Coordinating Data Access at I/O Forwarding Nodes

Jean Luca Bez*, Francieli Zanon Boito†, Lucas Mello Schnorr*, Philippe Olivier Alexandre Navaux*
* Institute of Informatics – Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil
† Department of Informatics and Statistics – Federal University of Santa Catarina (UFSC), Florianópolis, Brazil

***Abstract*—In High-Performance Computing, parallel file systems (PFS) are used by applications to obtain I/O performance even when handling large amounts of data. To alleviate the concurrency caused by thousands of nodes accessing a smaller number of PFS servers, intermediate I/O nodes are deployed between processing nodes and the file system servers. Each intermediate I/O node forwards requests from multiple clients to the file system. This scenario is suitable to perform optimizations like I/O scheduling.**

In this paper, we present and evaluate a new scheduling algorithm for the forwarding layer that coordinates intermediate I/O nodes' accesses to servers. Our proposal works to decrease concurrency at the data servers, a factor previously shown to negatively affect performance. The proposed algorithm is able to improve read performance by up to 30% over other scheduling algorithms and by up to 51% over not using an I/O forwarding layer.

I. Introduction

Scientific applications demand increasing performance from the High-Performance Computing (HPC) field. These requirements justify the appearance of ever increasing large scale parallel platforms. It is common for such platforms to have a shared storage infrastructure over a dedicated set of nodes with a parallel file system (PFS) deployment. Due to the historical gap between processing and data access speeds, parallel I/O is a limiting factor for many applications' performance. Furthermore, the applications' performance could be impaired if all processing nodes were to concurrently access the file system servers.

I/O forwarding is a technique employed by several large-scale clusters and supercomputers aiming at reducing the number of clients concurrently accessing the file system servers. In this context, some dedicated nodes receive I/O requests and forward them to the file system [1], as illustrated in Fig. 1. Besides performance, this additional layer between application and file system provides the opportunity to apply optimizations like requests reordering and aggregation.

Write operations have been the main focus of parallel I/O research. However, scientific applications have been reading increasing amounts of data to leverage previous knowledge into their analysis. Argonne National Laboratory analyzed the top ten applications regarding its I/O operations on a supercomputer. It was revealed that large amounts of data were being read, with three of these applications exclusively performing read operations during their executions [2]. For this reason, we focus our study on the performance of read requests.

In this paper, we evaluate a new scheduling algorithm for the forwarding layer that aims at decreasing contention when accessing the parallel file system data servers. It coordinates accesses using time windows so that in each window each I/O node focus all its requests to one data server and different I/O nodes focus on different data servers. We show performance improvements with our algorithm over existing schedulers.

The rest of the paper is organized in the following way: the next section provides a background and discusses related work. Section III discusses our new scheduling algorithm for the forwarding layer, including its implementation and requirements. The experimental methodology and results are presented in Section IV. Final remarks and future work are presented in Section V.

II. Background and Related Work

IOFSL [3] is an open-source framework that implements the I/O forwarding technique as an attempt to bridge the increasing performance scalability gap between computing and I/O components. IOFSL ships I/O calls from compute nodes to dedicated I/O nodes, that perform operations on behalf of those compute nodes. It uses the stateless ZOIDFS I/O protocol, the API from the ZOID forwarding infrastructure [4], and the Buffered Message Interface (BMI) [5] network abstraction layer to provide request forwarding over multiple parallel file systems and interconnection networks. The IOFSL software stack consists of two main components: a ZOIDFS client library running on the compute nodes and an I/O forwarding daemon running on the intermediate I/O nodes. The client

