UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ANDRÉ RAMOS CARNEIRO

Providing Support to Uncovering I/O Usage in HPC Platforms

Thesis presented in partial fulfillment of the requirements for the degree of Master of Computer Science

Advisor: Prof. Dr. Philippe O. A. Navaux
Coadvisor: Prof. Dr. Carla Osthoff

Porto Alegre
July 2022

CIP — CATALOGING-IN-PUBLICATION

Carneiro, André Ramos

Providing Support to Uncovering I/O Usage in HPC Platforms / André Ramos Carneiro. – Porto Alegre: PPGC da UFRGS, 2022.

93 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2022. Advisor: Philippe O. A. Navaux; Coadvisor: Carla Osthoff.

1. Visualization. 2. Lustre. 3. Parallel File System. 4. High-Performance Storage. 5. I/O Workload. 6. I/O Characterization. 7. Metadata. I. Navaux, Philippe O. A.. II. Osthoff, Carla. III. Título.

“A reader lives a thousand lives before he dies.
The man who never reads lives only one.”
— GEORGE R.R. MARTIN

ACKNOWLEDGMENTS

I would like to thank my advisor, Prof. Dr. Philippe Olivier Alexandre Navaux, and co-advisor, Prof. Dr. Carla Osthoff, for the guidance, support, and patience during the accomplishment of my master's.

I want to express my sincere gratitude to my friend and colleague Jean Luca Bez for providing invaluable insights and help in constructing this work.

I am grateful to my parents, Claudio and Helena Carneiro, to my wife, Priscila Carneiro, and to my friends Angélica de Fátima Tavares da Silva and Jamel Salles de Souza Leite, for always caring and supporting me during the most important times of my life, both good and bad.

I wish to show my appreciation to 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 document. URL: <http://sdumont.lncc.br>.

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 software, which have different requirements. These include input and output operations, directly impacting performance because the existing difference in processing and data access speeds. Thus, supercomputers must efficiently handle a 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. We used the Santos Dumont supercomputer as a study case. With our methodology's help, we identified inefficient usage and problematic performance factors, such as: (I) the system received an enormous amount of inefficient read operations, below 100 KiB for 75% of the time; (II) imbalance among storage resources, where the overload can correspond to $3\times$ the average load; and (III) high demand for metadata operations, accounting for 60% of all file system operations. We also provide some guidelines on how to tackle those issues.

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

Fornecendo Suporte Para a Descoberta das Demandas de E/S em plataformas de PAD

RESUMO

Plataformas de Processamento de Alto Desempenho (PAD) são necessárias para resolver os mais diversos problemas científicos de grande escala em várias áreas de pesquisa, tais como biologia, química, física e ciências da saúde. Pesquisadores utilizam uma infinidade de aplicações científicas, que por sua vez possuem diferentes requisitos. Dentre esses requisitos estão as operações de entrada e saída, que impactam diretamente o desempenho devido a diferença de velocidade existente entre o processamento e o acesso aos dados. Dessa forma, os supercomputadores devem lidar de forma eficiente com uma carga de trabalho mista ao armazenar os dados utilizados pelas aplicações. O entendimento do conjunto de aplicações e seus desempenhos ao executar em um supercomputador é primordial para entender a utilização do sistema de armazenamento, identificando possíveis gargalos, e orientando técnicas de otimização. Essa dissertação propõe uma metodologia e uma ferramenta de visualização para avaliar o desempenho da infraestrutura de armazenamento de dados de um supercomputador, levando em consideração as demandas e cargas de trabalho diversas do sistema durante um longo período de operação. Como estudo de caso, o supercomputador Santos Dumont foi estudado. Com a ajuda de nossa metodologia, identificamos uso ineficiente e fatores de desempenho problemáticos, como: (I) o sistema recebeu uma enorme quantidade de operações de leitura ineficientes, abaixo de 100 KiB por 75% do tempo; (II) desequilíbrio entre os recursos de armazenamento, onde a sobrecarga pode corresponder a $3\times$ a carga média; e (III) alta demanda por operações de metadados, representando 60% de todas as operações do sistema de arquivos. Também fornecemos algumas diretrizes sobre como lidar com esses problemas.

Palavras-chave: Lustre, Visualização, Sistemas de Arquivos Paralelos, Armazenamento de Alto Desempenho, Carga de Trabalho de E/S, Caracterização de E/S, Metadado.

LIST OF ABBREVIATIONS AND ACRONYMS

ACL	Access Control List
ALCF	Argonne Leadership Computing Facility
CF	Coverage Factor
CIFS	Common Internet File System
CN	Compute Node
CPU	Central Processing Unit
DKRZ	German Climate Computing Centre
DL	Deep Learning
ext4	Fourth Extended Filesystem
FLOPS	Floating-point Operations Per Second
FS	File System
GPFS	General Parallel File System
HDD	Hard Disk Drive
HPC	High-Performance Computing
I/O	Input and Output operation
ION	I/O Node
LDLM	Lustre Distributed Lock Management
LI	Load Imbalance
LMT	Lustre Monitoring Tool
LN	Login Node
LNET	Lustre Networking
LNCC	National Laboratory for Scientific Computing
MDS	Metadata Server
MDT	Metadata Target

ML	Machine Learning
MN	Management Node
MPI	Message Passing Interface
MPI-IO	Message Passing Interface Input/Output specification
NERSC	National Energy Research Scientific Computing Center
NFS	Network File System
NTFS	New Technology File System
OLCF	Oak Ridge Leadership Computing Facility
OSC	Object Storage Client
OSD	Object Storage Devices
OSS	Object Storage Server
OST	Object Storage Target
PFS	Parallel File System
QO	Quality of Operation
RAM	Random Access Memory
RDMA	Remote Direct Memory Access
RPC	Remote Procedure Call
SINAPAD	Brazil National High-Performance Computing System
SLURM	Simple Linux Utility for Resource Management
SMA	Simple Moving Averages
SSD	Solid State Drive
TOKIO	Total Knowledge of I/O
UFRGS	Federal University of Rio Grande do Sul
USB	Universal Serial Bus
VM	Virtual Machine

LIST OF SYMBOLS

μ Arithmetic Mean

σ Standard Deviation

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Supercomputers dominate the High-Performance Computing (HPC) environments with hundreds to thousands of compute nodes. These HPC systems solve the most diverse problems in various science domains: biology, chemistry, physics, and health sciences. Researchers of different areas use various scientific software, which has different requirements. For instance, 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 they can effectively read from and write to the storage system per second.

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 (MICROSYSTEMS, 2007) is one of the most adopted PFS on HPC systems, representing $\approx 30.5\%$ of the file systems used on IO500 list [1]. 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 cannot keep providing performance because of rising concurrency and interference (YILDIZ et al., 2016; YU et al., 2018). Aside from the I/O operations, another critical factor in the HPC storage management are the metadata operations, which are responsible for maintaining the file system directory tree, file access permissions, ownership, timestamps, attributes, etc. As the datasets increase, the metadata performance becomes critical and can quickly turn into a bottleneck (ALAM et al., 2011).

Lawrence et al. (2017) and Saini et al. (2012) demonstrate that different scientific

[1] https://io500.org/

applications have their performance impacted in diverse ways by Lustre, with some using the resources more efficiently than others. This variation is linked to specific workload requirements and Lustre simultaneously handling various applications under contention. Some factors that impose limitations and negatively impact the performance of Lustre are misaligned access patterns (BARTZ et al., 2015), load imbalance between storage servers (PATEL et al., 2019), and resource contention (NEUWIRTH et al., 2017). Besides, we should also consider that the existing I/O stack exposes many tunable parameters, such as stripe size of the PFS, seeking to provide performance improvements to diverse workloads. However, the misconfiguration of such parameters because of the users' lack of knowledge about its application's I/O operations can add to the observed poor performance, making understanding of I/O usage and behavior critical. Furthermore, a poorly performing I/O application could also negatively impact all others currently running in the system since the storage is a shared resource.

This research aims to understand the impact and uncover data storage needs in a supercomputer by evaluating Lustre's performance concerning the diverse I/O and metadata workloads from different domains and their demands. The utilization behavior was studied and compared over two periods comprising three months of operation, from March to May of 2020 and 2021, when there was 16.50 PiB of data movement through 109.79 billion I/O operations. We developed a methodology for continuous characterization and visualization of performance factors, such as small request sizes, load imbalance, and resource contention. We used the *Santos Dumont* Supercomputer (*SDumont*) [2] 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.

1.1 Contributions

The main objective is to obtain a comprehensive understanding of the utilization of the PFS of a supercomputer, identifying bottlenecks and providing possible guidelines on how to tackle those issues. Considering these goals, the main contributions are the following:

- A methodology developed to collect, analyze, and visualize I/O data from the PFS, using open-source software that does not require administrative privileges, allowing

[2] https://www.lncc.br/sdumont

it to be easily implemented and reproduced.

- A web application developed to streamline the visualization and analysis of the PFS usage data. Such a tool makes it easier to reproduce the analysis and study for different periods of interest.
- The comparison between periods from two years reveals how dynamic the I/O demands are, demonstrating the need for continuous PFS evaluation to better meet users' demands.
- Investigation of the I/O workload and usage behavior from Lustre's Object Storage Targets (OST) and SDumont's compute nodes. The study shows that the workload demand is not dominated by a single type of operation and can significantly vary across the period.
- An analysis of individual OST usage demonstrated a significant load imbalance across them during regular system operation.
- By crossing the I/O usage metrics from the compute nodes with information from the job scheduling management system was possible to identify problematic applications that could lead to overall performance degradation at the PFS servers.
- The analysis and characterization of the metadata operations show a considerable demand, with the metadata accounting for 60% of all file system operations.

1.2 Document Organization

The document is organized as follows. Chapter 2 presents some background concepts on the topics of this dissertation, and Chapter 3 discusses related work. Chapter 4 details the methodology developed to study a supercomputer's parallel file system. Results are discussed in Chapter 5. Chapter 6 summarizes the lessons learned and compares our findings with other systems. Finally, concluding remarks and future work are presented in Chapter 7.

2 BACKGROUND

In the following sections, we explain some important concepts that serve as a base for this dissertation. A brief overview of Parallel File Systems in High-Performance Computing is presented, followed by a description of the Lustre PFS and the Santos Dumont supercomputer.

2.1 Parallel File Systems

Supercomputers dominate the High-Performance Computing (HPC) environments. They are clusters of machines composed of hundreds to thousands of compute nodes (CN) used for processing, login nodes (LN) through which users connect to the system and submit their jobs (i.e., a particular application execution), a resource and scheduler manager (which can be collocated with the LN or placed in a dedicated management node – MN) that orchestrate the concurrent execution of users' applications on the CNs, a shared storage system to house the data, and a high-speed network to connect all the components. These HPC systems solve the most diverse problems in various science domains: biology, chemistry, physics, and health sciences. Researchers of different areas use various scientific software, which has different requirements. For instance, applications can be serial or parallel and read/write different amounts of data in multiple formats and sizes. This scenario leads to the supercomputers having to handle mixed workloads.

The rapid expanse in the technology of processing chips and high-speed networks allows the building of massive supercomputers, which have millions of processing cores and PiB of main memory. For example, Fugaku [1], the fastest supercomputer according to the November 2021 edition of the Top500 list [2], has a total of $7,630,848$ computing cores and 5.087 PiB of memory.

Moreover, the infrastructure that stores the datasets used by scientific applications also has to provide high-performance access to efficiently perform the input and output (I/O) operations. I/O is commonly used by scientific applications to achieve goals like storing the numerical output from simulations for later analysis and visualization, loading initial conditions or datasets for processing, checkpointing to files that save the state of an application in case of system failure, or Implementing 'out-of-core' techniques for algo-

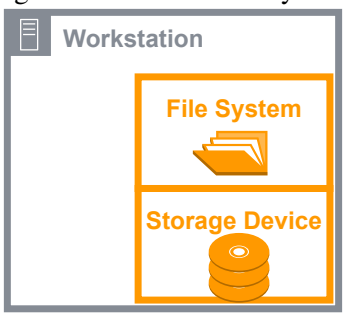
[1] https://www.r-ccs.riken.jp/en/fugaku/project
[2] https://www.top500.org/lists/top500/2021/11/

rithms that process more data than can fit in system memory. But as the supercomputers grow, the I/O needs are also getting bigger because (i) there is more "room" to execute applications concurrently, and (ii) the data size growth accompanies the total available system memory. This scenario leads to the datasets generated by scientific applications increasing exponentially in both volume and complexity (XIE et al., 2012; WANG et al., 2016).

Although advances in data storage architectures provide a better performance, for instance, by using SSD devices on the shared storage system (CAULFIELD; GRUPP; SWANSON, 2009) or as client-side cache device (QIAN et al., 2019), 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, making the shared storage infrastructure one of the main bottlenecks for achieving sustainable performance.

In computing environments, the file system has two key roles: organizing and maintaining the file namespace (directory tree structure) and storing and retrieving the contents of files and their attributes on a storage device (e.g., HDD, SSD, USB stick, CD-ROM) (ARPACI-DUSSEAU; ARPACI-DUSSEAU, 2018). Local file systems (Figure 2.1) are used by a single client with direct access to the disk (e.g., NTFS [3] or ext4 [4] in a workstation or CN). Networked file systems (Figure 2.2) provide storage space and single namespace access to one or more clients who might not have direct access to the disk (e.g., NFS [5] or CIFS [6] provided by a server).

Figure 2.1 – Local File System.

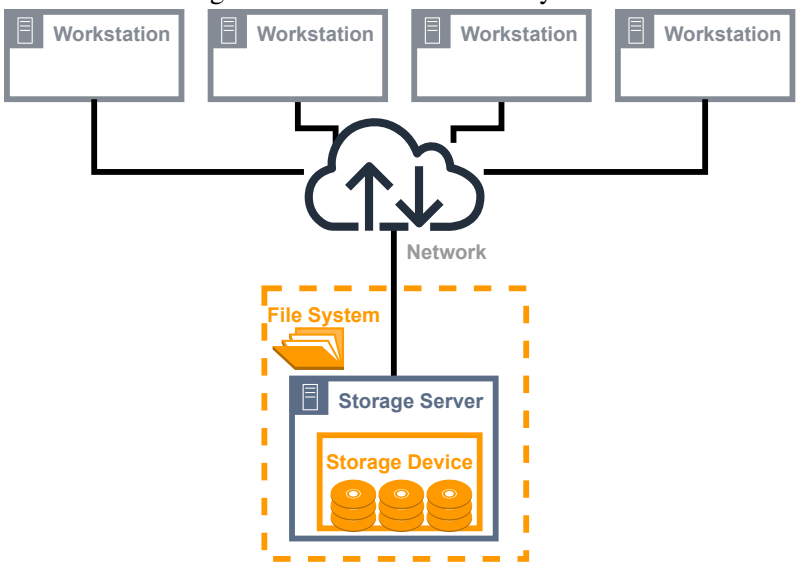


Source: Author

The traditional centralized architectures are not designed to have many outstanding data requests and have scalability issues because there is only one access point to

[3] https://www.ntfs.com/index.html
[4] https://ext4.wiki.kernel.org/index.php/Main_Page
[5] https://datatracker.ietf.org/doc/html/rfc7931
[6] https://cifs.com/

Figure 2.2 – Networked File System.



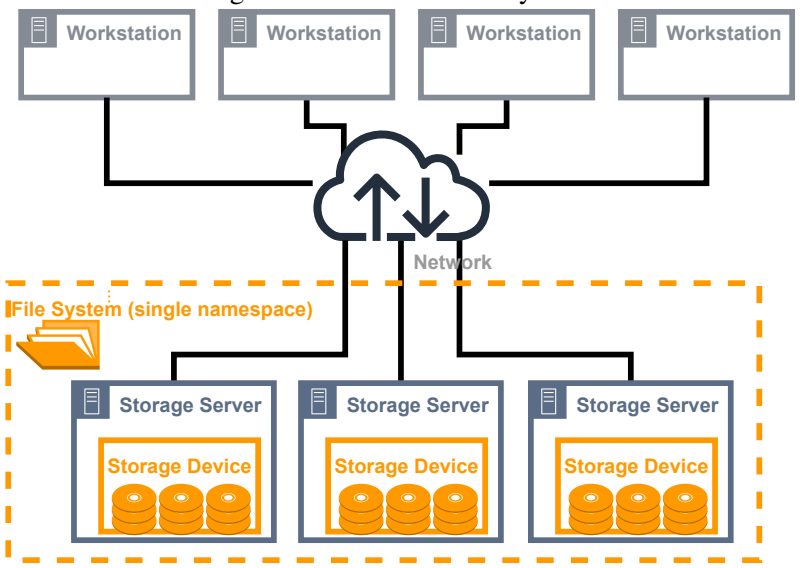
Source: Author

the data (OLIVARES et al., 2001), unable to handle requests from thousands of CNs. Parallel File Systems (PFS) were designed to provide greater data access throughput for concurrent and independent access from scientific applications to overcome the deficiencies of the centralized architecture of traditional storage that cannot cope with current supercomputers. In the same way that supercomputers use various CNs together to obtain more computational power, PFS (Figure 2.3) is a special kind of networked file system that uses several dedicated storage servers to aggregate the individual bandwidth and provide high throughput while reducing the overhead of processing I/O requests (PRABHAT; KOZIOL, 2014; CHING et al., 2007). A PFS differs from a non-parallel (centralized) file system in that the data can be distributed across multiple storage servers, which can be accessed simultaneously to increase the bandwidth of the file system.

According to Prabhat and Koziol (2014), there are some key requirements that PFSs should meet in HPC environments:

- **Division of files in data blocks:** The PFS should be capable of dividing a file into blocks of data (commonly called *striping*) across storage servers, allowing parallel access to different parts of the file by the processes of a parallel application. This way, the data access is not restricted to a single storage server and increases the performance.
- **Access to individual data blocks:** Aside from dividing the file, the PFS should provide a mechanism for each process in a parallel application to access distinct subsets of the data.

Figure 2.3 – Parallel File System.



Source: Author

- **Single namespace:** Since the file system is scattered across several storage servers, the PFS should deliver the aggregated file systems as a single namespace. CNs should see the whole file system as a single unity instead of various storage islands.
- **Fault-Tolerance:** Since the PFS comprises two or more storage servers, it must implement mechanisms, through hardware and/or software, to deal with failures and keep the service as highly available as possible. Being fault-tolerant overcomes the single point of failure of traditional centralized storage.
- **Locking:** The PFS must implement a locking mechanism to manage concurrent access to files, avoiding corruption. CNs should be able to obtain locks on data units of the file they will access before I/O occurs.
- **Cache coherency:** By allowing multiple access to the same file (or parts of the file) from numerous CNs, the PFS must implement a cache consistency protocol to avoid incoherent data. This coherency is usually implemented through locking mechanisms (as long as the CN has a lock, it knows its cached data is valid).
- **API:** Legacy applications that rely on standardized libraries to perform their operations should access the files in a PFS the same way on a local file system without recompilation. However, the PFS should provide a specialized API that allows developers to obtain greater control over how the data are divided and distributed. In this fashion, it is possible to optimize the application to take advantage of the special characteristics of the PFS.

- **Scalable capacity and performance:** To meet the increasingly high demands of I/O, both in performance and size, the PFS must be scalable in a scale-out fashion. Its capacity and performance should be easily increased by adding more storage servers. The PFS must be scalable to attend to the demands of thousands of clients, support hundreds of servers, house thousands of storage devices, and provide vast storage of capacity and hundreds of GiB/s of system bandwidth.

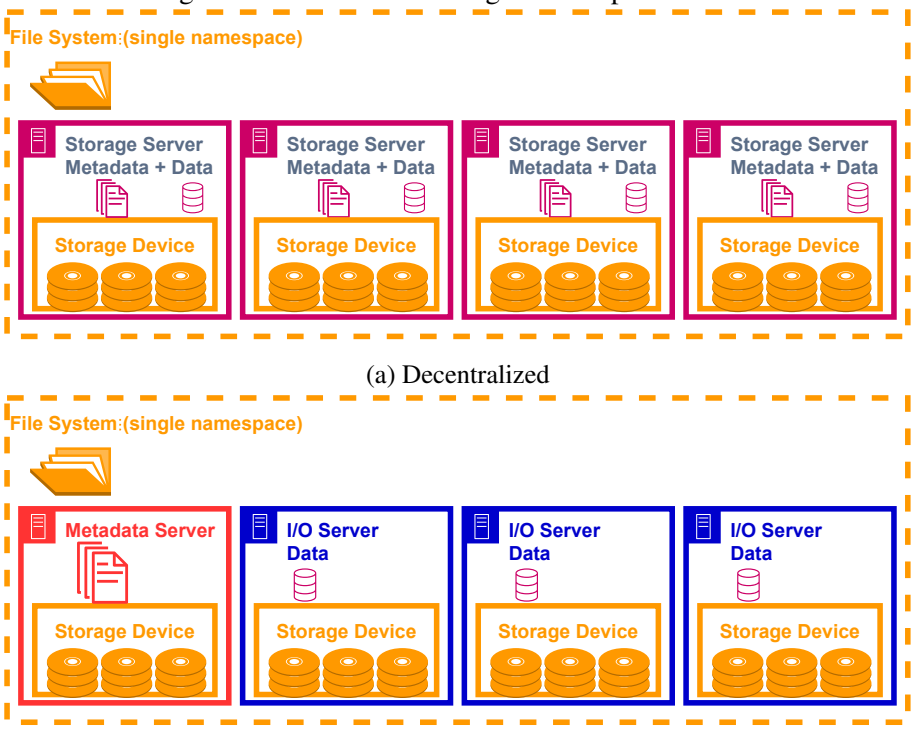
The PFSs cannot always keep up with the generated concurrency because of the rising number of CNs on supercomputers. For this reason, many leadership-class HPC platforms utilize an I/O forwarding layer (ALI et al., 2009; OHTA et al., 2010), which stands between the CNs and the PFS. This layer comprises dedicated I/O nodes (ION) responsible for receiving the I/O requests from the CNs, performing rescheduling or aggregation, and directly interacting with the storage servers. The I/O forward layer decreases contention in accessing the PFS because it handles requests from fewer nodes instead of thousands of CNs. The study presented in this work does not contemplate an I/O forwarding layer. However, the methodology developed, described at Chapter 4, could be adapted to study how the I/O forwarding layer interacts with the PFS based on the demands from the CNs.

Currently, there are numerous PFSs available, and most of them share various characteristics. These systems have two key aspects that distinguish them: how they store the data and handle metadata management. The file data can be stored as data stream blocks using the local file system on the storage servers or as multiple objects on object-based storage (MESNIER; GANGER; RIEDEL, 2003). Object-based storage is the general term for how data unities, called objects, are organized and handled. Each object is composed of three properties: (i) the file data as a variable-length sequence of bytes; (ii) an expandable amount of metadata, which holds contextual information about the data; and (iii) a unique global identifier, which is like an object's address to find it throughout the distributed storage system. The objects are stored on Object Storage Devices (OSD) that use a specialized local file system that implements the object storage primitives.

Metadata is information about data such as size, permissions, location among the storage servers, owner, access control lists (ACLs), and access times. As all basic file system operations involve metadata operations, metadata access scalability impacts the whole system. Regarding the metadata management in PFS, there are two types of architectures: decentralized and centralized (PRABHAT; KOZIOL, 2014). In the decentralized

architecture (Figure 2.4a), the metadata is collocated with the file data and spread throughout the file systems. The storage server manages both I/O operations and maintains the directory hierarchy. In this architecture, the directory tree structure of the file system is distributed to the storage servers. Some PFSs allow the utilization of dedicated servers to handle only the metadata operations, but the whole directory structure is spread among the servers.

Figure 2.4 – Metadata Management Representation.



