Performance Analysis of Remote File System Access over High Bandwidth Local Network

Brice Goglin, Loïc Prylli

Laboratoire de l'Informatique du Parallélisme

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

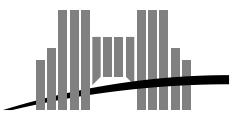
CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Performance Analysis of Remote File System Access over High Bandwidth Local Network

Brice Goglin,
Loïc Prylli

April 2003

Research Report N° 2003-22



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

INRIA

Performance Analysis of Remote File System Access over High Bandwidth Local Network

Brice Goglin, Loïc Prylli

April 2003

Abstract

We study the performance of file servers, comparing NFS implementation in Linux to our experimental lightweight system called ORFA. The aim is to find out NFS bottlenecks in the case of high bandwidth local network.

Using a simple protocol without cache allow us to get best performance from the underlying communication subsystem. Our user-level implementation avoids several kernel-level constraints and thus provides better performance than NFS. Moreover we explore several optimization techniques to reduce the server overhead which usually is the bottleneck.

Keywords: Distributed File System, Remote Access, High Bandwidth Local Network, Cluster, Linux, NFS

Résumé

On étudie les performances des serveurs de fichiers en comparant l'implémentation de NFS dans Linux avec notre système expérimental nommé ORFA. Le but est de déterminer les goulots d'étranglements de NFS dans le cas des réseaux locaux haute performance.

Utiliser un protocole simple sans cache nous permet de tirer profit des performances du système de communication sous-jacent. Notre implémentation au niveau utilisateur évite par ailleurs différentes contraintes du noyau et peut ainsi dépasser les performances de NFS. On explore de plus différentes techniques d'optimisation pour réduire la charge du serveur qui reste souvent le goulot d'étranglement.

Mots-clés: Système de fichiers distribué, accès distant, réseau local haut-débit, grappe, Linux, NFS

1 Introduction

This paper study issues in the performance of file servers, focusing on the cost of the remote file access protocol used as well as the underlying communication subsystem. We will focus on both the NFS[RFC95] system as implemented in Linux, and our own lightweight replacement which streamline performance by being more specialized than NFS, and removing the support for wide interoperability, network independence and fault-tolerance.

The goal of this paper is two-fold, first to determine what are the main bottlenecks of NFS, and then to evaluate the room for improvements through alternative designs. It has already been shown in the past that typical implementations of NFS have an heavy CPU usage, either when using old servers from the 1990s with 100 Mbit/s Ethernet or when using more recent servers with Gigabit-Ethernet, the saturation of CPU and other internal resources occurs frequently before the maximum network capacity of the server is attained.

For these purposes, we propose our functional replacement of NFS called ORFA (*Optimized Remote File system Access*), which shows what performance gains are possible by replacing a portable network stack like IP by some dedicated message passing system like BIP[Pry98] or GM[Myr00]. We also remove the constraint of supporting heterogeneous nodes. The ORFA system we use in this paper is implemented as a shared library overloading the system calls used for file access. For simplicity it does not implement any form of caching, and we will study it in benchmarks that do not benefits from caching. Extending this study for cache effects is beyond the context of this paper. We will mention the influence of different variants of operating systems techniques to handle the disk and network input/output.

A number of systems have been proposed in the context of clusters or wider networks to manages input/output to files. Most of the time they address high-bandwidth needs by distributing the data over different storage nodes. There are a number of open-source or commercial variants of these systems (PVFS [CLRT00, Lig01], GPFS[SH02], GFS[SRO96]), they can generally be used through the MPI-IO interface among others. Our work is quite distinct from these efforts, and focus on a general-purpose remote file access service dedicated to scalability in terms of clients through low CPU usage, with low latency through various specific implementation techniques. In theory such a remote file access protocol and implementation could be used to access a server relying on a distributed file system, like the ones mentioned above, to access the data, but this is not addressed in this paper.

This paper is to be put into the context of various efforts at analyzing, tuning, and optimizing NFS performance, such as [HSCL97], [Fab98] and [MC99]. By removing the constraint of compatibility with NFS, we can experiment further, and develop a quicker prototype. Also by specializing with high bandwidth, low-latency protocols over Myrinet such as BIP or GM, we can have ORFA benefit from the performance gain of these protocols over IP.

Several steps have been taken to isolate factors independent of the protocol in overall performance. While the ORFA protocol can be generally used to export a local file system to remote client machines, we have also implemented it on top of specialized memory file system.

