Lawrence Berkeley National Laboratory
LBL Publications

Title
Uncovering I/O demands on HPC platforms: Peeking under the hood of Santos Dumont

Permalink
https://escholarship.org/uc/item/3s83m0wm

Authors
Carneiro, André Ramos
Bez, Jean Luca
Osthoff, Carla
et al.

Publication Date
2023-12-01

DOI
10.1016/j.jpdc.2023.104744

Copyright Information
This work is made available under the terms of a Creative Commons Attribution License, available at https://creativecommons.org/licenses/by/4.0/

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Uncovering I/O Demands on HPC Platforms: Peeking Under the Hood of Santos Dumont

André Ramos Carneiro[a,c,*], Jean Luca Bez[b], Carla Osthoff[c], Lucas Mello Schnorr[a], Philippe O. A. Navaux[a]

[a] *Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS), Brazil*
[b] *Lawrence Berkeley National Laboratory (LBNL), USA*
[c] *National Laboratory of Scientific Computation (LNCC), Brazil*

Abstract

High-Performance Computing (HPC) platforms are required to solve the most diverse large-scale scientific problems in various research areas, such as biology, chemistry, physics, and health sciences. Researchers use a multitude of scientific softwares, which have different requirements. These include input and output operations, which directly impact performance due to the existing difference in processing and data access speeds. Thus, supercomputers must efficiently handle mixed workload when storing data from the applications. Understanding the set of applications and their performance running in a supercomputer is paramount to understanding the storage system's usage, pinpointing possible bottlenecks, and guiding optimization techniques. This research proposes a methodology and visualization tool to evaluate a supercomputer's data storage infrastructure's performance, taking into account the diverse workload and demands of the system over a long period of operation. As a study case, we focus on the Santos Dumont supercomputer, identifying inefficient usage, problematic performance factors, and providing guidelines on how to tackle those issues.

Keywords: Lustre, Parallel File System, High-Performance Storage, I/O Workload, I/O Characterization, Metadata

1. Introduction

Supercomputers, with hundreds to thousands of compute nodes, dominate the High-Performance Computing (HPC) environments. These HPC systems are used to solve the most diverse problems in various science domains: biology, chemistry, physics, and health sciences. Researchers of different areas use a multitude of scientific software, which have different requirements. For instance,

*Corresponding author
Email addresses: arcarneiro@inf.ufrgs.br (André Ramos Carneiro), jlbez@lbl.gov (Jean Luca Bez), osthoff@lncc.br (Carla Osthoff), schnorr@inf.ufrgs.br (Lucas Mello Schnorr), navaux@inf.ufrgs.br (Philippe O. A. Navaux)

applications can be serial or parallel, and read/write different amounts of data in various formats and sizes. This scenario leads to the supercomputers having to handle mixed workloads.

The evolution of processing chips and high-speed networks allows supercomputers to process larger datasets. Moreover, the infrastructure that stores these datasets also has to provide high-performance access so that the applications can perform their input and output (I/O) operations efficiently. For an HPC environment, it's not just the amount of floating-point operations per second (FLOPs) that affects the performance but also how much data per second they can effectively read from and write to the storage system.

Parallel File Systems (PFS), a decentralized storage system in which dedicated machines act as data servers that reduce the overhead of processing I/O requests, are the *de-facto* file system type for HPC systems. Lustre [1] is one of the most adopted PFS on HPC systems, representing $\approx 20\%$ of the file systems used on IO500 list [2]. Although advances in data storage architectures provide a better performance, for instance by using SSD devices, there is still a considerable performance gap between how fast the system can handle I/O operations and how fast it can process the data. This difference affects how supercomputers can be used productively for new scientific discoveries. More research is being done with the rapid expanse of supercomputers while generating more data to be read and written, making the shared data storage infrastructure one of the main bottlenecks for achieving sustainable performance. The PFS is unable to keep providing performance due to rising concurrency and interference [3, 4]. Aside from the I/O operations, another key factor in the HPC storage management are the metadata operations, which are responsible for maintaining the file system directory tree, file access permissions and ownership, timestamps, attributes, etc. As the datasets gets bigger, the metadata performance becomes critical and can quickly turn into a bottleneck [5].

Lawrence et al. [6] demonstrate that different scientific applications have their performance impacted in diverse ways by Lustre, with some using the resources more efficiently than others. Some factors that impose limitations and negatively impact performance of Lustre are misaligned access patterns [7], load imbalance between storage servers [8], and resource contention [9]. Besides, we should also consider that the existing I/O stack exposes a plethora of tunable parameters seeking to provide performance improvements to diverse workloads. However, the misconfiguration of such parameters due to the users' lack of knowledge about its application's I/O operations can add to the observed poor performance. Furthermore, a poorly performing I/O application could also negatively impact all others currently running in the system because storage is shared.

Our research aims to understand the impact of data storage in a supercomputer by evaluating Lustre's performance concerning the diverse I/O and metadata workloads from different domains and their demands. We studied and compared the behavior over two periods comprising three months of operation, from March to May 2020 and 2021, when there was 16.50 PiB of data movement through 109.787 Bi I/O operations. We provide a methodology to visualize performance factors, such as small request sizes, load imbalance, and resource

contention. We use the *Santos Dumont* Supercomputer (*SDumont*) [10] as a case study because little is known about the impact of its storage and I/O stack configuration on the application set that runs daily in that production machine. We make the following contributions:

- We developed a methodology to collect, analyze, and visualize I/O data from the PFS. We used open-source software that does not require administrative privileges, allowing it to be easily implemented and reproduced.
- We developed a web application to streamline the visualization and analysis of the PFS usage data. Such a tool makes it easier to reproduce the analysis and study different periods of interest.
- We investigated the I/O workload and usage behavior from Lustre's Object Storage Targets (OST), and SDumont's compute nodes. Our study shows that the workload demand is not dominated by a single type of operation and can significantly vary across the period.
- We analyzed individual OST usage and demonstrated a significant load imbalance across them during normal system operation.
- By crossing the I/O usage metrics from the compute nodes with information from the job scheduling management system, we were able to identify problematic applications that could lead to overall performance degradation at the PFS servers.
- The analysis and characterization of the metadata operations show that there is a considerable demand, with the metadata accounting for 60% of all file system operations.
- All the data collected for this study and the toolset developed to analyze and visualize the data are available at the Companion Repository ([11]).

The remainder of this paper is structured as follows. Section 2 contextualizes this study and discusses related work. Section 3 presents Lustre's architecture and the infrastructure of SDumont, and in Section 4 our methodology. Results are discussed in Section 5. Section 6 summarizes the lessons learned and compares our findings with other systems. Finally, concluding remarks and future work are presented in Section 7.

2. Related Work

Efficient I/O performance is a critical part of modern supercomputing. Studies show a growing need for a better understanding of the storage infrastructure and explaining the attained I/O performance by the applications. Huong et al. [12] analyzed Darshan's logs [13] from the execution of more than one million jobs during 2014 on three leading HPC supercomputer platforms: Intrepid and Mira at ALCF and Edison at NERSC. The authors observed that the aggregate

throughput for three-quarters of the applications never exceeds 1 GB/s, roughly 1% of the average peak platform bandwidth available.

Lockwood et al. [14] used system monitoring, benchmarking, and active probing on the I/O storage infrastructure (Lustre File System and GPFS) of two leadership-class HPC centers (NERSC and ALCF) over a year of production operation to identify the reasons for low performance and how to improve performance. They investigated applications on multiple time scales, seeking to identify trends in absolute performance and variability, the high CPU load on the data servers, and contention for bandwidth. The results demonstrate that variations occur in the I/O performance during regular operation and different execution periods. Wan et al. [15] follows a similar approach by designing a lightweight parallel test harness to periodically collect I/O performance and job status traces for applications running on production HPC systems. They seek to model the transitions between different I/O performance states on production HPC systems as a classification problem and harness that model to mitigate contention while improving data access.

Patel et al. [8] proposed a tool to analyze the log data of Lustre PFS obtained with the Lustre Monitoring Tool (LMT) [16] in one year of operation (2018) from the parallel storage system at NERSC HPC data center, shared by Edison and Cori supercomputers. This study shows that the Lustre used is dominated by read operations, even with a Burst-Buffer system (in Cori).

Another approach was proposed by Sivalingam et al. [17], where the authors used LASSi, a tool to analyze application usage and contention caused by shared resources (file system or network) on the Lustre File System deployed at ARCHER supercomputer. LASSi combines Lustre statistics and job information to calculate derived metrics, identifying a particular class of jobs that generated excessive I/O load. The information gathered could be used to avoid slowdowns, understand application I/O behavior in a shared file system, and guide the HPC file systems' expansions. Similarly, Wadhwa et al. [18] introduce an end-to-end control plane, to optimize I/O resource allocation. They employ a user-level library to intercept file I/O calls and transparently provide runtime optimization for the client application. They seek to provide an application-agnostic global view of all resources to the MetaData Server.

Kunkel et al. [19] investigate and quantify the user-perceived slowdown on the shared file system deployed at supercomputers. They introduce systematic I/O performance monitoring using probes to derive a slowdown factor. They evaluate three European HPC systems: JASMIN at UK CEMS, ARCHER at UK NSS, and Mistral at DKRZ. To assess the short-term interference, they execute the IO-500's standards benchmarks MDTest and IOR [20] on ARCHER. They evaluated the long-term performance influence on up to 60 days of collected statistics. In both cases, the approach revealed interference in system utilization but still lacked the server-side information to complement and aid in explaining the performance on the server side.

Betke et al. [21] aims to identify anomalies or high workloads from jobs' telemetric data through a workflow based on Machine Learning. The analysis is automated by splitting each job's monitoring data into smaller portions and

generating a footprint. A classification method is applied to the footprint dataset to sort the applications with similar I/O behavior, isolating applications with harmful I/O patterns for optimization. They used one week's data from DKRZ's monitoring system at Mistral, which classified 33,193 jobs on 8 I/O behavior classes. Even though a promising approach, it needs some tuning on the automatic class labeling.

Those methodologies rely on the periodic execution of probes (usually I/O benchmarks that run on the shared file system), application-level profiling (Darshan), or tools that require administrative privileges. Moreover, profiling tools such as Darshan, commonly deployed and enabled by default in such platforms, cover only a tiny percentage of running jobs [12, 22]. Since users can unload this module, and until recently, it required applications to use MPI to collect metrics, more than those metrics are needed to provide a full image of the system. Our study proposes a broader methodology to provide a bigger picture of system I/O utilization, tracking inefficient behavior from the *Storage Devices* to the *Compute Nodes*. We used open-source software that does not require administrative privileges, allowing it to be easily implemented and reproduced.

The metadata performance on HPC storage systems is also gaining attention in various works, exposing the need for improvements and better understanding. Chasapis et al. [23] evaluates the performance of Lustre's metadata server (MDS). The study concludes that Lustre's metadata performance does not scale when increasing the number of sockets and cores, with the MDS's back-end device not being the limiting factor, but rather, its software not being multi-core ready. Zhao et al. [24] analyze and evaluate the performance of scientific applications on four representative file systems (S3FS, HDFS, Ceph, and FusionFS) on three cloud platforms (Kodiak cluster, Amazon EC2, and FermiCloud). They demonstrate that a distributed management approach brings orders of magnitude improvements over the performance of centralized deployments, presenting almost linear scalability (up to 512 nodes).

