PIOSS: A Simulation Model for the Analysis of Parallel I/O Performance Variability on Large-scale Applications

Eduardo C. Inacio[1], **Mario A. R. Dantas**[2]

[1]Centro Universitário SENAI/SC
Florianópolis – SC – Brazil

[2]Universidade Federal de Juiz de Fora
Juiz de Fora – MG – Brazil

eduardo.inacio@edu.sc.senai.br, mario.dantas@ice.ufjf.br

Abstract. *To meet ever increasing capacity and performance requirements of emerging data-intensive applications, parallel file systems (PFSs) have been employed in large-scale computing environments. In such complex storage systems, the load distribution on PFS data servers compose a major source of input/output (I/O) performance variability. Albeit mitigating such variability is desirable, understanding its sources and behavior remains a challenging task. In this research work, a differentiated approach for evaluating the parallel I/O performance variability perceived by large-scale applications is proposed. The Parallel I/O and Storage System (PIOSS) simulation model represents main components and mechanisms observed in typical PFS implementations and enables fast evaluations of large and complex scenarios. Experimental results presented in this paper demonstrate PIOSS can accurately reproduce the load balance on PFS data servers, with a confidence level of 95%.*

1. Introduction

Digital data production and consumption is increasing at unprecedented rates. By 2025, approximately 175 ZiB (*i.e.* 175×10^{12} GiB) of data is expected to be created, captured, or replicated globally [Reinsel et al. 2018]. In light of such data deluge, data-intensive scientific discovery grows even more in importance [Hey et al. 2009], which poses great challenges for the high performance computing (HPC) field.

To cope with such capacity and performance demands, most HPC environments present distributed, multilayered, and deep file input/output (I/O) paths [Inacio et al. 2017b]. A particularly relevant layer in such environments is the scratch file system, which is responsible for absorbing bulk data transfers from large-scale concurrent applications running on compute nodes [Boito et al. 2018]. The scratch file system is traditionally implemented through a parallel file system (PFS). In most PFSs, files are divided into multiple contiguous chunks, which are distributed across a cluster of data servers, a process known as file striping [Corbett and Feitelson 1996].

Although providing a high data rate, as read and write requests can be served by multiple data servers in parallel, file striping can be a significant source of I/O performance variability [Inacio et al. 2017a]. File chunks are distributed in a random or semi-random fashion across data servers, which implies the adopted file distribution method affects directly the load balance on the PFS. Also, as most large-scale data-intensive applications present a synchronous I/O phase [Song et al. 2011], this potential imbalance on PFS data servers' load can translate into a huge variability in the I/O performance perceived by the application across distinct I/O phases.

Many parameters can affect the I/O performance perceived by an application in such complex computing environments [Inacio et al. 2017a]. This variability effect, inherent to PFSs, makes finding optimal values for such parameters even more challenging. Performing experimental evaluations on target computing environments can provide very accurate results. However, this might become a daunting and time consuming approach, due to the large number of parameters involved and their possible values. In such conditions, simulation approaches present themselves as viable solutions.

Despite the existence of some PFS simulation models, to the best of our knowledge, none was designed with the purpose of investigating the load balance on data servers, neither favor examining its impact on the I/O performance variability perceived by applications. Towards addressing that particular issue, this research work proposes the Parallel I/O and Storage System (PIOSS) simulation model, a light-weight approach for analyzing the parallel I/O performance variability perceived by large-scale data-intensive applications over PFSs. Through a moderate level of detail, PIOSS enables fast evaluation of spatial-related aspects affecting I/O performance and variability provided by traditional PFS designs.

The remaining of this paper is organized as follows. Section 2 provides further details about file I/O on PFSs. In Section 3, related works on PFS simulation are examined. The PIOSS simulation model, proposed in this research work, is presented in Section 4, with experimental evaluation results discussed in Section 5. Finally, Section 6 concludes this paper with our final considerations and future directions of this research work.

2. File I/O on Parallel File Systems

Most large-scale HPC applications alternate between very distinguishable execution phases [Yu et al. 2017]. During the compute phase, application's processes are busy processing data and communicating with each other, whereas the I/O phase is dominated by processes reading or writing data from or to files in the back-end storage system. For most applications, the I/O phase is synchronous, as each process blocks until all other processes complete the I/O phase [Song et al. 2011].