The paper will be organized as follows, we will describe the architecture of NFS and ORFA in section 2, together with some variants, the architecture of the platforms on which we will use them. Then section 3 will focus on various benchmarks on our test platform while section 4 will detail optimizations that can improve remote file access performances. In section 5, we will discuss the future perspectives of this work, and finish by the conclusion.

2 Architecture Overview

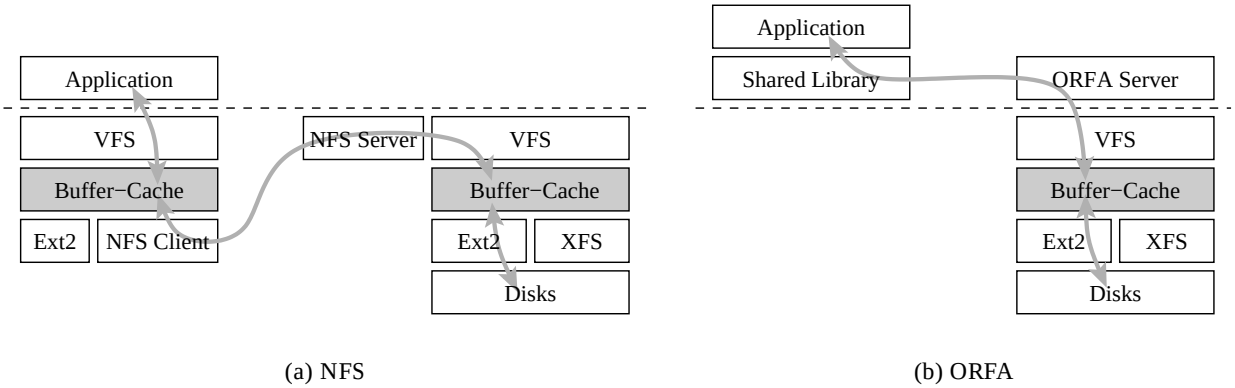


Figure 1: Schematic architectures of NFS and ORFA models. Grey parts are caching instances.

2.1 The Network File System

The Network File System is based on a client module which is considered as a new file system in the client host, and a server module on the server host which processes requests from the client and transmits them to the local file system. NFS requests are based on Remote Procedure Calls while the eXternal Data Representation standard (XDR) is used to deal with architecture dependent issues.

As described in figure 1(a), the NFS client is placed under the VFS layer. On Linux, NFS requests are generally based on pages or page-sized transfer even when for instance writing a large amount of data (a big write is split into several requests) or reading a single directory entry (a whole page of directory entries is then fetched). However NFS also benefits of the buffer cache advantages, especially caching and read-ahead. Moreover the NFS protocol try to reduce the amount of small requests by opportunistically updating the meta-data cache of the client (file attributes are updated by almost all requests concerning this file).

NFS servers are stateless. They are aware of what clients are doing only when processing their requests, but they do not maintain information about connections, open files and so on. Clients manipulate files using file handles associated to inodes on server's side. NFS version 2 clients used to update local modifications into the server synchronously (write-through). As it implied an important performance bottleneck, NFS version 3 now defers this update (write-back). This implies to use a new specific commit request when the client wants to ensure data has been written to disk.

Modern NFS server implementations are based on a NFS kernel server which avoid copying data between user and kernel space. Moreover some recent work in Linux even allow on the sending side to avoid the copying generally occurring at the interface with the network stack.

2.2 Optimized Remote File system Access

Three ways of exchanging file data between a server and a client can be distinguished :

Network Block Device gives access to a remote partition. Data are referenced by physical block in the remote device. This case is reserved for single client or read-only access because coherency is difficult to maintain across concurrent clients.

Network File System exports data as inodes and pages. The layer redundancy caused by buffer-caches in clients and servers either requires a complex protocol to maintain coherency or implies a non-concurrent model. Remote files are handled by the kernel VFS as local ones are. It provides automatic support to all Unix functionality.

User Level Access exports data as descriptor and offsets using a protocol close to the traditional Unix API: transforming these requests into remote commands. Clients are free to use cache assuming they are aware of the coherency implications.

Our system ORFA is in the third class, it is a user-level implementation which roughly provides the functionality of NFS in terms of transparency for applications with performance close to DAFS[MAF+02].