Kunkel et al. [25] developed a new metadata benchmark called MDWorkbench, capable of emulating many concurrent users, providing latency profile and throughput. They used it to evaluate four parallel file systems (GPFS IBM Spectrum Scale, Lustre, Cray's Datawarp, and DDN IME) on five computing platforms: Cooley at ACLF, Mistral at DKRZ, IME at Dusseldorf, Shaheen II at KAUST, and Cori at NERSC. The results show that capturing the contention caused by metadata changes and identifying the relation between observed throughput and latency was possible. To the best of our knowledge, our work is the first to integrate I/O and metadata analysis in the same approach while characterizing their needs on a petascale HPC system.

3. Lustre Architecture and its Deployment on SDumont

In this section, we discuss the Lustre's architecture and the motivation for using the SDumont supercomputer.

3.1. Lustre's Architecture

Lustre PFS is an open-source client-server file system implemented entirely on the Linux Kernel, developed for high-performance environments. It provides a POSIX compliant namespace and scalable I/O resources. Instead of using traditional block-based storage, where the files are divided into equal-sized blocks and the metadata management is coupled with the I/O management, Lustre uses a distributed object-based storage, where a file can be divided into objects of different sizes that store the data and the metadata management is decoupled. This approach entrusts the block storage management to dedicated backend servers, which diminish problems associated with scalability and performance of the traditional centralized file systems. There are two types of objects on Lustre: (I) data objects containing byte arrays used to store file data, and (II) metadata objects containing key-value data used to implement the directory tree structure and the file/directory attributes.

The key components of Lustre's architecture are Metadata Servers (MDS), Metadata Targets (MDT), Object Storage Servers (OSS), Object Storage Targets (OST), Clients, and Lustre Network (LNET). The MDS are responsible for managing all metadata operations on the file system, such as deciding where a data object will be stored, setting and retrieving file/directory attributes, and exporting MDTs to the clients. The MDTs are the backend storage responsible for holding the metadata objects. OSS handle the file I/O operations and export one or more OSTs to the clients, while OSTs are responsible for storing the file data object. Clients combine the MDTs and OSTs in a single FS while communicating the users' requests to the MDSs or OSSs. The Lustre Network (LNET) provides the communication infrastructure.

Lustre uses the *data striping* technique, which divides a file into data chunks among selected OSTs. The size of the chunks is referred to as `stripe_size`, and the number of OSTs by which the file will be split into is referred to as `stripe_count`. Striping can improve performance in file access as it makes possible to aggregate multiple data servers' bandwidth to access a single file using parallel I/O operations.

3.2. The SDumont

The *Santos Dumont* Supercomputer (*SDumont*) [10], a Bull/Atos machine located at the National Laboratory for Scientific Computing (LNCC) [26] in Brazil, is an example of an HPC environment with a significant heterogeneity of research on several areas of knowledge, each with a specific set of scientific software. The primary ones are Chemistry (21.3%), Physics (17.1%), Engineering (12.6%), and Biological Sciences (10,1%). SDumont is the Tier-0 of the National High-Performance Computing System (*SINAPAD*) [27] and one of the largest in Latin America. It currently has 133 research projects in progress, with a daily average of ≈ 200 concurrent jobs. In operation since 2016, SDumont has a total of $18,424$ CPU cores distributed across 758 compute nodes (CN).

To store all the data, SDumont has a shared storage with Lustre *PFS* deployed through the CRAY/HPE ClusterStor 9000 v3.3, composed of one MDS and

ten OSS. Each server has one target (MDT or OST) made out of 40 HDD in RAID6 format. The total storage capacity of the deployed system is 1.7 PiB. An Infiniband FDR (56 Gb/sec) fat-tree full-nonblocking network connects the storage servers and the client nodes. Lustre is mounted on the client nodes with the default `stripe_count = 1` and `stripe_size = 1 MiB`. According to the ClusterStor 9000 technical specifications [28], the peak performance a Lustre system with similar characteristics as the one implemented on SDumont should achieve, without considering the effects of cache, is 45 GB/s. The aggregate network bandwidth to access the Lustre system is 70 GB/s, which should not impose a bottleneck on the communication.

Bez et al. [29] investigated the performance difference between MPI implementations when issuing collective operations, focusing on two specific applications on the SDumont supercomputer. The study includes an initial Lustre workload characterization, using monitoring data collected during a week of operation. However, there was still no comprehensive data on Lustre's behavior and its usage on SDumont. Without a detailed analysis over a representative period, it is challenging to assess if the Lustre delivers the required performance or limits the scalability of applications.

4. Analysis and Visualization Methodology

We propose a pipeline workflow to study the Lustre PFS utilization on supercomputers. It is comprised of three steps, as depicted in Figure 1: (1) **Collect** performance metrics from the nodes; (2) **Pre-process** the raw metric data and store it in an *easy-to-use* format; and (3) **Analyze** the data.

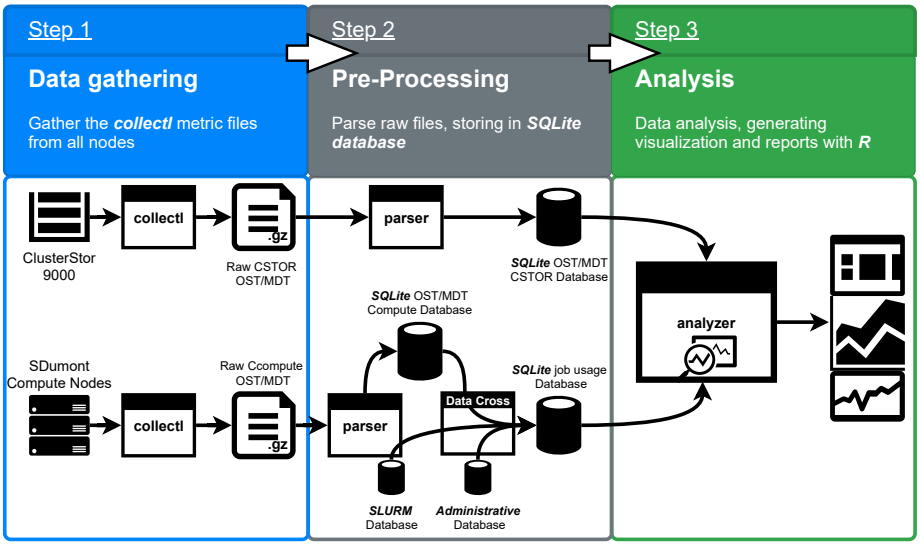

Figure 1: Data collection and analysis workflow

Table 1: Lustre I/O metrics

Metric	Description
$reads$	Number of read operations.
$read_{kb}$	Volume of data read (KiB).
$read_{size}$	Block size of read operation ($read_{kb}/reads$).
$read_{qo}$ [17]	Quality of read operation (($reads * 1024)/read_{kb}$).
$writes$	Number of write operations.
$write_{kb}$	Volume of data written (KiB).
$write_{size}$	Block size of write operation ($write_{kb}/writes$).
$write_{qo}$ [17]	Quality of write operation (($writes * 1024)/write_{kb}$).
CF_{bw} [14]	Bandwidth Coverage Factor of a job.
LI	Load Imbalance.
SMA_{3HR}	Simple Moving Averages of three hours.

4.1. Data Gathering Step

We favored *collectl*[30], an open-source system performance monitoring tool capable of collecting metrics from various subsystems, such as CPU, disk, inodes, memory, and network. The advantages of *collectl* are that it is easily deployed and configured, does not need administrative privileges to install or use, is capable of storing the collected metrics on a locally compressed file or sending them over the network, and has a flexible API enables the development of custom modules (plugins). *collectl-lustre*[31] is a special plugin to collect I/O metrics and metadata counters exposed by Lustre's kernel modules on servers and clients. The default I/O metrics provided by the *collectl-lustre* plugin are the number of read and write operations and the volume of data transferred in KiB. In the following steps, additional metrics are calculated, summarized at Table 1.

The Lustre's metadata kernel module exposes counters common to the MDS and Compute Node clients, and other counters exclusive to each. For this reason, *collectl-lustre* retrieves information about $fopen$ and $fclose$ (file open and close requests), *getattr* and *setattr* (get and set file/dir attributes), and *sync* (synchronizes data to the FS) counters on both the MDS and clients. Among the counters exclusive to the MDS are *unlink* (file/directory removals) and $statfs$ (return FS statistics). The *seek* (change file pointer position) counter is available only on clients.

The *collectl* tool version 4.3.1, with *collectl-lustre*, was installed on all ClusterStor Lustre nodes (MDS and OSSes), and the 758 Compute Nodes. We store the metrics on a daily compressed file at the local `/tmp` FS to avoid interference with Lustre. The overhead imposed by *collectl* on the monitored nodes was negligible ($< 0.1\%$ of CPU). The collection of data in short time intervals creates significant demand for storage space and CPU resources during the *Pre-Processing* and *Analysis* steps. For this reason, we collected the Lustre utilization metrics every 15 seconds. Our initial evaluation considered 1s intervals. However, we noticed that for a single node, *collectl* would generate a file of $>$ 1MB per day.

Considering the 756 nodes in the system, each day would require ≈ 760MB, which in a month would account for ≈ 21GB worth of metrics. In contrast, when increasing the interval to 15s, we observed a monthly file of ≈ 1.5GB, a 14× reduction. Despite losing a certain degree of precision (because *collectl* provides an average value in that time interval), we found this choice still gave us a good overall picture of the system's utilization.

4.2. Data Pre-Processing Step

The files generated by *collectl* are collected from each node daily for **Pre-Processing**, which consists of parsing, cleaning, and inserting the data into an *SQLite*[32] database. The choice to use the SQLite format for storing the data was due to its mobility (the database is a single compact file, and any dataset stored in it can be easily transferred from one system to another), and it comes pre-installed on most Linux distributions. Besides, it has APIs for various data analysis tools and languages. We have two distinct datasets, one from the data collected at the *ClusterStor Nodes*(Figures 2a and 2b) and one from the SDumont Base *Compute Nodes* (Figures 2c and 2d).

The data collected from the *Compute Nodes* alone does not provide enough insight as to *"who, how, and why"* was using the Lustre file system. To contextualize the Lustre utilization, an intermediate *Data Crossing* process was utilized, combining data from (1) *Compute Nodes* dataset; (2) *SLURM*[1] resource manager, regarding the number of submitted jobs, application name, runtime, start and end time, the number of compute nodes, and the total number of cores; and (3) an internal administrative database used by SDumont's managers, which provides information about the Science Domain of each research project using the SDumont. By crossing the data from which nodes a job used, with which domain the job belongs to, and the metrics from the *Compute Nodes* dataset, a new dataset composed of time-series data was generated, called *Job Usage* (Figures 2e and 2f). Figure 2 depicts the structure of the datasets, where, aside from the previously mentioned metrics, `Timestamp` is the moment the metrics were collected, `OST Name` is the name of the Object Storage Target where the I/O metrics were collected, `Node Name` is the name of the compute node that used the Lustre PFS (through the OSTs for I/O or through the MDT for metadata), `jobid` is the identification number of the job that used the Lustre PFS through the compute nodes, `account` is the name of the Linux group that submitted the job, `Science Domain` is the name of the science domain belonging to the `account`, and `application` is the name of the application used by the submitted job.