Data access parallelism can be achieved in different ways at the application level. Most adopted approaches include the N-1 and N-N process-to-file mappings [Son et al. 2017]. In the N-1 process-to-file mapping, all application processes transfer data from/to a single file. This approach favors data sharing among processes, but may incur in performance degradation due to serialization of concurrent accesses. In the N-N process-to-file mapping, each application process transfers data from/to an individual and independent file. This approach is easier to implement and avoids serialization drawbacks, but may pose a great pressure on the metadata service of the back-end storage system as the number of files increases.

The back-end storage system is usually implemented through a PFS, such as OrangeFS and Lustre [Boito et al. 2018]. Although differences in design and implementation can be observed, in general, PFSs present a cluster architecture composed of three main components: clients, data servers, and metadata servers. *Clients*, commonly located at compute nodes, provide the interface to the file system namespace. *Data servers* are responsible for storing files contents, while *metadata servers* keep up-to-date information about files (*e.g.* location, name, timestamps, owner, permissions, etc).

In order to provide high data rate, especially for large I/O requests, most PFSs stripe file data across multiple data servers, a technique known as file striping [Corbett and Feitelson 1996]. An example of the file striping technique is presented in Figure 1. In this example, a write of 400 KiB is requested by the application. The PFS client, upon receiving the request, divides the request data into fixed-size stripe fragments, and distributes them across a subset of the available data servers. This way, the PFS can serve a single I/O request from multiple data servers in parallel, leveraging the aggregated throughput to improve the data transfer rate.

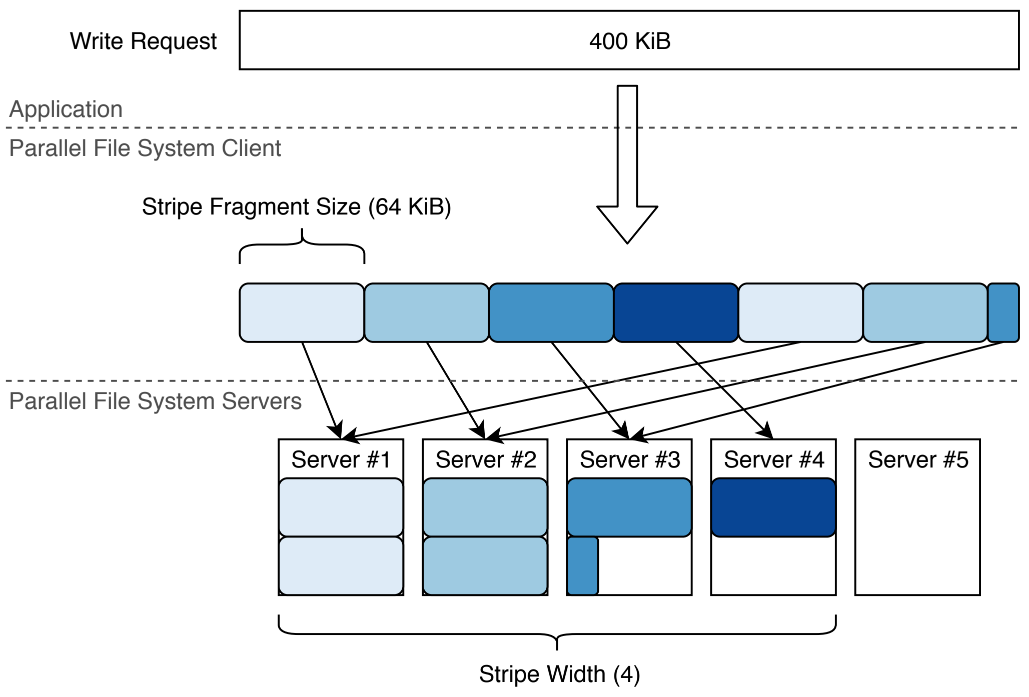

Figure 1. A simple example of the file striping technique of a PFS.

The subset of data servers is selected for a given file according to the file distribution method defined for the PFS. Typical approaches, adopted by most popular PFSs, such as OrangeFS and Lustre, include the *random* method, in which a subset of data servers is randomly selected, and the *round-robin* method, in which the first data server is randomly selected and the remaining data servers of the subset are taken sequentially according to the PFS configuration. The size of the subset of data servers is also a parameter of the PFS, known as the *stripe width*.

3. Related Works

Some PFS simulation models have been proposed in previous research works. Most of the existing simulation models focus on a specific aspect of the hardware and software architecture from such high performance storage systems. The detailing level of elements and processes involved also varies from model to model.

