User-based I/O Profiling for Leadership Scale HPC Workloads

Ahmad Hossein Yazdani
Virginia Tech
Blacksburg, Virginia, United States
ahmadyazdani@vt.edu

Arnab K. Paul
BITS Pilani KK Birla
Goa, India
arnabp@goa.bits-pilani.ac.in

Ahmad Maroof Karimi
Oak Ridge National Laboratory
Oak Ridge, Tenesee, United States
karimiahmad@ornl.gov

Feiyi Wang
Oak Ridge National Laboratory
Oak Ridge, United States
fwang2@ornl.gov

Ali Butt
Virginia Tech
Blacksburg, United States
butta@cs.vt.edu

Abstract

I/O constitutes a significant portion of most of the application runtime. Spawning many such applications concurrently on an HPC system leads to severe I/O contention. Thus, understanding and subsequently reducing I/O contention induced by such multi-tenancy is critical for the efficient and reliable performance of the HPC system. In this study, we demonstrate that an application's performance is influenced by the command line arguments passed to the job submission. We model an application's I/O behavior based on two factors: past I/O behavior within a time window and user-configured I/O settings via command-line arguments. We conclude that I/O patterns for well-known HPC applications like `E3SM` and `LAMMP` are predictable, with an average uncertainty below 0.25 (A probability of 80%) and near zero (A probability of 100%) within a day. However, I/O pattern variance increases as the study time window lengthens. Additionally, we show that for 38 users and at least 50 applications constituting approximately 93000 job submissions, there is a high correlation between a submitted command line and the past command lines made within 1 to 10 days submitted by the user. We claim the length of this time window is unique per user.

CCS Concepts

• **Information systems → Distributed storage**.

Keywords

High Performance Computing, I/O characterization, I/O scheduler, Darshan, I/O profiling

ACM Reference Format:
Ahmad Hossein Yazdani, Arnab K. Paul, Ahmad Maroof Karimi, Feiyi Wang, and Ali Butt. 2025. User-based I/O Profiling for Leadership Scale HPC Workloads. In *26th International Conference on Distributed Computing and Networking (ICDCN 2025), January 04–07, 2025, Hyderabad, India.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3700838.3700865

1 Introduction

High Performance Computing (HPC) is essential for addressing complex and large-scale computational problems. These extensive computations generally require interaction with storage systems to handle substantial volumes of data. However, the interaction with storage systems often degrades the application throughput due to the extensive sharing among numerous users who collectively transfer multiple petabytes of data. Another factor for the increase in the I/O bottleneck is the faster evolution of computation technologies than the storage systems, resulting into much shorter time for computation [17, 27, 30, 36]. The notable decrease in I/O performance is observable even for single tasks repeated within a specified time frame [26, 40–43]. This issue worsens in large-scale applications using accelerators such as GPUs, TPUs, and DPUs, which provide computational speeds much higher than traditional CPUs [9, 10, 12, 13, 18, 25, 34]. In addition, the most recent technological advancements of supercomputers, such as Aurora [1] at Argonne National Laboratory, and most recently Frontier at Oak Ridge National Laboratory [35], enable the HPC applications to massively increase their computation and I/O in order to achieve more parallelism at the order of several millions, or billions of computations per second. This however leads to a significant I/O bandwidth contention on the file system. Unfortunately this cannot be resolved by simply scaling up the I/O resources especially in a multi-tenant HPC system where the infrastructure is shared by numerous users. This phenomenon is attributed to the intricate interconnections between computation and I/O nodes, coupled with the interference arising from concurrent workloads operating on the allocated resources [22]. To keep up with such an immense advancement in the computation capability of the HPC workloads, an extensive study of the I/O characteristics for these workloads is necessary with the hope that the I/O contention could be reduced by the insights instilled.

There have been several works characterizing the I/O behavior of HPC applications. Clustering I/O behavior on supercomputers like Mira or ALCF's Theta, using features extracted from the Darshan profiling tool, is an effort to characterize the application I/O pattern [7, 12, 20, 28]. Another method of achieving high I/O performance is to adjust the configuration of file systems based on previous runs of the applications [23]. Also, Paul *et al.* [31] present findings from the I/O behavior, exhibited by the machine learning jobs across a wide spectrum of science domains. They note only a few science domains on Summit leverage burst buffers, and these jobs usually produce a massive number of small reads

ICDCN 2025, January 04–07, 2025, Hyderabad, India

ACM ISBN 979-8-4007-1062-9/25/01
https://doi.org/10.1145/3700838.3700865

and writes. Costa *et al.* [15] take advantage of clustering to detect the I/O patterns the applications' job submissions on Blue Waters, which are made publicly available by NCSA. Bang *et al.* [8] also employ clustering for in-depth analysis of the applications' I/O behavior on Cori supercomputers in an unsupervised manner. Bez *et al.* [11] also characterize the I/O received by different layers of Summit and Cori, which are two multi-layer supercomputers. They provide more evidence of the imbalanced usage of different sublayers of these two systems, besides hinting at the increase in I/O done through STDIO. Nevertheless, it performs slower than some renowned I/O interfaces like POSIX and MPI-IO. Furthermore, Patel *et al.* [29] demonstrate how file access patterns are reused among submissions of the same application. They also show the patterns that are usually absorbed by the files that each application touches. It also demonstrates variability in time from one run to another. Besides, Wyatt *et al.* [39] extracts the key features of the image representations for the job scripts. Using the aforementioned features, they can predict the I/O runtime of the future submissions. However, the prediction accuracy fluctuates between 60% and 90%. This large fluctuation in accuracy can be attributed to variation in applications' I/O performance across different runs.