During this step, two complementary I/O metrics are generated during the pre-processing step: (I) the average block (transfer) size in KiB and (II) the **Q**uality of **O**peration. *QO*, as proposed by Sivalingam et al. [17]. The latter is based on the default 1 MiB `stripe_size` of the Lustre file system on SDumont. Based on *QO*, the operation would make an optimal usage of Lustre when at

[1] https://slurm.schedmd.com/documentation.html

Timestamp	OST Name	reads	readkb	readsize	readqo	writes	writekb	writesize	writeqo

(a) *ClusterStor* I/O (OST)

Timestamp	getattr	setattr	getxattr	setxattr	statfs	sync	link	unlink	fopen	fclose	mkdir	rmdir

(b) *ClusterStor* Metadata (MDT)

Timestamp	Node Name	OST Name	reads	readkb	readsize	readqo	writes	writekb	writesize	writeqo

(c) *Compute Nodes* I/O (OST)

Timestamp	Node Name	getattr	setattr	fopen	fclose	fsync	seek

(d) *Compute Nodes* Metadata (MDT)

Timestamp	jobid	account	Science Domain	application	Node Name	OST Name	reads	readkb	readsize	readqo	writes	writekb	writesize	writeqo

(e) *Job Usage* I/O (OST)

Timestamp	jobid	account	Science Domain	application	getattr	setattr	fopen	fclose	fsync	seek

(f) *Job Usage* Metadata (MDT)

Figure 2: Datasets Structure.

least 1 MiB is read or written. Hence, a value of "1" represents optimal usage, and values in a **higher** order of magnitude represent poor quality.

4.3. Data Analysis Step

During this step, additional features are calculated: Bandwidth **C**overage **F**actor (CF_{bw}), **L**oad **I**mbalance (LI), and **S**imple **M**oving **A**verages (SMA) of each metric. The $CF_{bw}(1)$, as proposed by Loockwood et al. [14], represents the fraction of the system bandwidth that can be attributed to the job and is given by the amount of data transferred by the job divided by the amount of data transferred to/from Lustre. For instance, a job with CF_{bw} of 0.60 indicates that other competing jobs consumed 40% of the available resources.

$$CF_{bw}(job) = \frac{N_{bytes}(job)}{N_{bytes}(Lustre)} \tag{1}$$

The **standard deviation** (σ) measures the dispersion in a distribution of values regarding its central tendency. Considering the OSTs' load (amount of data read/written at **each timestamp**), it is possible to use the σ to evaluate how severe the load imbalance is. A σ of zero means that the load is evenly distributed. When the distribution presents a *"high"* σ, few OSTs handle more data than others. In contrast, a *"low"* σ represents that the load is better distributed. Since the σ is quantified as high or low regarding the mean (μ) load observed on the OSTs at each timestamp, we define the $LI = \frac{\sigma}{\mu}$ metric as a

Coefficient of Variation [2] to express better how severe the imbalance is. LI values below 0.5 can be considered low imbalance, values around 1 are moderate, and values above are considered as a severe imbalance.

The SMA [33], a time series analysis technique commonly used in the financial market, is an arithmetic mean of a variable within a certain time frame (tf), and that moves through a time series. For a metric m at timestamp t, the SMA_{tf} of m is given by (2):

$$SMA_{tf}(m) = \frac{1}{tf} \sum_{i=t-tf}^{t} m_i \tag{2}$$

The financial market data have a high variation over time, similar to the I/O usage on HPC. With the use of SMA, it is possible to identify a tendency throughout the period. We have tested many time frame intervals and opted to use 3 hours, which proved to be a reasonable length to reduce part of the "noise" from the high variability and amount of data while still providing a good representation.

Reproducing this study's analysis with a new dataset from SDumont or another HPC platform can be daunting. To streamline the analysis process, we developed a *web application* using Shiny [34]. The app consumes the datasets generated in Step 2 and produces the statistical and visual analysis for any time frame of interest, facilitating the reproducibility of the results. A reduced version of the app used in this study is publicly available at `http://arcarneiro.shinyapps.io/sdumont_lustre`. The requirements to run a full version of the app used in this study are a system with the Shiny Server, 100 GiB of storage, 16 GiB RAM, and a quad-core 2.20 GHz processor.

5. Glancing at the Lustre Filesystem

We collected usage metrics from three months (March, April, and May) of both 2020 and 2021 to study utilization of the Lustre PFS on SDumont, and compare the findings between years. The analysis of the PFS was divided into two parts. First, we analyzed the whole period (three months) using the *ClusterStor* dataset (Section 5.1). Second, we focused on a specific period of interest using the *Job Usage* dataset (Section 5.2). In each one, we compare the data from the two years to assess the evolution in the PFS utilization.

5.1. Overview of Lustre Usage

In this section we provide an overview of the Lustre PFS utilization, analyzing the operation data obtained during March, April, and May from the years 2020 and 2021. On Section 5.1.1 we focus on the I/O operations performed on the system with data collected from the ClusterStor's OSSs and OSTs. On Section 5.1.2 we evaluate the Metadata operations with data from the MDS.

[2] The Coefficient of Variation is a statistical measure of the dispersion of data points in a data series around the mean.

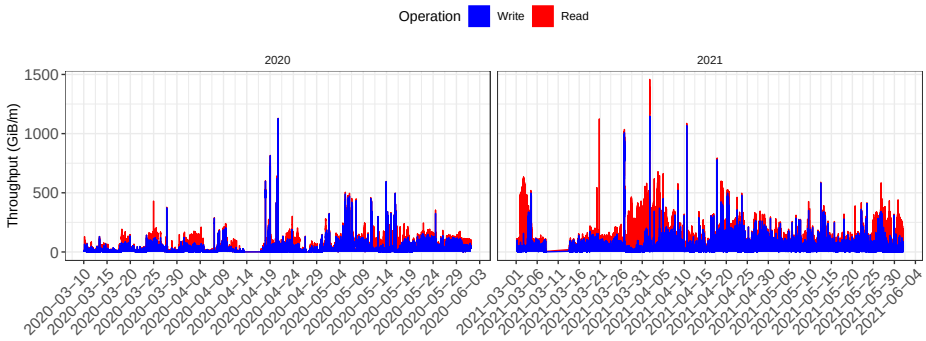

Figure 3: Data distribution for 3 months of 2020 (left) and 2021 (right).

5.1.1. I/O Data Analysis

We investigated the aggregated throughput of the storage system during the aforementioned period. Figure 3 illustrates the results in GiB/m for 2020 (left facet) and 2021 (right facet). Read (in red) and write (in blue) operations are stacked together based on when the metric was collected. In the two years, there were scheduled maintenance on ClusterStor to recover failed nodes and update the Lustre version. In 2020, the maintenance window was from April 13th to April 17th, and in 2021, from March 8th to March 12th. During those periods, the file system was inaccessible, resulting in no reported values.

Comparing both years it is possible to notice an increase in utilization. In terms of absolute values, in 2020, there was 1.8 PiB data read and 2.9 PiB written, and the system received 64.154 Bi read requests, and 1.234 Bi write requests. Comparing with 2021, the total volume of data read presented an increase of 4.7× (7.95 PiB), and the data written increased 1.5× (4.1 PiB). The number of read operations decreased 1.6× (39.102 Bi), and the write operations increased 4.3× (5.297 Bi). The increase in volume of data transferred to/from the system is accompanied by an increase in the number of jobs submitted to SDumont, where in 2020 there was 36,884 jobs and in 2021 145,793, representing 4× more jobs. The increase in read data volume but the decrease in operation suggests that applications are reading larger chunks of data, which is beneficial to attain high performance. However, writes seem to go in the opposite direction, with smaller request sizes which are proven to harm performance [35, 36].

During the whole period of the collected data, and considering the entire file system (sum of all OSTs), the highest throughput peaks are attributed to write operations on both years. The maximum achieved in 2020 was 1,127 GiB/m (at 2020-04-20 14:30:00) and in 2021 was 1,145 GiB/m (2021-04-01 18:50:00), which represents ≈ 41.74% and ≈ 42.41%, respectively, of the maximum bandwidth of the ClusterStor[3]. The average write throughput in 2020 was 25.336 GiB/m and

[3]According to the technical specifications, the peak performance a ClusterStor system such as the one installed on SDumont should achieve, without considering cache effects, is 45 GiB/s. Since we aggregated the data by minute, we assume the peak is 2700 GiB/m (45 GiB/s×60 s).

in 2021 as 34.452 GiB/m (1.3× increase). The read operations presented lower values in 2020 but with a significant increase in 2021. The peak throughput in 2020 316 GiB/m (at 2020-03-24 20:36:00) and in 2021 1,077 GiB/m (at 2021-03-20 18:50:00), accounting for ≈ 11.7% and ≈ 39.89%, respectively, of the max bandwidth. The average read throughput in 2020 was 15.825 GiB/m and in 2021 was 66.953 GiB/m (4.2× increase). If we focus on the peak throughput registered at each OST, the write occurred as a single event on all OSTs simultaneously, which is the peak throughput for the whole file system. On the other hand, the readings presented the peaks on each OST spread out through period, with none of the OSTs' peaks coinciding. This behavior indicates that the applications write data in a more coordinated manner than they read.

Table 2 depicts the distribution of the Size and Quality of Operations across the observed data of the two years, showing that in 2020 the writes use the Lustre file system more efficiently, with 75% being below 2 QO, and by using bigger request sizes. On the other hand, reads present a somewhat inefficient use, with most of its operations being above 2 QO and using less than 512 KiB for its request sizes. In 2021, the reads still were inefficient and presented smaller transfer sizes when compared to writes. Nonetheless, it is possible to notice a considerable improvement when comparing the read operations from 2020 against the 2021 values. As an example, the median read transfer size went from 23 KiB to 816 KiB. This increase in the read operation size is why there was a decrease in the number of reading operations but increased the amount of data transferred. The writing operations showed a slight decrease (lower quality and smaller transfer sizes). Another important observation in Table 2 is that in 2021, the performance gap between reading and writing operations narrowed. In 2020 the average write operation size was ≈ 3× bigger than the read size (1729 KiB and 651 KiB respectively), while in 2021 the write size was only ≈ 1.4× bigger than the read (1488 KiB and 1018 KiB). This difference in request sizes explains why we observed higher throughput on writing operations.

Table 2: Size (KiB) and Quality of Operations.

