I/O Performance of the Santos Dumont Supercomputer

Jean Luca Bez, André Ramos Carneiro, Pablo J Pavan, Valéria Soldera Girelli, Francieli Zanon Boito, Bruno Alves Fagundes, Carla Osthoff, Pedro Leite da Silva Dias, Jean-François Méhaut, Philippe O.A. Navaux

I/O Performance of the Santos Dumont Supercomputer

Journal Title
XX(X):1–17

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Jean Luca Bez[1], André Ramos Carneiro[2], Pablo José Pavan[1], Valéria Soldera Girelli[1], Francieli Zanon Boito[3], Bruno Alves Fagundes[2], Carla Osthoff[2], Pedro Leite da Silva Dias[4], Jean-François Méhaut[3] and Philippe O. A. Navaux[1]

Abstract

In this paper, we study the I/O performance of the Santos Dumont supercomputer, since the gap between processing and data access speeds causes many applications to spend a large portion of their execution on I/O operations. For a large-scale, expensive, supercomputer, it is essential to ensure applications achieve the best I/O performance to promote efficient usage. We monitor a week of the machine's activity and present a detailed study on the obtained metrics, aiming at providing an understanding of its workload. From experiences with one numerical simulation, we identified large I/O performance differences between the MPI implementations available to users. We investigated the phenomenon and narrowed it down to collective I/O operations with small request sizes. For these, we concluded the customized MPI implementation by the machine's vendor (used by more than 20% of the jobs) presents the worst performance. By investigating the issue, we provide information to help improve future MPI-IO collective write implementations, and practical guidelines to help users and steer future system upgrades. Finally, we discuss the challenge of describing applications I/O behavior without depending on information from users. That allows for identifying the applications I/O bottlenecks and proposing ways of improving its I/O performance. We propose a methodology to do so, and use GROMACS, the application with the largest number of jobs in 2017, as a case study.

Keywords

High-Performance Computing, Supercomputer, Storage, Parallel I/O, Workload Characterization

1 Introduction

Applications that execute on High-Performance Computing (HPC) infrastructures — large-scale clusters or supercomputers — often need to input or output data. This is usually accomplished by performing I/O operations to a Parallel File System (PFS), such as Lustre (SUN 2007) or GPFS (Schmuck and Haskin 2002). The PFS is deployed over a set of dedicated machines that act as metadata or data servers. Files are separated into fixed-size chunks and distributed across data servers through an operation called "data striping". High performance is achieved by allowing each client to access chunks from different servers in parallel. Nonetheless, I/O operations represent a bottleneck for an increasing number of applications due to the speed difference between computation and data access, as the latter depends on slower components like disks and the network.

Since I/O performance is a limiting factor for many scientific applications, in this paper, we evaluate the I/O infrastructure of the *Santos Dumont* supercomputer (*SDumont*). It is a Bull/Atos machine, located at the National Laboratory for Scientific Computing (LNCC) in Brazil. With a total of $18,144$ cores, it is one of the largest supercomputers from Latin America, acquired through an investment from the Brazilian government of approximately 60 million dollars (Crouch and ATOS 2015). Considering the high financial costs associated with acquiring and maintaining a supercomputer, the efficient usage of the machine is of paramount importance. Furthermore, for its users, achieving high performance while getting the required data from executions is essential. Therefore, our study provides valuable information that can be taken into consideration by the administrators to promote efficient system usage and to guide future upgrades.

We present our study in three main parts. In the first part, we study the I/O workload of the SDumont. We obtained traces for the machine's activity during a week, and investigate them aiming to identify bottlenecks the applications could encounter when issuing their requests, and consequently, to propose improvements to the system. From our analysis, we identify scientific applications and access patterns of high interest to the system.

In the second part, we focus on the performance of collective I/O operations in the machine. That was motivated by the observation of large I/O performance

[1] Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil
[2] Laboratory for Scientific Computing (LNCC), Petrópolis, Brazil
[3] Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, 38000 Grenoble, France
[4] Institute of Astronomy, Geophysics and Atmospheric Sciences, University of São Paulo (USP), São Paulo, Brazil

Corresponding author:
Jean Luca Bez, Instituto de Informtica – Universidade Federal do Rio Grande do Sul (UFRGS) – Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil
Email: jean.bez@inf.ufrgs.br

differences between MPI implementations available to users, which we replicated and investigated. We found this difference comes from the performance of small collective write operations, and that is due to some implementations having poor performance for asynchronous communications between pairs of processes. The customized MPI implementation by Bull (used by more than 20% of the jobs) presented the worst I/O performance. By investigating the reported phenomenon, we provide information that can be used to improve future MPI-IO implementations. Additionally, our conclusions provided guidelines so SDumont users can achieve higher I/O performance. Finally, our findings can also be applied to other similar machines.

In the third and final part, we propose a methodology for characterizing jobs I/O behavior. That was motivated by our experience in the second part when we learned an application was having a performance issue, but identifying and understanding the issue required extensive work. That happens because users are often not from the HPC domain, and even when they are, they are often not the developers of the applications, hence detailing their I/O behavior is not an easy task. To tackle this issue, we propose a technique to characterize the application I/O phases from coarse-grained aggregated traces and apply our technique to GROMACS (Pronk et al. 2013). We chose GROMACS as a case study because it was the most executed application in the SDumont in 2017.

Additionally, **we are making all the data we collected and analyzed in this paper freely available**[*]. Traces from real large-scale systems are seldom available, and we do so hoping to to encourage researchers to use this data to build new contributions.

In summary, **the main contributions of our work are**:

- we present a detailed analysis of the I/O workload of a medium-size production supercomputer, the SDumont;
- we study the I/O performance of two production scientific applications: an atmospheric application and a molecular dynamics application;
- we document an existing issue observed when using small MPI collective I/O operations, and provide details about it to help improve future MPI implementations;
- we provide guidelines to the SDumont users to help them achieve higher I/O performance, and those guidelines could be applied to other similar machines;
- we propose a methodology to identify and characterize common I/O behaviors of a job from coarse-grained aggregated traces;
- we made data collected from monitoring the production machine publicly available.

The remainder of this paper is organized as follows. Section 2 describes the Santos Dumont supercomputer. Sections 3, 4, and 5 present the three parts of our study about the machine's I/O performance. Related work is presented in Section 6. Finally, Section 7 summarizes our findings and discusses future work.

2 The Santos Dumont Supercomputer

The SDumont supercomputer, located at the LNCC in Brazil, has a total of 18,144 CPU cores. All 756 compute nodes have two Intel Xeon E5-2695v2 Ivy Bridge 2.4GHz 12-core processors, 64GB DDR3 RAM, and one 128GB SSD. There are three types of nodes:

- 504 B710 (regular) compute nodes;
- 198 B715 with two K40 GPUs each;
- 54 B715 with two Xeon Phi KNC co-processors each.

Compute, login, and storage nodes are connected through Infiniband FDR ($56Gb/sec$) on a fat-tree full-nonblocking topology. The Lustre parallel file system version 2.1 is deployed through the Xyratex/Seagate ClusterStor 9000 v1.5.0, with one MDS (Metadata Server) and 10 OSS (Object Storage Service), each with one OST (Object Storage Target), for a total storage capacity of 1.7 PB. Clients use the version 2.4.3 and mount the file system with the *flock* option. According to the Bull/ATOS technical specification, the maximum aggregate throughput the Lustre file system should achieve, without considering the effects of cache, is 30 GB/s. This performance is limited to the SAS link between the OSSs servers and the disk enclosure (3 GB/s for each OSS). The aggregate network access bandwidth to the Lustre system is 70 GB/s, which would not impose limitations on the overall storage performance.

Since 2016, SDumont is available to the Brazilian research community. Brazilian researchers, with relevant projects that demand high processing performance, can apply to use the SDumont computational resources (MCTIC-LNCC 2016). Currently, there are 135 ongoing scientific projects and over 800 users from 16 research areas. The areas with more projects are Chemistry, Physics, Engineering, Biology, and Computing Science. From August 2016 to the end of December 2018, 253,678 jobs were submitted to the machine.

3 Part I – Study of the I/O Workload

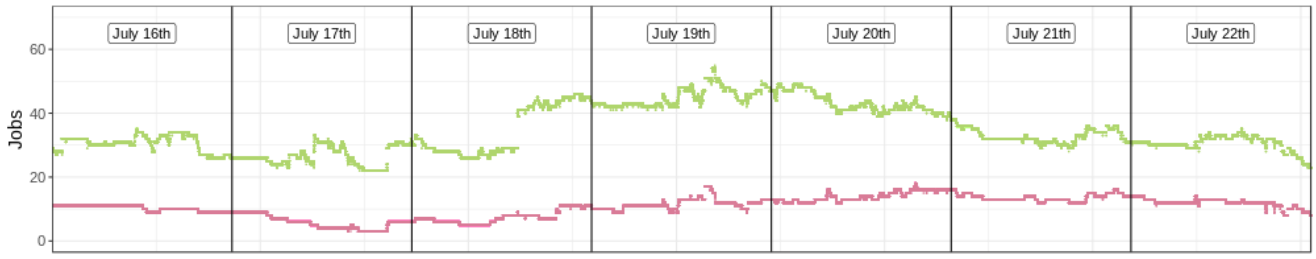
A global overview of the I/O sub-system is the first step to understand the I/O demands on the machine and detect possible issues that translate into poor I/O performance. We used the open-source *collectl* tool[†] to gather information about data and metadata access to the Lustre file system. The information was collected in each node every 2 seconds, during seven days – from July 16th to July 22th, 2017.

Data collection used, on average, 1.5% of the processing power of one core and 21 MB of RAM on each node. The logs were written to the local storage device of each node to avoid interfering with the observed I/O workload. This data is publicly available in the companion repository.

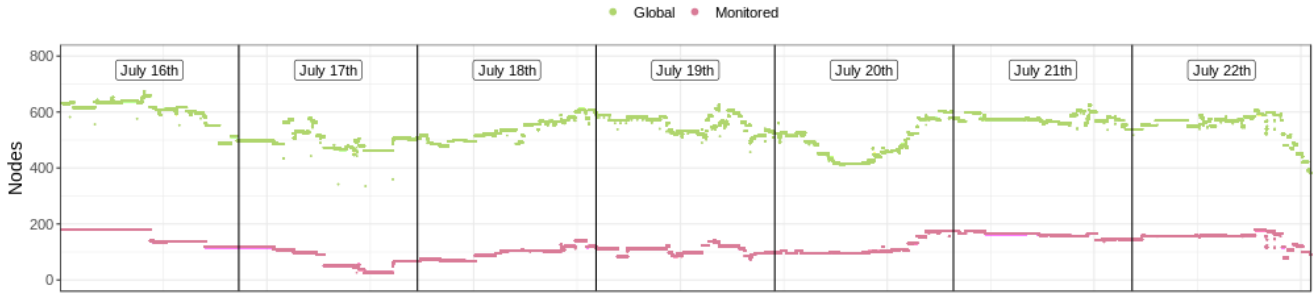
3.1 A week of I/O activity

Since we are working with a production system, seeking to minimize intrusion, we monitored the 192 nodes that were available among the B715 ones, which represents

[*] https://gitlab.com/jeanbez/ijhpca-sdumont
[†] http://collectl.sourceforge.net/



(a) Number of jobs



(b) Number of nodes in use

Figure 1. Comparison between the workload on the monitored nodes and on the whole system.

approximately 25% of the machine. Figure 1a depicts the number of jobs for all the nodes in the supercomputer (in green) and those using the monitored nodes (in red). Approximately 30% of the jobs were monitored. Table 1 presents the distribution of the number of concurrent jobs.

