I/O Access Patterns in HPC Applications: A 360-Degree Survey

JEAN LUCA BEZ and SUREN BYNA, Lawrence Berkeley National Laboratory, USA
SHADI IBRAHIM, Inria, University of Rennes, CNRS, IRISA, Rennes, France

The high-performance computing I/O stack has been complex due to multiple software layers, the interdependencies among these layers, and the different performance tuning options for each layer. In this complex stack, the definition of an "I/O access pattern" has been reappropriated to describe what an application is doing to write or read data from the perspective of different layers of the stack, often comprising a different set of features. It has become common to have to redefine what is meant when discussing a pattern in every new study, as no assumption can be made. This survey aims to propose a baseline taxonomy, harnessing the I/O community's knowledge over the past 20 years. This definition can serve as a common ground for high-performance computing I/O researchers and developers to apply known I/O tuning strategies and design new strategies for improving I/O performance. We seek to summarize and bring a consensus to the multiple ways to describe a pattern based on common features already used by the community over the years.

CCS Concepts: • **Information systems** → **Information storage systems**; *Storage architectures*; *Hierarchical storage management;*

Additional Key Words and Phrases: I/O access pattern, HPC I/O, storage, I/O characterization

ACM Reference format:
Jean Luca Bez, Suren Byna, and Shadi Ibrahim. 2023. I/O Access Patterns in HPC Applications: A 360-Degree Survey. *ACM Comput. Surv.* 56, 2, Article 46 (September 2023), 41 pages.
https://doi.org/10.1145/3611007

1 INTRODUCTION

In **High-Performance Computing (HPC)**, "I/O access pattern" or "I/O signature" is broadly used to express *how* an application is performing **Input and Output (I/O)** operations [21]. Although the word is broadly used, there is no globally accepted convention to describe which features define an I/O access pattern and how they differ based on the level in the HPC I/O software stack they are being used to describe. For instance, some studies do not consider temporal features as part of the description of an access pattern [22] but others do [27, 219]. Moreover, some studies describe the access pattern as seen by high-level libraries [26, 41, 66], others do so by the I/O middleware [9, 216], and still others do so by what the underlying file system is receiving [24, 128, 237]. Hence, it is a common practice to have to redescribe exactly what is meant when discussing the "I/O access pattern" in every new study, as no assumption can be made.

Authors' addresses: J. L. Bez and S. Byna, Lawrence Berkeley National Laboratory, Berkeley, CA; emails: jlbez@lbl.gov, sbyna@lbl.gov, shadi.ibrahim@inria.fr; S. Ibrahim, Inria, University of Rennes, CNRS, IRISA, Rennes, Rennes, France; email: shadi.ibrahim@inria.fr.

0360-0300/2023/09-ART46 $15.00
https://doi.org/10.1145/3611007

In this article, we aim to propose a baseline taxonomy, harnessing the I/O community's knowledge over the past 20 years, that researchers and application developers can use to define an application's I/O access pattern. This definition can serve as a common ground to apply known I/O tuning strategies as well as to design new strategies for improving I/O performance. The definitions we proposed here seek to summarize and bring a consensus to the multiple ways to describe a pattern based on common features already used by the community over the years. It does not seek to be set in stone but a baseline consensus, which can be extended to accommodate new applications from different domains.

Besides solidifying a taxonomy based on common-ground features, our survey has practical applications for current and future research in the area. For instance, a plethora of existing optimization techniques that seek to improve applications' I/O performance relies on the definition of an access pattern. Despite the strategies they apply, they all work by modifying how an application is accessing its data (i.e., its I/O access pattern) to be more suited to the underlying layer of the I/O stack. Request aggregation, reordering, scheduling, and collective operations [44, 59, 60, 113, 197, 216] are a few examples of techniques that optimization mechanisms apply at different layers of the I/O stack. In general, such optimizations typically improve the performance for a given system deployment and I/O patterns, but not for all. Moreover, they often rely on correctly applying such techniques to the workload represented by a set of access patterns.

As novel applications from diverse domains are harnessing HPC platforms and the systems are becoming more complex to handle more concurrent applications, it becomes paramount for those systems that seek to auto-tune their parameters to detect the I/O access patterns at runtime accurately. Such detection allows them to make decisions and apply the set of optimization techniques that are specifically designed for the observed patterns. Having an established taxonomy with clearly defined features can help bridge the gap between describing the access patterns and mapping them to existing techniques, allowing, for instance, AI-based and automatic tuning mechanisms to navigate the complex parameter space to find which optimizations and configurations can be applied for an observed I/O access pattern.

Contributions. Although the term *I/O access pattern* is used heavily in published literature, there has been no study that encompasses, discusses, and categorizes I/O access patterns to the breadth to which the term is used in HPC. White et al. [219] proposed a taxonomy for temporal I/O patterns of HPC jobs to aid in automatically detecting poorly performing jobs. Boito et al. [21] touched on several key points of the HPC I/O stack, including access pattern extraction. However, both suffer from the same issues of redefining what an access pattern means. Furthermore, those definitions do not encompass all features of I/O and their impact when glancing at the different layers of the parallel I/O stack. In this work, we seek to provide a broader taxonomy for I/O access patterns, taking into account temporal behavior and also integrating other commonly used features as understood by the community, and to describe how those patterns are represented, used, and transformed as we traverse the HPC I/O stack.

The remainder of the article is organized as follows. In Section 2, we discuss the traditional HPC I/O software stack. The discussion of access patterns is split into four sections representing the layers in the stack. In Section 3, we describe the common data models used by scientific applications. Section 4 discusses how those data models are represented by high-level I/O libraries. Section 5 approaches the translation of I/O accesses used by middleware libraries. Finally, in Section 6, we present the perspective of the file system. In Section 8, we present common I/O benchmarks and kernels used to exercise access patterns in different levels of the HPC I/O stack, and in Section 9, we describe popular tools to visualize those patterns by using profiling and log traces. In Section 10, we conclude this survey with a summary of our contributions, existing gaps, and highlight opportunities for further R&D.

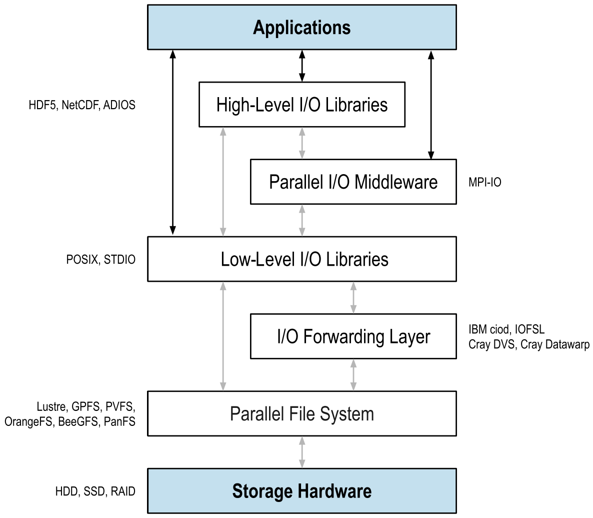


Fig. 1. The traditional HPC I/O software stack that includes several layers of libraries between applications and storage hardware.

2 HPC I/O STACK

To support the I/O workloads from serial or parallel scientific applications, HPC systems provide a multi-layered software stack, as illustrated in Figure 1. Between the applications and storage hardware, the parallel I/O stack consists of high-level I/O libraries, middleware I/O libraries, optimization layers, and **Parallel File Systems (PFS)**.

While traversing the stack, an access pattern is often reshaped via a series of data transformations originating from distinct abstractions and mappings between the data models used in the layers and the application of optimization techniques (e.g., scheduling, aggregation, and compression) before reaching the file system. Furthermore, some contextual information gets lost in this process. For instance, when requests arrive at the file system layer, the file system is unaware of which application or process the request originated from, or even if the request went through any data transformations and which ones. Consequently, an application may believe that it is accessing its data in one way, whereas something entirely different is happening in reality at the lowest layers of the stack.

High-level I/O libraries are used by applications to provide data models and file management abstractions that facilitate data portability and high performance. Examples of widely adopted libraries are HDF5 [202], NetCDF [119]/PnetCDF [121], and ADIOS [134]. Those libraries map the applications' data abstractions into files or objects and encode the data in portable file formats. These libraries allow users to add metadata to describe the data and their data structures. In addition to these parallel I/O high libraries, application domain-specific libraries also exist. Among these, ROOT [5] and FITS [80, 217] serve High-Energy Physics (HEP) and astronomy communities, respectively. Applications may also use MPI-IO (Message Passing Interface I/O) [46], POSIX I/O (Portable Operating System Interface I/O) [67], or STDIO (Standard Input and Output) interfaces directly to perform I/O to the file systems. We discuss these interfaces, their challenges, and their known impact on HPC I/O performance in Section 7.1.4.

In MPI-IO, a file is an ordered collection of typed data items. It presents a higher level of data abstraction than POSIX by allowing users to define data models that are natural to the application. Nonetheless, it supports defining complex data patterns for parallel write and read operations using independent and collective I/O calls. Furthermore, it allows taking advantage of optimization opportunities when using collective calls. In POSIX I/O, however, a file is viewed as a sequence of bytes. This interface allows transferring contiguous regions of bytes between the file and memory and non-contiguous regions of bytes from memory to a file by giving full, low-level control of the I/O operations. However, in the context of HPC, there is little in the interface that inherently supports parallel I/O. For instance, POSIX does not easily support collective access to files while leaving it to the programmer to coordinate access and ensure consistency. STDIO, in contrast, abstracts all file operations into operations on (input or output) streams of bytes. It comprises the C *stdio.h* family of functions [97], such as *fopen()*, *fprintf()*, and *fscanf()*. These I/O functions are commonly used in genomics and biology to store sequencing information in text format [182]. However, STDIO functions do not directly support random access to data files. Instead, the process relies on the programmer to create a stream, seek the position in the file, and then read/write bytes in sequence from/to the stream.

I/O forwarding [3], initially proposed for Blue Gene and later extended, seeks to reduce the number of (compute) nodes concurrently accessing the PFS servers by creating an additional transparent layer between the compute nodes and the data servers. Instead of the applications accessing the PFS directly, the I/O forwarding technique defines a set of *I/O nodes* that are responsible for receiving I/O requests from applications and forwarding them to the PFS in a controlled manner, allowing optimization techniques such as request scheduling, aggregation, and compression [2, 16, 159, 190, 232], to reshape the pattern and flow of I/O requests to better suit the underlying layers.

In large-scale systems, applications rely on PFS to provide a globally persistent shared storage infrastructure and a global namespace across many distributed storage servers to read and write data to files. A PFS comprises two types of servers with distinct roles: the data servers and the metadata servers. The latter handles information about the files (e.g., sizes and permissions) and their location in the system. Lustre [74, 93], IBM Spectrum Scale (previously known as GPFS) [174], BeeGFS [87], and PVFS [36], among others, are commonly used PFS on large-scale HPC systems. To achieve high performance, these file systems harness parallelism by using *data striping* [188], which consists of partitioning the files and distributing the data into fixed-size chunks across multiple storage nodes. Finally, the PFS servers provide a logical file system abstraction over diverse storage devices such as **Hard Disk Drives (HDDs)**, **Solid-State Drives (SSDs)**, or **Redundant Array of Independent Drives (RAID)**.

Summary #1

The multi-layered software and hardware HPC I/O stack is complex. To access data in HPC systems, applications issue requests that, while traversing the I/O stack, are reshaped via a series of data transformations. These originate from distinct abstractions and mappings between the data models used in each layer combined with optimization techniques applied before reaching the file system and, eventually, the storage hardware.

In the following sections, we discuss the I/O access patterns observed in the HPC stack's layers, from application data models and their I/O requests percolating through the underlying layers until the file systems handle them.

3 APPLICATION DATA MODELS AND ACCESS PATTERNS

Scientific applications often use data abstractions provided by high-level libraries (e.g., HDF5, NetCDF, ADIOS) to express data structures more naturally to a problem and domain. HPC simulations often describe their data objects using multi-dimensional data or meshes, arbitrary subsets, points and curves, and key values [151, 181]. Mesh data objects, in particular, can be further

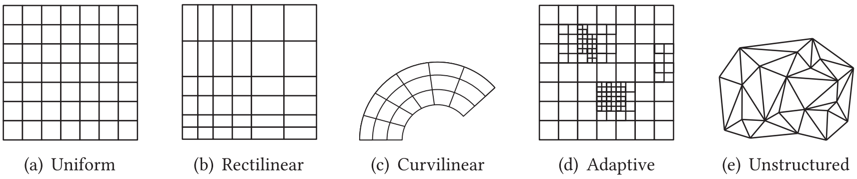


Fig. 2. Representation of common high-level data models used by HPC applications in different science domains.

represented by structured rectilinear, non-uniform rectilinear, grid-less points, structured (curvilinear), arbitrary polyhedral, constructive solid geometry (CSG), unstructured zoo (UCD), and adaptive mesh refinement (AMR) meshes. In Figure 2, we show these most common high-level data models used by HPC applications.

For instance, physics simulations rely on finite element methods to discretize the simulated domain by splitting it into smaller elements. Numerical methods are then applied to solve differential equations on these elements. These methods often assume that the domain is divided into a structured or unstructured mesh of smaller, simpler elements. The first has some advantages over the latter. It is simpler to use, requiring less memory as its coordinates can be calculated rather than stored. However, in the case of unstructured meshes, computations are irregular, causing problems of indirect, non-strided (i.e., no gaps between successive data accesses), or non-contiguous access to memory [153]. Yet structured meshes lack the flexibility to represent complex shapes needed for some domains [14].

Uniform rectilinear meshes (Figure 2(a)) divide the computation domain into a set of rectangular cells and are regular both in topology and geometry. If points and cells are organized into a 1D plane, they are often used to express image data. Volume can be represented by arranging this mesh into multiple stacked planes. Rectilinear grids (Figure 2(b)) differ in their regularity, where the spacing between points may vary (in any of the axes), but the rows and columns are still parallel to the axis of the Cartesian coordinate system. Curvilinear (Figure 2(c)) or structured grids (also known as mapped mesh or body-fitted mesh) have the same topology as a rectilinear grid but allow more variation in the shape of the mesh, as they can be warped into any configuration without overlap or intersection. These grids do not use Cartesian grid lines but a curvilinear coordinate system where an array of point coordinates explicitly represents the geometry. Curvilinear grids provide a more compact memory footprint and are regular in topology but present irregular geometry. They are used for finite difference computations such as flow [64], heat transfer, and combustion simulations [31]. Unstructured grids (Figure 2(e)) are a tessellation that conforms to nearly any desired geometry. However, they require more information to be stored and recovered than structured grids, for instance, to express the neighbor connectivity list. Those meshes are used in seismic wave [65, 91, 225], fluid dynamics [79, 184], and heat transfer [130, 153] simulations.

The high-level data models can be stored in a file using two data layouts with regard to interleaving: **Array of Structures (AoS)** or **Structure of Arrays (SoA)**, as depicted in Figure 3. A contiguous pattern in the memory means that multiple arrays are of the same basic data types, such as integer, float, and double. The non-contiguous pattern in memory, also referred to as an AoS or a derived data type, represents compound data types derived from basic data types. The first helps access adjacent work items in contiguous memory locations, whereas the latter is often more intuitive from the developer's perspective, as each structure is kept together. Once that data needs to be persisted in a file, it can use the same strategy or the opposite one used to represent the data in memory. Table 1 depicts this by comparing in-memory and in-file representations when

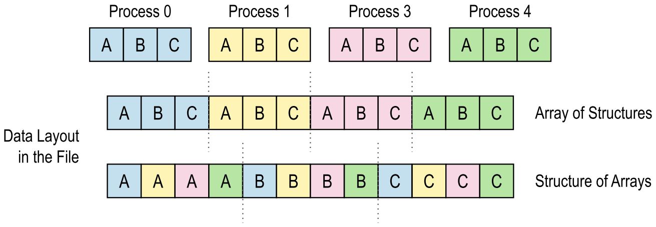
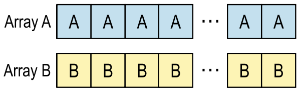
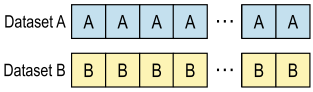
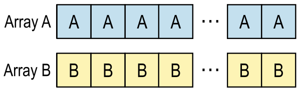
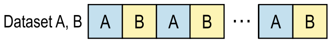
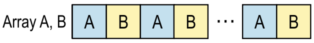
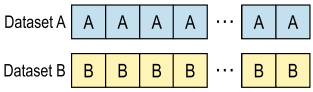
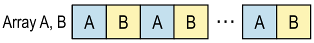
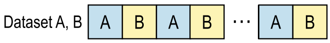


Fig. 3. Data layout in a file for AoS or SoA.

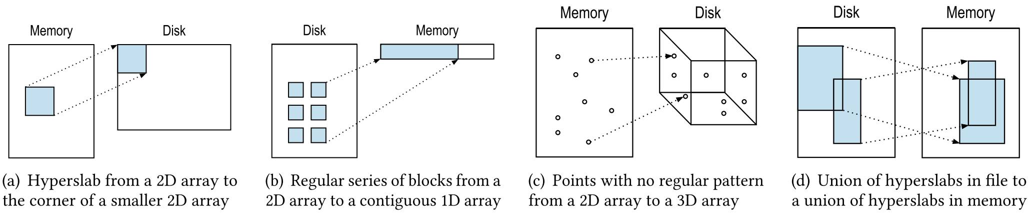
Table 1. In-Memory Data Structure and In-File Data Layout Mappings

In-memory representation	**In-file representation**
 Contiguous	 Contiguous
 Continuous	 Compound
 Compound	 Continuous
 Compound	 Compound

For illustration purposes, we restrict ourselves to 1D arrays.

using HDF5, for instance, to store data in a contiguous or compound fashion. Nonetheless, this representation is also used by other I/O libraries and interfaces such as MPI-IO, where one can define data types to describe both memory and file layout.

As an application's I/O requests need to transverse the I/O stack to ultimately reach the storage system, its I/O pattern is reshaped and transformed by various existing optimizations techniques (e.g., collective buffering and data sieving [9, 44, 136, 199, 216], request scheduling [6, 16, 20, 22], and request aggregation [96, 197, 208]). Often these transformations are transparent to the end user. Thus, what the application believes it is doing might differ from what the other levels of the I/O stack perceive of the application's behavior. Due to that, information related to how the application is accessing its data can be lost throughout the stack. For instance, when requests arrive at a PFS, it is nearly impossible to determine which rank issued that request and whether it was initially contiguous or not. To clarify such an example, if I/O middleware libraries such as MPI-IO apply collective I/O optimization, only the aggregators will issue the requests to the PFS, and only they will know to which ranks they should exchange data about the request. To further complicate the situation, suppose a forwarding layer [2, 98] (or any other transparent middleware) is present, possibly merging or scheduling and aggregating requests from multiple compute nodes. In that case, the PFS will not know which application rank originally issued the I/O request. However, when those requests are forwarded to the PFS (using a forwarding layer), the latter will often not



(a) Hyperslab from a 2D array to the corner of a smaller 2D array

(b) Regular series of blocks from a 2D array to a contiguous 1D array

(c) Points with no regular pattern from a 2D array to a 3D array

(d) Union of hyperslabs in file to a union of hyperslabs in memory

Fig. 4. A visual representation of common hyperslab selections between memory and file representations for partial I/O in HDF5.

know from which compute nodes they originated. Only the I/O nodes will have such information to forward back the data.

Summary #2

Parallel applications rely on data models that are naturally mapped to a problem domain. To be stored in files, data must be transformed by intermediate layers of the HPC I/O stack. Thus, the features we can use to describe an I/O access pattern at the file system level are not the same as the view we have at a higher level in the I/O stack.

4 ACCESS PATTERNS IN HIGH-LEVEL I/O LIBRARIES

High-level I/O libraries allow HPC applications to express scientific simulation data more naturally instead of being constrained or caught up by system-specific details. HDF5, NetCDF/PnetCDF, ADIOS, ROOT, and FITS are examples of such libraries, each providing a set of APIs to express complex multi-dimensional data, contiguous and non-contiguous data, seeking to attain performance and portability, and also increase productivity.