Year	Operation	Metric	Min.	1st Q.	Median	3rd Q.	Max.
2020	**Read**	$Size$	4.00	6.60	23.80	577.00	4096.00
		QO	0.25	1.77	43.00	155.00	256.00
	Write	$Size$	0.01	530.00	1458.00	2947.00	4096.00
		QO	0.25	0.35	0.70	1.93	525131.00
2021	**Read**	$Size$	4.00	271.00	816.00	1786.00	4096.00
		QO	0.25	0.57	1.25	3.78	256.00
	Write	$Size$	0.01	420.00	970.00	2149.00	4096.00
		QO	0.25	0.48	1.10	2.44	104865.00

As for the workload characterization, we analyzed how much each type of operation demands from the system. Overall, in 2020 the write workload dominated the throughput, representing 61.75% of all data transferred. Nevertheless,

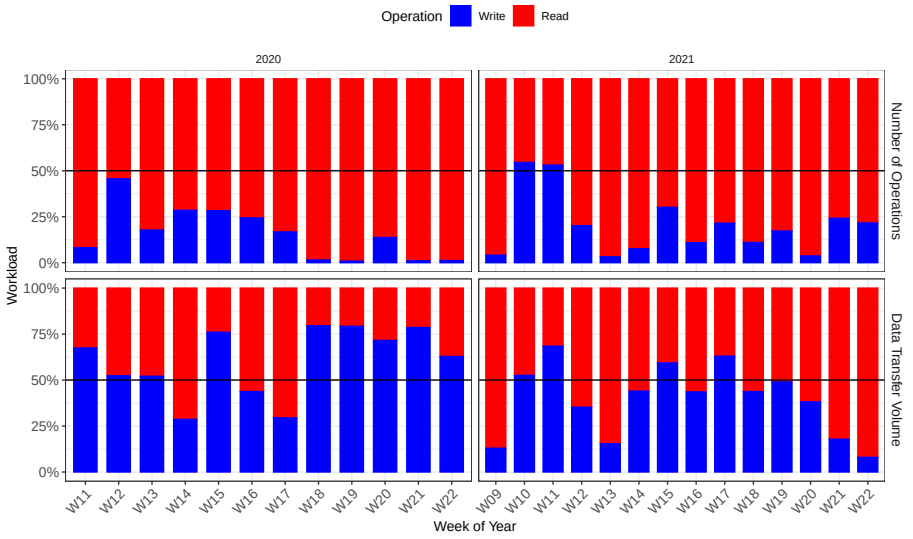

Figure 4: Workload distribution by week from 2020 (left) and 2021 (right)

considering the number of operations, the workload is primarily dominated by read operations, representing 98.11% of the total amount. There were $\approx 52\times$ more read operations than writes. The data from 2020 presents a scenario where Lustre received an enormous amount of inefficient reads. In 2021 the read operations dominated both workloads, accounting for 66% of the throughput and 88% for the number of operations.

The left facet at Figure 4 depicts the workload distribution between reads and writes grouped by week, starting on March 10th, 2020, and split by the number of operations (top) and the volume of data transferred (bottom). Checkpoints do not always dominate the I/O workload of HPC systems. Scientists are reading large volumes of data into HPC systems as part of their science [37, 38]. As new applications from Big Data and Machine Learning enter the HPC system, the ratio between read and write-bound workloads varies based on the application set running on the platform. Overall, during the three-month observation period, nine of the twelve weeks of data transferred are dominated by writes. Nonetheless, this trend is shifted towards read-intensive applications for three weeks (W14, W16, and W17). Regarding the number of operations, the workload was dominated by readings across all weeks, being fairly balanced on W12. Figure 4 shows how the demand for I/O operations changes across the period and how varied the workload is – while more read operations are occurring, more data are being written (especially in the last five weeks).

Unlike 2020, the right facet in Figure 4 denotes that in 2021 the volume of data transferred was dominated by reads, with only four out of fourteen weeks being dominated by writes. The number of read operations continues to dominate most weeks, with the exception of weeks W10 and W11, which were dominated

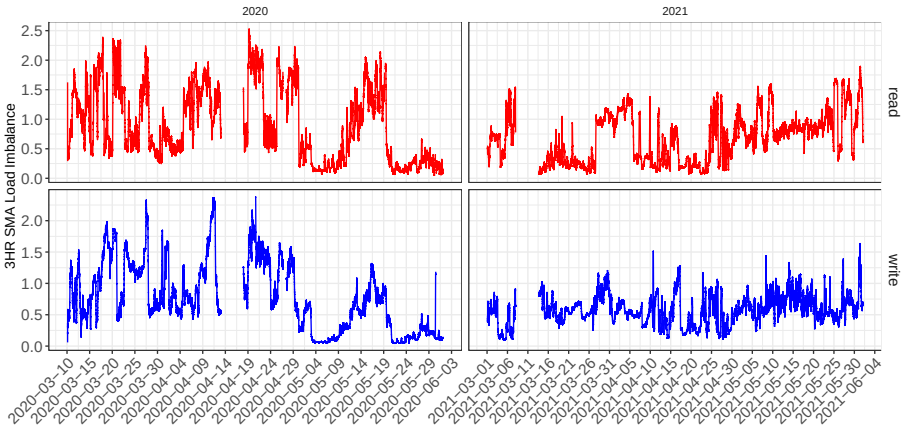

Figure 5: SMA_{3HR} of LI for the read (red) and write (blue) workload from 2020 (left) and 2021 (right). Values < 0.5 can be considered as low imbalance, values around 1 are moderate, and those above represent severe imbalance. Missing values refer to maintenance periods.

by writing both in number of operations and in volume of data transferred. The behavior of a greater number of read operations of small size still occur and can be observed specially on weeks W15 and W17, where the readings dominated the number of operations but the writings dominated the amount of data transferred. That prompts system administrators, I/O library developers, and I/O specialists to tackle automatic selective optimizations for read-intensive and write-intensive applications, taking into account the application's I/O demands and characteristics in future supercomputers' I/O subsystem.

The analysis of the OSTs' load distribution indicates that there's a considerable imbalance. Figure 5 (left facet) shows that for 2020, both read and write operations follow similar trends regarding the load imbalance. Despite ClusterStor presenting 0.6 LI for 50% of the time, there are some severe cases where the overload represented up to 300% of the OSTs' average load. An interesting imbalance case happened on April 10th, where it is possible to observe a spike in the writing load. Not only is the imbalance severe, but it also lasted for more than one day on certain occasions. The average for reading was 0.92 while for writing was 0.80. The level of imbalance between read and write was similar, with at least 25% of the time operating with LI above 1.

Lustre's default load balancing is focused on OST's available storage space, where the MDS uses a round-robin approach to designate where to store the data objects. This approach does not consider the current I/O load on the OSSes and allied with the default stripping policy on SDumont (`stripe_count` $= 1$ and `stripe_count` $= 1MiB$), potentially leads to hot-spots and resource contention that degrades performance. Hence, updating the default striping policy in the machine could alleviate such imbalance by better distributing the load among the selected OSTs. Another more invasive approach that system administrators

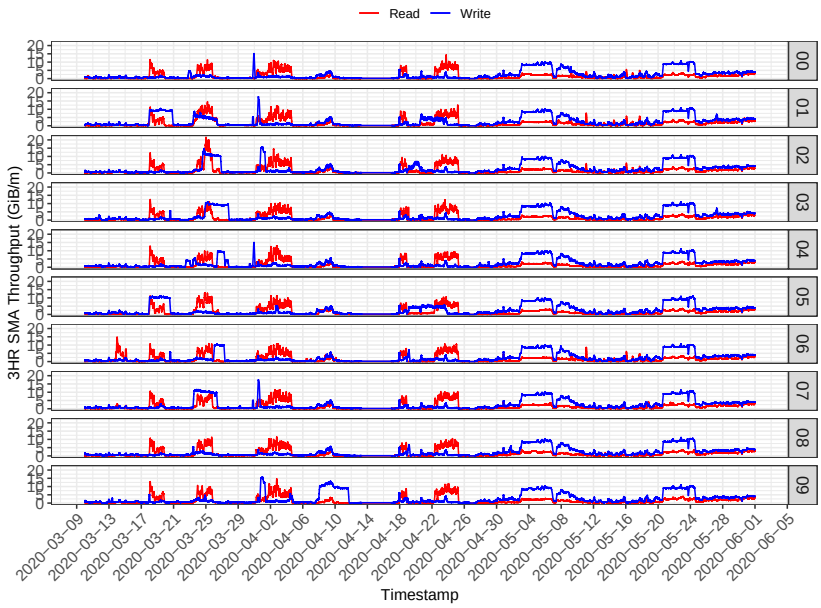

Figure 6: 2020 SMA_{3HR} of read and write throughput by OST.

might take is to adopt a dynamic load balancer that automatically coordinates the workload and data placement among I/O servers [9, 18].

Figure 6 depicts the SMA_{3HR} (Equation 2) of read and write throughput from 2020, broken down by OST. The spike in the imbalance on April 10th is due to OST 09 receiving a more significant write workload than the others. With the SMA, we can observe the general trend, identifying where usage peaks occur and periods when a specific OST receives more load than the others.

Figure 5 (right facet) depicts that the 2021 utilization data presented a decrease in the imbalance, with the average LI for reading being 0.68 while for writing, it was 0.58. The reading was more unbalanced, with at least 25% of the time operating with LI above 1. The increase in LI of the reads between March 28th and April 5th (also observed as an increase of read on Figure 3) is due to a load concentration on three OSTs (01, 03, and 08)[4].

5.1.2. Metadata Analysis

This subsection presents the analysis of the metadata operations performed on the Lustre PFS for the 2021 dataset. We were unable to collect some metrics during 2020 due to a bug on the *collectl-plugin* MDS module that prevents the collection of correct operation counters information exposed by the newer versions of the Lustre MDS kernel module (2.10 and later).

Table 3 summarizes the executions of each metadata operation during the

[4]Figure available at the Companion Repository [11].

Table 3: Overview of metadata operations.

Operation	Total	Min ops/s	Avg. ops/s	Max. ops/s
`fopen`	28,812,381,450	1	3,859	102,291
`fclose`	25,369,943,340	1	3,398	102,132
`getattr`	6,733,374,960	1	902	32,698
`setattr`	3,451,979,850	1	462	8,406
`unlink`	593,117,055	1	87	2,357
`getxattr`	345,187,575	1	47	7,833
`statfs`	280,998,450	1	38	62
`sync`	125,075,625	1	76	1,618
`mkdir`	94,034,205	1	14	1,228
`rmdir`	41,638,320	1	34	1,041
`setxattr`	4,354,485	1	83	1,061
`link`	1,649,205	1	139	2,357

whole period. The most demanding operations are `fopen`, `fclose`, `getattr`, and `setattr`. Considering all the metadata operations executed at the same timestamp, the mean ops/s was 8,920, and the maximum registered was 205,016 ops/s. It seems that there is a tendency by the users to treat the Lustre PFS as a regular file system, e.g., `HOMEDIR`, where they usually execute plenty of commands that return file and directory attributes (size and access mode). This behavior can generate an unnecessary overload on the MDS because this operation is costly (the MDS must query each OST to get information about the object's size to get the total file size). The `setattr` operation is associated with file creation, as we configure the default access mode. On the other hand, the manual change of file attributes by the users (through `chmod` or `chown` commands) is uncommon. Another behavior that stood out is the "low" number of file removals (`unlink`). This indicates that many old unused files occupy valuable space on Lustre PFS.

