Adaptive I/O Scheduling for Distributed Multi-applications Environments [*]

Adrien Lebre, Yves Denneulin, Guillaume Huard
Laboratoire Informatique et Distribution - IMAG
Montbonnot St Martin,France
firstname.lastname@imag.fr

Przemyslaw Sowa
Institute of Computer and Information Sciences
Czestochowa University pf Technology, Poland
sowa@icis.pcz.pl

Abstract

The aIOLi *project aims at optimizing the I/O accesses within the cluster by providing a simple POSIX API, thus avoiding the constraints to use a dedicated parallel I/O library.*

This paper introduces an extension of aIOLi *to address the issue of disjoint accesses generated by different concurrent applications in a cluster. In such a context, performance, fairness and response time are the criteria for which good tradeoffs have to be assessed. A test composed of two concurrent IOR benchmarks showed improvements on read accesses by a factor ranging from 3.5 to 35 with POSIX calls and from 3.3 to 5 with ROMIO.*

1 Overview

I/O bottlenecks have always been a major issue in Computer Science , it is likely to continue as I/O hardware performance increases slower than the one of CPU and memory. Furthermore, this gap is amplified by the increasing use of clusters of workstations or SMPs as well as the growing number of scientific application developments with more demanding I/O requirements using different access patterns.

However, contrary to "database accesses" which depend on the selection criteria and are usually more sparse, several studies [2, 5] have characterized parallel I/O accesses and have observed recurrent determined patterns based on stride parameters. For instance, in a parallel matrix product, each process has to access specific parts of the matrices according to the array-distributions used (BLOCK/BLOCK, BLOCK/CYCLIC, etc.). Unfortunately, even if these patterns are known, their interleaving dramatically increase the response time and affect the overall performance of the storage sub-system: remote file server receives several read/write requests on different offsets and sizes at the same time which potentially generate many disk head movements, one of the most time consuming operations in modern computers (approximately 9ms).

As a consequence, a lot of researchers have attempted to develop new I/O sub-systems that take these parallel computing accesses into account. Proposed solutions to reduce congestion issues and improve I/O access performance can be divided in two major categories: *Parallel File systems* and *Parallel I/O Libraries*. The former, [1, 6, 7], tries to find the best tradeoff between distributed file system constraints (coherency, fault tolerance, remote accesses, etc.) and efficient exploitation of hardware capabilities. Library solutions are focused on parallel I/O aspects and portability constraints. ROMIO, an implementation of the MPI I/O standard, is the best known; it includes two main algorithms: *data sieving* and *two-phase* [8].

Nevertheless, these systems often have too many features, which complicates development and the maintainability of scientific applications. Moreover, they require deep knowledge of specific APIs and model subtleties in order to reach good performance. These aspects have been emphasized in our previous work [3], in which we reported several inefficiencies in the classical way to deal with remote I/O accesses in a parallel HPC context and suggested an approach to favor sequential access which is known as always being more efficient [4].

In this paper, we focus on multiple concurrent parallel I/O applications striving for access to the storage system. Indeed, when programs exploit parallel I/O libraries to enhance performances, all provided optimizations deal with their own requests only. Thus, collective operations, such as *two-phases*, do not take I/O requirements from other concurrent applications into account. This leads to performance degradations due to conflicting individual I/O optimizations. In such a context, the storage system, local or remote, is the only one able to deal with requests in a global, multi-applications, manner. Unfortunately, most of existing storage systems do not provide appropriate I/O scheduling strategies. A naive approach to proceed the batch of waiting requests consists in dealing with them on a per program basis. It provides a good throughput by starving the other applications. Such a policy does not take into account the response time and fairness criteria which are mandatory in a multiple applications environment. We suggest to design an infrastructure made of two major elements: a transpar-

[*]This work has been done within the ID laboratory jointly supported by CNRS, INPG, INRIA, and UJF and the project LIPS between INRIA and BULL Lab.

ent parallel I/O aggregation mechanism and a scheduling algorithm to deal with intensive I/O applications in a HPC context. Our main contribution is to connect these two elements to make them work together to reach significant performance improvements. They are integrated into a generic framework pluggable into any I/O system. Unlike our former work, it only interacts with the I/O layers of distributed file systems (on client or server or on both sides) and has no direct interactions with the clients.

2 Experiments

We implemented our proposal as a Linux module and evaluated it on a Network File System server. Even if NFS is not really suited to high performance I/O, it remains the standard configuration for small and medium sized clusters.