(a) Decentralized

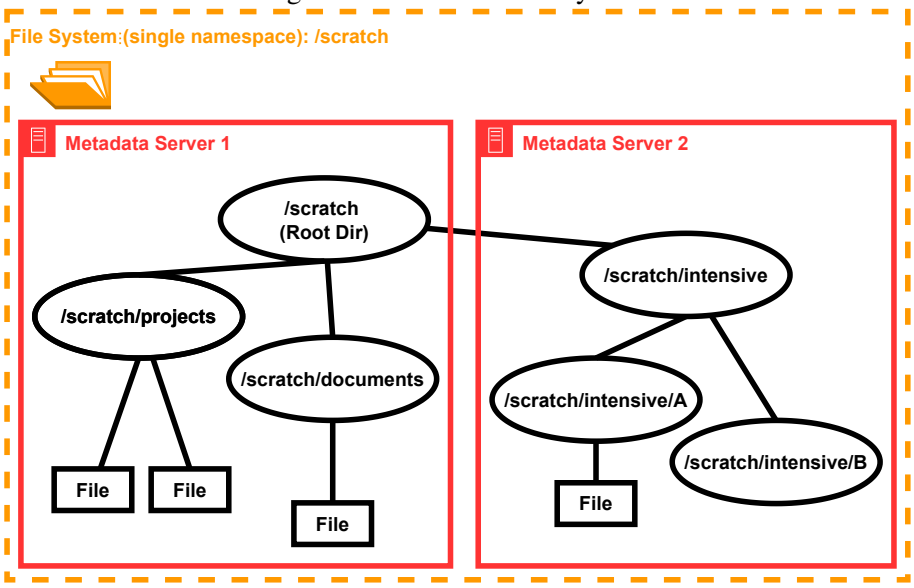
(b) Centralized

Source: Author

The centralized metadata architecture (Figure 2.4b) is characterized by having a dedicated server to handle only the metadata operations and maintaining the directory tree structure. Operations such as `open`, `close`, `remove` and `rename` are executed through the metadata server. For example, during a write request: (i) the client process contacts the metadata server with an `open` operation; (ii) the metadata server checks the file's ACL and informs where and how the file is stored; (iii) the client directly access the storage servers to write the data. Some PFSs that use this type of architecture allow the utilization of two or more metadata servers to divide the namespace logically. However, each server is still responsible for managing access to the whole directory tree branch. Figure 2.5 depicts a directory tree where the *Metadata Server 1* is responsible for managing the `/scratch` branch (the root directory of the file system) and everything below it, except

accesses to the `/scratch/intensive` branch, which are managed by the dedicated *Metadata Server 2*.

Figure 2.5 – Parallel File System.



Source: Author

PFSs are the de-facto file system type for HPC systems, and Lustre is considered the most popular one. Lustre is the most used PFS of the top 10 disclosed supercomputers from the November 2021 TOP500 list, with 4 deployments. Concerning the list from 2021 of the IO500 [7], with 72 submissions, Lustre accounts for 30.5%. The following section describes the Lustre file system.

2.2 Lustre Architecture and its Deployment on SDumont

This section discuss the Lustre's architecture and the motivation for using the SDumont supercomputer as a test-bed.

2.2.1 Lustre's Architecture

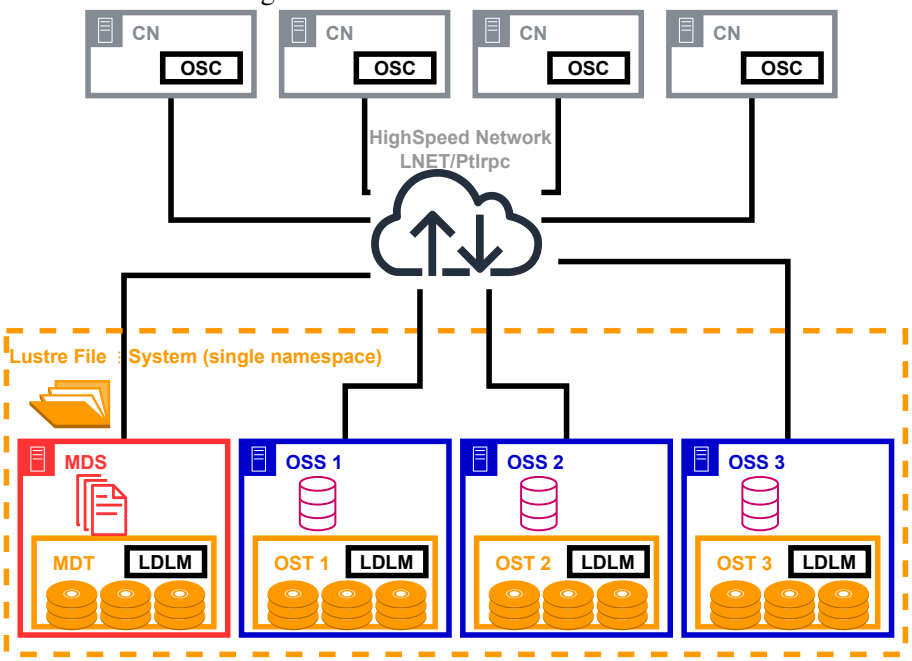
The Lustre PFS is an open-source client-server object-based file system implemented entirely on the Linux Kernel, developed for high-performance environments. It provides a POSIX-compliant namespace and scalable I/O resources. Instead of using

[7] https://io500.org/

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 (layout, size, access permissions, owner, timestamps, etc.). These objects are implemented by the Lustre object storage device, an abstraction that enables the use of different backend file systems (ldiskfs or ZFS). A single storage device instance corresponds to a single backend storage volume and is termed a *storage target*.

The key components of Lustre's architecture are Metadata Servers (MDS), Metadata Targets (MDT), Object Storage Servers (OSS), Object Storage Targets (OST), Clients, Lustre Distributed Lock Manager (LDLM), and the Lustre Networking (LNET), depicted by Figure 2.6:

Figure 2.6 – Lustre PFS Architecture.



Source: Author

- **MDS** is 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.

- **MDT** is the backend storage target responsible for holding the metadata objects.
- **OSS** handle the file I/O operations and export one or more OSTs to the clients.
- **OST** is the backend storage target responsible for holding the file data objects. It can be exported actively by only one OSS.
- **Clients** combine the MDTs and OSTs in a single namespace while communicating the users' requests to the MDSs or OSSs. Each Lustre I/O operation originates with a system call from a user process on a CN. Each CN has a file cache as its I/O buffer and invokes a local Lustre kernel module called Object Storage Client (OSC) to handle file operations and I/O. The OSC performs I/O by issuing Lustre RPC calls to OSSs. Each Lustre I/O operation to an OSS is a read or a write-on exactly one object which resides on an OST.
- **LDLM** is a service provided by storage targets (MDT and OST) in addition to object storage services. LDLM locks are used to serialize conflicting file system operations on objects managed by that target and are used to ensure distributed cache coherency.
- **LNET** provides the communication infrastructure stack for every component, allowing message passing, remote direct memory access (RDMA), and high-speed communication. The Lustre RPC layer (Ptlrpc) is built on top of LNET to provide robust client–server communications in the face of message loss and server failures.

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`. Summing up, the file is composed of *N* `stripe_count` objects that are stored in the OSTs in a round-robin fashion, and each object is composed of one or more data chunks (*stripes*) of size `stripe_size`. Lustre allows the user that creates the file to specify the `stripe_size`, `stripe_count`, and the starting OST, which will hold the 1st object. If not specified, the system uses the default values for `stripe_size` and `stripe_count` (varies from deployment to deployment), and selects the starting OST at random. Striping can improve performance in file access as it makes it possible to aggregate multiple data servers' bandwidth to access a single file using parallel I/O operations. When a file is open on Lustre, the client first contacts the MDS to retrieve the file layout. The file layout has information about the file striping, and on which OSTs

it is stored. After that, the client interacts directly with the OSSs for the subsequent I/O operations without communicating with the MDS.

Lustre PFSs are typically optimized for high bandwidth: they work best with a small number of large, contiguous I/O requests rather than a large number of small ones (i.e., small numbers of large files rather than large numbers of small files).

2.2.2 The SDumont

The *Santos Dumont* Supercomputer (*SDumont*) [8], a Bull/Atos machine located at the National Laboratory for Scientific Computing (LNCC) [9] in Brazil, is an example of an HPC environment with significant heterogeneity of research on several science domains, 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*) [10] 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 Lustre PFS shared storage, deployed through the CRAY/HPE ClusterStor 9000 v3.3, composed of one MDS and ten OSS. Each server has one storage target 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 [11], 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. (2019) 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 characteriza-

[8] https://sdumont.lncc.br/
[9] https://www.lncc.br/
[10] https://www.lncc.br/sinapad/
[11] https://www.hpe.com/psnow/doc/c04663932?jumpid=in_lit-psnow-red

tion using monitoring data collected during a week of operation. The study presented that the writing operations were responsible for 80% of the workload and that the aggregate performance did not reach 15% of the system's maximum peak. However, there was still no comprehensive data on Lustre's behavior and 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 executed on SDumont.

3 RELATED WORK

The previous chapter depicted how complex the storage for an HPC system can be, which contributes to many factors impacting the application's performance. This chapter first discusses some of the most common performance limiting factors that users, developers, and system administrators should pay attention to, alongside related work exposing how these factors impact production systems. The last section presents some works related to the methodology proposed in this dissertation to evaluate the PFS of an HPC system.

3.1 Performance Limiting Factors

This section discusses some performance limiting factors, divided into two fronts. First, we present how an application interacts with the PFS in an individual view. Second, we expose the problems that may arise when running multiple applications in a single environment.

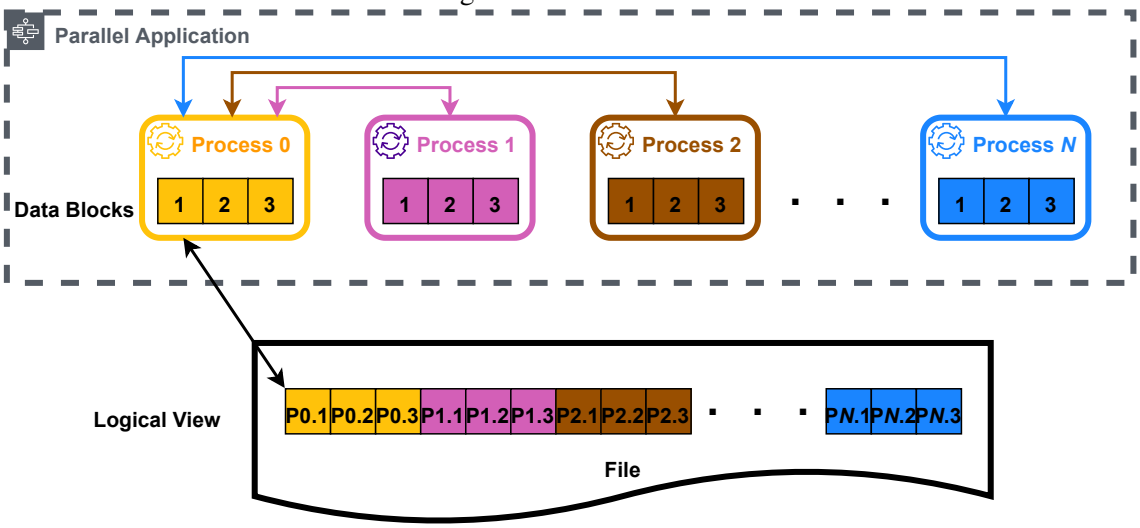
3.1.1 How applications interact with the PFS

Different scientific applications from diverse scientific domains have their performance impacted in myriad ways by the PFS, with some using the resources more efficiently than others. The variability in performance, influenced by the applications' access patterns, can frustrate users' expectations of how their applications will perform when using the PFS. The access pattern (NIEUWEJAAR et al., 1996) describes how the application interacts with the storage system concerning the type of operation (read or write) performed, whether it performs parallel or serial I/O, the number of files used, the spatial data locality accessed inside the file, and the operation's transfer size. Also, file type, I/O library, bandwidth, and PFS type can affect performance.

A parallel application can perform its I/O operations in a serial or parallel model (CHING et al., 2007). On the serial I/O (Figure 3.1), only one process (usually the "rank zero") of the application is responsible for performing all the I/O operations (i.e., reading or writing the entire file). Every other process must send/receive its data to/from the designated I/O process. This approach is simpler to implement but does not scale because

of memory and bandwidth constraints of the CN executing the process responsible for the I/O, and because a single CN cannot access the total aggregated bandwidth of the PFS. Also, the time spent with I/O increases with the amount of data and processes. On the other hand, the parallel I/O is characterized by all, or a subgroup of, processes performing the I/O operations. This way, each CN is responsible for handling a smaller amount of data, and the application can scale to a higher number of processes because of the aggregated bandwidth of all CNs involved.

Figure 3.1 – Serial I/O.

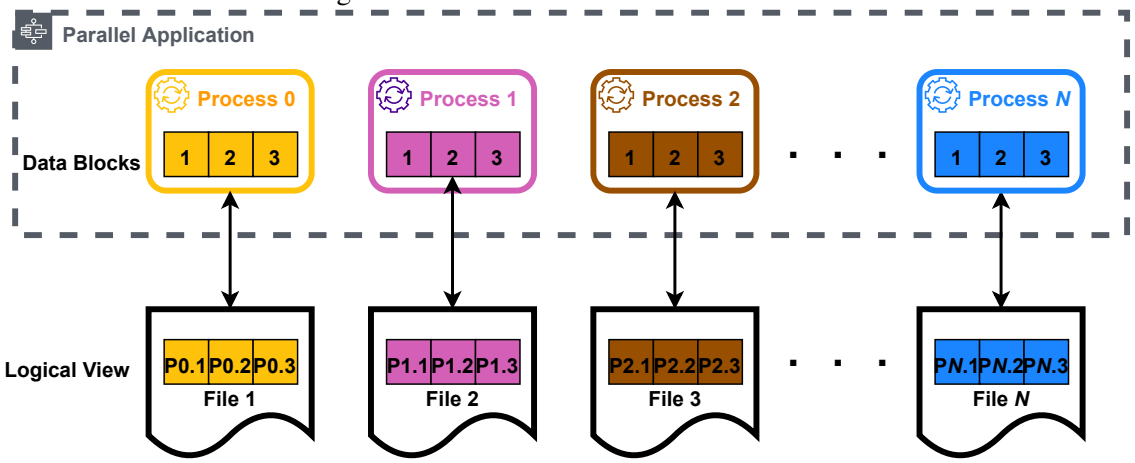


Source: Author, inspired by Ching et al. (2007)

Also according to (CHING et al., 2007), the parallel I/O is usually implemented in two different approaches regarding the number of files used. The applications can perform parallel I/O using *file-per-process* or *shared-file*. In the *file-per-process* approach, depicted by Figure 3.2, each process performs its I/O operations isolated in their individual files. On the other hand, in the *shared-file* approach, depicted by Figure 3.3), all processes perform their I/O operation on a single shared file. TThe file-per-process usually presents higher throughput because of not performing synchronization among processes during I/O operations or because it can avoid lock contention at the PFS level. However, it can present scalability problems at a high processes count because of metadata access overhead (ALAM et al., 2011). An application using *file-per-process* running on a thousand processes generates a thousand *create* and *open* requests on the metadata server. This scenario leads to severe bottlenecks as the number of cores increases on the supercomputer. Another problem with the *file-per-process* is that it usually needs an additional post-process to merge all files for later analysis or visualization. Despite the *shared-file* approach needing some synchronization, it can make use of aggregation and coordina-

tion optimizations like that provided by MPI-IO (THAKUR; GROPP; LUSK, 1999) (or libraries developed on top of it). The *file-per-process* and *shared-file* approaches are also referred to as N-to-N and N-to-1.

Figure 3.2 – Parallel I/O - File-Per-Process.

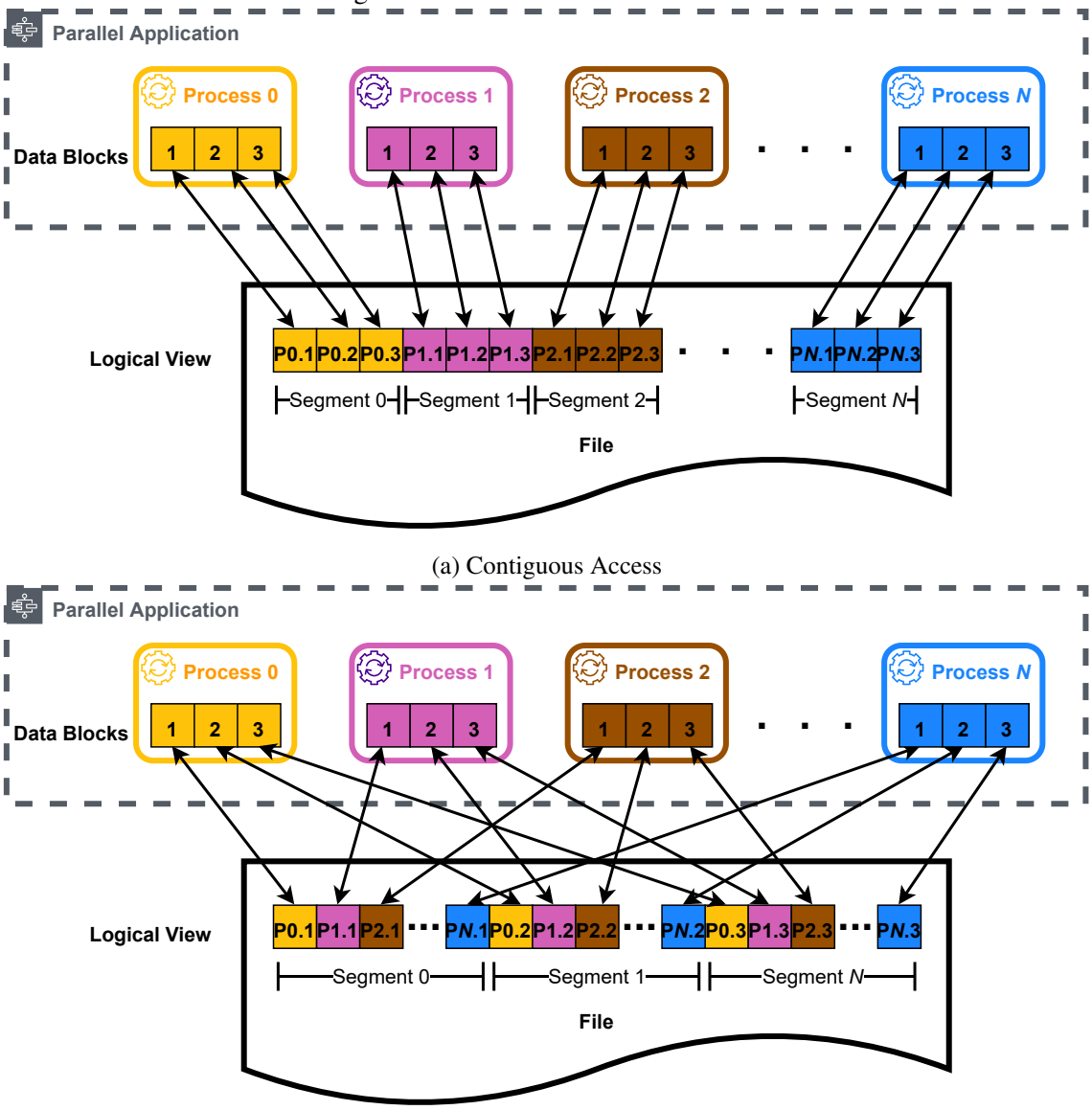


Source: Author, inspired by Ching et al. (2007)

There are different ways the data on the file can be placed and accessed by the processes while using the shared-file approach (BYNA et al., 2008; GE, 2010). The most typical ways the processes can access the file are: (i) contiguous (Figure 3.3a), where each process accesses whole segments of the file, from beginning to end, on consecutive requests that begin precisely where the previous request ended; and (ii) strided (Figure 3.3b), a non-contiguous pattern where each process access different segments of the file, performing requests of the same size and incrementing the file pointer by the same amount (displacement) between each request. The strided access can also be random, where the requests are different sizes, resulting in different displacements. Best performance usually comes when the data is accessed contiguously in memory and disk. Non-contiguous (strided) access results in random accesses by performing many seek operations to move the file pointer, which, in turn, breaks the buffering and prefetching policies employed at the PFS clients, storage servers, and storage devices (LIU et al., 2010; DING et al., 2007).

The transfer size (also known as request size) is the amount of data transferred at each I/O operation. Small transfer sizes harm the application's performance when using a PFS because the storage servers must be accessed multiple times for a small number of bytes, the latency of the network used to access the storage servers, and the latency of the storage devices. Larger I/O operations and matching the PFS stripe setting may improve performance because more data is being transferred on a single operation, hiding

Figure 3.3 – Parallel I/O - Shared-File.



(a) Contiguous Access

(b) Strided Access

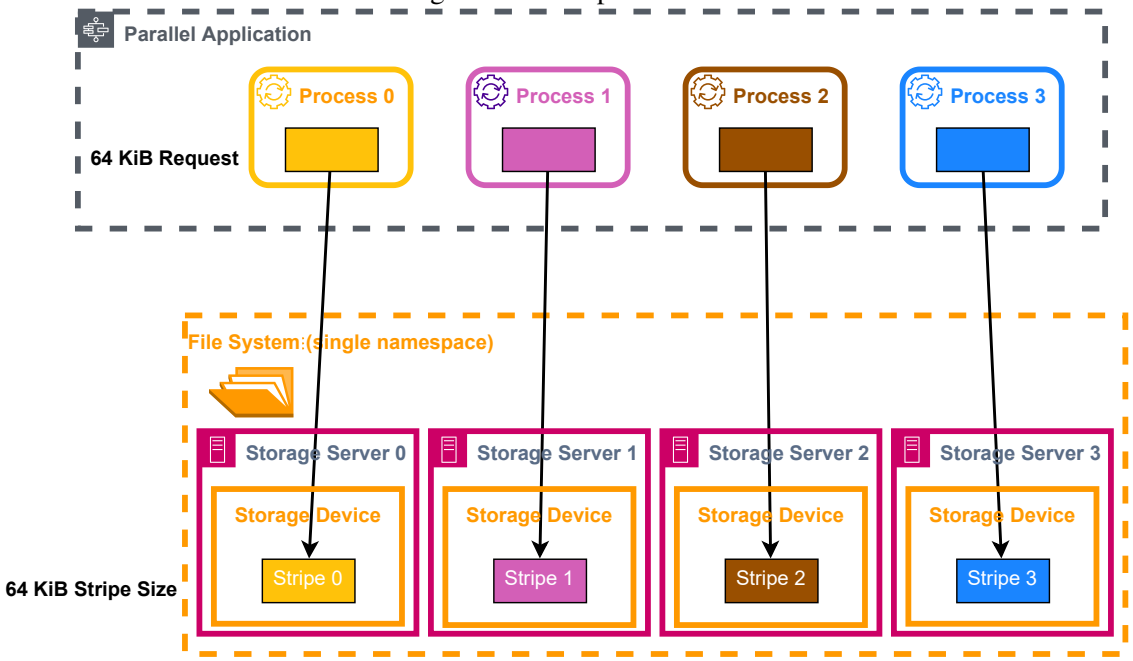
Source: Author, inspired by Byna et al. (2008), Ge (2010)

the latency of I/O operation.