IMPIOUS [Molina-Estolano et al. 2009] is a trace-driven PFS simulator whose design is centered on fundamental features, including data placement, replication, caching strategies, and locking disciplines, through simple and abstract components. HECIOS [Settlemyer 2009] is a trace-driven simulator designed to model the PVFS file system [Carns et al. 2000]. Developed over the OMNeT++ discrete-event simulation framework [Varga and Hornig 2008], HECIOS explores a detailed network model. SIMCAN [Núñez et al. 2012] is a trace-driven storage network simulation platform built upon the OMNeT++ framework that models different layers of the I/O system, such as the virtual file system, the volume manager, the block scheduler, and disk drive modules. Another simulator based on the OMNeT++ framework is the PFSSim [Liu et al. 2013]. PFSSim main purpose is the efficient evaluation of I/O scheduling algorithms. FileSim [Erazo et al. 2012] is a simulator developed specifically for the evaluation of end-to-end I/O performance of HPC systems. Components modeled by FileSim include clients, metadata servers, object storage devices, and the interconnection network. CODES [Cope et al. 2011] is a simulation framework, developed over the parallel discrete-event simulation framework ROSS [Carothers et al. 2002], focused on the evaluation of exascale storage system designs. Finally, HPIS3 [Feng et al. 2014] is a trace-driven simulator, also developed over the ROSS framework, particularly designed for the evaluation of hybrid PFSs (*i.e.* with both HDD-based and SSD-based data servers).

Although not explicitly stated, it is reasonable to infer that some of the aforementioned PFS simulators may represent I/O performance variability by modeling some sort of stochastic process. In the context of parallel I/O performance variability analysis, a relevant, and usually random, process is the file distribution method, as it is directly related to the load balance in data servers [Inacio et al. 2017a]. None of the previously referred proposals is focused on evaluating the impact of this mechanism on the PFS performance, although most of them provide a simple file distribution implementation, usually, based on the random method. Furthermore, many of the state-of-the-art simulators provide a high level of detail, which, usually, results in complex parametrization, increased computational resources demands, and long time to results.

4. The PIOSS Simulation Model

The Parallel I/O and Storage System (PIOSS) simulation model is a differentiated proposal devised for reproducing and evaluating the parallel I/O performance perceived by large-scale data-intensive applications executing over general PFS designs. Through a moderate level of detail, PIOSS enables fast evaluation of spatial-related aspects affecting I/O performance and variability. PIOSS light-weight design aims at improved execution performance for simulating extreme-scale applications and environments.

4.1. Modeling Approach

Main components and mechanisms observed in popular PFS implementations are modeled with a moderate level of detail in PIOSS. This detailing level was carefully chosen focusing on modeling most fundamental performance behaviors, sacrificing absolute timing accuracy in favor of generality and simulation performance. Figure 2 provides an overview of simulated PFS components and their interplay as modeled in PIOSS.

The *CLI* component models PFS clients. It is responsible for generating application workload, striping files, and transferring their contents to PFS data servers. PFS

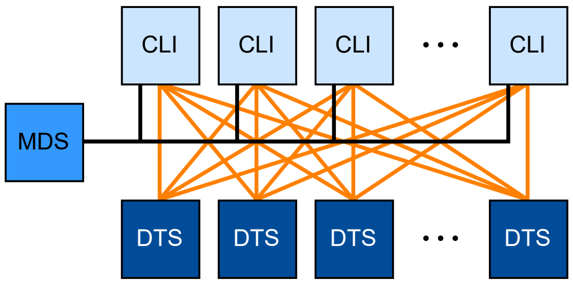

Figure 2. Main PFS components modelled by the PIOSS simulation model.

metadata servers are modeled by the *MDS* component and its main purpose in PIOSS is defining which data servers are allocated for each file created by the CLI component. Finally, the *DTS* component models PFS data servers, keeping the amount of data stored in each data server for a simulated scenario.

The simulation workflow in PIOSS is, in this primary version, sequential. For each PFS client defined for the simulated scenario, the CLI component requests a file creation for the MDS component. The MDS component, then, selects the PFS data servers to host the file according to the stripe width and file distribution method. In scenarios with N-1 process-to-file mapping, the MDS component returns the same set of data servers to every CLI request, as clients share the same file. For scenarios with N-N mapping, most probably, distinct sets of data servers are returned by the MDS component for each CLI request. After that, the CLI component simulates a file contents division according to the defined stripe fragment size. At that point, simulated file chunks are distributed across data servers through the CLI component sending to the DTS component the data server id and the size of the respective simulated file chunk. Finally, the DTS component accumulates the amount of data (in bytes) stored in each simulated data server for later reporting the load distribution in the PFS. Simulation scenarios can be defined through a set of parameters supported by PIOSS. Table 1 lists these parameters and their meaning.

