Access Patterns and Performance Behaviors of Multilayer Supercomputer I/O Subsystems under Production Load

Jean Luca Bez*
jlbez@lbl.gov
Lawrence Berkeley National Laboratory
Berkeley, California, USA

Ahmad Maroof Karimi*
karimiahmad@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Arnab K. Paul*†
arnabp@goa.bits-pilani.ac.in
Oak Ridge National Laboratory, USA
BITS Pilani, K K Birla Goa Campus, India

Bing Xie*
xieb@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Suren Byna
sbyna@lbl.gov
Lawrence Berkeley National Laboratory
Berkeley, California, USA

Philip Carns
carns@mcs.anl.gov
Argonne National Laboratory
Illinois, USA

Sarp Oral
oralhs@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Feiyi Wang
fwang2@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Jesse Hanley
hanleyja@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

ABSTRACT

Scientific computing workloads at HPC facilities have been shifting from traditional numerical simulations to AI/ML applications for training and inference while processing and producing ever-increasing amounts of scientific data. To address the growing need for increased storage capacity, lower access latency, and higher bandwidth, emerging technologies such as non-volatile memory are integrated into supercomputer I/O subsystems. With these emerging trends, we need a better understanding of the multilayer supercomputer I/O systems and ways to use these subsystems efficiently. In this work, we study the I/O access patterns and performance characteristics of two representative supercomputer I/O subsystems. Through an extensive analysis of year-long I/O logs on each system, we report new observations in I/O reads and writes, unbalanced use of storage system layers, and new trends in user behaviors at the HPC I/O middleware stack.

CCS CONCEPTS

• **Information systems** → **Information storage systems**; *Storage architectures*; *Computing platforms.*

KEYWORDS

Access Patterns, Parallel File Systems, In-System Storage, High-Performance Computing, Production System

*Jean Luca Bez, Ahmad Maroof Karimi, Arnab K. Paul, and Bing Xie, placed in alphabetic order, are the lead authors and contributed equally to this work.

†Arnab K. Paul conducted the majority of this work during his postdoctoral research at Oak Ridge National Laboratory, USA. He is presently working as an assistant professor at BITS Pilani, Goa, India.

HPDC '22, June 27-July 1, 2022, Minneapolis, MN, USA

ACM ISBN 978-1-4503-9199-3/22/06.
https://doi.org/10.1145/3502181.3531461

ACM Reference Format:
Jean Luca Bez, Ahmad Maroof Karimi, Arnab K. Paul, Bing Xie, Suren Byna, Philip Carns, Sarp Oral, Feiyi Wang, and Jesse Hanley. 2022. Access Patterns and Performance Behaviors of Multilayer Supercomputer I/O Subsystems under Production Load . In *Proceedings of the 31st Int'l Symposium on High-Performance Parallel and Distributed Computing (HPDC '22), June 27-July 1, 2022, Minneapolis, MN, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3502181.3531461

1 INTRODUCTION

In High-Performance Computing (HPC) platforms, storage subsystems have evolved rapidly in recent years. In particular, the traditional HPC storage has emphasized throughput for bulk-synchronous writes in order to accommodate numerical modeling and simulation applications that perform checkpointing to periodically store their state [29, 31]. With increasing investments in using HPC capabilities for artificial intelligence (AI) and machine learning (ML) across numerous domain sciences (e.g., earth science, astrophysics), I/O workloads in production HPC environments are becoming more read-intensive with less predictable access patterns. With the increasing performance gap between the processing speed of computing nodes and the I/O throughput of parallel file systems, new storage system layers, often built with flash or SSD technologies, are being integrated into the supercomputer I/O hierarchy.

These changes in I/O workloads and storage architectures require a better understanding of storage subsystem usage in supercomputers. In this study, we characterize the I/O workloads and user behaviors of two such systems: Summit and its storage subsystem housed at the Oak Ridge Leadership Computing Facility (OLCF), and Cori and its storage subsystem housed at the National Energy Research Scientific Computing Center (NERSC). Both of these storage subsystems consist of two layers: a parallel file system (PFS) layer and an in-system storage layer (discussed in §2.1). For both storage subsystems, the PFS layers provide the required capacity and throughput for traditional HPC workloads. In contrast, the in-system storage layers provide lower I/O access latencies and higher I/O throughput for emerging AI/ML workloads. These two systems are representative of the diversity of production HPC systems since

I/O Analysis Efforts	Study Objective				I/O Logs		
	A.A.P.	S.A.P.	P.A.	M.S.	A.L.	S.L.	S.D.
[6], [13], [20]	✓		✓		✓		
[10], [19], [22]		✓	✓			✓	
[9], [28]	✓	✓			✓	✓	
[7]	✓	✓	✓		✓	✓	
[11]	✓	✓	✓		✓	✓	✓
Our study	✓		✓	✓	✓		

Table 1: Comparison of existing HPC I/O characterization efforts and our study. We compare the studies with two criteria, including study objectives — application-level access patterns (A.A.P.), system-level access patterns (S.A.P.), performance analysis (P.A.), multilayer I/O subsystems (M.S.), and the I/O logs in use — application-level logs (A.L), system-level logs (S.L.), and sampling data (S.D.).

they employ different software and hardware technologies at each layer. A better understanding of the target systems will lead to more efficient use of existing multilayer storage and help guide the design of future storage architectures.

Several studies reported in the literature have characterized HPC I/O workloads. We summarize these studies in Table 1 based on two criteria: (1) *study objective*, including application-level I/O access patterns (A.A.P.), system-level I/O access patterns (S.A.P.), performance analysis (P.A.), and multilayer subsystems (M.S.); and (2) *profiling data*, including application-level logs (A.L.), system-level logs (S.L.), and sampling data (S.D.). A few studies [9, 10, 19, 28] characterized the behaviors of application I/O, files, and supercomputer I/O subsystems from the logs of I/O profiling tools running on the file system stack, such as metadata servers and storage targets. For example, TOKIO [11] is an I/O framework that characterizes the performance behaviors and variations via periodic sampling with representative HPC I/O benchmarks. Other studies analyzed application-side data. For instance, logs from Darshan [2], a lightweight I/O monitoring tool that captures application I/O behavior, were used to study I/O usage of a single storage layer [6, 7, 13, 20].

In contrast to the previous studies, we analyze two representative multilayer storage subsystems based on two separate year-long Darshan log collections. Our investigation focuses on understanding HPC I/O workloads and the usage/performance of separate storage layers and various I/O interfaces (e.g., MPI-IO, POSIX, and STDIO) in the HPC I/O middleware stack.

To the best of our knowledge, this study is the *first to characterize multilayer supercomputer I/O subsystems in-depth and is the first to analyze platform-wide STDIO for usage and performance.* A previous study hinted at the increasing use of STDIO in HPC workloads [13] because many genomics and biology production applications rely on I/O functions used to store sequencing information in text format. As a result, a new capability was added to Darshan to instrument STDIO functions (e.g., `fscanf()` and `fprintf()`). Our study includes an analysis on the STDIO usage in detail using this new profiling information.

We publish a month's worth of Darshan logs collected from both Summit[1] and Cori[2] to promote interest and research in the HPC I/O community. We summarize our primary findings below:

[1]Summit Dataset: https://doi.ccs.ornl.gov/ui/doi/384, DOI: 10.13139/OLCF/1865904
[2]Cori Dataset: https://doi.org/10.5281/zenodo.6476501, DOI: 10.5281/zenodo.6476501

(A) For the two systems we studied, the I/O workloads present different trends of read and write operations: Summit's two layers show opposite dominance on reads and writes, whereas Cori is dominated by reads, indicating the prevalence of more diverse and complex application I/O in HPC. We believe this new trend might be related to the evolution of scientific codes from numerical simulations to AI/ML and the shift from the heavy use of MPI-IO and POSIX to STDIO. More importantly, this diversity and complexity demand automatic and dynamic management within I/O middleware libraries or the availability of ad hoc on-demand file systems studied and tuned for each type of I/O workload.

(B) End users behave similarly on the target systems. We found that small data transfers still dominate HPC I/O workloads at both the file and process levels, despite optimization techniques such as data aggregation and adaption being available for quite some time. This finding suggests ways to seamlessly perform the aggregation at the middleware level (e.g., within higher-level I/O libraries or MPI-IO) without imposing it on end users.

(C) Both target systems utilize parallel file systems much more frequently than the in-system storage layer. Moreover, for both systems, the overwhelming majority of the files on parallel file systems could be staged to use the in-system layers for higher I/O throughput during application execution. This finding suggests a great need for more convenient data-staging tools to move data between the two layers with less involvement from end users.

(D) Although limited information on STDIO has been collected by Darshan, we observe that for both systems, the usage of STDIO is noticeably high on the in-system layers and surprisingly widespread across diverse science domains. When this observation is considered in conjunction with the flash and SSD devices used in the in-system layer and the recent evolution of scientific codes, it suggests the need for more detailed statistics from I/O monitoring tools, such as Darshan, to gather information about SSD-oriented access patterns (e.g., static/dynamic data, rewrites) beyond the traditional uses of HPC interfaces and parallel file systems. It also suggests that optimization techniques, such as separating static/dynamic data and caching rewrites, should be considered by I/O middleware libraries (e.g. HDF5 [26]) or in-system layer software (e.g. DataWarp [4]).

