC layer combined with the VMA SPY infrastructure may efficiently provide a way to enable zero-copy non-buffered file access (`O_DIRECT` files) by solving the problem of registering user memory and managing a pin-down cache from the kernel. Since applications may choose to use buffered access, for instance to benefit

from read-ahead or delayed write, we now study the case where data is cached in the client's page-cache. In this case, data is transfered between user application buffers and the page-cache while the page-cache is kept up-to-date with remote storage asynchronously through ORFA protocol. This page granularity may reduce the network utilization since transfered buffer are small.

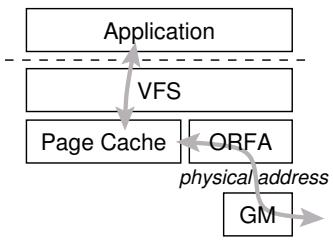
5.1 Registration of Kernel Pages

Instead of registering pages of the page-cache on-the-flight, we first imagined a better solution based on pre-allocation of a pool of registered pages that would be used to cache all ORFA files. It would avoid the overhead of on-the-flight registration. Instead of implementing these very intrusive modifications in the kernel to maintain this specific page-cache for ORFA files, we started by studying memory registration in this very specific context.

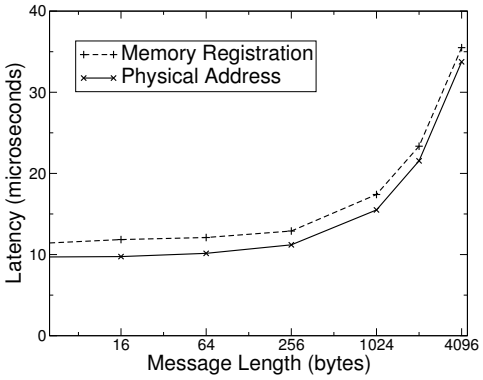
The memory registration model was designed to enable user-level zero-copy communications. User-level applications only manipulate virtual addresses while the kernel may also know their physical translation. The LINUX page-cache contains pages that may not be mapped in any address space, and thus may not have any virtual address. We explained in Section 4.3 that re-mapping pages is possible, but is either limited to few pages or implies too much overhead.

The idea here is not to use GM memory registration because it was not designed for such a case. In the context of the LINUX page-cache, pages are already locked in physical memory and their physical addresses are well-known. So, our implementation directly passes these addresses to the NIC.

5.2 Using the physical address



(a) Schematic view of buffered access. File pages are transfered between the page-cache and the network using their physical addresses.



(b) Latency comparison between communications using registered virtual memory and physical memory on both send and receive sides.

Figure 6: Buffered access in ORFA kernel implementation with physical address based communications.

We added to the GM firmware the ability to send or receive a message described by its

physical addresses. As GM does not support vectorial communications, these new routines are only available for buffers that are contiguous in physical memory. Figure 6(a) summarizes the way it is used in ORFA.

This model asks the caller to translate virtual addresses by itself if required and to ensure that the physical location may not change. It may allow to avoid GM memory registration primitives when sending or receiving pages that are not mapped (for the page-cache), headers or control messages that have been previously allocated as usual kernel memory, or even small user buffers (physically contiguous).

Figure 6(b) describes the benefit of this optimized model on message latency. We observed a 0.9 μs gain per page on both the sender and the receiver's side, that is about 20 % of small message latency. The reason is that the NIC directly uses the physical address passed in the request and does not have to find it in its hash table.

We also measured a 77 μs latency for an ORFA request to read or write one page-sized buffer. This kind of access has much more VFS overhead than non-buffered access because of the page-cache and an additional data copy between the page-cache and the application (about 5 μs per page on a modern station). However, the non-buffered and buffered latencies are almost equal. This shows that our physical address based communication model is very efficient.

5.3 Non-contiguous Communications

Buffered accesses are generally improved by grouping multiple pages in one request to maximize network bandwidth utilization and reduce the server workload. They may be used during read-ahead or delayed write. Such optimizations in most distributed file systems have led to the addition of generic routines in LINUX 2.6 kernel.

The fact that these pages may be not physically contiguous might not be problematic since most high-speed network interface support scatter-gather. These pages may for instance be contiguously mapped in virtual memory. However, as we explained in Section 4.3 that mapping may not be a good solution, and it is especially true when multiple pages need to be mapped contiguously. Thus, vectorial communications are required.

GM does not provide such a feature. We plan to add this support soon. Actually, vectorial communication based on physical addresses would be useful for optimized read-ahead and delayed write in buffered access. However, it might be also interesting for any communication initiated from the kernel, especially non-buffered access. Instead of using our on-the-flight registration implementation, a user-buffer may be simply pinned down in physical memory by the ORFA client and then a vector of physical pages would be passed to GM.