Table 1. Distribution of the number of concurrent jobs

	Min.	1st Q.	Median	3rd Q.	Max.
Global	22	29	32	42	55
Monitored	3	9	11	13	18

Figure 1b shows the number of nodes being used. Regarding the occupancy of the machine in this period, out of the 756 nodes, on average roughly 544 nodes were active, or 72%. Furthermore, out of the 192 monitored nodes, on average 122 nodes were being used, that represents approximately 64% of them. Table 2 presents the distribution of the number of nodes used. Moreover, a median of 8 nodes was used by each of the monitored jobs, whereas globally the median was of 10 nodes per job. This indicates that, on average, the number of nodes per job in the monitored nodes is 28% smaller than the observed for the whole system.

Table 2. Distribution of the number of nodes used

	Min.	1st Q.	Median	3rd Q.	Max.
Global	335	504	559	581	674
Monitored	26	97	118	157	180

The number of monitored jobs, depicted by Figure 1a, roughly matches the number of jobs performing I/O requests to the Lustre file system, as illustrated by Figure 2. This means that during most of the time all of them were issuing their I/O requests, represented by a median of 11 and a maximum of 18 jobs.

3.2 Results of the I/O workload study

Figure 3a presents the aggregated read bandwidth observed in the monitored nodes. Interactive versions of the plots are available online[‡] to improve the visualization of the data. The median bandwidth was of 71 MB/s and the maximum observed was of 2 GB/s. For write requests, depicted by Figure 3b, it is possible to see various behaviors coming from different applications and I/O demands. The median bandwidth is 698 MB/s and the maximum observed was of 3.6 GB/s.

Figure 4 presents the number of metadata operations per second over the studied period. Opening a file from Lustre requires contacting the metadata server first. We can see many peaks in metadata activity match the behavior observed in data write (Figure 3b), as the write workload is heavier and dominates the machine.

It is also important to consider if the data servers are getting a balanced workload over time, or if high I/O demands are directed to a single or a group of servers. Figure 5 depicts the aggregated write bandwidth during July 17th, 2017, divided by OST. It is possible to see that the bandwidth to all data servers is quite balanced, i.e., we do not have a OST that is more used than the others. Results for the other days were omitted because they display similar behavior.

We also consider the ratio between read and write operations to the PFS. Figure 6 presents the amount of data transferred by read operations divided by the total amount of data transferred (to both reading and writing). The read and write bandwidth, presented in Figure 3, gave an indication to writes dominating the workload, but those numbers could also come from a low read *performance* (rather than a low read *demand*). The results presented in Figure 6 confirm that the I/O workload of the machine is dominated by

[‡] https://jeanbez.gitlab.io/ijhpca-sdumont/

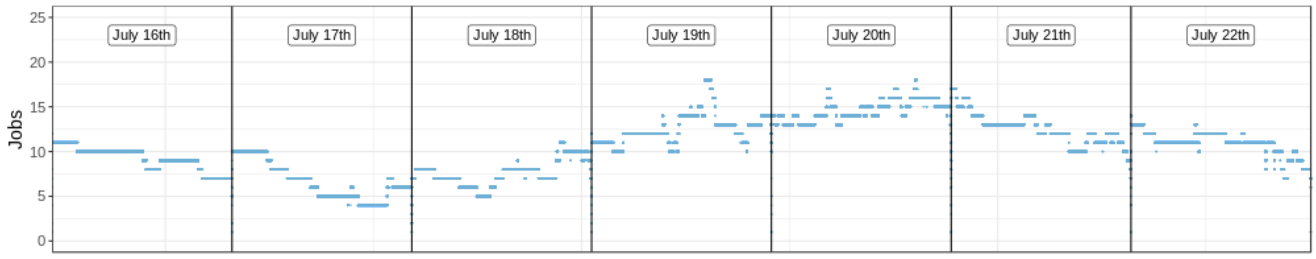
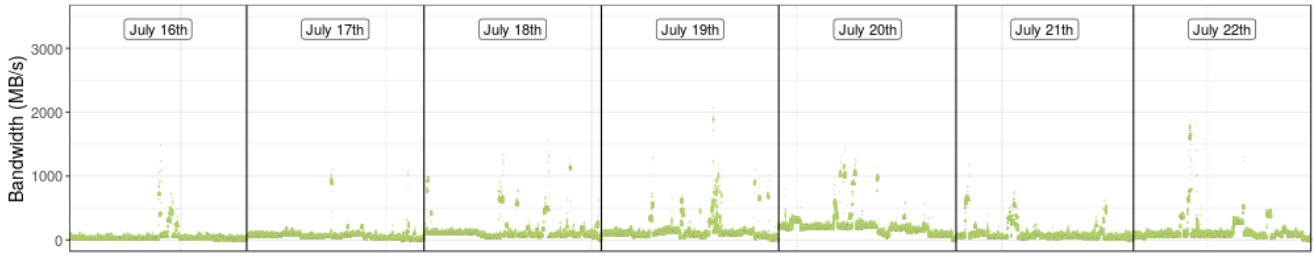
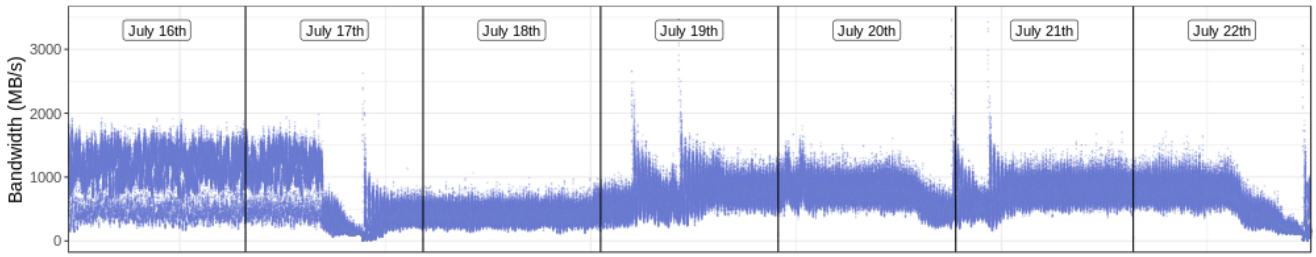


Figure 2. Number of monitored jobs performing I/O operations

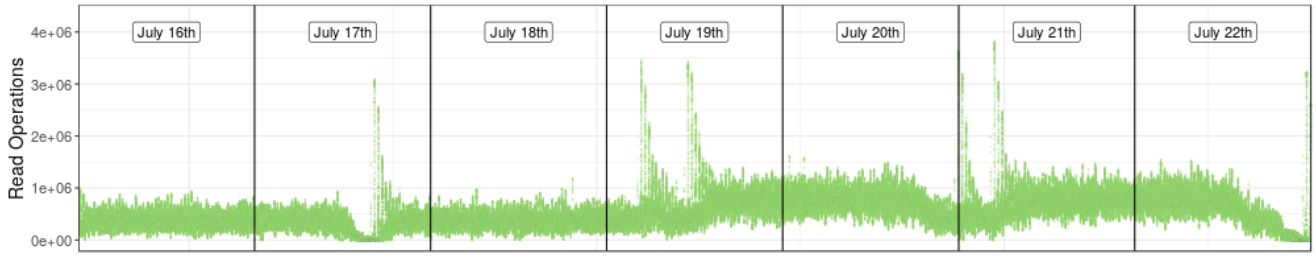


(a) Aggregated read bandwidth (MB/s)

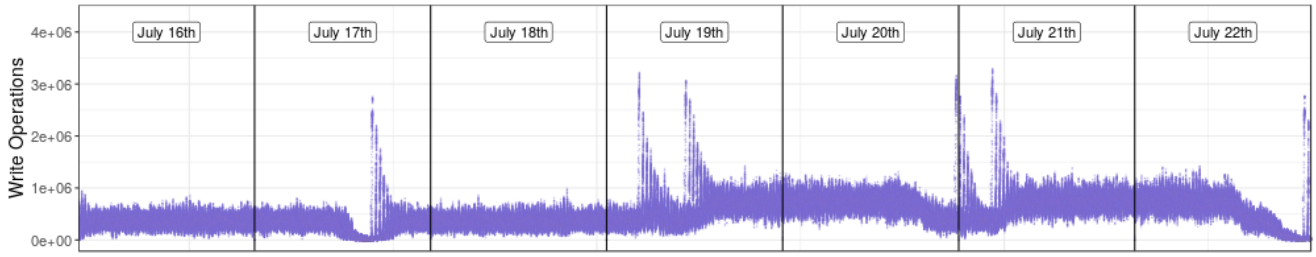


(b) Aggregated write bandwidth (MB/s)

Figure 3. Read and write workload in the SDumont supercomputer. Interactive version of the plots is also available at: `https://jeanbez.gitlab.io/ijhpca-sdumont`.



(a) Metadata read operations



(b) Metadata write operations

Figure 4. Read and write metadata operations per second in the SDumont. Interactive versions of the plots are available at: `https://jeanbez.gitlab.io/ijhpca-sdumont`.

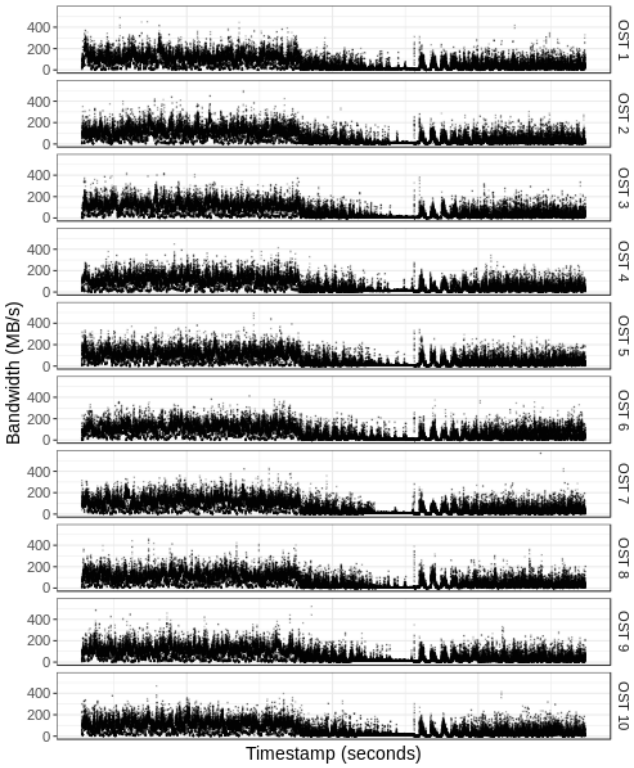


Figure 5. Aggregated write bandwidth per **data server** (OST) during July 17th, 2017 in the SDumont.

writes. Table 3 details the read distribution, in percentage, if compared to the total workload. Read operations represent less than 20% of the workload during over 75% of the studied period.

Table 3. Distribution of reads, in percentage, if compared to the total workload over the observed week.

