An I/O Performance Evaluation Tool for Distributed Data-Intensive Scientific Applications

Eduardo C. Inacio and Mario A. R. Dantas

[1]Programa de Pós-Graduação em Ciência da Computação (PPGCC)
Universidade Federal de Santa Catarina (UFSC)
Florianópolis, SC – Brasil

eduardo.camilo@posgrad.ufsc.br, mario.dantas@ufsc.br

Abstract. *I/O performance arises as a major bottleneck in nowadays data-intensive scientific applications. In order to identify execution parameters that provide an improved I/O performance, experimental efforts relying on synthetic I/O workload generators are widely employed. Focusing on addressing limitations of current workload generators, and to provide a more flexible, unified, and user-friendly approach for parallel I/O performance analysis, we have proposed a differentiated tool, called* IORE. *This paper focuses on demonstrating* IORE *applicability for I/O performance analysis of a workload derived from a real-world on scientific application. Our results indicate the potential of* IORE *as a parallel I/O experimental tool.*

1. Introduction

As data-intensive science evolves, both scientific and industry societies strive to develop more efficient methods and tools for efficiently deriving meaningful information from this unprecedented amount of data produced daily, a phenomena coined as data deluge [Bell et al. 2009]. Data sources can be as diverse as possible, ranging from digital social networks and Internet of Things devices to large-scale scientific instruments and high performance computing (HPC) simulations [Ackx 2014, Roten et al. 2016, CERN 2016]. In order to meet both space and time requirements posed by such data-intensive applications, modern large-scale computing infrastructures (*e.g.,* clusters, clouds, and supercomputers) rely on very sophisticated parallel input/output (I/O) software stacks and distributed storage systems, usually implemented through Parallel File Systems (PFSs).

Although significant improvements have been achieved in the last decades due to a great number of research works, the parallel I/O and storage performance still arises as one of the major bottlenecks in this context. In some cases, the I/O time can easily account for more than 70% of the total execution time of a large-scale scientific data visualization system [Nonaka et al. 2018]. Consequently, optimizing the I/O performance of data-intensive applications remains of utmost importance in this context.

The main challenge in optimizing a parallel I/O and storage system is identifying which configuration can produce the best performance in such a complex environment with a great number of parameters. As some previous research works demonstrated [Herbein et al. 2016, Wang et al. 2017, Inacio et al. 2017a], many aspects can affect the I/O performance perceived by an application, including its own access patterns, underlying architecture characteristics, devices properties, and environment configurations. A common practice adopted in research works is conducting a number of experiments in the target computing system in order to assess its I/O performance capabilities and the impact of changing some controlled parameters. For that purpose, a variety of approaches and tools have been proposed, with synthetic I/O workload generators, such the INTERLEAVED-OR-RANDOM (IOR) benchmark [Shan et al. 2008], arising as the most popular in the parallel I/O research community [Boito et al. 2018].

Focusing on addressing some limitations of the IOR benchmark and facilitating the I/O performance evaluation task on such complex environments, we have proposed IOR-EXTENDED (IORE) [Inacio and Dantas 2018], what can be considered as a differentiated I/O performance evaluation tool. Among the main new features introduced in IORE are the experiment-driven execution, facilitated generation of heterogeneous offset-based and dataset-based I/O workloads, integration with distributed storage systems for in-test configuration, and exporting of collected metrics and statistics. In this paper, we demonstrate IORE applicability in the analysis of the I/O performance of different approaches for writing and reading a large Cartesian dataset in a PFS, a typical workload in real-world data-intensive scientific simulation and visualization systems. Moreover, a variety of scenarios are explored using IORE parameters, whose experimental results are discussed shedding light on particular I/O performance behaviors and best performing configurations.

The remainder of this paper is organized as follows. Section 2 provides a brief overview of some related synthetic I/O workload generators. In Section 3, IORE is presented, focusing on its dataset-based workload generation capabilities. Details on scenarios and methods employed in the experiment conducted in this research work are provided in Section 4, while experimental results are discussed in Section 5. Section 6 concludes this paper with additional remarks about the observed results and planned directions for IORE development.