(E) With few exceptions, STDIO interfaces consistently deliver lower performance than do POSIX interfaces across transfer sizes, suggesting a lack of optimization in STDIO methods. Given our current relative lack of insight into STDIO usage, we need a deeper understanding of application needs in this space to identify the most promising opportunities for optimization. Thus, we suggest that process-level I/O characterizations for STDIO be added to I/O monitoring tools such as Darshan.

These findings are relevant to the HPC storage community as they draw attention to new trends in I/O workloads and HPC I/O middleware usage, thereby helping identify research gaps and the need to better characterize evolving user behaviors. We expect our findings on the new read and write load trends will be interesting to the system administrators at HPC facilities for the deployment and operation of storage subsystems. Moreover, our findings on the increased use of STDIO will be useful for enhancing I/O monitoring tools and optimizing high-level I/O libraries.

This paper is organized as follows. Section 2 presents the background information. Section 3 describes the analyses we conduct. Related work is reviewed in Section 4. We summarize our findings and conclusions in Section 5.

2 BACKGROUND

This work studies the behaviors of representative multilayer supercomputer I/O subsystems. In this section we present background information about the two target I/O subsystems of Summit (§2.1.1) and Cori (§2.1.2) and about the basics of the datasets collected by a user-level I/O characterization tool, Darshan (§2.2).

2.1 Supercomputer I/O Subsystems

2.1.1 Summit I/O Subsystem at OLCF. Figure 1a presents the Summit supercomputer and its two-layer I/O subsystem. As one of the fastest supercomputer in the world [27], Summit is a 148.8-petaFLOPS IBM-built supercomputer housed at OLCF. It consists of 4,608 AC922 compute nodes, each node equipped with two IBM POWER9 CPUs and 6 NVIDIA V100 GPUs. The I/O subsystem of Summit [17] is composed of two distinct layers: an *in-system layer* and a *center-wide parallel file system layer*. In this work the in-system layer refers to a collection of organized storage devices deployed within a supercomputer. In general, the devices in an in-system layer can be node local, rack local, or system local.

Residing within the supercomputer, the in-system storage layer of Summit, named *SCNL*, is built on distributed compute node-local NVMe devices and provides an aggregate of 7.4 PB raw capacity, with 26.7 TB/s and 9.7 TB/s as the peak read and write bandwidths, respectively. The other storage layer of Summit, named Alpine, is a center-wide parallel file system, built on IBM Spectrum Scale and utilizing the GPFS parallel file system software technology. Alpine provides roughly 250 PB of usable storage capacity and 2.5 TB/s peak I/O bandwidth.

Like SCNL in Summit, in-system storage deployments have recently emerged in supercomputing facilities, offering a cost-effective solution to meet the increasing performance demands in HPC I/O across science domains. In particular, equipped with software solutions such as Spectral [18] and UnifyFS [15], these in-system storage deployments provide individual supercomputer jobs an exclusively accessed filesystem namespace during the job lifetime.

Compared with SCNL, Alpine, the parallel file system layer of Summit, is a single POSIX namespace and deployed as a GPFS file system, where each Summit node runs a GPFS client software stack and provides I/O services for both metadata and data. At the file system side, Alpine comprises 154 GPFS Network Shared Disk (NSD) servers managing file data in parallel. To achieve parallel I/O for a file, GPFS first partitions the file data into a sequence of equal-size data blocks (GPFS block) and then distributes the block sequence across an NSD sequence in a round-robin way. The NSD sequence starts from a randomly chosen NSD server and may span over the entire server pool by following the system-configured NSD order. In Alpine, the GPFS block size is configured as 16 MB.

2.1.2 Cori I/O Subsystem at NERSC. Figure 1b depicts Cori and its two storage layers. Cori is a 30-petaFLOPs Cray XC40 supercomputer housed at NERSC. It is composed of 2,388 Intel Xeon Haswell processor nodes and 9,688 Intel Xeon Phi Knights Landing compute nodes. Similar to the I/O subsystem of Summit, Cori is connected to two storage layers: an in-system storage layer, called Cori Burst Buffer or CBB, and a parallel file system layer, called Cori Scratch.

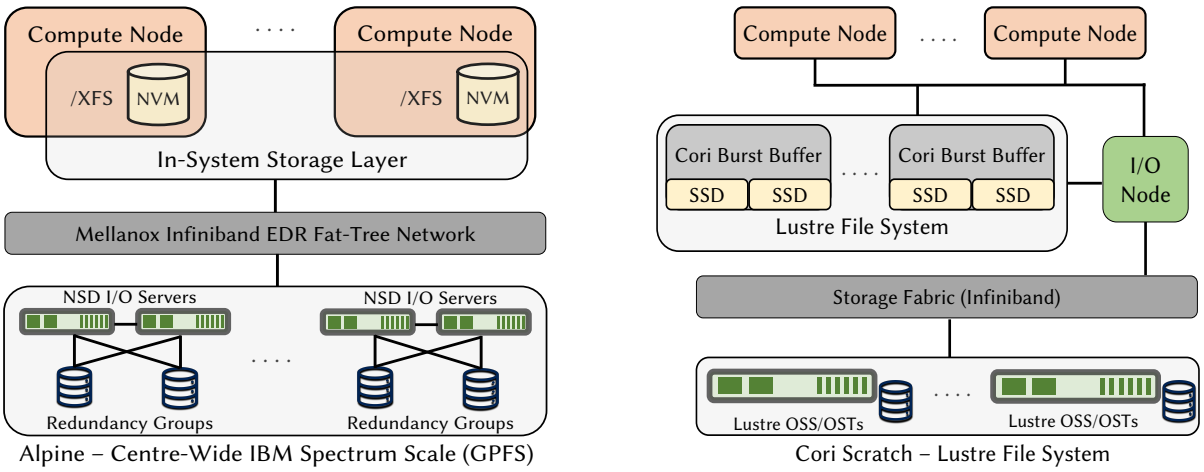
CBB provides an aggregate of 1.8 PB raw storage capacity and 1.7 TB/s peak I/O bandwidth. Different from the in-system layer of Summit that consists of node-local NVMe devices (§2.1.1), CBB is a system-local deployment with the flash devices attached to the Cray XC service nodes (also called burst buffer nodes) within Cori. Built on the Cray DataWarp software [4], CBB provides each Cori job an exclusively accessed file system namespace and storage system services during the job lifetime. Beyond the storage system services, CBB is also configured to integrate with Cori's batch job scheduler to further improve the usability of this layer. Through this integration, end users can define directives in their job description files to dictate the configurations (e.g., data striping, storage capacity) and I/O operations (e.g., staging operations, access mode) of their allocations on CBB. In addition to operating I/O within CBB, these directives can be used to arrange the staging operations between the two storage layers of Cori, enabling end users to stage directories and files in/out CBB before a job starts and/or after a job exits without user involvement. Clearly, compared with the in-system layer of Summit SCNL, CBB offers richer file system services and better usability to users. We compare the user behaviors in these two in-system deployments in §3.2.

The parallel file system layer, Cori Scratch, is a Lustre file system with 30 PB of usable disk space and 700 GB/s peak I/O bandwidth. Similar to the GPFS deployment of Summit, in Cori Scratch, each compute node is configured as a Lustre client that invokes the local Lustre kernel modules to access two file system services: object storage client and metadata client. On the system side, the Lustre file system is exposed as a single POSIX namespace with five metadata servers (MDSes) and 248 object storage servers (OSSes). Each MDS is responsible for a distinct portion of the global namespace, and each OSS manages one object storage target (OST). Each compute node is configured to connect to a single MDS and 248 OSSes/OSTs.

Unlike the GPFS data striping policy, Lustre allows users to customize the data distribution of their files. In particular, a file is partitioned into a sequence of equal-size data blocks, and each data block is distributed across a sequence of OSTs in a round-robin fashion. The block size, the length of the OST sequence, and the OST start index are the three configurable parameters in Lustre, called *stripe size, stripe count*, and *starting OST*, respectively. On Cori, the default stripe count is 1, and the stripe size is 1 MB.

2.2 I/O Characterization with Darshan

As a lightweight monitoring tool used in HPC, Darshan is designed to characterize the I/O behaviors of scientific codes at the application level. Loaded automatically as a default module on many of the U.S. DOE's large-scale supercomputers, Darshan instruments supercomputer jobs using up to hundreds of thousands of CPUs/GPUs to monitor, collect, and summarize the statistics of I/O workloads. Darshan is generally agnostic to the application programming language and programming model. In addition, researchers proposed Darshan eXtended Tracing (DXT) as an extension to provide high-resolution traces for in-depth analysis of HPC I/O workloads [33].



(a) Summit **(b) Cori**

Figure 1: I/O subsystem architecture for Summit and Cori. Both systems consist of two storage layers: an in-system layer and a parallel file system (PFS) layer, where the in-system layers are compute-node local (Summit) and system-local (Cori) and the PFS layers are IBM Spectrum Scale – GPFS (Summit) and Lustre (Cori) deployments (§2.1).

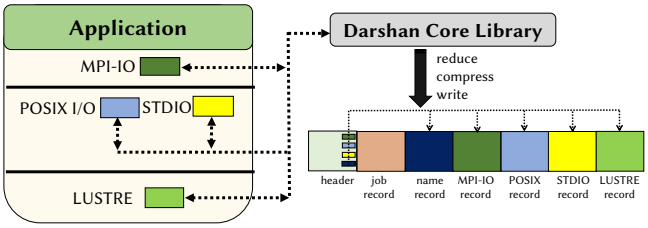


Figure 2: Interactions between the application executable, the Darshan runtime, and the self-describing log.