2.2.1 Work Hypotheses and Design Choices

Our main goal is to provide a high-performance access to remote file system in high bandwidth local networks or clusters. While several projects includes advanced caching protocols to reduce the server load and requests processing time, our idea is to first focus on performance in the case of non-repetitive access to data. Therefore we are not going to use any cache in the client and find the fastest way to transmit requests to the server. This model provides strong data coherency.

Our work will benefit of cluster network message latency which is small enough (less than 10 μs) that we may access a remote file system with response time not too far from a local access that hit the cache. In case of disk access, the network latency will be negligible compared to disk seek time. Cluster based architectures are also most of the time homogeneous. That is the reason why we avoid XDR-like layers whose important overhead[Fab98] would have been a bottleneck in our model.

In order to get a lightweight implementation avoiding useless overheads we designed a user level protocol. The ORFA architecture is summarized on figure 1(b).

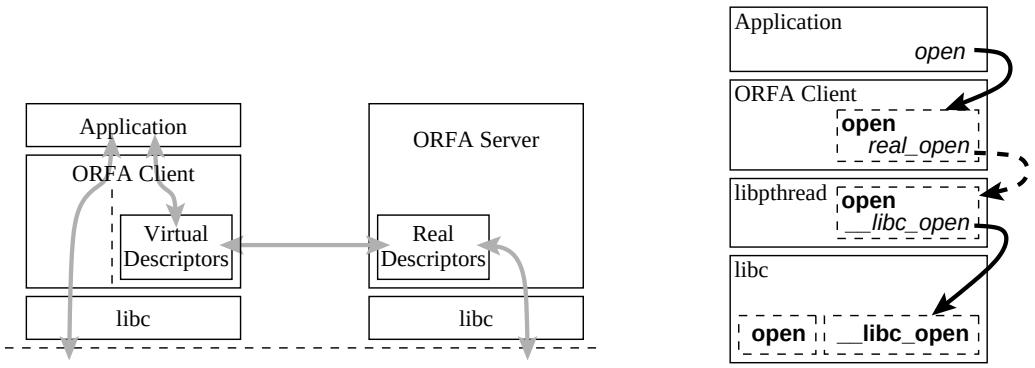
This work is close to work done in implementations of DAFS, except we do not take advantage of an advanced API, but focus on legacy applications.

2.2.2 Network Protocol

User level access to remote file system is based on the Unix file API which uses file descriptors. Therefore our network protocol is based on usual Unix calls (`open`, `write`, `fsync`, `readdir`, etc). Remote files are manipulated through virtual file descriptors which correspond to real file descriptor on server's side (contrary to NFS servers which are stateless, the ORFA server is not). This implies that in our case the client cannot survive a server failure or reboot.

2.2.3 Shared Library

Some other projects are based on a different API which requires to rewrite applications to take advantage of it. For example, DAFS introduces an original API which is assumed to fit needs of the class of applications that uses it[MAF+02].



(a) File descriptors handling

(b) Transparent I/O calls interception

Figure 2: Principles of ORFA shared client. Figure 2(a) shows how local and remote files are handled. Figure 2(b) presents the case of `open` interception when `libpthread` already replaces `libc` `open`. The dashed arrow shows dynamic link edition with the `dlfcn` library.

In our case, on modern unices, it is possible to avoid recompiling of legacy applications relying on POSIX I/O recompiling by using the `LD_PRELOAD` environment variable to force preloading of a shared library. Symbols of this library will be used instead of the standard libraries ones. This allow to intercept I/O and converts them into remote calls.

This method requires some work to keep `libc` behavior on local files. Other projects, as the `smbsh` Samba shell, generally emulate this by copying the original code into their implementation. This is a portability issue if the `libc` is modified. By the way, this prevents from supporting cases where another library overrides some `libc` functions (as `libpthread` does). The ORFA shared library supports transparently local file accesses by calling the original `libc` function which is found using the `dlfcn` library (see an example in figure 2(b)).

We also focussed on supporting Unix functionalities on remote calls. For now, almost all Unix I/O calls are supported, with proper behavior across `exec` and `fork`.

This allows shell scripts or any applications (even multi-threaded) to transparently manipulate remote files without being recompiled (see figure 2(a)). Needs for such a support has led most other projects to introduce kernel support[Lig01].