Figure 7A depicts the load distribution between I/O and metadata operations. It is possible to notice that the metadata operations dominate most weeks, except W12 and W13. This figure denotes how metadata-intensive the utilization of the Lustre PFS on SDumont is, with weeks where the I/O operations reach only 5% of the total load. The metadata was responsible for 60% of all file system operations, with $\approx$ 67 Billion requests against $\approx$ 44 Billion I/O requests. In Figure 7B, it is possible to observe an approximate constant trend of `fopen` and `fclose` operations across the whole period. On W10, however, there is an increase in the `getattr` and `setattr` operations, also observed on W17, W18, and W19. The system has a high demand for opening and closing files and retrieving a file or directory attributes. Most of the time, there are more operations to open and close files than to read and write them, indicating that the system handles many small files and the applications perform small throughput I/O (as shown in Section 5.1.1). The metadata workload characterized on SDumont would not be desirable on a scratch Lustre PFS designated for high throughput.

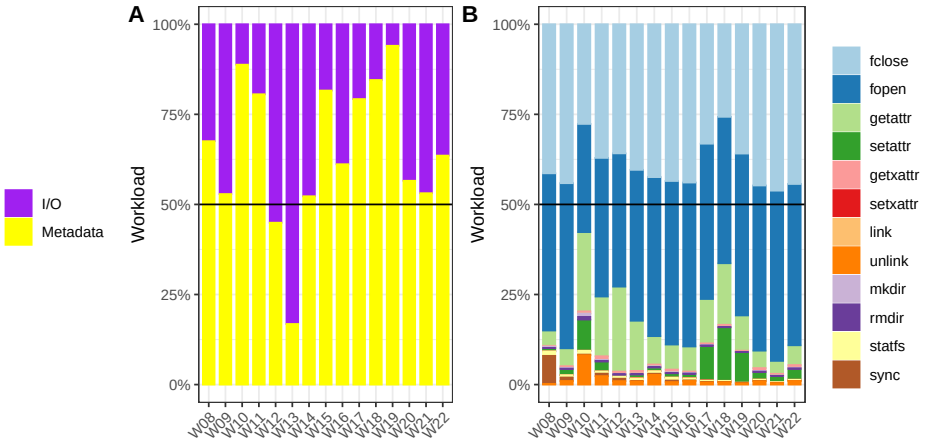

Figure 7: 2021 metadata load distribution by week. (A) depicts the load of I/O operations (purple) and metadata operations (yellow). (B) details the metadata operation type.

Metadata operations are becoming an increasingly hot development topic in high-performance storage and a genuine concern. It is possible to observe this trend on the latest releases of the IO500 [39], where the most significant growth in the max score is attributed to metadata while the I/O throughput presents little to no increase over the years. Among the improvements that boost metadata performance are the use of multiple servers [40], indexing mechanisms [41], and client-side pre-allocation [42]. There are features [43] specific to improve Lustre's metadata performance which system administrators should asses, such as Distributed Namespace (DNE – use of multiple metadata servers to distribute the load) and Data on Metadata (DoM – store small files on the MDT, significantly reducing the number of requests and access to OSTs). None of those improvements are implemented on SDumont.

5.2. Detailed View of a Region of Interest

Section 5.1 presented an overall view of the Lustre PFS usage on SDumont, which indicated that read operations seem to make inefficient use of the PFS, and it has a high demand for metadata operations. To make a more in-depth analysis, we selected two periods, one from each year, to analyze with the *Job Usage* dataset as we wanted to verify which applications contributed to specific events. From 2020 we choose the period between March 24th and March 28th because it comprises the peak throughput for the read operations (Figure 3 left), and from 2021, between March 28th and April 1st because of the striking increase in read volume (Figure 3 right). Crossing the information from the compute node dataset and SLURM's job list is very resource-demanding, depending on the date range. Nevertheless, this process can be executed on-demand on any period to understand better how the Lustre FS behaved during a given event.

5.2.1. Applications I/O Data Analysis

In this subsection, we used the information collected at the compute nodes to characterize the applications' I/O utilization. During the selected period of 2020, a total of 866 jobs have been executed on SDumont. With the SLURM's information, we were able to identify nine different applications: *AMBER* [44], *BIE* [45], *CASINO* [46], *DockThor* [47], *GROMACS* [48], *LAMMPS* [49], *NAMD* [50], *QUANTUM ESPRESSO* [51], and *SIESTA* [52]. We could not identify some job's applications solely with the executable name provided by SLURM, and therefore cataloged under three groups as *Bash Scripts* (executable name informed is `bash` and the user's job is executed directly through a script), *OpenMPI mpiexec* (executable name informed is `orted` and the application is started directly through `mpiexec/mpirun`, without using SLURM's `srun` command), and *unknown* (when the user compiles the application and the executable name informed is not present on the database of known applications).

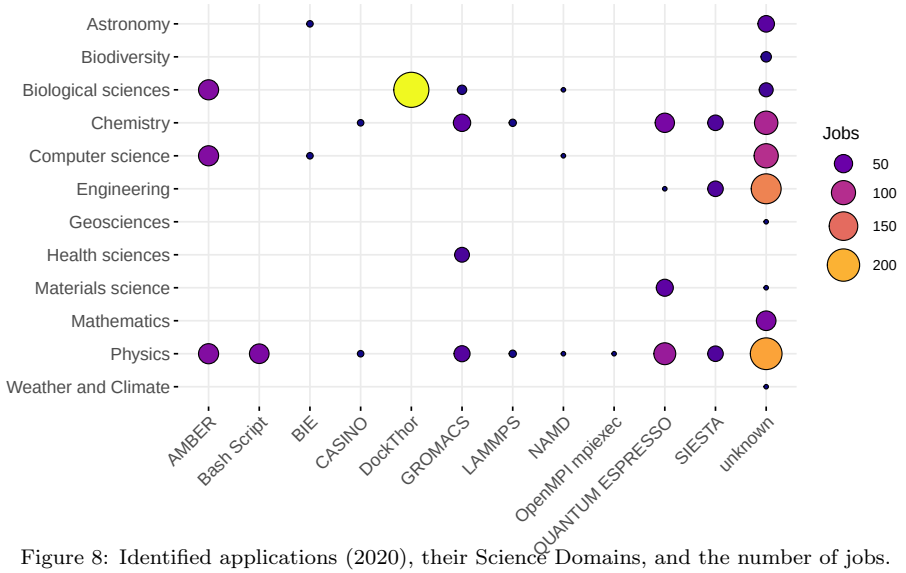

Figure 8: Identified applications (2020), their Science Domains, and the number of jobs.

Figure 8 details the applications and which "Science Domain" used them. It is interesting to notice that an application can be employed by different areas, such as *GROMACS*, used by projects in *Chemistry*, *Physics*, *Biological Sciences*, and *Health Sciences*. This fact contributes to an application having different I/O volumes, demands, and behaviors [29]. The Science Domain that used the largest application set was Physics, with nine different applications, followed by Chemistry, with five. The five most executed applications were *unknown* (37.41%), *DockThor* (27.83%), *QUANTUM ESPRESSO* (9.93%), *AMBER* (7.39%), and *Bash Script* (6.93%). For the 2021 dataset, we observed a total of 845 submitted jobs and recorded eleven different applications (plus the three groups): *AMBER*, *BIE*, *DockThor*, *GROMACS*, *LAMMPS*, *LHCB DIRAC* [53], *ORCA* [54], *Python* [55], *QUANTUM ESPRESSO*, *SIESTA*, and *VASP* [56]. The Science

Domain that used the largest application set was Physics and Chemistry, which used seven different applications each. The five most executed applications on 2021 were: *DockThor* (36.21%), *unknown* (17.75%), *QUANTUM ESPRESSO* (10.06%), *LHCB DIRAC* (8.88%), and *AMBER* (7.57%).

There are different types of optimizations that can be employed on the storage system to improve performance. I/O optimizations can be applied to applications at each layer of the stack [57], and automatic dynamic scheduler [58, 59] can schedule applications with same I/O patterns to alleviate congestion. One thing in common that these optimizations techniques need is the I/O access patterns characteristics of each application. The same application can have different I/O behavior [29] depending upon which science domain used it. In that way, having the information about the science domain that used the application together with the access pattern might help I/O libraries, schedulers and other techniques to obtain a better picture and comprehension when applying the optimizations.

Aside from the maximum throughput achieved for reads in 2020 (presented in Section 5.1), the peak for writes on this specific period was 370 GiB/m (at 2020-03-27). Table 4 describes the peak achieved by an individual application for reading and writing. It is important to notice that: (I) only one job (application) was responsible for $\approx$ 91.87% read throughput and $\approx$ 95.43% for the write throughput in the 2020 period, (II) both with high CF_{bw}, (III) there was a decrease in the throughput of individual application from 2020 to 2021 but with an increase in the bandwidth utilization by a greater number of applications (decrease in the CF_{bw} value). *QUANTUM ESPRESSO* can be categorized as a traditional HPC Scientific Application, highly parallelized and optimized [60].

Figure 9 depicts how the bandwidth of the whole Lustre file system was used, where the maximum (red), minimum (blue), and average (black) CF_{bw} of the jobs, at each timestamp, throughout the specific 2020 period is plotted. This behavior indicates that a few applications with high I/O throughput consume the bandwidth since the average value is closer to the minimum.

Table 4: Individual application's throughput.

Year	**Application**	**Operation**	**GiB/m**	CF_{bw}
2020	QUANTUM ESPRESSO	Read	290	0.84
	QUANTUM ESPRESSO	Write	353	0.94
2021	*unknown*	Read	153	0.70
	QUANTUM ESPRESSO	Write	90	0.31

As depicted by Figure 10, in 2021 a few jobs also dominated the bandwidth. We observed high-bandwidth jobs starting at May 31st, with the decrease in the Max. CF_{bw} and a slight increase in the average values.

Figure 11 presents the overall distribution for the 2020 period of the *Transfer Size* and *QO* metrics. We analyzed the distribution of each application's Quality of Operations and the majority of them presented inefficient reads, with $read_{qo}$ above 1 for most of the time. In 2020, the most notably read-inefficient

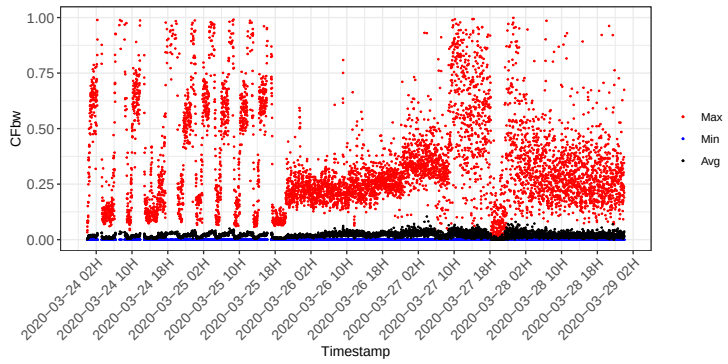

Figure 9: 2020 CF_{bw} of the jobs. The dots in red, black, and blue represent the max., avg., and min., respectively, of all jobs, observed on each timestamp.