For the target two systems, DXT is disabled by default. Furthermore, if enabled, it only collects POSIX and MPI-IO operations, not tracing STDIO calls. Thus, in this study, we build our analysis on the logs generated by the original Darshan module automatically loaded by default (discussed in §3.1).

Figure 2 illustrates the Darshan runtime architecture, where the Darshan core library interacts with an application executable about the I/O data of the instrumentation modules (e.g., MPI-IO, POSIX). In the meantime, the library generates, registers, and stores the I/O data records for each module and writes the compressed data to a *Darshan log* when the execution completes. In Figure 2 we also show the structure of such logs. Darshan log files utilize a self-describing file format that records the statistics of I/O operations starting from when a job calls `MPI_Init` and stopping when the job calls `MPI_Finalize`. A single production job may produce multiple Darshan logs if multiple application instances are executed within the job. Moreover, each Darshan log includes execution metadata, the I/O interfaces instrumented (e.g., MPI-IO, POSIX, STDIO), and the file system in use (e.g., Lustre). In particular, at the job level Darshan records the job ID, user ID, the number of CPU/GPU cores in use, and the start and end times of the log. For an I/O interface, Darshan also records the I/O operations and the I/O metadata of each module. By merging Darshan records with scheduler logs, the project ID of the jobs can also be attained.

In Darshan, the records of an instrumentation module are organized into a set of *counters* each tracking a type of I/O operation of the module. In this paper we focus our analysis on the Darshan counters for three I/O interfaces: MPI-IO, POSIX, and STDIO. For each interface, the total read and write bytes for a file is given by `Interface_BYTES_READ/WRITTEN`, and the total time spent in read and write is provided in `Interface_READ/WRITE_TIME` counters. The total data transfer size for a file in a particular interface is calculated by adding the total read and write byte counters. The read performance of a file (bytes/second) can be computed by dividing `Interface_BYTES_READ` by `Interface_READ_TIME`. The write performance can be calculated in a similar manner using `Interface_BYTES_WRITTEN` and `Interface_WRITE_TIME` counters.

Darshan also reports the statistics of read/write requests. To show the request size per I/O system call, Darshan uses a histogram of access sizes for MPI-IO and POSIX. This shows the number of read/write requests in a size range. The values are provided in the `Interface_SIZE_READ/WRITE_SizeRange` Darshan counters. The size ranges are as follows: 0 – 100 bytes, 100 bytes – 1 KB, 1 KB – 10 KB, 10 KB – 100 KB, 100 KB – 1 MB, 1 MB – 4 MB, 4 MB – 10 MB, 10 MB – 100 MB, 100 MB – 1 GB, and greater than 1 GB. These request size counters are unavailable for the STDIO interface.

The instrumentation of STDIO functions (e.g., `fscanf()` and `fprintf()`) is a relatively new capability in Darshan, developed in response to the findings on previous platform studies [13] that confirmed the noticeably increasing use of STDIO across supercomputer platforms. In this work we drive the first effort to analyze platform-wide STDIO instrumentation in depth. Anecdotal case studies have investigated the use of STDIO routines to process text data in biology applications [23]. In this work, we observe STDIO usage becoming more common across applications, science domains, and supercomputer platforms as new scientific domains and diverse data sources embrace the use of HPC. We return to this topic with a detailed discussion in Section 3.3.

3 I/O WORKLOADS ON MULTILAYER SUPERCOMPUTER I/O SUBSYSTEMS

This section analyzes the user behaviors characterized from the I/O subsystems of Summit and Cori, representing multilayer supercomputer I/O subsystems in production use. As outlined in §2.1, both systems are connected to two separate storage system layers: the in-system storage layer, SCNL at Summit and CBB at Cori, and the parallel file system layer. In the following discussions, we use the acronyms SCNL and CBB to refer to the two in-system layers of Summit and Cori, respectively, and use the term PFS to refer to the parallel file system layer of the two target systems.

Our analysis is built on the I/O characterization data collected by Darshan (§2.2). Specifically, we focus on understanding the user behaviors on separate storage system layers (§3.2) and in using separate I/O interfaces (e.g., POSIX-IO, MPI-IO, STDIO) in the HPC I/O middleware stack (§3.3). We also discuss the I/O performance of the target systems across storage system layers and I/O interfaces (§3.4). Although our observations are limited to the two target systems, we expect that our conclusions and suggestions are generally valid and applicable to the deployments and operations of other supercomputer I/O subsystems. Furthermore, we expect that they are valuable for I/O monitoring tools (e.g., Darshan) and I/O middleware libraries (e.g., HDF5), considering the runtime I/O characterizations and performance optimizations, respectively.

3.1 Darshan Data on Summit and Cori

As a lightweight I/O characterization tool, Darshan is enabled by default on both Summit and Cori . When a job starts, the Darshan runtime is loaded automatically as a default module unless end-users unload it explicitly. Table 2 summarizes the statistics of the Darshan data covered in this study. We analyze the Darshan logs generated in the year 2020 on Summit and the year 2019 on Cori. During these years the default Darshan module on Summit was v3.1.7 and on Cori were v3.0 and v3.1.

	Year	Version	Logs	Jobs	Files	Node/hs
Summit	2020	3.1.7	7.7M	281.6K	416M	16.4M
Cori	2019	3.0/3.1	4.3M	749.5K	1,294M	45.5M

Table 2: Summary of Darshan data on both systems. M and K represent the units for million and thousand, respectively.

To summarize, this study covers ≃7.74 million Darshan logs from ≃ 281.6K jobs on Summit and ≃4.36 million logs from ≃ 749.5K jobs on Cori, representing 16.4 million and 45.5 million node-hours on Summit and Cori , respectively. On a target system, each job generates 1–34, 341 (Summit) and 1–9, 999 (Cori) Darshan logs. Over the period of this study, ≃ 1, 294.85 million and ≃ 416.91 million *unique files* were collected from Summit and Cori, respectively. In this study we consider a file as a unique file if it can be uniquely identified by the combination of its path and name in a single Darshan log. We do not correlate files across Darshan logs and jobs.

Unlike previous studies that centered on the analysis of single-layer parallel file systems [1, 25, 30, 32, 33], we analyze user behaviors and performance on multilayer storage systems. For each layer of a target system, we analyze the Darshan counters (§2.2) on *data transfer size per file* and *data request size per process*.

In particular, the data transfer size of a file means the aggregate data size for read or write on a file in the lifetime of a Darshan log. When Darshan observes that a file is accessed by MPI-IO or POSIX, we analyze the data transfer size of POSIX since the information collected at the POSIX level reflects the actual I/O interactions between applications and the underlying file systems. The reason is that when end users choose MPI-IO to manage data transfer, MPI-IO will initiate POSIX system calls to access the files managed by the POSIX-compliant file systems such as the two PFSes discussed in this work. Moreover, when Darshan observes that a file is managed by STDIO, we analyze the corresponding data transfer on STDIO. In summary, we collect ≃ 202.18 PB (read) and ≃ 8, 280.74 PB (write) of file data transfer on Summit and ≃ 172.31 PB (read) and ≃ 24.3 PB (write) of file data transfer on Cori, respectively.

In addition to file data transfer, we analyze the read/write requests generated by individual processes, where each process performs on a different accelerator (e.g., CPU, GPU). In particular, for a job executable running on one or more coordinated processes, we analyze the read/write requests generated per process using the Darshan counters for the bins of request sizes (discussed in §2.2). Similar to the analysis of file data transfer, we focus on the I/O request information collected from POSIX and STDIO.

3.2 User Behaviors on Storage System Layers

This subsection discusses the application-level I/O characteristics on separate storage system layers, including the analyses of I/O access patterns (§3.2.1) and the usage of separate I/O layers (§3.2.2). Our goal is to build an in-depth understanding of the user behaviors across I/O layers and seek opportunities to improve the end-to-end I/O performance for users while at the same time maximizing I/O system utilization in HPC facilities.

3.2.1 I/O access patterns. We first look into the number of files and the total file data transfer size on the separate storage system layers. We report the results in Table 3.

To summarize, on both platforms the total number of file accesses to PFSes is 3.63× (Summit) and 28.87× (Cori) as is observed on SCNL and CBB. Similarly, for total file data transfer size, we observed 44.63× (read) and 3077.34× (write) of the volume in the PFS when compared with SCNL for Summit. For Cori, the total transfer size on the PFS is 28.86× (read) and 6.01× (write) of the CBB. These numbers suggest that for both of the target systems, a large group of end users choose PFSes over the in-system layers, although the latter layers provide much better performance. To better understand users' motives behind this choice, we conducted a separate analysis on the usage of storage layers and the science domains in §3.2.2.