2. Synthetic I/O Workload Generators

Synthetic I/O workload generators have been widely employed in research works focusing on parallel I/O and storage performance for data-intensive computational science applications [Boito et al. 2018]. The MPI-TILE-IO [ANL 2002] benchmark evaluates the I/O performance of non-contiguous file data accesses carried out by distributed processes using MPI-IO collective operations. Each participating process is assigned a partition (tile) of a two-dimensional matrix, where the number of tiles and the amount of data per tile can be specified by the user.

While MPI-TILE-IO is focused on a specific dataset type, the IOR benchmark [Shan et al. 2008] was designed for stress testing a large-scale storage system. Through a considerably large number of offset-oriented parameters, a variety of I/O workloads can be generated with IOR, including contiguous and non-contiguous, sequential and random data accesses. Moreover, tests can be carried out using I/O middlewares and high-level I/O libraries among the most commonly observed in real-world applications, such as POSIX, MPI-IO, HDF5, and NETCDF.

Although being a *de facto* standard in the parallel I/O research community, some specific dataset-based workloads are not easily or even precisely reproduced by IOR. Further, only homogeneous workloads, in the sense of the amount of data transferred per process, are supported in IOR. Focusing on addressing these issues, and providing a more flexible, unified, and user-friendly approach for research works on parallel I/O and storage performance evaluation, we have proposed the IORE I/O performance evaluation tool [Inacio and Dantas 2018].

3. IORE for Performance Evaluation of Dataset-based Workloads

As the name suggests, IORE was initially designed as an enhanced version of the IOR benchmark. However, the proposal of IORE goes beyond generating homogeneous offset-based I/O workloads and collecting performance metrics; it was designed to provide an experiment-oriented approach for I/O performance evaluation studies, contributing to reproducible results. Users define an IORE experiment through an intuitive JSON structure, specifying, in addition to expected workload parameters, the number of replications, order of run executions, among other options.

Also, IORE was designed focusing on extensibility, providing well specified interfaces for its main modules (Figure 1), such as Abstract File Storage Backend (AFSB) and Abstract File I/O (AFIO) interfaces. Another new feature introduced in IORE is the capability of exporting performance metrics after experiment completion. This aims at facilitating I/O performance research works, taking the burden from users of error-prone output parsing, and, thereby, reducing the time to results analysis. Finally, workload generation capabilities were also considerably extended in IORE, with the support for heterogeneous and dataset-based workloads.

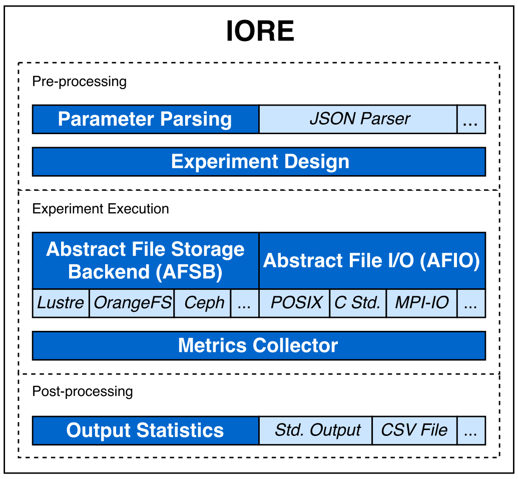
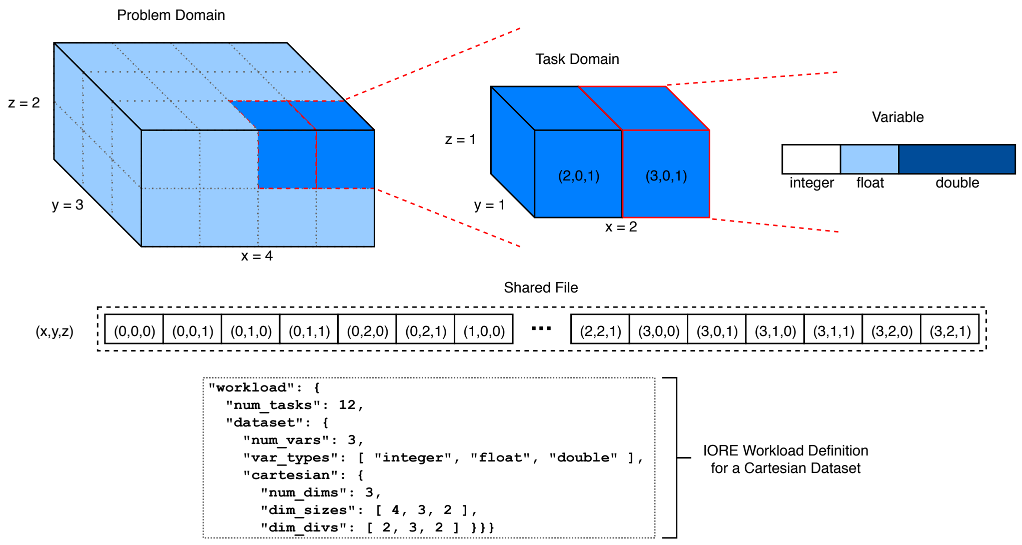