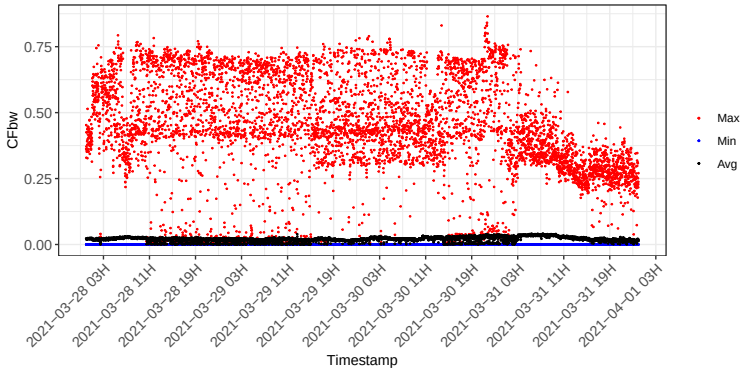

Figure 10: CF_{bw} (min, max, and average) of the jobs in 2021.

applications were *DockThor*, *BIE*, and *Bash Script*. Only the *OpenMPI mpiexec* category presented $read_{qo}$ under 1 for 75% of its execution, meaning that it read blocks of data closer to the Lustre's `stripe_size`. Regarding each Science Domain, *Biodiversity* and *Materials Science* presented the best behavior, using larger sizes for the operations with a good *QO* index.

As for the transfer sizes, most applications seldom use sizes larger than 1 MiB. Most issue 100 KiB or smaller at least 75% of the time. We did not observe transfer sizes above 4 MiB, which is due to Lustre's default maximum bulk I/O RPC size [61] from a client to the OST, even though applications might be issuing bigger operations. Requests bigger than 4 MiB need to be broken down by two or more RPCs. This parameter can be tuned up to 16 MiB on current Lustre version (2.1X), allowing fewer RPCs to transfer the same amount of data between clients and servers. Applications that used bigger size operations were *SIESTA*, *QUANTUM ESPRESSO*, *CASINO*, and *AMBER*, with at least 50% of the time above 1 MiB. The application category that presented the largest writing sizes was the *OpenMPI mpiexec*, using above 2 MiB during 75% of its

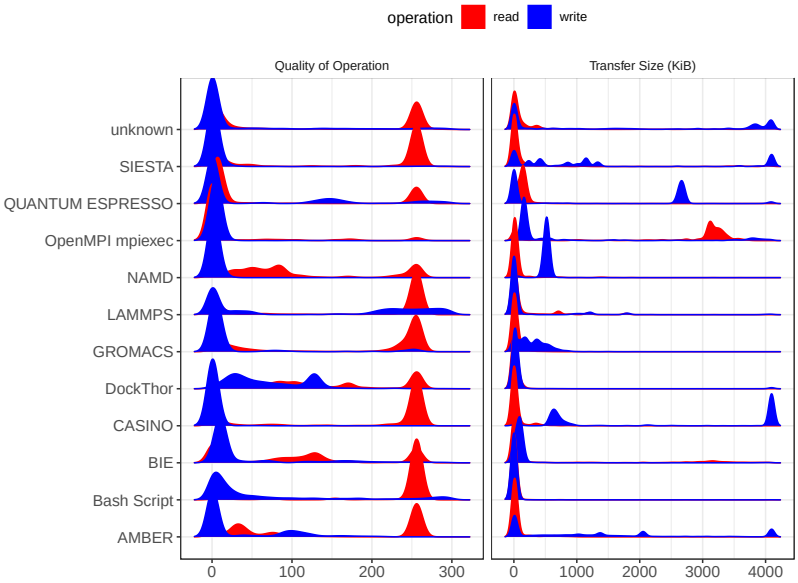

Figure 11: 2020 Distribution of the Quality of Operation (left) and Transfer Size (right).

execution. The applications with the most notably smallest sizes were *LAMMPS*, *DockThor*, and *Bash Script*, with 75% of their operations under 40 KiB.

The distribution of the quality of operation and transfer size of applications observed during the 2021 window presented a similar behavior as in 2020, and for that reason we are omitting the figure (but it is available at [11]). The most notably read-inefficient were *Bash Script* and *BIE*. The ones that presented the most efficient operations were *GROMACS*, *OpenMPI mpiexec*, and *Python*. Applications started through OpenMPI mpiexec presented the highest sizes for both reads (50% above 1 MiB) and writes (75% above 1 MiB). We were unable to identify those applications based only on the SLURM information, hence, there could be a broad mix of applications in this group, as occurs with the *unknown* group. The write transfer size was bigger than the read for most applications. Those that used the biggest write sizes were *OpenMPI mpiexec*, *ORCA*, and *SIESTA*, with 1.5 MiB or above for at least 50% of their execution time.

Most applications' workload in 2020 is dominated by write operations, both regarding the number of operations and volume of data transferred. Once again, only the *OpenMPI mpiexec* category presents read dominance (above 80% on the number of operations and data volume). Two applications presented inverse workload demands: (I) *CASINO*, with the number of operations dominated by reads and the data transferred dominated by writes (indicating its writing operations are bigger than readings) ; and (II) *BIE*, with writes dominating the number of operations, and reads the data transferred (indicating its reading operations are bigger than writings). Regarding the workload of each Science Domain, five presented dominance for the number of readings, with the most

notable being *Biodiversity* (90%) and *Health Sciences* (76%). Considering the amount of data transferred, only *Biodiversity* had its workload dominated by reading (55%). The most write-demanding Domains are *Geosciences*, *Biological Sciences* and *Climate and Weather* in both the number of operations and volume of data. Two interesting cases are *Astronomy* and *Health Sciences*, where reads dominate the number of operations, but the volume of data transferred is dominated by writes, indicating that the workflow of these two Science Domains utilize many small read operations and few larger write operations.

The 2021 period presented an increase in the number of read-intensive applications, where from the fourteen identified applications, four are dominated by reads (*BIE*, *OpenMPI mpiexec*, *Python*, and *unknown*) on both number of operations and volume of data transferred. *ORCA* presented a read-intensive workload regarding the number of operations but wrote more data than read, with a significant difference in the transfer sizes for each operation – while 75% of reading operations has transfer size below 45 KiB, writings presented transfer sizes above 1 MiB for 75% of the time. *Bash Script* presented a read-intensive data transfer but issued more write operations, indicating many small writes (75% of the time below 2 KiB). The other applications are write-intensive.

Looking at the five most data-intensive applications from 2021, the *unknown* group (the second most executed) have the highest average amount of data read per job (5,394 GiB – but writes only 23 GiB per job), being responsible for the expressive increase in the reading activities observed in Figure 3b. *BIE* come as the 2nd application, which on average read 793 GiB and write 60 GiB data per job. The succeeding applications were *OpenMPI mpiexec* (95 GiB and 42 GiB), *AMBER* (3 GiB and 44 GiB, and *QUANTUM ESPRESSO* (2 GiB and 22 GiB). Interestingly, none presented an even write/read workload.

Finally, we evaluate the level of I/O parallelism performed by each application using the data collected at Compute Node at each timestamp. We analyze it in two facets: (1) how many OSTs were used and (2) how many Compute Nodes performed I/O operations. In 2020, the average of simultaneous OST used by each application is ≈ 3.24, which is below half the available OSTs (10 in total). During 75% of the time, only up to four OSTs are used simultaneously by the applications. The average simultaneous Compute Nodes used during I/O activities was ≈ 1.64 and 75% of the cases used only up to two nodes, which is considerably lower than the OSTs used. This difference in proportions indicates that most of the time, the applications use few Compute Nodes to perform I/O on a greater number of OSTs. In fact, we observed cases where the application used only one Compute Node to write on all ten OSTs.

The level of parallelism in 2021 follows a similar trend as the one observed in 2020, with the average of simultaneous OST used being 3.6 while the average for computational nodes used for I/O operations was 1.65. The analysis of the utilization divided by the type of operation reveals that, on average, the jobs use ≈ 4 OSTs for concurrent reads. Writing operations showed slightly lower values, with the average of 3.3 simultaneous OSTs. In general, applications use fewer simultaneous nodes for the I/O operations than available OSTs, with only five applications using more than five compute nodes at the same time.

Through the analysis of how the applications used the Lustre PFS on SDumont, we were able to identify that most of them perform small I/O requests, especially for the reads, resulting in inefficient utilization. The high rate of small requests signals that most applications do not use specialized I/O libraries (e.g., HDF5 [62] and MPI-IO [63]) that have optimizations capable of aggregating and reorganizing data access patterns to read or write large contiguous chunks that potentially match the layout in the storage servers. As the *Coverage Factor of the Bandwidth* (Figures 9 and 10) shows, few applications issuing high throughput operations use most of the aggregate bandwidth provided by the storage system, leaving plenty of available resources. On the other hand, there is considerable demand for low latency performance to respond to the many small transfer size operations. We believe that an I/O Forwarding [64] or Burst Buffer [65] layer would significantly benefit SDumont in handling these low-latency small requests. The System Administrators can opt to direct efforts to improve the I/O performance of the most demanding applications or implement frameworks [57, 58, 59] that auto-tune and optimize I/O for a more extensive set of applications.

5.2.2. Applications Metadata Analysis

In this subsection, we used the information collected at the compute nodes to characterize the applications' metadata utilization for the periods defined in Section 5.2. Figure 12A illustrates the load ratio between I/O and metadata operations recorded by each application's jobs executed in the 2020 window. From the twelve identified applications, only *CASINO* presents a metadata-intensive behavior (more metadata than I/O). *AMBER* also makes heavy usage of metadata. The one with the lowest metadata usage are *GROMACS*, *LAMMPS*, and the *OpenMPI mpiexec* category. It is possible to notice that the usual load of metadata operations is between 10% and 25%.

Figure 12B depicts the types of metadata operations used by the applications. It is possible to observe that the seek operation dominates most of the applications (*AMBER*, *Bash Scripts*, *CASINO*, *NAMD*, *QUANTUM ESPRESSO*, *SIESTA*, and *unknown*), indicating a great number of random file access. Two applications (*BIE* and *OpenMPI mpiexec*) present a high load of `fopen` and `fclose` operations, indicating the use of many files during its workflow. Three applications (*DockThor*, *GROMACS*, and *LAMMPS*) presented a high load of the `getattr`, operations that retrieve file information from the MDS and should be avoided as much as possible due to increase in the overload of the MDS. The `setattr` operation is hardly used by the applications. These observations highlight the heterogeneous I/O workloads that a shared PFS has to handle efficiently. However, the plethora of available tunable parameters makes it hard for end-users to understand when and how they should tune it to avoid bottlenecks. Furthermore, due to its shared nature, poorly tuned applications could harm performance to themselves and other concurrent jobs. Future HPC storage systems should automatically isolate those or provide working mechanisms to tune the FS based on the observed I/O workloads.