Besides confirming the high popularity of the PFSes, we notice that on Summit and Cori the I/O loads present different trends. For Summit , SCNL and PFS are dominated by read and write, respectively, whereas the entire I/O subsystem of Cori is dominated by read loads. In particular, for the two I/O layers on Cori, reads are 3.16× (CBB) and 6.58× (PFS) of the write loads. Relative to the new trends of utilizing supercomputer capabilities for AI/ML and edge computing and the rapidly growing use of the other I/O interfaces (e.g., STDIO) across science domains (discussed in §3.3),

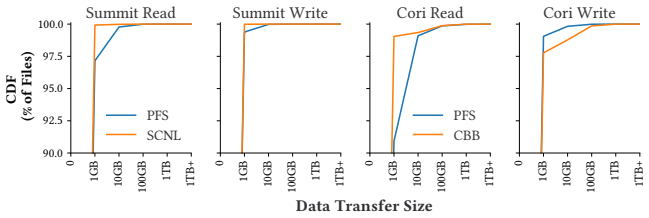


Figure 3: Cumulative distribution functions (CDF) for the number of files in a transfer-size bin for read (left) and write (right) in Summit and Cori's in-system layer and PFS. We define data transfer size in §3.1.

Figure 4: Cumulative distribution functions (CDF) for the number of requests in a given Darshan bin size for reads (left) and writes (right) in Summit and Cori's in-system layer and the parallel file system.

		Files (Mil.)	**read** (PB)	**write** (PB)
Summit	**SCNL**	279.39	4.43	2.69
	PFS	1,015.46	197.75	8278.05
Cori	**CBB**	13.96	13.71	4.34
	PFS	402.95	171.64	26.10

Table 3: Number of files and the total data transfer size on the storage layers of Summit and Cori. For file read/write using MPI-IO or POSIX, we collect the data transfer size in POSIX. For file read/write using STDIO, we collect the data transfer size in STDIO (discussed in §3.1).

		read files	**write** files
Summit	**SCNL**	0	0
	PFS	7232	78
Cori	**CBB**	513	950
	PFS	74	10,045

Table 4: Number of read/write files in different layers of the I/O stack of Summit and Cori where we observed total data transfer size > 1 TB.

this observation indicates that, with the evolution of scientific codes from simple numerical simulations to AI/ML training and inference, the HPC I/O workloads are becoming more diverse and complex.

The deeper HPC I/O hierarchy and the emergence of software/hardware technologies (§2.1.1 and §2.1.2) are making it more challenging for end users to efficiently use the various I/O resources on their own. Instead, HPC I/O middleware libraries, such as HDF5 [26], PnetCDF [8], and ADIOS [12], are designed to manage I/O operations on behalf of users across layers of I/O stack. Unfortunately, so far, all of the I/O libraries provide only simple heuristics as the defaults and allow users to customize the multilayer configurations on their own. Nevertheless, promisingly, these I/O libraries can collect the specific I/O access patterns at application runtime [24, 25] and accordingly do make it possible to address the diverse and complex HPC I/O patterns automatically and dynamically.

Next, we take a closer look at the characteristics of the I/O workloads on the two target systems. Figures 3 and 4 present the cumulative distribution functions (CDFs) of data transfer size per file and data request size per process (defined in §3.1). In particular, we observe that for Summit 97% of file reads and 99% of file writes are below 1 GB on PFS, whereas for SCNL 99% of both file reads and writes are below 1 GB. Moreover, for the analysis of I/O request size per process, on the PFS of Summit, both 0–100 and 1K–10K request-size ranges represent about 45% of read calls. For SCNL of Summit, the 10K–100K request-size range represents an even higher number of read and write calls covering 83% and 60%, respectively. Similarly, on Cori, 99.04% of file reads and 97.77% of file writes in CBB have a total transfer size of < 1 GB. Considering the PFS, 99.05% of file reads and 90.91% of file writes have a total transfer size of < 1 GB.

We also highlight large jobs (i.e., number of processes >1,024) in Figure 5. We observe the same trend in request sizes to the PFS in both systems, indicating that the initially reported results are not due to a lot of small jobs but rather a system-level trend also visible on large-scale runs. We also notice more large requests to the in-system storage layer. Furthermore, in Summit most of the total transfers are >1 GB for writes and >100 GB for reads, whereas in Cori the majority represents a total transfer size of over 1 TB.

In summary, we confirm that on the separate storage system layers of both Summit and Cori, the HPC I/O workloads are still dominated by small data transfers on both file and process levels and for both read and write. Compared with small read and write operations, large-size I/O can obtain better performance. Moreover, data aggregation at the application level (e.g., collective I/O in MPI-IO and I/O adaptation in ADIOS) can accumulate small read/write requests and communicate with storage systems with larger requests. Thus, data aggregation in I/O middleware libraries is an effective option to attain good performance.

Although small data transfers dominate, large files indeed occur. As summarized by Table 4, on Cori, 91.35% of > 1 TB files were written to PFS, whereas 87.39% of those large accessed files were instead read from CBB; in contrast, on Summit all files greater than 1 TB were only on PFS. Again, the data confirms the users' preference for PFSes.

Recommendation (1). For the two systems we studied, the overall I/O workloads present diverse trends on read and write accesses: the Summit's two layers show opposite dominance on read and write, whereas Cori is dominated by read. Relative to the new trends in scientific applications and HPC I/O middleware stack, this data indicates the more diverse and complex application I/O. This diversity and complexity highlights the need for automatic and dynamic management in I/O middleware libraries or the availability of ad-hoc on-demand FS best suited and tuned for each workload.

Figure 5: Cumulative distribution functions (CDFs) for read (left) and write (right) requests on Summit and Cori's in-system layer and the parallel file system. The analysis is similar to Figure 4 but only for large jobs (i.e., number of processes $> 1,024$).

	SCNL/CBB	SCNL/CBB + PFS	PFS
Summit	0	3.42K	241.5K
Cori	103.46K	35.9K	579.91K

Table 5: Number of jobs on Summit and Cori that access files exclusively on PFS, SCNL/CBB, or both.

Recommendation (2). Across storage systems and for both read and write, HPC I/O workloads are dominated by small data transfers at the file level and small I/O requests at the process level. Optimization techniques such as data aggregation and adaptation at the application level (e.g., at MPI-IO) could be useful in such scenarios to achieve better performance. Despite those techniques being available for quite some time, however, we still see a widespread use of small I/O requests. Thus, we recommend strategies to perform the aggregation seamlessly at the middleware level without imposing the task on end users.

3.2.2 Usage of Storage System Layers. Seeking opportunities to improve the usage of the two in-system layers from the target systems and, accordingly, achieve higher I/O throughput for HPC users, we look into the current use of those layers.

Table 5 reports the jobs on Summit and Cori that access files exclusively on PFS, SCNL/CBB, or both. The results show that 14.38% of the Cori jobs manage files on CBB exclusively, whereas the Summit jobs rarely make such exclusive accesses on SCNL. This difference reflects the diverse design choices on the software technologies of CBB and SCNL (discussed in §2.1). In particular, DataWarp on CBB manages data staging on directories and files between CBB and PFS before a job starts and after it exits, allowing end users seamlessly to access files that reside on CBB. Instead, UnifyFS and Spectral on Summit provide similar file system functions between SCNL and PFS at application runtime, leaving a minimal number of exclusive file accesses to SCNL (e.g., temporary files generated at application runtime) during job executions.

We also categorize file data transfers on SCNL/CBB and PFSes based on the I/O operations: read-only, read-write, and write-only. Figures 6a and 6b report the results. When we focus on the files on PFSes, we surprisingly find that 95.7% (Summit) and 90.1% (Cori) of the files are either read-only or write-only. This result indicates that although the read-only and write-only files can directly benefit from the data-staging techniques in DataWarp, UnifyFS, and Spectral, the overwhelming majority of these files have not yet harnessed such techniques but instead read/write directly to the PFSes. We also find that the users of these files are distributed across a wide range of science domains with no noticeable patterns. In this paper, we report this issue as an open question we observed, hoping that this will spark further study on improving the usability of data staging between the storage layers in HPC platforms.

We also look into the users of the in-system layers on Summit and Cori. Figure 7a shows the usage of Summit's SCNL storage layer by various domains. Darshan data recorded over 3K jobs that have used the SCNL storage layer and are distributed across 9 science domains. In particular, the computer science and physics domains combined cover 60% of the jobs running on the SCNL layer. We also observe that biology and materials science have used the SCNL layer exclusively for read-only operations. In contrast, chemistry jobs have used node-local storage for write-only I/O operations.

For Cori, Figure 7b summarizes the CBB usage by domain, in log scale. One can observe its widespread usage across 12 domains and not constrained only to computer science as one might expect. Physics applications are responsible for most data transfer to this layer, 71.95%. On the other hand, Earth science and materials science applications have a high CBB read but low write usage. Although not visible, engineering, nuclear energy, and mathematics are the domains with the lowest nonzero use.

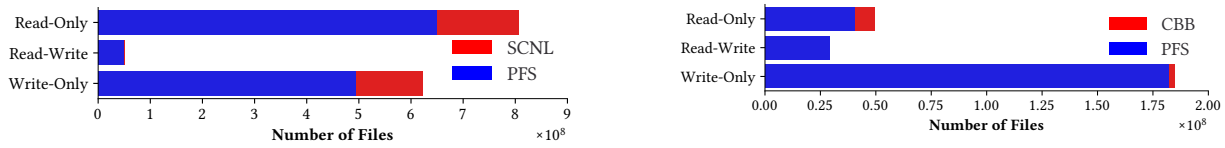
We conclude that, for both systems, the users of the in-system layer are distributed in a wide range of science domains, with no noticeable patterns or motivations for their use of this layer. We leave further study of this distribution as our future work.

Recommendation (3). Despite the low usage of the in-system layers of both target systems, the majority of the files on PFSes can be staged to use SCNL and CBB, suggesting the great need for more convenient data-staging tools that can move data between the two layers with less involvement from the end users.