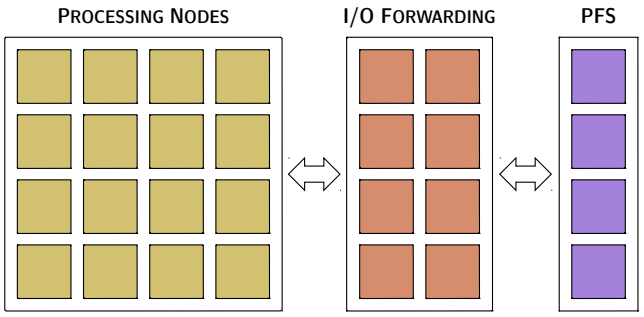

Fig. 1. I/O forwarding scheme on a large-scale cluster or supercomputer. The number of forwarding nodes is generally smaller than the number of processing nodes and larger than the number of parallel file system servers.

library transparently forwards I/O requests from the compute node to the corresponding I/O node.

Ohta et al. [6] improved the performance of the IOFSL framework by using I/O scheduling. They implemented two algorithms: a simple *First In, First Out* (FIFO) and a quantum-based algorithm they called *Handle-Based Round-Robin* (HBRR). The latter is based on an algorithm successfully applied to parallel file systems' data servers, and aims at performing requests reordering and aggregations to improve the generated access pattern.

We have chosen to use IOFSL in our research because it is the only available open-source tool already tested on large scale clusters and supercomputers. Furthermore, we could build on previous works and contributions to effectively compare our new solution with the state of the art.

AGIOS [7] is a scheduling library that can be used by I/O services to manage incoming I/O requests at the file level (file offsets). It implements five scheduling algorithms and it exposes a simple API to build new schedulers. Because we wanted our solution to be generic and possibly ported to other forwarding tools, or even used in the file system server context, we have integrated the AGIOS scheduling library in the IOFSL as a scheduling option and harness its API to prototype our new scheduler. With the AGIOS scheduling option, requests are added to the library's queues when they arrive at the forwarding nodes. When the algorithm applied by AGIOS decides it is time to process a request, a callback function inside IOFSL simply adds it to the dispatch queue. This ensures requests will be processed in the order dictated by the scheduling algorithm in use.

III. Coordinating PFS Server Access with TWINS

In this section, we present a new I/O scheduling algorithm for the I/O forwarding layer called *Time WINdows Scheduler* (TWINS). The main idea is coordinate intermediate I/O nodes accesses to the file system servers so that, at any given moment:

1) an I/O node is focusing all its accesses on one server;
2) different I/O nodes are focusing on different servers.

TWINS keeps multiple request queues, one per data server. During the execution, it iterates the queues in a round robin fashion, respecting the time window dedicated to each server. This implies in additional waiting time if there are no requests for the current server, even if there are incoming requests to other data servers. The pseudo-code for this scheduler is presented in Algorithm 1.

Our scheduling algorithm requires additional information to work, besides the typically available information - file handle, offset, type, and size - found I/O requests. It is necessary to know exactly where each stripe of a file is located so the requests could be grouped by data servers.

We have modified the IOFSL code to collect the file layout information from the file system when opening or creating a file. Since this information is requested only once and kept while the file is open, no significant overhead is expected. Furthermore, as the file distribution never changes during its lifetime and in several file systems it is possible to obtain the data file layout, our approach continues to be generic.

Algorithm 1 TWINS

Require: $Q[i]$ is the updated list of requests to server i

```
1:  i ← 0
2:  while true do
3:      resetTimer()
4:      while elapsedTime() < windowSize do
5:          if length(Q[i]) > 0 then
6:              processRequest(Q[i])
7:          else
8:              timeout ← windowSize − elapsedTime()
9:              timedWaitForRequests(Q[i], timeout)
10:         end if
11:     end while
12:     i ← nextServer(i)
13: end while
```

Using the file distribution information, the starting server for a request is obtained as a function of its starting offset and stripe size. An additional translation step is required so each IOFSL server focuses its window on a distinct server. This translation is done according to the IOFSL node identifier. The N_{th} I/O node will use the N_{th} permutation of the data servers list as a translation rule. Therefore, if the number of intermediate I/O nodes is larger than the number of data servers, more than one node may access the same server at the same time, but these concurrent accesses are minimized.