Table 1. PIOSS simulation parameters.

Parameter	**Description**
`num_cli`	number of simulated PFS clients
`data_size`	bytes transferred per simulated PFS client
`shared`	sets a N-1 process-to-file mapping (N-N mapping, otherwise)
`num_dts`	number of simulated PFS data servers
`stripe_width`	number of simulated PFS data servers per file
`stripe_size`	size of a simulated file chunk
`file_dist`	file distribution method
`rng_seed`	seed for the random number generator
`out_path`	path for storing simulation results report

4.2. Implementation Details

PIOSS is implemented entirely in C. Figure 3 presents an overview of its internal software structure. As can be observed, PIOSS implementation consists of three modules.

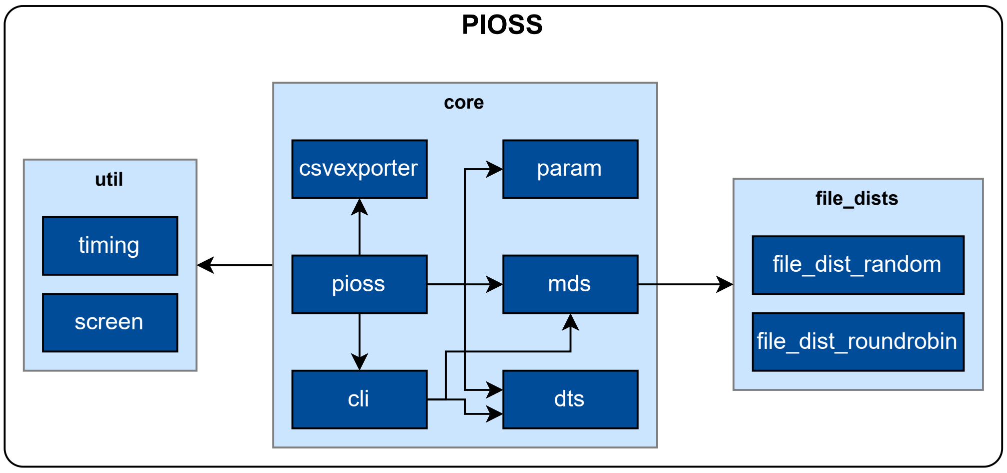

Figure 3. Overview of PIOSS internal software components.

The *core* module contains the main components of the simulation model. It is in the core module that the CLI, MDS, and DTS components are implemented. Also, the core module hosts the *param* component, which is responsible for parsing user-provided simulation parameters, and the *csvexporter* component, which enables exporting simulation results in a CSV file format. This is a particularly interesting feature that allows analysing simulation results in a varying of tools, from digital spreadsheets to more sophisticated data science workflows. Finally, the *pioss* component initializes simulation components and coordinates the simulation workflow.

Another important module in PIOSS is the *file_dists* module. This module hosts the file distribution methods supported by the PIOSS simulation model. Currently, two methods are implemented in PIOSS, namely, the random and the round-robin file distribution methods (*i.e.* *file_dist_random* and *file_dist_roundrobin* components, respectively). These methods were implemented in this primary version of PIOSS because they are adopted in most HPC environments and popular PFSs, such as OrangeFS and Lustre. As stated in Section 4.1, file distribution methods are employed by the MDS component at file creation requests to select the set of data servers that will store each file contents. Considering the impact of the file distribution method in the load balance on PFS data servers and, consequently, its relevance to the analysis of the I/O performance variability perceived by an application, the *file_dists* module was designed for flexibility. A *file_dist* struct is defined, specifying the standard structure for a file distribution method to be integrated in PIOSS. Such approach favors the research and evaluation of new file distribution methods, which can be implemented and integrated with PIOSS with minor effort.

Lastly, the *util* module provides auxiliary functions for the simulation execution. The *timing* component, as the name suggests, deals with execution time accounting and date and time formatting. The *screen* component, on the other hand, provide helper functions for displaying PIOSS messages in the standard output.

4.3. Positioning in the State-of-the-art

The moderate level of detail allows PIOSS to focus on most fundamental performance effects. The modularity of PIOSS implementation provides flexibility for developing and evaluating different PFS designs and mechanisms. For instance, different PFS file distribution methods can be easily included in PIOSS by implementing new *file_dist* components, which can be selected at simulation run time. Finally, by exporting collected results into CSV files, PIOSS facilitates the evaluation of simulation results, as this format can be consumed by a variety of analysis tools.