2.2.4 Kernel Support

Actually ORFA also provides the possibility to mount a remote file system in the kernel rather than using a shared library. Instead of porting the entire ORFA client as a kernel module, we based our kernel support on the FUSE project (File system in USEr-space) which allows to connect a file system implemented in a user program to the VFS layer. Mapping a remote file into a user program memory was the only function that was impossible to support with ORFA shared client while mounting a remote file system with FUSE gives automatic kernel support for all Unix functionality.

The main problem is that the path from the client to the server goes into the FUSE kernel module and then back in user space in the ORFA-FUSE client which communicates to the server. This

implies an important latency and limits the bandwidth in the client because of the two memory copies. These added costs only have an impact on the client-side, so it is still useful for situations where the shared-library approach is not suitable, but where we want to increase the scalability of the server-side. Anyway, all the following tests have been done with the ORFA shared-library client.

3 Performances and Comparison

3.1 Test Platform

The test platform is composed of 24 bi-Xeon 2.6 GHz with 2 GB of RAM and PCI-X busses. These nodes are connected through a Myrinet 2000 network (Lanai 9.3 at 200 MHz) which provides a 250 MB/s full-duplex bandwidth and a 7 μs one-way latency with Myricom GM driver (3 μs with BIP).

The cluster runs Linux 2.4.20 kernels with NFSv3 kernel servers. Even if high-performance hard drives are available is these nodes, we chose our working set in order to keep every data in the buffer-cache. It allow us to get more meaningful results by avoiding dealing with performance issues related to disks accesses on the server. NFS client caches where emptied before each test.

One node has been dedicated to be alternatively a NFS or ORFA server while the remaining 23 nodes where running until 8 clients at the same time. The ORFA client was always our shared library which is much more efficient (see section 2.2.4).

We removed the Fast-Ethernet network bottleneck by only running NFS on the IP layer of Myricom GM driver. ORFA is also able to use native Myrinet API, such as GM or BIP.

Even if BIP gets lower latencies than GM and is a better platform for implementing optimization (see section 4), it still has some scalability troubles with many clients per node. Therefore we only present ORFA performance on GM and TCP/GM and GM. BIP performance improvements will be published later.

Finally, some performance tests also show the case where the ORFA server runs in memory. This allow to measure the overhead of copying data between user and kernel space and handling I/O system calls.

3.2 Performance Evaluation Methodology

Several NFS benchmark have been widely used and are known to be interesting performance measurement methods. For example SPECsfs[Sta97] has been used in [MC99] to explore NFS behavior in different network configurations. Some other work as [HSCL97] are much more intrusive in order to find out NFS bottlenecks.

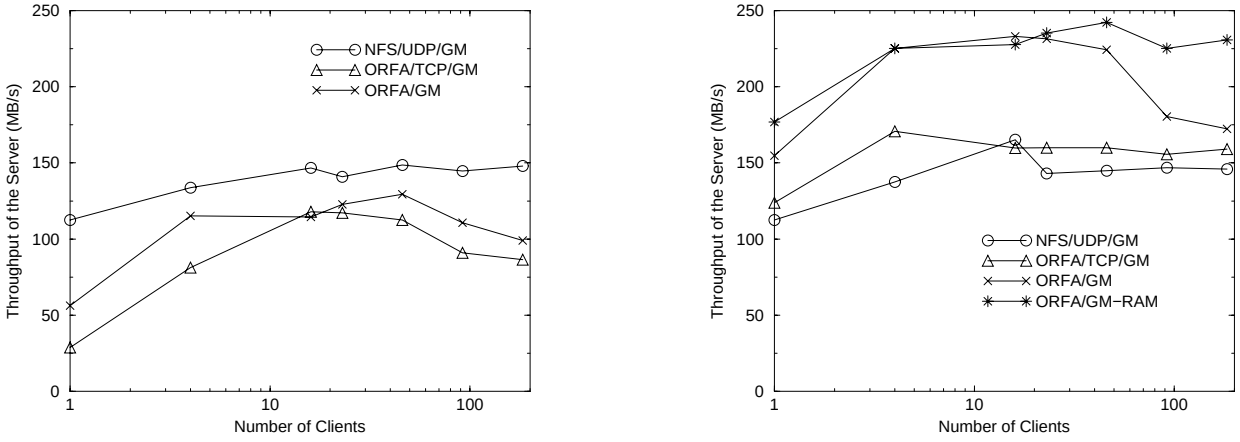
As ORFA uses a user level access, its API was first designed to be similar to the Unix I/O API. That is the reason why its protocol is distinct from NFS one. Thus it prevents us from benchmarks which are based on direct use of the NFS protocol rather than through the use of an application on a NFS client.