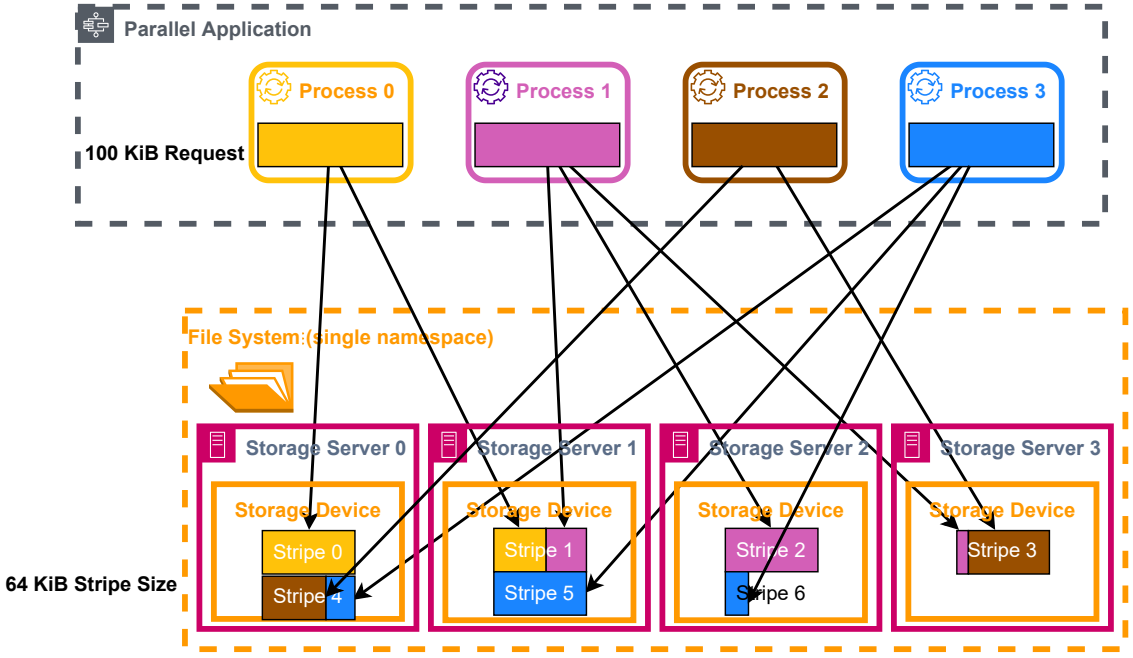
The transfer size is also related to another aspect when accessing a PFS, which is the misaligned stripe access (LIAO; CHOUDHARY, 2008; LANG et al., 2009; DICKENS; LOGAN, 2009). The PFS allows the files to be striped (divided into units of "stripe size") over multiple storage servers to provide parallel access and take advantage of aggregate I/O capacities, such as network and storage device bandwidth. With aligned accesses, the application's process performs requests whose sizes align perfectly with the file's stripe, directly accessing one storage server. Misaligned access occurs when the request size differs from the stripe's size. The issuing of misaligned data accesses degrades

the application's performance. Each request must be divided into uneven smaller requests to communicate with more than one storage server, limitng the resulting throughput to the PFS. Figure 3.4a depicts the aligned access, where each 64 KiB request is performed on individual 64 KiB stripes, accessing only one storage server per request. In Figure 3.4b, the 100 KiB requests have to be divided into two or more operations because of the stripe size of 64 KiB, enforcing accesses on different storage servers per request.

Figure 3.4 – Stripe Access.



(a) Aligned Access



(b) Misaligned Access

Source: Author, inspired by Dickens and Logan (2009)

The lock mechanism implemented by the PFS also affects the misaligned access. If a process makes a write request for an amount of data larger than a file stripe, it must acquire locks from those storage servers responsible for the stripes that are part of the request. The Lustre PFS, for example, enforces I/O atomicity by having the process obtain all the locks to these stripes and hold the locks until the servers receive the entire data to be written, which serialize the accesses to the stripe to mediate the contention. As an example, the accesses from processes 0 and 1 to write on the stripe 1 located at the storage device of the storage server 1, depicted on 3.4b, have to be serialized, seriously degrading I/O performance. The read performance is less affected by this characteristic because read locks are shareable among processes.

Various studies have shown how access patterns can influence the applications' performance. Liao and Choudhary (2008) investigate file domain partitioning methods regarding the PFS's lock boundaries. The experiments conclude that the applications achieve the best write performance when their requests are aligned with the stripe's lock-boundary of the PFS.

Lang et al. (2009) present a case study of the I/O performance and scalability challenges on Intrepid, a supercomputer with $163,840$ cores and served by a PFS composed of 128 storage servers, located at Argonne Leadership Computing Facility (ALCF). They used a collection of benchmarks testing different request sizes to measure the behavior of the storage system under various workloads. The results show that for small misaligned requests, the performance is severely reduced by a factor of $2\times$. The best performance was achieved with shared-file and aligned accesses. The results also point out two other deficiencies: (i) at a higher number of processes, there was roughly a 20-25% decrease in performance with file-pre-process compared to shared-file; and (ii) that for misaligned and file-per-process workloads, the storage device sees many concurrent smaller requests, often out of order, which makes the file system unable to aggregate I/O requests. Carns et al. (2011) is another work that studied the Intrepid. Using Darshan (CARNS et al., 2009) to explore the applications deployed at that machine, they found that despite the file-per-process model being the most popular method for I/O, it only scales up to about 8192 nodes ($32,768$ cores = 20% of the machine). The performance begins to suffer scalability issues at higher core counts, making the shared-file approach a better choice.

Saini et al. (2012) surveyed the NASA scientific and engineering applications, characterizing the I/O requirements of typical applications on the Pleiades supercomputer using the Lustre PFS. They show that file striping will improve the performance of appli-

cations that perform I/O to a single or multiple large shared files. However, it has little effect for applications that perform serial I/O and when multiple nodes perform I/O simultaneously to different small files (each < 10 MiB). The application needs to issue bigger transfer sizes to take advantage of the stripe size. Other critical aspects found are that applications performing many operations with small transfer sizes (<= 4 KiB) presented low performance and inefficient use of Lustre PFS, while bigger transfer sizes lead to better performance. Also, the performance of the file-per-process applications can be better than the single-shared file by a factor of 3.1 to 6.3 on a lower number of cores but suffers from scalability problems when the core count approaches half of the supercomputer.

Zhang et al. (2013) used a benchmark to investigate the effects of misaligned access on storage system performance in the Darwin cluster at Los Alamos National Laboratory. They found that misaligned data access results in consistently lower throughputs than their respective aligned counterparts. Using 16 processes and misaligned requests larger than the PFS stripe size, the throughput presented a reduction of 52%.

Yildiz et al. (2016) studied the interference on various levels of the storage subsystem. Regarding the data spatial locality, they show that when executing on a storage device made of HDDs, a strided (non-contiguous) application can be up to $12\times$ slower than a contiguous one. When using SSDs, which have no rotational delay and are less susceptible to seek operations originated by the random access, the decrease in performance is only $2.75\times$.

Lawrence et al. (2017) evaluate the PFSs of several European HPC centers using the benchio [1] benchmark. The study shows that it is necessary to use as many stripes as possible to get the best parallel write performance for a single-shared file case, while the file-per-process pattern presents better results when the files are not striped. It also shows that the I/O operation request and stripe sizes should match to get closer to the theoretical maximum performance. Generally, the larger the amount of data written per writer, the larger the stripe size should be.

Isakov et al. (2020) used an explainable machine learning (ML) platform to analyze $89,844$ Darshan logs collected on the Argonne Leadership Computing Facility (ALCF) Theta supercomputer. Among the main findings of applications' access patterns that correlate with bad performance are issuing small transfer sizes (in the 1–10 KiB range), using many files during the execution (file-per-process), and performing random access operations. Aspects that presented the most positive correlation with performance

[1] https://github.com/EPCCed/benchio

were consecutive and sequential I/O and large requests (1 MiB to 4 MiB, i.e., multiples of the PFS stripe size).

3.1.2 Concurrency, Contention, and Interference

Concurrency, contention, and interference are closely related concepts that significantly affect the system throughput, performance, and user experience of the PFS. Concurrency means multiple utilization is happening simultaneously on a shared resource (SKINNER; KRAMER, 2005; HWU; KEUTZER; MATTSON, 2008; ARPACI-DUSSEAU; ARPACI-DUSSEAU, 2018). Concurrency can occur on various processors in a CNs (and multiple cores in a single processor chip), various cores using the network interface or the storage disk, multiple CNs utilizing the network, multiple applications (or a parallel application) running on one CN, multiples applications running in a supercomputer on different sets of CNs, and multiples applications performing I/O on the PFS.

The contention is a conflict over simultaneous access to a shared resource (LIANG et al., 2019; JOKANOVIC et al., 2010; BLAGODUROV; FEDOROVA, 2012; CHUNDURI et al., 2019). The conflicts must be resolved through locks, semaphores, and scheduling, favoring one access over the other. In I/O for HPC, the primary shared resource that applications contend for is the PFS, which comprises components that can become a point of contention. The CNs interact with many storage servers in parallel to access data. As a result, storage servers have to serve many clients concurrently. An enormous amount of concurrent access to the storage server can cause severe I/O contention and impacts the overall performance of PFS. The I/O contention includes the competition of limited bandwidth of storage servers and the lock contention. For example, a parallel application can perform parallel access to a file stored on a PFS if the file is striped and each task of the application access exactly one individual stripe. If two or more tasks try to access the same stripe for a write operation, the PFS must mediate this access by performing locks on the stripe and serializing the accesses. This behavior results in performance degradation because the other tasks have to wait for the completion of the write request.

I/O interference in HPC can be defined as “the performance degradation observed by any single application performing I/O in contention with other applications running on the same platform” (YILDIZ et al., 2016). Therefore, the rise in the concurrency from the ever-larger supercomputers leads to an increase in contention scenarios, which, in turn,

increases the number of applications suffering from interference. Managing such a concurrency scale poses critical challenges to the HPC I/O system (HU et al., 2016; XIE et al., 2012). When a CN issues a large request to a striped file, this request is split into smaller requests sent in parallel to several storage servers. The operation completes only when all these servers have treated their part of the initial request. Hence, any slowdown experienced by a single server because of contention leads to a global slowdown for the entire operation. Suppose two servers decide to serve requests from different applications in a different order. In that case, both applications will suffer interference from a slowdown observed in servers that have not prioritized their request.

Various studies show the harmful effects of contention and interference on I/O for HPC. Zhang and Jiang (2010) point out that frequent disk head seeks, because of the access interference on each storage server while concurrently serving requests from different processes, can seriously hurt the performance of a system by a factor of up to $5\times$. The study of Hashimoto and Aida (2012) evaluates the performance degradation in each application program when Virtual Machines (VMs) are executing two applications concurrently in a physical computing server. The experimental results indicate that the interference among VMs running two HPC application programs with high memory usage and high network I/O in the physical computing server significantly degrades application performance.

A more general study by Xie et al. (2012) analyzed the behavior of a center-wide shared Lustre parallel file system on the Jaguar supercomputer and its performance variability. One performance degradation seen on Jaguar was caused by concurrent applications sharing the filesystem. Bhatele et al. (2013) investigated the performance variability in Cray machines and found that the interference of multiple jobs that share the same network links is a serious factor for high-performance variability, presenting a reduction of 27.8% in performance.

Yildiz et al. (2016) performed a systematic study of root causes of I/O interference in different components of the PFS, presenting a performance degradation by up to $2.5\times$ under interference. They found that significant variability often arose from poor flow control in the I/O path rather than the presence of a single bottleneck in only one component of the storage system. All these studies highlight the impact of application interference on HPC systems.

3.2 File System Evaluation

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. Luu et al. (2015) analyzed Darshan's logs (CARNS et al., 2011) 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 all the I/O paradigms (file-per-processes, shared file, and sub-setting I/O) are represented among the best and worst applications' performance, which means no single approach always delivers the optimal performance. Furthermore, one-third of all the jobs spent more than half of their execution time on metadata operations. The system-wide evaluation found that the aggregate throughput for three-quarters of the applications never exceeds 1 GB/s, roughly 1% of the average peak platform bandwidth available. Darshan is an I/O characterization tool designed to capture an accurate picture of application I/O behavior. There are two methods to get the I/O trace logs with Darshan. The application must be configured to use the Darshan runtime library during its execution or compile the application with special wrappers that directly link it to the Darshan library. The use of only Darshan logs to perform platform-wide analysis has some drawbacks. Darshan might be unable to trace every application executed on the supercomputer (because of lack of compatibility or not configured on the user compiled/installed application), leading to missing important behavior. To get a current and accurate state of the supercomputer usage, analyzing the enormous number of Darshan logs needs to be re-executed, which might be a significant time-consuming activity. There is no server-side information, missing the big picture of the PFS utilization.

Lockwood et al. (2018a) used system monitoring through the Total Knowledge of I/O (TOKIO) framework (LOCKWOOD et al., 2018b), benchmarks, and active probing on the I/O storage infrastructure (Lustre File System and GPFS) of two leadership-class HPC centers (NERSC and ALCF). They analyzed data over a year of production operation to identify the reasons for low performance and how to improve it. They investigated applications on multiple time scales, seeking to identify absolute performance and variability trends, the high CPU load on the data servers, and contention for bandwidth. The results demonstrate variations in the I/O performance during regular operation and different execution periods because of software updates and sustained I/O-intensive workloads. They also show that contention for resources generates transient I/O performance prob-

lems. TOKIO is a software framework designed to encapsulate the mechanics of various I/O monitoring and characterization tools used in HPC. TOKIO relies on Darshan logs to collect information from the compute nodes, bringing the previously mentioned drawbacks and not monitoring how the compute nodes used the PFS. The usage data from the storage servers come from the Lustre Monitoring Tool (LMT) (WARTENS; GARLICK, 2010), which might not be supported by the system vendor of the supercomputer (as was the case with SDumont). Also, despite providing user-perceived performance over time, active probing on the shared storage might insert some unwanted interference on the overall PFS usage.

Patel et al. (2019) proposed a tool to analyze the log data of Lustre PFS obtained with LMT in one year (2018) of operation 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). They also demonstrate that the OST usage is unbalanced regarding I/O activity. On such a large-scale platform, applications tend not to take advantage of I/O parallelism, often using few concurrent OSTs. LMT monitors Lustre File System servers (MDT, OST, and LNET routers), collecting data using the Cerebro monitoring system and storing it in a MySQL or MariaDB database. Aside from not being supported on SDumont, LMT only provides Lustre server-side activity, lacking information on which and how applications used the PFS.

Sivalingam et al. (2019) proposed another approach. The authors used LASSi, a tool to analyze application usage and contention caused by the use of shared resources (file system or network) on the Lustre File System deployed at the ARCHER supercomputer, deployed at the UK National Supercomputing Service. LASSi combines Lustre statistics and job information to calculate derived metrics, identifying a particular class of jobs that generated excessive I/O load. They identified that over 50% of the use in the analysis period was for jobs that read less than 4 GiB and wrote less than 32 GiB. Overall, twice as much data was written on ARCHER than it was read. LASSi is an approach similar to the one presented in this dissertation. However, it relies on a MySQL database to store all the collected information, which might impose limitations on its replicability. As an example, the installation and utilization of a MySQL server on the administrative infrastructure of SDumont is not supported by the system vendor nor allowed by the System Administrators. Despite LASSi collecting metrics from the Lustre's servers, it does not assess the load imbalance among the storage devices nor characterize the metadata uti-

lization of the file system, two aspects we tackle in our methodology. Another difference between our methodology and LASSi is that we provide an application characterization regarding its transfer size, allowing the identification of problematic behavior. In contrast, LASSi does not provide such distinction.

Betke and Kunkel (2019) 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, lacking the maturity to be employed in a production system.

Those methodologies tend to rely on the periodic execution of probes (usually I/O benchmarks that ran on the shared file system), application-level profiling (Darshan), or tools that require administrative privileges (LMT) and might not be supported by storage vendors. Also, they often do not collect information from compute nodes to monitor system health and track performance. This dissertation proposes a broader methodology to provide a bigger picture of the whole system's I/O utilization, allowing continuous analysis from the Storage Devices to the Compute Nodes and tracking inefficient behavior. Our methodology intents to track the storage servers' utilization, assessing causes of possible performance drawbacks, such as load imbalance. Monitoring the CNs allows for investigating which applications were executing on the supercomputer, how they utilized the PFS, and their I/O demands. We adopted the use of 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. (2014) evaluates the performance of Lustre's metadata server (MDS) concerning multi-core and multi-socket platforms. 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. (2015) 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) regarding the

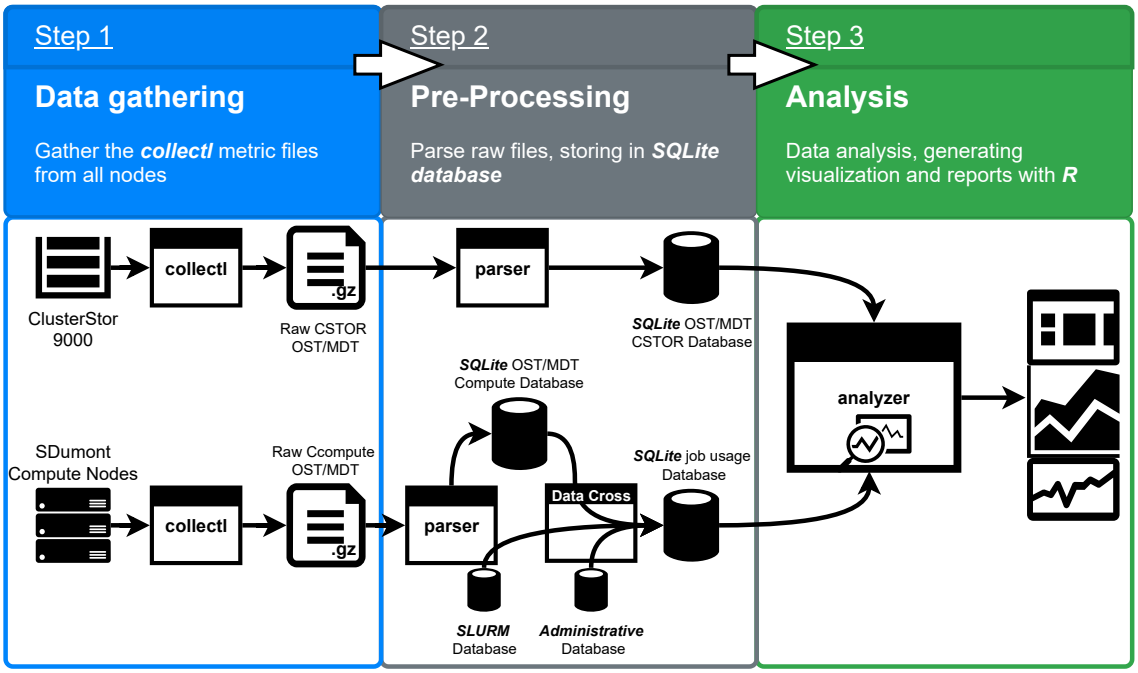
metadata performance. They demonstrate that a distributed management approach brings orders of magnitude improvements over the performance of centralized deployments. The distributed metadata presented almost linear scalability (up to 512 nodes), justifying its employment for intensive metadata accesses.

Kunkel and Markomanolis (2018) 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, the present work is the first to integrate I/O and metadata analysis in the same approach while characterizing their needs in an HPC system.

4 ANALYSIS AND VISUALIZATION METHODOLOGY

This chapter presents the methodology developed in this work and how it was implemented in the SDumont environment. To study the Lustre PFS utilization in supercomputers we propose a pipelined workflow, comprised of three steps, as depicted in Figure 4.1: (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 4.1 – Data collection and analysis workflow.



Source: Author

4.1 Data Gathering Step

We selected the *collectl* [1] tool to perform the utilization metrics gathering. It is an open-source system performance monitoring tool that collects metrics from various subsystems, such as CPU, disk, inodes, memory, and network, public available by Linux's kernel modules on the `/proc` and `/sys` pseudo-filesystems. *collectl* is made of a collection of Perl scripts. Its advantages 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 local compressed file or sending them over the network, and has a flexible API that enables the development of custom modules (plugins). *collectl-lustre* [2] is a special

[1] http://collectl.sourceforge.net
[2] https://github.com/pcpiela/collectl-lustre

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 reads and writes operations and the volume of data transferred in KiB. Additional metrics are calculated in steps 2 and 3 of the workflow. All the metrics used are summarized in Table 4.1.

Table 4.1 – Lustre I/O metrics.

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

Source: Author

The Lustre's metadata kernel module exposes counters common to the MDS and compute node clients and other counters exclusive to each of them. For this reason, *collectl-lustre* retrieves information about `fopen`, `fclose`, `getattr`, `setattr`, and `sync` counters on both the MDS and clients. Among the counters exclusive to the MDS are `unlink` and `statfs`. The `seek` counter is available only on clients. Table 4.2 summarizes the metadata counters of the MDS and client nodes.

The *collectl* tool version 4.3.1, with *collectl-lustre*, was installed on all ClusterStor Lustre nodes (MDS and OSSs) and on the 758 Compute Nodes. The execution of *collectl* was configured to collect metrics in 15 seconds intervals and store it on a daily compressed file at the local `/tmp` file system to avoid interference with Lustre. The overhead imposed by *collectl* on the monitored nodes was neglectable ($< 0.1\%$ 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 example, the pre-processing of one-day data collected with 15 seconds intervals from the compute nodes took around 2.5 hours and generated a database with 7 GiB size. Our experiments with intervals of 1, 5, and 10 seconds increased those values by up to $4\times$ but did not bring an increase in data representativeness that justified the excess in resource consumption. For this reason, we collected the Lustre utilization metrics every 15 seconds. Despite

Table 4.2 – Lustre Metadata Counters.

Counter	Node	Description
`getattr`	MDS	Operation that get file/dir attributes.
`setattr`		Operation that set file/dir attributes.
`getxattr`		Operation that get file/dir extended attributes.
`setxattr`		Operation that set file/dir extended attributes.
`unlink`		File/dir removals.
`statfs`		Operation that return statistics about the file system.
`sync`		Operation that synchronizes data to the file system.
`fopen`		File open requests.
`fclose`		File close requests.
`link`		Hard or symbolic link creation.
`mkdir`		Directory creation requests.
`rmdir`		Directory removal requests.
`fopen`	Client	File open requests.
`fclose`		File close requests.
`getattr`		Operation that get file/dir attributes.
`setattr`		Operation that set file/dir attributes.
`seek`		Operation that change the file pointer.
`fsync`		Operation that synchronizes data to the file system.

Source: Author

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 **Pre-Processing** step consists of a collection of scripts developed in Python [3] v3.8, designed to collect the *collectl* raw files (generated at Step 1) from each node daily and perform parsing, cleaning, and inserting the data into an *SQLite*[4] 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). It comes pre-installed on most Linux distributions (even if it is not already installed, SQLite can be easily deployed and executed without administrative privileges since it is just a single executable). Besides, it has APIs for various data analysis tools and languages. At this point, there are two distinct datasets, one from the data collected

[3] https://www.python.org/
[4] https://www.sqlite.org/index.html

at the *ClusterStor Nodes* and one from the SDumont *Compute Nodes*.

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*[5] 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*. The structure of the datasets generated is depicted by Figure 4.2, where, aside from the metrics presented by the Tables 4.1 and 4.2, `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.

Figure 4.2 – Datasets Structure.

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)

Source: Author

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

During the **Pre-Processing** step, two complementary I/O metrics are generated: (i) the average transfer size in KiB and (ii) the **Q**uality of **O**peration (*QO*), as proposed by Sivalingam et al. (2019). The latter, given by the Equation 4.1, is based on the default 1 MiB `stripe_size` of the Lustre file system on SDumont, making 1 MiB the optimal size for read or write per operation. Based on *QO*, the operation would make an optimal usage of Lustre when 1 MiB is read or written. Hence, a value of "1" represents optimal usage, and values in a **higher** order of magnitude represent that the transfer size is smaller than `stripe_size`, leading to poor quality because of misaligned access. The operation is also considered inefficient if the *QO* metric present values lower than "1" because of misalignment, with a transfer size bigger than `stripe_size`. However, lower *QO* (< 1) is preferable to higher *QO* because each operation transfers more data, leading to higher throughput.

$$Operation\_Type_{qo} = \frac{Number\_Of\_Operations * 1024}{KiB\_Transferred} \tag{4.1}$$

4.3 Data Analysis Step

During this step, additional I/O metrics are calculated: **B**andwidth **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} (Equation 4.2), as proposed by Lockwood et al. (2018a), 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 a CF_{bw} of 0.60 indicates that other competing jobs consumed 40% of the available resources.

$$CF_{bw}(job) = \frac{N_{bytes}(job)}{N_{bytes}(Lustre)} \tag{4.2}$$

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 the others. In contrast, a *"low"* σ represents a better load distribution. Since the σ is quantified as high or low regarding the mean (μ) load observed on the OSTs at each timestamp, the LI metric (Equation 4.3) is defined

as a **Coefficient of Variation** [6] 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 regarded as a severe imbalance.

$$LI = \frac{\sigma}{\mu} \tag{4.3}$$

The SMA (HANSUN, 2013), a time series analysis technique commonly used in the financial market, is an arithmetic mean of a variable within a specific time frame (*tf*) and moves through a time series. For a metric *m* at timestamp *t*, the SMA_{tf} of *m* is given by Equation 4.4:

$$SMA_{tf}(m) = \frac{1}{tf} \sum_{i=t-tf}^{t} m_i \tag{4.4}$$

The financial market data have a high variation through 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 tested various time frame intervals, and the value of 3 hours was selected, 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 analyses with a new dataset from SDumont or other HPC platforms can be daunting. We developed a *web application* using R+Shiny [7] to streamline the analysis process. 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 minimum 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.