HDF5 (Hierarchical Data Format, Version 5) is a well-known self-describing file format and an I/O library [202] that provides flexibility, extendibility, and portability. It is used widely in many science domains to manage various data models [28]. HDF5 uses the concept of dataspace objects to control data transfer when data is read or written. A dataspace defines the layout of the data (the organization in rows, columns, etc.) in a file and memory. Data is rearranged by the library when the different layouts are used to represent a given dataset in memory or a file. However, both source and destination are stored as contiguous blocks of storage with the elements ordered as defined by the dataspace. HDF5 allows an application to read or write to a portion of a dataset (partial I/O) by using hyperslabs and points. Hyperslabs are portions of datasets whose selection can be a logically contiguous collection of points in a dataspace or a regular pattern of points or blocks in a dataspace. Figure 4 illustrates four types of partial I/O in HDF5.

An HDF5 hyperslab can be viewed as a rectangular pattern defined by four arrays: offsets of the starting location for the hyperslab, the stride or number of elements to separate each element or block to be selected, the number of elements or blocks to select along each dimension, and the size of the blocks selected from the dataspace. Figure 5 depicts a hyperslab selection (left) in a dataset.

NetCDF provides scientific programmers with a self-describing and machine-independent portable format for storing array-oriented data [119]. NetCDF-4 is the current version of classical NetCDF file format. NetCDF-4 supports parallel file access to the classic NetCDF and HDF5 files. Parallel I/O to the NetCDF-4 formatted files is supported through the HDF5 library, and that to the classic NetCDF files is supported through PnetCDF. PnetCDF is a high-performance parallel I/O

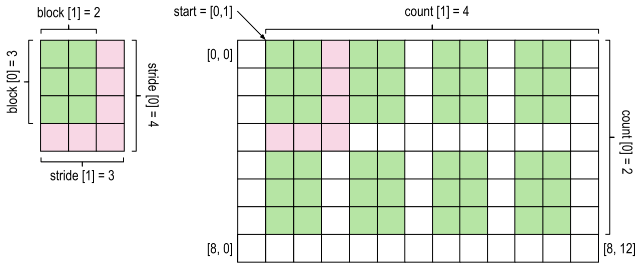


Fig. 5. Example of a hyperslab selection for partial I/O operations using the HDF5 library.

library for accessing NetCDF files providing higher-level data structures (e.g., multi-dimensional arrays of typed data). The NetCDF-4 read and write API functions allow defining hyperslab parameters such as `start` and `count` vectors. For instance, the function `nc_put_vara_int()`—to write an array of integer values to a variable—has arguments to specify the `start` index for each dimension of an array and corresponding `count` specifying the edge lengths along each dimension of the block of data values to be written.

ADIOS (Adaptable Input Output System) [134, 139] provides an I/O abstraction framework for portable and scalable I/O to aid scientific applications when data transfer volumes exceed the capabilities of traditional file I/O. Different from HDF5, ADIOS is not a hierarchical model but rather sits on a layer of abstraction beneath those. However, it also relies on self-describing data in binary-packed (.bp) format for rapid metadata extraction, but it can use different backend file storage formats such as HDF5 and NetCDF. ADIOS can also extract relevant information from large datasets, transporting and transforming groups of self-describing data variables and attributes across different media. This library uses an external metadata file in XML format to describe variables, types, and the path to take from memory to disk. It also has built-in optimization modules for buffering and scheduling [76]. The ADIOS-2 API allows specifying the start and count vectors for setting the offsets and dimensions for the MPI ranks, respectively.

ROOT [5] is an object-oriented C++ framework conceived in the HEP community, designed for storing and analyzing petabytes of data efficiently. ROOT has been used for storing over one exabyte of HEP events [140]. In ROOT, objects in memory go under serialization and compression before reaching the binary representation in files. These are self-descriptive files comprised of a header and data following a hierarchical directory format. ROOT can also use columnar representation for data in files, allowing I/O optimizations such as partial reading (i.e., reading only a subset of relevant columns), prefetching, and read-ahead to improve performance. For instance, HEP applications benefit from such file layouts when analyzing many statistically independent collision events.

FITS (Flexible Image Transport System) [80, 217] is a standardized data format in astronomy. Initially conceived as a standard interchange format for digital images, FITS files are used as a working data format to store ASCII or binary tabular data, in addition to images and spectra. Files consist of a sequence of one or more **Header and Data Units (HDUs)**. A header is composed of ASCII card images (usually read into a string array variable) that describe the content of the associated data unit, which might be a spectrum (vector), an image (array), or tabular data in ASCII or binary format (often read as a structure). Tabular data cannot appear in the first HDU, whereas image and vector data can be present in any HDU. The HDUs following the first (or primary) HDU are also known as extensions.

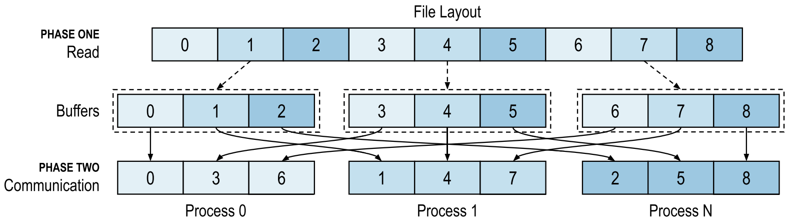


Fig. 6. Two-phase I/O for read requests.

Summary #3

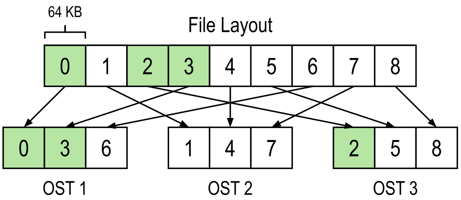
High-level I/O libraries present a layer of abstractions so that applications can easily map their data models to files. Several of these libraries are designed to provide portability of file formats as well as a self-describing feature that allows adding metadata to data. I/O access patterns of multi-dimensional data structures at this layer are designed to hide the complexity of converting data models to their file layouts.

5 ACCESS PATTERNS AT THE I/O MIDDLEWARE LAYER

Before reaching the persistent media, an application's request can go through a series of data transformations enabled by I/O optimizations. For instance, using MPI-IO, if a group of MPI ranks knows which parts of a file each rank is accessing, it becomes possible to merge these requests into a smaller number of larger and more contiguous accesses that span over a large portion of the file. When applied at the client level, this optimization is described as two-phase I/O [53, 199] with collective buffering and data sieving [200]. Such optimizations effectively change how the application issues its I/O request—that is, it changes its access pattern.

Collective buffering aims to reduce I/O time by making file accesses as large and as contiguous as possible, even if it requires additional communication between the ranks. In two-phase I/O, aggregator processes are responsible for carrying out the writes and reads. Each one manages a chunk of contiguous data from a subset of processes in a file. During the write process, an aggregator gathers data from a subset of processes into contiguous chunks in memory and writes the aggregated data to the file system. During reads, aggregators load part of the file and distribute smaller chunks of data to a subset of processes, as shown in Figure 6. For instance, ROMIO, which is a portable, high-performance implementation of MPI-IO, exposes two user-defined tuning options that can control the application of this technique: the number of processes that actually issue the I/O requests in the I/O phase (`cb_nodes`), often referred to as aggregators, and the maximum buffer size on each process (`cb_buffer_size`). These options help define the access pattern perceived by underlying layers of the I/O stack.

Data sieving is another optimization in MPI-IO aiming to reduce I/O latency by making as few requests to the PFS as possible. For read operations, when a process issues non-contiguous requests, instead of reading each piece of data separately, ROMIO reads a single contiguous chunk that ranges from the first to the last requested byte in the file into a temporary buffer in memory. ROMIO provides two user-defined parameters to control the buffer size for reads (`ind_rd_buffer_size`) and writes (`ind_wr_buffer_size`) [199]. If a user requests a large portion of the file that would not fit in the allocated memory, ROMIO implementation performs the



(a) Request aligned to the stripe

(b) Request not aligned to the stripe

Fig. 7. Aligned and misaligned requests to a PFS with file striping.

data sieving in parts delimited by the buffer size. The caveat of data sieving is when there are large gaps in access, which can outweigh the costs of reading the extra data.

Summary #4

The middleware layer provides opportunities to apply optimization techniques to transform the data to be more suitable for the underneath file system. Collective buffering and data sieving are two solutions available in MPI-IO to improve data access by reshaping the I/O access pattern.

6 ACCESS PATTERNS AT THE FILE SYSTEM LAYER

Large-scale HPC systems use PFS to provide a persistent shared storage infrastructure, as discussed in Section 2. A PFS is deployed over a set of dedicated nodes and offers a shared namespace, so applications can seamlessly access remote files. They harness parallelism by breaking the files into chunks or stripes and distributing them across multiple storage nodes to achieve high performance. This operation is often referred to *data striping* [188]. Figure 7(a) illustrates how a file is striped among multiple storage servers, called **Object Storage Targets (OSTs)** in Lustre.

As stripes can be located in different storage targets, to complete a write/read operation, the PFS might need to access multiple targets. Unaligned requests can also require access to multiple OSTs to complete an operation and introduce inefficiencies [105, 127, 238] due to false data sharing. Figure 7(b) depicts such scenario. For instance, consider an application issuing 64-KB requests to a file stored with a stripe size of 64KB. If the first 136KB of the file is used for some header representation, all data accesses are shifted by the header. Thus, instead of issuing a single call to a single OST to write/read the data, the PFS client will need to break the request—for instance, in Figure 7(b), to complete the second request (in pink), two targets (OST 2 and 3) should be contacted to access non-contiguous regions of the file stripe to complete the request. It is easy to extrapolate the impact of misaligned requests on larger scales.

Furthermore, because of this centralized shared infrastructure, for clients to access the OSTs, they need to go over the network, which could introduce overhead and contention, especially if the request size is small and bursty. The previous example could cause a lot of smaller requests (64KB) to be issued because of the misalignment. Moreover, in the file system servers, contiguous data access usually yields higher I/O performance than that of non-contiguous ones [231] for both HDDs and SSDs. Zimmer et al. [242], among others, confirm that small and random request patterns negatively impact file system performance. Therefore, applications observe benefits when accessing a file by issuing fewer requests, reducing the high I/O latency.

Another pivotal aspect that has a direct impact on performance when discussing access patterns at the file system layer is the metadata accesses. In Unix-based operating systems, metadata is stored in an index node (*i-node*) comprising information about ownership, permission, the object's type (e.g., file or directory), size, and modified timestamp [170, 192]. Furthermore, since PFS tend to rely on POSIX I/O semantics (which were not conceived with parallel accesses in mind), the scalability of metadata accesses is often impaired. For instance, serialization is expected to happen in scenarios where a large number of files are created by multiple processes in a single directory. This is a common pattern observed in HPC applications [1, 13, 51, 205, 226].

Moreover, because these PFS tend to adopt the concept of data striping (to allow parallel access and improve performance), before accessing data, the PFS client must fetch permissions and obtain the file layout (including striping locations and sizes) from one of the metadata servers. In the Lustre PFS, a metadata service provides the index, or namespace, for a Lustre file system. The metadata content is stored in volumes called **Metadata Targets (MDTs)**. Since most basic operations involve metadata, it is paramount to ensure scalability of metadata accesses. For instance, prior to Lustre 2.4, only a single MDT could be used to store metadata. The Lustre 2.4 release introduced the concept of DNE (Distributed NamespacE), where the metadata workload could be distributed across multiple MDTs, which usually spread across multiple metadata servers. Nonetheless, metadata servers are often fewer than data servers if not centralized into a single server to avoid complex cache coherence issues and overheads. Needless to say, an application creating or accessing a large number of files might be limited by metadata, possibly impacting other applications in the system due to the shared nature of the metadata servers.

Different approaches [129, 145, 164, 166, 176, 221] have been proposed to tackle metadata issues covering how to handle, scale, and index metadata efficiently. For instance, Liao et al. [129] present a metadata management system that uses a database to record the information of datasets and manages metadata while providing a suitable I/O interface. Paul et al. [166] propose a metadata indexing and search tool specifically designed for large-scale HPC storage systems. Their solution relies on using an in-tree design with a parallel leveled partitioning approach to partition the file system namespace into disjoint subtrees. They maintain an internal metadata index database that uses a two-level database sharding technique to increase indexing and querying performance, combined with a changelog-based approach to keep track of the metadata changes and reindex the file system. Wu et al. [221] proposed StageFS, a PFS optimized for SSD-based clusters. StageFS stores both the metadata and small files in LSM trees for fast indexing. Seeking to avoid frequent small writes, StageFS uses buffering to better utilize the bandwidth of SSD devices. They demonstrate up to 21.28× performance improvements in metadata operations compared to state-of-the-art solutions.

Due to the shared nature of these storage deployments, multiple concurrent applications submitting a large number of metadata operations simultaneously can easily saturate the shared PFS metadata resources. On that front, MetaFS, proposed by Shaffer and Thain [176], seeks to address the bursts of metadata activity during program loading. It indexes the static metadata content of applications and delivers it in bulk to execution nodes, where it can be cached and queried, essentially trading metadata activity for data transfer. Their approach observed order of magnitude decreases in metadata load on the shared file system. On the same front, Macedo et al. [145] present a storage middleware that enables system administrators to proactively control and ensure QoS over metadata workflows in HPC storage systems. Their solution seeks to avoid saturating the shared metadata resources, which could lead to unresponsiveness of the storage backend and overall performance degradation.

Summary #5

Due to its shared nature, a PFS receives interleaved requests from multiple concurrently running applications. Thus, the I/O access pattern seen by these storage targets can present few resemblances to what the application initially issued. Furthermore, metadata requests play a pivotal role in scalability and performance due to the centralized characteristics of such systems.

7 ACCESS PATTERN TAXONOMY

HPC applications issue their I/O requests to a file system in diverse ways, depending on how their data was modeled and coded. They also tend to present a consistent I/O behavior, with a few access patterns being repeated multiple times over an extensive period [35, 61, 62, 70, 89, 137]. A better understanding of such patterns and what optimizations are suited for each one can lead to performance improvement on the application side and when considering the system as a whole. Based on that, some features can be used together to describe the application's *access pattern*. Although there is no globally accepted convention to describe which elements or features define an access pattern, researchers in the HPC I/O area often examine a common subset of factors or parameters—for instance, the file approach (single file or shared file), the number of requests, their sizes, and the spatial locality in the file [16, 34, 137, 138, 227]. However, other features such as temporal behavior, intensity or burstiness, and overlapping accesses are considered for specific applications or optimization techniques [16, 61, 187, 214, 228, 229, 234]. The access pattern does have a direct impact on achieved performance, which justifies the different research efforts put into optimizing data access [30, 81, 114, 138, 231].

We seek to provide a taxonomy for I/O access patterns based on collective understanding from the community and its usage over time. We believe that this formalization is helpful to the scientific community, as applications often observe poor I/O performance due to bottlenecks in the system that could be a result of the lack of translation between metric collection, bottleneck detection, and optimization solutions. A defined and globally accepted taxonomy will aid in translating metrics into patterns and guide end users on how to harness the various existing optimization techniques to improve I/O application performance. Figure 8 uses a node-link hierarchical tree diagram of classes positioned in polar coordinates to describe the taxonomy from different layers of the HPC I/O stack.

Furthermore, besides the features used in each layer, an *I/O access pattern* can be observed from different scopes. Yin et al. [231] classify the access pattern as *local*, *global*, or *system-wide*. The local pattern describes an application's behavior in the context of a process or task, whereas the global pattern describes it at the application level, considering all processes and tasks. However, the system-wide pattern describes the patterns of the diverse concurrent applications when using the shared storage infrastructure or I/O nodes. The local access pattern information is usually employed to identify and apply optimizations on the client side. In contrast, the global access pattern is more suitable for I/O middleware, the forwarding layer, or file system servers since it has an overview of the application's data accesses. The system-wide pattern can also be used in the data servers [23, 118, 169, 187, 236] and forwarding layer [2, 16, 20, 159, 232] to coordinate accesses and optimize I/O performance of the whole system.

7.1 Access Pattern Features

In this section, we discuss the features often used to describe an I/O access pattern and how they are used in the I/O stack. We classify the patterns based on I/O operations, synchronicity, file approach, spatial locality, interfaces, consistency, and temporal behavior.

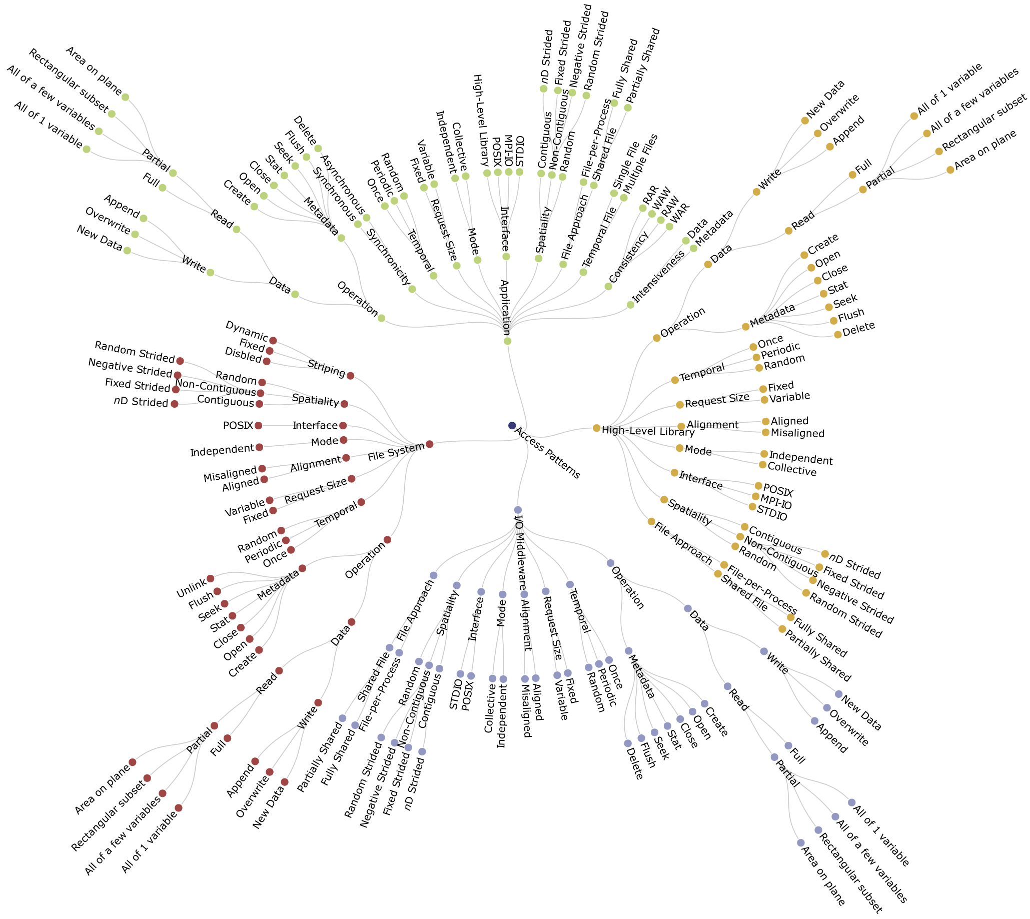


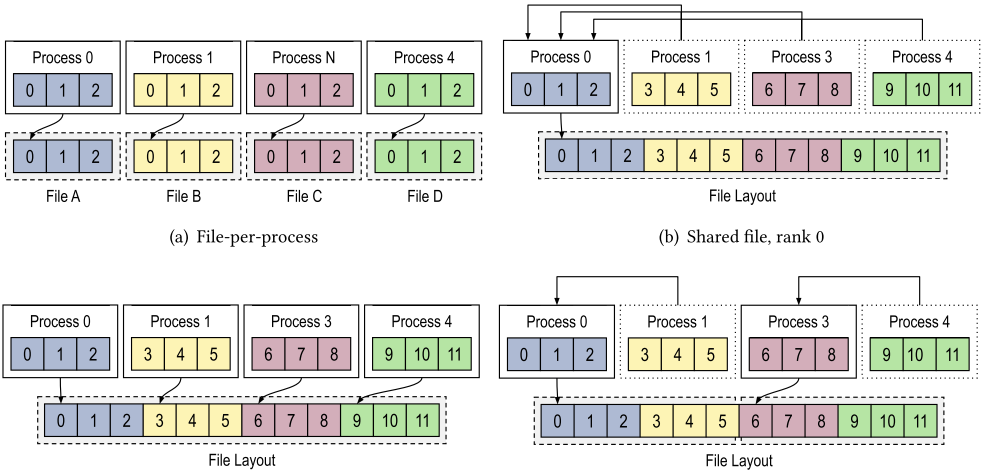
Fig. 8. Taxonomy of features used to describe an I/O access pattern at different layers of the HPC I/O software stack: application side, high-level I/O libraries, middleware layer, and file system. Some features are repeated as they are meaningful across layers of the stack, whereas others are intrinsic to a particular layer. Section 7.2 groups these features based on community usage over the years.

