Current aIOLi evaluations
Impact of the Linux low schedulers on a NFS server in presence of an Intensive I/O parallel application [*]

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 the results obtained for the evaluation of the different low level schedulers in Linux under an NFS server in presence of an I/O intensive parallel application.

Discussion

Disk scheduling algorithms, because they are all implemented at a low level, cannot have a good overview of distributed applications accessing the file system. Besides being limited by the size for their queues, they are strictly dependent on the implementation of the upper file system: for instance, if the file system is synchronous and mono-threaded, only one request can be handled at

[*]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/`).

the same time which limits potential optimizations. As a direct consequence, such approaches are not suited to large workloads generated by HPC Intensive I/O applications. That we shew in this experiment.

We checked aggregation capabilities of these schedulers using the IOR benchmark[1]. 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 experiments consists in evaluating IOR over 32 MPI instances decomposing a 4GB file on a Linux NFS server exporting an ext3 file-system stored on a single IDE disk (57MB/s peak)[2].

Test have been performed on each Linux I/O low level scheduler, making the file access granularity vary from 8KB to 4096KB and by using POSIX API. To provide more aggregation opportunities to the scheduler, the number of active `nfsd` daemons ranged from 8 (default configuration) to 512 (more incoming simultaneous requests). The results are similar whatever the I/O scheduler in use.

As we can notice, the greater the number of daemons, the better the performances are. Nevertheless, even with 512 daemons, the low-level scheduler performance is far from sequential performance (around 50MB/sec) in most cases.

[1] `http://www.llnl.gov/asci/purple/benchmarks/limited/ior/`

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

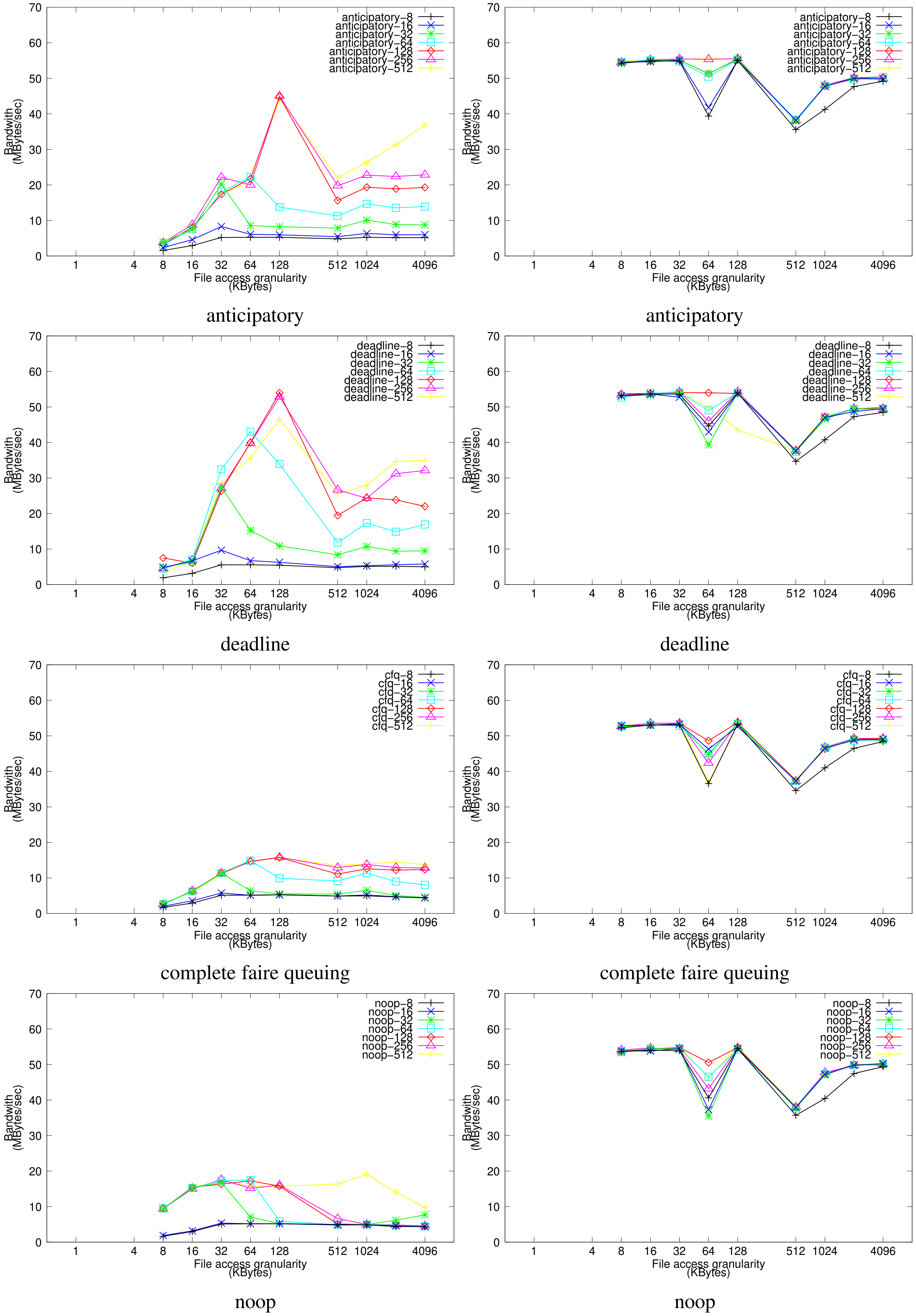

Figure 1: low scheduler impacts on a traditional NFS server and a NFS server plugged with aIOLi
4 GBytes file decomposition (IOR benchmark) on 32 MPI instances deployed on 32 nodes.
The purpose of this evaluation is to observe the impact of the low schedulers. On the right, each Linux scheduler is tested on a traditional NFS server with a varying number of `nfsd` dameon growing from 8 to 512. On the left, the same experiments are done but with a NFS server plugged to *aIOLi*. But for some erratic points, *aIOLi* is far better and have still the same performance whatever the low scheduler.