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

[7] https://shiny.rstudio.com/

5 GLANCING AT THE LUSTRE FILE SYSTEM

This chapter presents the evaluation of the Lustre PFS deployed on SDumont using the proposed methodology. We collected the usage metrics for three months (March, April, and May) of 2020 and 2021 to study the utilization of the Lustre PFS and compare the findings between years. We divided the analysis of the PFS into two parts. First, the whole period (three months) is analyzed using the *ClusterStor* dataset (Section 5.1). Second, a specific period of interest was analyzed 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

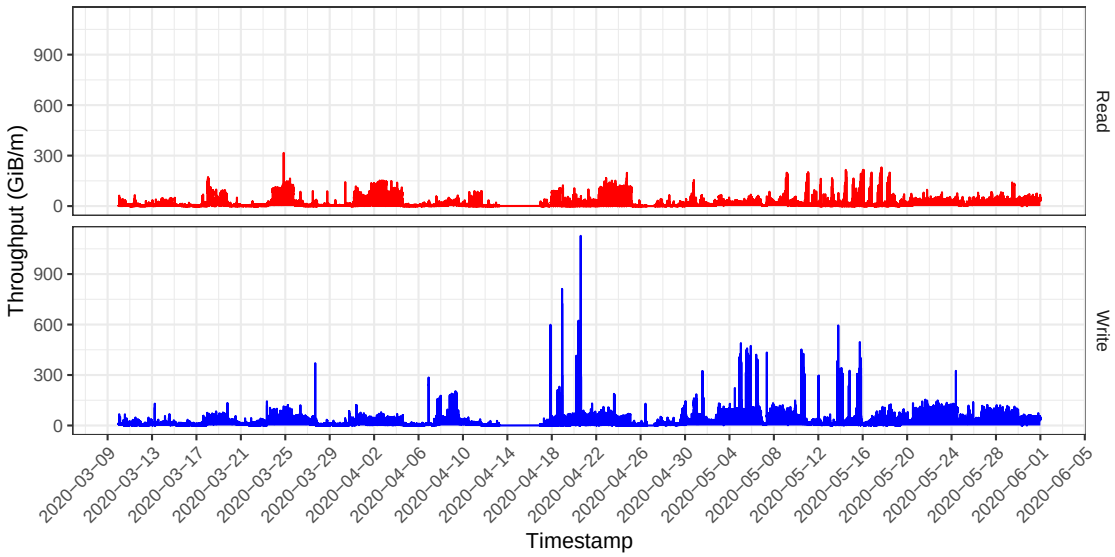
This section provides an overview of the Lustre PFS utilization, analyzing the operation data obtained during March, April, and May from the years 2020 and 2021. Section 5.1.1 focuses on the I/O operations performed on the system with data collected from the ClusterStor's OSSs and OSTs. Section 5.1.2 presents the evaluation of the Metadata operations with data from the MDS.

5.1.1 I/O Data Analysis

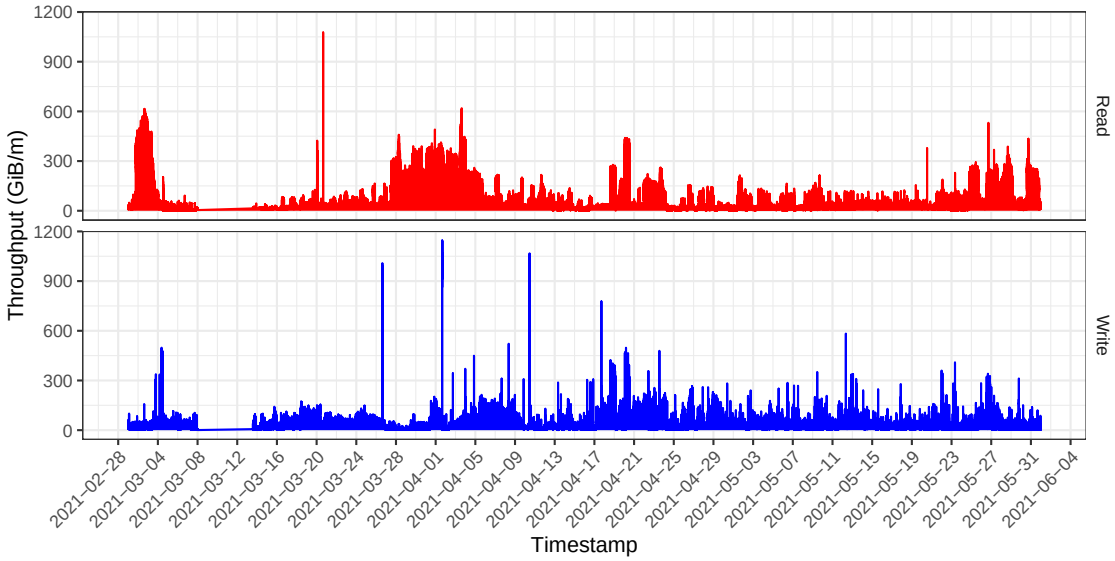
We investigated the aggregated throughput of the storage system during the aforementioned period. Figure 5.1 illustrates the results in GiB/m for 2020 and 2021. Read (red, upper facet) and write (blue, lower facet) operations are stacked together based on when the metric was collected. In the two years, there were scheduled maintenances 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.

By comparing both years was possible to notice an increase in utilization. In terms of absolute values, in 2020, there were 1.8 PiB data read and 2.9 PiB written, and the system received 64.154 billion read requests and 1.234 billion writes. Compared with 2021, the total volume of data read presented an increase of $4.7\times$ (7.95 PiB), and the

Figure 5.1 – Data distribution for 3 months. The y-axis is the throughput in GiB by minute. The figures have different scales.



(a) 2020



(b) 2021

Source: Author

data written increased $1.5\times$ (4.1 PiB). The number of reading operations decreased $1.6\times$ (39.102 billion), and the write operations increased $4.3\times$ (5.297 billion). The increase in the 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 were $36,884$ jobs, and in 2021 $145,793$, representing $4\times$ more jobs. The increase in reading 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 (LANG et

al., 2009; ISAKOV et al., 2020).

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 writing operations in both years. The maximum achieved in 2020 was $1,127$ GiB/m (at 2020-04-20T14:30:00) and in 2021 was $1,145$ GiB/m (2021-04-01T18:50:00), which represents $\approx 41.74\%$ and $\approx 42.41\%$, respectively, of the maximum bandwidth of the ClusterStor[1]. The average write throughput in 2020 was 25.336 GiB/m and in 2021 was 34.452 GiB/m ($1.3\times$ increase). The read operations presented lower values in 2020 but significantly increased in 2021. The peak read throughput in 2020 was 316 GiB/m (at 2020-03-24T20:36:00) and in 2021 was $1,077$ GiB/m (at 2021-03-20T18:50:00), accounting for $\approx 11.7\%$ and $\approx 39.89\%$, respectively, of the max bandwidth. The average read throughput in 2020 was 15.825 GiB/m and in 2021 was 66.953 GiB/m ($4.2\times$ increase). If we focus on the peak throughput registered at each OST, the write occurred simultaneously as a single event on all OSTs, 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 the period, with none of the OSTs' peaks coinciding. This behavior indicates that the applications write data more coordinatedly than they read.

Table 5.1 – Transfer 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

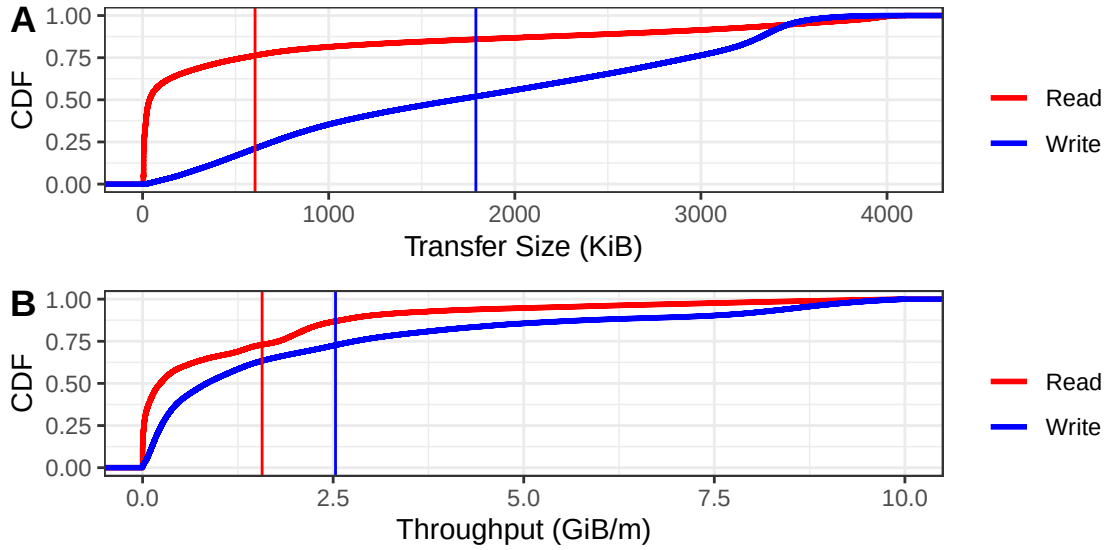
Source: Author

Table 5.1 depicts the distribution of the Transfer Size and Quality of Operation 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 transfer 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 transfer sizes. In 2021, the

[1]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$\times 60$ s).

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. For 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 while increasing the amount of data transferred. The writing operations slightly declined (lower quality and smaller transfer sizes). Another important observation in Table 5.1 is that in 2021, the performance gap between reading and writing operations narrowed. In 2020 the average write transfer size was $\approx 3\times$ bigger than the read size (1729 KiB and 651 KiB respectively), while in 2021, the write size was only $\approx 1.4\times$ bigger than the read (1488 KiB and 1018 KiB). This difference in request sizes explains why we observed higher throughput on writing operations.

Figure 5.2 – 2020 *CDF* of the Transfer Size (**A**) and Throughput (**B**) for the Read (Red) and Write (Blue) operations among *OSTs*. The vertical lines represent the mean observed values: 604 KiB Read and 1791 KiB Write for Size, and 1.568 GiB/m Read and 2.531 GiB/m Write for Throughput.



Source: Author

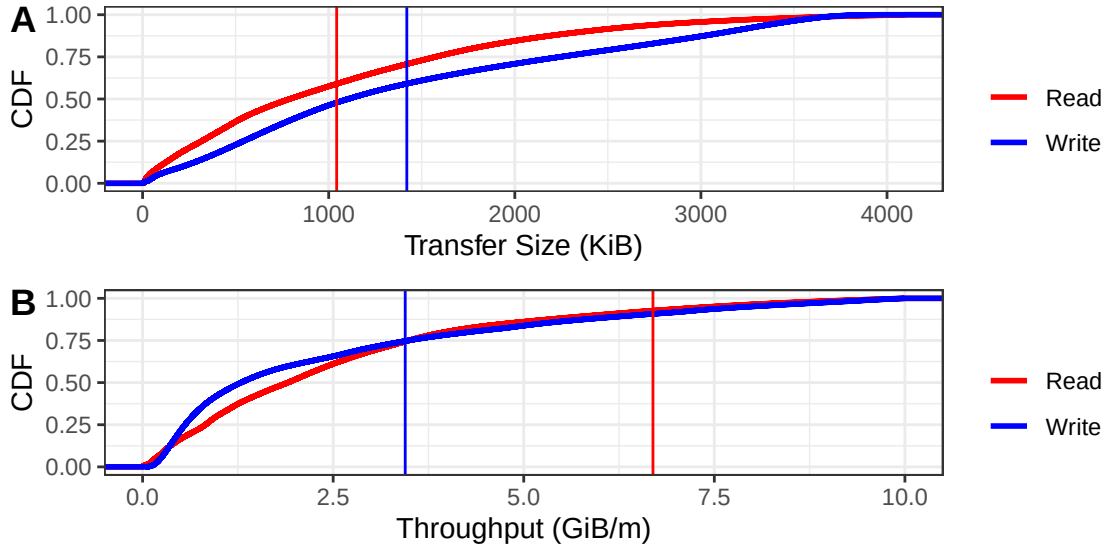
The CDF analysis of the throughput and transfer size observed among the OSTs demonstrates how the difference in performance between reading and writing operations narrowed from 2020 to 2021. In 2020, as Figure 5.2A shows, the average write transfer size was $\approx 3\times$ bigger than the read size (1791 KiB and 604 KiB respectively). Also, the read size presented a lower variance because its overall slope is steeper, with 75% of the read samples having sizes below the mean, while the rest have much bigger transfer sizes. Steeper slopes indicate a tighter range of values and lower variability. The write

size presented values slightly evenly spread out between the observed spectrum. It is also possible to notice through Figure 5.2A that the read size is shifted left the furthest towards lower values, indicating that it usually is smaller than write sizes. The average write throughput observed among the OSTs was $\approx 1.6\times$ higher than the read (2.531 GiB/m and 1.568 GiB/m respectively), as depicted by Figure 5.2B, which also shows that almost 75% of the activities have data transfer rates less than the average values.

In 2021, the average transfer size for reading increased $\approx 1.7\times$, reaching 1043 KiB, while the size for writing presented a slight decrease (from 1791 to 1420 KiB). The read operation showed $\approx 62\%$ of occurrences below average the average value, and $\approx 55\%$ of the size of write operations were below the average value. This behavior in the transfer size is reflected in the average throughput, which increased $\approx 4.2\times$ for reading and $\approx 1.3\times$ for writing from 2020 to 2021. Differently from 2020, in 2021, the reading throughput now has a higher average value than writing ($\approx 2\times$). The throughput of reading and writing operations from the 2021 data presented values below the mean for 75% of the operation time, as observed in 2020.

In 2020, the difference between the transfer size of reading and writing operations was 1187 KiB, while in 2021, it narrowed to 377 KiB, which contributed to the observed increase in reading throughput.

Figure 5.3 – 2021 *CDF* of the Transfer Size (**A**) and Throughput (**B**) for the Read (Red) and Write (Blue) operations among *OSTs*. The vertical lines represent the mean observed values: 1043 KiB Read and 1420 KiB Write for Size, and 6.695 GiB/m Read and 3.445 GiB/m Write for Throughput.



Source: Author

We computed the correlation coefficient between the quality of operation and the

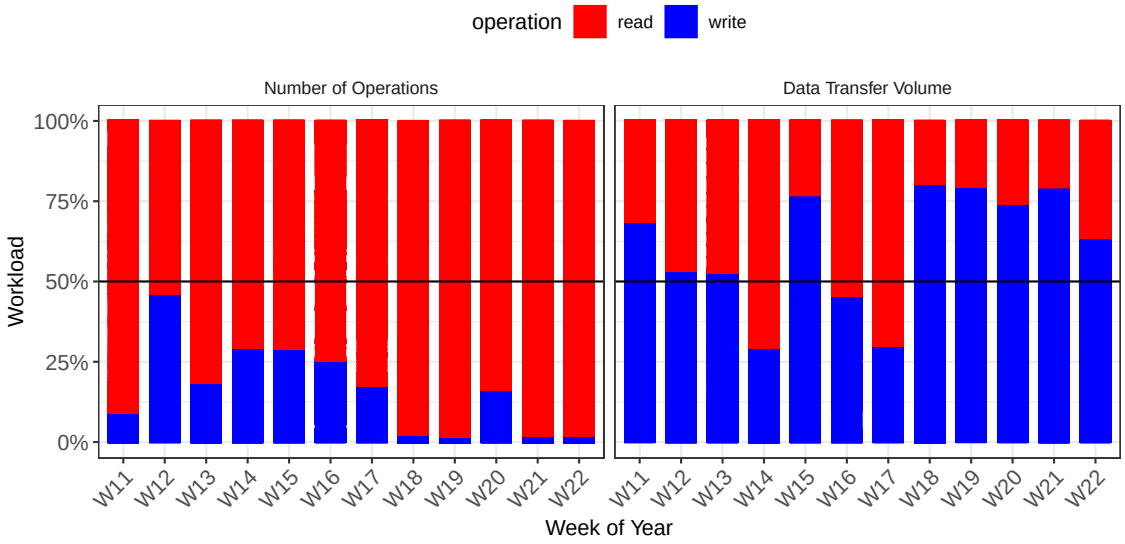
amount of transferred data using Spearman's (SPEARMAN, 1904) correlation test, as the Kolmogorov-Smirnov (DODGE, 2008) test could not conclude that all these results follow a normal distribution. The statistical test has indicated a strong inverse correlation between the quality of operation and the transfer size ($\rho = -0.87$). Since the optimal value for the quality operation is 1 (which indicates using operations with 1 MiB blocks), a greater amount of data is read or written in a single timestamp, leading to more efficient use of the resources. As for the operation's size compared to the transfer volume, there is a strong positive correlation ($\rho = 0.94$). As expected, the bigger the size of the operation, the more data is transferred, achieving better performance.

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, 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 data transferred and number of operations, accounting for 66% of the throughput and 88% for the number of operations.

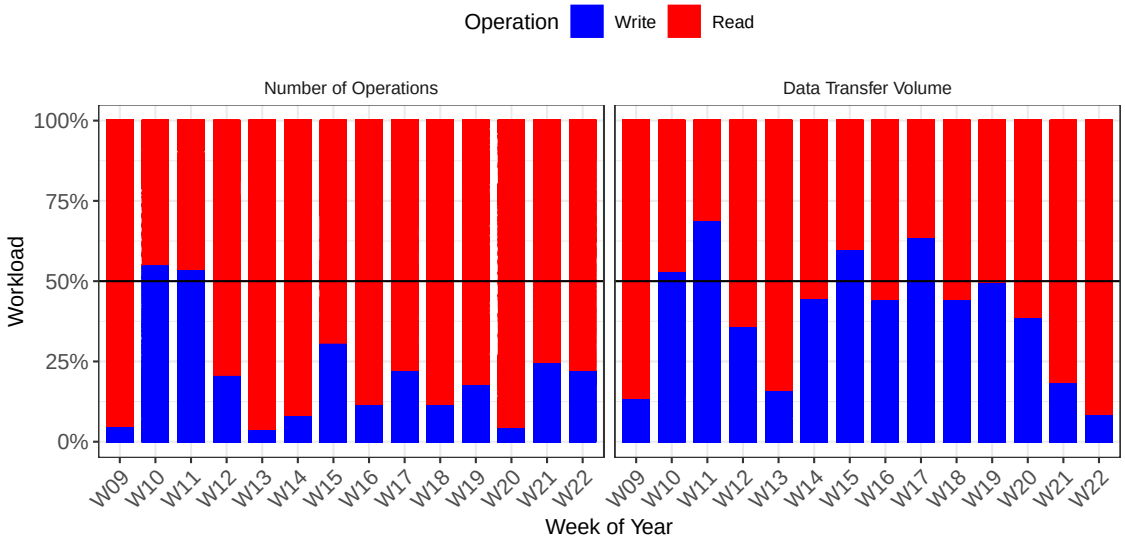
The Figure 5.4a depicts the workload distribution between reads and writes grouped by week and split by the number of operations (left) and the volume of data transferred (right). Checkpoints do not always dominate the I/O workload of HPC systems. Scientists are reading large volumes of data into HPC systems as part of their science (LOFSTEAD et al., 2011; LATHAM et al., 2014). As new applications from Big Data and Machine Learning (ML) enter the HPC system, the ratio between reading and writing-bound workloads varies based on the application set running on the platform. During the three-month observation of the 2020 period, writing operations dominated nine of the twelve weeks of data transferred. Nonetheless, this trend shifts towards reading-intensive applications for three weeks (W14, W16, and W17). Regarding the number of operations, the reading operations dominated the workload across all weeks, fairly balanced on W12. This behavior demonstrates 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, Figure 5.4b denotes that in 2021 the volume of data transferred was dominated by reads, with writes dominating only four out of fourteen weeks. The number of reading operations continues to dominate most weeks, except for weeks W10 and

Figure 5.4 – Workload distribution by week. The x-axis represents the week of the year.



(a) 2020



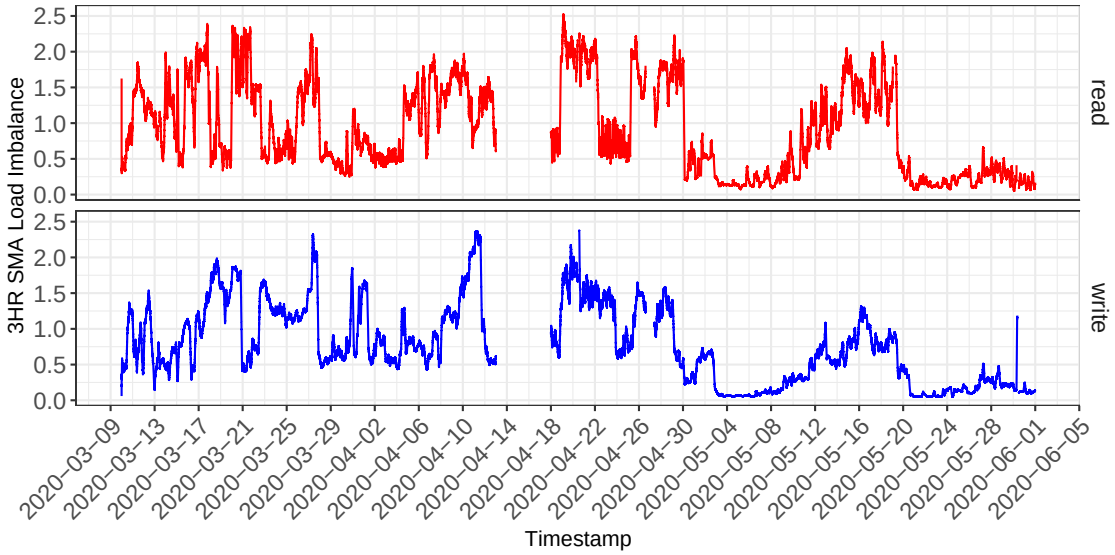
(b) 2021

Source: Author

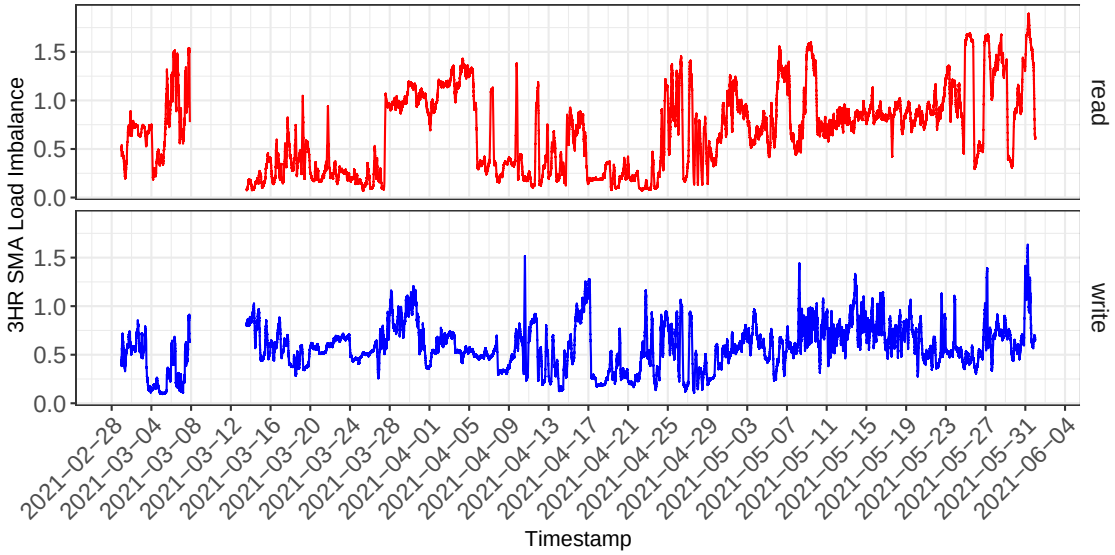
W11, which were dominated by writing in both the number of operations and the volume of data transferred. The behavior of a greater number of reading operations of small size still occurs and can be observed mainly on weeks W15 and W17, where the readings dominated the number of operations. Still, the 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 subsystems.