Figure 1. Overview of IORE main modules.

Generating dataset-based workloads is of particular interest for evaluation of I/O performance on large-scale computational science simulation and scientific data visualization, as most of these applications process datasets in well-known formats. Figure 2 illustrates an example of a Cartesian dataset workload representation using IORE. The problem domain (*i.e.,* the total dataset) in this example is composed of a three-dimensional structured grid with size $4 \times 3 \times 2$ (`dim_sizes`). This dataset is distributed across 12 processes, where each process receives a part of the grid with size $2 \times 1 \times 1$, by dividing each dimension of the dataset according to the workload definition (`dim_divs`). Each element of the Cartesian grid holds, in this example, three variables: an integer, a single-precision, and a double-precision floating-point (`var_types`). In a real-world application, these variables could refer to physical properties, such as temperature, pressure, and velocity. It can be observed in Figure 2 that the dataset in this example is stored in a single shared file, using a row-major order approach. IORE also supports accessing data in a file per process fashion, even though this parameter is not presented in Figure 2.



```
"workload": {
  "num_tasks": 12,
  "dataset": {
    "num_vars": 3,
    "var_types": [ "integer", "float", "double" ],
    "cartesian": {
      "num_dims": 3,
      "dim_sizes": [ 4, 3, 2 ],
      "dim_divs": [ 2, 3, 2 ] }}}
```

Figure 2. Example of a Cartesian dataset workload representation using IORE.

The dataset-based workload definition proposed in IORE, although simple, is very flexible and scalable, allowing for the definition of varying dataset sizes and shapes. Further, complex and sophisticated experiments can be conducted combining dataset-based workload definitions with other orthogonal parameters, such as different I/O APIs, middlewares, and high-level I/O libraries. This becomes a notably helpful feature for characterizing the impact of different parameters in dataset access performance, and for identifying parameters alternatives with improved performance. Moreover, the intrinsic experimental-oriented execution of IORE favors more reproducible results, leading to more accurate conclusions and more efficient optimization decisions.

4. Experimental Scenarios and Methods

To demonstrate IORE capabilities for I/O performance evaluation of dataset-based workloads, an experiment was conducted. This experiment consists of multiple distributed processes writing and reading a three-dimensional Cartesian dataset with size $11520 \times 5760 \times 94$, where each element of the dataset corresponds to a single-precision floating-point variable, to and from one or more files in a PFS. Consequently, a total of 23 GiB of data is transferred between compute nodes and the storage system at each write/read test. It is worth mentioning this workload is derived from a real-world large-scale scientific data visualization problem [Inacio et al. 2017b], in which a dataset with the same characteristics, produced by a sub-kilometer global simulation of deep moist atmospheric convection, was post-processed at the K supercomputer, in Japan. The workload specified for these experiments corresponds to one time step of the original real-world application, a reasonable consideration given that time steps are processed sequentially and independently at the post-hoc visualization system.