Compared to state-of-the-art proposals for PFS simulation, considering the level of detail of the models, PIOSS differs significantly from previous approaches based on the OMNeT++ simulation framework [Settlemyer 2009, Núñez et al. 2012, Liu et al. 2013]. These related works rely on the high-fidelity network models provided by OMNeT++, focusing on time-related analysis and accuracy. Although this approach does not prevent supporting the analysis of the I/O performance variability, such level of detail regarding the network communication does not contribute to the referred effect, adding unnecessary complexity and computational power demands nonetheless. In terms of generality, most of the state-of-the-art proposals models a specific PFS implementation or environment. PIOSS, as well as IMPIOUS [Molina-Estolano et al. 2009] and SIMCAN [Núñez et al. 2012], provide simulation models aiming at evaluating general PFS designs. Finally, a practical concern refers to the code availability of the existing PFS simulation models. Only HECIOS [Settlemyer 2009], SIMCAN [Núñez et al. 2012], and PFSSim [Liu et al. 2013] are, at the moment this paper was elaborated, available for usage. It worth mentioning that none of the aforementioned publicly available simulation models, at their current stage of development, provide sufficient results for the analysis of the load distribution effects on the parallel I/O performance of PFSs and, thereby, prevent a fair comparison with PIOSS. The current prototype version of the PIOSS simulation model is publicly available at the following GitHub repository: `http://github.com/ecamiloinacio/pioss`.

5. Experimental Environments and Results

In order to verify the fidelity of the PIOSS simulation model, with respect to the load distribution on PFS data servers, an experimental evaluation was performed, comparing simulated results with measurements from a real environment. A full factorial experimental design was adopted, combining all values for the parameters listed in Table 2. As a result, 144 distinct experimental scenarios were evaluated in both simulated and real environments. For more reproducible and statistically sound conclusions, the entire experiment was replicated five times, with experimental scenarios being evaluated in a random order in each replication.

PIOSS simulations were executed in a local shared-memory machine, while an experimental environment representative of real-world configurations, using OrangeFS 2.9.7, was deployed on *grimoire* and *grisou* clusters, from the Grid'5000 testbed. Each of the 59 cluster nodes consists of 2× Intel Xeon E5-2630 v3 (8 cores/CPU), 128 GiB RAM, 200 GB SSD, 5× 600 GB HDD, 4× 10 Gbps Ethernet, and 56 Gbps Infiniband. A CentOS 7 (Linux kernel 3.10.0) image, with MPICH 3.2.1, was deployed over all nodes. For this experimental evaluation, 8 nodes of the *grimoire* cluster were set up as PFS data servers, while 32 nodes from the *grisou* cluster were defined as compute nodes. I/O workload was generated using the IOR-Extended (IORE) [Inacio and Dantas 2018], a scalable and distributed tool especially designed for experimental analysis of parallel I/O performance.

Table 2. Parameters and values considered in PIOSS evaluation.

Parameter	**Value**
Application Workload	
Number of processes	8, 32, 128
Process-to-file mapping	N-1, N-N
Data size per process	32 MiB
PFS Configuration	
Stripe fragment size	64 KiB, 256 KiB, 1 MiB
Stripe width	1, 2, 4, 8
File distribution method	random, round-robin

Figure 4 presents a comparison of histograms of the load distribution on PFS data servers simulated with PIOSS and observed on the OrangeFS environment for experimental scenarios comprising applications with N-1 process-to-file mapping and a PFS configured with a stripe fragment size of 64 KiB. Each cell of the plot grid refers to a distinct experimental scenario, regarding the number of processes and the stripe width, and combines histograms from PIOSS and OrangeFS results for both random and round-robin file distribution methods. The bin width of histograms for 8, 32, and 128 processes were set to 32, 128, and 512, respectively, for improved readability. The x-axis denotes the number of bytes per data server in MiB, while the y-axis denotes the number of occurrences in the respective bin of the x-axis.

The accuracy of PIOSS simulations for N-1 mapping scenarios is noticeable in Figure 4. In all the evaluated experimental scenarios, including those with different stripe fragment sizes, not presented in this paper for the sake of space, PIOSS provided the exact load distribution on data servers. This accuracy can be mostly attributed to the precise implementation of PFSs file distribution methods on PIOSS, and the deterministic behavior of the load distribution on scenarios with N-1 process-to-file mapping. For instance, for scenarios with a stripe width of 1, a single data server receives all the load, while other remain idle. On the other hand, for scenarios with a stripe width of 8, the load is evenly distributed across all data servers of this experimental environment.