7.1.1 *Operation.*

We can broadly classify the I/O operations as *writes* and *reads*. For *append* operations, the file offset is first positioned at the end of the file using a *seek* operation, then a *write* operation appends the data. The modification of the file offset and the write operation is performed as a single atomic step.

7.1.2 *File Approach.*

There are various scenarios for executing parallel I/O depending on how many processes (MPI ranks) are performing I/O and on how many files are accessed by the processes. In the first scenario, each process of an application issues its operations to an individual file, which is called the *file-per-process* approach, as shown in Figure 9(a). This scenario is represented by having multiple files and multiple writers/readers. When the number of processes is too large, instead of accessing a file per process, data can be aggregated to a small subset of processes, and they can access a smaller number of files. This is called the *subfiling approach* [28, 29]. Although that might achieve performance by harnessing the parallelism inherited from having multiple data servers, future use of those files for post-processing or analysis will have to access those multiple



(a) File-per-process

(b) Shared file, rank 0

(c) Shared file, coordinated

(d) Shared file, aggregators

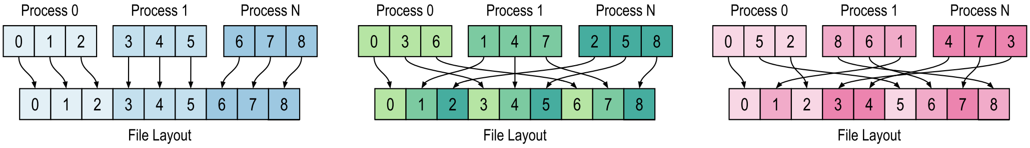
Fig. 9. Number of files and writes/readers.

files to get the required data, as it is scattered. The scalability of this approach is limited when handling metadata operations for extreme-scale applications.

In the second scenario, all processes share a common file (*shared file*). We can further distinguish such a scenario based on the number of writers. At the opposite extreme of the file-per-process, a single writer (commonly rank 0) receives data from many or all ranks (typically using collective MPI calls), rearranges it, and writes it to a single shared file, as depicted in Figure 9(b). The performance of this approach is limited by the memory available in the aggregator node (to receive and handle the data from the entire application); in addition, it cannot utilize the total available bandwidth to the storage servers effectively. Instead of a single writer, we can have a subset of ranks that aggregate and issue the I/O operations, as in Figure 9(d). This strategy implies communication between each rank and its aggregator, and the latter also has an additional data rearrangement step before dispatching the requests to the storage system. Finally, another known approach is having all ranks write their data to the file in a pre-defined non-overlapping location, avoiding inter-rank communication but relying on implicit coordination, as illustrated in Figure 9(c). This performance of this approach is also limited, as there is no coordination or aggregation of I/O requests between ranks on the same compute node.

7.1.3 Spatial Locality. The spatial locality or spatiality refers to the file offsets between consecutive I/O accesses. Typical spatial access patterns are contiguous, strided, or random. This feature directly impacts I/O performance because the storage infrastructure (at hardware and software levels) is affected by the sequentiality of the requests [21]. For instance, file systems can cache or prefetch data when they predict a regular pattern to avoid the costly seek operations between consecutive I/O requests, thereby improving the I/O performance of HPC applications.

We can define spatial locality of an I/O request by its file offset off_i and a size $size_i$ where i identifies the i^{th} request. If the access to a file is sequential, each process accesses contiguous chunks of the file (Figure 10(a)) and the relation $off_{p,i+1} = off_{p,i} + size_{p,i}$ holds for all subsequent requests. However, in a strided (1D, 2D, nD) pattern, each process accesses portions of the data with a fixed-size gap (or stride) between them (Figure 10(b)). The file pointer is incremented by the same amount between each request (i.e., the stride), hence $off_{p,i+1} = off_{p,i} + stride_i$, where $stride_i = \sum_p size_{p,i}$ is often a constant. Furthermore, strided accesses are common when



(a) Shared-file contiguous (b) Shared-file 1D strided (c) Shared-file random

Fig. 10. Spatial locality of I/O requests in the file.

Table 2. Opportunities and Challenges Presented by Each I/O Interface When Used in the Context of HPC

Interface	Opportunities	Challenges
POSIX	• Portability • Strong consistency guarantees • Shallow learning curve (wide adoption)	• Not designed for HPC • Strong consistency vs. scalability • Collective access (locking) • Optimizations (lack of whole application view)
MPI-IO	• Designed for HPC • Relaxed consistency • Flexibility (express patterns natural to applications) • High-level I/O optimizations	• Hard adoption (source-code changes) • Complex tuning
STDIO	• Simple and buffered stream interface	• Not designed for HPC • Scalability • Optimizations (lack of whole application view)

accessing shared files. For the file-per-process approach, it is fairly common for a file to be accessed contiguously. Despite random access (Figure 10(c)) being less common for traditional HPC [177], novel workloads from machine learning applications present such behavior [43, 55, 240, 241] often due to shuffling data between iterations and epochs, which usually results in a large number of concurrent data writes to the file system.

7.1.4 *Interfaces.*

Interfaces seek to provide a convenient and easy-to-use way to access resources. In the context of I/O, these have an important role when accessing files (locally or remotely) by defining APIs and semantics. Furthermore, in HPC, these interfaces should strive to balance usability and high performance, often divergent goals. We can consider three main interfaces that are used directly by applications or high-level libraries to express their access patterns: POSIX I/O, MPI-IO, and STDIO. High-level I/O libraries provide various APIs that simplify mapping data models at the application level with MPI-IO and POSIX interfaces. For instance, HDF5 uses the MPI-IO interface for parallel I/O and POSIX IO for sequential applications. ADIOS and PnetCDF use MPI-IO and POSIX similarly. Hereafter, we briefly discuss each interface and summarize their opportunities and challenges in the context of HPC in Table 2.

POSIX I/O. POSIX (Portable Operating System Interface) is a set of standards defined by IEEE to maintain compatibility among diverse operating systems, allowing an application to obtain basic services from an operating system. POSIX also defines an I/O API used to interact with the file system. Its I/O interface was first introduced in 1988 in the POSIX.1 specification, and it was designed for local file systems accesses. POSIX.1b [92] introduced asynchronous and synchronous behaviors. Despite the fact it was designed for local file systems that used to support sequential applications, POSIX is widely employed by a wide range of applications due to its portability.

However, the portability of POSIX comes with a price when used in HPC. The POSIX semantics define what is and is not guaranteed when its API is used. For instance, it specifies that write operations must be strongly consistent—that is, a *write()* call is required to block the application execution until the system can guarantee that any following *read()* calls will actually read the data that was just written. In the case of HPC, these strict requirements introduce complexity for distributed and parallel file systems where remote processes are unaware of what local processes might be modifying in a file and vice versa. HPC centers often provide POSIX-based parallel file systems (e.g., Lustre and GPFS), which adhere to strong consistency semantics forcing sequential accesses [209]. The required semantics force many parallel file systems to implement distributed locking mechanisms to ensure consistency, thereby penalizing I/O accesses at a large scale. However, modern HPC applications often do not require such strong consistency guarantees [132, 209].

Since POSIX was not designed specifically for HPC applications, it may also impose a burden on the end users. For instance, it is possible to use the shared-file parallel I/O approach. But, the complexity of coordinating parallel accesses, buffering, and flushing is explicitly delegated to the end user. Furthermore, as files are viewed as opaque byte streams, applications are unable to express or hint to the file system about how its data is organized. Such information is essential for data placement strategies and for optimizations. For example, the MPI-IO interface uses such information to express complex accesses and attain high performance. Nonetheless, there were some efforts that sought to extend POSIX I/O to account for HPC needs. Vilayannur et al. [206] designed a proposed POSIX extension to support shared file descriptors/group open, lazy metadata attributes, non-contiguous read/write interfaces, and bulk metadata operations. Such efforts have not been integrated into major storage solutions yet.

MPI-IO. MPI-IO [68] was proposed as an extension to the MPI standard, defining I/O operations by reusing the message passing concepts of MPI. Writing to a file is like sending a message, and reading from a file is like receiving a message. MPI-IO provides a high-level interface to describe the data partitioning among processes, and a collective interface to describe transfers of global data structures between process memories and files. In addition, it supports asynchronous I/O operations. As a result, MPI-IO allows computation to be overlapped with I/O and enables optimization of physical file layout on storage devices [47]. Furthermore, MPI-IO's semantics differ from POSIX's semantics, relaxing some consistency requirements, while offering an atomic mode for applications that rely on stricter semantics.

To express flexible I/O access patterns that are natural to the application, MPI-IO relies on MPI-derived data types. These are used to represent how data is laid out in the memory and also in the file. Furthermore, there are three orthogonal features to data access in MPI-IO: positioning (explicit offset or implicit file pointer), synchronization (blocking or non-blocking), and coordination (independent or collective). All are expressed using file pointers (individual or shared).

The MPI-IO interface is implemented on top of a portable abstract-device interface for parallel I/O called *ADIO* [198], which can be optimized for various file systems. ADIO itself is not intended to be used directly by application programmers but rather as an internal to the implementation of some other user-level I/O interfaces. For instance, ROMIO [201] is a high-performance, portable implementation of MPI-IO optimized for non-contiguous access patterns, which are common in parallel applications. It relies on the portability of ADIO to be used with any MPI implementation (ROMIO is often included as a part of several MPI implementations, e.g., MPICH, Cray MPI, and OpenMPI).

As MPI-IO is layered atop POSIX, it generates complex I/O access patterns. The pattern that reaches the file system may greatly differ from what was initially expressed in the scientific application code due to optimizations and transformations (e.g., collective buffering and data sieving [53, 199]) as requests traverse the I/O stack.

STDIO. The standard I/O library (STDIO) provides a simple and buffered stream I/O interface. It abstracts all file operations into operations on streams of bytes. STDIO comprises the C *stdio.h* family of functions [97] (e.g., *fopen*, *fprintf*, and *fscanf*). However, STDIO functions do not directly support random access to data. In such cases, the application must open a stream, seek to the desired location in the file, and then write/read bytes in sequence from the stream.

Recently, STDIO has been increasingly used for HPC workloads [144, 182], especially for genomics and biology production applications that rely on I/O functions to store sequencing information in text format. Analysis of traces from supercomputer facilities confirmed the noticeably increasing use of STDIO across supercomputer platforms and for a wide range of science domains [17]. The study also revealed that although STDIO can obtain high bandwidths for some transfer sizes, it consistently delivers lower performance than POSIX does across various transfer sizes in Cori (NERSC) and Summit (OLCF) supercomputers, indicating overall poor I/O performance.

7.1.5 I/O Mode. The I/O mode refers to how parallel processes (MPI ranks) access a file: each rank individually or collectively (by a subset of all ranks). Collective operations are readily available in interfaces such as MPI-IO, and these operations provide a big picture of the overall data movement across ranks. These functions require all processes that collectively open the same file to participate in the calls, thus allowing optimizations such as collective buffering and data sieving [199] to improve performance by building larger and contiguous accesses to the underlying storage system.

The I/O mode can directly transform the access pattern perceived by the underlying layer when using collective operations. Instead of each rank issuing its individual operations, the aggregate file access region targeted by a collective I/O call is divided among the aggregators into non-overlapping regions (file regions). In the communication phase, all ranks send their I/O requests to the aggregators based on their file domain. In the I/O phase, aggregators issue the requests to the system. Hence, aggregators effectively merge requests into larger and contiguous ones before percolating to the POSIX layer or the file system.

7.1.6 Synchronicity. Synchronous or blocking I/O routines are not considered successful before an I/O operation is completed. However, asynchronous or non-blocking I/O operations allow applications to hide the cost associated with I/O operations by overlapping it with computation or communication steps, allowing the application to progress. The latter is becoming popular among scientific applications to access large amounts of data and improve user-perceived performance. POSIX and MPI-IO provide asynchronous APIs to write and read data from files. POSIX standard provides the `aio_*` calls, whereas MPI-IO has `MPI_File_i*` calls for independent and collective I/O operations. Some high-level I/O libraries, such as ADIOS and HDF5, also expose those non-blocking interfaces [195]. In contrast, data management systems such as PDC (Proactive Data Containers) [194] offer asynchronous data movement to and from their server nodes through network data transfer. Novel object storage file systems such as DAOS (Distributed Asynchronous Object Storage) [124] were built around the asynchronous concept to deliver performance. The synchronicity feature will help shape the temporal behavior of the application's access pattern.

7.1.7 Temporal Behavior. Toward the automatic detection of poorly performing HPC jobs, Buneci and Reed [27] generated temporal signatures containing performance features from time-series metrics to group applications into two groups: those that performed as expected and those that did not. They combine high-level states provided by users, based on previous executions, with low-level metrics to detect factors affecting performance. Although their approach uses two I/O metrics to build the signature, they do not focus on that; instead, they focus on the combination of CPU, memory, disk, and network usage. Dorier et al. [61, 62] proposed Omnisc'IO, which builds

```
"query": {
    AllField: (("I/O access pattern" OR "I/O characterization" OR "I/O characteristic" OR "I/O signature") AND "HPC"))
}
"filter": {Publication Date: (01/01/2000 TO 12/31/2021)}, {ACM Content: DL}
```

Fig. 11. ACM Digital Library query and filter parameters used in this survey.

```
("Full Text & Metadata":"I/O access pattern") AND ("Full Text & Metadata":"HPC") OR
("Full Text & Metadata":"I/O characterization") AND ("Full Text & Metadata":"HPC") OR
("Full Text & Metadata":"I/O characteristic") AND ("Full Text & Metadata":"HPC") OR
("Full Text & Metadata":"I/O signature") AND ("Full Text & Metadata":"HPC")          Filters Applied: 2000 - 2021
```

Fig. 12. IEEE Xplore query and filter parameters used in this survey.

a grammar-based model of any HPC application I/O behavior to predict future use. They seek to predict when I/O operations will occur—that is, the inter-arrival time between requests and how much data will be accessed, including offset and size within the file. To make time-related predictions, Omnisc'IO stores statistics such as minimum and maximum observed time between requests, the average, and variance and relies on weighted inter-arrival average time to react to changes. From those, they can anticipate whether an operation will immediately follow the current one in a predictable amount of time and whether the time before the subsequent operations is more unpredictable.

White et al. [219] placed a particular focus on I/O by proposing a taxonomy for temporal I/O patterns of HPC jobs to aid in automatically detecting poorly performing jobs. They describe the design of a simple heuristic classification algorithm that categorized jobs based on a very coarse measure of when most of the I/O occurred. The authors observed a small number of common I/O access patterns: primary I/O usage near the start of the job, main I/O usage near the end of the job, I/O activity at the beginning and end but not during the job, low I/O at the start or end but high in the middle, regular activity throughout the job, and regular periodic I/O activity.

7.1.8 Consistency. When checking for overlapping I/O patterns, Wang et al. [211] consider the consistency of I/O operations. They seek to understand whether or not a process ever writes/reads to the same part of a file more than once, whether multiple processes write/read the same part of a file, and the order in which operation occurs in a given offset. For that, they consider read after read (RAR), write after write (WAW), read after write (RAW), and write after read (WAR) metrics to compose the pattern. The consistency policy used by an application can also aid in determining whether caching techniques are feasible or not.

7.2 Community-Based Usage Survey

Seeking to understand how I/O access patterns are approached and used by the broad HPC community to describe their applications, we filtered the ACM Digital Library[1] and IEEE Xplore[2] considering a 20-year window, covering publications between 2000 and 2021 that mention the following: "I/O access pattern," "I/O characterization," "I/O characteristic," or "I/O signature." The text should also refer to "HPC" for any of the terms. After filtering for conferences and journal publications, that search yielded 74 results in ACM and 161 in IEEE. In Figures 11 and 12, we show our queries used in ACM Digital Library and IEEE Xplore, respectively. We classify these papers based on the features used by the authors to describe the I/O access pattern at multiple levels of the I/O stack, as illustrated in Figure 13.

[1] https://dl.acm.org
[2] https://ieeexplore.ieee.org

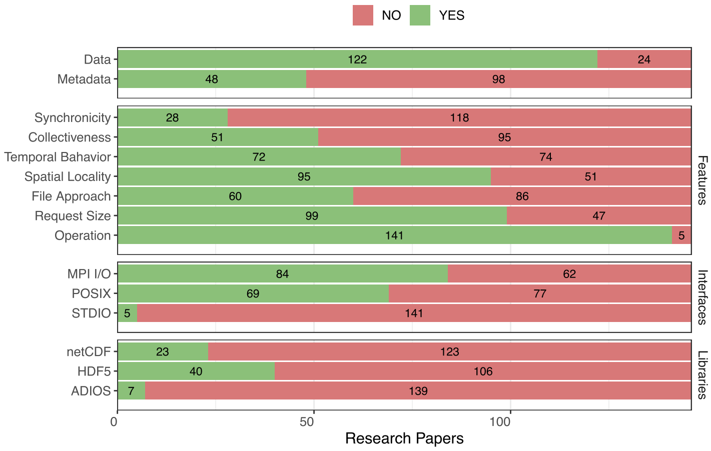


Fig. 13. I/O access pattern features, interfaces, and libraries in ACM Digital Library and IEEE Xplore papers.

Our methodology consisted of looking for common pre-defined keywords in the entire text that is used to describe each I/O access pattern feature. Each manuscript was pre-filtered and manually inspected to avoid false positives. We defined a set of tags corresponding to each feature and its usage. Due to the selection approach and the broad use of the term in correlated areas (e.g., memory), some of the selected papers were not, in fact, relevant to this survey. Therefore, they were later excluded from the analysis. In the end, we considered 146 papers for the analysis presented in this section (62.13% of the 235 results). Specifically, we used the following criteria to filter the initial set of papers:

- The keyword should have been used in the experiments or considered IN the proposed technique or solution.
- Merely citing or using a keyword does not make the paper fall into that classification (e.g., mentioning HDF5 as an interface for IOR does not make it fall into the HDF5 category unless it was used in the evaluation).
- If the feature is used solely to describe related work, that does not make the paper be marked in that category
- Some of those keywords' definitions are overloaded to describe features outside the I/O realm—for instance, memory, communication, or even computing (e.g., synchronous/asynchronous). In such cases, they were not considered as relevant in the context of the I/O access pattern.
- To avoid bias in classification, if the authors did not clearly state a feature, we assume that they did not consider that (unless it is evident from context); in doubt, we assume that it is not used.

In Figure 13, we summarize our findings. When discussing access patterns, 122 (i.e., 83.56%) of the papers cover data operations, whereas only 48 (32.89%) consider metadata operations. However, these do not go into the depth of describing their metadata I/O patterns in detail. Regarding features, operation (96.57%), request size (67.81%), and spatial locality (65.07%) are the ones taken into account by the majority of the research papers. Despite the file approach being strongly related to the spatial locality, the first is not explicitly addressed in 58.90% of the surveyed papers. Collectiveness and synchronicity are the less targeted features when discussing or describing access patterns, and both are related to I/O optimization techniques.

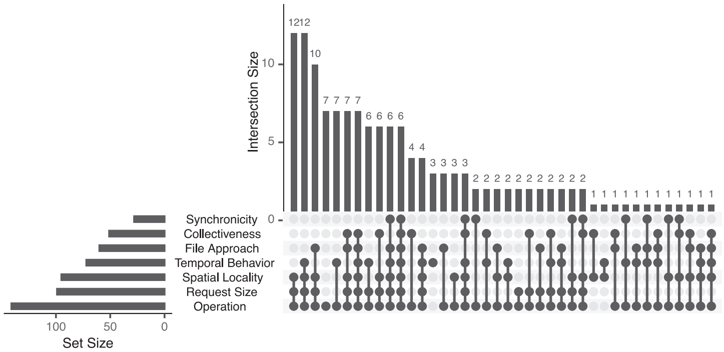


Fig. 14. Analysis of the set of features used to describe an I/O access pattern in ACM Digital Library and IEEE Xplore research papers.