The analysis of the OSTs' load distribution indicates that a considerable imbal-

Figure 5.5 – SMA_{3HR} of LI for the read (red) and write (blue) workload. 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.



(a) 2020



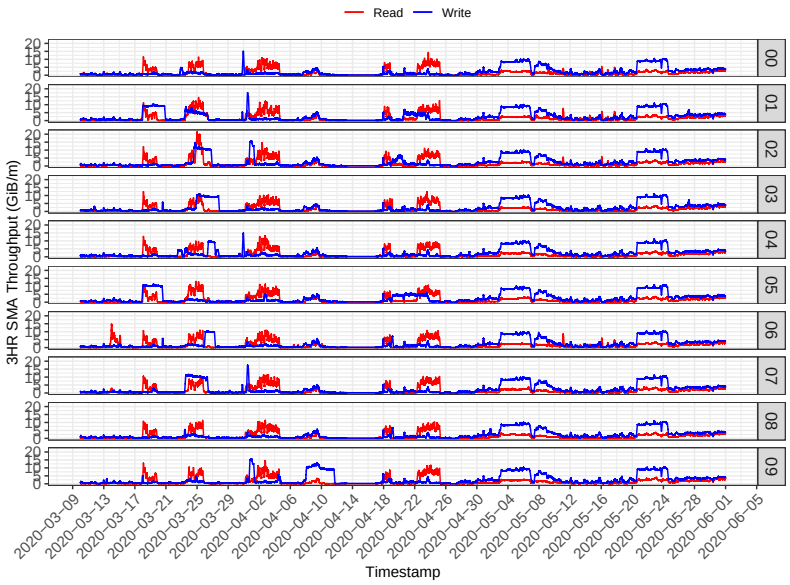
(b) 2021

Source: Author

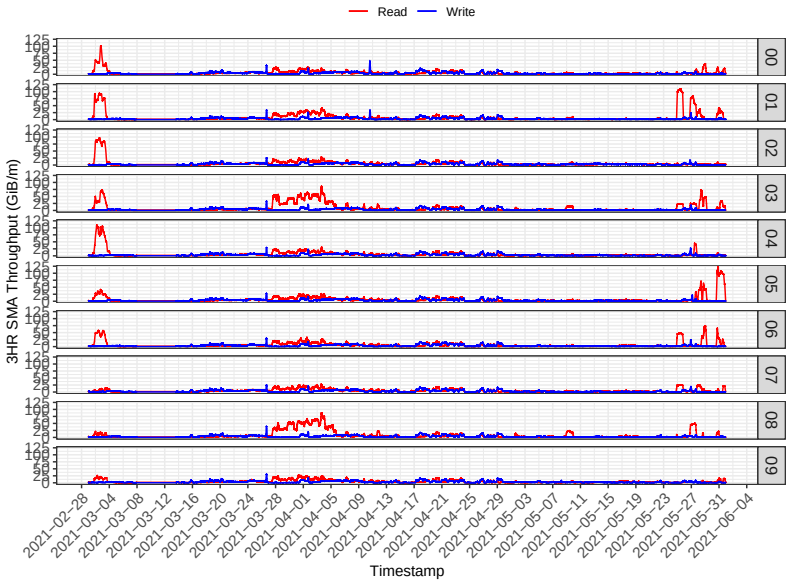
ance. Figure 5.5a depicts the SMA_{3HR} (Equation 4.4) of the reading and writing LI metric, showing that for 2020, both operations follow similar trends regarding the load imbalance. Despite ClusterStor presenting 0.6 LI for 50% of the time, there are some severe cases where the overload represented up to 300% of the OSTs' average load. An interesting imbalance case happened on April 10th, 2020, 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 imbalance between reading and writing was similar, with at least 25% of the time operating with LI above 1. Figure 5.5b 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.

Figure 5.6 – SMA_{3HR} of read and write throughput by OST.



(a) 2020



(b) 2021

Source: Author

Lustre's default load balancing focuses on OST's available storage space, whereas 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 OSSs and, allied with the default

striping policy on SDumont (`stripe_count` $= 1$ and `stripe_count` $= 1$ MiB), 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 rather invasive approach system administrators might take is to adopt a dynamic load balancer that automatically coordinates the workload and data placement among I/O servers (NEUWIRTH et al., 2017; WADHWA et al., 2019).

The analysis of the SMA_{3HR} of reading and write throughput from 2020, broken down by OST, indicates that the spike in the imbalance on April 10th was due to OST 09 receiving a more significant write workload than the others (Figure 5.6a). The increase in LI of the reads between March 28th and April 5th, 2021 (also observed as an increase of reading on Figure 5.1) is due to a load concentration on three OSTs 01, 03, and 08 (Figure 5.6b). With the SMA, we can observe the general trend, identifying regions of interest where usage peaks occur and periods when a specific OST receives more load than the others.

5.1.2 Metadata Analysis

This subsection presents the analysis of the metadata operations performed on the Lustre PFS for only the 2021 dataset. It was impossible to collect all metrics during 2020 because of a bug in 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 5.2 summarizes the number of executions of each metadata operation during the 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 the 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 since it is necessary to configure the default access mode every time a file is created. 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.

Table 5.2 – Amount of Metadata Operations

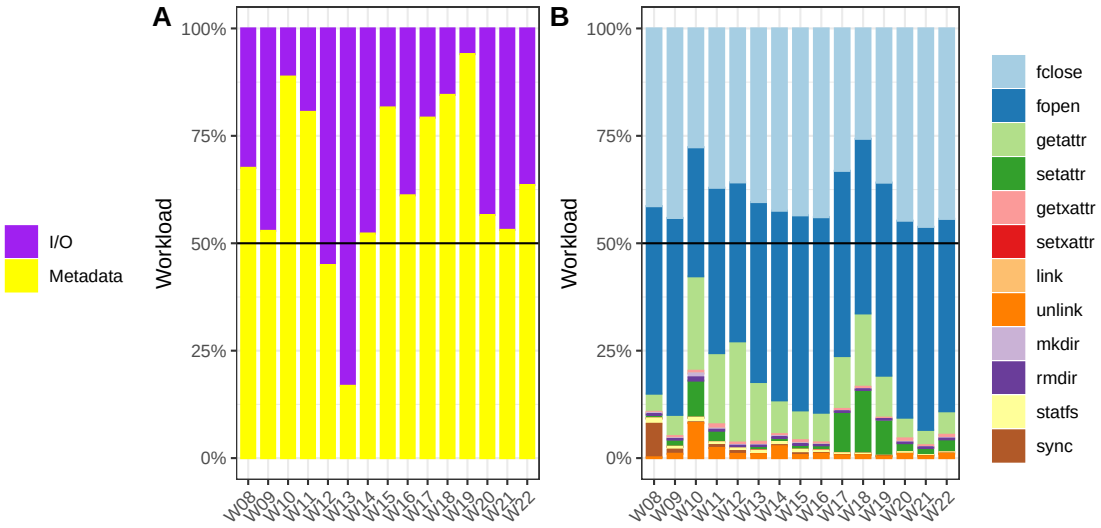
Operation	Total	Min ops/s	Avg. ops/s	Max. ops/s
`fopen`	28,812,381,450	1	3,859	102,291
`fclose`	25,369,943,340	1	3,398	102,132
`getattr`	6,733,374,960	1	902	32,698
`setattr`	3,451,979,850	1	462	8,406
`unlink`	593,117,055	1	87	2,357
`getxattr`	345,187,575	1	47	7,833
`statfs`	280,998,450	1	38	62
`sync`	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

Source: Author

Figure 5.7A depicts the load distribution between I/O and Metadata operations. It is possible to notice that the Metadata operations dominate most weeks, except for 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 67 billion requests against 44 billion I/O requests. In Figure 5.7B 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 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 indicated in Section 5.1.1). The metadata workload characterized on SDumont would not be desirable on a scratch Lustre PFS designated for high throughput, where it is desirable that the metadata operations be kept at a minimum.

The metadata operations are becoming an increasingly hot development topic and a genuine concern on high-performance storage. It is possible to observe this trend on the latest release of the IO500 (KUNKEL et al., 2021), where the most significant growth in the max score is attributed to metadata while the I/O throughput presents little to no

Figure 5.7 – 2021 metadata load distribution by week. (A) depicts the load of I/O operations (purple) and and metadata operations (yellow). (B) details the metadata operation type.



Source: Author

increase over the years. Among the improvements that boost metadata performance are the use of multiple servers (RODRíGUEZ-QUINTANA et al., 2016), indexing mechanisms (PAUL et al., 2020), and client-side pre-allocation (LI et al., 2019). There are features (FRAGALLA; LOEWE; PETERSEN, 2020) specific to improve Lustre's metadata performance that system administrators should assess, such as Distributed Namespace (DNE – use of multiple metadata servers to distribute the load) and Data on MDT (DoM – store small files on the MDT instead of placing all data on OSTs as is normally done in Lustre, significantly reducing the number of requests and accesses to OSTs). SDumont implements none of those improvements.

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. We made a more in-depth analysis using the *Job Usage* dataset with two periods, one from each year because we wanted to verify which applications contributed to specific events. We selected the period between March 24th and March 28th of 2020 because it comprises the peak throughput for the read operations (Figure 5.1a) of that year, and for 2021, between March 28th and April 1st because of the striking increase in reading volume (Figure 5.1b). Crossing the information from

the compute node dataset and SLURM's job list is 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

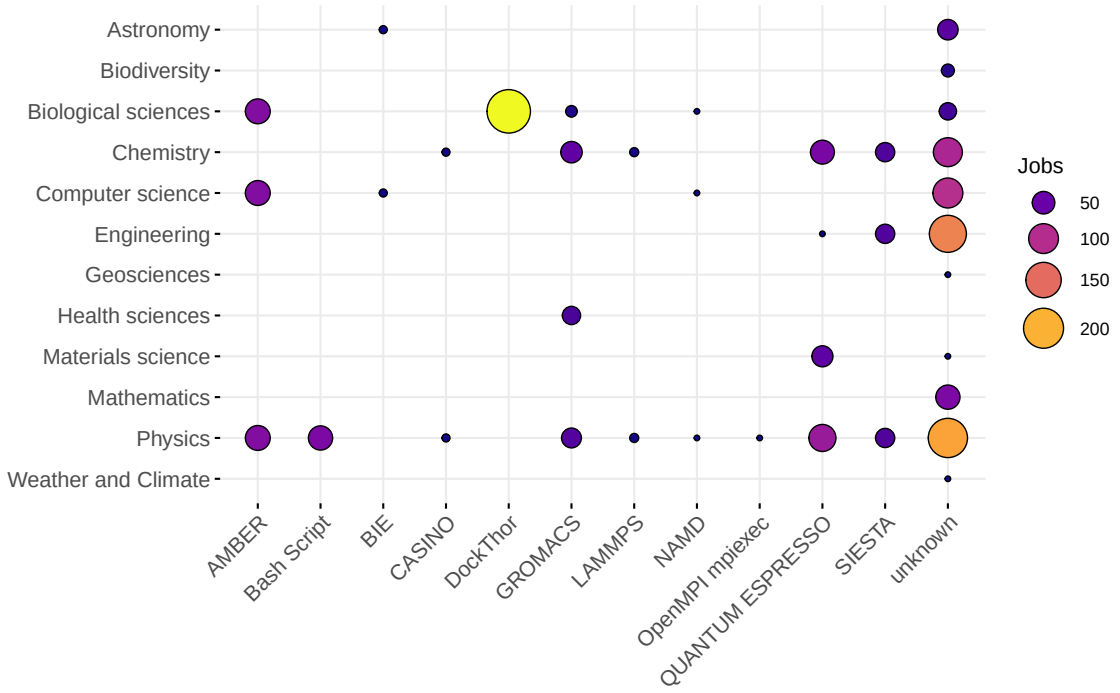
The analysis presented in this subsection used the information collected at the compute nodes to characterize the applications' I/O utilization. During the selected period of 2020, a total of 866 jobs have been executed on SDumont. With the SLURM's information, we were able to identify nine different applications: *AMBER* [2], *BIE* [3], *CASINO* [4], *DockThor* [5], *GROMACS* [6], *LAMMPS* [7], *NAMD* [8], *QUANTUM ESPRESSO* [9], and *SIESTA* [10]. It was not possible to identify some job's applications solely based on 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 reported by SLURM 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 system administrators' database of known applications).

Figure 5.8 details the applications and which "Science Domain" used them. It is interesting to notice that an application can be employed in different areas, such as *GROMACS*, used by projects in *Chemistry*, *Physics*, *Biological Sciences*, and *Health Sciences*. This fact contributes to applications having different I/O volumes, demands, and behaviors (BEZ et al., 2019). 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 depicted in Figure 5.9: *AMBER*, *BIE*, *DockThor*, *GROMACS*, *LAMMPS*,

[2] https://ambermd.org/
[3] https://people.astro.umass.edu/ weinberg/BIE/
[4] https://vallico.net/casinoqmc/
[5] https://www.dockthor.lncc.br/
[6] https://www.gromacs.org/
[7] https://www.lammps.org/
[8] https://www.ks.uiuc.edu/Research/namd/
[9] https://www.quantum-espresso.org/
[10] https://departments.icmab.es/leem/siesta/

LHCB DIRAC [11], *ORCA* [12], *Python* [13], *QUANTUM ESPRESSO*, *SIESTA*, and *VASP* [14]. 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%).

Figure 5.8 – Identified applications (2020), their Science Domains, and number of jobs

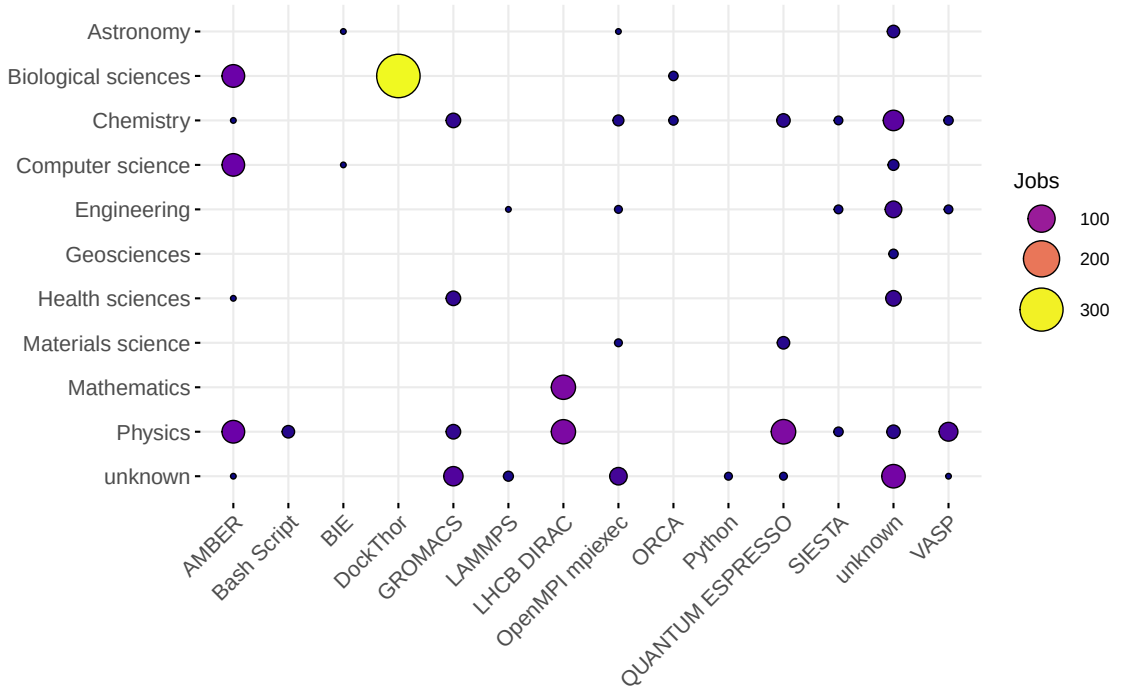


Source: Author

Different types of optimizations can be employed on the storage system to improve performance. I/O optimizations can be applied to applications at each layer of the stack (BEHZAD et al., 2019), and automatic dynamic scheduler (GAINARU et al., 2015; DORIER et al., 2014) can schedule applications with the same I/O patterns to alleviate congestion. One thing in common that these optimizations techniques need is each application's I/O access patterns characteristics. The same application can have different I/O behavior (BEZ et al., 2019) depending upon which science domain used it—in that way, having the information about the science domain that used the application and the access pattern might help I/O libraries, schedulers, and other techniques to obtain a better picture and comprehension when applying the optimizations.

[11] https://lhcb-dirac.readthedocs.io/en/latest/
[12] https://orcaforum.kofo.mpg.de/index.php
[13] https://www.python.org/
[14] https://www.vasp.at/

Figure 5.9 – Identified applications (2021), their Science Domains, and number of jobs



Source: Author

Aside from the maximum throughput achieved for reads in 2020 (presented in Section 5.1 – 316 GiB/m), the peak for writes in this specific period was 370 GiB/m (at 2020-03-27). Table 5.3 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}, and (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 more significant number of applications (the decrease in the CF_{bw} value indicates an increase in the contention for resources). The *QUANTUM ESPRESSO* application, which presented the highest throughput, can be categorized as a traditional HPC Scientific Application, highly parallelized and optimized (GIANNOZZI et al., 2020).

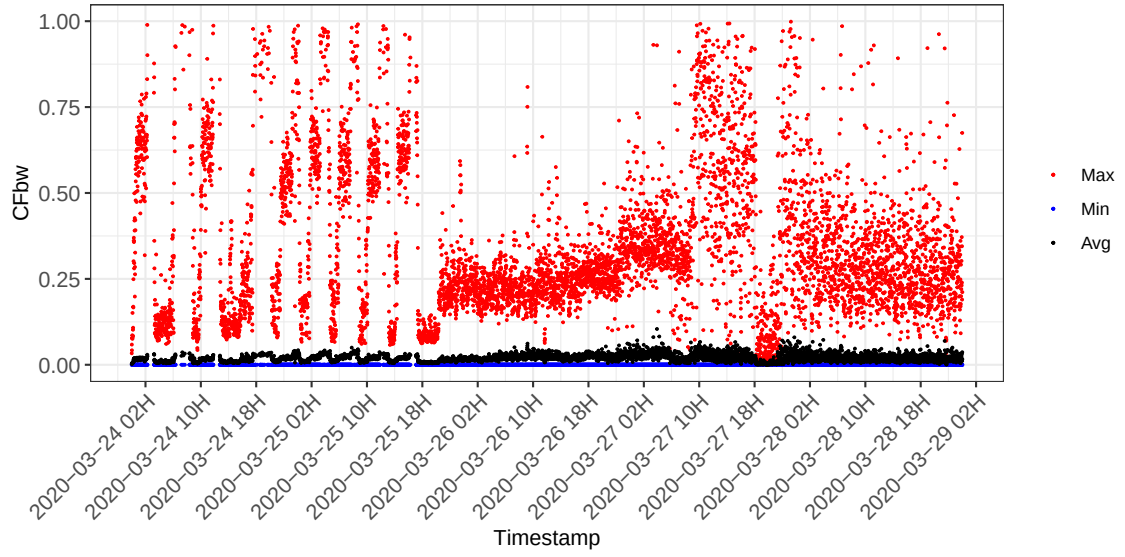
Table 5.3 – Individual application's peak throughput

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

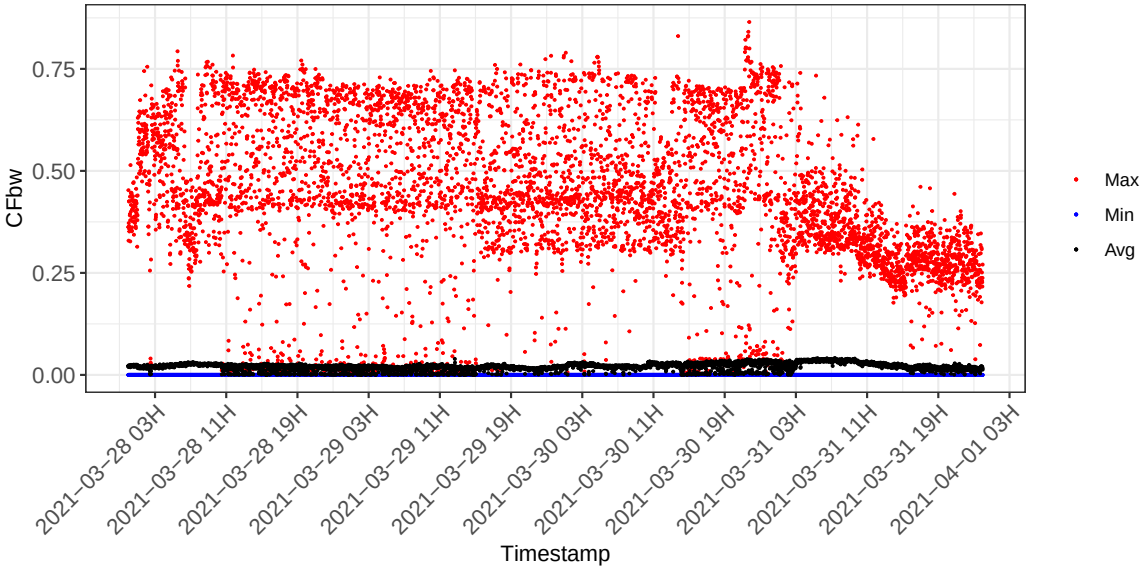
Figure 5.10a depicts how the bandwidth of the whole Lustre file system was used,

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. As depicted by Figure 5.10b, in 2021, a few jobs also dominated the bandwidth, but the utilization is more constant, as opposed to the bursty behavior observed in 2020. It is possible to observe high-bandwidth jobs starting on March 31st, with the decrease in the Max. CF_{bw} and a slight increase in the average values.

Figure 5.10 – 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.



(a) 2020

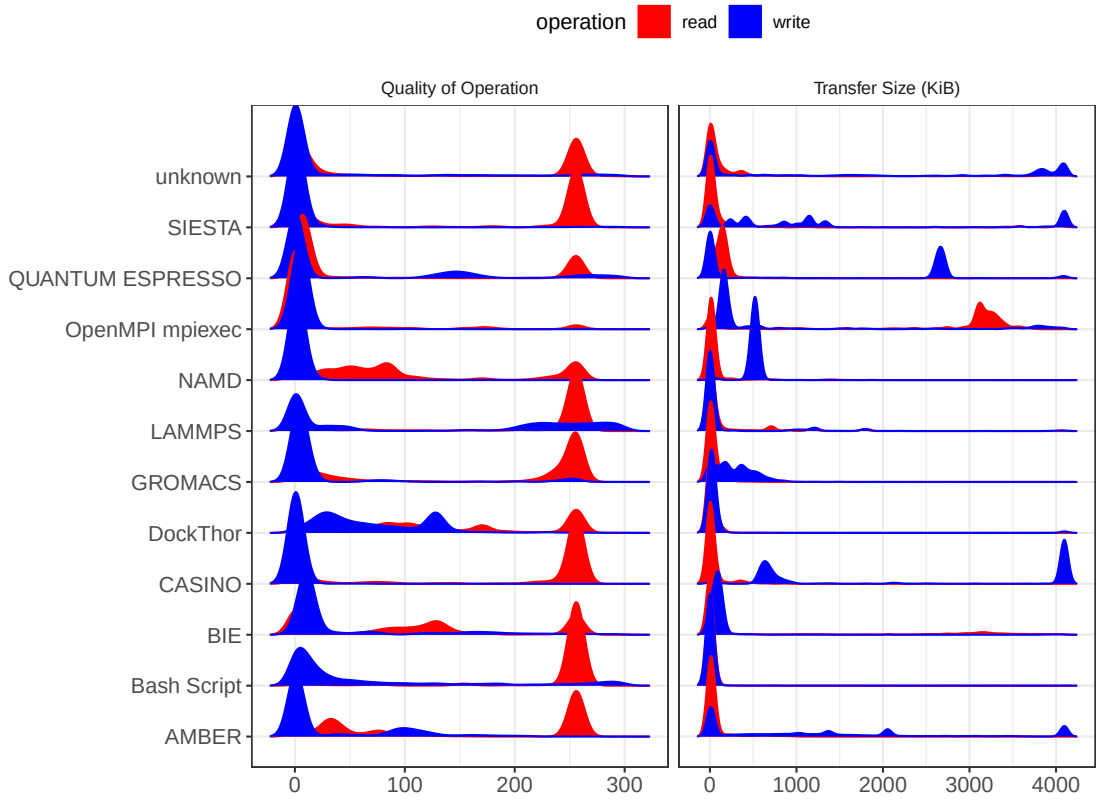


(b) 2021

Source: Author

Figure 5.11 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 most of the time. In 2020, the most notably read-inefficient applications were *DockThor*, *BIE*, and *Bash Script*. Only the *OpenMPI mpiexec* category presented $read_{qo}$ under 1 for 75% of its execution, meaning 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. The other Domains follow the inefficient usage behavior.