Results comparing the variability of the load distribution on PFS data servers simulated with PIOSS and observed on the OrangeFS environment, considering applications with N-N process-to-file mapping, are presented in Figure 5. Each cell of the plot grid combines box plots for both random and round-robin file distribution methods, and refers to a different experimental scenario with respect to the stripe fragment size and the number of processes. The x-axis denotes the stripe width, while the y-axis denotes the number of bytes per data server in MiB. Boxes enclose 50% of the observed data, with the inner horizontal line denoting the median value. Upper and lower whiskers refer to data within 1.5 times the interquartile range (IQR) from, respectively, the third and first quartiles. Small circles beyond whiskers refer to outliers.

As can be observed in Figure 5, results demonstrate the high accuracy of PIOSS simulations for experimental scenarios with N-N mapping. Not only the median of simulated and real scenarios are very similar for each scenario, the variance is also comparable.

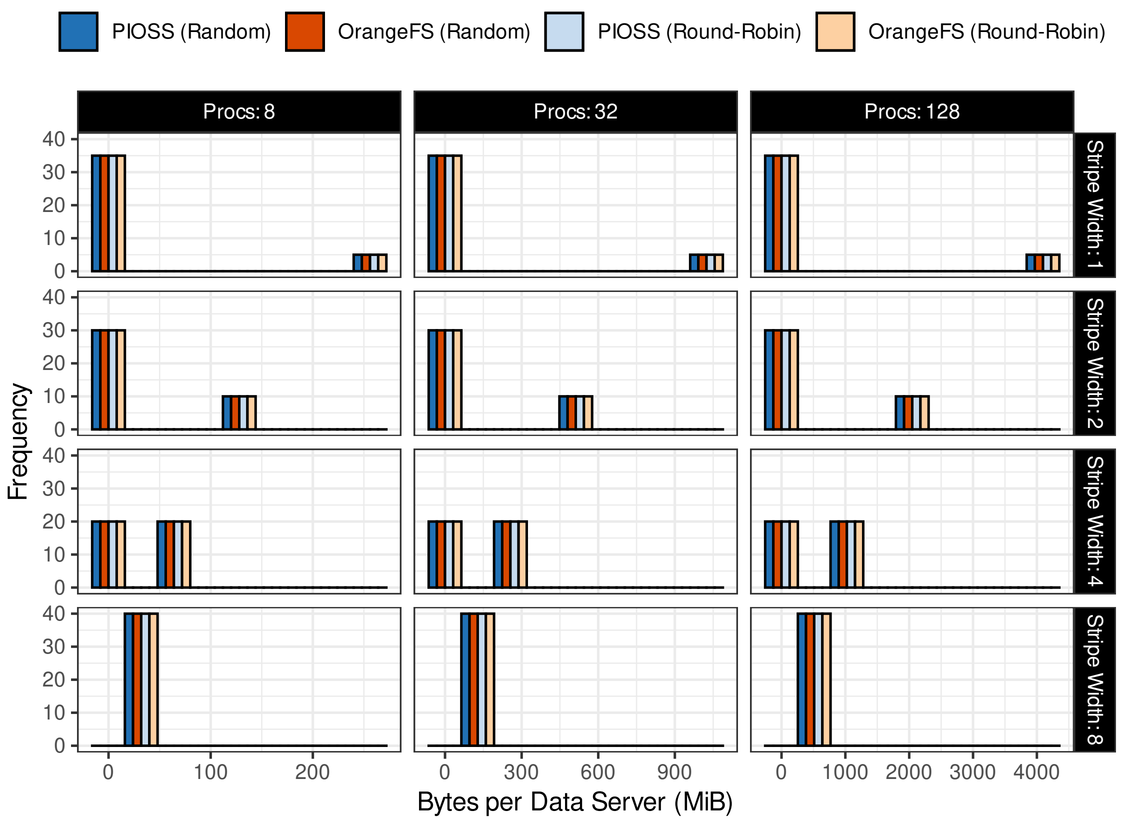

Figure 4. Comparison of histograms of the load distribution on PFS data servers simulated with PIOSS and observed on an OrangeFS environment, considering applications with N-1 process-to-file mapping and a PFS configured with a stripe fragment size of 64 KiB.

Furthermore, the characteristic reduction of variance with the increase of the stripe width is successfully represented by PIOSS. It is worth noting that, for all evaluated experimental scenarios, when the stripe width is equal to the number of data servers available (*i.e.*, 8, for this experimental environment), the load is evenly distributed across data servers. Thus, in such conditions, there is no variance with respect to data servers load, which, thereby, explain the flat boxes observed in respective plots.

