Current aIOLi evaluations
Impact of an Intensive I/O application on a less intensive one *

Adrien Lebre
Laboratoire Informatique et Distribution - IMAG
Montbonnot Saint-Martin, France
Adrien.Lebre@imag.fr

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

May 6, 2006

Abstract

The *aIOLi* project aims at optimizing the I/O accesses within the cluster and providing a simple POSIX API. Currently, we are working on an extension of *aIOLi* to address the issue of disjoint accesses generated by different concurrent applications in a cluster. In such a context, good trade-off has to be assessed between performance, fairness and response time. To achieve this, an I/O scheduling algorithm together with a «*requests aggregator*» that take into account both application access patterns and global system load, have been designed and merged into *aIOLi*. This improvement led to the implementation of a new generic framework pluggable into any I/O file system layer. This short paper presents an experimentation done to study the impact of a parallel I/O intensive application on a non I/O intensive program started during its execution.

Discussion

The parallel I/O intensive applications consist in one IOR instance deployed on 32 single processor nodes. The IOR Stress Benchmark Code is a parallel file system program developed by the Scalable I/O project at Lawrence Livermore National Laboratory. This parallel program performs parallel writes and reads to/from a file using the POSIX or MPI-IO API and reports the throughput rates. The file size has been set to 4GBytes and the file access granularity ranges from 8KBytes to

*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. Computer resources are provided by the g5k cluster (further information at `http://www.grid5000.fr/`).

4MBytes. For this application, there are two cases: in case one, Posix calls are used for accesses and in the second case, MPI-I/O is used for accesses.

The non I/O intensive program is a `cat`-like program retrieving sequentially a 16MB file on a single node using granularities ranging from 8KB to 512KB. This latter application is small (run alone, it completes in roughly 2s) and is started during the execution of the first one (15s after its beginning). For this application, Posix is always used for accesses. The platform configuration consists in a Linux NFS server exporting an ext3 file-system stored on a single IDE disk (57MB/s peak)[1]. Due to the large difference between the completion times, a log scale for the Y axis has been used. The figure presents the results of this test.

An usual NFS server (on the left) is compared to an NFS server plugged with our aIOLi module. Whatever the granularity exploited for the `cat`-like program, curves follow the same forms. As we can see, *aIOLi* improves the efficiency for both applications. For smaller granularities and due to the adaptive window, the IOR application benefits more than the `cat` program. As expected, the adaptive window becomes more efficient for sequential accesses when large granularities are exploited.

For case 2, the parallel I/O intensive application has been launched with the MPI I/O optimisations (the `cat` program was still based on the POSIX API). First, we can note that *aIOLi* under POSIX provides better performance than the MPI I/O under usual NFS server for the IOR benchmark. Moreover, as we expected, the MPI I/O routines reduce congestion issues on the NFS server side which lead to better performance for the `cat` program. However, when we compare the MPI I/O performance under NFS and NFS+aIOLi, we can see that *aIOLi* does not benefit from the MPI I/O optimization. Indeed the synchronisation overhead implied by MPI I/O is added to *aIOLi* performance.

[1]Further details are avalaible on the Sophia-Antipolis part of `https://www.grid5000.fr`

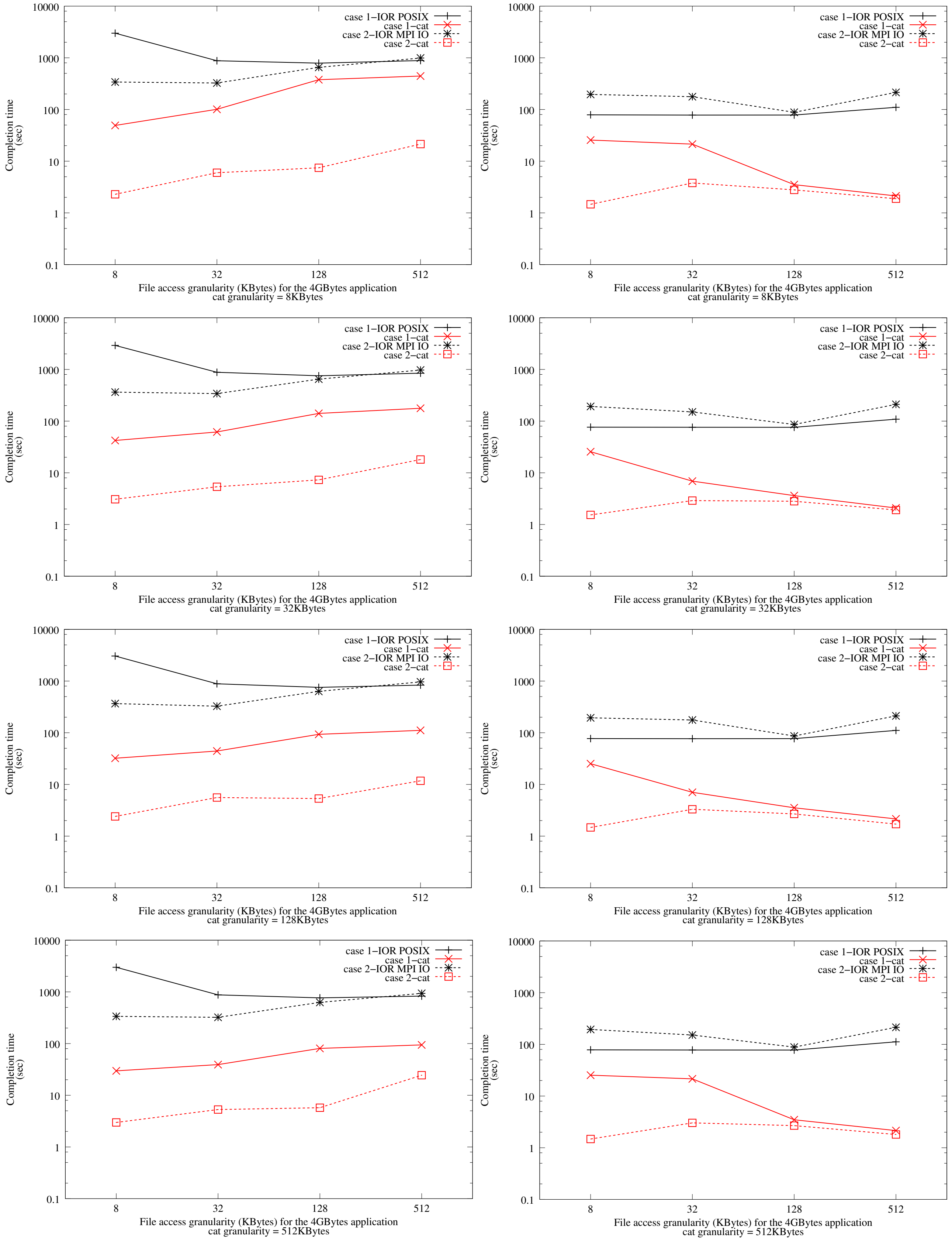

Figure 1: The impacts of an intensive I/O application on a lesser one

Traditional NFS server (left) and a NFS server plugged with aIOLi (right)