In this experiment, the impact of three parameters in the I/O performance was evaluated, namely, the number of processes accessing the dataset, the file mode adopted for storing the dataset, and the AFIO implementation. Table 1 lists these parameters and their respective values. As the dataset is fixed for this experiment, changing the number of processes implies in changing the dataset partitioning, which, in summary, results in less data per process. Focusing on demonstrating some optimizations provided by MPI-IO, collective I/O (col) and file view (fv) options from the MPI-IO AFIO implementation were also explored in this experiment. These two options only make sense for distributed data accesses at a shared file, and, thereby, were only evaluated for the Nx1 file mode. A full factorial experimental design was employed, resulting in 14 experiment runs, each consisting of a different combination of parameter values. The experiment was replicated two times, with the run execution order randomized across replications, favoring statistically independent measurements.

Table 1. Parameters and values considered in this research work using the IORE performance evaluation tool.

Parameter	Description	Values
`num_tasks` (`dim_divs`)	number of processes (dataset partitioning)	20 ($5 \times 4 \times 1$)
		40 ($10 \times 4 \times 1$)
`file_mode`	dataset file storage mode	Nx1 (shared file)
		NxN (file per process)
`afio`	AFIO implementation (specific options:	POSIX
	col = collective I/O, fv = file view)	C Stream
		MPI-IO
		MPI-IO (col + fv)

The experimental environment is composed of two clusters from the Grid'5000 testbed, in France. Data and metadata servers of the ORANGEFS PFS were deployed into the four nodes of the Hercule cluster, where each node contains two octa-core Intel Xeon E5-2620 CPUs, 32 GB RAM, and 6 TB SATA HDD. IORE was compiled with the MPICH MPI implementation and deployed at 20 nodes of the Nova cluster, each node containing two octa-core Intel Xeon E5-2620 CPUs, 64 GB RAM, and the ORANGEFS PFS served by the Hercule cluster mounted locally. A switched 10 Gigabit Ethernet network interconnects both storage and compute nodes.

5. Results and Discussion

Results of the experiment conducted in this research work are presented in Figure 3. The y axis refers to the average of the two experiment replications for the time taken by all processes participating in a given write/read test to conclude transferring the dataset to/from the PFS, while the x axis presents different AFIO implementations and options. Plots are organized in a grid, with upper plots presenting results of write tests, and lower plots presenting results of read tests. From left to right, plots refer to variations in the file mode and number of processes parameters, in this order: Nx1 with 20 processes, Nx1 with 40 processes, NxN with 20 processes, and NxN with 40 processes. As the variance of the response variable (*i.e.,* latency) was very small for all experiment runs, with a coefficient of variation smaller than 10% in 13 out of 14 runs, error bars were not included in these plots for readability.