3.3 User Behaviors of I/O Middleware

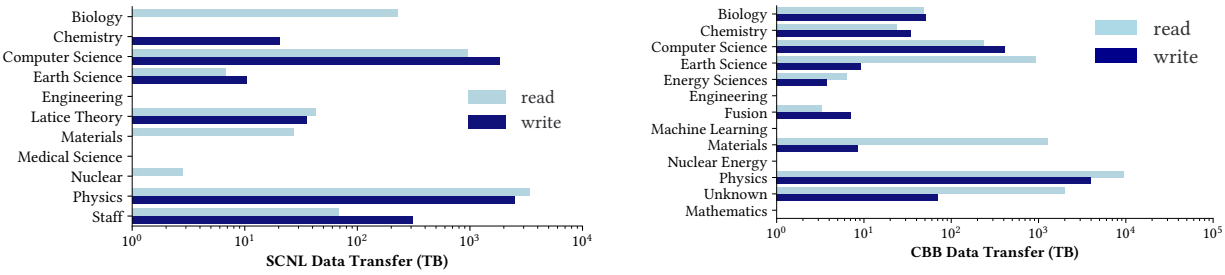
This section covers the user behaviors in using the HPC I/O middleware stack. We start from understanding the usage and access patterns of POSIX, MPI-IO, and STDIO interfaces on the separate storage system layers of Summit and Cori.

3.3.1 I/O Access Patterns. Table 6 presents the total number of files using the three I/O interfaces on the separate storage system layers. First, we confirm that POSIX is still the most popular I/O API in HPC with an overall 50.2% and 51.2% of the files on Summit and Cori being managed by it. Second, we find that MPI-IO is not favored, especially on Summit. We observe that only ≈10% of the files on Summit are managed by MPI-IO. Third, we surprisingly find that the use of STDIO is growing rapidly on both of the target systems, with an overall 39.8% (Summit) and 14.2% (Cori) of the files managed by STDIO. When looking further into the usage of I/O interfaces on separate storage system layers, even more surprisingly we observe on SCNL of Summit that the use of STDIO is even dominant, with 4.37× the use of POSIX and over 200× the use of MPI-IO, respectively. For the two storage system layers of Cori, although the combined use of POSIX and MPI-IO is still dominant on both of the I/O layers, the use of STDIO is noticeable: 14.6% of the files on the PFS of Cori is managed by it. These numbers suggest the increasing use of STDIO in HPC as a new trend.



(a) Summit (b) Cori

Figure 6: Classification of files in Summit and Cori using both POSIX and STDIO interfaces based on their usage of in-system storage or parallel file system.



(a) Summit (b) Cori

Figure 7: Usage of the in-system layers across science domains on Summit and Cori. We consider the aggregate file data transfer using both POSIX and STDIO (discussed in §3.1).

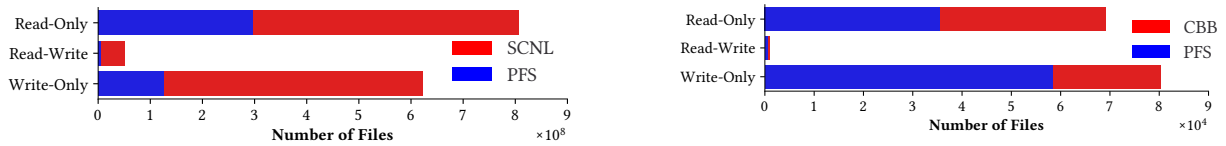
		POSIX (Mil.)	MPI-IO (Mil.)	STDIO (Mil.)
Summit	**SCNL**	52	0	227
	PFS	743	157	404
Cori	**CBB**	13	13	0.65
	PFS	313	207	89

Table 6: Files using the STDIO, MPI-IO, and POSIX I/O interfaces. For the MPI-IO usage on the SCNL layer on Summit, we observe only 6 outstanding files using MPI-IO.

To obtain a better understanding of the use of STDIO, we look into the STDIO-managed files the same way we did for the usage analysis on PFSes (discussed in §3.2.2): we categorize the STDIO-managed files based on the I/O operations on read-only, read-write, and write-only. Figure 8 summarizes the total file data transfer using only STDIO on the separate storage system layers in the target two systems. Compared with the statistics of the file data transfer using both POSIX and STDIO presented in Figures 6a and 6b, we surprisingly find that on both Summit and Cori the files managed by STDIO show much higher usage on the in-system layers than the overall statistics. In particular, on Summit the STDIO-managed files on SCNL show dominance for all of the I/O operations, with 2.66× (read-only), 13.2× (read-write), and 4.8× (write-only) more use than on PFS. On Cori, although less than half of the files on SCNL are managed by STDIO, the ratio of STDIO over POSIX on CBB is 4.2× (read-only), 23.6× (read-write), and 4.39× (write-only) the ratio on PFS, suggesting that the STDIO users on Cori utilize the in-system layer much more frequently than they do on PFS. We conclude that for the STDIO users on both systems, the use and popularity of the in-system storage systems are noticeably high.

Unfortunately, the detailed process-level information of STDIO (e.g., the operations such as `fread`/`fwrite`, the start/end timestamps of the operations) are currently not instrumented by Darshan, leaving us with limited knowledge on the low-level use of STDIO. Moreover, relative to the in-system layers built with SSD devices (e.g., NVMe) that present the unique write amplification issue [5], the characterizations of data requests at the process level are critical for these SSD-built layers because the performance and lifespan of such layers can be reduced significantly by random writes and frequent rewrite (dynamic data). Undoubtedly, careful management and optimization at the application and/or file system software levels can significantly alleviate write amplification on the in-system layer. However, it demands an in-depth understanding of the STDIO usage for the I/O behaviors at the process level (e.g., sequential/random writes, data compression, static/dynamic data) and the optimization techniques (e.g., separating static and dynamic data, reducing small and random writes, caching for rewrites) adopted by HPC users, I/O middleware libraries, or PFS software.

In adition to analyzing the use of STDIO, we shed some light on the data transfer size on the separate I/O interfaces across storage system layers, using the results collected from Summit as an example. Figure 9 presents the CDFs of file data transfer with different I/O interfaces on Summit. In summary, we confirm that following the trends on file data transfer in HPC presented in Figure 3, the file data transfer managed by STDIO is also dominated by small reads and writes for both the target platforms. In particular, on Summit and Cori, more than 98.7% (SCNL) and 100% (PFS) of the file data transfer for reads are <1 GB, and more than 82.4% (SCNL) and 97.6% (PFS) of the file data transfer for writes are <1 GB. As the analysis on Summit confirms, for STDIO-managed files, small reads and writes dominate both the in-system layer and PFS. We expect a similar report from Cori or other supercomputer I/O subsystems.



(a) Summit (b) Cori

Figure 8: File classification in Summit and Cori using only STDIO based on their usage of in-system storage or PFS.

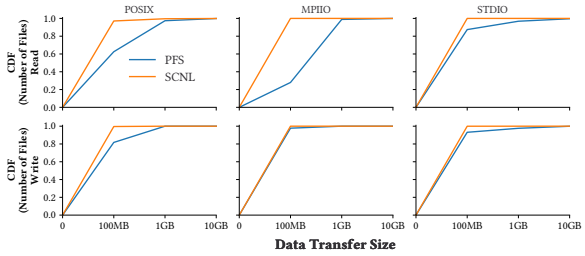


Figure 9: Overview of Summit file operations: cumulative distribution functions (CDFs) for POSIX, MPI-IO, and STDIO for file read and write.

Recommendation (4). For both systems, the usage of STDIO on the in-system storage layer is surprisingly high, suggesting that HPC users have been shifting from the dominant use of POSIX and MPI-IO to the heavy use of STDIO, especially on the SSD-built in-system layer. Considering a lack of instrumentation on the detailed process-level counters of STDIO and the unique write amplification issue of flash memory and SSDs used in the in-system layer and its high STDIO usage, we recommend that the counters of the process-level (e.g., operations on `fread`/`fwrite`, I/O request sizes and timestamps) and SSD-oriented I/O characterizations (e.g., rewrite, static/dynamic data) should be considered in I/O monitoring tools such as Darshan. Furthermore, we recommend that I/O optimization techniques (e.g., separating static/dynamic data, caching rewrites) should be supported in I/O middleware libraries.

3.3.2 Insights on STDIO Usage. Seeking to complement the discussions on the access patterns that employ STDIO to access the data, we also sought to understand its usage among running applications by categorizing them based on their science domains. The workload manager on Summit records the domain information of jobs in their scheduling logs, but Slurm on Cori does not record this information. Thus, we collected the science domain information from the NERSC NEWT API [16] for the projects that have submitted jobs in 2019 and combined it with our dataset. On Cori, each project is associated with only one science domain.

Figures 10a and 10b show the total transfer size of files accessed via the STDIO interface, grouped by science domain. From 287, 164 jobs reported by Darshan to have used STDIO operations, 258, 510 have a science domain associated with their project, that is, a 90.02% coverage. It becomes clear that multiple domains are using STDIO operations in their applications, adding up to 18.76PB of transferred data (5.94 TB written and 12.82 TB read). When we observe the combined usage per domain, as illustrated by Figure 10b, we can see that physics applications have the most intensive STDIO writes (5.43PB) and reads (12.57PB). On Summit, over 175K jobs have used the STDIO library to read or write data, representing over 62% of the jobs recorded by Darshan. Like Cori, Summit also has widespread use of STDIO across many scientific domains. In Figure 10a, we depict those domains that have a significant STDIO usage of >1 TB.