IV. Experimental Results

Experiments were conducted in clusters from the Nancy site of Grid'5000 [8]. Four machines from the Grimoire cluster were selected as PVFS2 servers and 32 machines from the Grisou cluster as clients. Four additional machines from Grisou were configured to act as the forwarding layer. Clients are equally distributed among the I/O nodes.

Grimoire's nodes have two 8-core Intel Xeon E5-2630 v3 and 126 GB of RAM. Grisou's nodes are identical to Grimoire's ones. A 558 GB hard disk is used for storage at the servers. Nodes are interconnected through a 10 Gbps Ethernet network, and there is a 10 Gbps link between the clusters.

PVFS version 2.8.2 was deployed with its default parameters. Data servers were configured to bypass buffer caches. IOFSL uses the PVFS2 client library to communicate directly with the file system. The maximum number of requests that can be aggregated from the dispatch queue was 16, the default.

The MPI-IO Test benchmark was executed by 128 processes. Each one generates 1024 requests of 32 KB (32 MB per process), a total of 4 GB per test. Processes read a shared file using a 1D strided access pattern. From each execution, we take the completion time of the slowest process (makespan).

Experiments were repeated 8 times in a random order, and error bars were calculated using a 99.7% confidence interval, i.e. there is a 99.7% probability that the true mean lies between the lower and upper bounds of the interval. These bounds are

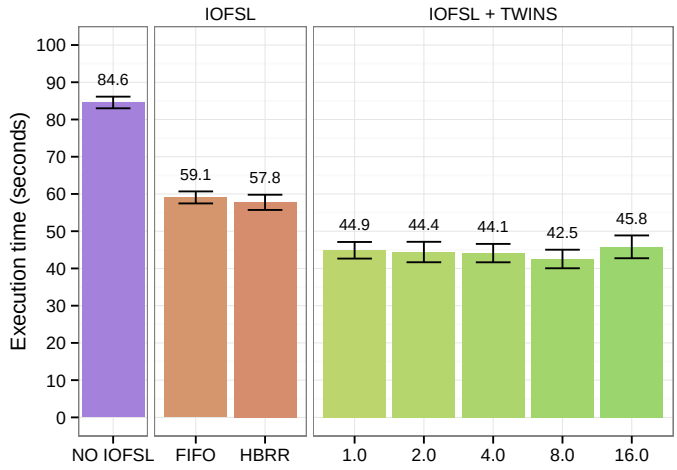

Fig. 2. Comparison of different window sizes for TWINS scheduler and the default IOFSL schedulers - FIFO and HBRR - for the single shared-file with 1D strided access pattern.

equivalent to three times the standard deviation divided by the square root of the number of measurements.

Figure 2 depicts the results obtained by TWINS and compares them with results obtained with other employed scheduling algorithms for the I/O forwarding layer and with not using IOFSL. For the 1D strided access pattern, TWINS provides a performance improvement of approximately 30% over the HBRR algorithm (with IOFSL) and of approximately 51% over not using IOFSL. This access pattern is ideal for TWINS because the scheduler always has requests for all servers since processes start their accesses at different ones. Therefore it has the opportunity to perform meaningful coordination, reducing competition for resources.

The time window duration also has an impact on the execution time. A small window does not allow an effective access coordination because it does not hold requests to other servers long enough for them to be aggregated. On the other hand, a large window imposes overhead as there are not enough requests to each data server to fill a whole window, so the scheduler spends too much time waiting. Figure 2 also shows that the window size that presented the best performance is of 8 milliseconds. This initial results demonstrate that TWINS is able to coordinate I/O nodes access by reducing contention when accessing the file system data servers.

V. Conclusions

In this paper, we studied read performance in the I/O forwarding layer. We evaluated two algorithms from related work - FIFO and HBRR - in the IOFSL framework. Our analysis has shown that, despite improving read performance, techniques to adjust the access patterns (requests aggregation and reordering) are only partially effective because the access pattern is not the main factor for read performance through the I/O nodes.

