A Full Year I/O Request Size Analysis of HPC Applications on the Intrepid Supercomputer

Valéria S. Girelli[1], Jean Luca Bez[1], Francieli Z. Boito[2], Pablo J. Pavan[1] and Philippe O. A. Navaux[1]
[1]Universidade Federal do Rio Grande do Sul — Instituto de Informática, Brazil
{vsgirelli, jlbez, pablo.pavan, navaux}@inf.ufrgs.br
[2]Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, 38000 Grenoble, France

Abstract

This study used data from an entire year of characterization with Darshan on the Intrepid Blue Gene/P supercomputer, at Argonne. Considering that data access is a bottleneck for several HPC applications, understanding their I/O behavior may help us apply optimization techniques at the I/O system. By analyzing the collected data, we could observe that POSIX is still widely used by the applications running on Intrepid. Additionally, some of the requests issued by the applications are very small which generally translates into poor I/O performance.

1. Introduction

In High Performance Computing systems (HPC), besides the high processing performance, the applications also demand a high storage capacity and efficiency in the data access. These applications deal with a great amount of data, and hundreds of computing nodes can access the storage system concurrently. Therefore, performing I/O operations may become a bottleneck for an increasing number of HPC applications, due to the historical gap between processing and data access speed.

Parallel File Systems (PFS) act providing an abstraction of the data on the storage system. They receive requests from the computing nodes, process these requests and access the storage devices. However, some problems may appear depending on the way these requests are performed. Occasionally, the requests sent to the storage system are too small, transferring small data amounts that hardly compensate the cost of accessing the storage devices [3, 10]. One solution to this problem is to aggregate the requests, one of the possible optimizations that can be done at the I/O system. Another situation, though is not the focus of this work, is when the access is not aligned to the stripe used by the PFS, what may further degrade performance. In this case, applications can use interfaces such as MPI-IO, that may assist in the aligned access.

However, the applications can present different access patterns. In consequence, to apply optimization techniques, we must at first understand the application I/O behavior. Tools like Darshan [4] provide a characterization of such behavior by creating profiles of the operations. Darshan was developed at Argonne Leadership Computing Facility (ALCF)[1] and captures this sort of information at the application level. Therefore, with the objective of understanding the I/O global behavior in a supercomputer, this study analyzed data collected using Darshan on the Intrepid Blue Gene/P, the ALCF supercomputer.

The remainder of this paper is organized as follows. The information about the collected data used and the methodology applied are detailed in Section 2. Analysis and results are presented in Section 3. Section 4 discusses related work. Finally, Section 5 concludes this paper and discusses future work.

2. Methodology

During the years of 2010, 2012 and 2013, data from the execution of a variety of scientific applications was collected by the Darshan I/O Characterization Tool on the supercomputer Intrepid Blue Gene/P, at ALCF. Darshan intercepts I/O function calls and records a fixed-size collection of statistics for each file that is opened by the application [1]. The collected information includes access patterns, access sizes, operation counters and time spent in I/O operations. Darshan generates a separate log file for each job, and stores this information in a compressed binary format [1].

In this study we analyzed the data collected during the year of 2012 generated by Darshan versions 1.23, 1.24 and 2.0, resulting in 36, 359 and 91.603 jobs, respectively. Until the version 2.0, Darshan only instrumented applications that successfully called `MPI_Init()` and `MPI_Finalize()`.