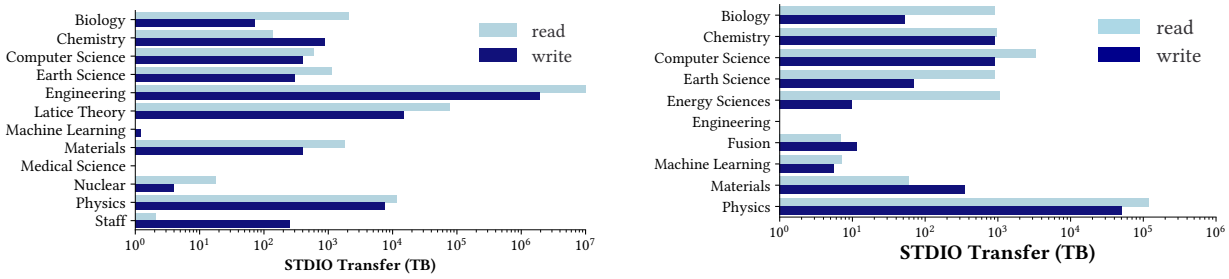
Moreover, we look into the file extensions characterized by Darshan. Although end users may change their file extensions in arbitrary ways, these extensions still give us a hint about files and data managed by STDIO. For example, in Cori we find that ≈ 70% of the STDIO-managed files present the extensions as `.rst`, `.dat`, `.vol`, indicating that the files are highly likely human-readable logs (`.rst`) and visualization data (`.dat`, `.vol`).

We conclude that on both systems, a wide range of domains actively uses STDIO, which is highly likely for managing logs and visualization data. In contrast to the initial observation on the use of STDIO [13], we see increased usage across more domains. Although the file-extension analysis gives us some insights, the domain-related analysis does not provide much more information about STDIO users. To further explain the growing and widespread use of STDIO, the system-level information about application runtime and their executables (e.g., the programming language used on data transfer, the modules loaded at application runtime) should be included for a more comprehensible study. We leave this study as our future work.

Recommendation (5). The STDIO users are distributed across a wide range of science domains in both systems, suggesting that diverse areas embrace HPC jobs. Accordingly, application runtime details and metrics should be considered to comprehend the motivations for using STDIO.

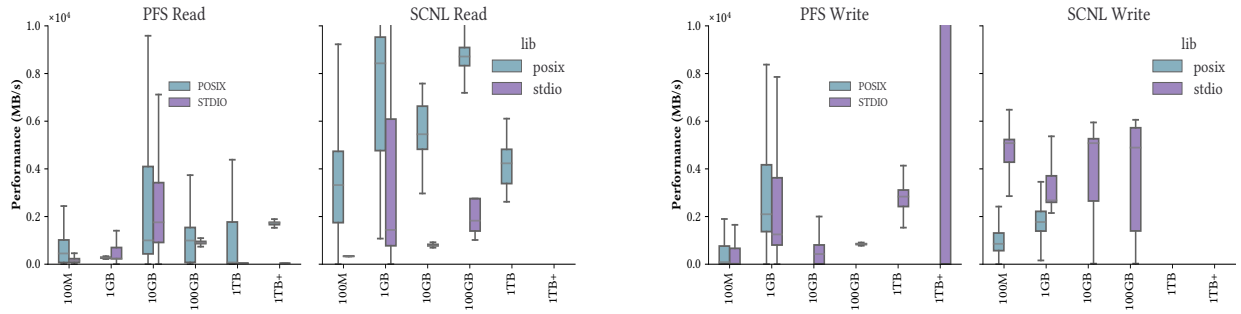
3.4 Performance Analysis

In this section we explore the performance of separate storage system layers when using different I/O interfaces. Similar to the preceding sections, we focus on the performance analysis of POSIX and STDIO (explained in §3.1). Moreover, since Darshan does not instrument process-level I/O request statistics for STDIO, to make a fair comparison between I/O interfaces, we limit our analysis to the performance of file data transfers. To conduct an accurate performance study, we further focus on the data transfer of single-shared files, where all processes participate in file read or write operations. Darshan records these file accesses as a single entry (recorded with rank -1). This restriction enables us to take into account concurrent accesses correctly. Otherwise, when a subset of the processes read/write a file, the accesses of each one will be recorded as a different entry (the first and last access only), leaving the synchronization in the process subset uncertain since we do not



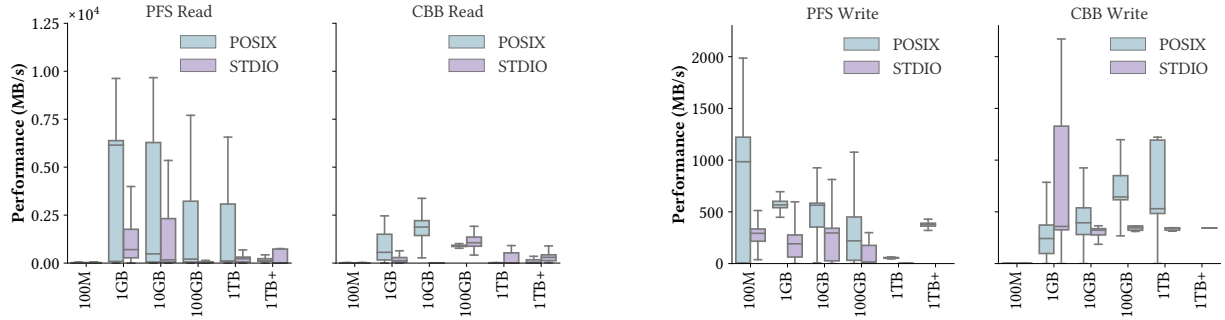
(a) Summit **(b) Cori**

Figure 10: Total transfer size for jobs with STDIO operations grouped by science domain as defined by OLCF and NERSC.



(a) Summit Read Performance **(b) Summit Write Performance**

Figure 11: Read and write performance on Summit for files using the POSIX and STDIO I/O interfaces across PFS and in-system storage layer (SCNL). Observations are grouped in bins of total data transfer size.



(a) Cori Read Performance **(b) Cori Write Performance**

Figure 12: Read and write performance on Cori for files using the POSIX and STDIO I/O interfaces across PFS and in-system storage layer (CBB). Observations are grouped in bins of total data transfer size.

have the exact timestamp of when each operation happened. It is worth noting that the performance numbers reported in this section represent the I/O performance received by individual single-shared files from individual supercomputer jobs. Considering the consistently busy supercomputers and their shared-mode I/O subsystems, the system-wide aggregate I/O bandwidths are generally higher or much higher than we report here.

Figures 11a and 11b show the performance measures on the I/O layers of POSIX and STDIO in MB/s for read and write operations to shared files, respectively. The performance for the POSIX layer, in general, is better than for the STDIO layer. I/O read operations have a more significant performance difference than the writes. Moreover, the SCNL layer has a higher performance contrast. The I/O performance of the POSIX layer for files in the range of 100 GB to 1 TB transfer size is about 40× more than for the STDIO layer. Similarly, for smaller files, those less than 100 GB, the I/O performance of the POSIX layer is 3× higher. On the SCNL layer, the median read I/O performance for files between 100 MB and 1 GB is 5× better on the POSIX layer, which increases to 8× for the files in the range of 10 GB to 100 GB. The I/O performance for

write operations in Figure 11b for the POSIX layer for files between 100 MB and 1 GB is 1.6× larger than those using STDIO. For files of other data transfer sizes, the performance is comparable. On the other hand, for the SCNL layer, the write I/O performance for STDIO is 1.5× more than POSIX for files in the range of 100 MB to 1 GB. Since some of the boxplots are missing because of the absence of files in that size range, we could not compare the write operations across some data transfer bins. Nevertheless, the plots in Figures 11a and 11b exhibit the general trend in I/O performance behavior of the POSIX and STDIO layers on Summit.

We observe similar behavior for Cori. In Figures 12a and 12b, we summarize the read and write I/O performance (in MB/s) for the shared files using POSIX and STDIO interfaces in Cori. We depict the results grouped by the total data transfer size for a given operation. Despite the natural variability, if we consider the median performance, 1 GB data transfer sizes in POSIX are 6.78× faster than STDIO. A similar trend can be seen with 10 GB transfer sizes, where the median speedup is 2.9×.

Moreover, for write requests, we observed that POSIX achieved higher performance, 3.67× for 100 MB files and 2.02× for 1 GB files (with a maximum of 8.47×) than STDIO achieved on the PFS. When considering the files written to the CBB using POSIX, larger data transfer sizes gained more performance. In summary, we observed low performance for applications that used the STDIO interface when compared with POSIX, for both Summit and Cori. Furthermore, in both systems, there were no large data transfers to shared files using STDIO except for 5 files on Summit , shown in Figure 11b for a size range greater than 1 TB (1 TB+).

Recommendation (6). In contrast to the POSIX performance, although obtaining higher I/O bandwidth on a few transfer sizes, STDIO consistently delivers lower performance across storage system layers and platforms. Given the limited STDIO metrics in Darshan logs, we emphasize that a better understanding of the users' needs and behaviors on STDIO is critical to explore performance-improvement opportunities in higher-level I/O libraries.