Some previous works such as the NFSstone[SCW89] benchmark and [Fab98] have studied NFS performance at user level. This provides real user performance but prevents from analyzing and tuning internal parameters.

We focussed on analyzing user level performance by isolating different kinds of traffic, that are meta-data and real data. This will show the pros and the cons of NFS page-oriented protocol which groups lots of meta-data requests and splits data access in pages accesses. Therefore we

are going to present performances as quantities of processed data, that are throughput or files processing rate.

3.3 Read requests



(a) 4 kB per read request

(b) 64 kB per read request

Figure 3: Throughput of the server when clients are reading fixed size chunks of huge files (from 1 to 184 clients).

Our first test is supposed to measure the throughput of the server when lots of clients are reading big chunks of long files. Results are shown in figure 3. We observe that NFS throughput does not vary when the chunk size increases. This is due to its inherent splitting into pages. The throughput remains far from the network capability (150 against 250 MB/s) even if 184 clients are simultaneously reading.

When using ORFA, read requests are handled without being split into pages. The throughput also vary with the chunk size. We have observed that ORFA is equivalent to NFS when reading about 8 kB per requests (2 pages). This may be due to NFS using read-ahead for the next page when reading the first one. The case of 64 kB chunks uses 90 % of the network bandwidth when using native GM API while ORFA on TCP/GM remains near NFS performance.

Thinking about the read request size that parallel computing applications may need, these results show that ORFA/GM can increase their read performance by more than 50 %.

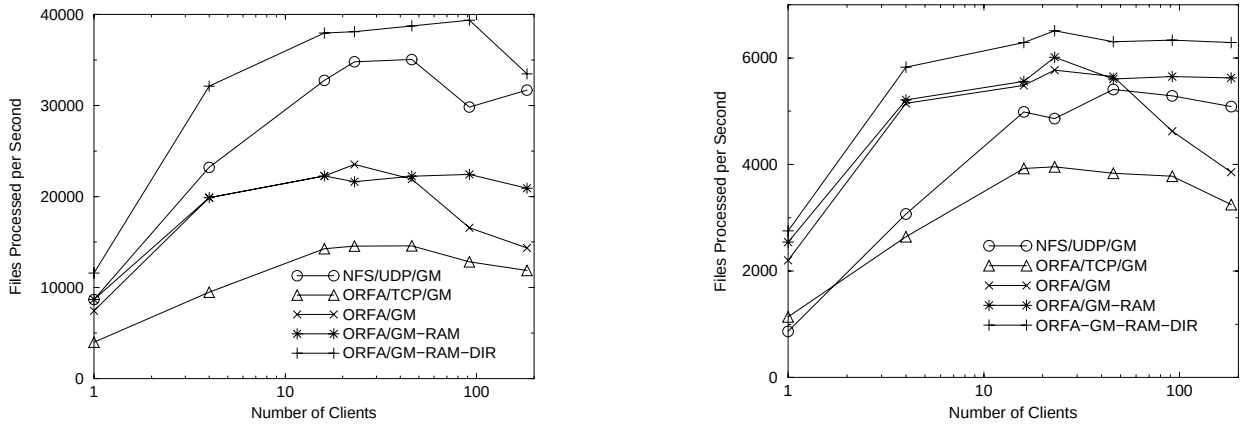
Comparing native and memory file system shows the impact of reading file data from the buffer-cache in the server. The graph shows that hundreds of clients can saturate the disk server (even if all data was already in the buffer-cache), while a memory file system keeps maximal throughput by avoiding this copy. Avoiding copies will be more discussed in section 4.2.

3.4 Scanning a tree and handling small requests

To compare the processing rate of small requests in different cases, we first present on table 1 the observed latency of a single `stat` request. ORFA provides an attributes access time between four and seven times lower than NFS in this case.

NFS / UDP / GM	ORFA / TCP / GM	ORFA / GM	ORFA / GM-RAM
342 μs	86 μs	57 μs	47 μs

Table 1: Latency of `stat` requests.



(a) Stat each file of a tree

(b) Stat and read each file

Figure 4: Impact of small requests on the processing rate of the server. The RAM-DIR graph shows performance when using a RAM file system and optimizing directory handling by getting several dirents at once.

