AGIOS: Application-guided I/O Scheduler

Francieli Zanon Boito[1,2], Rodrigo Virote Kassick[1,2], Philippe O. A. Navaux[1], Yves Denneulin[2]
[1]Institute of Informatics – Federal University of Rio Grande do Sul – Porto Alegre, Brazil
{fzboito, rvkassick, navaux}@inf.ufrgs.br
[2]LIG Laboratory – INRIA – University of Grenoble – Grenoble, France
{yves.denneulin}@imag.fr

Abstract

In this paper, we improve server-side I/O scheduling on parallel file systems by including information about the applications' future accesses. Our approach shows performance improvements of 46.3% *on average when compared to a scenario without an I/O scheduler.*

1. Introduction

When multiple applications concurrently access the same parallel file system, their requests can arrive interleaved in the servers. In this situation, even if each application's individual access pattern is ideal, they will perform poorly because the server will not observe the same contiguous access pattern as the applications. I/O scheduling techniques are used to alleviate this problem.

We developed an I/O scheduling library named *AGIOS*. Our library can be easily integrated into a PFS's server code to manage incoming I/O requests using a variation of the MLF algorithm [4], improving performance through requests reordering and aggregation.

We improve the previously proposed scheduling algorithm by including information on future accesses via trace files. This information is used to guide the scheduler's decisions during execution. Our new approach led to performance improvements of up to 35.4% over the base algorithm. To the best of our knowledge, there is no other I/O scheduling technique which uses such information to improve its performance.

2. Application-guided Scheduling

We developed a new module for the library named *Prediction Module*. This module obtains information about the application through trace files, generated by the scheduler itself without modifications to the application or to the file system. The prediction module is initialized by the scheduler when trace files are present. A thread then reads the traces and generates a set of predicted future requests.

Having this list of future requests, the module then evaluates all possible aggregations. For every **predicted request** $R_\mathbf{i}$, we can obtain its **time of arrival** $AT_\mathbf{i}$ and its **size** $S_\mathbf{i}$. We also estimate, by micro-benchmarking, a **time to process function** $TTP(x)$ for x being a request size.

If contiguous predicted requests R_1 and R_2 are observed, the scheduler evaluates their aggregation considering that it should aggregate them when the time to process them separately is big enough (when comparing with the time to process them aggregated) to make it worth keeping the first request in queue. The ability to overlap this waiting time with processing requests to other files is represented by the α factor.

Therefore, R1 and R2 should be aggregated if:

$$TTP(S_1)+TTP(S_2) > TTP(S_1+S_2)+(|AT_2-AT_1|)*\alpha$$

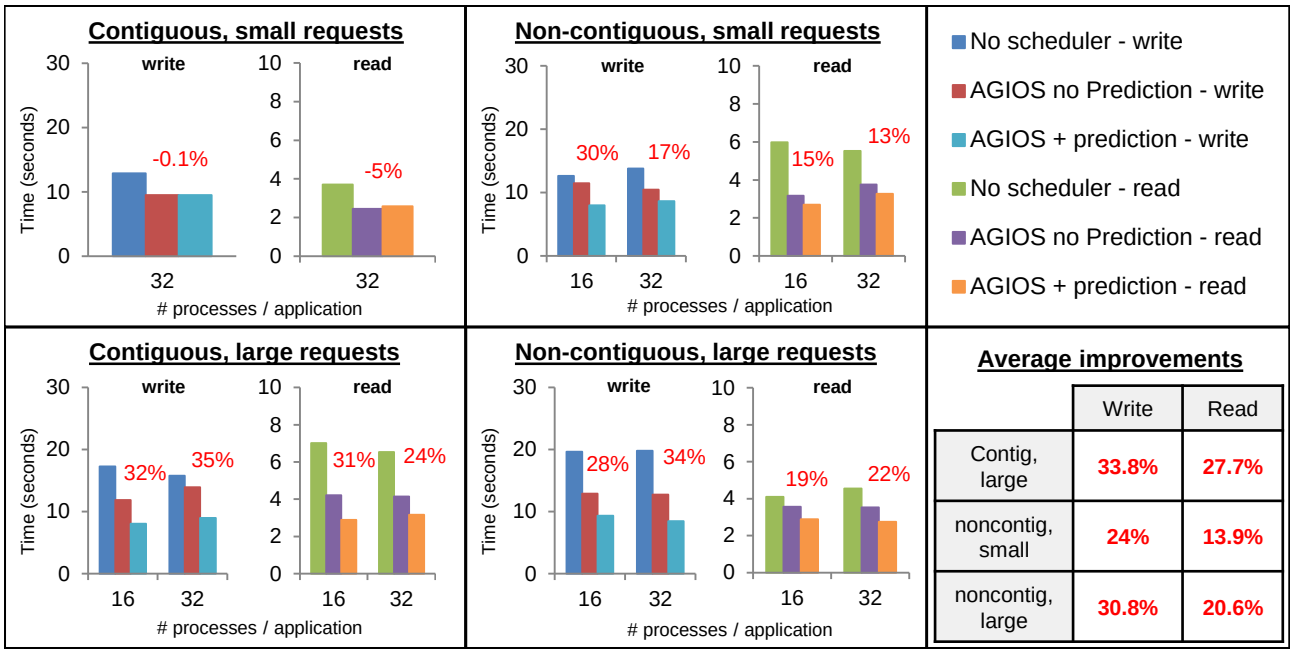
When an actual request arrives, the scheduler connects it to its predicted version, is existent. When this request is selected to be served, the prediction module analyzes the aggregation predicted for it. If the actual aggregation is not as big as the predicted one, the scheduler waits before processing it.