4 RELATED WORK

Many studies have analyzed and characterized the I/O behaviors of HPC applications and systems. We can group existing efforts based on the I/O logs used to characterize and their objectives. There are three kinds of I/O logs: application-level logs (e.g., Darshan [2]); filesystem-level logs (e.g., Lustre monitoring tool logs [14]); and data sampled from running benchmarks (e.g., IOR [21]). The analysis of such logs can, in turn, be used to study either application-level or system-level I/O access patterns, analyze or tune I/O performance, or study the multilayer I/O usage.

Application-level logs collected by Darshan [2] have been used in several studies to analyze and characterize the I/O behaviors of applications and gauge their performance. Isakov et al. [6] use 89,884 Darshan logs from the Argonne Leadership Computing Facility (ALCF) to build a clustering hierarchy of HPC jobs and a dashboard allowing a system owner or developer to gain insight into these clusters. Luu et al. [13] use Darshan I/O logs to provide a broad view of I/O behaviors on three previous-generation HPC platforms – Intrepid, Mira, and Edison. They find gaps in the adoption of best practices by scientific application developers to achieve good I/O performance. Patel et al. [20] study Darshan logs on Cori to identify reuse patterns of files in workflows. Costa et al. [3] leverage Darshan logs to identify similarities across jobs from the same application and detect potential I/O performance variability.

Other efforts, such as [10], [22], and [19], use server-side log analytics to investigate performance and study system-side file access characteristics. Liu et al. [10] build AID, which mines application-specific I/O patterns from existing supercomputer server-side I/O traffic logs and batch job history jobs. AID also makes suggestions for I/O-aware scheduling and observations about the predictable nature of HPC I/O patterns. Patel et al. [19] draw insights into the temporal, spatial, and correlative behaviors of HPC I/O by analyzing server-side I/O logs. They observe that read I/O dominates the data transfer size on the file systems while write I/O is generally bursty. Shantharam et al. [22] analyze statistics from the Lustre file system server-level collected over three years. They demonstrate a shift from write-intensive to read-heavy workloads with the prevalence of machine learning and point out how metadata servers are unbalanced, leading to reduced I/O performance.

Lim et al. [9] use daily file system metadata snapshots collected over 500 days to study the behavioral trends of 1,362 active users and 380 projects across 35 science domains. This is one of the first papers that use metadata analysis to provide application and system-level insights. The study shows the metadata overhead and the bursty I/O behavior of applications, along with a snapshot of file access behavior. Wang et al. [28] use both Lustre file system statistics and application-level profiling data to analyze the stripe pattern disconnect between system design and actual user/application behavior, which suggests the importance of features such as progressive file layout in Lustre. These two efforts use both application-level and system-level I/O logs to gauge the application and system-level I/O access patterns.

Other research efforts that analyze application, system-level file access patterns, and performance depend on both application-level and system-level logs. Kim et al. [7] collect multiple system logs and integrate those with Darshan logs to use various combinations of regression algorithms to predict the I/O performance of HPC applications. TOKIO [11] uses I/O performance probes in the form of sampled data from four I/O benchmark runs along with the Darshan and LMT logs from the Cori system to provide insights into the performance variation on production systems and motivate the need for online analysis.

Our work in this paper is the first analytical study that uses application-level Darshan logs to provide insights on the usage of different layers in multilayer I/O subsystems. Our work also shows the variability in the I/O usage patterns of two supercomputers with different parallel file systems and in-system storage layers.

5 CONCLUSIONS

To gain an in-depth understanding of the utilization and I/O workload trends of multilayer supercomputer I/O subsystems, this paper analyzed two sets of year-long application-level I/O logs collected on Summit and Cori. In particular, we studied I/O access patterns, user behaviors, usage and end-to-end performance of storage layers, and the different I/O interfaces (e.g., MPI-IO, POSIX, and STDIO). To the best of our knowledge, this paper is the first in-depth analysis

of the access patterns and performance characteristics of multilayer I/O subsystems also considering STDIO.

We summarize the observations and suggestions covering the design, deployment, and operation of I/O subsystems and the gaps we need to close in order to attain better performance by enhancing I/O monitoring tools and translating those metrics into automatic optimization in high-level I/O libraries, middleware, or parallel file system software.

Unbalanced use of storage system layers. We observed that HPC users heavily use the capacity layer PFSes on both Summit and Cori, although their in-system fast storage layers offer much higher performance. We observed that the majority of file data transfers to/from the PFSes can be staged to use the in-system layer. The under-utilization of the high-performance in-system storage and the potential for data staging between storage layers show the critical need for tools and software infrastructures that provide convenient and transparent data staging for HPC users.

Increasing use of STDIO in HPC applications. We observed that the use of STDIO is surprisingly high on both Summit and Cori. Furthermore, the STDIO-managed files also show much higher usage of the in-system layers than the other two I/O interfaces. Both these observations suggest new trends in user behaviors when using the I/O interfaces. Relative to the unique write amplification issue in the SSD-based in-system layer, we suggest that Darshan or other I/O monitoring tools add process-level counters (e.g., sequential/random access patterns) and SSD-oriented counters (data compression, static/dynamic data) for a better understanding on the optimization opportunities. We further suggest that I/O middleware libraries (e.g., HDF5) or in-system layer software (e.g., DataWarp) add features (e.g., separating static and dynamic data, reducing small and random writes, caching for rewrites) to enable SSD-oriented performance optimization.

Low I/O bandwidth delivered by STDIO. We observed that on both Summit and Cori, although STDIO obtains higher bandwidths on a few transfer sizes, it consistently delivers lower performance than POSIX does across various transfer sizes, indicating poor STDIO performance, in general. Recognizing the users' need for using STDIO with the integration of applications' runtime information (e.g., the modules loaded at runtime), we suggest optimizing STDIO via data aggregation and dynamic I/O adaptation within high-level libraries (e.g., HDF5) and encouraging end users to employ these libraries.

In summary, these observations seek attention from HPC facilities to develop tools for better utilization of in-system storage layers, from I/O libraries to understand the I/O workloads to optimize, and from profiling tool developers to collect relevant information from new I/O interfaces. We plan our future work in multiple dimensions. Although the HPC I/O stack offers various tuning parameters, such as collective buffering at the MPI-IO level, striping settings in using Lustre, and the number of burst buffer nodes in Cori, understanding how users take advantage of the tuning parameters may lead to better default settings or dynamic settings at application runtime. We plan to explore the current usage of these tuning options by users. Another focus of this future study will be how many users tune their I/O in subsequent application executions.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers and our shepherd, Haiying Xu from HPDC'22, for their invaluable comments that improved this paper. We are thankful to Alex May, Olga Kuchar, and Ross Miller from OLCF for their helps in the publication of the Darshan dataset collected from the Summit supercomputer.

This work used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This research used resources of the National Energy Research Scientific Computing Center under Contract No. DE-AC02-05CH11231. This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357.

REFERENCES

[1] Jean Luca Bez, Houjun Tang, Bing Xie, David Williams-Young, Rob Latham, Rob Ross, Sarp Oral, and Suren Byna. 2021. I/O Bottleneck Detection and Tuning:Towards Connecting the Dots using Interactive Log Analysis. In *Proceedings of 2021 IEEE/ACM 6th International Parallel Data Systems Workshop (PDSW'21)*. IEEE, St. Louis, MO, USA, 15–22. https://doi.org/10.1109/PDSW54622.2021.00008

[2] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. 2011. Understanding and Improving Computational Science Storage Access through Continuous Characterization. *ACM Transactions on Storage (TOS)* 7, 3, Article 8 (Oct. 2011), 26 pages. https://doi.org/10.1145/2027066.2027068

[3] Emily Costa, Tirthak Patel, Benjamin Schwaller, Jim M. Brandt, and Devesh Tiwari. 2021. Systematically Inferring I/O Performance Variability by Examining Repetitive Job Behavior. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 33, 15 pages. https://doi.org/10.1145/3458817.3476186

[4] Dave Henseler, Benjamin Landsteiner, Doug Petesch, Cornell Wright, and Nicholas J Wright. 2016. Architecture and Design of Cray Datawarp. In *Cray User Group (CUG'16)*. Cray User Group, London, UK, 11 pages.

[5] Xiao-Yu Hu, Evangelos Eleftheriou, Robert Haas, Ilias Iliadis, and Roman Pletka. 2009. Write Amplification Analysis in Flash-Based Solid State Drives. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference* (Haifa, Israel) *(SYSTOR '09)*. Association for Computing Machinery, New York, NY, USA, Article 10, 9 pages. https://doi.org/10.1145/1534530.1534544

[6] Mihailo Isakov, Eliakin del Rosario, Sandeep Madireddy, Prasanna Balaprakash, Philip Carns, Robert B. Ross, and Michel A. Kinsy. 2020. HPC I/O Throughput Bottleneck Analysis with Explainable Local Models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Atlanta, Georgia) *(SC'20)*. IEEE Press, Atlanta, Georgia, Article 33, 13 pages.