Regarding the Science Domain, only Materials Science and Astronomy are metadata-intensive, with their load reaching 55% and 49%, respectively. Health

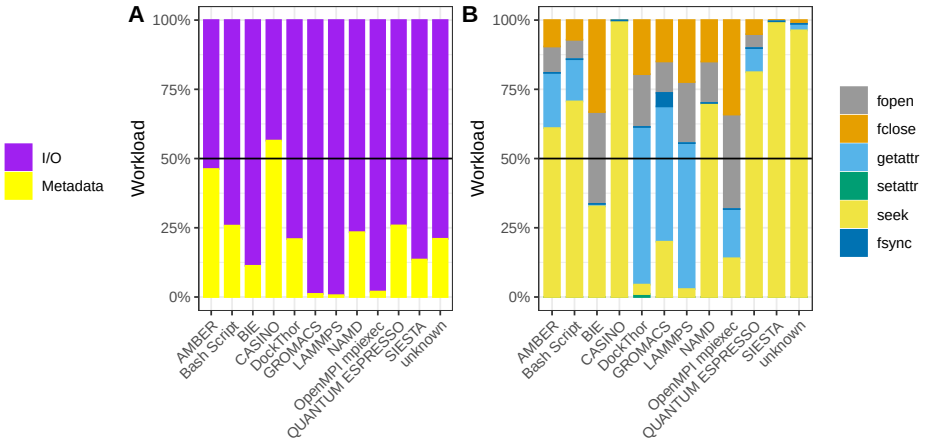

Figure 12: 2020 applications' metadata load distribution. (A) presents the load division between I/O (purple) and metadata operations (yellow). (B) presents the division among each metadata operation type. The x-axis is the application's name and y-axis is the load (%)

Science makes minor use of metadata operations, presenting the lowest workload. The other domains present a metadata workload between 10% and 25%.

We omit the metadata load distribution plot from the 2021 data because most applications identified on the 2021 period appear on 2020 and the characterization was similar (the plot is available at the companion repository [11]). For the applications identified on the 2021 period and not present in Figure 12, only *Python* and *LHCB DIRAC* present a metadata-intensive behavior, with a load of respectively 95% and 65%. *ORCA* and *VASP* are I/O intensive, with their metadata load reaching only 5%. Regarding the metadata operations most used by each application, *LHCB DIRAC* is dominated by `seek` and `getattr`, *ORCA* presents a high load of file open and close, *Python* is dominated by `getattr` executions, and *VASP* presents a high load of `seek`.

6. Discussion

In this section, we discuss the lessons learned and compare how the Lustre PFS deployment on SDumont against the storage system of other HPC platforms.

I/O Workload. The I/O workload on HPC systems is not dominated by a single type of operation, as some studies imply. While Kim et al. [66] presented only a marginal difference between the read (42.2%) and write (57.8%) workload, the system analyzed by Gunasekaran et al. [67] shows a write dominance by 75%. In contrast, Patel et al. [8], observed that the read constantly dominates the workload. The data transfer volume observed on SDumont presented a certain seasonality (Figure 4). On the 2020 period the read operations dominated part of the first half, with writings dominating the last period. But in terms of the number of requests, the workload is massively dominated by reads. In general terms, the total data written was $\approx 1.6\times$ the data read, but the system received

$\approx 52\times$ more read requests than writes. This indicates that more data is written using large request sizes. Our finding helps designing future storage system acquisition for HPC systems that have this type of heterogeneous environment.

Operation Size. The system studied by Kim et al. [66] presents most request sizes that are either less than 16 KiB or between 512 KiB and 1 MiB. Gunasekaran et al. [67] concluded that 60% of write requests are 4 KiB or less, and over 50% of reads were at least 1 MiB. The request sizes on SDumont presents an inverted proportion, with reads being small (50% below 23 KiB) while writes are larger (50% reaching ≈ 1.5 MiB). The write operation size on SDumont was $\approx 3\times$ larger than the read requests. This difference in operation size confirms that there are far more small reads than writes. On the quality of operations, the reads are notably worse than writes. This information helps optimize block devices since the PFS throughput is highly dependant on the request size.

Bandwidth Usage. SDumont presented low bandwidth usage if compared with other systems. Kim et al. [66] observed that the peak read throughput reached 75% of the maximum bandwidth, while the writes reached 54%. The system analyzed by Gunasekaran et al. [67] presented up to 80% of the bandwidth for reads and 70% for writes. The peak of writes on SDumont reached 41.74% of the maximum bandwidth, while the reads reached only 11.7%. Aside from being lower, we see an inverted behavior. This low performance of the read operations is associated with the fact that the their sizes is far smaller than the writes.

Load Imbalance. This is a problem that still has not been fully addressed in large-scale platforms. As shown with our results (Figure 5), the load imbalanced on SDumont can be as severe as a single OST receiving 100% of the load. Previous works ([66, 8]) reported similar problems. High load imbalance results in low throughput and under-utilization. System administrators might consider adopting a dynamic load balancer that automatically coordinates the workload and data placement among I/O servers to ease this problem.

Low I/O Parallelism. HPC applications still do not take full advantage of a *PFS* or of high-level I/O libraries and middleware that has the potential to improve I/O performance. On SDumont 75% of the observed jobs used only up to 4 of the 10 available OSTs. This is another indication of why the bandwidth usage is low. Application developers should consider high-level I/O libraries to leverage the data access parallelism and end-users should be instructed on how to tune the file system's stripping parameters for efficient usage.

7. Conclusion

This paper evaluated the storage of SDumont supercomputing which houses research from different Science Domains. We collected metrics and analyzed the machine's I/O behavior for three months from two years. Our proposed methodology provides insights to understand Lustre's usage. We were able to identify critical aspects that negatively impact the performance on SDumont:

- *Inefficient read operations*: There's a high count of operations using small transfer sizes, which often translate into poor performance. Furthermore, it shows that applications are not taking full advantage of high-level I/O libraries and middleware to aggregate small requests into larger ones.

- *Imbalance among resources*: With the individual OST usage view was possible to assess how the load is distributed, denoting some severe and lasting cases where the overload corresponds to 3× the average OSTs' load.

- *Problematic applications*: We were able to identify problematic behaviors, such as *BIE*, which exhibits the worst $read_{qo}$ and have the data transfer workload dominated by reading operations.

- *Demand for metadata operations*: The metadata analysis shows a considerable demand for this type of operation on SDumont, with it accounting for 60% of all file system operations. This scenario raises a warning for the system administrators to quickly take some action about it.

Identifying these aspects guides the administration of SDumont to focus the efforts to help improve the system's performance and usability. They can evaluate the adoption of an I/O forwarding layer or other transparent middleware solution to aggregate the small-sized operations and relieve the high read demand; revise the default *striping* policy; implement an automatic load balancer to smooth storage server imbalance; assess Lustre's internal metadata improvements; evaluate the implementation of a framework to auto-tune the I/O stack; and contact the demanding projects to address special requirements.

As future work, we plan to improve the application identification, especially in those large detected groups. This better identification would also help provide additional information for frameworks oriented for I/O optimizations guided for specific applications. We also plan to improve the scalability and performance of our analysis tools. All data, source codes, and analysis conducted in this paper are available at the Companion Repository ([11]).

Acknowledgement

This study was financed by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and from the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil. 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 research has used resources from the U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231.

References

[1] SUN, High-Performance Storage Architecture and Scalable Cluster File System, Tech. rep., Sun Microsystems (2007).

[2] IO500 - SC21 - Full List (2021).
URL https://io500.org/list/sc21/full

[3] O. Yildiz, M. Dorier, S. Ibrahim, R. Ross, G. Antoniu, On the root causes of cross-application i/o interference in hpc storage systems, in: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2016, pp. 750–759. doi:10.1109/IPDPS.2016.50.

[4] J. Yu, G. Liu, X. Li, W. Dong, Q. Li, Cross-layer coordination in the i/o software stack of extreme-scale systems, Concurrency and Computation: Practice and Experience 30 (10) (2018) e4396, e4396 cpe.4396. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4396, doi:https://doi.org/10.1002/cpe.4396.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4396

[5] S. R. Alam, H. N. El-Harake, K. Howard, N. Stringfellow, F. Verzelloni, Parallel i/o and the metadata wall, in: Proceedings of the Sixth Workshop on Parallel Data Storage, PDSW '11, Association for Computing Machinery, New York, NY, USA, 2011, p. 13–18. doi:10.1145/2159352.2159356.
URL https://doi.org/10.1145/2159352.2159356

[6] B. Lawrence, C. Maynard, A. Turner, X. Guo, D. Sloan-Murphy, J. R. Herrera, Parallel I/O Performance Benchmarking and Investigation on Multiple HPC Architectures (Archer White Paper), Tech. rep., Edinburgh Parallel Computing Centre (EPCC) (2017).
URL https://www.archer.ac.uk/documentation/white-papers/parallelIO-benchmarking/ARCHER-Parallel-IO-1.4.pdf

[7] C. Bartz, K. Chasapis, M. Kuhn, P. Nerge, T. Ludwig, A best practice analysis of hdf5 and netcdf-4 using lustre, in: ISC High Performance 2015, 2015, pp. 274–281. doi:10.1007/978-3-319-20119-1_20.
URL https://doi.org/10.1007/978-3-319-20119-1_20

[8] T. Patel, S. Byna, G. K. Lockwood, D. Tiwari, Revisiting i/o behavior in large-scale storage systems: The expected and the unexpected, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '19, Association for Computing Machinery, New York, NY, USA, 2019. doi:10.1145/3295500.3356183.
URL https://doi.org/10.1145/3295500.3356183

[9] S. Neuwirth, F. Wang, S. Oral, U. Bruening, Automatic and transparent resource contention mitigation for improving large-scale parallel file system performance, in: 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS), 2017, pp. 604–613. doi:10.1109/ICPADS.2017.00084.

[10] SDumont - Brazilian Petaflop Computing System.
URL https://sdumont.lncc.br/

[11] A. R. Carneiro, J. L. Bez, C. Osthoff, L. M. Schnorr, P. O. A. Navaux, Companion repository for the JPDC Special Issue for SBAC-PAD 2021 paper (2022).
URL https://gitlab.com/andre.es/jpdc_sbac-pad-2021

[12] H. Luu, M. Winslett, W. Gropp, R. Ross, P. Carns, K. Harms, M. Prabhat, S. Byna, Y. Yao, A multiplatform study of i/o behavior on petascale supercomputers, in: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, HPDC '15, Association for Computing Machinery, New York, NY, USA, 2015, p. 33–44. doi:10.1145/2749246.2749269.
URL https://doi.org/10.1145/2749246.2749269

[13] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, R. Ross, Understanding and improving computational science storage access through continuous characterization, ACM Trans. Storage 7 (3). doi:10.1145/2027066.2027068.
URL https://doi.org/10.1145/2027066.2027068

[14] G. K. Lockwood, S. Snyder, T. Wang, S. Byna, P. Carns, N. J. Wright, A year in the life of a parallel file system, in: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, 2018, pp. 931–943. doi:10.1109/SC.2018.00077.

[15] L. Wan, M. Wolf, F. Wang, J. Y. Choi, G. Ostrouchov, J. Chen, N. Podhorszki, J. Logan, K. Mehta, S. Klasky, D. Pugmire, I/o performance characterization and prediction through machine learning on hpc systems, in: Proc. of the Cray User Group conference, 2020.