Complementing the visual interpretation of the accuracy of the PIOSS simulation model, an analysis of variance (ANOVA) was carried out over experimental results. The ANOVA F-test indicates, for the evaluated experimental scenarios, that no statistically significant difference is observed between PIOSS simulated results and real measurements from the OrangeFS environment, with a confidence level of 95%. Based upon this analysis, it is possible to conceive that PIOSS is capable of simulating the load distribution on PFSs data servers with statistically significant accuracy.

Although experimental environments and scenarios evaluated in this research work can be considered smaller in scale than real-world HPC deployments, we argue that PIOSS can provide accurate results even for yet non-evaluated huge-scale scenarios. The adopted evaluation method and achieved results, combined with the precise representation of the impact of most relevant environment and workload factors on the load

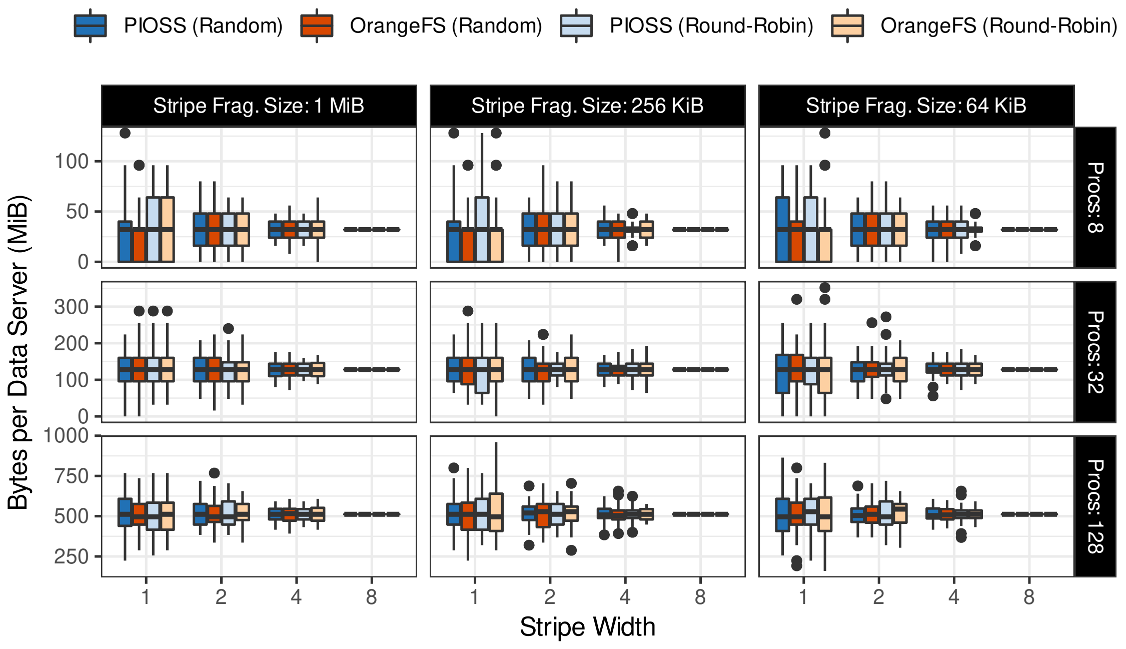

Figure 5. Comparison of box plots of the load distribution on PFS data servers simulated with PIOSS and observed on an OrangeFS environment, considering applications with N-N process-to-file mapping.

distribution on PFSs data servers, can be considered to support that claim.

6. Conclusions and Future Works

In large-scale HPC environments, where PFSs are the main component of the back-end storage system, variability has a considerable impact in the I/O performance perceived by data-intensive applications. Dealing with such variability is a complex and challenging task, considering the number of elements with potential impact in the I/O performance variability and the uncertainty inherent to random processes part of the system design. In the presence of such challenges, this research work proposes the PIOSS simulation model, a light-weight approach for fast analysis of parallel I/O performance variability perceived by large-scale data-intensive applications on PFS-based back-end storage systems.

The PIOSS simulation model was intended for modeling main conditions and mechanisms of typical PFSs affecting the I/O performance perceived by applications. Fast evaluation of spatial-related aspects affecting the I/O performance on PFSs is provided by PIOSS through an efficient implementation combined with a moderate level of detail. Furthermore, the modular design of PIOSS favors its extension, in a way it can be used for easily investigating different, or even new, methods and techniques applied to any component of a general PFS design.