Table 3. Surveyed Papers from ACM Digital Library and IEEE Xplore Grouped by the Features They Use When Discussing I/O Access Patterns

Operation	Request Size	Spatiality	File Approach	Temporal	Collectiveness	Synchronicity	Papers	References
✓	✓	✓	✗	✗	✗	✗	12	[10, 38, 57, 69, 77, 90, 101, 167, 168, 175, 186, 222]
✓	✓	✓	✗	✓	✗	✗	12	[4, 50, 78, 82, 86, 99, 100, 104, 110, 117, 160, 224]
✓	✓	✓	✓	✗	✗	✗	10	[15, 43, 48, 71, 83, 94, 95, 106, 115, 177]
✓	✗	✗	✗	✗	✗	✗	7	[7, 32, 63, 131, 146, 152, 220]
✓	✗	✗	✗	✓	✗	✗	7	[70, 112, 123, 137, 212, 219, 233]
✓	✓	✓	✓	✗	✓	✗	7	[11, 40, 102, 107, 191, 215, 235]
✓	✓	✓	✓	✓	✓	✗	7	[34, 116, 133, 141, 183, 196, 210]
✓	✓	✗	✗	✓	✗	✗	6	[58, 103, 149, 150, 165, 239]
✓	✓	✓	✗	✗	✓	✗	6	[72, 81, 84, 135, 203, 213]
✓	✓	✓	✓	✓	✗	✓	6	[54, 61, 126, 161, 179, 204]
✓	✓	✓	✓	✓	✓	✓	6	[21, 111, 147, 180, 193, 230]
✓	✗	✗	✗	✗	✓	✗	4	[33, 56, 143, 172]
✓	✓	✓	✓	✓	✗	✗	4	[37, 75, 156, 163]
✗	✗	✗	✗	✓	✗	✗	3	[148, 157, 173]
✓	✗	✗	✓	✗	✗	✗	3	[49, 125, 189]
✓	✗	✓	✗	✗	✗	✗	3	[8, 162, 218]
✓	✓	✓	✗	✓	✓	✓	3	[85, 185, 236]

In Figure 14, we depict the intersecting sets of features used in the surveyed research papers, considering the seven features detailed in Figure 13. To summarize all intersections and their distributions, we rely on the UpSet plot, a state-of-the-art visualization technique for the quantitative analysis of intersecting sets and their properties [45, 120]. From this, it becomes clear how not all relevant features are properly described and considered in most of the works and which ones are commonly considered together (e.g., operation, request size, and spatial locality). Table 3 maps these different features and references those works so that readers can find detailed information on how the features were employed in practice under diverse scenarios and science domains.

We also grouped the research papers according to the interface and high-level libraries they use. For interfaces, the majority (57.53%) used MPI-IO in their experiments or explicitly considered that interface when discussing the applicability of the proposed solution or optimization techniques. POSIX follows up with 47.26% and STDIO is targeted by merely 3.42% of the surveyed papers.

Table 4. Summary of Access Pattern Features Exercised by Each Benchmark and I/O Kernel

Name	Synthetic	Kernel(s)	Data	Metadata	Write	Read	Request Size	Independent	Collective	Temporal	Shared File	File-per-process	Synchronous	Asynchronous	Interface
IOR [88]	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	POSIX, MPI-IO, HDF5, HDFS, S3, NCMPI, IME, MMAP, RADOS
MADbench2 [25]	✗	✓	✓	✗	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	POSIX, MPI-IO
IFER [227]	✓	✗	✓	✗	✓	✗	✓	✗	✓	✓	✓	✗	✓	✗	MPI-IO
S3D [39]	✗	✓	✓	✗	✓	✓	✗	✗	✓	✗	✓	✗	✓	✓	PnetCDF
NAS BT-IO [155]	✓	✓	✓	✗	✓	✓	✗	✗	✓	✓	✓	✗	✓	✗	MPI-IO
S3aSim [42]	✗	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✗	✓	✗	MPI-IO
h5bench [122]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	HDF5
HACC-IO [207]	✗	✓	✓	✗	✓	✓	✗	✓	✓	✗	✓	✓	✓	✗	POSIX, MPI-IO
MACSio [151]	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	STDIO, MPI-IO, HDF5
MPI Tile I/O [171]	✓	✗	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗	✓	✗	POSIX, MPI-IO

The check in orange indicates that h5bench does support asynchronous operations; however, it requires the HDF5 ASYNC VOL Connector [195] to be available and enabled.

However, Bez et al. [18] highlight a widespread use of STDIO across a wide range of science domains in HPC applications on both Summit (OLCF) and Cori (NERSC) supercomputers, suggesting a possible new trend due to the shift from traditional numerical simulations to AI/machine learning applications for training and inference while processing and producing ever-increasing amounts of scientific data. Regarding high-level libraries, the majority do not explicitly acknowledge using a particular library, although HDF5 is used by 27.40%.

Summary #6

The community has been using common features (e.g., operation, size, and spatiality) to describe an I/O access pattern, with additional information depending on the targeted layer or optimization context. Furthermore, metadata access is often not as detailed as data access.

8 EXERCISING I/O ACCESS PATTERNS

This section briefly covers existing benchmarks and I/O kernels that are often used in scientific I/O research to exercise access patterns. We describe the features benchmarks used to represent I/O accesses and the I/O workload characteristics of different scientific application kernels.

Table 4 summarizes the benchmarks and I/O kernels used by the community to exercise the HPC I/O stack under diverse data workloads. We group the tools by their representation (exclusively synthetic workloads or extracted as a representative I/O kernel of an application), focus (data or metadata), operation (write or read), support to set the request size, mode (independent or collective operations), temporal behavior, file approach (shared file or file-per-process), and synchronicity (synchronous or asynchronous requests). We also describe the supported I/O interfaces.

IOR [88] is an I/O benchmark to test the performance of parallel storage systems using various interfaces and access patterns. It supports different interfaces or APIs (POSIX, MPI-IO, HDF5, HDFS, S3, NCMPI, IME, MMAP, or RADOS). IOR is flexible enough to express patterns by configuring the operation, the contiguous bytes to write per task (block size), transfer size, number of segments, whether it uses collective or individual operations (where applicable), and whether each task writes to its own file or a shared file.

MADbench2 [25] is an I/O kernel extracted from the MADspec application. MADbench2 allows testing the integrated performance of the I/O, communication, and calculation subsystems of massively parallel architectures under the stresses of a real scientific application. It is derived directly from a large-scale cosmic microwave background data analysis package. It calculates the maximum likelihood angular power spectrum of the cosmic microwave background radiation from a noisy pixelized map of the sky and its pixel-pixel noise correlation matrix. MADbench2 has a regular mode, in which the full code is executed, and an I/O mode where all calculation/communication is replaced with busy work. The kernel has three component functions, each with different access patterns, named *S*, *W*, and *C*. In *S*, N_{bin} writes each of ${N_{pix}}^2$ bytes on N_p processors. In *W*, N_{bin} reads each of ${N_{pix}}^2$ bytes on N_p processors and N_{bin} writes each of ${N_{pix}}^2$ bytes on N_p/N_{gang} processors. In *C*, ${N_{bin}}^2/N_{gang}$ reads each of ${N_{pix}}^2$ bytes on N_p/N_{gang} processors, where N_p defines the number of processes and N_{pix} sets the size of the pseudo-data, in which all component matrices have $N_{pix} \times N_{pix}$ elements. N_{bin} sets the size of the pseudo-dataset composed on N_{bin} component matrices. Finally, N_{gang} sets the level of gang parallelism, allowing MADbench2 to run as a single or multi-gang. In the former, all matrix operations are carried out by being distributed over all processors. The kernel can use the POSIX or MPI-IO interfaces to synchronously or asynchronously issue its I/O operations to a unique or shared file.

IFER is a microbenchmark similar to IOR but instead seeks to provide insights on I/O contention [227]. It splits the ranks into two groups running on two separate sets of nodes to emulate two competing applications. Each group of processes executes a series of collective I/O operations following a pre-defined pattern. Although IFER only provides support for write requests to a shared file by application, it considers two patterns: contiguous and 1D strided. IFER also relies on two additional parameters: the block size, which represents the contiguous bytes to write per process, and the block count. The number of blocks will be continuously written per process in the contiguous pattern. For the strided pattern, the blocks are distributed along the file depending on their offsets. Because its original goal was to study I/O interference, IFER allows users to define the inter-arrival time between the I/O phases.

The S3D I/O kernel [39] is a continuum-scale first principles direct numerical simulation code that solves the compressible governing equations of mass continuity, momenta, energy, and mass fractions of chemical species, including chemical reactions. It creates N checkpoints at regular intervals, where it writes 3D and 4D arrays of doubles into a newly created file. All 3D arrays are partitioned among the MPI processes using block partitioning in all x-y-z dimensions, whereas the fourth dimension (the most significant one) is not partitioned. The kernel can be configured to use PnetCDF blocking or non-blocking APIs. For the latter, a checkpoint has four non-blocking write calls, one per variable, followed by a call to wait and flush the write requests [128].

NAS BT-IO [155] is a benchmark based on the block triagonal (BT) problem of the NAS Parallel Benchmarks (NPB). Each rank is responsible for multiple Cartesian subsets of the dataset, whose number increases as the square root of the number of ranks participating in the computation. The entire solution, consisting of five double-precision words per mesh point, must be written to a file at every five timesteps. In the end, all data belonging to a single timestep must be stored in the same file and must be sorted by vector component, x, y, and z-coordinates.

S3aSim [42] is an I/O kernel based on a sequence similarity search framework. It uses a master-slave parallel programming model with database segmentation, mimicking the mpiBLAST [52] access pattern. Given input query sequences, S3aSim divides up the database sequences into fragments. Workers request a query and fragment information from the master and search the query against the database fragment assigned. The results are sent to the master to be sorted and then written to a single shared file. Without synchronizing after every query, this application uses individual I/O operations to write data to a single shared file.

Parallel I/O Kernels (https://github.com/hpc-io/PIOK) provides the parallel I/O portion of various scientific simulation codes that use HDF5. These kernels have been expanded with h5bench to cover a variety of HDF5 I/O patterns. h5bench [122] is a set of HDF5 I/O kernels representing I/O patterns that are commonly used in HDF5 applications on HPC systems. It provides a framework to test, exercise, and tune I/O performance using novel features introduced in HDF5 and understanding how the library performs in different machines under such I/O workloads. It measures I/O performance from various aspects, including the raw and observed I/O time and rate.

HACC-IO [207] is a kernel extracted from the HACC (Hardware Accelerated Cosmology Code) cosmology framework (Gordon Bell Award Finalist 2012, 2013). It uses the N-body to simulate collisionless fluids under the influence of gravity. The kernel includes the checkpoint, restart, and analysis outputs produced by the simulation. Hence, it is quite I/O intensive. It also supports both POSIX and MPI-IO (with independent and collective operations) interfaces. Regarding the file approach, HACC-IO can be configured to write to a single shared file, a file-per-process, or a mix of both (i.e., file per group). It only takes as an input argument the number of particles (n), where each particle is composed of seven 4-byte floats, an 8-byte integer, and a 2-byte integer. Thus, each process writes/reads $n \times 38$ bytes.

MACSio (Multi-purpose, Application-Centric, Scalable I/O Proxy Application) [151] was built for I/O performance testing and evaluation of tradeoffs in data models, I/O library interfaces, and parallel I/O paradigms for multi-physics HPC applications. It differs from other benchmarks in the sense that it actually constructs and marshals data as real data objects commonly used in scientific computing applications. Hence, MACSio allows closely mimicking I/O workloads from the multiphysics domain, where data object distribution and composition vary within and across parallel processes. It also supports representing the data using multiple file approaches (segmented and strided single shared file, multiple independent files, or file-per-process), using independent and collective operations.

MPI Tile I/O [171] is a benchmark suited to test the performance of an underlying MPI-IO and file system implementation under a non-contiguous access workload. It logically divides a data file into a dense 2D set of tiles based on the number of tiles in the x and y dimensions. It allows the end user to configure the number of elements in each tile dimension and the size of an element. It can express overlap elements by defining how many of them are shared between adjacent tiles in each dimension. MPI Tile I/O has support for collective I/O allowing fine-tuning of the list of nodes involved in the aggregation.

Toward emulating scientific deep learning workloads that are becoming popular on HPC systems, DLIO [55] provides an I/O benchmark suite. DLIO supports various scientific deep learning applications, including Neutrino and Cosmic Tagging with UNet, FFN (Distributed Flood Filling Networks), CNN (Convolutional Neural Networks), CosmoFlow for cosmology datasets, FRNN (Fusion Recurrent Neural NetN), and CANDLE (Cancer Distributed Learning Environment). DLIO allows reading data from different file formats and APIs, such as HDF5, CSV, and tfrecord formats.

Summary #7

A plethora of benchmarks and I/O kernels are available to the community to exercise access patterns at different layers of the stack. There is not a single one that encompasses all features; however, when combined, they cover distinct features, interfaces, and application data models.

9 PROFILING AND VISUALIZING I/O ACCESS PATTERNS

Darshan [34] is a popular tool to collect I/O profiling information from applications in a lightweight manner. Darshan aggregates I/O profile information to provide valuable insights without adding overhead or perturbing application behavior. It also provides an extended tracing module (DXT) [223] to obtain a fine-grained view of the application behavior to understand I/O performance issues. Once enabled, DXT collects detailed traces from the POSIX and MPI-IO layers reporting the operation (write/read), the rank that issued the call, the segment, the offset in the file, and the size of each request. It also captures the start and end timestamps of all operations issued by each rank.

Recorder [211] is a multi-level I/O tracing framework to capture I/O function calls at multiple levels of the I/O stack, including HDF5, MPI-IO, and POSIX I/O. As a shared library, it requires no modification or recompilation of the application and allows users to control tracing levels. Recorder captures timestamps, function names, and all parameters from intercepted I/O calls using function interposing to intercept I/O calls.

TAU (Tuning and Analysis Utilities) [178] is an integrated toolkit for performance instrumentation, measurement, and analysis. It can capture file I/O (serial and parallel), communication, memory, and CPU. Regarding I/O, TAU can handle profiling and tracing, observing inclusive (including all child regions) and exclusive (for a region only) measurements. It uses library wrapping to characterize I/O performance, which helps automate the instrumentation of external I/O packages and libraries. Thus, TAU can capture POSIX and MPI-IO and instrument libraries such as HDF5.

IOPin [107] proposes a dynamic instrumentation framework to understand the complex interactions across different I/O layers from applications to the underlying PFS. It leverages Pin lightweight binary instrumentation using probe mode to instrument applications and components of the I/O stack, providing a hierarchical view for parallel I/O. Their implementation supports the MPI library and PVFS. Their approach traces and instruments only the process that has been identified by Pin to have the maximum I/O latency. This dynamic instruction reduces the overhead and focuses on detecting only one critical I/O path that affects performance in the stack. The metrics provided by IOPin include latency, disk throughput, number of requests from client to server, and number of disk accesses for each request. However, it does not provide a characterization of each I/O request.

ScalaIOTrace [142] is a multi-level I/O tracing tool based on ScalaTrace [158], an MPI communication tracing framework for parallel applications. ScalaIOTrace supports both MPI-IO and POSIX I/O interposition. MPI-IO tracing relies on the MPI profiling layer (PMPI) to intercept and collect MPI calls. At the same time, POSIX is captured via wrappers using GNUlink time entry interpositioning with domain-specific parameter compression, similar to PMPI. This tracing tool captures I/O events as singletons, vectors, and regular section descriptors to describe the application behavior. Those are stored in a single, lossless, and order-preserving trace file. Their goal is to generate a trace that can be extrapolated into target sizes of nodes and replayed to assert I/O scalability.

Score-P [109] is a measurement tool suite for profiling and event tracing of HPC applications. The instrumentation allows users to insert measurement probes in their codes to collect performance-related data when triggered by linking against several provided runtime libraries for serial execution, OpenMP or MPI parallelism, or hybrid combinations. It also allows selective filtering in both profiling and tracing mode to restrict the recording to specific regions. For I/O operations, Score-P can collect data on POSIX I/O (e.g., `open`/`close`), POSIX asynchronous I/O (e.g., `aio_read`/`aio_write`), STDIO (e.g., `fopen`/`fclose`), and MPI-IO calls. Visualizing Score-P output files using Periscope [12], Scalasca [73], TAU [154], and Vampir [108] is possible. Periscope is an online profiling analysis that evaluates performance properties and tests hypotheses about typical performance problems. Scalasca allows post-mortem analysis of event traces and automatically detects performance-critical situations. It is also possible to use the TAU visualization toolset

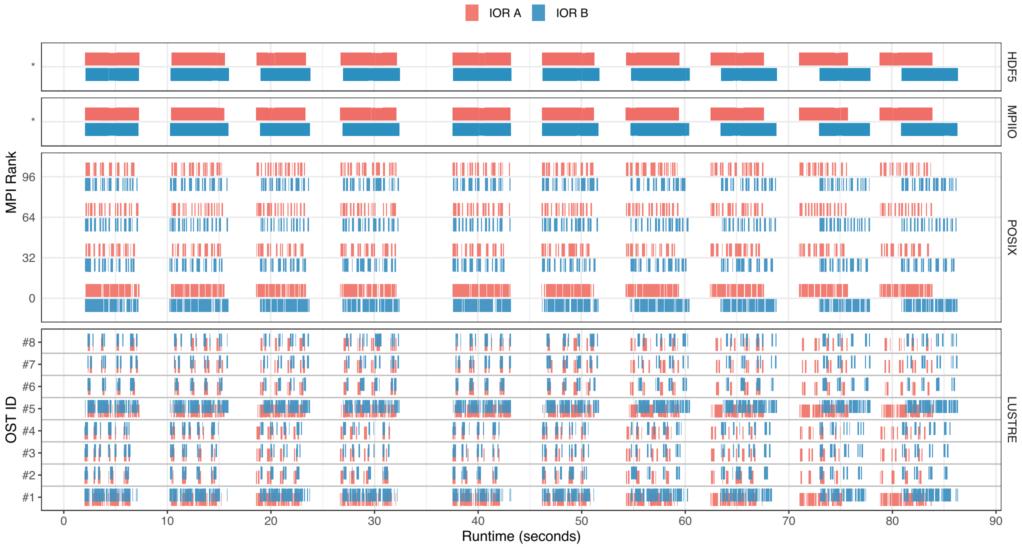


Fig. 15. Two concurrent IOR instances using DXT Explorer. We depict the access pattern from the high-level library (HDF5) and their corresponding transformations until they reach the POSIX layer and the underlying OSTs in Lustre.

to correlate performance data collected with Score-P or Vampir, which works as a post-mortem interactive event trace visualization software.

DXT Explorer [19] is an interactive web-based log analysis tool to visualize Darshan DXT traces and help in understanding the I/O behavior of applications. The tool adds an interactive component to Darshan trace analysis that can aid researchers, developers, and end users to visually inspect their applications' I/O behavior, zoom in on areas of interest, and have a clear picture of where the I/O problem is.

Gaps in Visualizing Access Pattern Transformations

As discussed in Section 3, the way the application issues its I/O requests will differ from what the intermediate layers and the file system actually perceive. To illustrate the transformations an application's I/O requests undergo as they traverse the stack, we use Darshan traces and DXT Explorer to visualize the I/O access pattern at different levels.

Figure 15 depicts such transformations, and Figure 16 zooms in on the first 2 seconds of the experiment reported in Figure 15. In the experiment in Figure 15, we have two instances of IOR (one in red and another in blue). We executed each one in two non-overlapping sets of 16 compute nodes, with 8 ranks per node, totaling 128 ranks. We configured IOR to write 10 iterations of one segment with a 32-MB block size using 4-MB transfer sizes to a shared file using the HDF5 API and collective MPI operations. Both instances were started simultaneously. We collected profile and tracing data using Darshan. As Darshan Extended Tracing does not yet capture fine-grained information about high-level libraries, such as HDF5, we rely upon manually instrumenting the code to collect timestamps before performing the write operations and after the dataset is completely written to a given file. We condensed both plot facets as all ranks collectively issued the I/O calls to the MPI-IO layer. We represented these collective calls by the star symbol on the y-axis.

As far as the application is concerned, its data in memory is a 1D dataset represented by HDF5. HDF5 will define a hyperslab based on the start offset, count, stride, and block to access the data.

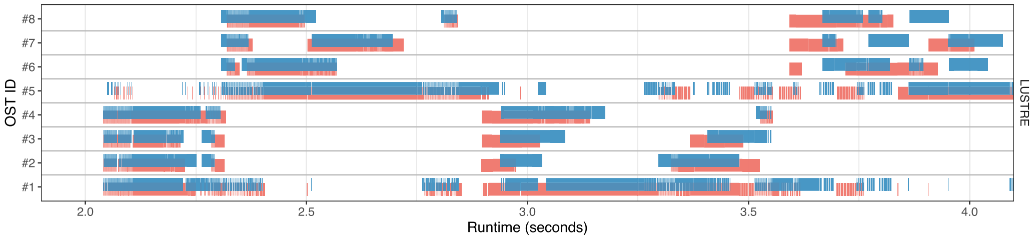


Fig. 16. I/O request from the two concurrent IOR instances (one in red another in blue) as they arrive in each of the eight Lustre storage servers. We zoom in on the first 2 seconds of the experiment reported in Figure 15.