Figure 5.11 – 2020 Distribution of the Quality of Operation (left) and Transfer Size (right). The x-axis are the QO index and size in KiB, respectively.

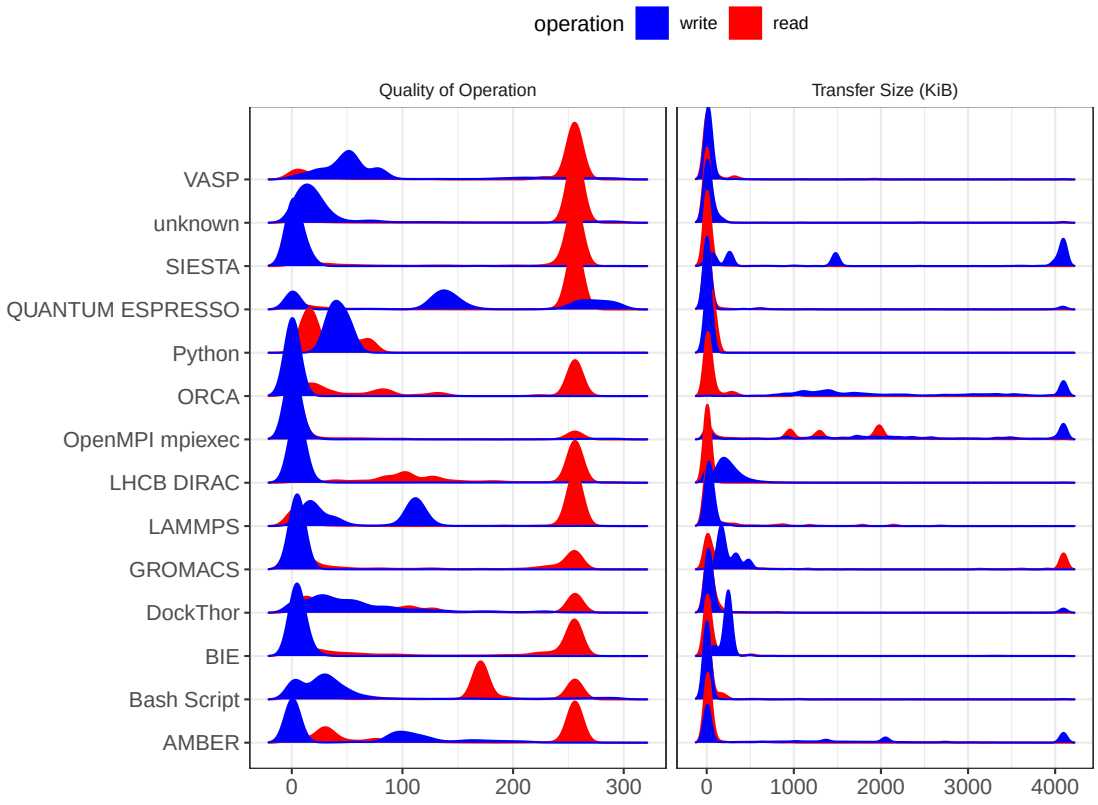


Source: Author

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 due to Lustre's default maximum bulk I/O RPC size (CORPORATION, 2017) 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 the 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. These applications could benefit from increasing the maximum bulk I/O RPC size since they presented the use of 4 MiB transfer size of write operations. The application category that presented the largest reading sizes was the *OpenMPI mpiexec*, using above 2 MiB during 75% of its execution. The applications with the most notably smallest sizes were *LAMMPS*, *DockThor*, *BIE*, and *Bash Script*, with 75% of their operations under 40 KiB.

Figure 5.12 – 2021 Distribution of the Quality of Operation (left) and Transfer Size (right). The x-axis are the QO index and size in KiB, respectively.

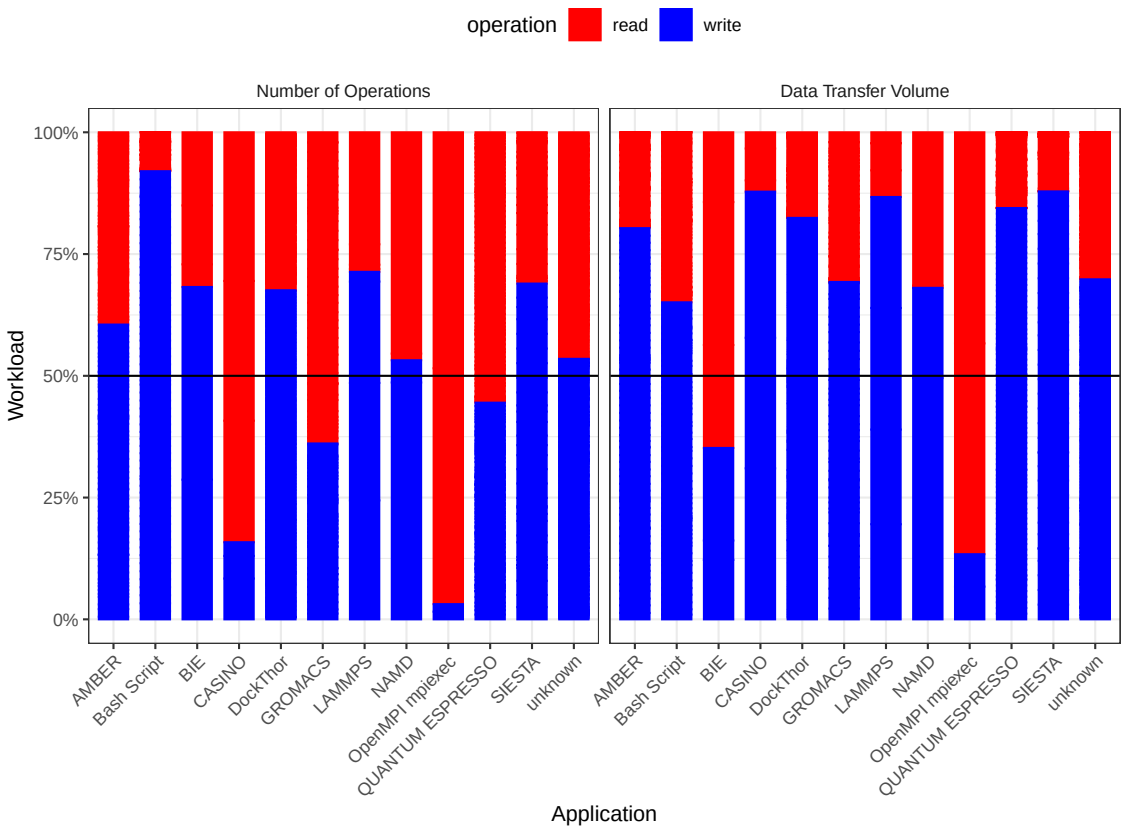


Source: Author

Figure 5.12 depicts that the applications observed during the 2021 observation window present a similar behavior as in 2020. 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 largest sizes for both reads (50% above 1 MiB) and writes (75% above 1 MiB). We could not 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.

Figure 5.13 – 2020 applications' workload distribution.



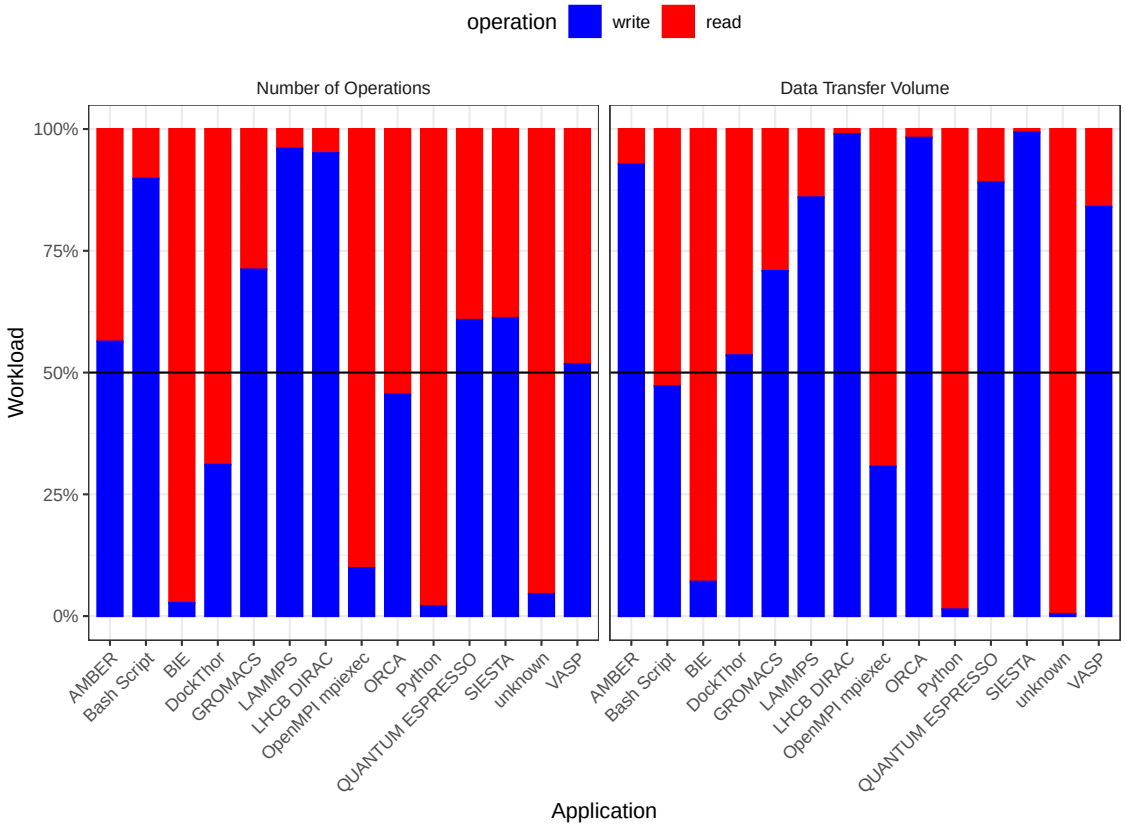
Source: Author

Most applications' workload in 2020 is dominated by write operations, regarding the number of operations and volume of data transferred. Figure 5.13 depicts that 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. Still, the volume of data transferred is dominated by writes, indicating that the workflow of these two Science Domains utilizes many small read operations and few larger write operations.

Figure 5.14 – 2021 applications' workload distribution.



Source: Author

The Figure 5.14 shows that 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 the number of operations and volume of data transferred.

The emergence of jobs using applications developed with *Python* and presenting a read-intensive workload lead us to believe that those are ML applications since the main ML frameworks (e.g., TensorFlow [15], Keras [16], Caffe [17], pyTorch [18], Apache MXNet [19], scikit-learn [20]) are implemented with (or have an interface to) Python (WANG et al., 2019) and are read-intensive (DRYDEN et al., 2021; CHOWDHURY et al., 2019; CHIEN et

[15] https://www.tensorflow.org/
[16] https://keras.io/
[17] https://caffe.berkeleyvision.org/
[18] https://pytorch.org/
[19] https://mxnet.apache.org/
[20] https://scikit-learn.org/stable/

Table 5.4 – Average Data Transfer per Job from 2021

Application	Read (GiB)	Write (GiB)
unknown	5,394	23
BIE	793	60
OpenMPI mpiexec	95	42
AMBER	3	44
QUANTUM ESPRESSO	2	22

Source: Author

al., 2018). However, at the current development stage of our methodology, we cannot guarantee that all the jobs using Python are ML jobs because SLURM does not provide other information besides the name of the executable used. This prompts the need for a better application identification in the resource management and job scheduling system (e.g., SLURM, LoadLeveler [21], PBS Pro [22]), a mechanism that would allow the user to specify the application's name, greatly facilitating the system administrator's task to recognize the purposes of a job. The precise application identification and characterization enable supercomputers to meet the scientific community's emerging needs better.

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 to transfer size below 45 KiB, writings presented transfer sizes above 1 MiB for 75% of the time. *Bash Script* showed 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.

Table 5.4 presents the five most data-intensive applications from 2021. The applications categorized under the *unknown* group (the second most executed) have the highest average amount of data read per job ($5,394$ GiB), being responsible for the expressive increase in the reading activities observed in Figure 5.1b. Other noteworthy applications are *BIE* and the *OpenMPI mpiexec* group, with a significant difference between the amount of data read and written. Interestingly, none of the five applications presented an evenly write/read workload.

Finally, we evaluated each application's level of I/O parallelism using the data collected at Compute Node at each timestamp. We present the results in two facets: (1) how many OSTs the application used and (2) how many Compute Nodes performed I/O operations. In 2020, each application's average simultaneous OST was ≈ 3.24, which

[21] https://publibfp.boulder.ibm.com/epubs/pdf/c2367902.pdf
[22] https://www.altair.com/pbs-professional

is below half the available OSTs (10 in total). During 75% of the time, the applications use only up to four OSTs simultaneously. 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 more significant number of OSTs. It was possible to observe cases where the application used only one Compute Node to write on all ten OSTs. It is essential to notice that the parallel I/O performed by the applications of multiple OSTs might be due to accessing a single file striped across two or more OSTs or by accessing, at the same time, different files stored individually on different OSTs. It is currently impossible to identify which type of access the application performs.

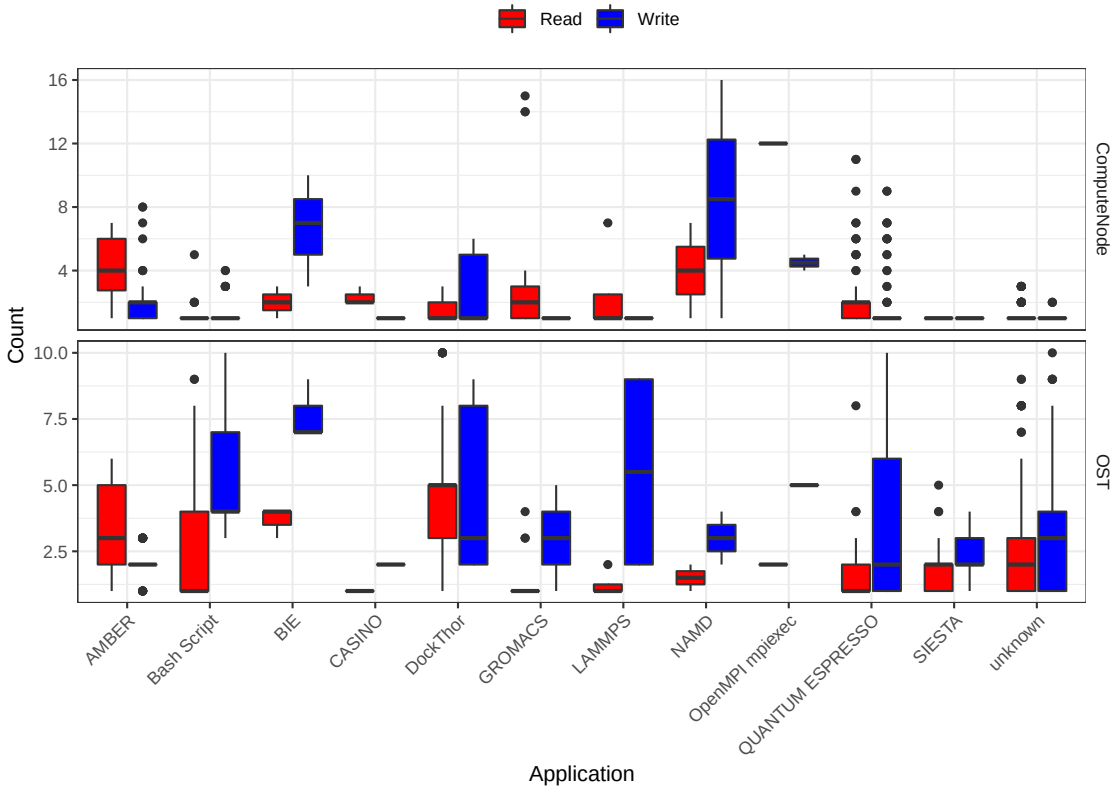
Figure 5.15 depicts the level of parallelism used by each application, distinguished by the read and write operations. It is possible to observe that the level of parallelism among applications is higher on the write operations on both Compute Node and OST usage, except for the *AMBER*. Interesting applications are (i) *GROMACS* and *LAMMPS*, which, on average, use more simultaneous Compute Nodes for reading operations and only one (in both cases) for writes; and (ii) *CASINO*, that uses multiple compute nodes to read from a single OST but uses a single Compute Node to write to multiple OSTs.

The level of parallelism in 2021 follows a similar trend as the one observed in 2020, with the average 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 an 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 simultaneously.

As presented by Figure 5.16, eight use more OSTs for reading operations than for writing from the fourteen identified applications. Interesting cases we highlight are: (i) *ORCA* and *Python* used only one Compute Node to perform their I/O operations but used up to ten OSTs for simultaneous I/O operations; (ii) *LAMMPS* and *SIESTA* used up to four Compute Nodes for reading operations and only one for writes, but on 75% of the cases used up to nine OSTs for writes and only one for reads; and (iii) the *OpenMPI mpiexec* group presented the highest simultaneous Compute Node utilization (25 nodes).

The analysis of how the applications used the Lustre PFS on SDumont made it

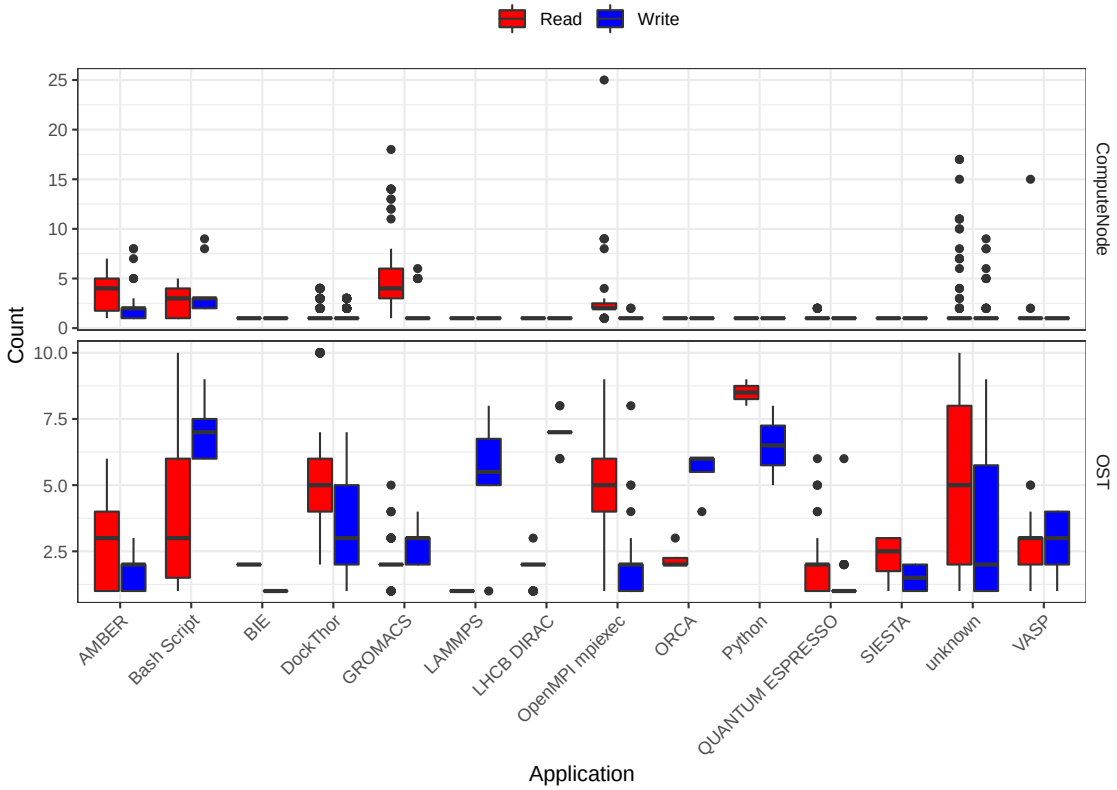
Figure 5.15 – 2020 Distribution of the Simultaneous Resource Used by each application in read (red) and write (blue). The y-axis (count) represents the amount of resource simultaneously used by each job of the application.



Source: Author

possible 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 (FOLK; CHENG; YATES, 1999) and MPI-IO (CORBETT et al., 1996)) 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 OSTs. As the *Coverage Factor of the Bandwidth* (Figures 5.10a and 5.10b) shows, few applications issuing high throughput operations use most of the aggregate bandwidth provided by the storage system, leaving plenty of available bandwidth. On the other hand, there is considerable demand for low latency performance to respond to the many small transfer size operations. It is possible to suggest that an I/O Forwarding (ALI et al., 2009) or Burst Buffer (LIU et al., 2012) 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 (BEHZAD et al., 2019; GAINARU et al., 2015; DORIER et al., 2014) that auto-tune and optimize I/O for a more extensive set of applications.

Figure 5.16 – 2021 Distribution of the Simultaneous Resource Used by each application in read (red) and write (blue). The y-axis (count) represents the amount of resource simultaneously used by each job of the application.



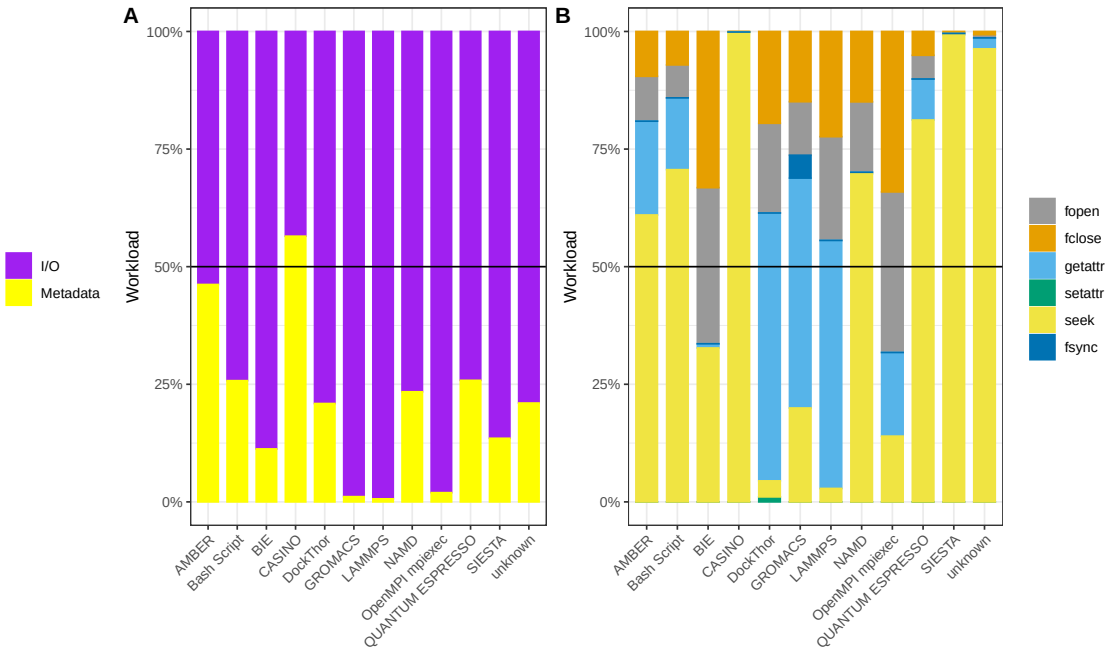
Source: Author

5.2.2 Applications Metadata Analysis

The analysis presented in this subsection used the information collected at the compute nodes to characterize the applications' metadata utilization for the periods defined in Section 5.2. Figure 5.17A illustrates the load ratio between I/O and metadata operations recorded by each application's jobs executed in the 2020 window. Only *CASINO* presents a metadata-intensive behavior (more metadata than I/O) from the twelve identified applications. *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 5.17B 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 access. Two applications (*BIE* and *OpenMPI mpiexec*) present a high load of `fopen` and `fclose` operations, (and, to a lesser extent, the ap-

Figure 5.17 – 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.