Our testing system is a part of the grid "grid5000"[1] located at the INRIA Sophia-Antipolis site. A dedicated NFS server (version 3, TCP, 32Kb read size, cache disabled) above an ext3 file system and several SMP nodes have been exploited. We executed two instances of the IOR benchmark [2] to evaluate real I/O intensive HPC applications, each deployed on 16 nodes and retrieving a 4 GBytes file.

As expected, the execution of two I/O intensive applications impacts on performance (figure 1) but *aIOLi* minimizes this phenomenon. We can see that even if the collective MPI I/O performances show a light improvement for small granularities, they quickly reach the POSIX ones for granularities greater than 128Kb. Moreover, even if the MPI I/O collective approach is improved by *aIOLi* (MPI I/O + aIOLi curve), the standard API POSIX under *aIOLi* provides the best performance (since *aIOLi* doesn't require any synchronization mechanisms). For the smallest granularity, Posix requires more than 1 hour and half, MPI IO more than 11 minutes whereas *aIOLi* only takes 2 minutes and half.

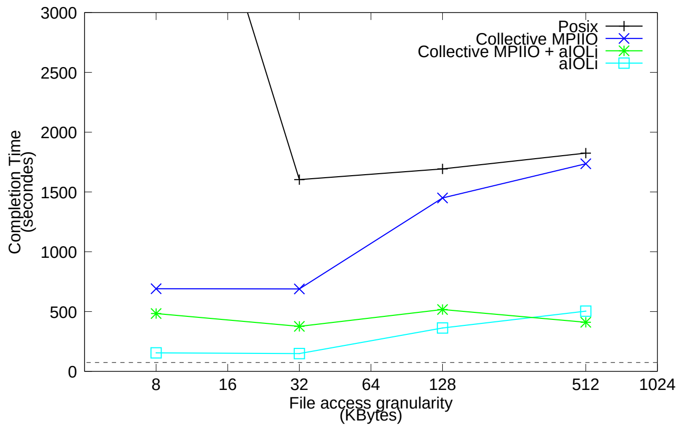

Figure 1. 2 IOR - read evaluation
2*4GB file decomposition / **2*32** MPI instances on **2*16 nodes**.

[1] `http://www.grid5000.fr/`.
[2] Input/Output Stress benchmark from the LLNL,

Regarding the fairness (table 1), the 2 IOR benchmarks have been launched at the same time: at worst, the gap between the two completion time is 8.5 seconds for 8Kb but with a 35 times improvement for POSIX and near 5 for MPI I/O. By choosing more specific scheduling params for each application, it is possible to set up a particular quality of service for each parallel applications running on the cluster.

	File access granularity			
	8kB	32kB	128KB	512KB
Posix	0	6	0	0
MPI I/O	1	1	2.5	0
aIOLi	8.5	5	5	7
MPI I/O+aIOLi	1.5	1	4.5	2

Table 1. Read - completion time gap (in sec)

3 Conclusion

We introduced *aIOLi*, a framework for high performance file access in distributed multi-applications environments. Preliminary results show that our approach dramatically improves read performances. Furthermore, as expected, our solution maintains fairness between applications and does not significantly degrade interactivity. We currently study how *aIOLi* could be connected to a parallel file system as Lustre.

Future work will include *cascading scheduling*, the integration of specialized schedulers at different levels in the I/O resolving process.

References

[1] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A parallel file system for linux clusters. *Proceedings of the 4th Annual Linux Showcase and Conference*, GA, October 2000.

[2] P. E. Crandall, R. A. Aydt, A. A. Chien, and D. A. Reed. Input/output characteristics of scalable parallel applications. *Proceedings of Supercomputing '95*, CA, December 1995.

[3] A. Lebre and Y. Denneulin. aioli: An input/output library for cluster of smp. *Proceeding of the 5th International Symposium on Cluster Computing and Grid, UK*, May 2005.

[4] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A fast file system for UNIX. *Computer Systems*, 1984.

[5] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. S. Ellis, and M. Best. File-access characteristics of parallel scientific workloads. *IEEE Transactions on Parallel and Distributed Systems*, October 1996.

[6] R. L. F. B. Schmuck. Gpfs: A shared-disk file system for large computing clusters. *Proceedings of the 5th Conference on File and Storage Technologies*, January 2002.

[7] P. Schwan. Lustre : Building a file system for 1,000-node clusters. *Proceedings of the Linux Symposium, Ottawa*, July 2003.

[8] R. Thakur, W. Gropp, and E. Lusk. Data sieving and collective I/O in ROMIO. *Proceedings of the Seventh Symposium on the Frontiers of Massively Parallel Computation*, February 1999.