A hyperslab represents a portion of the datasets that can be a logically contiguous collection of points in a dataspace or a regular pattern of points or blocks in a dataspace. In our experiments, for a shared file, IOR defines the start offset as `offset` module `segmentSize`, count as 1, and a stride and a block equal to the transfer size (i.e., 4 MB). However, once the requests reach the MPI-IO layer, they are further broken down by the four collective aggregators into a larger number of 1-MB POSIX requests, considering the underlying PFS striping configuration before sending them to each storage device. We have defined Lustre to use 1-MB stripes over eight servers to make it easier to visualize. Once we delve into lower levels of the I/O stack, we are to lose contextual information from the applications and start to observe the effect of natural interference in this shared storage infrastructure. For instance, if we glance at one OST, the requests arrive at the storage servers in an interleaved fashion, coming from the two applications that the file system is unaware of. At this point, the original contiguous requests issued by the application using 4-MB requests arrive at the server much smaller (in 1-MB requests) and with a different spatiality (non-contiguous).

Furthermore, it is essential to highlight the inter-application interference caused by other applications sharing those data servers. Figure 15 clearly depicts how two identical applications that started simultaneously begin to diverge in time toward the end of our experiment. Such observation also highlights the importance of taking into account temporal features when discussing access patterns.

Summary #8

Different tools extract and visualize I/O access patterns from coarse-grained profilers to fine-grained traces as I/O requests pass through the stack. However, we could not find a complete solution that allows observing patterns and all of their transformations in the context of each layer. Because of the complexity of the current stack, this gap might not easily reflect the root causes of bottlenecks.

10 CONCLUSION

The HPC I/O stack has been complex due to multiple layers of hardware and software, their various tuning options, and inter-dependencies among the layers. This survey extensively discussed the overloaded “I/O access pattern” terminology used to describe how accesses are done from the major layers of the HPC I/O stack, covering the high-level models used by scientific applications, and how those are represented by high-level I/O libraries and translated by middleware libraries before reaching the PFS. We have also highlighted I/O benchmarks and kernels employed to exercise access patterns in different levels, alongside existing tools to visualize those patterns using profiling and tracing.

Harnessing the I/O community's knowledge over the past 20 years, we surveyed 146 papers from ACM Digital Library and IEEE Xplore to propose a baseline taxonomy that could define an application's I/O access patterns. Our effort targets bringing a consensus to the varying ways to describe a pattern based on features already used by the community over the years, serving as a common ground among the end user, application developers, and system administrators when discussing, proposing, and applying I/O tuning strategies to improve I/O performance.

Furthermore, the existing I/O stack exposes a plethora of tunable parameters and enables different, often complementary, optimization techniques to improve performance. However, there is little to no guidance to developers and end users on how and when to apply them. Besides the lack of knowledge that those options are available and could help for a particular set of access patterns, to the best of our own knowledge, there has not been a single set of instructions to define a set of tuning parameters. Reaching a list of best practices, even for a single system, is challenging due to various factors affecting I/O performance. Finally, not having a common ground to identify and refer to access patterns could add to this complexity and makes it difficult to map I/O access patterns to their performance behaviors and then to optimization strategies.

As the HPC platforms become more complex and specialized to host novel applications from machine learning to scientific workflows, it becomes paramount for those systems that seek to auto-tune their parameters to accurately detect the I/O access patterns at runtime. An established taxonomy can help bridge the gap among metric collection, access patterns representation, and the application of AI-based and automatic tuning mechanisms to navigate the complex parameter space, seeking optimizations and configurations to apply for an observed application workload.

Consequently, despite having tools to collect profiles and metrics about I/O performance and features that can be used to describe the application's access patterns at different layers of the HPC I/O stack, there are still gaps between visualizing and understanding what the application is doing, identifying the bottlenecks, and correctly reshaping its pattern to perform better in the system. Reporting and automatically mapping performance problems into actionable items based on the observed pattern require novel tools, models, and further R&D.

ACKNOWLEDGMENTS

The work presented in this article was the result of a collaboration between the Myriads project team at Inria and Lawrence Berkeley National Laboratory in the framework of the Hermes Associate team.

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. The research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under contract number DE-AC02-05CH11231. This research also used resources from 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 number DE-AC05-00OR22725. S. Ibrahim's research was partly supported by the ANR KerStream project (ANR-16-CE25-0014-01). This research also received funding from The Ohio State University (AWD-114169).

REFERENCES

[1] Sadaf R. Alam, Hussein N. El-Harake, Kristopher Howard, Neil Stringfellow, and Fabio Verzelloni. 2011. Parallel I/O and the metadata wall. In *Proceedings of the 6th Workshop on Parallel Data Storage (PDSW'11)*. ACM, New York, NY, 13–18. https://doi.org/10.1145/2159352.2159356

[2] Nawab Ali, Philip Carns, Kamil Iskra, Dries Kimpe, Samuel Lang, Robert Latham, Robert Ross, Lee Ward, and P. Sadayappan. 2009. Scalable I/O forwarding framework for high-performance computing systems. In *Proceedings of*

the 2009 IEEE International Conference on Cluster Computing and Workshops. IEEE, Los Alamitos, CA, 1–10. https://doi.org/10.1109/CLUSTR.2009.5289188

[3] George Almási, Ralph Bellofatto, José Brunheroto, Călin Caşcaval, José G. Castanos, Luis Ceze, Paul Crumley, C. Christopher Erway, Joseph Gagliano, Derek Lieber, Xavier Martorell, José E. Moreira, and Alda Sanomiya. 2003. An overview of the Blue Gene/L system software organization. In *Euro-Par 2003 Parallel Processing*. Lecture Notes in Computer Science, Vol. 2790, 543–555. https://doi.org/10.1007/978-3-540-45209-6_79

[4] Jonathon Anderson, Patrick J. Burns, Daniel Milroy, Peter Ruprecht, Thomas Hauser, and Howard Jay Siegel. 2017. Deploying RMACC Summit: An HPC resource for the Rocky Mountain Region. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success, and Impact (PEARC'17)*. ACM, New York, NY, Article 8, 7 pages. https://doi.org/10.1145/3093338.3093379

[5] I. Antcheva, M. Ballintijn, B. Bellenot, M. Biskup, R. Brun, N. Buncic, Ph. Canal, D. Casadei, O. Couet, V. Fine, L. Franco, G. Ganis, A. Gheata, D. Gonzalez Maline, M. Goto, J. Iwaszkiewicz, A. Kreshuk, D. Marcos Segura, R. Maunder, L. Moneta, A. Naumann, E. Offermann, V. Onuchin, S. Panacek, F. Rademakers, P. Russo, and M. Tadel. 2009. ROOT - A C++ framework for petabyte data storage, statistical analysis and visualization. *Computer Physics Communications* 180, 12 (2009), 2499–2512. https://doi.org/10.1016/j.cpc.2009.08.005

[6] Guilllaume Aupy, Ana Gainaru, and Valentin Le Fèvre. 2019. I/O scheduling strategy for periodic applications. *ACM Transactions on Parallel Computing* 6, 2 (July 2019), Article 7, 26 pages. https://doi.org/10.1145/3338510

[7] Robert A. Ballance and Jonathan Cook. 2010. Monitoring MPI programs for performance characterization and management control. In *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC'10)*. ACM, New York, NY, 2305–2310. https://doi.org/10.1145/1774088.1774566

[8] Jiwoo Bang, Chungyong Kim, Kesheng Wu, Alex Sim, Suren Byna, Sunggon Kim, and Hyeonsang Eom. 2020. HPC workload characterization using feature selection and clustering. In *Proceedings of the 3rd International Workshop on Systems and Network Telemetry and Analytics (SNTA'20)*. ACM, New York, NY, 33–40. https://doi.org/10.1145/3391812.3396270

[9] Ayşe Bağbaba. 2020. Improving collective I/O performance with machine learning supported auto-tuning. In *Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW'20)*. IEEE, Los Alamitos, CA, 814–821. https://doi.org/10.1109/IPDPSW50202.2020.00138

[10] Babak Behzad, Surendra Byna, Prabhat, and Marc Snir. 2015. Pattern-driven parallel I/O tuning. In *Proceedings of the 10th Parallel Data Storage Workshop (PDSW'15)*. ACM, New York, NY, 43–48. https://doi.org/10.1145/2834976.2834977

[11] Babak Behzad, Surendra Byna, Prabhat, and Marc Snir. 2019. Optimizing I/O performance of HPC applications with autotuning. *ACM Transactions on Parallel Computing* 5, 4 (March 2019), Article 15, 27 pages. https://doi.org/10.1145/3309205

[12] Shajulin Benedict, Ventsislav Petkov, and Michael Gerndt. 2010. PERISCOPE: An online-based distributed performance analysis tool. In *Tools for High Performance Computing 2009*, Matthias S. Müller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel (Eds.). Springer, Berlin, Germany, 1–16.

[13] John Bent, Garth Gibson, Gary Grider, Ben McClelland, Paul Nowoczynski, James Nunez, Milo Polte, and Meghan Wingate. 2009. PLFS: A checkpoint filesystem for parallel applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage, and Analysis*. 1–12. https://doi.org/10.1145/1654059.1654081

[14] Marshall Bern and David Eppstein. 1992. Mesh generation and optimal triangulation. In *Computing in Euclidean Geometry*, Ding-Zhu Du and Frank Hwang (Eds.). Lecture Notes Series on Computing. World Scientific, Singapore, 23–90. https://doi.org/10.1142/9789814355858_0002

[15] Jean Luca Bez, Francieli Zanon Boito, Ramon Nou, Alberto Miranda, Toni Cortes, and Philippe O. A. Navaux. 2019. Detecting I/O access patterns of HPC workloads at runtime. In *Proceedings of the 2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'19)*. IEEE, Los Alamitos, CA, 80–87. https://doi.org/10.1109/SBAC-PAD.2019.00025

[16] Jean Luca Bez, Francieli Zanon Boito, Lucas Mello Schnorr, Philippe Olivier Alexandre Navaux, and Jean-François Méhaut. 2017. TWINS: Server access coordination in the I/O forwarding layer. In *Proceedings of the 2017 25th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'17)*. IEEE, Los Alamitos, CA, 116–123. https://doi.org/10.1109/PDP.2017.61

[17] 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 multi-layer supercomputer I/O subsystems under production load. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing (HPDC'22)*. ACM, New York, NY, 43–55. https://doi.org/10.1145/3502181.3531461

[18] 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 multi-layer supercomputer I/O subsystems under production load. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing (HPDC'22)*. ACM, New York, NY, 43–55. https://doi.org/10.1145/3502181.3531461

[19] 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: Connecting the dots using interactive log analysis. In *Proceedings of the 2021 IEEE/ACM 6th International Parallel Data Systems Workshop (PDSW'21)*. IEEE, Los Alamitos, CA, 15–22. https://doi.org/10.1109/PDSW54622.2021.00008

[20] Jean Luca Bez, Francieli Zanon Boito, Ramon Nou, Alberto Miranda, Toni Cortes, and Philippe O. A. Navaux. 2020. Adaptive request scheduling for the I/O forwarding layer using reinforcement learning. *Future Generation Computer Systems* 112 (2020), 1156–1169. https://doi.org/10.1016/j.future.2020.05.005

[21] Francieli Zanon Boito, Eduardo C. Inacio, Jean Luca Bez, Philippe O. A. Navaux, Mario A. R. Dantas, and Yves Denneulin. 2018. A checkpoint of research on parallel I/O for high-performance computing. *ACM Computing Surveys* 51, 2 (March 2018), Article 23, 35 pages. https://doi.org/10.1145/3152891

[22] Francieli Zanon Boito, Rodrigo Virote Kassick, Philippe O. A. Navaux, and Yves Denneulin. 2013. AGIOS: Application-guided I/O scheduling for parallel file systems. In *Proceedings of the 2013 International Conference on Parallel and Distributed Systems*. IEEE, Los Alamitos, CA, 43–50. https://doi.org/10.1109/ICPADS.2013.19

[23] Francieli Zanon Boito, Rodrigo Virote Kassick, Philippe O. A. Navaux, and Yves Denneulin. 2016. Automatic I/O scheduling algorithm selection for parallel file systems. *Concurrency and Computation: Practice and Experience* 28, 8 (2016), 2457–2472. https://doi.org/10.1002/cpe.3606

[24] Francieli Zanon Boito, Ramon Nou, Laércio Lima Pilla, Jean Luca Bez, Jean-François Méhaut, Toni Cortes, and Philippe O. A. Navaux. 2019. On server-side file access pattern matching. In *Proceedings of the 2019 International Conference on High Performance Computing Simulation (HPCS'19)*. IEEE, Los Alamitos, CA, 217–224. https://doi.org/10.1109/HPCS48598.2019.9188092

[25] Julian Borrill, Leonid Oliker, John Shalf, and Hongzhang Shan. 2007. Investigation of leading HPC I/O performance using a scientific-application derived benchmark. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing (SC'07)*. ACM, New York, NY, Article 10, 12 pages. https://doi.org/10.1145/1362622.1362636

[26] David A. Boyuka, Sriram Lakshminarasimhan, Xiaocheng Zou, Zhenhuan Gong, John Jenkins, Eric R. Schendel, Norbert Podhorszki, Qing Liu, Scott Klasky, and Nagiza F. Samatova. 2014. Transparent in situ data transformations in ADIOS. In *Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGRID'14)*. IEEE, Los Alamitos, CA, 256–266. https://doi.org/10.1109/CCGrid.2014.73

[27] Emma S. Buneci and Daniel A. Reed. 2008. Analysis of application heartbeats: Learning structural and temporal features in time series data for identification of performance problems. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC'08)*. IEEE, Los Alamitos, CA, Article 52, 12 pages.

[28] Suren Byna, M. Scot Breitenfeld, Bin Dong, Quincey Koziol, Elena Pourmal, Dana Robinson, Jerome Soumagne, Houjun Tang, Venkatram Vishwanath, and Richard Warren. 2020. ExaHDF5: Delivering efficient parallel I/O on exascale computing systems. *Journal of Computer Science and Technology* 35, 1 (Jan. 2020), 145–160. https://doi.org/10.1007/s11390-020-9822-9

[29] Suren Byna, Mohamad Chaarawi, Quincey Koziol, John Mainzer, and Frank Willmore. 2017. Tuning HDF5 subfiling performance on parallel file systems. In *Proceedings of the 2017 Cray User Group Meeting*. https://www.osti.gov/biblio/1398484

[30] Surendra Byna, Yong Chen, Xian-He Sun, Rajeev Thakur, and William Gropp. 2008. Parallel I/O prefetching using MPI file caching and I/O signatures. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC'08)*. IEEE, Los Alamitos, CA, 1–12. https://doi.org/10.1109/SC.2008.5213604

[31] Wenxiang Cai, Jianxing Zhao, Xiaosong Wu, and Bo Sun. 2009. Influence of radiation model on numerical prediction of two-phase reaction flow. In *Proceedings of the 2009 International Conference on Computational Intelligence and Software Engineering*. 1–4. https://doi.org/10.1109/CISE.2009.5366244

[32] Jinrui Cao, Om Rameshwar Gatla, Mai Zheng, Dong Dai, Vidya Eswarappa, Yan Mu, and Yong Chen. 2018. PFault: A general framework for analyzing the reliability of high-performance parallel file systems. In *Proceedings of the 2018 International Conference on Supercomputing (ICS'18)*. ACM, New York, NY, 1–11. https://doi.org/10.1145/3205289.3205302

[33] André Ramos Carneiro, Jean Luca Bez, Carla Osthoff, Lucas Mello Schnorr, and Philippe O. A. Navaux. 2021. HPC data storage at a glance: The Santos Dumont experience. In *Proceedings of the 2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'21)*. IEEE, Los Alamitos, CA, 157–166. https://doi.org/10.1109/SBAC-PAD53543.2021.00027

[34] 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* 7, 3 (Oct. 2011), Article 8, 26 pages. https://doi.org/10.1145/2027066.2027068

[35] Philip Carns, Robert Latham, Robert Ross, Kamil Iskra, Samuel Lang, and Katherine Riley. 2009. 24/7 characterization of petascale I/O workloads. In *Proceedings of the 2009 IEEE International Conference on Cluster Computing and Workshops (CLUSTER'09)*. IEEE, Los Alamitos, CA, 1–10. https://doi.org/10.1109/CLUSTR.2009.5289150

[36] Philip H. Carns, Walter B. Ligon, Robert B. Ross, and Rajeev Thakur. 2000. PVFS: A parallel file system for Linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference (ALS'00)*, Volume 4. 28.

[37] Yan-Tyng Sherry Chang, Henry Jin, and John Bauer. 2016. Methodology and application of HPC: I/O characterization with MPIProf and IOT. In *Proceedings of the 2016 5th Workshop on Extreme-Scale Programming Tools (ESPT'16)*. IEEE, Los Alamitos, CA, 1–8. https://doi.org/10.1109/ESPT.2016.005

[38] Junjie Chen, Philip C. Roth, and Yong Chen. 2013. Using pattern-models to guide SSD deployment for big data applications in HPC systems. In *Proceedings of the 2013 IEEE International Conference on Big Data*. IEEE, Los Alamitos, CA, 332–337. https://doi.org/10.1109/BigData.2013.6691592

[39] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. S. Yoo. 2009. Terascale direct numerical simulations of turbulent combustion using S3D. *Computational Science & Discovery* 2, 1 (Jan. 2009), 015001. https://doi.org/10.1088/1749-4699/2/1/015001

[40] Steven W. D. Chien, Artur Podobas, Ivy B. Peng, and Stefano Markidis. 2020. tf-Darshan: Understanding fine-grained I/O performance in machine learning workloads. In *Proceedings of the 2020 IEEE International Conference on Cluster Computing (CLUSTER'20)*. IEEE, Los Alamitos, CA, 359–370. https://doi.org/10.1109/CLUSTER49012.2020.00046

[41] Avery Ching, Wei-Keng Liao, Alok Choudhary, Robert Ross, and Lee Ward. 2007. Noncontiguous locking techniques for parallel file systems. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing (SC'07)*. ACM, New York, NY, Article 26, 12 pages. https://doi.org/10.1145/1362622.1362658

[42] A. Ching, Wu-Chun Feng, Heshan Lin, Xiaosong Ma, and A. Choudhary. 2006. Exploring I/O strategies for parallel sequence-search tools with S3aSim. In *Proceedings of the 2006 15th IEEE International Conference on High Performance Distributed Computing*. IEEE, Los Alamitos, CA, 229–240. https://doi.org/10.1109/HPDC.2006.1652154

[43] Fahim Chowdhury, Yue Zhu, Todd Heer, Saul Paredes, Adam Moody, Robin Goldstone, Kathryn Mohror, and Weikuan Yu. 2019. I/O characterization and performance evaluation of beeGFS for deep learning. In *Proceedings of the 48th International Conference on Parallel Processing (ICPP'19)*. ACM, New York, NY, Article 80, 10 pages. https://doi.org/10.1145/3337821.3337902

[44] Giuseppe Congiu, Sai Narasimhamurthy, Tim Suss, and Andre Brinkmann. 2016. Improving collective I/O performance using non-volatile memory devices. In *Proceedings of the 2016 IEEE International Conference on Cluster Computing (CLUSTER'16)*. IEEE, Los Alamitos, CA, 120–129. https://doi.org/10.1109/CLUSTER.2016.37

[45] Jake R. Conway, Alexander Lex, and Nils Gehlenborg. 2017. UpSetR: An R package for the visualization of intersecting sets and their properties. *Bioinformatics* 33, 18 (06 2017), 2938–2940. https://doi.org/10.1093/bioinformatics/btx364

[46] Peter Corbett, Dror Feitelson, Sam Fineberg, Yarsun Hsu, Bill Nitzberg, Jean-Pierre Prost, Marc Snir, Bernard Traversat, and Parkson Wong. 1995. Overview of the MPI-IO parallel I/O interface. In *Proceedings of the 3rd Workshop on I/O in Parallel and Distributed Systems (IPPS'95)*. 1–15.