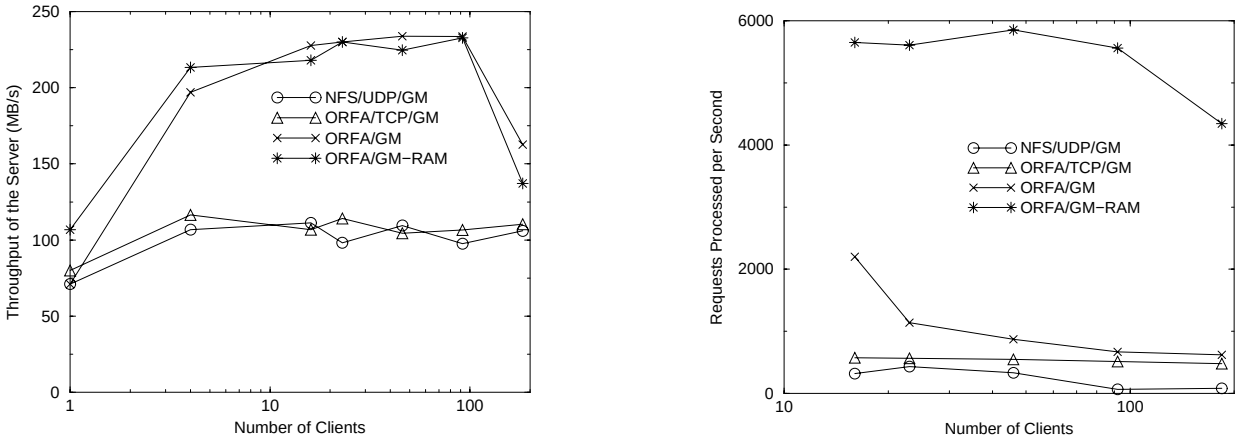
The second test concerns a huge tree of small files (a few pages per file). We measured the time needed to scan each file of the tree (this includes recursively getting directory entries and then obtaining their statistics). Indeed this evaluates how much small requests a server can handle.

This test shows NFS benefits of manipulating directory pages. Moreover, the `READDIRPLUS` request of NFS protocol allows it to get lots of entry at once and their statistics at the same time[RFC95]. The user level ORFA protocol requires one `readdir` request and then one `stat` request for each directory entry. We can estimate that ORFA needs about 32 requests of each type while NFS needs only one `READDIRPLUS`. We can consider that NFS makes up for its huge latency by benefiting of an efficient page oriented protocol. Thus performance results on figure 4(a) show that ORFA remains 40 % lower than NFS.

When optimizing ORFA to use read-ahead when getting directory entries (the client now gets several entries at once but still needs to stat all of them later), ORFA can pass NFS. This is coherent with table 1, NFS implies a huge request overhead (its protocol tries to minimize the amount of small requests to the server).

Moreover, figure 4(b) shows that NFS almost loose its scanning performance when files are scanned and read. These files are a few pages long. Previous results (see 3.3) proved that ORFA and NFS have same performance when reading such files.

3.5 Concurrent Access



(a) All clients writing 64 kB chunks

(b) Concurrent R / W accesses

Figure 5: Write throughput of the server, and impact of concurrent Read / Write accesses on processing rate.

The last two tests study the performance of write accesses and their impact on other requests. Figure 5(a) proves that ORFA gets same throughput (about 90 % of network bandwidth) when writing big chunks than when reading. NFS performance is about 30 % lower than its read performance. The memory file system does not increase performance because it does not avoid a copy when writing.

Figure 5(b) shows the impact of concurrent read / write accesses. The ORFA server is about five times faster than NFS to process such requests. This reveals the important overhead imposed by

the NFS protocol when not able to cache meta-data. You have to remember that we are using the NFSv3 Linux implementation. This revision introduced write-back caching in order to remove the performance bottleneck of the write-through in previous revisions.

You may also note that our memory file system shows that the overhead needed to transfer data between user-space and the buffer-cache is an important limitation here too.

The first graph seems to show ORFA / GM saturation with 184 clients. The reason is that the server cannot post all GM receive buffers that are needed. Thus about one third of clients waste network bandwidth by sending chunk data which will not be received. This problem will be discussed in section 4.1.

4 Performance Issues

4.1 Benefiting from Cluster Network API

Previous results show that using the IP layer on cluster network gives performances far from their native API. Implementing a remote file system access on this kind of API requires a precise study of the network traffic.

The fact that the server concentrates lots of traffic is far from the usual need of parallel computing applications which homogenize the traffic all over the network. As parallel file systems often distribute the workload across different servers, we may use several network interfaces to reduce the network bottleneck.