Moreover, vectorial communications based on virtual addresses would allow to implement efficiently non-contiguous access. Such a feature is already available in INFINIBAND, for instance because it is important for MPI and PVFS [WWP03]. All these ideas show the usefulness of this approach for efficient remote file access.

6 Conclusion and Perspectives

This paper exposes several aspects of the implementation of a remote file access client on a high-speed interconnect. We detailed why memory registration has been introduced to enable high performance zero-copy user-level communications. This specific model full-fits

the needs of MPI communications in parallel applications. But it shows obvious limitations when using it in a very different context, such as remote file access, especially in a kernel implementation.

Different kinds of general purpose remote file access were studied. An application may first choose non-buffered access, for instance to implement its own cache. We show that this model may be successfully integrated in the specific programming model of high-speed interconnects such as GM on MYRINET. A pin-down cache is used in ORFA to avoid deregistration overhead. The application may easily use a user-level library which allows more specific file access APIs. Generic file access may also benefit of a kernel implementation which ensures compatibility with any legacy application and provides metadata caching. However, this case requires much more work to maintain the pin-down cache up-to-date. We introduced a method to handle multiple user-level address spaces with the same GM port and add hooks in the memory management system of the kernel to transparently update the cache. An application may also choose to use buffered file access in order to benefit of read-ahead and delayed write. We detailed how the LINUX page-cache may be connected to remote storage through the GM API and showed that a communication model based on physical addresses is more adapted to this context. These implementations do not impact on GM performance and improve the way distributed file system clients may use it.

We plan to study non-contiguous file access which is one of the most important feature that is not available in traditional I/O while it is in MPI-IO and other specific API such as DAFS. Another important functionality to study is asynchronous I/O (AIO) which was just introduced in LINUX kernel 2.6. File systems may now define their specific routines for this type of file access, including both buffered and non-buffered access. An asynchronous file access implementation in ORFA might be simple because GM is an asynchronous network access layer. Network communication overlapping would naturally lead to file I/O overlapping. Making AIO and non-contiguous file access efficiently available in distributed file system in a cluster environment would contribute to reduce the advantage of specific file access API such as MPI-IO or DAFS. Moreover, such techniques may be useful on the server's side where scalability generally requires asynchronous processing.

This may lead to an almost complete study of the interaction between the high-speed interconnect programming model and the different kinds of general purpose file access that a traditional LINUX system provides. This work applies to the GM API on MYRINET. However, the memory registration issues that we studied here should also concern several other network access layers used in clusters such as INFINIBAND or VIA. We plan to validate our work by implementing our optimizations in a mature parallel file system such as LUSTRE.

References

[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.

[Bra99] Peter Braam Braam. The InterMezzo File System, 1999.

[CLRT00] 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.

[Clu02] Cluster File Systems, Inc. Lustre: A Scalable, High Performance File System, November 2002. http://www.lustre.org.

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

[GP03] Brice Goglin and Loïc Prylli. Design and Implementation of ORFA. Technical Report TR2003-01, LIP, ENS Lyon, Lyon, France, September 2003.

[GP04a] 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.

[GP04b] 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)*, Las Vegas, Nevada, June 2004.

[ISSW97] M. Ibel, K. Schauser, C. Scheiman, and M. Weis. High-Performance Cluster Computing Using SCI. In *Hot Interconnects*, July 1997.

[MAF+02] K. Magoutis, S. Addetia, A. Fedorova, M. Seltzer, J. Chase, A. Gallatin, R. Kisley, R. Wickremesinghe, and E. Gabber. Structure and Performance of the Direct Access File System. In *Proceedings of USENIX 2002 Annual Technical Conference*, pages 1–14, Monterey, CA, June 2002.

[MAFS03] 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.

[Myr03] Myricom, Inc. GM: A message-passing system for Myrinet networks, 2003. http://www.myri.com/scs/GM-2/doc/html/.

[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.

[Pfi01] Gregory F. Pfister. Aspects of the InfiniBand Architecture. In *Proceedings of the 2001 IEEE International Conference on Cluster Computing*, Newport Beach, CA, October 2001.

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

[SASB99] 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.

[SGK+85] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and Implementation of the Sun Network Filesystem. In *Proceedings of the Summer 1985 USENIX Conference*, Berkeley, CA, 1985.

[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.

[TGL99] 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.

[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.

[vEBBV95] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *15th ACM Symposium on Operating Systems Principles (SOSP)*, pages 40–53, December 1995.

[WWP03] Jiesheng Wu, Pete Wyckoff, and Dhabaleswar Panda. Supporting Efficient Noncontiguous Access in PVFS over InfiniBand. Technical Report OSU-CISRC-05/03-TR, May 2003.