[47] Peter Corbett, Dror Feitelson, Sam Fineberg, Yarsun Hsu, Bill Nitzberg, Jean-Pierre Prost, Marc Snirt, Bernard Traversat, and Parkson Wong. 1996. Overview of the MPI-IO parallel I/O interface. In *Input/Output in Parallel and Distributed Computer Systems*. Kluwer International Series in Engineering and Computer Science, Vol. 362. Springer, 127–146. https://doi.org/10.1007/978-1-4613-1401-1_5

[48] Chuck Cranor, Milo Polte, and Garth Gibson. 2013. Structuring PLFS for extensibility. In *Proceedings of the 8th Parallel Data Storage Workshop (PDSW'13)*. ACM, New York, NY, 20–26. https://doi.org/10.1145/2538542.2538564

[49] Dong Dai, Yong Chen, Philip Carns, John Jenkins, and Robert Ross. 2017. Lightweight provenance service for high-performance computing. In *Proceedings of the 2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT'17)*. IEEE, Los Alamitos, CA, 117–129. https://doi.org/10.1109/PACT.2017.14

[50] Dong Dai, Yong Chen, Dries Kimpe, and Robert Ross. 2014. Two-choice randomized dynamic I/O scheduler for object storage systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'14)*. IEEE, Los Alamitos, CA, 635–646. https://doi.org/10.1109/SC.2014.57

[51] Marco Dantas, Diogo Leitão, Peter Cui, Ricardo Macedo, Xinlian Liu, Weijia Xu, and João Paulo. 2022. Accelerating deep learning training through transparent storage tiering. In *Proceedings of the 2022 22nd IEEE International Symposium on Cluster, Cloud, and Internet Computing (CCGrid'22)*. 21–30. https://doi.org/10.1109/CCGrid54584.2022.00011

[52] A. E. Darling, L. Carey, and W. C. Feng. 2003. The design, implementation, and evaluation of mpiBLAST. In *Proceedings of the ClusterWorld Conference and Expo*. https://www.osti.gov/biblio/976625

[53] Juan Miguel del Rosario, Rajesh Bordawekar, and Alok Choudhary. 1993. Improved parallel I/O via a two-phase run-time access strategy. *ACM SIGARCH Computer Architecture News* 21, 5 (Dec. 1993), 31–38. https://doi.org/10.1145/165660.165667

[54] Hariharan Devarajan, Anthony Kougkas, Prajwal Challa, and Xian-He Sun. 2018. Vidya: Performing code-block I/O characterization for data access optimization. In *Proceedings of the 2018 IEEE 25th International Conference on High Performance Computing (HiPC'18)*. 255–264. https://doi.org/10.1109/HiPC.2018.00036

[55] Hariharan Devarajan, Huihuo Zheng, Anthony Kougkas, Xian-He Sun, and Venkatram Vishwanath. 2021. DLIO: A data-centric benchmark for scientific deep learning applications. In *Proceedings of the 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud, and Internet Computing (CCGrid'21)*. 81–91. https://doi.org/10.1109/CCGrid51090.2021.00018

[56] James Dickson, Steven Wright, Satheesh Maheswaran, Andy Herdman, Mark C. Miller, and Stephen Jarvis. 2016. Replicating HPC I/O workloads with proxy applications. In *Proceedings of the 2016 1st Joint International Workshop on Parallel Data Storage and Data Intensive Scalable Computing Systems (PDSW-DISCS'16)*. 13–18. https://doi.org/10.1109/PDSW-DISCS.2016.007

[57] Bin Dong, Verónica Rodríguez Tribaldos, Xin Xing, Suren Byna, Jonathan Ajo-Franklin, and Kesheng Wu. 2020. DASSA: Parallel DAS data storage and analysis for subsurface event detection. In *Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS'20)*. 254–263. https://doi.org/10.1109/IPDPS47924.2020.00035

[58] Wenrui Dong, Guangming Liu, Jie Yu, and You Zuo. 2015. Using FTracer to characterize an I/O-intensive application. In *Proceedings of the 2015 4th International Conference on Computer Science and Network Technology (ICCSNT'15)*, Volume 01. 324–327. https://doi.org/10.1109/ICCSNT.2015.7490761

[59] Matthieu Dorier, Gabriel Antoniu, Franck Cappello, Marc Snir, Robert Sisneros, Orcun Yildiz, Shadi Ibrahim, Tom Peterka, and Leigh Orf. 2016. Damaris: Addressing performance variability in data management for post-petascale simulations. *ACM Transactions on Parallel Computing* 3, 3 (2016), 1–43.

[60] Matthieu Dorier, Gabriel Antoniu, Rob Ross, Dries Kimpe, and Shadi Ibrahim. 2014. CALCioM: Mitigating I/O interference in HPC systems through cross-application coordination. In *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*. 155–164. https://doi.org/10.1109/IPDPS.2014.27

[61] Matthieu Dorier, Shadi Ibrahim, Gabriel Antoniu, and Rob Ross. 2014. Omnisc'IO: A grammar-based approach to spatial and temporal I/O patterns prediction. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'14)*. IEEE, Los Alamitos, CA, 623–634. https://doi.org/10.1109/SC.2014.56

[62] Matthieu Dorier, Shadi Ibrahim, Gabriel Antoniu, and Rob Ross. 2016. Using formal grammars to predict I/O behaviors in HPC: The Omnisc'IO approach. *IEEE Transactions on Parallel and Distributed Systems* 27, 8 (2016), 2435–2449. https://doi.org/10.1109/TPDS.2015.2485980

[63] Nikoli Dryden, Roman Böhringer, Tal Ben-Nun, and Torsten Hoefler. 2021. Clairvoyant prefetching for distributed machine learning I/O. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'21)*. ACM, New York, NY, Article 92, 15 pages. https://doi.org/10.1145/3458817.3476181

[64] David Ellsworth, Bryan Green, and Patrick Moran. 2004. Interactive terascale particle visualization. In *Proceedings of the 2004 Conference on Visualization (VIS'04)*. IEEE, Los Alamitos, CA, 353–360. https://doi.org/10.1109/VISUAL.2004.55

[65] V. Etienne, E. Chaljub, J. Virieux, and N. Glinsky. 2010. An *hp*-adaptive discontinuous Galerkin finite-element method for 3-D elastic wave modelling. *Geophysical Journal International* 183, 2 (2010), 941–962. https://doi.org/10.1111/j.1365-246X.2010.04764.x

[66] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. 2011. An overview of the HDF5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases (AD'11)*. ACM, New York, NY, 36–47. https://doi.org/10.1145/1966895.1966900

[67] International Organization for Standardization. 1996. *Information Technology–Portable Operating System Interface (POSIX)*. IEEE, Los Alamitos, CA.

[68] Message Passing Interface Forum. 1994. *MPI: A Message-Passing Interface Standard—Version 4.0.* Technical Report. University of Tennessee, Knoxville, TN.

[69] Michael Frasca, Ramya Prabhakar, Padma Raghavan, and Mahmut Kandemir. 2011. Virtual I/O caching: Dynamic storage cache management for concurrent workloads. In *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'11)*. 1–11.

[70] Ana Gainaru, Guillaume Aupy, Anne Benoit, Franck Cappello, Yves Robert, and Marc Snir. 2015. Scheduling the I/O of HPC applications under congestion. In *Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium*. 1013–1022. https://doi.org/10.1109/IPDPS.2015.116

[71] Rong Ge, Xizhou Feng, Sindhu Subramanya, and Xian-He Sun. 2010. Characterizing energy efficiency of I/O intensive parallel applications on power-aware clusters. In *Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing, Workshops, and Phd Forum (IPDPSW'10)*. 1–8. https://doi.org/10.1109/IPDPSW.2010.5470904

[72] Rong Ge, Xizhou Feng, and Xian-He Sun. 2012. SERA-IO: Integrating energy consciousness into parallel I/O middleware. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGRID'12)*. IEEE, Los Alamitos, CA, 204–211. https://doi.org/10.1109/CCGrid.2012.39

[73] Markus Geimer, Felix Wolf, Brian J. N. Wylie, Erika Ábrahám, Daniel Becker, and Bernd Mohr. 2010. The Scalasca performance toolset architecture. *Concurrency and Computation: Practice & Experience* 22, 6 (April 2010), 702–719.

[74] Anjus George, Rick Mohr, James Simmons, and Sarp Oral. 2021. *Understanding Lustre Internals* (2nd Ed.). Oak Ridge National Laboratory, Oak Ridge, TN. https://doi.org/10.2172/1824954

[75] Pilar Gomez-Sanchez, Sandra Mendez, Dolores Rexachs, and Emilio Luque. 2017. A parallel I/O behavior model for HPC applications using serial I/O libraries. In *Proceedings of the 2017 International Conference on High Performance Computing and Simulation (HPCS'17)*. 244–251. https://doi.org/10.1109/HPCS.2017.45

[76] Junmin Gu, Scott Klasky, Norbert Podhorszki, Ji Qiang, and Kesheng Wu. 2018. Querying large scientific data sets with adaptable IO system ADIOS. In *Supercomputing Frontiers*, Rio Yokota and Weigang Wu (Eds.). Springer International Publishing, Cham, Switzerland, 51–69.

[77] Peng Gu, Jun Wang, and Robert Ross. 2008. Bridging the gap between parallel file systems and local file systems: A case study with PVFS. In *Proceedings of the 2008 37th International Conference on Parallel Processing*. 554–561. https://doi.org/10.1109/ICPP.2008.43

[78] Raghul Gunasekaran, Sarp Oral, Jason Hill, Ross Miller, Feiyi Wang, and Dustin Leverman. 2015. Comparative I/O workload characterization of two leadership class storage clusters. In *Proceedings of the 10th Parallel Data Storage Workshop (PDSW'15)*. ACM, New York, NY, 31–36. https://doi.org/10.1145/2834976.2834985

[79] Ioan Hadade, Timothy M. Jones, Feng Wang, and Luca di Mare. 2020. Software prefetching for unstructured mesh applications. *ACM Transactions on Parallel Computing* 7, 1 (March 2020), Article 3, 23 pages. https://doi.org/10.1145/3380932

[80] R. J. Hanisch, A. Farris, E. W. Greisen, W. D. Pence, B. M. Schlesinger, P. J. Teuben, R. W. Thompson, and A. Warnock. 2001. Definition of the Flexible Image Transport System (FITS)*. *Astronomy & Astrophysics* 376, 1 (2001), 359–380. https://doi.org/10.1051/0004-6361:20010923

[81] Jun He, John Bent, Aaron Torres, Gary Grider, Garth Gibson, Carlos Maltzahn, and Xian-He Sun. 2013. I/O acceleration with pattern detection. In *Proceedings of the 22nd International Symposium on High-Performance Parallel and Distributed Computing (HPDC'13)*. ACM, New York, NY, 25–36. https://doi.org/10.1145/2493123.2462909

[82] Jun He, Huaiming Song, Xian-He Sun, Yanlong Yin, and Rajeev Thakur. 2011. Pattern-aware file reorganization in MPI-IO. In *Proceedings of the 6th Workshop on Parallel Data Storage (PDSW'11)*. ACM, New York, NY, 43–48. https://doi.org/10.1145/2159352.2159363

[83] Shuibing He, Yan Liu, and Xian-He Sun. 2014. PSA: A performance and space-aware data layout scheme for hybrid parallel file systems. In *Proceedings of the 2014 International Workshop on Data Intensive Scalable Computing Systems*. 41–48. https://doi.org/10.1109/DISCS.2014.10

[84] Shuibing He, Yanlong Yin, Xian-He Sun, Xuechen Zhang, and Zongpeng Li. 2020. Optimizing parallel I/O accesses through pattern-directed and layout-aware replication. *IEEE Transactions on Computers* 69, 2 (2020), 212–225. https://doi.org/10.1109/TC.2019.2946135

[85] Weiping He, David H. C. Du, and Sai B. Narasimhamurthy. 2015. PIONEER: A solution to parallel I/O workload characterization and generation. In *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGRID'15)*. IEEE, Los Alamitos, CA, 111–120. https://doi.org/10.1109/CCGrid.2015.32

[86] Youbiao He, Dong Dai, and Forrest Sheng Bao. 2019. Modeling HPC storage performance using long short-term memory networks. In *Proceedings of the 2019 IEEE 21st International Conference on High Performance Computing and Communications, the IEEE 17th International Conference on Smart City, and the IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS'19)*. 1107–1114. https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00157

[87] Frank Herold and Sven Breuner. 2018. *An Introduction to BeeGFS*. Technical Report. ThinkParQ. https://www.beegfs.io/docs/whitepapers/Introduction_to_BeeGFS_by_ThinkParQ.pdf

[88] GitHub. 2022. HPC IO Benchmark Repository. Retrieved August 3, 2023 from github.com/hpc/ior

[89] Wei Hu, Guang-Ming Liu, Qiong Li, Yan-Huang Jiang, and Gui-Lin Cai. 2016. Storage wall for exascale supercomputing. *Frontiers of Information Technology and Electronic Engineering* 17, 11 (Nov. 2016), 1154–1175. https://doi.org/10.1631/FITEE.1601336

[90] Dachuan Huang, Xuechen Zhang, Wei Shi, Mai Zheng, Song Jiang, and Feng Qin. 2013. LiU: Hiding disk access latency for HPC applications with a new SSD-enabled data layout. In *Proceedings of the 2013 IEEE 21st International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*. 111–120. https://doi.org/10.1109/MASCOTS.2013.19

[91] Tsuyoshi Ichimura, Kohei Fujita, Seizo Tanaka, Muneo Hori, Maddegedara Lalith, Yoshihisa Shizawa, and Hiroshi Kobayashi. 2014. Physics-based urban earthquake simulation enhanced by 10.7 BlnDOF × 30 K time-step unstructured FE non-linear seismic wave simulation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'14)*. IEEE, Los Alamitos, CA, 15–26. https://doi.org/10.1109/SC.2014.7

[92] IEEE. 2013. *1003.1, 2013 Edition - Standard for Information Technology—Portable Operating System Interface (POSIX(TM)) Base Specifications, Issue 7* IEEE (incorporates IEEE Std 1003.1-2008 and IEEE Std 1003.1-2008/Cor 1-2013). IEEE, Los Alamitos, CA. https://doi.org/10.1109/IEEESTD.2013.6506091

[93] Sun Microsystems. 2007. *High-Performance Storage Architecture and Scalable Cluster File System*. Technical Report. Sun Microsystems.

[94] Mihailo Isakov, Eliakin del Rosario, Sandeep Madireddy, Prasanna Balaprakash, Philip Carns, Robert B. Ross, and Michel A. Kinsy. 2020. Toward generalizable models of I/O throughput. In *Proceedings of the 2020 IEEE/ACM International Workshop on Runtime and Operating Systems for Supercomputers (ROSS'20)*. 41–49. https://doi.org/10.1109/ROSS51935.2020.00010

[95] 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 (SC'20)*. IEEE, Los Alamitos, CA, 1–13. https://doi.org/10.1109/SC41405.2020.00037

[96] Tanzima Zerin Islam, Kathryn Mohror, Saurabh Bagchi, Adam Moody, Bronis R. de Supinski, and Rudolf Eigenmann. 2012. MCREngine: A scalable checkpointing system using data-aware aggregation and compression. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'12)*. 1–11. https://doi.org/10.1109/SC.2012.77

[97] ISO. 2018. *ISO/IEC 9899:2018 Information Technology—Programming Languages—C* (4th Ed.). ISO, Geneva, Switzerland. https://www.iso.org/standard/74528.html

[98] IBM Journal of Research and Development Staff. 2008. Overview of the IBM Blue Gene/P project. *IBM Journal of Research and Development* 52, 1-2 (Jan. 2008), 199–220.

[99] Myoungsoo Jung, Wonil Choi, Shekhar Srikantaiah, Joonhyuk Yoo, and Mahmut T. Kandemir. 2014. HIOS: A host interface I/O scheduler for solid state disks. *SIGARCH Computer Architecture News* 42, 3 (June 2014), 289–300. https://doi.org/10.1145/2678373.2665715

[100] Myoungsoo Jung and Mahmut Kandemir. 2013. Revisiting widely held SSD expectations and rethinking system-level implications. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'13)*. ACM, New York, NY, 203–216. https://doi.org/10.1145/2465529.2465548

[101] Donghe Kang, Oliver Rübel, Suren Byna, and Spyros Blanas. 2020. Predicting and comparing the performance of array management libraries. In *Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS'20)*. 906–915. https://doi.org/10.1109/IPDPS47924.2020.00097

[102] Qiao Kang, Scot Breitenfeld, Kaiyuan Hou, Wei-Keng Liao, Robert Ross, and Suren Byna. 2021. Optimizing performance of parallel I/O accesses to non-contiguous blocks in multiple array variables. In *Proceedings of the 2021 IEEE International Conference on Big Data (Big Data'21)*. 98–108. https://doi.org/10.1109/BigData52589.2021.9671638

[103] Harsh Khetawat, Christopher Zimmer, Frank Mueller, Scott Atchley, Sudharshan S. Vazhkudai, and Misbah Mubarak. 2019. Evaluating burst buffer placement in HPC systems. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'19)*. 1–11. https://doi.org/10.1109/CLUSTER.2019.8891051

[104] 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 (HPDC'20)*. ACM, New York, NY, 77–88. https://doi.org/10.1145/3369583.3392678

[105] Sunggon Kim, Alex Sim, Kesheng Wu, Suren Byna, Teng Wang, Yongseok Son, and Hyeonsang Eom. 2019. DCA-IO: A dynamic I/O control scheme for parallel and distributed file systems. In *Proceedings of the 2019 19th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGRID'19)*. IEEE, Los Alamitos, CA, 351–360. https://doi.org/10.1109/CCGRID.2019.00049

[106] Sunggon Kim, Alex Sim, Kesheng Wu, Suren Byna, Teng Wang, Yongseok Son, and Hyeonsang Eom. 2019. DCA-IO: A dynamic I/O control scheme for parallel and distributed file systems. In *Proceedings of the 2019 19th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGRID'19)*. 351–360. https://doi.org/10.1109/CCGRID.2019.00049

[107] Seong Jo Kim, Seung Woo Son, Wei-Keng Liao, Mahmut Kandemir, Rajeev Thakur, and Alok Choudhary. 2012. IOPin: Runtime profiling of parallel I/O in HPC systems. In *Proceedings of 2012 SC Companion: High Performance Computing, Networking Storage, and Analysis*. 18–23. https://doi.org/10.1109/SC.Companion.2012.14

[108] Andreas Knüpfer, Holger Brunst, Jens Doleschal, Matthias Jurenz, Matthias Lieber, Holger Mickler, Matthias S. Müller, and Wolfgang E. Nagel. 2008. The Vampir performance analysis tool-set. In *Tools for High Performance Computing*, Michael Resch, Rainer Keller, Valentin Himmler, Bettina Krammer, and Alexander Schulz (Eds.). Springer, Berlin, Germany, 139–155.

[109] Andreas Knüpfer, Christian Rössel, Dieter an Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen Malony, Wolfgang E. Nagel, Yury Oleynik, Peter Philippen, Pavel Saviankou, Dirk Schmidl, Sameer Shende, Ronny Tschüter, Michael Wagner, Bert Wesarg, and Felix Wolf. 2012. Score-P: A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir. In *Tools for High Performance Computing 2011*, Holger Brunst, Matthias S. Müller, Wolfgang E. Nagel, and Michael M. Resch (Eds.). Springer, Berlin, Germany, 79–91.