We focussed on keeping only one interface and studying the network traffic. The main problem we met is the fact that any asynchronous API requires receive buffers to be provided before the message arrives. If no receive has been posted, the message will be lost and has to be resent. A part of the network bandwidth has been wasted (for instance about one third of bandwidth is wasted on the last point of figure 5(a)).

Therefore it is important to ensure that enough receive have been posted before allowing lots of clients to send requests and their associated data. This could be an important limitation with several hundreds of clients because receive posting is limited by the network interface memory. Another solution could be to waste a message allowing a client to send data when the associated request has been handled and the needed receive posted.

4.2 Reducing Copies

Test Platform	kB limit	4 kB pages limit
Pentium 2 350 MHz	150-250	$\simeq 50$
bi-Pentium 2 450 MHz	150-200	$\simeq 45$
bi-Pentium 3 1.26 GHz	$\simeq 200$	$\simeq 50$
bi-Athlon 1.2 GHz	$\simeq 400$	$\simeq 100$
bi-Xeon 2.6 GHz	$\simeq 500$	$\simeq 125$
quadri-Pentium 3 550 MHz	$\simeq 130$	$\simeq 32$

Table 2: Evaluation of the amount of data which makes `mmap` and `read` have the same overhead. `mmap` has almost constant overhead and thus is more efficient for big chunks while `read` has linear overhead and is the fastest for small chunks.

The easiest way to avoid a copy when sending data from a file to the network is to map the file in the process address space. We measured the overhead of the `mmap` syscall on Linux and observed it was equivalent to copy more than 100 kB of data using `read`. The `mmap` overhead is due to the TLB flush, especially when using multiprocessor systems (see table 2). Thus mapping a file could be interesting for large requests, especially if several requests can be processed with the same mapping.

The other way to avoid this copy is based on the `sendfile` syscall that was introduced to optimize data transferring from files to sockets in web servers (it avoids a copy in user space). While `sendfile` is included in standard Linux kernels to send file data on sockets, it is usually not supported by cluster network API. No implementation is available for GM. Measurements over TCP/GM showed a 30% performance enhancement for large requests (hundreds of kilobytes). However no equivalent `recvfile` support has been announced.

DAFS (*Direct Access File System*) which has similar goals to ours is based on a server which is entirely in the BSD kernel. This removes lots of overhead by giving fast access to all data structures [Mag02]. However kernel implementation requires strong development effort.

For now our choice is to keep the server at user level and improve cluster network API integration into the kernel and its file system structure to provide better functionality to the server. That is what we have done by implementing `sendfile` and the opposite `recvfile` in BIP.

4.3 Events Handling

`epoll` is the scalable alternative to the standard poll/select strategy in Linux kernels. Its overhead is linear with the number of reported events instead of the number of scanned sources for `poll`/`select`. `epoll` was first aimed to speed-up web server that handle thousands of high latency connections.

Using `epoll` with low latency connections make the ratio of active connections bigger and also reduces the difference between the number of events and the number of sources. As `epoll` is lower than `poll` when lots of sources are active, we cannot expect `epoll` to imply a huge performance enhancement. However we observed a 10% performance enhancement when using 184 clients on TCP/GM.

Using threads for events handling remains difficult when high performance are needed. However a thread pool could exploit multiprocessor system without to much context switches.

4.4 Asynchronous Input/Output

Linux AIO (*Asynchronous Input/Output*) aims at providing an asynchronous model for all I/O in Linux, based on asynchronous primitives and a event queue for notification of completion. This will allow easy implementation dealing with network and files.

First implementations are available and show that asynchronous large file requests provides 25% of performance gain on a highly loaded bi-processor server. Asynchronous I/O benefits from multiprocessor system by delegating deferred requests to other processors. Thus using threads is not needed to fully use multiprocessor systems.

The performance gain is only seen for huge requests because current Linux 2.4 AIO implementation splits requests into chunks of 128 kB and processes the first chunk synchronously.

5 Perspectives

5.1 Dedicated File System in the Server

Our memory file system gives almost always best performance with ORFA because of its user level implementation which reduces overhead and naturally removes some copies. This led us to the idea of porting a similar system on a physical disk partition which would be dedicated to our ORFA server. Being dedicated will allow our server to avoid all the kernel overhead that is useless without multiple accesses, while disk storage removes the persistency issue of our memory file system.