Min.	1st Q.	Median	Mean	3rd Q.	Max.
0.00	4.79	10.86	14.04	19.48	100.00

Related work point out different proportions between the data transferred by read and by write operations. Nieuwejaar et al. (1996) studied traces obtained from the Intel iPSC/860 at NASA Ames Numerical Aerodynamics Simulation (NAS) and the Thinking Machines CM-5 at the National Center for Supercomputing Applications (NCSA). They observed a greater amount of written data, as well as a greater number of files opened for write operations. On the other hand, Carns et al. (2011), studying the workload on Intrepid, the IBM BG/P system at the Argonne Leadership Computing Facility (ALCF), showed that the applications transferring the most data in that machine usually read more data than write. Consequently, reads represented 78.8% of their I/O activity during the period studied by them.

Carns et al. (2011) also analyzed the access sizes, showing that the most common read size was between 100KiB and 1MiB, while the most common write size was between 100 bytes and 1KiB. On the other hand, a study conducted in a supercomputer at Oak Ridge National Laboratory (ORNL) Kim et al. (2010), using Spider, a Lustre-based storage cluster, observed three main request sizes: less than 16KB, 512KB and 1MB. This three sizes represented more than 95% of the total requests.

Table 4 summarizes the observed request sizes in SDumont. Read requests are on average to 92 KB, whereas write requests are on average a little over 4 MB. Read requests are in general ≈ 46 times smaller than write requests. It is possible to observe that the applications running in SDumont are mostly performing write operations in greater sizes than the observed by related work. Previous studies (Carns et al. 2009; Boito et al. 2018) already demonstrated that issuing larger requests results in higher I/O performance.

Table 4. Request size (KB) distribution

Operation	Min.	1st Q.	Median	Mean	3rd Q.	Max.
READ	0.3	24.3	41.9	92.5	80.6	2119.0
WRITE	4.0	3996.0	4282.0	4162.0	4455.0	88424.0

Table 5 lists the three most executed applications from the $89,106$ jobs submitted in 2017. These are the ones for which it was possible to identify the application from the executable name as users can compile and run their own source-codes on SDumont. From those, $3,331$ jobs were for GROMACS, which represents approximately 3.7% of the total, and account for $20,851,588.15$ core-hours. The second and third applications with the largest number of jobs had $3,095$ and $2,880$ jobs, representing $9,392,925.77$ and $4,781,320.89$ core-hours, respectively. GROMACS was also the most used application in the machine in 2016, with $2,292$ jobs.

Table 5. The most executed applications in the SDumont.

Application	Jobs	CPU Time (h)
Gromacs `gromacs.org`	$3,331$	$20,851,588.15$
VASP `vasp.at`	$3,095$	$9,392,925.77$
Quantum Espresso `quantum-espresso.org`	$2,880$	$2,919,831.06$

3.3 Discussion

We monitored 25% of the SDumont machine during a week, covering 34% of the jobs in the period, and presenting a similar workload to the whole machine. Almost all the monitored jobs were performing I/O, demonstrating the importance of a deeper investigation on the machine's I/O demand. We were able to conclude that the number of jobs is quite distributed during the day (Figure 1a), not having a period with greater activity. The equally divided bandwidth among the Lustre data servers also shows a balanced demand (Figure 5).

Write operations dominate the workload in the machine during over 75% of the observed period, representing 80% of the workload (Figure 6). This disproportion between read and write transference might influence the observed aggregated bandwidth. As depicted by Figure 3, write bandwidth is greater than the read bandwidth. This difference between read and write could also be influenced by the

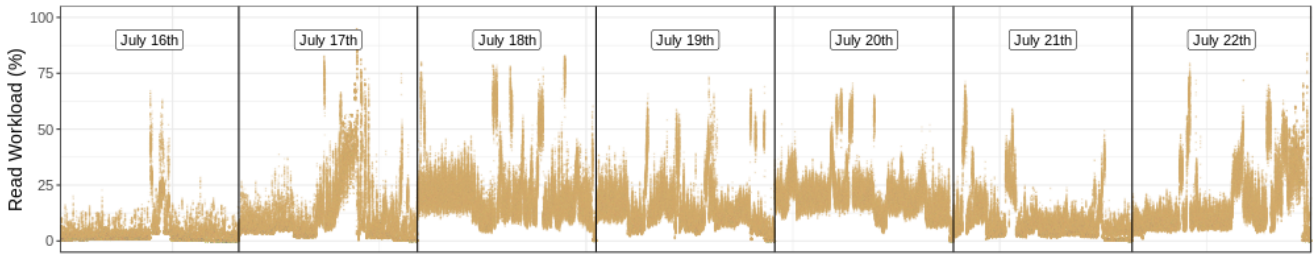


Figure 6. Amount of data transferred by read operations divided by the total amount of read/written data.

access sizes used by the applications when issuing read and wire requests. The write sizes observed are on average a little over 4 MB, while the read sizes are on average 92 KB, i.e., approximately 97% smaller. Related work already pointed out that larger access sizes improve performance. Therefore, these small access sizes are probably translating into low bandwidth, while the larger write access sizes are translating into high bandwidth.

It is also important to note the different behaviors in the write bandwidth over time, that may indicate different access patterns. If we consider the higher write bandwidth observed in the first two days (Figure 3b) and the lower number of metadata operations at the same period (Figure 4), they suggest a shared file with collective access pattern.

Even though we monitored $1/4$ of the machine, with a similar occupation and job characteristics from the whole system, the aggregated write performance never reached $1/4$ of the system's peak (30 GB/s). One of the reasons for low write performance is discussed in the next Section, the second part of our study.

4 Part II – Collective I/O Performance

During the study of the machine's I/O workload, described in the Part I, it was reported an unexpected behavior for the OLAM application[§]. The I/O performance of the application would change considerably when using different MPI implementations. In this second part of the paper, we investigate the reported issue by first analyzing OLAM's I/O performance in Section 4.1. We then generalize it to studying collective I/O performance and deeply investigating the observed phenomenon in Section 4.2.

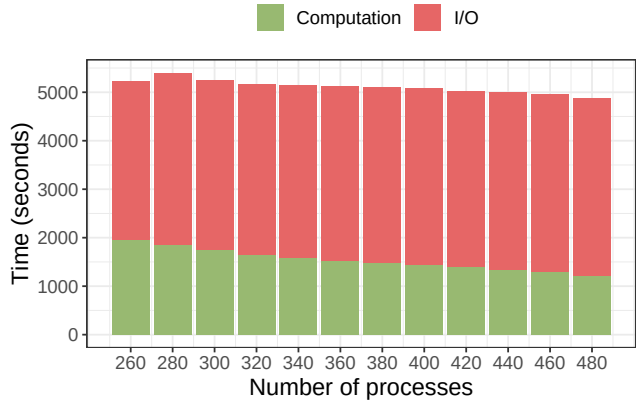


Figure 7. Initial results with OLAM in the SDumont.

Although OLAM is not in the the list of the most executed applications (Table 5), it is an strategic one for the machine because of ongoing efforts of Brazilian institutions (CPTEC/INPE, IAG/USP, LNCC, IME/USP, UFCG, UFSC, and EMBRAPA) to develop it. Moreover, its behavior is typical of simulations used for climate and weather studies. Finally, previous work Osthoff et al. (2012) has pointed I/O operations to be critical for OLAM. This situation is illustrated by the results presented in Figure 7. Bars stack time spent in I/O (in red) and computation (in green) for different numbers of processes. Up to 73% of the execution time was spent in I/O operations. Moreover, despite computation time decreasing as the scale increases, I/O compromised the application's scalability.

4.1 Performance Evaluation of OLAM

OLAM is an MPI application. Each process completely reads the initialization files, which contain initial global conditions at a certain date and time and information representing the environment. Next, OLAM simulate time steps, exchanging messages between neighbor processes at the end of each time step. After executing a number of time steps, the variables representing the atmosphere are written to a history file. During this phase, processes use the HDF5 (The HDF Group 1997–2016) library to write to the shared file, and the library generates MPI collective I/O operations. These output history files can have from a few MB to many GB, depending on the grid definition and model refinement.

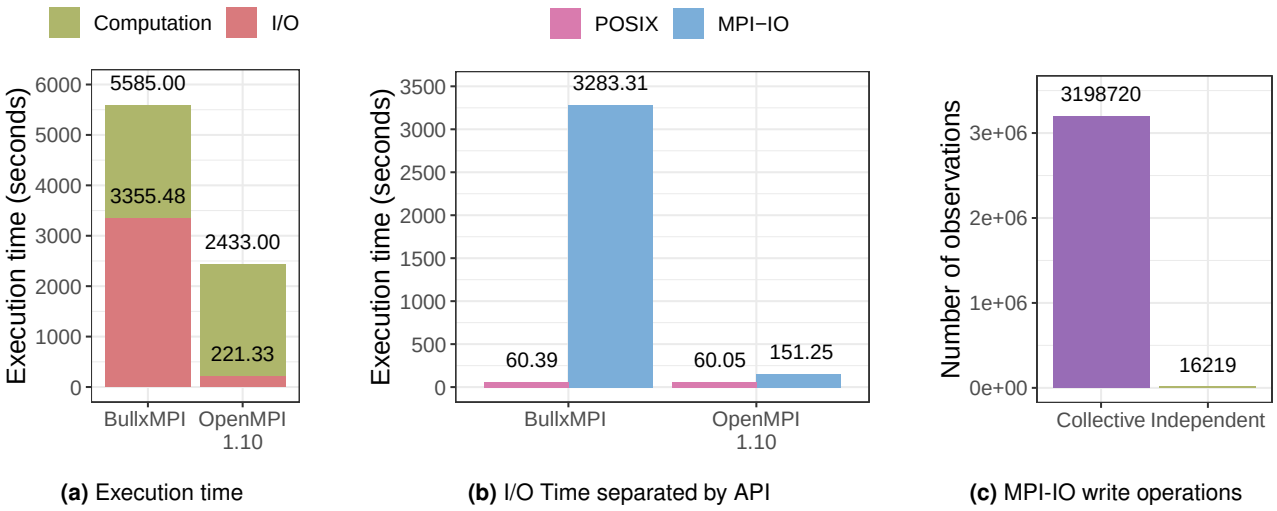
In this section, we discuss the I/O performance results of OLAM using two workloads: seven grids (Section 4.1.2) and four grids (Section 4.1.3). Our experimental experimental methodology is discussed in Section 4.1.1.

4.1.1 Experimental Methodology Experiments were conducted in the SDumont with OLAM version 4.10 (r544), modeling the northwest region of the São Paulo state, in Brazil ($21°00'00.0''S$ $51°00'00.0''W$). The configuration for these simulations has six vertical levels. Two days are simulated, using timesteps of ten seconds and writing output every simulated hour. A total of 49 files are generated, with approximately 1.1 GB each.

OLAM was compiled with Intel Parallel Studio XE 2017 update 1 and HDF5 version 1.8.18. The Darshan tool (Carns et al. 2011) version 3.1.4 was used to profile executions.

The results presented in this section are the medians of five executions on 10 compute nodes (sdumont[5004–5013]) using the 24 cores per node, for a total of 240 cores. No

[§]https://sourceforge.net/projects/olam-model/



(a) Execution time **(b)** I/O Time separated by API **(c)** MPI-IO write operations

Figure 8. Results for OLAM with 7 grids comparing BullxMPI and OpenMPI 1.10, reported by Darshan.

additional hints were passed to HDF5 or MPI-IO, and all MPI implementations use the same parameters. We used the default stripe size of 1 MB and a stripe count of 10, i.e., files are distributed among all 10 Lustre OSTs.