[110] Andy Konwinski, John Bent, James Nunez, and Meghan Quist. 2007. Towards an I/O tracing framework taxonomy. In *Proceedings of the 2nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing'07 (PDSW'07)*. ACM, New York, NY, 56–62. https://doi.org/10.1145/1374596.1374610

[111] Anthony Kougkas, Hariharan Devarajan, and Xian-He Sun. 2020. Bridging storage semantics using data labels and asynchronous I/O. *ACM Transactions on Storage* 16, 4 (Oct. 2020), Article 22, 34 pages. https://doi.org/10.1145/3415579

[112] Anthony Kougkas, Matthieu Dorier, Rob Latham, Rob Ross, and Xian-He Sun. 2016. Leveraging burst buffer coordination to prevent I/O interference. In *Proceedings of the 2016 IEEE 12th International Conference on e-Science (e-Science'16)*. 371–380. https://doi.org/10.1109/eScience.2016.7870922

[113] Sidharth Kumar, Avishek Saha, Venkatram Vishwanath, Philip Cards, John A. Schmidt, Giorgio Scorzelli, Hemanth Kolla, Ray Grout, Robert Latham, Robert Ross, Michael E. Papkafa, Jacqueline Chen, and Valerio Pascucci. 2013. Characterization and modeling of PIDX parallel I/O for performance optimization. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'13)*. ACM, New York, NY, 1–12. https://doi.org/10.1145/2503210.2503252

[114] Chih-Song Kuo, Aamer Shah, Akihiro Nomura, Satoshi Matsuoka, and Felix Wolf. 2014. How file access patterns influence interference among cluster applications. In *Proceedings of the 2014 IEEE International Conference on Cluster Computing (CLUSTER'14)*. IEEE, Los Alamitos, CA, 185–193. https://doi.org/10.1109/CLUSTER.2014.6968743

[115] Cristiano A. Künas, Matheus S. Serpa, Jean Luca Bez, Edson L. Padoin, and Philippe O. A. Navaux. 2021. Offloading the training of an I/O access pattern detector to the cloud. In *Proceedings of the 2021 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW'21)*. 15–19. https://doi.org/10.1109/SBAC-PADW53941.2021.00013

[116] Sriram Lakshminarasimhan, David A. Boyuka, Saurabh V. Pendse, Xiaocheng Zou, John Jenkins, Venkatram Vishwanath, Michael E. Papka, and Nagiza F. Samatova. 2013. Scalable in situ scientific data encoding for analytical query processing. In *Proceedings of the 22nd International Symposium on High-Performance Parallel and Distributed Computing (HPDC'13)*. ACM, New York, NY, 1–12. https://doi.org/10.1145/2493123.2465527

[117] Svetlana Lazareva and Ilia Demianenko. 2015. Automatic request analyzer for QoS enabled storage system. In *Proceedings of the 11th Central Eastern European Software Engineering Conference in Russia (CEE-SECR'15)*. ACM, New York, NY, Article 3, 8 pages. https://doi.org/10.1145/2855667.2855670

[118] Adrien Lebre, Guillaume Huard, Yves Denneulin, and Przemyslaw Sowa. 2006. I/O scheduling service for multi-application clusters. In *Proceedings of the 2006 IEEE International Conference on Cluster Computing*. IEEE, Los Alamitos, CA, 1–10. https://doi.org/10.1109/CLUSTR.2006.311854

[119] Choonghwan Lee, MuQun Yang, and Ruth Aydt. 2008. *NetCDF-4 Performance Report*. HDF Group (THG). https://www.hdfgroup.org/pubs/papers/2008-06_netcdf4_perf_report.pdf

[120] Alexander Lex, Nils Gehlenborg, Hendrik Strobelt, Romain Vuillemot, and Hanspeter Pfister. 2014. UpSet: Visualization of intersecting sets. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 1983–1992. https://doi.org/10.1109/TVCG.2014.2346248

[121] Jianwei Li, Wei-Keng Liao, Alok Choudhary, Robert Ross, Rajeev Thakur, William Gropp, Rob Latham, Andrew Siegel, Brad 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)*. ACM, New York, NY, 39. https://doi.org/10.1145/1048935.1050189

[122] Tonglin Li, Suren Byna, Houjun Tang, and Quincey Koziol. 2021. *H5bench: A Benchmark Suite for Parallel HDF5 (H5bench) V0.6*. Oak Ridge National Laboratory, Oak Ridge, TN. https://doi.org/10.11578/dc.20210624.4

[123] Weihao Liang, Yong Chen, and Hong An. 2019. Interference-aware I/O scheduling for data-intensive applications on hierarchical HPC storage systems. In *Proceedings of the 2019 IEEE 21st International Conference on High Performance Computing and Communications, the IEEE 17th International Conference on Smart City, and the IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS'19)*. IEEE, Los Alamitos, CA.

[124] Zhen Liang, Johann Lombardi, Mohamad Chaarawi, and Michael Hennecke. 2020. DAOS: A scale-out high performance storage stack for storage class memory. In *Supercomputing Frontiers*, Dhabaleswar K. Panda (Ed.). Springer International Publishing, Cham, Switzerland, 40–54.

[125] Jianwei Liao, Li Li, Huaidong Chen, and Xiaoyan Liu. 2015. Adaptive replica synchronization for distributed file systems. *IEEE Systems Journal* 9, 3 (2015), 865–877. https://doi.org/10.1109/JSYST.2014.2300611

[126] Jianwei Liao, François Trahay, and Guoqiang Xiao. 2016. Dynamic process migration based on block access patterns occurring in storage servers. *ACM Transactions on Architecture and Code Optimization* 13, 2 (June 2016), Article 20, 20 pages. https://doi.org/10.1145/2899002

[127] Wei-Keng Liao, Avery Ching, Kenin Coloma, Alok Choudhary, and Lee Ward. 2007. An implementation and evaluation of client-side file caching for MPI-IO. In *Proceedings of the 2007 IEEE International Parallel and Distributed Processing Symposium*. IEEE, Los Alamitos, CA, 1–10. https://doi.org/10.1109/IPDPS.2007.370239

[128] Wei-Keng Liao and Alok Choudhary. 2008. Dynamically adapting file domain partitioning methods for collective I/O based on underlying parallel file system locking protocols. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'08)*. IEEE, Los Alamitos, CA, Article 3, 12 pages.

[129] Wei-Keng Liao, Xaiohui Shen, and Alok Choudhary. 2000. Meta-data management system for high-performance large-scale scientific data access. In *High Performance Computing—HiPC 2000*, Mateo Valero, Viktor K. Prasanna, and Sriram Vajapeyam (Eds.). Springer, Berlin, Germany, 293–300.

[130] Don Liu, Yong-Lai Zheng, and Arden Moore. 2016. Three dimensional simulations of fluid flow and heat transfer with spectral element method. In *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale (XSEDE'16)*. ACM, New York, NY, Article 8, 9 pages. https://doi.org/10.1145/2949550.2949569

[131] Jialin Liu, Debbie Bard, Quincey Koziol, Stephen Bailey, and Prabhat. 2017. Searching for millions of objects in the BOSS spectroscopic survey data with H5Boss. In *Proceedings of the 2017 New York Scientific Data Summit (NYSDS'17)*. 1–9. https://doi.org/10.1109/NYSDS.2017.8085044

[132] Jialin Liu, Quincey Koziol, Gregory F. Butler, Neil Fortner, Mohamad Chaarawi, Houjun Tang, Suren Byna, Glenn K. Lockwood, Ravi Cheema, Kristy A. Kallback-Rose, Damian Hazen, and Prabhat. 2018. Evaluation of HPC application I/O on object storage systems. In *Proceedings of the IEEE/ACM 3rd International Workshop on Parallel Data Storage and Data Intensive Scalable Computing Systems*. 24–34. https://doi.org/10.1109/PDSW-DISCS.2018.00005

[133] Mingliang Liu, Ye Jin, Jidong Zhai, Yan Zhai, Qianqian Shi, Xiaosong Ma, and Wenguang Chen. 2013. ACIC: Automatic cloud I/O configurator for HPC applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'13)*. ACM, New York, NY, Article 38, 12 pages. https://doi.org/10.1145/2503210.2503216

[134] Qing Liu, Jeremy Logan, Yuan Tian, Hasan Abbasi, Norbert Podhorszki, Jong Youl Choi, Scott Klasky, Roselyne Tchouam, Lofsteadm Jay, Ron Oldfield, Manish Parashar, Nagiza Samatova, Karsten Schwan, Arie Shoshani, Matthew Wolf, Kesheng Wu, and Weikuan Yu. 2014. Hello ADIOS: The challenges and lessons of developing leadership class I/O frameworks. *Concurrency and Computation: Practice and Experience* 26, 7 (2014), 1453–1473. https://doi.org/10.1002/cpe.3125

[135] Wei Liu, Kai Wu, Jialin Liu, Feng Chen, and Dong Li. 2017. Performance evaluation and modeling of HPC I/O on non-volatile memory. In *Proceedings of the 2017 International Conference on Networking, Architecture, and Storage (NAS'17)*. 1–10. https://doi.org/10.1109/NAS.2017.8026869

[136] Weifeng Liu, Linping Wu, and Xiaowen Xu. 2020. Topology aware algorithm for two-phase I/O in clusters with tapered hierarchical networks. *IEEE Access* 8 (2020), 66917–66930. https://doi.org/10.1109/ACCESS.2020.2985928

[137] 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 (SC'16)*. 819–829. https://doi.org/10.1109/SC.2016.69

[138] Jay Lofstead, Milo Polte, Garth Gibson, Scott Klasky, Karsten Schwan, Ron Oldfield, Matthew Wolf, and Qing Liu. 2011. Six degrees of scientific data: Reading patterns for extreme scale science IO. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing (HPDC'11)*. ACM, New York, NY, 49–60. https://doi.org/10.1145/1996130.1996139

[139] 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 (CLADE'08)*. ACM, New York, NY, 15–24. https://doi.org/10.1145/1383529.1383533

[140] Javier López-Gómez and Jakob Blomer. 2021. Exploring object stores for high-energy physics data storage. *EPJ Web of Conferences* 251 (2021), 02066. https://doi.org/10.1051/epjconf/202125102066

[141] Yin Lu, Yong Chen, Rob Latham, and Yu Zhuang. 2014. Revealing applications' access pattern in collective I/O for cache management. In *Proceedings of the 28th ACM International Conference on Supercomputing (ICS'14)*. ACM, New York, NY, 181–190. https://doi.org/10.1145/2597652.2597686

[142] Xiaoqing Luo, Frank Mueller, Philip Carns, Jonathan Jenkins, Robert Latham, Robert Ross, and Shane Snyder. 2017. ScalaIOExtrap: Elastic I/O tracing and extrapolation. In *Proceedings of the 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS'17)*. IEEE, Los Alamitos, CA, 585–594. https://doi.org/10.1109/IPDPS.2017.45

[143] Huong Luu, Babak Behzad, Ruth Aydt, and Marianne Winslett. 2013. A multi-level approach for understanding I/O activity in HPC applications. In *Proceedings of the 2013 IEEE International Conference on Cluster Computing (CLUSTER'13)*. 1–5. https://doi.org/10.1109/CLUSTER.2013.6702690

[144] Huong Luu, Marianne Winslett, William Gropp, Robert Ross, Philip Carns, Kevin Harms, 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 (HPDC'15)*. ACM, New York, NY, 33–44. https://doi.org/10.1145/2749246.2749269

[145] Ricardo Macedo, Mariana Miranda, Yusuke Tanimura, Jason Haga, Amit Ruhela, Stephen Lien Harrell, Richard Todd Evans, and João Paulo. 2022. Protecting metadata servers from harm through application-level I/O control. In *Proceedings of the 2022 IEEE International Conference on Cluster Computing (CLUSTER'22)*. 573–580. https://doi.org/10.1109/CLUSTER51413.2022.00075

[146] Sandeep Madireddy, Prasanna Balaprakash, Philip Carns, Robert Latham, Robert Ross, Shane Snyder, and Stefan M. Wild. 2017. Analysis and correlation of application I/O performance and system-wide I/O activity. In *Proceedings of the 2017 International Conference on Networking, Architecture, and Storage (NAS'17)*. 1–10. https://doi.org/10.1109/NAS.2017.8026844

[147] Adam Manzanares, John Bent, Meghan Wingate, and Garth Gibson. 2012. The power and challenges of transformative I/O. In *Proceedings of the 2012 IEEE International Conference on Cluster Computing*. 144–154. https://doi.org/10.1109/CLUSTER.2012.86

[148] Sandra Mendez, Dolores Rexachs, and Emilio Luque. 2017. Analyzing the parallel I/O severity of MPI applications. In *Proceedings of the 2017 17th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGRID'17)*. IEEE, Los Alamitos, CA, 953–962. https://doi.org/10.1109/CCGRID.2017.45

[149] Mitesh R. Meswani, Pietro Cicotti, Jiahua He, and Allan Snavely. 2010. Predicting disk I/O time of HPC applications on flash drives. In *Proceedings of the 2010 IEEE GLOBECOM Workshops*. 1926–1929. https://doi.org/10.1109/GLOCOMW.2010.5700279

[150] Mitesh R. Meswani, Michael A. Laurenzano, Laura Carrington, and Allan Snavely. 2010. Modeling and predicting disk I/O time of HPC applications. In *Proceedings of the 2010 DoD High Performance Computing Modernization Program Users Group Conference*. 478–486. https://doi.org/10.1109/HPCMP-UGC.2010.27

[151] Mark C. Miller. 2022. MACsio—Lawrence Livermore National Laboratory. Retrieved August 3, 2023 from https://computing.llnl.gov/projects/co-design/macsio

[152] Christopher Mitchell, James Nunez, and Jun Wang. 2009. Overlapped checkpointing with hardware assist. In *Proceedings of the 2009 IEEE International Conference on Cluster Computing and Workshops*. 1–10. https://doi.org/10.1109/CLUSTR.2009.5289154

[153] Pavanakumar Mohanamuraly and Gabriel Staffelbach. 2020. Hardware locality-aware partitioning and dynamic load-balancing of unstructured meshes for large-scale scientific applications. In *Proceedings of the Platform for Advanced Scientific Computing Conference (PASC'20)*. ACM, New York, NY, Article 7, 10 pages. https://doi.org/10.1145/3394277.3401851

[154] Bernd Mohr, Darryl Brown, and Allen Malony. 1994. TAU: A portable parallel program analysis environment for pC++. In *Parallel Processing: CONPAR 94—VAPP VI*, Bruno Buchberger and Jens Volkert (Eds.). Springer, Berlin, Germany, 29–40.

[155] NASA Advanced Supercomputing Division. 2003. *NAS Parallel Benchmarks*. NASA. https://www.nas.nasa.gov/publications/npb.html

[156] Sarah Neuwirth, Feiyi Wang, Sarp Oral, and Ulrich Bruening. 2017. Automatic and transparent resource contention mitigation for improving large-scale parallel file system performance. In *Proceedings of the 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS'17)*. 604–613. https://doi.org/10.1109/ICPADS.2017.00084

[157] Bogdan Nicolae, Pierre Riteau, and Kate Keahey. 2014. Bursting the cloud data bubble: Towards transparent storage elasticity in IaaS clouds. In *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*. 135–144. https://doi.org/10.1109/IPDPS.2014.25

[158] Michael Noeth, Jaydeep Marathe, Frank Mueller, Martin Schulz, and Bronis de Supinski. 2006. Scalable compression and replay of communication traces in massively parallel environments. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC'06)*. ACM, New York, NY, 144–es. https://doi.org/10.1145/1188455.1188605

[159] Kazuki Ohta, Dries Kimpe, Jason Cope, Kamil Iskra, Robert Ross, and Yutaka Ishikawa. 2010. Optimization techniques at the I/O forwarding layer. In *Proceedings of the 2010 IEEE International Conference on Cluster Computing*. IEEE, Los Alamitos, CA, 312–321. https://doi.org/10.1109/CLUSTER.2010.36

[160] Sarp Oral, James Simmons, Jason Hill, Dustin Leverman, Feiyi Wang, Matt Ezell, Ross Miller, Douglas Fuller, Raghul Gunasekaran, Youngjae Kim, Saurabh Gupta, Devesh Tiwari, Sudharshan S. Vazhkudai, James H. Rogers, David Dillow, Galen M. Shipman, and Arthur S. Bland. 2014. Best practices and lessons learned from deploying and operating

large-scale data-centric parallel file systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'14)*. IEEE, Los Alamitos, CA, 217–228. https://doi.org/10.1109/SC.2014.23

[161] 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 (SC'19)*. ACM, New York, NY, Article 63, 14 pages. https://doi.org/10.1145/3295500.3356157

[162] Lu Pang, Anis Alazzawe, Krishna Kant, and Jeremy Swift. 2019. Data heat prediction in storage systems using behavior specific prediction models. In *Proceedings of the IEEE 38th International Performance Computing and Communications Conference (IPCCC'19)*. 1–8. https://doi.org/10.1109/IPCCC47392.2019.8958715

[163] 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 (SC'19)*. ACM, New York, NY, Article 65, 13 pages. https://doi.org/10.1145/3295500.3356183

[164] Swapnil Patil and Garth Gibson. 2011. Scale and concurrency of GIGA+: File system directories with millions of files. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST'11)*. 13.

[165] Arnab K. Paul, Olaf Faaland, Adam Moody, Elsa Gonsiorowski, Kathryn Mohror, and Ali R. Butt. 2020. Understanding HPC application I/O behavior using system level statistics. In *Proceedings of the 2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC'20)*. 202–211. https://doi.org/10.1109/HiPC50609.2020.00034

[166] Arnab K. Paul, Brian Wang, Nathan Rutman, Cory Spitz, and Ali R. Butt. 2020. Efficient metadata indexing for HPC storage systems. In *Proceedings of the 2020 20th IEEE/ACM International Symposium on Cluster, Cloud, and Internet Computing (CCGRID'20)*. 162–171. https://doi.org/10.1109/CCGrid49817.2020.00-77

[167] Pablo J. Pavan, Jean Luca Bez, Matheus S. Serpa, Francieli Zanon Boito, and Philippe O. A. Navaux. 2019. An unsupervised learning approach for I/O behavior characterization. In *Proceedings of the 2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'19)*. 33–40. https://doi.org/10.1109/SBAC-PAD.2019.00019

[168] Ramya Prabhakar, Mahmut Kandemir, and Myoungsoo Jung. 2013. Disk-cache and parallelism aware I/O scheduling to improve storage system performance. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. 357–368. https://doi.org/10.1109/IPDPS.2013.59

[169] Abhishek Rajimwale, Vijayan Prabhakaran, and John D. Davis. 2009. Block management in solid-state devices. In *Proceedings of the 2009 USENIX Annual Technical Conference (USENIX'09)*. 21.

[170] Dennis M. Ritchie and Ken Thompson. 1974. The UNIX time-sharing system. *Communications of the ACM* 17, 7 (July 1974), 365–375. https://doi.org/10.1145/361011.361061

[171] Rob Ross. 2002. MPI-Tile-IO Benchmark. Retrieved August 3, 2023 from https://www.mcs.anl.gov/research/projects/pio-benchmark/

[172] Philip C. Roth. 2007. Characterizing the I/O behavior of scientific applications on the Cray XT. In *Proceedings of the 2nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing'07 (PDSW'07)*. ACM, New York, NY, 50–55. https://doi.org/10.1145/1374596.1374609

[173] Salem El Sayed, Matthias Bolten, Dirk Pleiter, and Wolfgang Frings. 2016. Parallel I/O characterisation based on server-side performance counters. In *Proceedings of the 1st Joint International Workshop on Parallel Data Storage and Data Intensive Scalable Computing Systems (PDSW-DISCS'16)*. IEEE, Los Alamitos, CA, 7–12. https://doi.org/10.1109/PDSW-DISCS.2016.006

[174] Frank Schmuck and Roger Haskin. 2002. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST'02)*. 19–es.

[175] Saba Sehrish, Grant Mackey, Pengju Shang, Jun Wang, and John Bent. 2013. Supporting HPC analytics applications with access patterns using data restructuring and data-centric scheduling techniques in MapReduce. *IEEE Transactions on Parallel and Distributed Systems* 24, 1 (2013), 158–169. https://doi.org/10.1109/TPDS.2012.88

[176] Tim Shaffer and Douglas Thain. 2017. Taming metadata storms in parallel filesystems with MetaFS. In *Proceedings of the 2nd Joint International Workshop on Parallel Data Storage and Data Intensive Scalable Computing Systems (PDSW-DISCS'17)*. ACM, New York, NY, 25–30. https://doi.org/10.1145/3149393.3149401

[177] 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 (SC'08)*. 1–12. https://doi.org/10.1109/SC.2008.5222721

[178] Sameer Shende, Allen D. Malony, Wyatt Spear, and Karen Schuchardt. 2011. Characterizing I/O performance using the TAU performance system. In *Applications, Tools and Techniques on the Road to Exascale Computing*, Koen De Boss-

chere, Erik H. D'Hollander, Gerhard R. Joubert, David A. Padua, Frans J. Peters, and Mark Sawyer (Eds.) Advances in Parallel Computing, Vol. 22. IOS Press, Amsterdam, The Netherlands, 647–655. https://doi.org/10.3233/978-1-61499-041-3-647

[179] Xuanhua Shi, Ming Li, Wei Liu, Hai Jin, Chen Yu, and Yong Chen. 2017. SSDUP: A traffic-aware SSD burst buffer for HPC systems. In *Proceedings of the International Conference on Supercomputing (ICS'17)*. ACM, New York, NY, Article 27, 10 pages. https://doi.org/10.1145/3079079.3079087

[180] Xuanhua Shi, Wei Liu, Ligang He, Hai Jin, Ming Li, and Yong Chen. 2020. Optimizing the SSD burst buffer by traffic detection. *ACM Transactions on Architecture and Code Optimization* 17, 1 (March 2020), Article 8, 26 pages. https://doi.org/10.1145/3377705

[181] Arie Shoshani and Doron Rotem. 2009. *Scientific Data Management: Challenges, Technology, and Deployment*. Chapman & Hall/CRC, Boca Raton, FL. https://doi.org/10.1201/9781420069815

[182] 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 (ESPT'16)*. IEEE, Los Alamitos, CA, 9–17.