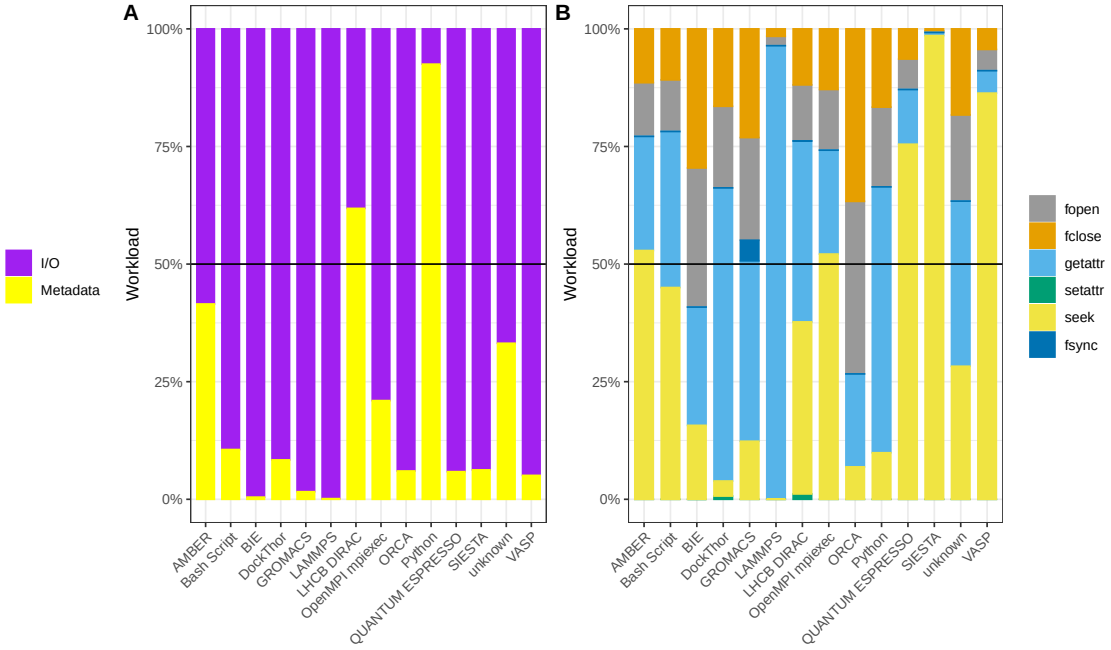
Source: Author

plications *DockThor* and *LAMMPS*), indicating the use of many files during its workflow. Large numbers of file opens and closures lead to poor scalability as overhead (latency) in open and close in Lustre is very high (SAINI et al., 2012). Three applications (*DockThor*, *GROMACS*, and *LAMMPS*) presented a high load of the `getattr`, operations that retrieve file information from the MDS. This type of metadata operation should be avoided as much as possible because it increases the overload on the MDS (i.e., for every getattr operation, the MDS has to contact the OSS and OST to get the file's attributes, increasing the traffic on the network. If the file is striped across many OSTs, this scenario gets even worse.). 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 to provide performance. 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, poorly tuned applications executing on the shared PFS could harm performance to themselves and other concurrent jobs. Future HPC storage systems should automatically isolate or provide mechanisms to adjust the PFS 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 Science makes mi-

nor use of metadata operations, presenting the lowest workload. The other domains show a metadata workload between 10% and 25%.

Figure 5.18 – 2021 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.



Source: Author

The metadata load distribution from the 2021 is depicted by Figure 5.18. For the applications identified in the 2021 period and not present in 2020, 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`.

An option to better service the observed workload is to install a second MDS, tailored for small access and low latency, on the Lustre to house those metadata-intensive applications and projects. This way would be possible to logically split the namespace into "normal" and "demanding" areas regarding the metadata usage.

6 DISCUSSION

This chapter discusses the lessons learned with the analysis of SDumont and compares the findings against the storage system of other HPC platforms described in the following works: Kim and Gunasekaran (2015) studied Spider, a Lustre-based storage cluster located at the Oak Ridge Leadership Computing Facility (OLCF) and used by the Titan supercomputer. Spider is composed of 192 OSSs, $1,344$ OSTs and have a bandwidth of 240 GB/s; Gunasekaran et al. (2015) presents an evaluation of the Spider 2 storage system, also located at OLCF. It comprises 288 OSSs, $2,016$ OSTs, and has a bandwidth of 1 TB/s; and Patel et al. (2019) studied the Lustre parallel storage system shared by Edison and Cori supercomputers, deployed at the National Energy Research Scientific Computing Center (NERSC). The Lustre file system is composed of 248 OSSs, each having a corresponding OST, with a bandwidth of 700 GB/s.

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 and Gunasekaran (2015) presented only a marginal difference between the read (42.2%) and write (57.8%) workload, the system analyzed by Gunasekaran et al. (2015) shows a write dominance of 75%. In contrast, Patel et al. (2019) observed that the read constantly dominates the workload. The data transfer volume observed on SDumont presented a certain seasonality (Figure 5.4). The 2020 period presented an overall workload of data movement dominated by writings, with only three weeks demonstrating reading dominance. The data movement of 2021, on the other hand, presented an inverse behavior, mainly being dominated by readings. However, regarding 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. The findings of SDumont help design future storage system acquisition for HPC systems with this type of heterogeneous environment.

Transfer Size: The system studied by Kim and Gunasekaran (2015) presents most transfer sizes that are either less than 16 KiB or between 512 KiB and 1 MiB. Gunasekaran et al. (2015) concluded that 60% of write requests are 4 KiB or less, and over 50% of reads were at least 1 MiB. The transfer sizes on SDumont present an inverted proportion, with reads being small (50% below 23 KiB) while writes are larger (75% above 530 KiB). The write transfer 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 the writes. This information helps optimize block devices since the PFS throughput is highly dependent on the transfer size.

Bandwidth Usage: SDumont presented low bandwidth usage if compared with other systems. Kim and Gunasekaran (2015) observed that the peak read throughput reached 75% of the maximum bandwidth, while the writes reached 54%. The system analyzed by Gunasekaran et al. (2015) presented up to 80% of the bandwidth for reads and 70% for writes. The peak of writes on SDumont reached 42.41% of the maximum bandwidth, while the reads reached 39.89%. Aside from being lower, we see an inverted behavior.

Load Imbalance: This is a problem that has not been fully addressed in large-scale platforms. As shown with our results (Figure 5.5), the load imbalanced on SDumont can be as severe as a single OST receiving 100% of the load. The works presented by Kim and Gunasekaran (2015), Patel et al. (2019) reported similar problems. High load imbalance results in low throughput and under-utilization. To ease this problem, system administrators might consider adopting a dynamic load balancer that automatically coordinates the workload and data placement among I/O servers.

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. Even on a large-scale system, as depicted by Patel et al. (2019), in more than 98% of the cases, the applications used only 20 of the available 248 OSTs. 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. The peak throughput observed was when a single application performed a write operation using all OSTs at once. 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 striping parameters for efficient usage.

7 CONCLUSION

This work evaluated the PFS storage of the SDumont supercomputer, which houses research from different Science Domains. The study used metrics collected from storage servers and compute nodes to analyze the machine's I/O behavior for three months from two years. The utilization of the proposed methodology provided insights into understanding Lustre's usage and the I/O needs of scientific applications. It was possible to identify critical aspects that negatively impact the performance of SDumont, such as:

- *Inefficient read operations*: There's a high count of operations using small transfer sizes ($\approx 52\times$ more read requests than writes but on average, write size was $\approx 3\times$ bigger than the reads), which often translates into poor performance. Since the Lustre file system is oriented for large file I/O performance, the current scenario might limit applications with high read demand.
- *Low latency demand*: The utilization of SDumont is not even near saturating the bandwidth of the storage system, with the peak throughput not reaching 50% of the available resources. This indicates that there is still room to service applications that perform high I/O throughput. However, the high count of small operations and the high load of `seek` operations denotes an important demand for low latency operations.
- *High-level libraries*: The applications are not taking full advantage of high-level I/O libraries and middleware to aggregate the small requests into larger ones.
- *Imbalance among resources*: The analysis of individual OST usage and the LI metric indicates how the load is distributed, denoting some severe and lasting cases where the overload corresponds to $3\times$ the average OSTs' load.
- *Problematic applications*: Identification of problematic behaviors, such as *BIE*, which exhibits the worst $read_{qo}$ and is read-intensive. The Science Domain analysis shows that *Biodiversity* and *Health Sciences* also have high demands for reading operations.
- *Demand for metadata operations*: The metadata analysis shows a considerable demand for this type of operation on SDumont, accounting for 60% of all file system operations. This scenario raises a warning for the system administrators to take some action about it quickly.

Identifying these aspects guides the administration of SDumont to focus their ef-

forts on helping improve the system's performance and usability. The actions can be divided into *production-ready* and experimental approaches. The *production-ready* are techniques or tools already available as a "product on the market", implemented, tested, and operational in many HPC centers. These are the starting point that system administrators can directly implement. Under this category, the system administrators of SDumont can:

- Evaluate the adoption of an I/O forwarding layer (ALI et al., 2009; OHTA et al., 2010; VISHWANATH et al., 2010; LANDSTEINER et al., 2016; SUGIYAMA; WALLACE, 2008), that can receive the small-sized operations from the CNs, aggregate, reorder and reschedule them, and then directly interact with the storage servers, thus relieving the high read demand.
- To attend to the low latency operations demand, they can adopt SSDs, which can be done on two fronts: on the CNs to be used as a client-side cache device (QIAN et al., 2019), and on the PFS's servers (CAULFIELD; GRUPP; SWANSON, 2009; LOCKWOOD et al., 2019) by replacing the HDD.
- Revise Lustre's configuration parameters, such as the default *striping* policy (to help with the load imbalance among the OSTs) and increase maximum bulk I/O RPC size (CORPORATION, 2017) (to help applications issuing bigger transfer sizes).
- Implement Lustre improvements (FRAGALLA; LOEWE; PETERSEN, 2020) provided by newer versions (2.x). The Distributed Namespace (CROWE; LAVENDER; SIMMS, 2015; SIMMONS et al., 2016) (DNE) allows the use of multiple metadata servers, thus distributing the load, and Data on MDT (HAN; KIM; EOM, 2016) (DoM) allows to store small files on the MDT instead of placing all data on OSTs as is normally done in Lustre, significantly reducing the number of requests and accesses to OSTs.
- Regarding the most demanding projects and problematic applications, the system administrators can gather a task force to perform a detailed analysis and address particular requirements, such as implementing a high-level I/O library (e.g., ADIOS (GODOY et al., 2020), NetCDF (REW; DAVIS, 1990), HDF5 (FOLK; CHENG; YATES, 1999), MPI-IO (CORBETT et al., 1996)) to improve performance.

The *experimental* group comprise approaches under research and not yet widely implemented, thus requiring careful assessment before employing on a production system. Under this category, the system administrators of SDumont can:

- Assess the use of a load balancer that automatically coordinates the workload and data placement among Object Storage Servers (NEUWIRTH et al., 2017; WADHWA et al., 2019), thus smoothing the imbalance.
- Evaluate the implementation of a framework to auto-tune and optimize the I/O stack (BEHZAD et al., 2019; GAINARU et al., 2015; DORIER et al., 2014) for a extensive set of applications.
- Study the indexing mechanisms (PAUL et al., 2020) and client-side pre-allocation (LI et al., 2019) to improve the metadata performance.

7.1 Future Work

Future work will focus on improving the application identification. The significant number of jobs categorized under the *Bash Scripts*, *OpenMPI mpiexec*, and *unknown* groups presents a challenge to a better understanding of the storage system, since many applications are grouped under them. This better identification would also help provide additional information for frameworks oriented for I/O optimizations guided for specific applications.

For future work, there is a need for improving the application identification. The 2021 application analysis (5.2.1) how critical this is, where the group of jobs *Bash Scripts*, *OpenMPI mpiexec*, *Python*, and *unknown* represented 24% of the jobs submitted. Those groups presents a challenge to a better understanding of the storage system because we have insufficient knowledge about its purposes and many applications are grouped under them. For example, the *Python* group, that presented a read-intensive workload, might be ML applications since the most popular frameworks are implemented with (or have an interface to) Python (WANG et al., 2019) and are read-intensive (DRYDEN et al., 2021; CHOWDHURY et al., 2019; CHIEN et al., 2018). However, at the current development stage of our methodology, we cannot determine how many jobs using Python are ML jobs because SLURM does not provide other information besides the name of the executable used (in this case, it only informs `python`). One option is to parse the user's submission script and check which environment modules it loads. Another possibility is to revise the system administrators' database of known applications, looking for possible improvements. The precise application identification and characterization enable supercomputers to meet the scientific community's emerging needs better and also help provide additional

information for frameworks oriented to I/O optimizations guided for specific applications.

The scalability and performance of our methodology also need attention. For example, the *Data Crossing* process to generate the *Job Usage* dataset of five days took around 7 hours. All the Python scripts developed for the Step 2 of the methodology (**Pre-Processing**) are serial, not exploring the multi-thread capabilities of current CPUs. Implementing some sort of parallelism in the data processing would speed up the Step 2 operations and enable studies of more extended periods.

Furthermore, the utilization metrics are continuously collected on the compute nodes, even when no jobs are running. This generates data containing zero value records that only occupy storage space. In our future work we plan to integrate the collection tool with the SLURM resource manager to collect the metrics only when a job starts running on the compute nodes. We also plan to add the collection of utilization metrics from other subsystems, such as CPU, network, and the CNs' local disks (SSD). This modification aims to assess how the I/O utilization interacts with and demands from those other subsystems. For example, how much the imbalance among the OSTs costs from the OSSs' CPU, or if the amount of data sent to the OSTs through the network is the same arriving at the OSTs.

Another objective in our plans is to assess the impacts of implementing new technologies and modifications in the storage stack of SDumont. For example, currently on SDumont, the network to access the Lustre PFS, with an aggregate bandwidth of 70 GB/s, does not impose a bottleneck on the communication. However, what will happen if the administration of SDumont increases the maximum bulk I/O RPC size to 16 MiB? The applications will be capable of transferring larger amounts of data per operation, possibly increasing network congestion, which might turn into a bottleneck. If the administration of SDumont implements an I/O forwarding layer, how many small operations can it aggregate, and how would this impact the Lustre PFS? These questions need careful assessment to provide efficient resource utilization of the SDumont supercomputer.

7.2 Publications

The following papers were produced during this dissertation, first listing the ones related to the this work and to the HPC storage research field, including those submitted and under review:

- **CARNEIRO, A. R.**; BEZ, J. L.; OSTHOFF, C.; SCHNORR, L. M.; NAVAUX, P. O. A. Uncovering I/O Demands on HPC Platforms: Peeking Under the Hood of Santos Dumont. In: Journal of Parallel and Distributed Computing, 2022 (*Submitted*).

  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 software, 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.

- **CARNEIRO, A. R.**; BEZ, J. L.; OSTHOFF, C.; SCHNORR, L. M.; NAVAUX, P. O. A. HPC Data Storage at a Glance: The Santos Dumont Experience. In: 2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), 2021.

  ABSTRACT: High-Performance Computing (HPC) platforms are used to solve the most diverse scientific problems in research areas, such as biology, chemistry, physics, and health sciences. Researchers use a multitude of scientific software, which have different requirements. These requirements 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 a mixed workload scenario when storing data from the applications. Knowledge of the application set and its performance running in a supercomputer is needed to understand the storage system's usage, pinpoint possible bottlenecks, and guide 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, where we were able to identify inefficient usage and problematic factors of performance.

- BEZ, J. L.; **CARNEIRO, A. R.**; PAVAN, P. J.; GIRELLI, V. S.; BOITO, F. Z.; FAGUNDES, B. A.; OSTHOFF, C.; SILVA DIAS, P. L.; MEHAUT, J.-F.; NAVAUX, P. O. A. I/O Performance of the Santos Dumont Supercomputer. In: The International Journal of High Performance Computing Applications, 2019.

  ABSTRACT: In this article, 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 that 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 application's 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.

- **CARNEIRO, A. R.**; BEZ, J. L.; BOITO, F. Z.; FAGUNDES, B. A.; OSTHOFF, C.; NAVAUX, P. O. A. Collective I/O Performance on the Santos Dumont Supercomputer. In: 2018 26th Euromicro International Conference on Parallel, Distributed and Networkbased Processing (PDP), 2018.

  ABSTRACT: The historical gap between processing and data access speeds causes many applications to spend a large portion of their execution on I/O operations. From the point of view of a large-scale, expensive, supercomputer, it is important to ensure applications achieve the best I/O performance to promote an efficient us-

age of the machine. In this paper, we evaluate the I/O infrastructure of the Santos Dumont supercomputer, the largest one from Latin America. More specifically, we investigate the performance of collective I/O operations. By conducting an analysis of a scientific application that uses the machine, we identify large performance differences between the available MPI implementations. We then further study the observed phenomenon using the BT-IO and IOR benchmarks, in addition to a custom microbenchmark. We conclude that the customized MPI implementation by Bull (used by more than 20% of the jobs) presents the worst performance for small collective write operations. Our results are being used to help the Santos Dumont users to achieve the best performance for their applications. Additionally, by investigating the observed phenomenon, we provide information to help improve future MPI-IO collective write implementations.

The following papers were also published but are not directly related to the I/O research field, though they are still relevant to the HPC research field:

- **CARNEIRO, A. R.**; SERPA, M. S.; NAVAUX, P. O. A. Lightweight Deep Learning Applications on AVX-512. In: 2021 IEEE Symposium on Computers and Communications (ISCC), 2021.

  ABSTRACT: Machine Learning and Deep Learning applications are of paramount importance these days. Different areas of academia and industry use daily workloads based on these applications. Several aspects are relevant regarding their applicability, such as the complexity and accuracy of the models and their performance and energy efficiency. Currently, there is a trend to usually favor the use of GPUs to train and execute Deep Learning models, intensified by specialized hardware. However, this article demonstrates that using a CPU with AVX-512 instructions can achieve comparable performance to current GPUs and, depending on the workload, suppress it by $\approx 1.8\times$.

- HERRERA, S.; RIBEIRO, W.; TEIXEIRA, T.; **CARNEIRO, A. R.**; CABRAL F.; BORGES, M.; OSTHOFF, C. Avaliação de Desempenho no Supercomputador SDumont de uma Estratégia de Decomposição de Domínio usando as Funcionalidades de Mapeamento Topológico do MPI para um Método Numérico de Escoamento de Fluidos. In: Anais da VI Escola Regional de Alto Desempenho do Rio de Janeiro, 2020.

  ABSTRACT: Oil and gas simulations need new high-performance computing tech-

niques to deal with the large amount of data allocation and the high computational cost that we obtain from the numerical method. The domain decomposition technique (domain division technique) was applied to a three-dimensional oil reservoir, where the MPI (Message Passing Interface) allowed the creation of a uni, bi and three-dimensional topology, where a subdivision of a reservoir could be solved in each MPI process created. A performance study was developed with these domain decomposition strategies in 20 computational nodes of the SDumont Supercomputer, using a Cascade Lake architecture.

REFERENCES

ALAM, S. R. et al. Parallel i/o and the metadata wall. In: **Proceedings of the Sixth Workshop on Parallel Data Storage**. New York, NY, USA: Association for Computing Machinery, 2011. (PDSW '11), p. 13–18. ISBN 9781450311038. Available from Internet: <https://doi.org/10.1145/2159352.2159356>.

ALI, N. et al. Scalable i/o forwarding framework for high-performance computing systems. In: **2009 IEEE International Conference on Cluster Computing and Workshops**. [s.n.], 2009. p. 1–10. Available from Internet: <https://doi.org/10.1109/CLUSTR.2009.5289188>.

ARPACI-DUSSEAU, R. H.; ARPACI-DUSSEAU, A. C. **Operating Systems: Three Easy Pieces**. 1.00. ed. [S.l.]: Arpaci-Dusseau Books, 2018.

BARTZ, C. et al. A best practice analysis of hdf5 and netcdf-4 using lustre. In: **ISC High Performance 2015**. [s.n.], 2015. Available from Internet: <https://doi.org/10.1007/978-3-319-20119-1_20>.

BEHZAD, B. et al. Optimizing i/o performance of hpc applications with autotuning. Association for Computing Machinery, New York, NY, USA, v. 5, n. 4, mar 2019. ISSN 2329-4949. Available from Internet: <https://doi.org/10.1145/3309205>.

BETKE, E.; KUNKEL, J. M. Footprinting parallel i/o – machine learning to classify application's i/o behavior. In: **International Conference on High Performance Computing**. Springer International Publishing, 2019. p. 214–226. Available from Internet: <https://doi.org/10.1007/978-3-030-34356-9_18>.

BEZ, J. L. et al. I/o performance of the santos dumont supercomputer. In: **The International Journal of High Performance Computing Applications**. SAGE, 2019. p. 227–245. Available from Internet: <https://doi.org/10.1177/1094342019868526>.

BHATELE, A. et al. There goes the neighborhood: Performance degradation due to nearby jobs. In: **SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis**. [s.n.], 2013. p. 1–12. Available from Internet: <https://doi.org/10.1145/2503210.2503247>.

BLAGODUROV, S.; FEDOROVA, A. Towards the contention aware scheduling in hpc cluster environment. **Journal of Physics: Conference Series**, IOP Publishing, v. 385, p. 012010, oct 2012. Available from Internet: <https://doi.org/10.1088/1742-6596/385/1/012010>.

BYNA, S. et al. Parallel i/o prefetching using mpi file caching and i/o signatures. In: **SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing**. [s.n.], 2008. p. 1–12. Available from Internet: <https://doi.org/10.1109/SC.2008.5213604>.

CARNS, P. et al. Understanding and improving computational science storage access through continuous characterization. **ACM Trans. Storage**, Association for Computing Machinery, New York, NY, USA, v. 7, n. 3, oct 2011. ISSN 1553-3077. Available from Internet: <https://doi.org/10.1145/2027066.2027068>.

CARNS, P. et al. 24/7 characterization of petascale i/o workloads. In: . [s.n.], 2009. p. 1 – 10. Available from Internet: <https://doi.org/10.1109/CLUSTR.2009.5289150>.

CAULFIELD, A. M.; GRUPP, L. M.; SWANSON, S. Gordon: Using flash memory to build fast, power-efficient clusters for data-intensive applications. In: **Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems**. New York, NY, USA: Association for Computing Machinery, 2009. (ASPLOS XIV), p. 217–228. ISBN 9781605584065. Available from Internet: <https://doi.org/10.1145/1508244.1508270>.

CHASAPIS, K. et al. Evaluating lustre's metadata server on a multi-socket platform. In: **2014 9th Parallel Data Storage Workshop**. [s.n.], 2014. p. 13–18. Available from Internet: <https://doi.org/10.1109/PDSW.2014.5>.

CHIEN, S. W. D. et al. Characterizing deep-learning i/o workloads in tensorflow. In: **2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS)**. [S.l.: s.n.], 2018. p. 54–63.

CHING, A. et al. High-performance techniques for parallel i/o. In: **Handbook of Parallel Computing**. [S.l.]: Chapman and Hall/CRC, 2007. p. 887–910.