4.1.2 OLAM with 7 grids The first set of experiments that we will discuss compare BullxMPI and OpenMPI 1.10. OLAM was executed with seven grids (one global and six refined) of resolutions 200km (global), 100km, 50km, 25km, $12.5km$, 6.25km, and 3.125km. Results, as reported by Darshan, can be seen in Figure 8. The first graph, in Figure 8a show time spent in I/O (in red) and computation (in green). We can see a large decrease in execution time when using OpenMPI 1.10, due to a much shorter I/O time (computation time was similar to both implementations).

Since we could not conclude all these results follow a normal distribution, we have used the Wilcoxon-Mann-Whitney test (Feltovich 2003) to compare them, and compare the medians instead of the means. The statistical test has indicated the computation time for the two MPI implementations is not significantly different, but I/O and total execution time are.

Figure 8b separates time spent in I/O operations per API – POSIX and MPI-IO. It shows most of the I/O time is spent in MPI-IO operations (through HDF5), and the performance difference between the MPI implementations comes mostly from this part. In Figure 8c the number of MPI-IO write calls is presented by type. We can see most of the write operations are collective. Furthermore, Table 6 shows the most usual sizes for I/O requests generated by OLAM. Requests are rather small, with most MPI-IO operations accessing approximately 1300 bytes.

4.1.3 OLAM with 4 grids The large performance difference between the MPI implementations was unexpected, and thus we have conducted more comprehensive experiments, including all MPI implementations that are available at the machine. To decrease the processing time in the supercomputer while keeping the same I/O behavior we executed an OLAM configuration with four grids of resolution 200km (global), 100km, 50km, and 25km.

Table 6. Size of I/O operations generated by OLAM, as reported by Darshan.

API	BullxMPI		OpenMPI-1.10	
	Size (bytes)	Count	Size (bytes)	Count
POSIX	8192	8585280	8192	8585280
	8190	6785760	8190	6785760
	512	126186	512	126186
	1048576	46011	1048576	46011
MPI-IO	1308	94521	1308	110397
	1312	55713	1312	94521
	1316	44394	1316	63798
	59040	43904	58860	59584

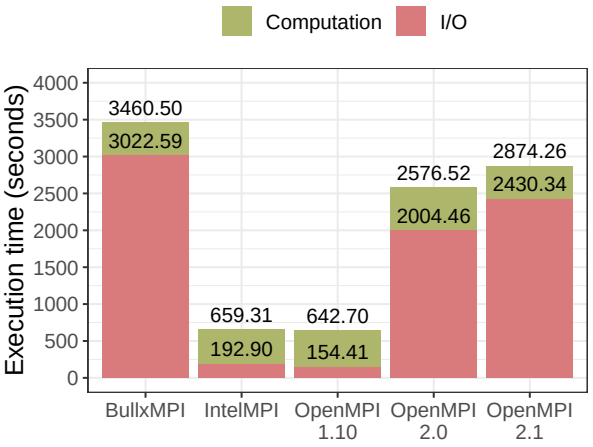


Figure 9. Results for OLAM with 4 grids.

Because of incompatibilities between Darshan and some MPI implementations, we have modified the OLAM source code to measure and report execution and I/O time internally.

Results are presented in Figure 9, and show one more time that BullxMPI is the alternative that causes OLAM to spend the most time on I/O operations, 87% of the execution time with four grids. The large difference to OpenMPI 1.10 performance was still present, IntelMPI presented similar results to OpenMPI 1.10, and OpenMPI 2.0 and 2.1 were

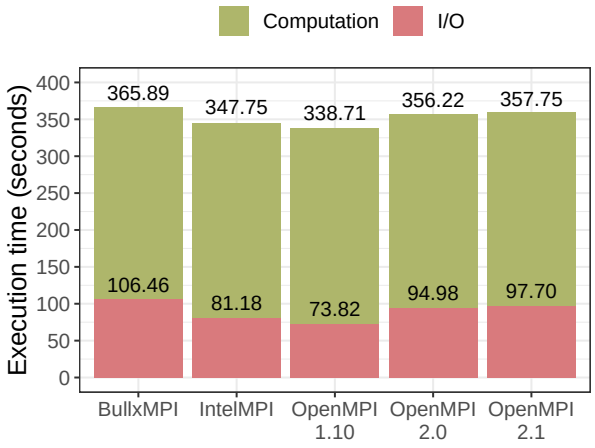


Figure 10. Results for BT-IO class D.

better than BullxMPI, but worse than OpenMPI 1.10 and IntelMPI.

Similarly to the OLAM configuration with seven grids, we have not concluded that all the sets of results follow a normal distribution, and thus used a non-parametric test to compare them. The Dunn test (Dunn 1961) could not conclude results for IntelMPI are significantly different from results with OpenMPI 1.10 (not for I/O nor for total execution time). Similarly, results for BullxMPI are not statistically different from results for OpenMPI 2.1. Finally, results for OpenMPI 2.0 and 2.1 are not significantly different.

4.2 Collective I/O Performance

In the previous section, experiments with OLAM indicated large performance differences between MPI implementations. Because of the application's characteristics, reported by Darshan, these differences were believed to be related to MPI collective write operations. Consequently, this section presents an investigation of collective I/O performance through a series of experiments.

4.2.1 Experiments with the BT-IO benchmark To confirm that it was not something specific to the application, we conducted experiments with the BT-IO benchmark from the NPB (NASA 1994), the second most used benchmark in the parallel I/O research field, as pointed by Boito et al. (2018). We used the D class, which generates a file of approximately 132.6 GB and yields an execution time in order of minutes. This benchmark generates MPI-IO collective write calls. These experiments were executed over eight nodes (sdumont[5004–5011]), using 18 cores per node, for a total of 144 cores.

Results are presented in Figure 10, and represent the median values of 5 repetitions. The Dunn test was used to compare all sets of results. It indicated that the results for BullxMPI are significantly different from IntelMPI and OpenMPI 1.10, and the results for OpenMPI 1.10 are different from OpenMPI 2.0 and OpenMPI 2.1. Nonetheless, we can see this difference is quite small if compared to what was observed before. Studying the information provided by Darshan, we observed the size of requests generated by the benchmark was approximately 18 MB, much larger than requests generated by OLAM.

4.2.2 Experiments with IOR benchmarking tool The fact we observed large performance differences between different MPI implementations with OLAM, but not with the BT-IO benchmark, indicates this difference does not happen for large collective write requests. To confirm it happens for small requests (that it was not something specific related to OLAM), we conducted more experiments using the IOR benchmarking tool [¶].

IOR experiments were executed with all the MPI implementations available, including the OMPIO implementation of MPI-IO, available for OpenMPI versions. We only used ROMIO for previous results (detailed in Section 4.1.3) because of incompatibilities between OMPIO and OLAM. IOR was configured to perform collective read and write tests, generating a file of 1.5 GB using MPI-IO and HDF5 (the latter also used by OLAM, as discussed in Section 4). Tested request sizes were 1024, 1312, 58864, and 65536 bytes. Request sizes of 1312 and 58864 bytes are the most common access sizes used by OLAM, as seen in Table 6. They generate misaligned access on the Lustre parallel file system that is configured with a stripe size of 1 MB. To verify the impact of accesses that are not aligned with the stripe size, we also included requests of 1024 and 65536 bytes, which do not lead to misaligned accesses. Table 7 details the IOR parameters used for these experiments. The results are depicted by Figure 11, and they represent the median values of five executions on 10 compute nodes (sdumont[5004–5013]) using 24 cores per node, for a total of 240 cores.

Table 7. Parameters of IOR experiments.

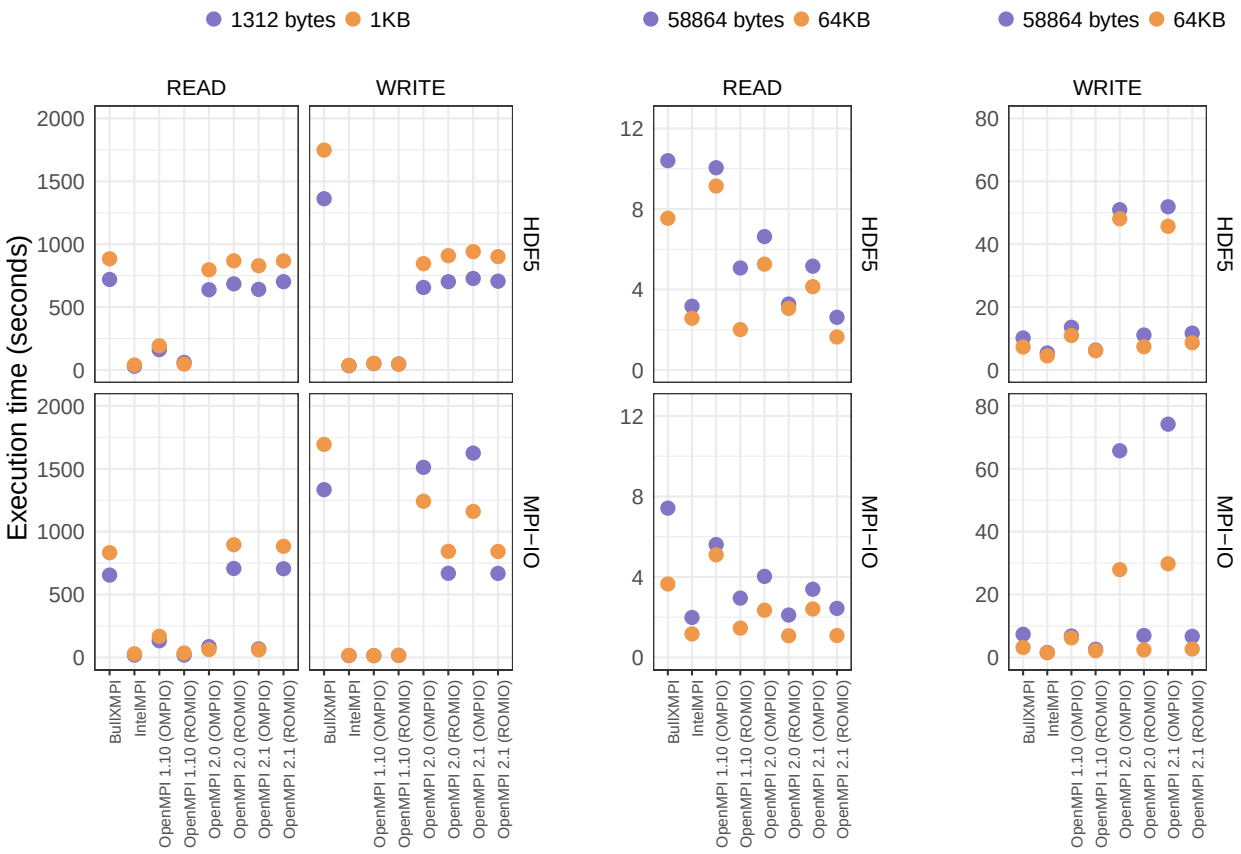
Transfer Size	Block Size	Segment Size	Segment Count
1024	1024	245760	6554
1312	1312	314880	5116
58864	58864	14127360	116
65536	65536	15728640	104

For the small request sizes (1024 and 1312 bytes), shown in Figure 11a, we can see performance differences between MPI implementations (for both read and write tests) that are very similar to what was observed for OLAM in Figure 9. These differences are smaller and show different behavior for the large request sizes (58864 and 65536 bytes), shown in Figure 11b and 11c. This confirms the differences observed in Section 4 are **not** specific to OLAM, but happen when small requests (of up to approximately 1 KB) are generated.