1 `https://www.alcf.anl.gov/`

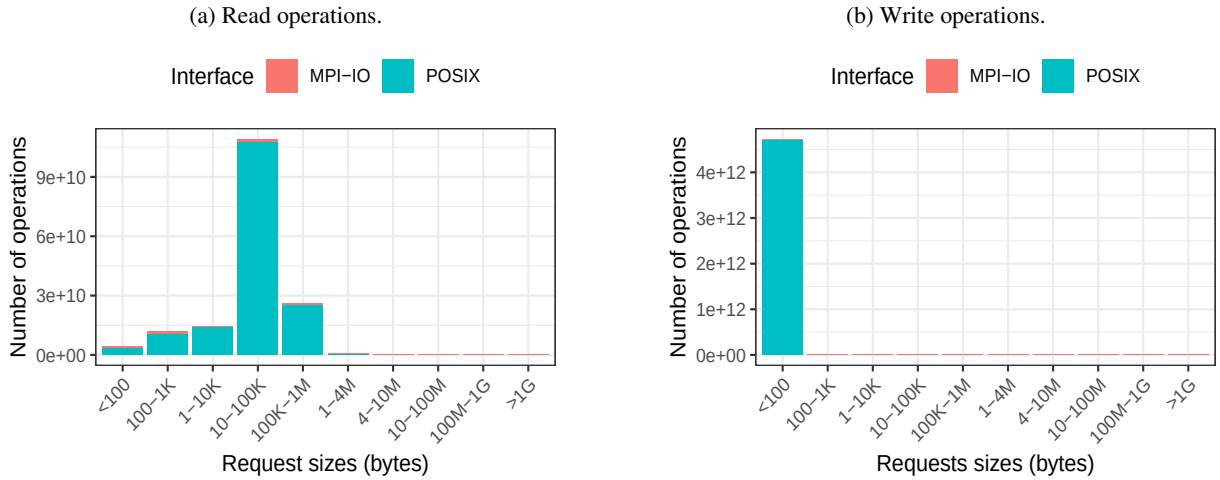


Figure 1: Access size distribution for operations by interface. The y-axis has different scales.

Therefore, although Darshan was enabled for all users, the application coverage rate varied between 20% and 80% from week to week [1]. Some of the collected information was anonymized before being publicly available. The anonymized information includes the job identifier, user identifier, and application name.

Since the Darshan logs are stored in a compressed binary format, to extract and analyze interesting information for our analysis, we used the darshan-parser tool, provided by Darshan, to generate a text file. This text file is organized in two sections[2]. The first one, at the top of the file, describes information about the executed application. The second section contains the observations of each counter captured by Darshan, in a tabular format.

For our analysis, the relevant information about the application was: application identifier, represented by the column `exec`; the job identifier, represented by `jobid`; user identifier, as `uid`; number of processes, as `nproc`, and `runtime`. Likewise, we calculated total I/O time in the same way Darshan does, using information about the slowest MPI rank. Additionally, Darshan captures request information and stores in counters separated in operation type (read or write) and interface (`AGG` for MPI-IO, and POSIX). For each operation in each interface, the access sizes are divided in the following bins: 0-100 bytes, 100 bytes - 1 KB, 1-10 KB, 10-100 KB, 100 KB - 1 MB, 1-4 MB, 4-10 MB, 10-100 MB, 100 MB - 1 GB, and 1 GB - PLUS. Thus, we used Python to extract all the useful data from the text file, storing them in CSV format, and made the analysis using R.

2 `http://www.mcs.anl.gov/research/projects/darshan/docs/darshan-util.html`

3. Results and Analysis

Analyzing the number of read and write operations performed using each interface, we observed that surprisingly 97.3% of read operations were being performed by POSIX. For the write operations, the percentage of POSIX utilization reaches 99.2%. As expected and presented by previous work [2, 7, 8], POSIX is still more widely used if compared to MPI-IO. However, the data collected in 2012 further highlights the gap between the POSIX and MPI-IO utilization on the Intrepid supercomputer.

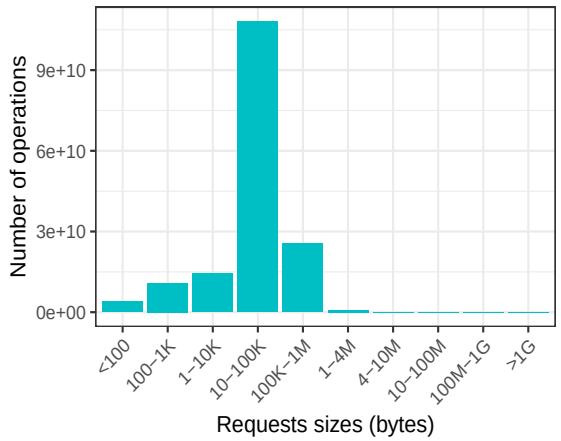
Figure 1a shows the distribution of the read access sizes observed for both POSIX and MPI-IO interfaces. The most common read size was between 10KB and 100KB, representing 64.3% of the read operations. Figure 1b shows the distribution of the write access sizes, also for both interfaces, and the most used write size was up to 100 bytes, which is a really small access size. This size represents 98.7% of the write operations.

We can notice, therefore, that the applications observed during the characterization made even smaller requests than the observed in previous work [2, 9, 6], mainly on write operations. This type of small accesses may degrade performance [3, 10], once they define the transference size between processing nodes and the storage devices.