The accuracy of PIOSS was evaluated through a number of experiments, comparing simulated results to measurements from real environments deployed over clusters from the Grid'5000 testbed. After evaluating a total of 144 experimental scenarios, considering different file distribution methods, number of processes, process-to-file mappings, and PFS configurations, it was demonstrated that PIOSS can accurately estimate the load distribution on PFS data servers with a confidence level of 95%. For scenarios with applications adopting a N-1 process-to-file mapping, experimental results demonstrated PIOSS replicates precisely the load distribution observed in real environments.

PIOSS is an ongoing effort, with code development in progress. Despite promising results presented in this paper, in short-term, we expect to conclude the code development, including orthogonal requirements and documentation. Moreover, we intend to elaborate on PIOSS evaluation, considering larger workloads and experimental environments. In long-term, we foresee the potential for additional features, such as supporting other file distribution methods. We expect that such extensions could make PIOSS a helpful tool for fast investigation of the impact of user-defined configurations on the parallel I/O performance provided by PFSs to large-scale data-intensive applications.

Acknowledgements

This research work was conducted while Eduardo C. Inacio was pursuing his Doctoral degree at Universidade Federal de Santa Catarina (UFSC), and was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. 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`).

References

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*, 51:1–35.

Carns, P. H., Walter B. Ligon, I., Ross, R. B., and Thakur, R. (2000). Pvfs: A parallel file system for linux clusters. In *ALS'00 Proceedings of the 4th annual Linux Showcase & Conference*, volume 4, pages 317–328. USENIX Association.

Carothers, C. D., Bauer, D., and Pearce, S. (2002). Ross: A high-performance, low-memory, modular time warp system. *Journal of Parallel and Distributed Computing*, 62:1648–1669.

Cope, J., Liu, N., Lang, S., Carns, P., Carothers, C. D., and Ross, R. B. (2011). Codes: Enabling co-design of multi-layer exascale storage architectures. In *WEST '11 Proceedings of the Workshop on Emerging Supercomputing Technologies 2011*, pages 303–312.

Corbett, P. F. and Feitelson, D. G. (1996). The vesta parallel file system. *ACM Transactions on Computer Systems*, 14:225–264.

Erazo, M. A., Li, T., Liu, J., and Eidenbenz, S. (2012). Toward comprehensive and accurate simulation performance prediction of parallel file systems. In *DSN '12 Proceedings of the 2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 1–12. IEEE.

Feng, B., Liu, N., He, S., and Sun, X.-H. (2014). Hpis3: Towards a high-performance simulator for hybrid parallel i/o and storage systems. In *PDSW '14 Proceedings of the 9th Parallel Data Storage Workshop*, pages 37–42. IEEE.

Hey, T., Tansley, S., and Tolle, K. (2009). *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research.

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*, pages 1026–1031. 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.

Liu, Y., Figueiredo, R., Xu, Y., and Zhao, M. (2013). On the design and implementation of a simulator for parallel file system research. In *MSST '13 Proceedings of the IEEE 29th Symposium on Mass Storage Systems and Technologies*, pages 1–5. IEEE.

Molina-Estolano, E., Maltzahn, C., Bent, J., and Brandt, S. A. (2009). Building a parallel file system simulator. *Journal of Physics: Conference Series*, 180:1–7.

Núñez, A., Fernández, J., Filgueira, R., García, F., and Carretero, J. (2012). Simcan: A flexible, scalable and expandable simulation platform for modelling and simulating distributed architectures and applications. *Simulation Modelling Practice and Theory*, 20:12–32.

Reinsel, D., Gantz, J., and Rydning, J. (2018). The digitization of the world from edge to core. Technical report, IDC.

Settlemyer, B. W. (2009). *A Study of Client-Based Caching For Parallel I/O*. PhD thesis, Clemson University.

Son, S. W., Sehrish, S., Liao, W.-K., Oldfield, R., and Choudhary, A. (2017). Reducing i/o variability using dynamic i/o path characterization in petascale storage systems. *The Journal of Supercomputing*, 73:2069–2097.

Song, H., Yin, Y., Sun, X.-H., Thakur, R., and Lang, S. (2011). Server-side i/o coordination for parallel file systems. In *SC '11 Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM Press.

Varga, A. and Hornig, R. (2008). An overview of the omnet++ simulation environment. In *Simutools '08 Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Yu, J., Liu, G., Dong, W., Li, X., Zhang, J., and Sun, F. (2017). On the load imbalance problem of i/o forwarding layer in hpc systems. In *ICCC '17 Proceedings of the 3rd IEEE International Conference on Computer and Communications*, pages 2424–2428. IEEE.