The Dunn statistical test was used to compare all sets of results. Write/read time obtained with IntelMPI and OpenMPI 1.10 with ROMIO were **not** significantly different. Results for OpenMPI 1.10 with OMPIO were **not** significantly different from the other two sets in small tests with HDF5 and small write tests with MPI-IO.

Time obtained with BullxMPI was significantly different from OpenMPI 1.10 and IntelMPI in most small tests, except small read tests with MPI-IO, where it was **not** significantly different from OpenMPI 1.10 with OMPIO. For large tests, BullxMPI and OpenMPI 1.10 were similar in most cases — except BullxMPI and OpenMPI 1.10 with OMPIO in read

[¶] https://github.com/hpc/ior



(a) Small requests **(b)** Large read requests **(c)** Large write requests

Figure 11. IOR results. It is important to notice the scale is **not** the same in all graphs.

tests with HDF5 and 64 KB requests, and in read tests with MPI-IO and requests of 58864 bytes.

Comparing results for BullxMPI to the ones for OpenMPI 2.0 and 2.1, they are different when using ROMIO for small read experiments and large write experiments — except OpenMPI 2.0 with ROMIO in the tests with MPI-IO and requests of 58864 bytes. In small write experiments, BullxMPI results are significantly different from the ones for OpenMPI 2.0, except using MPI-IO with OMPIO for 1 KB requests and with both ROMIO and OMPIO for 1312 bytes requests, and from the ones for OpenMPI 2.1 with ROMIO, except using MPI-IO with requests of 1312 bytes. Finally, in large read experiments, results for BullxMPI are **not** significantly different from results for OpenMPI 2.0 and 2.1 with OMPIO in tests using MPI-IO with 64 KB requests.

As expected, larger requests lead to higher performance from all MPI implementations (all differences were confirmed with the Wilcoxon-Mann-Whitney test). The misaligned access only presented a large negative impact on versions 2.0 and 2.1 of OpenMPI with OMPIO, using the MPI-IO API and large request sizes. Nonetheless, the difference was **not** confirmed by the statistical test for read experiments with large requests, using MPI-IO through OpenMPI 2.1 with OMPIO.

Regarding the I/O APIs, HDF5 presented an inferior performance than MPI-IO, due to its added overhead. This difference was **not** confirmed by the Wilcoxon-Mann-Whitney test for large write tests with BullxMPI, write tests with requests of 58864 bytes with OpenMPI 1.10 with ROMIO, small read tests with OpenMPI 2.1 with ROMIO, read tests with 1 KB requests with IntelMPI, read tests with 58864 bytes requests with IntelMPI, OpenMPI 1.0 with ROMIO, OpenMPI 2.0 with ROMIO, and OpenMPI 2.1; and read tests with 64 KB requests with OpenMPI 1.10 with ROMIO and OpenMPI 2.1 with OMPIO.

4.2.3 *Experiments using a customized microbenchmark*

After detecting the performance difference between MPI implementations in the OLAM experiments, and confirming through benchmarks that it happens for small collective I/O requests, we developed a microbenchmark to further investigate this phenomenon. It was developed based on the two-phase collective write operations as implemented by ROMIO (the Lustre-specific implementation). The same MPI calls internally used by ROMIO are used to implement the different steps, and the microbenchmark reports time spent on each step. Table 8 details these steps. The source code is freely available [‖].

This experiment was configured to issue collective write requests of 1312 bytes, using 10 compute nodes (sdumont[5004–5013]) and all 24 cores per node, for a total of 240 cores. Figure 12 shows results, which are median values from five executions. It is possible to notice that most of the time was spent in Step 3, where processes

‖ https://github.com/francielizanon/pretend_coll

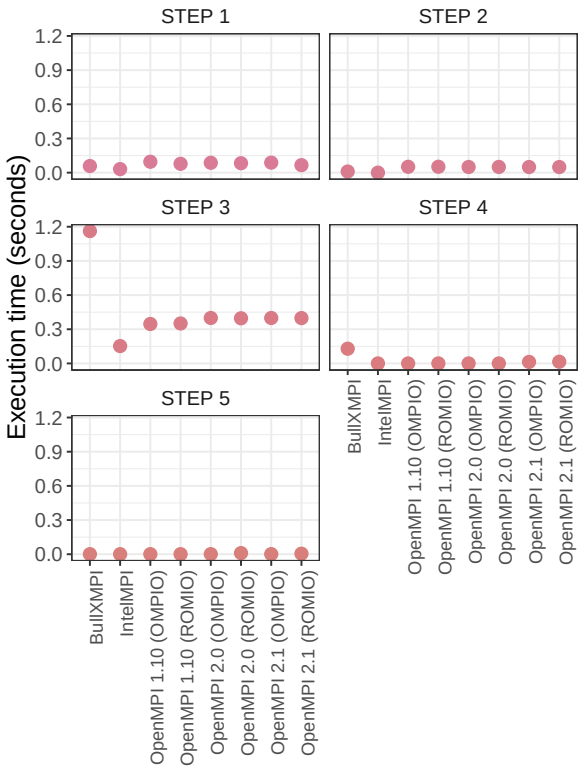


Figure 12. Time on each step of the collective write operation.

communicate request information (offset and size) to their aggregators. Most of the difference in performance between the MPI implementations comes from this step. There is some difference in Step 4, but on a much smaller scale.

We applied the Dunn test in each step to compare results for different MPI implementations. In step 3, BullxMPI results are significantly different from results obtained for OpenMPI 1.10, and IntelMPI results are different from OpenMPI 1.0, OpenMPI 2.0 with OMPIO, and OpenMPI 2.1 with ROMIO. In step 4, BullxMPI results are different from results with OpenMPI 1.10, and results with IntelMPI are different from OpenMPI 2.0 and OpenMPI 2.1.

Table 8. Steps of the custom microbenchmark that mimics two-phase collective write operations.

Step 1	Exchange messages between all processes so that every one knows the start and end offsets of the whole requested portion (`MPI_Allgather`).
Step 2	Exchange messages between all processes so that every one knows the number and size of original requests (`MPI_Allreduce`).
Step 3	Exchange messages between aggregators and processes to communicate the offsets and access sizes (`MPI_Isend` and `MPI_Irecv`).
Step 4	Exchange messages between aggregators and processes so that every aggregator obtain the data to perform the write operation (`MPI_Isend` and `MPI_Irecv`).
Step 5	Aggregators execute the I/O operation to the parallel file system (`MPI_File_write_at`).

As detailed in Table 8, steps 3 and 4, where we can see differences between the MPI implementations, use the same asynchronous MPI calls. The main difference between them is that step 3 sends two numbers, while step 4 sends data (which size depends on the size of the operation). In this case, with requests of 1312 bytes by 240 processes, less data is transferred by step 4 than by step 3.

It is also interesting to observe how performing the operation to the remote parallel file system, which could be expected to be the most important step for performance, does not account for most of the time. Instead, most of the time spent on such small collective write operations is used for coordinating processes and exchanging data. We have not observed differences for large collective calls in Sections 4.2.1 and 4.2.2 because their steps 4 and 5 will be longer, decreasing the impact of step 3.

4.2.4 Discussion We investigated the I/O performance of OLAM because a large portion of its execution time was spent in I/O operations, and also because it was reported to have an unexpected behavior: large I/O performance differences between MPI implementations. We reproduced the scenario and confirmed these differences. SDumont users can choose between the three available implementations: BullxMPI v1.2.8.4, based on OpenMPI, IntelMPI v5.1.3 build 20160120, based on MPICH, and OpenMPI versions 1.10, 2.0, and 2.1. Figure 13 shows the percentage of jobs using each MPI implementation, as observed from June 15 to July 22, 2017. We can verify that 22.76% of the submitted jobs used the BullxMPI implementation, which presented the longest I/O times.

These differences were believed to be due to MPI-IO collective write operations, as inspecting Darshan traces we found most operations generated by the application were collective writes. A further investigation was conducted with benchmarks. Results demonstrated that the observed difference happens for small requests (of approximately 1 KB), and comes from the step of the collective I/O operation where processes exchange small asynchronous messages (with *Isend* and *Irecv* calls) to communicate with aggregators.

We believe the performance difference between BullxMPI and the other OpenMPI implementations is due to BullxMPI being based on an older OpenMPI version than 1.10. Neau et al. (2013) observed similar behavior. The results with the microbenchmark show this difference comes from the *Isend*/*Irecv* pairs, as previously discussed. Our results with OLAM (Figure 9) and with the IOR benchmark (Figure 11a) also pointed to a performance difference between OpenMPI 1.10 and 2. That difference was *not* observed in the results with the microbenchmark. Upon further investigation, we observed the OpenMPI 1.10 installation in the SDumont did *not* use the Lustre-specific collective write operation (emulated by the microbenchmark). Moreover, the generic implementation used in the experiments with OpenMPI 1.10 chooses *not* to actually perform the collective operation if the different processes' requests are not interleaved. For these small requests in the SDumont, not performing the collective operation was hence a better decision.

With this investigation, we provide valuable information that can be used to improve future versions of collective write

implementations, especially when considering tiny request sizes. Furthermore, the most concrete contribution is advice to be given to SDumont users, as it was observed that 22.76% of the submitted jobs used BullxMPI, the implementation that presented the worst results in our analysis. By helping users achieve better performance for their applications, we promote better usage of the machine.

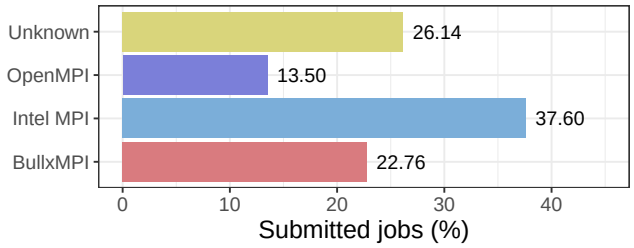


Figure 13. MPI usage among jobs submitted in the SDumont.

5 Part III – Characterization of Jobs' I/O Behavior

The Part II of this paper was motivated by a report on a performance issue from an application using the SDumont machine. Investigating it required conducting a performance evaluation of the application, and using a profiling tool (Darshan). A challenge is that users are often not familiar with the specifics of the application, hence they cannot provide enough details about it. In the case of OLAM, users familiar with its source code could have told us it uses the HDF5 library to write to shared files, but that would not tell us it generated small collective operations.

The observation of this challenge motivated the systematic use of such a tool for all jobs in the machine, so future performance issues require less effort to investigate. However, multiple executions of an application will generate multiple coarse-grained traces, with aggregated statistics about their I/O activities. Obtaining information from those traces is another challenge, specially if we are interested in temporal aspects of the application's I/O behavior.

To tackle this issue, in this third part of the paper we propose a strategy to characterize jobs' I/O phases from coarse-grained aggregated traces. We present our method by applying it to the OLAM's application, studied in the Part II, in order to demonstrate its usefulness. We then apply it to traces obtained from production executions of the GROMACS application (van der Spoel and Hess 2011) as a case study. GROMACS was selected for this analysis because it is the most executed application in SDumont, as discussed in Section 3.