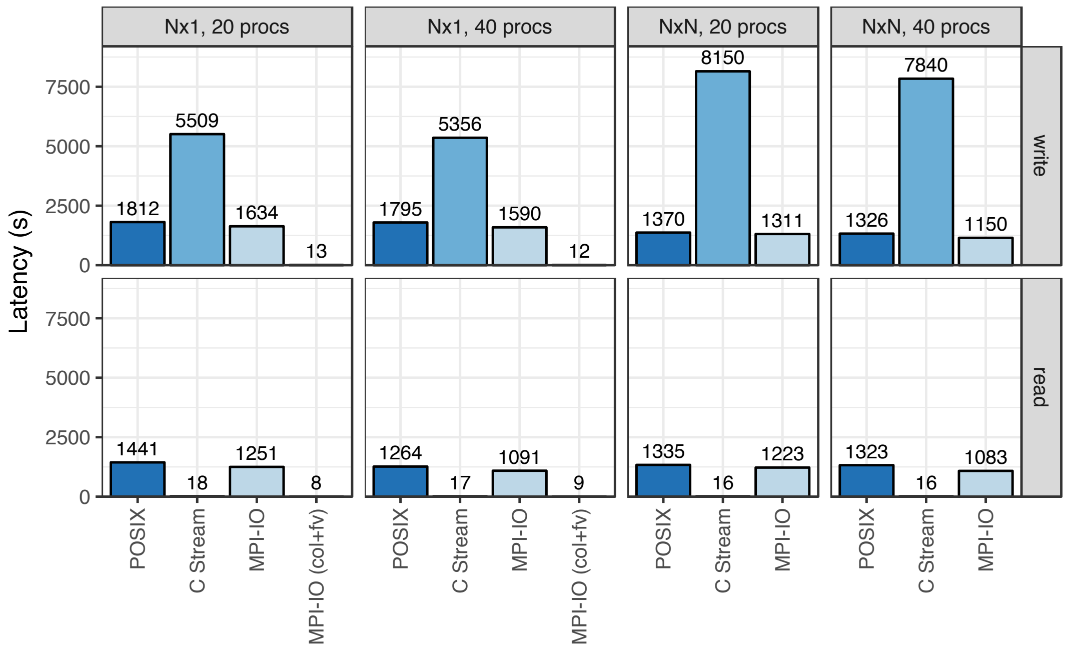


Figure 3. Average latency of read/write operations for a Cartesian dataset of 23 GiB, considering different I/O APIs, with 20 and 40 processes. Two file modes were evaluated: single shared file (Nx1) and file per process (NxN). Two MPI-IO options were also evaluated: collective I/O (col), and file view (fv).

Some interesting I/O performance behaviors can be observed through these results. First, the most notably result is the performance improvement observed with MPI-IO when file view and collective I/O options are used together (*i.e.,* MPI-IO (col+fv)) for either writing or reading the dataset to a shared file. Collective MPI-IO was in average 130 times faster than independent MPI-IO in write tests, and 135 times faster in read tests. These enhanced performance can be attributed to internal optimizations of the ROMIO MPI-IO implementation used by MPICH, which transforms and coordinates distributed data accesses in order to explore better performing I/O access patterns. In this particular experimental scenario, each process issues very small sized requests (*i.e.,* 376 bytes long), a well-known undesired access pattern because of its resulting I/O performance degradation. By exchanging request information, ROMIO can perform larger data transfers, better exploring the storage system throughput capabilities.

A very contrasting behavior is observed with C standard stream I/O functions. While write performance was the worst among all results, being 3 to 7 times slower than POSIX and independent MPI-IO, C stream read performance was considerably better, being 65 to 85 times faster than POSIX and independent MPI-IO. The improved performance of read tests with C stream is a result of its buffered I/O and prefetching algorithms. At each read request, a larger part of the dataset is retrieved from the storage system, which causes subsequent request to be serviced from data cached locally. It is worth mentioning that, even though read tests happens after write tests in IORE, the `read_reorder_offset` parameter was set to 1 at all tests, which mean that a process

i reads the data transferred by the $i - 1$ process at each run, focusing on avoiding the impact of write cached data in read test results. Regarding write performance, these same features can have imposed an extra cost on I/O, degrading the overall performance.

Less expressive results were observed with POSIX and independent MPI-IO AFIO implementation. Independent MPI-IO was in average between 4% to 18% faster than POSIX at all experiment runs, which can be attributed to enhanced operations provided by ROMIO support to ORANGEFS. Nonetheless, without any other particular improvement, such as collective operations and buffered I/O, both APIs demonstrate a degraded performance due to the considerably large number of small I/O requests.

6. Conclusions and Future Works

This paper contribution can be understood as a demonstration of some features provided by IORE, our proposal for a differentiated I/O performance evaluation tool, and how IORE can be used to assist in the analysis and improvement of the I/O performance of data-intensive scientific applications. Using IORE newly introduced dataset-based workload generation feature, an experimental study was conducted to analyze the I/O performance of writing and reading a large Cartesian dataset, whose parameters were inspired in a real-world application. Fourteen scenarios were evaluated, combining different I/O APIs and options, number of tasks, and file sharing modes.

In terms of usability, through a single and plain text experiment definition file, available at `https://doi.org/10.6084/m9.figshare.6376979`, IORE coordinated, executed, and exported results for all experiment runs and replications carried out in this research work. This feature alleviates the burden on users that, previously, had to prepare complex and error prone scripts and parsers to achieve the same result. Moreover, such approach favors cooperation, as experiment definitions can be easily shared and executed among different groups.