[7] Sunggon Kim, Alex Sim, Kesheng Wu, Suren Byna, Yongseok Son, and Hyeonsang Eom. 2020. Towards HPC I/O Performance Prediction through Large-Scale Log Analysis. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing* (Stockholm, Sweden) *(HPDC '20)*. Association for Computing Machinery, New York, NY, USA, 77–88. https://doi.org/10.1145/3369583.3392678

[8] Jianwei Li, Wei keng Liao, Alok Choudhary, Rob Ross, Rajeev Thakur, William Gropp, Rob Latham, Andrew Siegel, B. Gallagher, and Michael Zingale. 2003. Parallel netCDF: A High-Performance Scientific I/O Interface. In *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing (SC'03)*. IEEE, Phoenix, AZ, USA, 39–39. https://doi.org/10.1109/SC.2003.10053

[9] Seung-Hwan Lim, Hyogi Sim, Raghul Gunasekaran, and Sudharshan S. Vazhkudai. 2017. Scientific User Behavior and Data-Sharing Trends in a Petascale File System. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC'17)*. Association for Computing Machinery, New York, NY, USA, Article 46, 12 pages. https://doi.org/10.1145/3126908.3126924

[10] Yang Liu, Raghul Gunasekaran, Xiaosong Ma, and Sudharshan S. Vazhkudai. 2016. Server-Side Log Data Analytics for I/O Workload Characterization and Coordination on Large Shared Storage Systems. In *Proceedings of the International*

Conference for High Performance Computing, Networking, Storage and Analysis (Salt Lake City, Utah) *(SC'16)*. IEEE Press, Salt Lake City, Utah, Article 70, 11 pages.

[11] Glenn K. Lockwood, Shane Snyder, Teng Wang, Suren Byna, Philip Carns, and Nicholas J. Wright. 2018. A Year in the Life of a Parallel File System. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis* (Dallas, Texas) *(SC '18)*. IEEE Press, Dallas, Texas, Article 74, 13 pages. https://doi.org/10.1109/SC.2018.00077

[12] Jay F. Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Chen Jin. 2008. Flexible IO and Integration for Scientific Codes through the Adaptable IO System (ADIOS). In *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments* (Boston, MA, USA) *(CLADE'08)*. Association for Computing Machinery, New York, NY, USA, 15–24. https://doi.org/10.1145/1383529.1383533

[13] Huong Luu, Marianne Winslett, William Gropp, Robert Ross, Philip Carns, Kevin Harms, Mr Prabhat, Suren Byna, and Yushu Yao. 2015. A Multiplatform Study of I/O Behavior on Petascale Supercomputers. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing* (Portland, Oregon, USA) *(HPDC'15)*. Association for Computing Machinery, New York, NY, USA, 33–44. https://doi.org/10.1145/2749246.2749269

[14] Ross Miller, Jason Hill, David A Dillow, Raghul Gunasekaran, Galen M Shipman, and Don Maxwell. 2010. Monitoring Tools for Large Scale Systems. In *Proceedings of Cray User Group Conference (CUG'10)*. Cray User Group, Edinburgh, UK, 5 pages.

[15] Adam Moody, Danielle Sikich, Ned Bass, Michael J. Brim, Cameron Stanavige, Hyogi Sim, Joseph Moore, Tony Hutter, Swen Boehm, Kathryn Mohror, Dmitry Ivanov, Teng Wang, Craig P. Steffen, and US DOE National Nuclear Security Administration. 2022. UnifyFS: A Distributed Burst Buffer File System - 0.1.0. https://github.com/LLNL/UnifyFS

[16] NERSC. 2021. *NEWT - A Nice and Easy Web API for HPC*. National Energy Research Scientific Computing Center. https://newt.nersc.gov/api/ Accessed: Jan. 1, 2022.

[17] Sarp Oral, Sudharshan S. Vazhkudai, Feiyi Wang, Christopher Zimmer, Christopher Brumgard, Jesse Hanley, George Markomanolis, Ross Miller, Dustin Leverman, Scott Atchley, and Veronica Vergara Larrea. 2019. End-to-End I/O Portfolio for the Summit Supercomputing Ecosystem. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC'19)*. Association for Computing Machinery, New York, NY, USA, Article 63, 14 pages. https://doi.org/10.1145/3295500.3356157

[18] ORNL. 2022. *Spectral Library*. Oak Ridge National Laboratory. https://www.olcf.ornl.gov/spectral-library

[19] Tirthak Patel, Suren Byna, Glenn K. Lockwood, and Devesh Tiwari. 2019. 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* (Denver, Colorado) *(SC'19)*. Association for Computing Machinery, New York, NY, USA, Article 65, 13 pages. https://doi.org/10.1145/3295500.3356183

[20] Tirthak Patel, Suren Byna, Glenn K. Lockwood, Nicholas J. Wright, Philip Carns, Robert Ross, and Devesh Tiwari. 2020. Uncovering Access, Reuse, and Sharing Characteristics of I/O-Intensive Files on Large-Scale Production HPC Systems. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST'20)*. USENIX Association, USA, 91–102.

[21] Hongzhang Shan, Katie Antypas, and John Shalf. 2008. Characterizing and Predicting the I/O Performance of HPC Applications Using a Parameterized Synthetic Benchmark. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing* (Austin, Texas) *(SC'08)*. IEEE Press, Austin, Texas, Article 42, 12 pages.

[22] Manu Shantharam, Mahidhar Tatineni, Dongju Choi, and Amitava Majumdar. 2018. Understanding I/O Bottlenecks and Tuning for High Performance I/O on Large HPC Systems: A Case Study. In *Proceedings of the Practice and Experience on Advanced Research Computing* (Pittsburgh, PA, USA) *(PEARC'18)*. Association for Computing Machinery, New York, NY, USA, Article 54, 6 pages. https://doi.org/10.1145/3219104.3219120

[23] Shane Snyder, Philip Carns, Kevin Harms, Robert Ross, Glenn K. Lockwood, and Nicholas J. Wright. 2016. Modular HPC I/O Characterization with Darshan. In *Proceedings of the 5th Workshop on Extreme-Scale Programming Tools* (Salt Lake City, Utah) *(ESPT'16)*. IEEE Press, Salt Lake City, Utah, 9–17.

[24] Houjun Tang, Suren Byna, N Anders Petersson, and David McCallen. 2021. Tuning Parallel Data Compression and I/O for Large-scale Earthquake Simulation. In *Proceedings of 2021 IEEE International Conference on Big Data (Big Data'21)*. IEEE, Orlando, FL, USA, 2992–2997. https://doi.org/10.1109/BigData52589.2021.9671876

[25] Houjun Tang, Bing Xie, Suren Byna, Philip Carns, Quincey Koziol, Sudarsun Kannan, Jay Lofstead, and Sarp Oral. 2021. SCTuner: An Autotuner Addressing Dynamic I/O Needs on Supercomputer I/O Subsystems. In *Proceedings of 2021 IEEE/ACM Sixth International Parallel Data Systems Workshop (PDSW'21)*. IEEE, St. Louis, MO, USA, 29–34. https://doi.org/10.1109/PDSW54622.2021.00010

[26] The HDF Group. 2022. *The HDF5 Library and File Format*. The HDF Group. https://www.hdfgroup.org/HDF5/

[27] TOP500.org. 2022. *TOP 500 - November 2021*. TOP500.org. Retrieved April 28 2022 from https://www.top500.org/lists/top500/2021/11

[28] Feiyi Wang, Hyogi Sim, Cameron Harr, and Sarp Oral. 2017. Diving into Petascale Production File Systems through Large Scale Profiling and Analysis. In *Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems* (Denver, Colorado) *(PDSW-DISCS'17)*. Association for Computing Machinery, New York, NY, USA, 37–42. https://doi.org/10.1145/3149393.3149399

[29] Bing Xie, Jeffrey Chase, David Dillow, Oleg Drokin, Scott Klasky, Sarp Oral, and Norbert Podhorszki. 2012. Characterizing Output Bottlenecks in a Supercomputer. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (Salt Lake City, Utah) *(SC'12)*. IEEE Computer Society Press, Washington, DC, USA, Article 8, 11 pages.

[30] Bing Xie, Yezhou Huang, Jeffrey S. Chase, Jong Youl Choi, Scott Klasky, Jay Lofstead, and Sarp Oral. 2017. Predicting Output Performance of a Petascale Supercomputer. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing* (Washington, DC, USA) *(HPDC'17)*. Association for Computing Machinery, New York, NY, USA, 181–192. https://doi.org/10.1145/3078597.3078614

[31] Bing Xie, Sarp Oral, Christopher Zimmer, Jong Youl Choi, David Dillow, Scott Klasky, Jay Lofstead, Norbert Podhorszki, and Jeffrey S. Chase. 2020. Characterizing Output Bottlenecks of a Production Supercomputer: Analysis and Implications. *ACM Transactions on Storage (TOS)* 15, 4, Article 26 (2020), 39 pages.

[32] Bing Xie, Zilong Tan, Philip Carns, Jeffrey Chase, Kevin Harms, Gerald Lofstead, Sarp Oral, Sudharshan Vazhkudai, and Feiyi Wang. 2019. Applying Machine Learning to Understand Write Performance of Large-scale Parallel Filesystems. In *Proceedings of 2018 IEEE/ACM 4th International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS'19)*. IEEE, Denver, CO, USA, 30–39. https://doi.org/10.1109/PDSW49588.2019.00008

[33] Bing Xie, Houjun Tang, Suren Byna, Jesse Hanley, Quincey Koziol, Tonglin Li, and Sarp Oral. 2021. Battle of the Defaults: Extracting Performance Characteristics of HDF5 under Production Load. In *Proceedings of 2021 the 21th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid'21)*. IEEE, Melbourne, Australia, 51–60. https://doi.org/10.1109/CCGrid51090.2021.00015