5.2 Protocol Enhancement

First performance measurements show that ORFA does not improve meta-data processing due to its necessity to contact the server for almost all client I/O request. Even if the network latency allows very fast request exchanging, the amount of work may be too large for the server.

Small optimizations already showed in figure 4 that this could be enhanced by using read-ahead when getting directory entries. We are going to explore meta-data caching in order to reduce the amount meta-data requests. The aim will be to find a compromise between the Unix API ORFA already uses and the one in NFS protocol.

This could also lead us to move some parts of our clients into the kernel. We saw in 3.4 that ORFA has a very small latency but has to deal with much more requests than NFS. This could be improved by using FUSE (see 2.2.4) to handle meta-data in the kernel while real data will remain at user level.

5.3 Improving cluster network API

We saw in section 4.1 that cluster network are not necessarily well adapted for remote file access because of its inherent traffic centralization. We are going to enhance our Myrinet interface BIP in order to support this kind of traffic and improve its scalability.

Other work will focus on increasing copy avoidance between storage and network. We saw in 4.2 that we are now able to send or receive data between the network and the buffer-cache of the server with BIP. We may also handle `O_DIRECT` file by transferring data between the network interface and disks as OPIOM does[Geo02].

Finally we are working on interoperability of cluster network API and Unix standard API. All cluster network hardware has its own API and they usually are far from the standard Unix API to submit asynchronous requests or get events. We are trying to move BIP API into the standard `poll` model in order to be able to use it in `epoll` and in the future unified I/O system, Linux AIO.

6 Conclusion

We exposed performance measurement for NFS and ORFA accesses with hundreds of clients. NFS benefits of VFS and buffer-cache optimizations to reduce the amount of small requests but has to split huge requests into pages. On the other hand, our lightweight access protocol increases huge requests performance by saturating the high performance network link of the server while its very small latency cannot hide its need to contact the server for each request. Our transparent user-level

access supports almost all Unix functionalities allowing any applications to use it without being recompiled.

Optimization techniques such as Linux AIO and `sendfile` give ORFA better performance. Directory handling tests also led us to the idea of looking for a compromise between our protocol and NFS one to reduce the amount of small requests and thus increase meta-data access performance.

ORFA already provides an efficient enough remote access to replace NFS for big clusters (its scalability in number of clients for a server allows more flexibility in the configuration of the system). It can be used for massively parallel applications which require big data movements and very few meta-data requests, and where the total volume of I/O throughput can be handled by one node once you removed the unecessary overheads. It also could be used as an underlying remote access protocol for leaf nodes in a distributed system to link them with storage nodes implementing the file system to get the best performance of the underlying communication subsystem.

References

[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.

[Fab98] Theodore Faber. Optimizing Throughput in a Workstation-based Network File System over a High Bandwidth Local Area Network. In *ACM SIGOPS Operating System Review*, volume 32(1), pages 29–40, January 1998.

[Geo02] Patrick Geoffray. OPIOM: Off-Processor I/O with Myrinet. *Future Generation Computer Systems*, 18(4):491–500, 2002.

[HSCL97] J. Hall, R. Sabatino, S. Crosby, and I. Leslie. Counting the Cycles: A Comparative Study of NFS Performance over High Speed Networks. In *Proceedings of the 1997 IEEE Conference on Local Computer Network (LCN 1997)*, volume 22, pages 8–19, Minneapolis, MN, November 1997. 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.

[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.

[Mag02] 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.

[MC99] Richard P. Martin and David E. Culler. NFS Sensitivity to High Performance Networks. In *Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, Atlanta, GE, 1999.

[Myr00] Myricom, Inc. The GM Message Passing System, July 2000. http://www.myri.com/scs/GM/doc/gm_toc.html.

[Pry98] Loïc Prylli. BIP Messages User Manual for BIP 0.94, June 1998. http://www.ens-lyon.fr/LIP/RESAM/bip.html.

[RFC95] RFC 1813. NFS Version 3 Protocol Specification, June 1995.

[SCW89] B. Shein, M. Callahan, and P. Woodbury. NFSSTONE - A Network File Server Performance Benchmark. In *Proceedings of the 1989 USENIX Summer Conference*, pages 269–274, Baltimore, MD, June 1989.

[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.

[Sta97] Standard Performance Evaluation Corp. SPEC SFS97 Benchmarks, 1997. http://www.spec.org/sfs97.