We have proposed a new scheduling algorithm for the I/O forwarding layer called TWINS. Our algorithm uses time windows to coordinate the I/O nodes' accesses to different data servers, working to decrease contention. Our performance evaluation has shown improvements for 1D strided access patterns of 30% over the FIFO and HBRR algorithms. Compared to not using I/O forwarding nodes this gains goes up to 51%. These initial results demonstrates that our new scheduler is able to reduce contention and improve performance.

Future work will focus on studying other access patterns and exploring additional I/O forwarding layer configurations such as the ratio of clients to I/O nodes and additional benchmarks. Furthermore, we also expect to focus our study on proposing an automatic mechanism to tune the window size based on the system configuration and on the application access pattern.

Acknowledgments

This research has received funding from the MCTI/RNP-Brazil under the HPC4E Project, grant agreement nº 689772.

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr/).

References

[1] J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-C. Andre, D. Barkai, J.-Y. Berthou, T. Boku, B. Braunschweig, F. Cappello, B. Chapman, X. Chi, A. Choudhary, S. Dosanjh, T. Dunning, S. Fiore, A. Geist, B. Gropp, R. Harrison, M. Hereld, M. Heroux, A. Hoisie, K. Hotta, Z. Jin, Y. Ishikawa, F. Johnson, S. Kale, R. Kenway, D. Keyes, B. Kramer, J. Labarta, A. Lichnewsky, T. Lippert, B. Lucas, B. Maccabe, S. Matsuoka, P. Messina, P. Michielse, B. Mohr, M. S. Mueller, W. E. Nagel, H. Nakashima, M. E. Papka, D. Reed, M. Sato, E. Seidel, J. Shalf, D. Skinner, M. Snir, T. Sterling, R. Stevens, F. Streitz, B. Sugar, S. Sumimoto, W. Tang, J. Taylor, R. Thakur, A. Trefethen, M. Valero, A. Van Der Steen, J. Vetter, P. Williams, R. Wisniewski, and K. Yelick, "The international exascale software project roadmap," *International Journal of High Performance Computing Applications*, vol. 25, no. 1, pp. 3–60, Feb. 2011.

[2] R. Ross, "Future HPC systems and some implications for storage software," http://www.opensfs.org/wp-content/uploads/2014/04/D2_S26_2020Panel_Ross.pdf, Accessed: August 2015.

[3] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, and P. Sadayappan, "Scalable I/O forwarding framework for high-performance computing systems," in *Proceedings...*, IEEE International Conference on Cluster Computing and Workshops. IEEE, Aug. 2009, pp. 1–10.

[4] K. Iskra, J. W. Romein, K. Yoshii, and P. Beckman, "ZOID: I/O forwarding infrastructure for petascale architectures," in *Proceedings...* 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2008, pp. 153–162.

[5] P. Carns, R. Ross, W. L. III, and P. Wyckoff, "BMI: a network abstraction layer for parallel I/O," in *19th IEEE International Parallel and Distributed Processing Symposium*, April 2005, pp. 8 pp.–.

[6] K. Ohta, D. Kimpe, J. Cope, K. Iskra, R. Ross, and Y. Ishikawa, "Optimization techniques at the I/O forwarding layer," in *Proceedings...*, IEEE International Conference on Cluster Computing. IEEE, Sep. 2010, pp. 312–321.

[7] F. Z. Boito, R. V. Kassick, P. O. A. Navaux, and Y. Denneulin, "AGIOS: Application-guided I/O scheduling for parallel file systems," in *Proceedings...*, International Conference on Parallel and Distributed Systems (ICPADS). IEEE, Dec. 2013, pp. 43–50.

[8] D. Balouek, A. C. Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, *Adding Virtualization Capabilities to the Grid'5000 Testbed*, ser. Communications in Computer and Information Science. Springer International Publishing, 2013, vol. 367, pp. 3–20.