5.1 Proof-of-concept: OLAM characterization

We used the Darshan profiles of two OLAM executions, one that used BullxMPI and another that used OpenMPI 1.10 by extracting the information in Darshan's counters. As discussed in the Part II of this paper, these two MPI implementations show the worst and the best I/O performances for OLAM, respectively.

Seeking to understand and characterize the I/O behavior of these executions, we use the concept of I/O phases to identify intervals where I/O operations are made with a certain access pattern. As we base the start and end of each phase on the timestamps reported by Darshan, we cannot assure that throughout that entire period I/O operations are indeed happening, but we can be sure that if any I/O operations are happening those patterns will characterize them. That is due to the way Darshan capture its metrics, collecting, for each file handle, the time of the first and the last operations of a given pattern. Furthermore, as these phases may overlap in time, due to multiple processes issuing simultaneous requests, with the help of the GenomicRanges (Lawrence et al. 2013) library, we identified the overlapping I/O phases that represented the behavior throughout the application executions.

Figure 14 depicts the four most relevant I/O phases of the two OLAM executions. The green phases are the ones where the two MPI implementations took different times to execute the same set of operations, and it is the only phases to have collective MPI-IO writes. Therefore, this analysis of the jobs I/O phases would have quickly pointed the collective write operations were the source of the performance difference between the MPI implementations.

5.2 Case study: GROMACS characterization

GROMACS is a widely used molecular dynamics (MD) software. From the I/O point of view, it uses the master-slave paradigm to write its results to file. At each output phase, the master synchronously receives data from all files through MPI, and then writes it to the file system. During I/O operations, the simulation is blocked, what may affect performance and force scientists to reduce the output frequency (Dreher and Raffin 2014).

Figure 15 depicts the execution time of all the 3,331 GROMACS jobs from 2017. It is possible to see that most of the jobs execute for more than one hour. Approximately 40% execute in less than one hour, and roughly half of the remainder jobs execute for more than 24 h. This distribution comes from different usages of the application to different scientific scenarios, what may translate into different I/O behaviors and performance. These jobs were obtained from versions 4.x, 5.x and 2016 of GROMACS.

We applied the same methodology employed to evaluate OLAM's I/O phases, by harnessing data from Darshan logs, with the jobs that ran GROMACS with the profiler enabled from July 5th, 2017 to July 20nd, 2017. 165 jobs had their I/O profile captured by Darshan 3.1.4. This represents approximately 5% of all the GROMACS' jobs that ran in that period. After characterizing the I/O phases, we selected only the phase that mostly represents the I/O of the application. For that, we multiplied the average duration of each phase by the number of observations and selected the phase with the largest value. Following this approach, we could group all the jobs into four distinct classes, with well defined I/O behaviors. Despite considering executions from distinct users, it was also possible to notice differences in the I/O workload among executions of the same user. Table 9 presents the number of jobs in each group.

We selected four representatives, one for each group, by taking the jobs with the longest I/O time and plotted all their phases in Figure 16. We normalized the execution times to facilitate visualization. It is important to notice that since

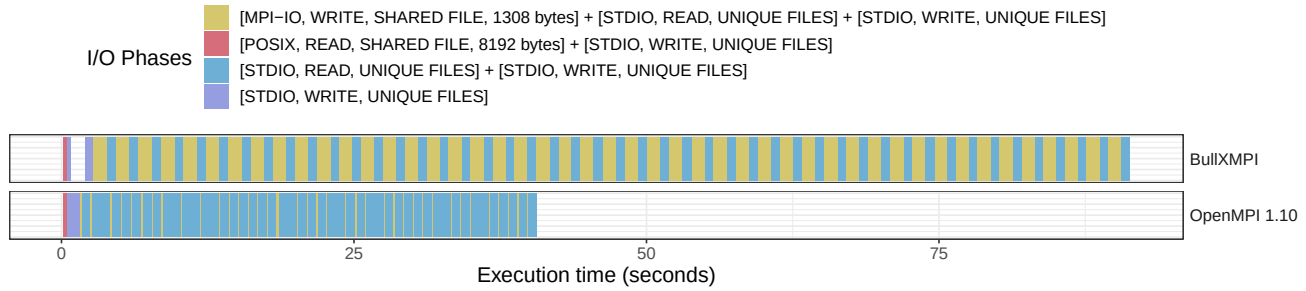


Figure 14. I/O phases of the OLAM application using two MPI implementations.

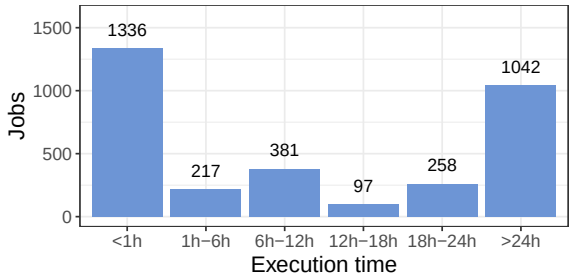


Figure 15. Execution time of the GROMACS jobs

Table 9. Number of jobs grouped by the characterization of their most representative I/O phase.

Group	Jobs	Percentage
A	17	10.30
B	73	44.24
C	31	18.79
D	44	26.67
Total	165	100.00

we are not using a fine-grained trace from the application, but rather aggregated information on its I/O profile, it is not possible to pinpoint when exactly those operations occur in that phase. Nevertheless, this gives us an overview of the expected I/O behavior. Group **A** writes to unique files during most of the execution, with a phase that includes reads that happens by the end of the execution. On the other hand, group **B** is mainly characterized by read operations to a shared-file. In group **C**, GROMACS jobs can be issuing read operations to a shared-file, and read and write operations to individual file throughout the execution. Finally, group **D** has the same I/O phases than group **A**, but distributed differently over the execution time.

By grouping all the jobs we can also have an estimate of the median execution time, I/O time and transfer size. Table 10 presents these data. The total absolute time spent in I/O, as reported by Darshan, is considerably smaller than the total runtime for all the groups. If we compute the achieved bandwidth by each group, we have approximately 7 MB/s for group A, 9 MB/s for group B, 126 MB/s for group C, and 69 MB/s for group D.

Nonetheless, we also observed two extreme behaviors that are summarized by Table 11:

I. a job that spent less than one minute on I/O, but that issued a large number of requests (over 49 million); and

Table 10. Metrics of each group of GROMACS jobs.

Group	Runtime (s)	I/O Time (s)	Total Data (KB)
A	295	0.175	1232.39
B	1	0.005	44.67
C	93587	22.509	2904276.42
D	79563	23.304	1669392.70

II. a job that spent most of its execution on I/O operations, but issued 33 times fewer requests than I.

Job I is a representative of the I/O characterization defined by group **C**, whereas job II is of group **D**. We picked these two to delve into its I/O phases. Job I has six different types of phases in total, and five of them only happen at the beginning of the execution. Table 12 details the identified phases, their characterization, the number of times each phase was observed, their average duration, and the difference between the start time of phases with that same characterization. The one with the most extended duration took approximately 1.3 hours, and it is characterized by read and write operations to a unique file and reads to a shared file. By unique files we used the name denomination used by Darshan, which means any files that were not opened by every rank in the job. This includes independent files (opened by one process) and partially shared files (opened by a proper subset of the job's processes). It is important to recall that Darshan captures the first and last operation for a given file handle. Therefore, these phases do not mean that the job spent 1.3 hours making I/O requests, but instead tells us that if any I/O operation happened during that amount of time, it would be characterized as described. This phase was also the one that appeared the most (12 times).

On the other hand, job II, that took 29.23 hours, presents a quite different behavior than job I, having less I/O phases. It also differs from job I by not using collective operations. Table 13 present further details. In this job, we have a phase characterized by reads to unique files with 10 ms duration, followed by a phase with writes to unique files, with five occurrences and an average length of 3 minutes. The predominant one, characterized by read and write operations also to unique files, was observed 11 times, with an average duration of 2.6 hours.

Considering the performance achieved by the job I, out of the 15.89 hours of execution time, only 50.92 seconds were spent in I/O operations to transfer approximately 1 GB of data. This translated into a bandwidth of roughly 20.39 MB/s. Differently, job II, that spent 29.23 hours running, needed 8, 22 minutes to transfer 3.74 GB

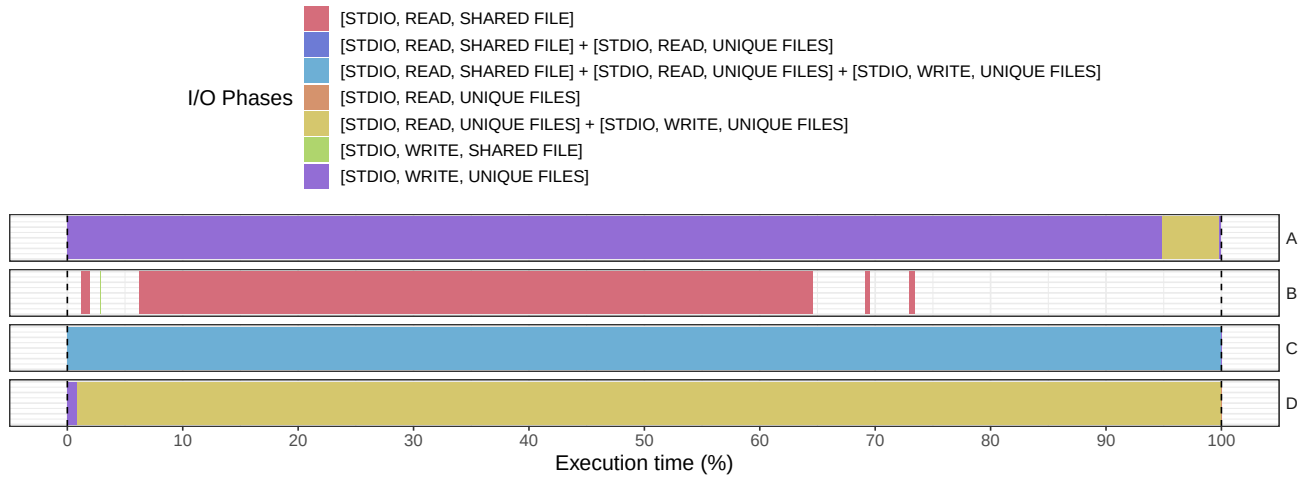


Figure 16. Representative jobs of GROMACS and their different I/O behaviors throughout execution, normalized.

Table 11. (I) Job that made the largest number of I/O operations; (II) Job that spent the most time in I/O.

Execution	Processes	Runtime (s)	I/O time (s)	Reads	Writes	Read (MB)	Write (MB)
I	48	57,227	27.77	47,734,940	1,226,680	683	413
II	384	105,257	291.49	27,992	1,428,605	2,103	1,723

Table 12. Characterization and statistics of each I/O phase detected in execution I.

I/O Phase Characterization	N	Duration (ms)	Difference (ms)
STDIO, READ, SHARED	1	34.00	0.00
STDIO, READ, SHARED STDIO, READ, UNIQUE	3	39.67	47.50
STDIO, READ, SHARED STDIO, READ, UNIQUE STDIO, WRITE, UNIQUE	12	4,768,497.17	5,201,996.40
STDIO, READ, UNIQUE	1	64.00	0.00
STDIO, READ, UNIQUE STDIO, WRITE, UNIQUE	3	79.00	5.50
STDIO, READ, UNIQUE	3	957.67	835.00

Table 13. Characterization and statistics of each I/O phase detected in execution II.