Results observed in this research work shed light on some I/O performance issues faced by real-world data-intensive scientific applications. Even for conventional and structured datasets, the performance of read and write operations can vary from dozens of seconds to hours, depending on the I/O API and options explored. Particularly, we demonstrated through this IORE experiment how MPI-IO can be highly efficient on dataset transferring using collective I/O and file view. Furthermore, results indicate that C standard stream I/O is a likely alternative for read operations, while it should be avoided for small writes. POSIX system calls and independent MPI-IO functions, on the other, presents a more stable performance, although higher average latency.

Although being part of an ongoing research work, initial results obtained with IORE are promising and indicate its potential contribution to the field. In short-term, we intend at including new features, such as support for particle datasets, and new AFIO implementations, such as HDF5, and NETCDF. Later, we intend to investigate other types of I/O workloads, such as those observed in Big Data and Internet of Things applications, and how they could be reproduced using IORE.

Acknowledgements

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`). We also would like to thank INESC P&D Brasil support.

References

Ackx, S. (2014). Emerging Technologies, Disrupt or be Disrupted. In Reimer, H., Pohlmann, N., and Schneider, W., editors, *ISSE 2014 Securing Electronic Business Processes*, pages 177–187. Springer, Wiesbaden.

ANL (2002). Parallel I/O Benchmarking Consortium. `http://www.mcs.anl.gov/research/projects/pio-benchmark/`.

Bell, G., Hey, T., and Szalay, A. (2009). Beyond the Data Deluge. *Science*, 323(5919):1297–1298.

Boito, F. Z., Inacio, E. C., Bez, J. L., Navaux, P. O. A., Dantas, M. A. R., and Denneulin, Y. (2018). A Checkpoint of Research on Parallel I/O for High-Performance Computing. *ACM Computing Surveys*, 51(2):1–35.

CERN (2016). Processing: What to record? `http://home.cern/about/computing/processing-what-record`.

Herbein, S., Ahn, D. H., Lipari, D., Scogland, T. R., Stearman, M., Grondona, M., Garlick, J., Springmeyer, B., and Taufer, M. (2016). Scalable I/O-Aware Job Scheduling for Burst Buffer Enabled HPC Clusters. In *HPDC '16 Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, pages 69–80. ACM Press.

Inacio, E. C., Barbetta, P. A., and Dantas, M. A. R. (2017a). A Statistical Analysis of the Performance Variability of Read/Write Operations on Parallel File Systems. *Procedia Computer Science - Special Issue: International Conference on Computational Science, ICCS 2017*, 108:2393–2397.

Inacio, E. C. and Dantas, M. A. R. (2018). IORE : A Flexible and Distributed I / O Performance Evaluation Tool for Hyperscale Storage Systems. In *ISCC '18 Proceedings of the IEEE Symposium on Computers and Communication*, page (to appear). IEEE.

Inacio, E. C., Nonaka, J., Ono, K., and Dantas, M. A. R. (2017b). Analyzing the I/O Performance of Post-Hoc Visualization of Huge Simulation Datasets on the K Computer. In *WSCAD '17 - Anais do XVIII Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 148–159. SBC.

Nonaka, J., Inacio, E. C., Ono, K., Dantas, M. A. R., Kawashima, Y., Kawanabe, T., and Shoji, F. (2018). Data I/O management approach for the post-hoc visualization of big simulation data results. *International Journal of Modeling, Simulation, and Scientific Computing*.

Roten, D., Cui, Y., Olsen, K. B., Day, S. M., Withers, K., Savran, W. H., Wang, P., and Mu, D. (2016). High-Frequency Nonlinear Earthquake Simulations on Petascale Heterogeneous Supercomputers. In *SC '16 Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE.

Shan, H., Antypas, K., and Shalf, J. (2008). Characterizing and predicting the I/O performance of HPC applications using a parameterized synthetic benchmark. In *SC '08 Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE.

Wang, F., Sim, H., Harr, C., and Oral, S. (2017). Diving into petascale production file systems through large scale profiling and analysis. In *PDSW-DISCS '17 Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems*, pages 37–42. ACM Press.