3. Performance Evaluation

We tested AGIOS with the dNFSp parallel file system [2], using 4 data servers and up to 32 clients, on the Edel cluster from Grid5000 [3].

We developed a set of tests using the MPI-IO Test benchmarking tool [1]. We tested situations where the application issue contiguous or non-contiguous (1D strided pattern) requests, and small (16KB) or large (256KB) requests. We also vary the number of processes performing I/O.

We simulate multi-application scenarios by running 4 instances of the same benchmark at the same time. From each



	Write	Read
Contig, large	**33.8%**	**27.7%**
noncontig, small	**24%**	**13.9%**
noncontig, large	**30.8%**	**20.6%**

Figure 1. Results with the AGIOS's prediction module.

execution, we take the completion time of the slowest process.

Fig. 1 presents the results for the AGIOS library with and without the prediction module, comparing with the file system without a scheduler. The numbers in red above the bars show the difference between the two versions of the scheduler, and so the table on the lower right corner. Our approach led to aggregations 25.1% bigger on average.

The prediction module does not provide performance improvement in the test with contiguous and small requests because, in this test, the previous version of the algorithm was already able to aggregate as much as possible. Still, we can see that the prediction module does not degrade performance significantly in this case. Considering the other three workloads, the prediction module was able to improve over the performance of the base algorithm in up to 35.4% - 29.5% on average - for write operations and in up to 31.4% - 20.7% on average - for read operations.

The improvements for tests with large requests are more expressive than for small requests. This happens because large requests provide more aggregation opportunities than small ones. Also, read operations did not benefit from the new approach as much as the write ones. We believe this difference is due to reads being faster than writes, therefore being more affected by the algorithm's waiting times.

4. Conclusion

We proposed an approach to improve I/O scheduling by using information about applications' future accesses obtained from traces. This look into the workload's future is used by the scheduler in its decision-making. Our approach improved performance in up to 35.4% and in 25.1% on average over the base scheduling algorithm.

The **total performance improvement** of the library with the new approach (comparing with not using the library) was of up to 58.9% and of 40.76% on average. We did not observe any case where the use of AGIOS resulted in performance degradation.

References

[1] Mpi-io test user's guide, 2008.

[2] R. B. Avila, P. O. A. Navaux, P. Lombard, A. Lebre, and Y. Denneulin. Performance evaluation of a prototype distributed nfs server. In *16th Symposium on Computer Architecture and High Performance Computing, 2004. SBAC-PAD 2004*, pages 100–105, 2004.

[3] R. Bolze, F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jegou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, 2006.

[4] A. Lebre, Y. Denneulin, G. Huard, and P. Sowa. I/o scheduling service for multi-application clusters. In *Proceedings of IEEE Cluster 2006, conference on cluster computing*, sep 2006.