CHOWDHURY, F. et al. I/o characterization and performance evaluation of beegfs for deep learning. In: **Proceedings of the 48th International Conference on Parallel Processing**. New York, NY, USA: Association for Computing Machinery, 2019. (ICPP 2019). ISBN 9781450362955. Available from Internet: <https://doi.org/10.1145/3337821.3337902>.

CHUNDURI, S. et al. Gpcnet: Designing a benchmark suite for inducing and measuring contention in hpc networks. In: . New York, NY, USA: Association for Computing Machinery, 2019. (SC '19). ISBN 9781450362290. Available from Internet: <https://doi.org/10.1145/3295500.3356215>.

CORBETT, P. et al. Overview of the mpi-io parallel i/o interface. In: ____. **Input/Output in Parallel and Distributed Computer Systems**. Boston, MA: Springer US, 1996. p. 127–146. ISBN 978-1-4613-1401-1. Available from Internet: <https://doi.org/10.1007/978-1-4613-1401-1_5>.

CORPORATION, I. **Lustre Software Release 2.x**. 2017. Available from Internet: <https://doc.lustre.org/lustre_manual.xhtml>.

CROWE, T.; LAVENDER, N.; SIMMS, S. Scalability testing of dne2 in lustre 2.7. **Lustre Users Group**, 2015.

DICKENS, P. M.; LOGAN, J. Y-lib: A user level library to increase the performance of mpi-io in a lustre file system environment. In: **Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing**. New York, NY, USA: Association for Computing Machinery, 2009. (HPDC '09), p. 31–38. ISBN 9781605585871. Available from Internet: <https://doi.org/10.1145/1551609.1551617>.

DING, X. et al. Diskseen: Exploiting disk layout and access history to enhance i/o prefetch. In: **USENIX Annual Technical Conference**. [s.n.], 2007. v. 7, p. 261–274. Available from Internet: <https://www.usenix.org/legacy/events/usenix07/tech/full_papers/ding/ding.pdf>.

DODGE, Y. The concise encyclopedia of statistics. In: ____. New York, NY: Springer New York, 2008. chp. Kolmogorov–Smirnov Test, p. 283–287. ISBN 978-0-387-32833-1. Available from Internet: <https://doi.org/10.1007/978-0-387-32833-1_214>.

DORIER, M. et al. Calciom: Mitigating i/o interference in hpc systems through cross-application coordination. In: **2014 IEEE 28th International Parallel and Distributed Processing Symposium**. [s.n.], 2014. p. 155–164. Available from Internet: <https://doi.org/10.1109/IPDPS.2014.27>.

DRYDEN, N. et al. Clairvoyant prefetching for distributed machine learning i/o. In: **Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis**. New York, NY, USA: Association for Computing Machinery, 2021. (SC '21). ISBN 9781450384421. Available from Internet: <https://doi.org/10.1145/3458817.3476181>.

FOLK, M.; CHENG, A.; YATES, K. Hdf5: A file format and i/o library for high performance computing applications. In: **Proceedings of supercomputing**. [S.l.: s.n.], 1999. v. 99, p. 5–33.

FRAGALLA, J.; LOEWE, B.; PETERSEN, T. K. New lustre features to improve lustre metadata and small-file performance. **Concurrency and Computation: Practice and Experience**, Wiley Online Library, v. 32, n. 20, p. e5649, 2020. Available from Internet: <https://doi.org/10.1002/cpe.5649>.

GAINARU, A. et al. Scheduling the i/o of hpc applications under congestion. In: **2015 IEEE International Parallel and Distributed Processing Symposium**. [s.n.], 2015. p. 1013–1022. Available from Internet: <https://doi.org/10.1109/IPDPS.2015.116>.

GE, R. Evaluating parallel i/o energy efficiency. In: **2010 IEEE/ACM Int'l Conference on Green Computing and Communications Int'l Conference on Cyber, Physical and Social Computing**. [s.n.], 2010. p. 213–220. Available from Internet: <https://doi.org/10.1109/GreenCom-CPSCom.2010.130>.

GIANNOZZI, P. et al. Quantum espresso toward the exascale. **The Journal of Chemical Physics**, v. 152, n. 15, p. 154105, 2020. Available from Internet: <https://doi.org/10.1063/5.0005082>.

GODOY, W. F. et al. Adios 2: The adaptable input output system. a framework for high-performance data management. **SoftwareX**, v. 12, p. 100561, 2020. ISSN 2352-7110. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S2352711019302560>.

GUNASEKARAN, R. et al. Comparative i/o workload characterization of two leadership class storage clusters. In: **Proceedings of the 10th Parallel Data Storage Workshop**. New York, NY, USA: Association for Computing Machinery, 2015. (PDSW '15), p. 31–36. ISBN 9781450340083. Available from Internet: <https://doi.org/10.1145/2834976.2834985>.

HAN, J.; KIM, D.; EOM, H. Improving the performance of lustre file system in hpc environments. In: **2016 IEEE 1st International Workshops on Foundations and Applications of Self Systems (FAS*W)**. [S.l.: s.n.], 2016. p. 84–89.

HANSUN, S. A new approach of moving average method in time series analysis. In: **2013 Conference on New Media Studies (CoNMedia)**. [s.n.], 2013. p. 1–4. Available from Internet: <https://doi.org/10.1109/CoNMedia.2013.6708545>.

HASHIMOTO, Y.; AIDA, K. Evaluation of performance degradation in hpc applications with vm consolidation. In: **2012 Third International Conference on Networking and Computing**. [s.n.], 2012. p. 273–277. Available from Internet: <https://doi.org/10.1109/ICNC.2012.50>.

HU, W. et al. Storage wall for exascale supercomputing. **Frontiers of Information Technology and Electronic Engineering**, v. 17, n. 11, p. 1154–1175, Nov 2016. ISSN 2095-9230. Available from Internet: <https://doi.org/10.1631/FITEE.1601336>.

HWU, W.-m.; KEUTZER, K.; MATTSON, T. G. The concurrency challenge. **IEEE Design Test of Computers**, v. 25, n. 4, p. 312–320, 2008. Available from Internet: <https://doi.org/10.1109/MDT.2008.110>.

ISAKOV, M. et al. Hpc i/o throughput bottleneck analysis with explainable local models. In: **SC20: International Conference for High Performance Computing, Networking, Storage and Analysis**. [s.n.], 2020. p. 1–13. Available from Internet: <https://doi.org/10.1109/SC41405.2020.00037>.

JOKANOVIC, A. et al. Impact of inter-application contention in current and future hpc systems. In: **2010 18th IEEE Symposium on High Performance Interconnects**. [s.n.], 2010. p. 15–24. Available from Internet: <https://doi.org/10.1109/HOTI.2010.25>.

KIM, Y.; GUNASEKARAN, R. Understanding i/o workload characteristics of a peta-scale storage system. **Journal of Supercomputing**, Springer, v. 71, n. 3, p. 761–780, 2015. Available from Internet: <https://doi.org/10.1007/s11227-014-1321-8>.

KUNKEL, J. M. et al. **The 9^{th} IO500 and the Virtual Institute of I/O Report**. 2021. Available from Internet: <https://io500.org/files/sc21-io500-slides.pdf>.

KUNKEL, J. M.; MARKOMANOLIS, G. S. Understanding metadata latency with mdworkbench. In: SPRINGER. **International Conference on High Performance Computing**. 2018. p. 75–88. Available from Internet: <https://doi.org/10.1007/978-3-030-02465-9_5>.

LANDSTEINER, B. R. et al. Architecture and design of cray datawarp. In: . [S.l.: s.n.], 2016.

LANG, S. et al. I/o performance challenges at leadership scale. In: **Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis**. [s.n.], 2009. p. 1–12. Available from Internet: <https://doi.org/10.1145/1654059.1654100>.

LATHAM, R. et al. **HPC I/O for Computational Scientists**. 2014. Available from Internet: <https://extremecomputingtraining.anl.gov//files/2014/01/hpc-io-all-final.pdf>.

LAWRENCE, B. et al. **Parallel I/O Performance Benchmarking and Investigation on Multiple HPC Architectures**. [S.l.], 2017. Available from Internet: <https://www.archer.ac.uk/documentation/white-papers/parallelIO-benchmarking/ARCHER-Parallel-IO-1.4.pdf>.

LI, H. et al. 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)**. [s.n.], 2019. p. 413–420. Available from Internet: <https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00069>.

LIANG, W. et al. Cars: A contention-aware scheduler for efficient resource management of hpc storage systems. **Parallel Computing**, v. 87, p. 25–34, 2019. ISSN 0167-8191. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S016781911830382X>.

LIAO, W.-k.; CHOUDHARY, A. Dynamically adapting file domain partitioning methods for collective i/o based on underlying parallel file system locking protocols. In: **SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing**. [s.n.], 2008. p. 1–12. Available from Internet: <https://doi.org/10.1109/SC.2008.5222722>.

LIU, N. et al. On the role of burst buffers in leadership-class storage systems. In: **2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)**. [s.n.], 2012. p. 1–11. Available from Internet: <https://doi.org/10.1109/MSST.2012.6232369>.

LIU, Y. et al. Raf: A random access first cache management to improve ssd-based disk cache. In: **2010 IEEE Fifth International Conference on Networking, Architecture, and Storage**. [s.n.], 2010. p. 492–500. Available from Internet: <https://doi.org/10.1109/NAS.2010.9>.

LOCKWOOD, G. K. et al. A quantitative approach to architecting all-flash lustre file systems. In: **High Performance Computing**. [S.l.]: Springer International Publishing, 2019. p. 183–197.

LOCKWOOD, G. K. et al. A year in the life of a parallel file system. In: **SC18: International Conference for High Performance Computing, Networking, Storage and Analysis**. [s.n.], 2018. p. 931–943. Available from Internet: <https://doi.org/10.1109/SC.2018.00077>.

LOCKWOOD, G. K. et al. Tokio on clusterstor: Connecting standard tools to enable holistic i/o performance analysis. 1 2018. Available from Internet: <https://www.osti.gov/biblio/1632125>.

LOFSTEAD, J. et al. Six degrees of scientific data: Reading patterns for extreme scale science io. In: **Proceedings of the 20th International Symposium on High Performance Distributed Computing**. New York, NY, USA: Association for Computing Machinery, 2011. (HPDC '11), p. 49–60. ISBN 9781450305525. Available from Internet: <https://doi.org/10.1145/1996130.1996139>.

LUU, H. et al. A multiplatform study of i/o behavior on petascale supercomputers. In: **Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing**. New York, NY, USA: Association for Computing Machinery, 2015. (HPDC '15), p. 33–44. ISBN 9781450335508. Available from Internet: <https://doi.org/10.1145/2749246.2749269>.

MESNIER, M.; GANGER, G.; RIEDEL, E. Object-based storage. **IEEE Communications Magazine**, v. 41, n. 8, p. 84–90, 2003. Available from Internet: <https://doi.org/10.1109/MCOM.2003.1222722>.

MICROSYSTEMS, S. **High-Performance Storage Architecture and Scalable Cluster File System**. [S.l.], 2007. Available from Internet: <http://www.csee.ogi.edu/~zak/cs506-pslc/lustrefilesystem.pdf>.

NEUWIRTH, S. et al. 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)**. [s.n.], 2017. p. 604–613. Available from Internet: <https://doi.org/10.1109/ICPADS.2017.00084>.

NIEUWEJAAR, N. et al. File-access characteristics of parallel scientific workloads. **IEEE Transactions on Parallel and Distributed Systems**, v. 7, n. 10, p. 1075–1089, 1996. Available from Internet: <https://doi.org/10.1109/71.539739>.

OHTA, K. et al. Optimization techniques at the i/o forwarding layer. In: **2010 IEEE International Conference on Cluster Computing**. [s.n.], 2010. p. 312–321. Available from Internet: <https://doi.org/10.1109/CLUSTER.2010.36>.

OLIVARES, T. et al. Performance study of nfs over myrinet-based clusters for parallel multimedia applications. In: **Canadian Conference on Electrical and Computer Engineering 2001. Conference Proceedings (Cat. No.01TH8555)**. [s.n.], 2001. v. 2, p. 999–1004 vol.2. Available from Internet: <https://doi.org/10.1109/CCECE.2001.933579>.

PATEL, T. et al. 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**. New York, NY, USA: Association for Computing Machinery, 2019. (SC '19). ISBN 9781450362290. Available from Internet: <https://doi.org/10.1145/3295500.3356183>.

PAUL, A. K. et al. Efficient metadata indexing for hpc storage systems. In: **2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)**. [s.n.], 2020. p. 162–171. Available from Internet: <https://doi.org/10.1109/CCGrid49817.2020.00-77>.

PRABHAT; KOZIOL, Q. **High Performance Parallel I/O**. 1st. ed. [S.l.]: Chapman & Hall/CRC Computational Science, 2014. ISBN 1466582340.

QIAN, Y. et al. Lpcc: Hierarchical persistent client caching for lustre. In: **Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis**. New York, NY, USA: Association for Computing Machinery, 2019. (SC '19). ISBN 9781450362290. Available from Internet: <https://doi.org/10.1145/3295500.3356139>.

REW, R.; DAVIS, G. Netcdf: an interface for scientific data access. **IEEE Computer Graphics and Applications**, v. 10, n. 4, p. 76–82, 1990.

RODRíGUEZ-QUINTANA, C. et al. A new scalable approach for distributed metadata in hpc. In: . [s.n.], 2016. v. 10048, p. 106–117. ISBN 978-3-319-49582-8. Available from Internet: <https://doi.org/10.1007/978-3-319-49583-5_8>.

SAINI, S. et al. I/o performance characterization of lustre and nasa applications on pleiades. In: **2012 19th International Conference on High Performance Computing**. [s.n.], 2012. p. 1–10. Available from Internet: <https://doi.org/10.1109/HiPC.2012.6507507>.

SIMMONS, J. S. et al. Lustre distributed name space (dne) evaluation at the oak ridge leadership computing facility (olcf). **ORNL/TM–2015/608**, 2016. Available from Internet: <https://lustre.ornl.gov/ecosystem-2016/documents/papers/LustreEco2016-Simmons-DNE.pdf>.

SIVALINGAM, K. et al. Lassi: Metric based i/o analytics for hpc. In: **2019 Spring Simulation Conference (SpringSim)**. [s.n.], 2019. p. 1–12. Available from Internet: <https://doi.org/10.23919/SpringSim.2019.8732903>.

SKINNER, D.; KRAMER, W. Understanding the causes of performance variability in hpc workloads. In: **IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.** [s.n.], 2005. p. 137–149. Available from Internet: <https://doi.org/10.1109/IISWC.2005.1526010>.

SPEARMAN, C. The proof and measurement of association between two things. **The American Journ. of Psychology**, University of Illinois Press, v. 15, n. 1, p. 72–101, 1904.

SUGIYAMA, S.; WALLACE, D. Cray dvs: Data virtualization service. In: **Cray User Group Annual Technical Conference**. [S.l.: s.n.], 2008.

THAKUR, R.; GROPP, W.; LUSK, E. On implementing mpi-io portably and with high performance. In: **Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems**. New York, NY, USA: Association for Computing Machinery, 1999. (IOPADS '99), p. 23–32. ISBN 1581131232. Available from Internet: <https://doi.org/10.1145/301816.301826>.

VISHWANATH, V. et al. Accelerating i/o forwarding in ibm blue gene/p systems. In: **SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis**. [S.l.: s.n.], 2010. p. 1–10.

WADHWA, B. et al. iez: Resource contention aware load balancing for large-scale parallel file systems. In: **2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)**. [s.n.], 2019. p. 610–620. Available from Internet: <https://doi.org/10.1109/IPDPS.2019.00070>.

WANG, K. et al. Exploring the design tradeoffs for extreme-scale high-performance computing system software. **IEEE Transactions on Parallel and Distributed Systems**, v. 27, n. 4, p. 1070–1084, 2016. Available from Internet: <https://doi.org/10.1109/TPDS.2015.2430852>.

WANG, Z. et al. Various frameworks and libraries of machine learning and deep learning: A survey. **Archives of Computational Methods in Engineering**, Feb 2019. ISSN 1886-1784.

WARTENS, C. M. H.; GARLICK, J. **lmt: Lustre Monitoring Tool**. 2010. Available from Internet: <https://github.com/LLNL/lmt>.

XIE, B. et al. Characterizing output bottlenecks in a supercomputer. In: **SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis**. [s.n.], 2012. p. 1–11. Available from Internet: <https://doi.org/10.1109/SC.2012.28>.

YILDIZ, O. et al. On the root causes of cross-application i/o interference in hpc storage systems. In: **2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)**. [s.n.], 2016. p. 750–759. Available from Internet: <https://doi.org/10.1109/IPDPS.2016.50>.

YU, J. et al. Cross-layer coordination in the i/o software stack of extreme-scale systems. **Concurrency and Computation: Practice and Experience**, v. 30, n. 10, p. e4396, 2018. E4396 cpe.4396. Available from Internet: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4396>.

ZHANG, X.; JIANG, S. interferenceremoval: Removing interference of disk access for mpi programs through data replication. In: **Proceedings of the 24th ACM International Conference on Supercomputing**. New York, NY, USA: Association for Computing Machinery, 2010. (ICS '10), p. 223–232. ISBN 9781450300186. Available from Internet: <https://doi.org/10.1145/1810085.1810116>.

ZHANG, X. et al. ibridge: Improving unaligned parallel file access with solid-state drives. In: **2013 IEEE 27th International Symposium on Parallel and Distributed Processing**. [s.n.], 2013. p. 381–392. Available from Internet: <https://doi.org/10.1109/IPDPS.2013.21>.

ZHAO, D. et al. High-performance storage support for scientific applications on the cloud. In: **Proceedings of the 6th Workshop on Scientific Cloud Computing**. New York, NY, USA: Association for Computing Machinery, 2015. (ScienceCloud '15), p. 33–36. ISBN 9781450335706. Available from Internet: <https://doi.org/10.1145/2755644.2755648>.

APPENDIX A — RESUMO EXPANDIDO

Supercomputadores, com centenas a milhares de nós computacionais, dominam os ambientes de Processamento de Alto Desempenho (PAD). Esses sistemas de PAD são utilizados para resolver os mais diversos problemas em vários domínios da ciência: biologia, química, física e ciências da saúde. Os pesquisadores das diferentes áreas utilizam uma infinidade de aplicações científicas, onde cada uma possui diferentes requisitos. Por exemplo, as aplicações podem ser seriais ou paralelas, e ler ou escrever diferentes quantidades de dados em vários formatos e tamanhos. Esse cenário faz com que os supercomputadores precisem lidar com cargas de trabalho mistas.

A evolução dos chips de processamento e das redes de alta velocidade permitiu aos supercomputadores processar enormes *datasets*. Além disso, a infraestrutura que armazena esses *datasets* também precisa fornecer acesso de alto desempenho para que as aplicações possam realizar suas operações de entrada e saída (E/S) de forma eficiente. Para um ambiente de HPC, não é apenas a quantidade de FLOPS que afeta o desempenho, mas também a quantidade de dados por segundo que eles conseguem efetivamente ler e escrever no sistema de armazenamento.

Os Sistemas de Arquivo Paralelos (SAPs), um sistema de armazenamento descentralizado onde máquinas dedicadas atuam como servidores de dados que reduzem a sobrecarga do processamento das requisições de E/S, são os os principais tipos de sistemas de arquivo utilizados em PAD. O Lustre (MICROSYSTEMS, 2007) é o SAP mais adotado em sistemas PAD, representando $\approx 30.5\%$ dos sistemas de arquivo presentes na lista do IO500 [1]. Apesar dos avanços nas arquiteturas de armazenamento de dados fornecerem melhor desempenho, como a utilização de dispositivos SSD, ainda há uma diferença considerável de desempenho entre o quão rápido o sistema consegue lidar com as operações de E/S e quão rápido ele consegue processar os dados. Essa diferença afeta como os supercomputadores conseguem ser utilizados produtivamente para novas descobertas científicas. Mais pesquisas estão sendo realizadas com a rápida expansão dos supercomputadores enquanto gerando mais dados a serem lidos e escritos, tornando o sistema compartilhado de armazenamento de dados um dos principais gargalos para atingir um desempenho sustentável. O SAP não é capaz de continuar fornecendo desempenho devido ao aumento da concorrência e interferência (YILDIZ et al., 2016; YU et al., 2018). Além das operações de E/S, outro importante fator no gerenciamento do armazenamento para PAD são as op-

[1] https://io500.org/

erações de metadados, que são responsáveis por manter a árvore de diretórios do sistema de arquivo, permissões de acesso aos arquivos, *timestamps*, atributos etc. Conforme os *datasets* ficam cada vez maiores, o desempenho dos metadados se torna um fator crítico e pode rapidamente se tornar um gargalo (ALAM et al., 2011).

Lawrence et al. (2017) e Saini et al. (2012) demostram que diferentes aplicações científicas têm seu desempenho impactado em diversas maneiras pelo Lustre, com algumas utilizando os recursos de forma mais eficiente do que outras. Essa variação está ligada à requisitos de carga de trabalho específicos e o Lustre tendo que lidar simultaneamente com várias aplicações sob contenção. Alguns fatores que impõe limitações e afetam negativamente o desempenho do Lustre são padrões de acesso desalinhados (BARTZ et al., 2015), carga desbalanceada entre os servidores de armazenamento (PATEL et al., 2019), e contenção por recursos (NEUWIRTH et al., 2017). Além disso, é necessário considerar que a pilha de E/S existente expõe uma infinidade de parâmetros ajustáveis, direcionados a fornecer melhorias no desempenho das aplicações. Entretanto, a má configuração de tais parâmetros devido ao usuário não conhecer o comportamento das operações de E/S da sua aplicação pode contribuir para o baixo desempenho observado. Além do mais, uma aplicação que esteja apresentando desempenho ineficiente pode também afetar negativamente todas as outras em execução concorrentemente no sistema, pois o armazenamento é compartilhado entre todas elas.

O objetivo do presente estudo é entender o impacto e identificar as necessidade do armazenamento de dados em um supercomputador através da avaliação do desemepnho do sistema de arquivos Lustre em relação à variadas cargas de trabalho de E/E e metadados, provenientes de diferentes domínios. O comportamento da utilização do Lustre foi estudado, realizando uma comparação entre dois período compreendendo três meses de operação de dois anos: de Março a Maio de 2020 e 2021, quando foi movimentado $16,50$ PiB de dados através de $109,79$ bilhões de operações de E/S. O supercomputador *Santos Dumont* (*SDumont*) [2] foi utilizado como um estudo de caso porque pouco é conhecido sobre o impacto do seu sistema armazenamento e a configuração da pilha de E/S, utilizados pelo conjunto de aplicações executadas diariamente. O trabalho apresenta as seguintes contribuições:

- Foi desenvolvida uma metodologia para coletar, analisar e visualizar dados de E/S provenientes do sistema de arquivos paralelo, utilizando softwares *open-source* que não necessitam de privilégios administrativos para serem instalados e utilizados,

[2] https://sdumont.lncc.br/

facilitando sua implementação e reprodução.

- Foi desenvolvida uma aplicação web para simplificar a visualização e análise dos dados de utilização do sistema de arquivos paralelo. Essa ferramenta facilita a reprodução da análise e o estudo de diferentes períodos de interesse.
- Foi investigado a carga de trabalho de E/S e o comportamento da utilização dos Dispositivos de Armazenamento de Objetos do Lustre, e dos Nós Computacionais do SDumont. O estudo demonstrou que a demanda por carga de trabalho não é dominada por um único tipo de operação e pode variar significativamente através do período.
- A análise individualizada dos dispositivos de armazenamento de objetos do Lustre demonstrou haver um desbalanceamento de carga considerável através deles durante a operação normal do sistema.
- Através do cruzamento das métricas de utilização de E/S dos nós computacionais com informações provenientes do sistema gerenciador de *jobs* do SDumont, foi possível identificar aplicações problemáticas que poderiam levar a degradação geral do desempenho dos servidores do sistema de arquivos paralelo.
- A análise e caracterização das operações de metadados mostrou que há uma demanda considerável por esse tipo de operação, com os metadados sendo responsáveis por 60% de todas as operações do sistema de arquivos.