I/O Phase Characterization	N	Duration (ms)	Difference (ms)
STDIO, READ, UNIQUE	1	10.00	0.00
STDIO, READ, UNIQUE STDIO, WRITE, UNIQUE	11	9,485,752.50	10,434,495.00
STDIO, WRITE, UNIQUE	5	181,500.8	263,125.18

throughout its execution, yielding a bandwidth of 7.78 MB/s. If we consider the I/O characterization of both jobs, as depicted by Figure 16 the difference between the two is the existence of shared-file read operations in group C, and its absence in group D.

With this phase analysis, we can observe how the I/O behavior of the application changes between different executions. In the job I, that made more I/O operations, we observed more I/O phases, which did not happen for the job II. The largest amount of time spent by job II might be related to the fact that this execution is using individual operations, where few optimization opportunities are available.

5.3 Discussion

In this section, we presented our proposed strategy to harness information from Darshan's coarse-grained profiling of jobs which facilitates investigating I/O issues. As a proof of concept, we applied it to two HPC applications that run of SDumont: the OLAM and the GROMACS.

The technique's usefulness was illustrated by the OLAM investigation, where it was possible to point to collective write operations as the source of the performance difference between the MPI implementations. We also applied it to GROMACS, the most executed application in SDumont supercomputer. With this phase analysis, we can observe how the I/O behavior of the application changes between different executions. By analyzing the behavior of all the 165 executions of GROMACS during the studied period, we could detect four groups with different I/O characteristics.

Therefore, in addition to providing information to help identify I/O performance issues, our proposal can be valuable to guide I/O optimizations at the application or system level. Furthermore, it can also be a source of information to researchers from the parallel I/O field by identifying common I/O behaviors in an HPC system using already available Dashan traces.

6 Related Work

In a previous work, we conducted an in-depth investigation of the performance differences between MPI implementations for small collective I/O operations on the SDumont. To the best of our knowledge, ours was the first work (Carneiro et al. 2018) to document this phenomenon.

It is fairly common that vendors prepare customized MPI implementations for use in their supercomputers and large-scale clusters. These commercial solutions often claim improved performance over open-source alternatives. Some studies seek to evaluate such implementations considering distinct workloads and applications.

In (Vinter et al. 2004) the authors evaluated three MPI implementations, two open-source (MPICH and LAM-MPI) and one proprietary (MESH-MPI). They employed benchmarks such as the NPB – NAS Parallel Benchmarks – and showed the commercial implementation is significantly faster than the open-source alternatives. Additionally, they demonstrated that the customized solution yields much better performance for small collective communication operations. Closely related is the evaluation conducted in (Brightwell 2005) for the Cray Red Storm computing platform. The vendor-supported MPICH2 implementation (MPICH2-0.97) was compared to two other solutions based on MPICH (MPICH-1.2.6 and MPICH-1.2.6 using SHMEM (Brightwell 2004)). They demonstrated that the first is slightly outperformed by an open-source alternative in terms of latency and bandwidth. However, they did not take into account in their evaluation the use of collective I/O.

This paper extends our previous one (Carneiro et al. 2018) by presenting a more comprehensive study of the I/O performance in the SDumont. In addition to the investigation on collective operations, we characterize the workload of the machine and propose a strategy for obtaining information about jobs I/O behaviors from traces.

A wide range of factors can negatively impact I/O performance in HPC. Understanding and characterizing a platform can provide insights on how the applications should perform I/O operations to obtain the best performance. Zoll et al. (2010) studied a set of application-side I/O traces from the ASCI cluster at Lawrence Livermore National Laboratory, using the Lustre parallel file system for storage. Their traces were obtained in 2003 with the *strace* tool, ranged from tens of seconds to half an hour, and included two scientific applications from the physics domain and three benchmarks generated with IOR (file-per-process, shared-contiguous and shared-strided). They concluded that a Markov model could not represent the request arrival rate of applications' I/O streams present self-similarity. They presented a stochastic model to predict I/O arrival rate.

Wang et al. (2004) studied request size behavior from the same traces, and showed applications performed large numbers of small requests (from a few bytes to 1 MB) in small time intervals. Their results for request size and inter-arrival time are specific to the traced applications.

To motivate their work on cross-application coordination, Dorier et al. (2014) used data from the Parallel Workload Archive, from the period between January and September 2009. Through a simple optimistic model, they used the distribution of some concurrent jobs to show that there was a high probability of having multiple applications concurrently performing I/O operations, even when applications spent as little as 5% of their execution time on I/O.

Kim et al. (2010) characterized the scientific workload of the Spider (Lustre) HPC storage cluster, at Oak Ridge Leadership Computing Facility (OLCF). They considered the system utilization, the demands of read and write operations, idle time, and the distribution of read requests to write requests. The study summarized six months of observation. They demonstrated that the bandwidth usage and the inter-arrival time of requests can be modeled as a Pareto distribution.

Luu et al. (2015) analyzed the Darshan logs of over a million jobs executed during 2013 over Intrepid and Mira supercomputers, at the Argonne Leadership Computing Facility (ALCF), and Edison, at the National Energy Research Scientific Computing Center (NERSC). The work aimed at identifying behaviors that impacted performance to guide future optimizations. They pointed out that every widely adopted I/O paradigm (file per process, shared file, subsetting I/O) is represented among the best-performing and worst-performing applications. Hence, the usage of a paradigm could not provide any guarantees in terms of performance. Regarding throughput, they demonstrated that almost a third of the jobs had an aggregated throughput of no more than 256MB/s. They also pointed out that over a third of the jobs spent more time in metadata operations than actually transferring data. Additionally, despite the existence of high-level parallel libraries, three-quarters of the jobs used only POSIX to perform I/O.

In (Xie et al. 2012), the authors presented a characterization of the storage performance of the Cray XK6 Jaguar supercomputer while examining the implications of those results for application performance. They observed and quantified limitations from competing traffic, concurrency, interference, and stragglers of writes on shared files.

In the same way as the works presented in this section, we also seek to characterize the workload of the machine by proposing a strategy to extract information about the jobs' I/O behaviors from traces. Nonetheless, differently from Zoll et al. (2010) and Wang et al. (2004), we based our approach on traces transparently collected by the Darshan profiler, a method also used by Luu et al. (2015). However, differently from the latter, we also employ the concept of I/O phases to group similar I/O behaviors and to characterize the I/O workload of the machine. Such information can be used to improve the I/O performance of the applications and to guide system administrators to improve resource usage.

7 Conclusion and Future Work

In this paper, we have evaluated the I/O performance of Santos Dumont, one the largest supercomputers in Latin America. Such analysis is essential to ensure the efficient use of the machine, as many applications spend a significant portion of their execution time in I/O operations. Additionally, our study provides valuable information that can be taken to guide future upgrades on the I/O subsystem.

This paper was organized in three main parts. In Part I, we presented a study of the I/O sub-system, by monitoring a week of the supercomputer's activity. This study aimed to understand the I/O demands and detect possible issues that translate into poor I/O performance, in addition to identifying the most representative applications. As rarely done for data from production systems, we made the data set of metrics collected from the machine publicly available to encourage future research: `https://gitlab.com/jeanbez/ijhpca-sdumont`.

During the study presented in Part I, an unexpected behavior observed with the OLAM was reported. In Part II, we reproduce the behavior to find large performance differences between the MPI implementations available for users. We have further investigated this difference by conducting experiments with the BT-IO benchmark, the IOR benchmarking tool, and with a custom benchmark. Results indicated that the observed difference happens for small requests (of approximately 1 KB), and comes the fact some implementations have a lower performance for asynchronous point-to-point communication. Our detailed documentation of the phenomenon provides information that can be used to improve future implementations of collective I/O operations. Furthermore, an important contribution of this work is advice to be given to SDumont users, as it was observed over 20% of jobs use BullxMPI, the implementation that presented the worst results in our analysis. By helping users achieve better performance for their applications, we promote better usage of the machine.

From the report of unexpected performance for an application, identifying and understanding the issue require considerable effort. In Part III, we discuss the challenge of obtaining information about jobs I/O behaviors without relying on users. That is important because users often do not know the specificities of applications, and describing I/O behavior is not trivial. We proposed a strategy for obtaining a characterization of jobs I/O phases from Darshan traces. After presenting our approach by applying it to OLAM traces, we used GROMACS as a case study. In our analysis of the GROMACS application, we were able to detect four distinct groups that characterize and represent the I/O behavior of these jobs.

As future work we plan to extend our investigation with Darshan traces to the ten most used applications in the machine and the top ten regarding I/O demands. Such analysis will further assist in characterizing and optimizing SDumont I/O infrastructure.

Acknowledgment

The authors received funding from various sources during this work. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. these words), nothing we can do about it It has also been partially supported by: the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 800144; the EU H2020 programme and from MCTI/RNP-Brazil under the HPC4e Project, grant agreement n° 689772; CNPq Brazil; the Petrobras project, grant n°2016/00133-9; and from the European Commission (EC) (H2020/RIA) through the EoCoE project — "Energy-oriented Centre of Excellence for computer applications", project code 676629. We would also like to thank RICAP, partially funded by CYTED, Ref. 517RT0529. The authors acknowledge the National Laboratory for Scientific Computing (LNCC/MCTI, Brazil) for providing HPC resources of the SDumont supercomputer, which have contributed to the research results reported within this paper. URL: `http://sdumont.lncc.br`.

This project was partially funded by the European Union.

References

Boito FZ, Inacio EC, Bez JL, Navaux POA, Dantas MAR and Denneulin Y (2018) A Checkpoint of Research on Parallel I/O for High-Performance Computing. *ACM Computing Surveys* 51(2): 23:1–23:35. DOI:10.1145/3152891.

Brightwell R (2004) A New MPI Implementation for Cray SHMEM. In: Kranzlmüller D, Kacsuk P and Dongarra J (eds.) *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 11th European PVM/MPI Users' Group Meeting Budapest, Hungary, September 19 - 22, 2004. Proceedings*. Berlin, Heidelberg: Springer, pp. 122–130.

Brightwell R (2005) A Comparison of Three MPI Implementations for Red Storm. In: Martino BD, Kranzlmüller D and Dongarra J (eds.) *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 12th European PVM/MPI Users' Group Meeting Sorrento, Italy, September 18-21, 2005. Proceedings*. Berlin, Heidelberg: Springer, pp. 425–432.

Carneiro AR, Bez JL, Boito FZ, Fagundes BA, Osthoff C and Navaux POA (2018) Collective I/O Performance on the Santos Dumont Supercomputer. In: *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. pp. 45–52.

Carns P, Harms K, Allcock W, Bacon C, Lang S, Latham R and Ross R (2011) Understanding and Improving Computational Science Storage Access Through Continuous Characterization. *Trans. Storage* 7(3): 8:1–8:26.

Carns P, Lang S, Ross R, Vilayannur M, Kunkel J and Ludwig T (2009) Small-file access in parallel file systems. In: *2009 IEEE International Symposium on Parallel Distributed Processing*. pp. 1–11. DOI:10.1109/IPDPS.2009.5161029.

Crouch C and ATOS (2015) ATOS supports Brazil in the race to High-Performance Computing with the installation of a Bull supercomputer in Petropolis (RJ). URL `https://atos.net/en/2015/press-release/deals-contracts-press-releases_2015_09_07/pr-2015_09_07_01`.