[16] Lawrence Livermore National Laboratory, lmt: Lustre Monitoring Tools (2019).
URL https://github.com/LLNL/lmt

[17] K. Sivalingam, H. Richardson, A. Tate, M. Lafferty, Lassi: Metric based i/o analytics for hpc, in: 2019 Spring Simulation Conference (SpringSim), 2019, pp. 1–12. doi:10.23919/SpringSim.2019.8732903.

[18] B. Wadhwa, A. K. Paul, S. Neuwirth, F. Wang, S. Oral, A. R. Butt, J. Bernard, K. W. Cameron, iez: Resource contention aware load balancing for large-scale parallel file systems, in: 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2019, pp. 610–620. doi:10.1109/IPDPS.2019.00070.

[19] J. M. Kunkel, E. Betke, Tracking User-Perceived I/O Slowdown via Probing, in: International Conference on High Performance Computing, Springer International Publishing, 2019, pp. 169–182. doi:10.1007/978-3-030-34356-9_15.

[20] HPC IO Repository, IOR (2020).
URL https://github.com/hpc/ior

[21] E. Betke, J. M. Kunkel, Footprinting Parallel I/O – Machine Learning to Classify Application's I/O Behavior, in: International Conference on High Performance Computing, Springer International Publishing, 2019, pp. 214–226. doi:10.1007/978-3-030-34356-9_18.

[22] J. L. Bez, A. M. Karimi, A. K. Paul, B. Xie, S. Byna, P. Carns, S. Oral, F. Wang, J. Hanley, Access patterns and performance behaviors of multi-layer supercomputer i/o subsystems under production load, in: Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing, HPDC '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 43–55. doi:10.1145/3502181.3531461.
URL https://doi.org/10.1145/3502181.3531461

[23] K. Chasapis, M. F. Dolz, M. Kuhn, T. Ludwig, Evaluating lustre's metadata server on a multi-socket platform, in: 2014 9th Parallel Data Storage Workshop, 2014, pp. 13–18. doi:10.1109/PDSW.2014.5.

[24] D. Zhao, X. Yang, I. Sadooghi, G. Garzoglio, S. Timm, I. Raicu, High-performance storage support for scientific applications on the cloud, in: Proceedings of the 6th Workshop on Scientific Cloud Computing, ScienceCloud '15, Association for Computing Machinery, New York, NY, USA, 2015, p. 33–36. doi:10.1145/2755644.2755648.
URL https://doi.org/10.1145/2755644.2755648

[25] J. M. Kunkel, G. S. Markomanolis, Understanding metadata latency with mdworkbench, in: International Conference on High Performance Computing, Springer, 2018, pp. 75–88. doi:10.1007/978-3-030-02465-9_5.
URL https://doi.org/10.1007/978-3-030-02465-9_5

[26] National Laboratory for Scientific Computing - LNCC.
URL https://www.lncc.br/

[27] Brazil's National High-Performance Computing System.
URL https://www.lncc.br/sinapad/

[28] HPE, ClusterStor 9000 Data Sheet.
URL https://www.hpe.com/psnow/doc/c04663932?jumpid=in_lit-psnow-red

[29] J. L. Bez, A. R. Carneiro, P. J. Pavan, V. S. Girelli, F. Z. Boito, B. A. Fagundes, C. Osthoff, P. L. da Silva Dias, J.-F. Méhaut, P. O. Navaux, I/O performance of the santos dumont supercomputer, in: The International Journal of High Performance Computing Applications, SAGE, 2019, pp. 227–245. doi:10.1177/1094342019868526.
URL https://doi.org/10.1177/1094342019868526

[30] Collectl.
URL http://collectl.sourceforge.net/

[31] P. Piela, Collectl Lustre Data Collection Plugins.
URL https://github.com/pcpiela/collectl-lustre

[32] SQLite.
URL https://www.sqlite.org

[33] S. Hansun, A new approach of moving average method in time series analysis, in: 2013 Conference on New Media Studies (CoNMedia), 2013, pp. 1–4. doi:10.1109/CoNMedia.2013.6708545.

[34] Shiny.
URL https://shiny.rstudio.com

[35] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, W. Allcock, I/o performance challenges at leadership scale, in: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, 2009, pp. 1–12. doi:10.1145/1654059.1654100.

[36] M. Isakov, E. d. Rosario, S. Madireddy, P. Balaprakash, P. Carns, R. B. Ross, M. A. Kinsy, Hpc i/o throughput bottleneck analysis with explainable local models, in: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, 2020, pp. 1–13. doi:10.1109/SC41405.2020.00037.

[37] J. Lofstead, M. Polte, G. Gibson, S. Klasky, K. Schwan, R. Oldfield, M. Wolf, Q. Liu, Six Degrees of Scientific Data: Reading Patterns for Extreme Scale Science IO, in: Proceedings of the 20th International Symposium on High Performance Distributed Computing, HPDC '11, Association for Computing Machinery, New York, NY, USA, 2011, p. 49–60. doi:10.1145/1996130.1996139.
URL https://doi.org/10.1145/1996130.1996139

[38] R. Latham, R. Ross, Q. Koziol, A. Ching, HPC I/O for Computational Scientists (2014).
URL https://extremecomputingtraining.anl.gov//files/2014/01/hpc-io-all-final.pdf

[39] J. M. Kunkel, A. Dilger, D. Hildebrand, J. Lofstead, G. Markomanolis, The 9^{th} IO500 and the Virtual Institute of I/O Report (2021).
URL https://io500.org/files/sc21-io500-slides.pdf

[40] C. Rodríguez-Quintana, A. Díaz, J. Ortega, R. Hernández Palacios, A. Ortiz, A new scalable approach for distributed metadata in hpc, in: Algorithms and Architectures for Parallel Processing, Vol. 10048, 2016, pp. 106–117. doi:10.1007/978-3-319-49583-5_8.

[41] A. K. Paul, B. Wang, N. Rutman, C. Spitz, A. R. Butt, Efficient metadata indexing for hpc storage systems, in: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), 2020, pp. 162–171. doi:10.1109/CCGrid49817.2020.00-77.

[42] H. Li, P. Cheng, Z. Chen, N. Xiao, Pream: Enhancing hpc storage system performance with pre-allocated metadata management mechanism, in: 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2019, pp. 413–420. doi:10.1109/HPCC/SmartCity/DSS.2019.00069.

[43] J. Fragalla, B. Loewe, T. Kling Petersen, New lustre features to improve lustre metadata and small-file performance, Concurrency and Computation: Practice and Experience 32 (20) (2020) e5649.

[44] The Amber Molecular Dynamics Package.
URL https://ambermd.org/

[45] M. D. Weinberg, Computational statistics using the Bayesian Inference Engine, Monthly Notices of the Royal Astronomical Society 434 (2) (2013) 1736–1755. arXiv:https://academic.oup.com/mnras/article-pdf/434/2/1736/18499007/stt1132.pdf, doi:10.1093/mnras/stt1132.
URL https://doi.org/10.1093/mnras/stt1132

[46] CASINO Quantum Monte Carlo package.
URL https://vallico.net/casinoqmc/

[47] DockThor - Protein-ligand Docking Portal.
URL https://www.dockthor.lncc.br/

[48] GROMACS molecular dynamics package.
URL https://www.gromacs.org/

[49] LAMMPS Molecular Dynamics Simulator.
URL https://www.lammps.org/

[50] NAMD - Scalable Molecular Dynamics.
URL https://www.ks.uiuc.edu/Research/namd/

[51] Quantum ESPRESSO - integrated suite for electronic-structure calculations and materials modeling.
URL https://www.quantum-espresso.org/

[52] SIESTA.
URL https://departments.icmab.es/leem/siesta/

[53] LHCbDIRAC.
URL https://lhcb-dirac.readthedocs.io/en/latest/

[54] ORCA quantum-chemical software package.
URL https://orcaforum.kofo.mpg.de/index.php

[55] Python.
URL https://www.python.org/

[56] VASP - The Vienna Ab initio Simulation Package.
URL https://www.vasp.at/

[57] B. Behzad, S. Byna, Prabhat, M. Snir, Optimizing i/o performance of hpc applications with autotuning, ACM Trans. Parallel Comput. 5 (4). doi:10.1145/3309205.
URL https://doi.org/10.1145/3309205

[58] A. Gainaru, G. Aupy, A. Benoit, F. Cappello, Y. Robert, M. Snir, Scheduling the i/o of hpc applications under congestion, in: 2015 IEEE International Parallel and Distributed Processing Symposium, 2015, pp. 1013–1022. doi:10.1109/IPDPS.2015.116.

[59] M. Dorier, G. Antoniu, R. Ross, D. Kimpe, S. Ibrahim, Calciom: Mitigating i/o interference in hpc systems through cross-application coordination, in: 2014 IEEE 28th International Parallel and Distributed Processing Symposium, 2014, pp. 155–164. doi:10.1109/IPDPS.2014.27.

[60] P. Giannozzi, O. Baseggio, P. Bonfà, D. Brunato, R. Car, I. Carnimeo, C. Cavazzoni, S. de Gironcoli, P. Delugas, F. Ferrari Ruffino, A. Ferretti, N. Marzari, I. Timrov, A. Urru, S. Baroni, Quantum espresso toward the exascale, The Journal of Chemical Physics 152 (15) (2020) 154105. doi:10.1063/5.0005082.
URL https://doi.org/10.1063/5.0005082

[61] Oracle, I. Corporation, Lustre Software Release 2.x (2017).
URL https://doc.lustre.org/lustre_manual.xhtml

[62] M. Folk, A. Cheng, K. Yates, Hdf5: A file format and i/o library for high performance computing applications, in: Proceedings of supercomputing, Vol. 99, 1999, pp. 5–33.

[63] P. Corbett, D. Feitelson, S. Fineberg, Y. Hsu, B. Nitzberg, J.-P. Prost, M. Snirt, B. Traversat, P. Wong, Overview of the MPI-IO Parallel I/O Interface, Springer US, Boston, MA, 1996, pp. 127–146. doi:10.1007/978-1-4613-1401-1_5.
URL https://doi.org/10.1007/978-1-4613-1401-1_5

[64] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, P. Sadayappan, Scalable i/o forwarding framework for high-performance computing systems, in: 2009 IEEE International Conference on Cluster Computing and Workshops, 2009, pp. 1–10. doi:10.1109/CLUSTR.2009.5289188.

[65] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, C. Maltzahn, On the role of burst buffers in leadership-class storage systems, in: 2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), 2012, pp. 1–11. doi:10.1109/MSST.2012.6232369.

[66] Y. Kim, R. Gunasekaran, Understanding I/O workload characteristics of a Peta-scale storage system, Journal of Supercomputing 71 (3) (2015) 761–780. doi:10.1007/s11227-014-1321-8.

[67] R. Gunasekaran, S. Oral, J. Hill, R. Miller, F. Wang, D. Leverman, Comparative i/o workload characterization of two leadership class storage clusters, in: Proceedings of the 10th Parallel Data Storage Workshop, PDSW '15, Association for Computing Machinery, New York, NY, USA, 2015, p. 31–36. doi:10.1145/2834976.2834985.
URL https://doi.org/10.1145/2834976.2834985