As presented in Figure 1, the great number of requests issued using POSIX determine the overall request size distribution of the collected data. Therefore, the distribution for both read and write operations using POSIX only is very similar to the overall distribution, as shown in Figure 2.

For the requests realized with MPI-IO, we can observe a spread distribution in Figure 3. The two most common access sizes intervals for read operations were between 100 bytes and 1KB and between 10KB and 100KB. These two

(a) Read operations.



(b) Write operations.

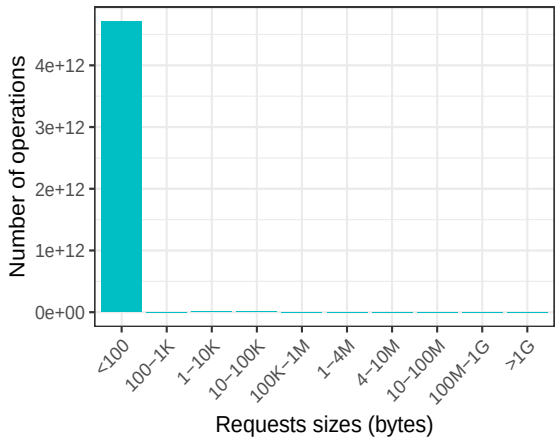


Figure 2: Access size distribution for operations using POSIX. The y-axis has different scales.

intervals represent up to 34.2% and 23.7% of the total read operations performed by MPI-IO, respectively. For the write operations, the most popular access size was also up to 100 bytes, followed by requests ranging between 10KB and 100KB. These two intervals represent up to 53.6% and 20.6% of the total write operations realized using MPI-IO, respectively.

If compared to the requests performed by POSIX, the access size distribution observed in the requests made using MPI-IO is wider. One explanation for this is the possibility to define more complex datatypes with MPI-IO. When a process wants to access small and sparse portions of a file, a datatype can be defined by aggregating this portions into a large request. Moreover, when applications use MPI-IO collective operations, usually a small number of processes do the accesses and then distribute the data to the others [5]. The number of proccess is defined following some heuristics or can be defined by the user.

The write operations presented a condensed distribution variation when compared to the read access size distribution, for both interfaces. This may be caused by the individual behavior of the applications and requires a deeper study to understand.

4. Related Work

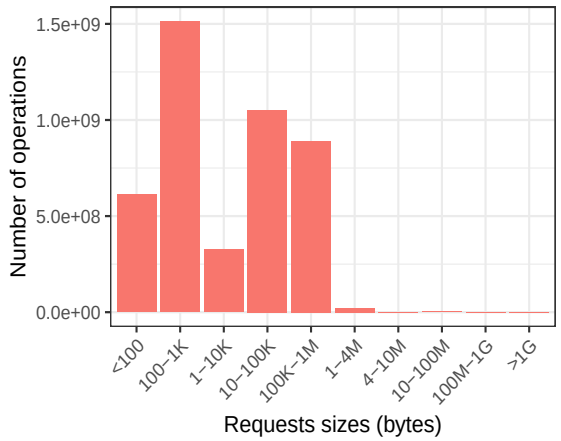
Wang *et al.*, in 2004, analyzed two physics applications and the *ior* benchmark on a large Linux cluster with more than 800 dual processor nodes at the Lawrence Livermore National Laboratory (LLNL) [9]. Usually, each application presented only one or two typical request sizes. Large requests from several hundred KB to several MB were very common. In some cases, however, small requests accounted for more than 90% of all requests, but the greatest amount of data was still transferred by large requests.

In a work made in 2010 [6], Kim *et al.* characterized the workload of Spider, a Lustre-based storage cluster at Oak Ridge National Laboratory (ORNL). They observed three main request sizes: less than 16KB, 512KB and 1MB. This three request sizes represented more than 95% of the total requests. About 50% of the request of less than 16KB were write operations, while the read operations represented 20%. They also correlated the access sizes with the bandwidth, observing that the higher bandwidth was attained with 1MB large requests.