Dorier M, Antoniu G, Ross R, Kimpe D and Ibrahim S (2014) CALCioM: Mitigating I/O interference in HPC systems through cross-application coordination. In: *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. IEEE, pp. 155–164.

Dreher M and Raffin B (2014) A Flexible Framework for Asynchronous in Situ and in Transit Analytics for Scientific Simulations. In: *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. pp. 277–286.

Dunn OJ (1961) Multiple comparisons among means. *Journal of the American Statistical Association* 56(293): 52–64.

Feltovich N (2003) Nonparametric Tests of Differences in Medians: Comparison of the Wilcoxon–Mann–Whitney and Robust Rank-Order Tests. *Experimental Economics* 6(3): 273–297.

Kim Y, Gunasekaran R, Shipman GM, Dillow DA, Zhang Z and Settlemyer BW (2010) Workload Characterization of a Leadership Class Storage Cluster. In: *2010 5th Petascale Data Storage Workshop (PDSW '10)*. pp. 1–5.

Lawrence M, Huber W, Pags H, Aboyoun P, Carlson M, Gentleman R, Morgan MT and Carey VJ (2013) Software for computing and annotating genomic ranges. *PLOS Computational Biology* 9(8): 1–10. DOI:10.1371/journal.pcbi.1003118.

Luu H, Winslett M, Gropp W, Ross R, Carns P, Harms K, Prabhat M, Byna S and Yao Y (2015) A multiplatform study of I/O behavior on petascale supercomputers. In: *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, pp. 33–44.

MCTIC-LNCC (2016) Supercomputador Santos Dumont opera com 51 projetos científicos e tecnológicos. URL `http://www.lncc.br/lncc/supercomputador.php?idt_noticia=988`.

NASA (1994) Nas Parallel Benchmarks (NPB). `https://www.nas.nasa.gov/publications/npb.html`. Accessed: November 2017.

Neau H, Fede P, Laviéville J and Simonin O (2013) High Performance Computing (HPC) for the Fluidization of Particle-Laden Reactive Flows. In: *14th International Conference on Fluidization - From Fundamentals to Products (2013)*.

Nieuwejaar N, Kotz D, Purakayastha A, Ellis CS and Best ML (1996) File-access characteristics of parallel scientific workloads. *IEEE Transactions on Parallel and Distributed Systems* 7(10): 1075–1089.

Osthoff C, Souto RP, Vilasbas F, Grunmann P, Dias PLS, Boito F, Kassick R, Pilla L, Navaux P, Schepke C, Maillard N, Panetta J, Lopes PP and Walko R (2012) Improving atmospheric model performance on a multi-core cluster system. In: Yucel I (ed.) *Atmospheric Model Applications*, chapter 1. IntechOpen.

Pronk S, Pll S, Schulz R, Larsson P, Bjelkmar P, Apostolov R, Shirts MR, Smith JC, Kasson PM, van der Spoel D, Hess B and Lindahl E (2013) Gromacs 4.5: a high-throughput and highly parallel open source molecular simulation toolkit. *Bioinformatics* 29(7): 845–854.

Schmuck F and Haskin R (2002) GPFS: A Shared-Disk File System for Large Computing Clusters. In: *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST '02. Berkeley, CA, USA: USENIX Association.

SUN (2007) High-performance storage architecture and scalable cluster file system. Technical report, Sun Microsystems, Inc. URL `http://www.csee.ogi.edu/~zak/cs506-pslc/lustrefilesystem.pdf`.

The HDF Group (1997–2016) Hierarchical Data Format, v5.

van der Spoel D and Hess B (2011) Gromacsthe road ahead. *Wiley Interdisciplinary Reviews: Computational Molecular Science* 1(5): 710–715.

Vinter B, Bjrndalen JM, Anshus OJ and Larsen T (2004) A Comparison of Three MPI Implementations. In: *Proceedings of Communicating Process Architectures (CPA 2004)*. Morgan Kaufman, pp. 127–136.

Wang F, Xin Q, Hong B, Brandt SA, Miller E, Long D and McLarty T (2004) File system workload analysis for large scale scientific computing applications. Technical report, Lawrence Livermore National Lab (LLNL), Livermore, CA, USA.

Xie B, Chase J, Dillow D, Drokin O, Klasky S, Oral S and Podhorszki N (2012) Characterizing output bottlenecks in a supercomputer. In: *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. pp. 1–11.

Zoll Q, Zhu Y and Feng D (2010) A study of self-similarity in parallel I/O workloads. In: *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. pp. 1–6.

Author Biographies

Jean Luca Bez Ph.D. student in Computer Science (2017) at the Federal University of Rio Grande do Sul (UFRGS), under the supervision of Prof. Dr. Philippe O. A. Navaux (UFRGS) and Prof. Dr. Toni Cortes (UPC, BSC), in the Parallel and Distributed Processing area. Master in Computing (2015) by the Federal University of Rio Grande do Sul (UFRGS) in the same area. He received the 1st place in the Brazilian Contest of Master's Dissertations in Computer Architecture and High-Performance Computing (WSCAD CTD 2017). Bachelor of Science in Computer Science from the Universidade Regional do Alto Uruguai e das Misses (URI) Erechim, having received the URI 2015 Best Student Award and the SBC Highlight Student Award. His main areas of research are Parallel File Systems, High-Performance I/O, Parallel I/O.

Andr Ramos Carneiro has Technical License in Information and Communication Technology (ISTCC-P, 2006) and is Technologist at the National Laboratory for Scientific Computing (LNCC), leading the HPC Support team, and supervising the Information Technology Support Service and Computational Infrastructure since 2012. He is an assistant instructor of the System Administrator training courses for the Networks School (ESR) of the Brazilian National Research and Educational Network (RNP). From 2005 to 2009 worked as Systems Administrator and technical support for the Weather Forecast and Atmospheric Research Group at LNCC. Research interests in high performance computing, distributed computing, numerical weather forecast models, high performance storage, and parallel I/O.

Pablo J. Pavan is Bachelor in Computer Science from the Regional University of the Northwest of the State of Rio Grande do Sul (UNIJU), Brazil. He currently is a Master's Degree student in Computer Science from the Federal University of Rio Grande do Sul (UFRGS), Brazil, in the Parallel and Distributed Processing area. His experiences are in the field of Computer Science, with emphasis on I/O in High-Performance Computing.

Valria S. Girelli is an undergraduate student in Computer Science at Federal University of Rio Grande do Sul (UFRGS), Brazil. She is currently an undergraduate researcher at the Parallel and Distributed Processing Group (GPPD) at the Informatics Institute, UFRGS, with experience in I/O in High-Performance Computing systems and Computer Architecture and Organization.

Francieli Zanon Boito is a Marie Sklodowska Curie fellow in the Informatics Laboratory of Grenoble (LIG), in France. She received her Ph.D. in Computer Science in 2015 by the Federal University of Rio Grande do Sul, in Brazil, and the University of Grenoble Alpes, in France. Her research interests include performance and energy efficiency of storage systems for high-performance computing.

Bruno Alves Fagundes is a network security specialist at UNESA (2011) and a technologist at the Laboratrio Nacional de Computao Cientfica since 2013, working in the support and administration of the computational platform of LNCC and Supercomputer SDumont. His current research includes Input and Output Data in Supercomputing Environments.

Carla Osthoff is a researcher at National Laboratory for Scientific Computing (LNCC-Brazil), she is LNCC High Performance Computing Center (CENAPAD/LNCC) coordinator and head from CENAPAD HPC research group. She received his D.Sc. degree in Computer and Systems Engineering from the Federal University of Rio de Janeiro, Brazil. She has been involved in research projects in bioinformatics, Oil and Gas and Atmospheric Simulations research areas. Her main research areas are in high performance Computing, Cluster Computing, Hybrid Parallel Computing, High Performance Scientific Computing Applications, Parallel Programming Models, and Parallel I/O.

Pedro Leite da Silva Dias Bachelor degree in Applied Mathematics (Univ. of So Paulo/USP in 1974), MSc and Ph.D.in

Atmospheric Sciences by the Colorado State University in 1977 and 1979, respectively. Professor of Institute of Astronomy, Geophysics and Atmospheric Sciences (IAG) of the University of So Paulo - IAG/USP since 1975. Director of IAG since 2017. Former Director of the National Laboratory for Scientific Computing of the Ministry of Science, Technology, and Innovation from 2007 to 2015. Member of the Brazilian Academy of Sciences. Scientific Honor Recognition by the Ministry of Science and Technology in 2002 and honored by the American Meteorological Society for his contribution to the South American Meteorology in 2012. Visiting Researcher at NCAR and NCEP in the USA on several occasions since 1982. Senior researcher of the National Institute for Space Research (INPE) and head of the Center of Weather Forecasting and Climate Research (CPTEC) between 1988 and 1990. President of the Brazilian Meteorological Society between 1992 and 1994 and Science Director of the Society from 2006 to 2008. Coordination of the Environmental Area of the Institute of Advanced Studies of USP from 1996 to 2008 and of the Regional Weather and Climate Studies Laboratory (MASTER) at USP since 1995. Published about 120 papers, book chapters, mostly in international journals and about 240 complete papers in national and international scientific events. Advisor of approximately 65 MSc and Ph.D. students. Participated in several international committees of the World Meteorological Organization and international and national science panels. Research interests in complex modeling of the atmosphere, parallelization of atmospheric models, climate variability, weather forecasting, air pollution modeling, the role of tropical heat sources, uncertainty estimates in weather and climate predictions, climate change issues and interaction with users of meteorological forecasts. More than 11.000 citations in Google Citations (H=43) see `http://scholar.google.com.br/citations?hl=en&user=sJs905UAAAAJ`.

Jean-Franois Mhaut is Professor of Computer Science at the Universit Grenoble Alpes (UGA) since 2003. He is a member of the Computer Science Laboratory of Grenoble (LIG). His current research includes embedded systems as well as all aspects of high performance computing including runtime systems, multithreading and memory management in NUMA multiprocessors, multi-core and hybrid programming. Jean-Franois Mhaut supervised 35 PhD students. He was also involved in the European FP7 ICT Mont-Blanc projects. He is member of the joint laboratory for exascale computing (JLESC) between Inria and the University of Illinois at Urbana Champaign (UIUC), Argonne National Laboratory (ANL/MCS) and the Barcelona Supercomputing Center (BSC). He was also PI of the FP7 IRSES HPC GA (High Performance Computing for Geophysics Applications). In 2017, he held a 6 months position of visiting researcher at LNCC funded by the French Consulate in Rio de Janeiro and the CNPq.

Philippe Olivier Alexandre Navaux Professor Informatics Institute, university UFRGS, Graduated Electronic Engineering, UFRGS, 1970, Master Applied Physics, UFRGS, Ph.D. Computer Science, INPG, Grenoble. Professor graduate and undergraduate courses Computer Architecture and High Performance Computing. Leader GPPD, Parallel and Distributed Processing Group. Projects financed by government agencies H2020, Finep, RNP, CNPq, Capes. International Cooperation with France, Germany, Spain, and the USA. Industrial projects with Microsoft, Intel, HP, DELL. Has oriented near 100 Master and Ph.D. students, has published near 400 papers in journals and conferences. Member SBC, SBPC, ACM, IEEE. Consultant funding organizations DoE (USA), ANR (FR), FINEP, CNPq, CAPES, FAPERJ, FAPESP, FAPERGS, FAPEMIG, and others.