[183] Shane Snyder, Philip Carns, Robert Latham, Misbah Mubarak, Robert Ross, Christopher Carothers, Babak Behzad, Huong Vu Thanh Luu, Surendra Byna, and Prabhat. 2015. Techniques for modeling large-scale HPC I/O workloads. In *Proceedings of the 6th International Workshop on Performance Modeling, Benchmarking, and Simulation of High Performance Computing Systems (PMBS'15)*. ACM, New York, NY, Article 5, 11 pages. https://doi.org/10.1145/2832087.2832091

[184] Lizandro Solano-Quinde, Zhi Jian Wang, Brett Bode, and Arun K. Somani. 2011. Unstructured grid applications on GPU: Performance analysis and improvement. In *Proceedings of the 4th Workshop on General Purpose Processing on Graphics Processing Units (GPGPU'11)*. ACM, New York, NY, Article 13, 8 pages. https://doi.org/10.1145/1964179.1964197

[185] Huaiming Song, Yanlong Yin, Yong Chen, and Xian-He Sun. 2011. A cost-intelligent application-specific data layout scheme for parallel file systems. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing (HPDC'11)*. ACM, New York, NY, 37–48. https://doi.org/10.1145/1996130.1996138

[186] Huaiming Song, Yanlong Yin, Xian-He Sun, Rajeev Thakur, and Samuel Lang. 2011. A segment-level adaptive data layout scheme for improved load balance in parallel file systems. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. 414–423. https://doi.org/10.1109/CCGrid.2011.26

[187] Huaiming Song, Yanlong Yin, Xian-He Sun, Rajeev Thakur, and Samuel Lang. 2011. Server-side I/O coordination for parallel file systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'11)*. ACM, New York, NY, Article 17, 11 pages. https://doi.org/10.1145/2063384.2063407

[188] Jan Stender, Björn Kolbeck, Felix Hupfeld, Eugenio Cesario, Erich Focht, Matthias Hess, Jesús Malo, and Jonathan Martí. 2008. Striping without sacrifices: Maintaining POSIX semantics in a parallel file system. In *Proceedings of the 1st USENIX Workshop on Large-Scale Computing (LASCO'08)*. Article 6, 8 pages.

[189] Kohei Sugihara and Osamu Tatebe. 2020. Design of direct read from sparse segments in MPI-IO. In *Proceedings of the 2020 IEEE 22nd International Conference on High Performance Computing and Communications, the IEEE 18th International Conference on Smart City, and the IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS'20)*. 1308–1315. https://doi.org/10.1109/HPCC-SmartCity-DSS50907.2020.00168

[190] Stephen Sugiyama and David Wallace. 2008. *Cray DVS: Data Virtualization Service*. Cray.

[191] Hanul Sung, Jiwoo Bang, Alexander Sim, Kesheng Wu, and Hyeonsang Eom. 2019. Understanding parallel I/O performance trends under various HPC configurations. In *Proceedings of the ACM Workshop on Systems and Network Telemetry and Analytics (SNTA'19)*. ACM, New York, NY, 29–36. https://doi.org/10.1145/3322798.3329258

[192] Andrew S. Tanenbaum and Herbert Bos. 2014. *Modern Operating Systems* (4th Ed.). Prentice Hall.

[193] Houjun Tang, Suren Byna, Stephen Bailey, Zarija Lukic, Jialin Liu, Quincey Koziol, and Bin Dong. 2019. Tuning object-centric data management systems for large scale scientific applications. In *Proceedings of the 2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC'19)*. 103–112. https://doi.org/10.1109/HiPC.2019.00023

[194] Houjun Tang, Suren Byna, François Tessier, Teng Wang, Bin Dong, Jingqing Mu, Quincey Koziol, Jerome Soumagne, Venkatram Vishwanath, Jialin Liu, and Richard Warren. 2018. Toward scalable and asynchronous object-centric data management for HPC. In *Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid'18)*. IEEE, Los Alamitos, CA, 113–122. https://doi.org/10.1109/CCGRID.2018.00026

[195] Houjun Tang, Quincey Koziol, John Ravi, and Suren Byna. 2022. Transparent asynchronous parallel I/O using background threads. *IEEE Transactions on Parallel and Distributed Systems* 33, 4 (2022), 891–902. https://doi.org/10.1109/TPDS.2021.3090322

[196] François Tessier, Preeti Malakar, Venkatram Vishwanath, Emmanuel Jeannot, and Florin Isaila. 2016. Topology-aware data aggregation for intensive I/O on large-scale supercomputers. In *Proceedings of the 2016 1st International Workshop on Communication Optimizations in HPC (COMHPC'16)*. 73–81. https://doi.org/10.1109/COMHPC.2016.013

[197] Francois Tessier, Venkatram Vishwanath, and Emmanuel Jeannot. 2017. TAPIOCA: An I/O library for optimized topology-aware data aggregation on large-scale supercomputers. In *Proceedings of the 2017 IEEE International Conference on Cluster Computing (CLUSTER'17)*. IEEE, Los Alamitos, CA, 70–80. https://doi.org/10.1109/CLUSTER.2017.80

[198] Rajeev Thakur, William Gropp, and Ewing Lusk. 1996. An abstract-device interface for implementing portable parallel-I/O interfaces. In *Proceedings of the 6th Symposium on the Frontiers of Massively Parallel Computation (Frontiers'96)*. 180–187. https://doi.org/10.1109/FMPC.1996.558080

[199] Rajeev Thakur, William Gropp, and Ewing Lusk. 1999. Data sieving and collective I/O in ROMIO. In *Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation (Frontiers'99)*. 182–189. https://doi.org/10.1109/FMPC.1999.750599

[200] Rajeev Thakur, William Gropp, and Ewing Lusk. 2002. Optimizing noncontiguous accesses in MPI-IO. *Parallel Computing* 28, 1 (Jan. 2002), 83–105. https://doi.org/10.1016/S0167-8191(01)00129-6

[201] Rajeev Thakur, Ewing Lusk, and William Gropp. 1997. *Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation*. Technical Report. Argonne National Laboratory, Argonne, IL. https://doi.org/10.2172/564273

[202] The HDF Group. 2022. *Hierarchical Data Format, Version 5*. The HDF Group. https://www.hdfgroup.org/HDF5/

[203] Yuichi Tsujita, Atsushi Hori, Toyohisa Kameyama, Atsuya Uno, Fumiyoshi Shoji, and Yutaka Ishikawa. 2018. Improving collective MPI-IO using topology-aware stepwise data aggregation with I/O throttling. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia'18)*. ACM, New York, NY, 12–23. https://doi.org/10.1145/3149457.3149464

[204] Sudharshan S. Vazhkudai, Bronis R. de Supinski, Arthur S. Bland, Al Geist, James Sexton, Jim Kahle, Christopher J. Zimmer, Scott Atchley, Sarp Oral, Don E. Maxwell, Veronica G. Vergara Larrea, Adam Bertsch, Robin Goldstone, Wayne Joubert, Chris Chambreau, David Appelhans, Robert Blackmore, Ben Casses, George Chochia, Gene Davison, Matthew A. Ezell, Tom Gooding, Elsa Gonsiorowski, Leopold Grinberg, Bill Hanson, Bill Hartner, Ian Karlin, Matthew L. Leininger, Dustin Leverman, Chris Marroquin, Adam Moody, Martin Ohmacht, Ramesh Pankajakshan, Fernando Pizzano, James H. Rogers, Bryan Rosenburg, Drew Schmidt, Mallikarjun Shankar, Feiyi Wang, Py Watson, Bob Walkup, Lance D.Weems, and Junqi Yin. 2018. The design, deployment, and evaluation of the CORAL pre-exascale systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'18)*. IEEE, Los Alamitos, CA, Article 52, 12 pages. https://doi.org/10.1109/SC.2018.00055

[205] Marc-André Vef, Vasily Tarasov, Dean Hildebrand, and André Brinkmann. 2018. Challenges and solutions for tracing storage systems: A case study with spectrum scale. *ACM Transactions on Storage* 14, 2 (April 2018), Article 18, 24 pages. https://doi.org/10.1145/3149376

[206] M. Vilayannur, S. Lang, R. Ross, R. Klundt, and L. Ward. 2008. *Extending the POSIX I/O Interface: A Parallel File System Perspective*. Argonne National Laboratory, Argonne, IL. https://doi.org/10.2172/946036

[207] Venkatram Vishwanath. 2022. HACC I/O. Retrieved August 3, 2023 from https://asc.llnl.gov/coral-benchmarks#hacc

[208] Venkatram Vishwanath, Mark Hereld, Vitali Morozov, and Michael E. Papka. 2011. Topology-aware data movement and staging for I/O acceleration on Blue Gene/P supercomputing systems. In *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'11)*. ACM, New York, NY, Article 19, 11 pages. https://doi.org/10.1145/2063384.2063409

[209] Chen Wang, Kathryn Mohror, and Marc Snir. 2021. File system semantics requirements of HPC applications. In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing (HPDC'21)*. ACM, New York, NY, 19–30. https://doi.org/10.1145/3431379.3460637

[210] Chen Wang, Kathryn Mohror, and Marc Snir. 2021. File system semantics requirements of HPC applications. In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing (HPDC'21)*. ACM, New York, NY, 19–30. https://doi.org/10.1145/3431379.3460637

[211] Chen Wang, Jinghan Sun, Marc Snir, Kathryn Mohror, and Elsa Gonsiorowski. 2020. Recorder 2.0: Efficient parallel I/O tracing and analysis. In *Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW'20)*. IEEE, Los Alamitos, CA, 1–8. https://doi.org/10.1109/IPDPSW50202.2020.00176

[212] Lipeng Wang, Songgao Ye, Baichen Yang, Youyou Lu, Hequan Zhang, Shengen Yan, and Qiong Luo. 2020. DIESEL: A dataset-based distributed storage and caching system for large-scale deep learning training. In *Proceedings of the 49th International Conference on Parallel Processing (ICPP'20)*. ACM, New York, NY, Article 20, 11 pages. https://doi.org/10.1145/3404397.3404472

[213] Teng Wang, Suren Byna, Glenn K. Lockwood, Shane Snyder, Philip Carns, Sunggon Kim, and Nicholas J. Wright. 2019. A zoom-in analysis of I/O logs to detect root causes of I/O performance bottlenecks. In *Proceedings of the 2019 19th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGRID'19)*. 102–111. https://doi.org/10.1109/CCGRID.2019.00021

[214] Teng Wang, Sarp Oral, Yandong Wang, Brad Settlemyer, Scott Atchley, and Weikuan Yu. 2014. BurstMem: A high-performance burst buffer system for scientific applications. In *Proceedings of the 2014 IEEE International Conference on Big Data (Big Data'14)*. 71–79. https://doi.org/10.1109/BigData.2014.7004215

[215] Teng Wang, Kevin Vasko, Zhuo Liu, Hui Chen, and Weikuan Yu. 2014. BPAR: A bundle-based parallel aggregation framework for decoupled I/O execution. In *Proceedings of the 2014 International Workshop on Data Intensive Scalable Computing Systems*. 25–32. https://doi.org/10.1109/DISCS.2014.6

[216] Zhixiang Wang, Xuanhua Shi, Hai Jin, Song Wu, and Yong Chen. 2014. Iteration based collective I/O strategy for parallel I/O systems. In *Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGRID'14)*. IEEE, Los Alamitos, CA, 287–294. https://doi.org/10.1109/CCGrid.2014.61

[217] Donald C. Wells and Eric. W. Greisen. 1981. FITS: A flexible image transport system. *Astronomy & Astrophysics Supplement Series* 44 (1981), 363–370.

[218] Joseph P. White, Martins Innus, Matthew D. Jones, Robert L. DeLeon, Nikolay Simakov, Jeffrey T. Palmer, Steven M. Gallo, Thomas R. Furlani, Michael Showerman, Robert Brunner, Andriy Kot, Gregory Bauer, Brett Bode, Jeremy Enos, and William Kramer. 2017. Challenges of workload analysis on large HPC systems: A case study on NCSA blue waters. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success, and Impact (PEARC'17)*. ACM, New York, NY, Article 6, 8 pages. https://doi.org/10.1145/3093338.3093348

[219] Joseph P. White, Alexander D. Kofke, Robert L. DeLeon, Martins Innus, Matthew D. Jones, and Thomas R. Furlani. 2018. Automatic characterization of HPC job parallel filesystem I/O patterns. In *Proceedings of the Practice and Experience on Advanced Research Computing (PEARC'18)*. ACM, New York, NY, Article 3, 8 pages. https://doi.org/10.1145/3219104.3219121

[220] Hanpei Wu, Tongliang Deng, Yanliang Zou, Shu Yin, Si Chen, and Tao Xie. 2021. ADA: An application-conscious data acquirer for visual molecular dynamics. In *Proceedings of the 50th International Conference on Parallel Processing (ICPP'21)*. ACM, New York, NY, Article 61, 9 pages. https://doi.org/10.1145/3472456.3473509

[221] Huijun Wu, Liming Zhu, Kai Lu, Gen Li, and Dongyao Wu. 2016. StageFS: A parallel file system optimizing metadata performance for SSD based clusters. In *Proceedings of the 2016 IEEE Trustcom/BigDataSE/ISPA Conference*. 2147–2152. https://doi.org/10.1109/TrustCom.2016.0330

[222] Tzuhsien Wu, Jerry Chou, Shyng Hao, Bin Dong, Scott Klasky, and Kesheng Wu. 2017. Optimizing the query performance of block index through data analysis and I/O modeling. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'17)*. ACM, New York, NY, Article 12, 10 pages. https://doi.org/10.1145/3126908.3126934

[223] Cong Xu, Shane Snyder, Omkar Kulkarni, Vishwanath Venkatesan, Phillip Carns, Surendra Byna, Robert Sisneros, and Kalyana Chadalavada. 2019. DXT: Darshan eXtended Tracing. In *Proceedings of the 2019 Cray User Group Meeting*. 1–8. https://www.osti.gov/biblio/1490709

[224] Tianqi Xu, Kento Sato, and Satoshi Matsuoka. 2018. Explorations of data swapping on burst buffer. In *Proceedings of the 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS'18)*. 517–526. https://doi.org/10.1109/PADSW.2018.8644561

[225] Y. Xu and X. Chen. 2019. Numerical simulation of unstructured grids discontinuous galerkin finite element method for complex surface. In *Proceedings of the American Geophysical Union Fall Meeting*.

[226] Bin Yang, Wei Xue, Tianyu Zhang, Shichao Liu, Xiaosong Ma, Xiyang Wang, and Weiguo Liu. 2023. End-to-end I/O monitoring on leading supercomputers. *ACM Transactions on Storage* 19, 1 (Jan. 2023), Article 3, 35 pages. https://doi.org/10.1145/3568425

[227] Orcun Yildiz, Matthieu Dorier, Shadi Ibrahim, Rob Ross, and Gabriel Antoniu. 2016. On the root causes of cross-application I/O interference in HPC storage systems. In *Proceedings of the International Parallel and Distributed Processing Symposium*. IEEE, Los Alamitos, CA, 750–759. https://doi.org/10.1109/IPDPS.2016.50

[228] Orcun Yildiz, Amelie Chi Zhou, and Shadi Ibrahim. 2017. Eley: On the effectiveness of burst buffers for big data processing in HPC systems. In *Proceedings of the 2017 IEEE International Conference on Cluster Computing (CLUSTER'17)*. 87–91. https://doi.org/10.1109/CLUSTER.2017.73

[229] Orcun Yildiz, Amelie Chi Zhou, and Shadi Ibrahim. 2018. Improving the effectiveness of burst buffers for big data processing in HPC systems with Eley. *Future Generation Computer Systems* 86, C (Sept. 2018), 308–318. https://doi.org/10.1016/j.future.2018.03.029

[230] Yanlong Yin, Surendra Byna, Huaiming Song, Xian-He Sun, and Rajeev Thakur. 2012. Boosting application-specific parallel I/O optimization using IOSIG. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGRID'12)*. IEEE, Los Alamitos, CA, 196–203. https://doi.org/10.1109/CCGrid.2012.136

[231] Yanlong Yin, Jibing Li, Jun He, Xian-He Sun, and Rajeev Thakur. 2013. Pattern-direct and layout-aware replication scheme for parallel I/O systems. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing (IPDPS'13)*. IEEE, Los Alamitos, CA, 345–356. https://doi.org/10.1109/IPDPS.2013.114

[232] Jie Yu, Guangming Liu, Xiaoyong Li, Wenrui Dong, and Qiong Li. 2018. Cross-layer coordination in the I/O software stack of extreme-scale systems. *Concurrency and Computation: Practice and Experience* 30, 10 (2018), e4396. https://doi.org/10.1002/cpe.4396

[233] Jie Yu, Guangming Liu, Xin Liu, Wenrui Dong, Xiaoyong Li, and Yusheng Liu. 2018. Rethinking node allocation strategy for data-intensive applications in consideration of spatially bursty I/O. In *Proceedings of the 2018 International Conference on Supercomputing (ICS'18)*. ACM, New York, NY, 12–21. https://doi.org/10.1145/3205289.3205305

[234] Jie Yu, Wenxiang Yang, Fang Wang, Dezun Dong, Jinghua Feng, and Yuqi Li. 2020. Spatially bursty I/O on supercomputers: Causes, impacts and solutions. *IEEE Transactions on Parallel and Distributed Systems* 31, 12 (2020), 2908–2922. https://doi.org/10.1109/TPDS.2020.3005572

[235] Weikuan Yu, Jeffrey S. Vetter, and H. Sarp Oral. 2008. Performance characterization and optimization of parallel I/O on the Cray XT. In *Proceedings of the 2008 IEEE International Symposium on Parallel and Distributed Processing*. 1–11. https://doi.org/10.1109/IPDPS.2008.4536277

[236] Xuechen Zhang, Kei Davis, and Song Jiang. 2010. IOrchestrator: Improving the performance of multi-node I/O systems via inter-server coordination. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'10)*. IEEE, Los Alamitos, CA, 1–11. https://doi.org/10.1109/SC.2010.30

[237] Xuechen Zhang, Song Jiang, and Kei Davis. 2009. Making resonance a common case: A high-performance implementation of collective I/O on parallel file systems. In *Proceedings of the 2009 IEEE International Symposium on Parallel Distributed Processing*. 1–12. https://doi.org/10.1109/IPDPS.2009.5161070

[238] X. Zhang, K. Liu, K. Davis, and S. Jiang. 2013. iBridge: Improving unaligned parallel file access with solid-state drives. In *Proceedings of the International Parallel and Distributed Processing Symposium*. IEEE, Los Alamitos, CA, 381–392. https://doi.org/10.1109/IPDPS.2013.21

[239] Zhou Zhou, Xu Yang, Dongfang Zhao, Paul Rich, Wei Tang, Jia Wang, and Zhiling Lan. 2015. I/O-aware batch scheduling for petascale computing systems. In *Proceedings of the 2015 IEEE International Conference on Cluster Computing*. 254–263. https://doi.org/10.1109/CLUSTER.2015.45

[240] Yue Zhu, Fahim Chowdhury, Huansong Fu, Adam Moody, Kathryn Mohror, Kento Sato, and Weikuan Yu. 2018. Entropy-aware I/O pipelining for large-scale deep learning on HPC systems. In *Proceedings of the 2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'18)*. 145–156. https://doi.org/10.1109/MASCOTS.2018.00023

[241] Yue Zhu, Weikuan Yu, Bing Jiao, Kathryn Mohror, Adam Moody, and Fahim Chowdhury. 2019. Efficient user-level storage disaggregation for deep learning. In *Proceedings of the 2019 IEEE International Conference on Cluster Computing (CLUSTER'19)*. 1–12. https://doi.org/10.1109/CLUSTER.2019.8891023

[242] Christopher Zimmer, Saurabh Gupta, and Veronica G. Vergara Larrea. 2016. Finally, a way to measure frontend I/O performance. In *Proceedings of the 2016 Cray User Group Meeting*. 1–8.

Received 11 October 2022; revised 15 May 2023; accepted 11 July 2023