In a previous work published in 2011 [2], Carns *et al.* analyzed the behavior of 66 science and engineering applications. The data was collected with Darshan during two months of 2010, also on the supercomputer Intrepid, Argonne. They demonstrated that the most popular read size was between 100KiB and 1MiB, while the most popular write size was between 100 bytes and 1KiB. However, a deeper investigation revealed that a few applications influenced the access size observed. If those applications were removed from the analysis, then the most popular size for both reads and writes was 100KiB to 1MiB. The work also demonstrated that some of the applications increased their performance when using larger accesses sizes.

Although Carns *et al.* also analyzed data from the I/O workload on Intrepid, their study used a smaller dataset. On the other hand, this study was the first to investigate the entire year of 2012 in this same supercomputer. We observed the access size distributions taking into account the different I/O interfaces.

(a) Read operations.



(b) Write operations.

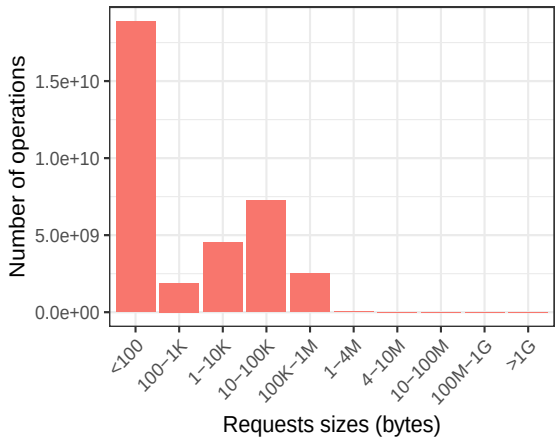


Figure 3: Access size distribution for operations using MPI-IO. The y-axis has different scales.

5. Conclusion and Future Work

This study used data collected during an entire year of characterization by Darshan. We could observe with a considerable amount of data that POSIX is still surprisingly used by the applications running on Intrepid, overcoming 97% of utilization for read operations and 99% for write operations. Applications are also performing very small write requests that do not exceed 100 bytes, what may indicates that I/O is being performed in a really inefficient way, probably accessing a few variables at a time. Therefore, this great amount of small operations using POSIX is not taking advantage of the request aggregation made by MPI-IO and neither of any other high-level interface.

In future work, we would like to investigate if this behavior is the result of a small group of applications performing a huge part of the requests, or if it represents the global behavior observed in Intrepid. We should also look into the amount of data being transferred by each access size observed, and compare which are the access sizes responsible for the largest amount of transferred data. Another possible analysis is to investigate the system time spent in the operations performed with each access size.

Acknowlegments

The research has received funding from PIBIC CNPq-UFRGS and PROBIC FAPERGS-UFRGS. It was also supported by Petrobras project, grant n. 2016/00133-9.

This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

References

[1] P. Carns. ALCF I/O Data Repository. Technical report, Argonne Leadership Computing Facility, Feb 2013.

[2] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross. Understanding and improving computational science storage access through continuous characterization. *Trans. Storage*, 7(3):8:1–8:26, Oct. 2011.

[3] P. Carns, S. Lang, R. Ross, M. Vilayannur, J. Kunkel, and T. Ludwig. Small-file access in parallel file systems. In *2009 IEEE International Symposium on Parallel Distributed Processing*, pages 1–11, May 2009.

[4] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley. 24/7 Characterization of petascale I/O workloads. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10, Aug 2009.

[5] J. M. del Rosario, R. Bordawekar, and A. Choudhary. Improved parallel i/o via a two-phase run-time access strategy. *SIGARCH Comput. Archit. News*, 21(5):31–38, Dec. 1993.

[6] Y. Kim, R. Gunasekaran, G. M. Shipman, D. A. Dillow, Z. Zhang, and B. W. Settlemyer. Workload characterization of a leadership class storage cluster. pages 1–5, Nov 2010.

[7] H. Luu, M. Winslett, W. Gropp, R. Ross, P. Carns, K. Harms, M. Prabhat, S. Byna, and Y. Yao. A multiplatform study of i/o behavior on petascale supercomputers. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, pages 33–44, New York, NY, USA, 2015. ACM.

[8] E. Smirni and D. Reed. Lessons from characterizing the input/output behavior of parallel scientific applications. *Performance Evaluation*, 33(1):27 – 44, 1998.

[9] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, D. D. E. Long, and T. T. Mclarty. File system workload analysis for large scientific computing applications. Apr. 2004.

[10] F. Zanon Boito. Transversal I/O Scheduling: from Applications to Devices. page 171, 03 2015.