The primary objective of this paper is to model the I/O behavior of applications using historical user configuration data over a specified period, a topic not thoroughly explored in prior research. Users significantly influence the I/O patterns of applications. For instance, one user may allocate 10 nodes, while another might over-provision with 100 nodes, affecting both the application's I/O behavior and performance. Furthermore, there may be a link between consecutive submissions, as users often base the configuration of subsequent jobs on the outputs of previous ones, impacting I/O performance.

In this paper, we examine the I/O behavior of HPC applications and pinpoint user behavior as a source of variance across different runs. Our analysis reveals that the I/O behavior of these applications is influenced by user-specified parameters. We initially create a dataset from Darshan logs for the full year of 2020 from the leadership scale *Summit* supercomputer, which is among the fastest supercomputers in the world [37]. From the raw Darshan data, we create a *fine-grained job-level I/O profile* for every job successfully executed on the HPC system.

We cluster job executions by user and application based on their I/O behavior. These clusters represent I/O patterns for the jobs. We quantitatively link the I/O patterns of application instances to commands submitted within a week. Our study indicates that the I/O pattern of an application, even with the same execution command, depends on the submission timeframe and past submissions within the last 1 to 10 days. In addition, our study reveals that these temporal relationships are unique to users. This implies the past activities of the users and the way they configure the application influences the execution of an application. We confirm that this observation holds for most users running well-known HPC applications like `LAMMPS` [24] and `E3SM` [16].

In summary, we make the following contributions.

(1) ***We make an I/O profile for each job submission of a user***, by extracting and engineering the Darshan features the corresponding job produces.

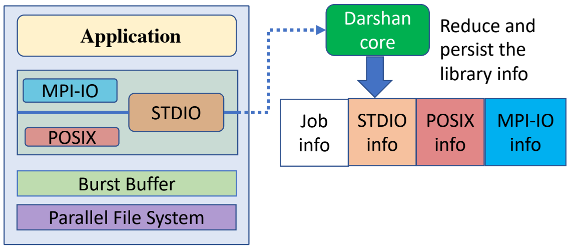


Figure 1: The interaction between the I/O libraries and Darshan logs comprising each library I/O information.

(2) ***We group the job submissions on Summit, in a user and application-based manner***, which helps us discover the latent patterns of any type in the underlying job submissions, and the important features shaping the I/O behavior of that application.

(3) ***We show the correlation between the application's I/O pattern and the history of the activities of the user when submitting the same application.*** across 38 users submitting 50 applications, which constitute 93000 jobs. We use the conventional statistical methodologies to attribute the I/O patterns of a job execution to 1) The command line options provided by the executor (user), and 2) The history of the submitted command line options for the same executor and application within a certain time window. *We argue this time-window is unique to the executor of the application, implying each user has to be studied differently.*

(4) Henceforth, ***we conclude the application's behavior is not solely determined by the application or the surrounding runtime***, but the users' historical attitude towards configuring these applications.

2 Background

This section provides a brief overview of Darshan I/O characterization tool, which helps generate the logs used in our work. We also briefly describe *Summit*- the HPC system, whose logs and user I/O activities are analyzed.

2.1 Darshan - I/O Characterization Tool

Darshan [12] is a lightweight profiling tool which is capable of capturing a variety of I/O-related measures from the I/O stack starting from the time the run begins until the application is shut down. Figure 1 illustrates the software I/O stack and Darshan. It records job information such as job ID, executable, start, and end times. For each I/O interface (e.g. MPI-IO, POSIX, STDIO), library-specific data is logged, including the frequency and average time of operations (open, read, write, close). Key features include the total I/O for each interface, the total time, and the number of processes. Each Darshan log captures about thirty job-related features and over 200 features for each access to the file [2].

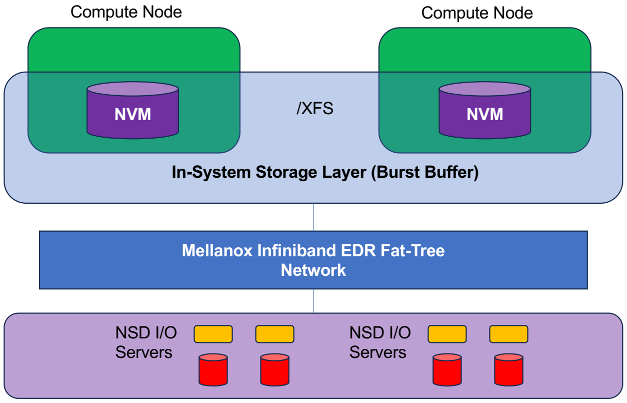

Compute Node
Compute Node
NVM
/XFS
NVM
In-System Storage Layer (Burst Buffer)
Mellanox Infiniband EDR Fat-Tree Network
NSD I/O Servers
NSD I/O Servers

Figure 2: Architecture of Parallel File System in *Summit*, comprising of an in-system storage layer (Or Burst-Buffer), and *Alpine*, a 250 PB IBM Spectrum Scale (GPFS). In *Summit*, this layer is called GPFS.

2.2 *Summit*

Summit- one of the fastest supercomputers in the world [37], was manufactured by IBM, and deployed at *Oak Ridge National Laboratory (ORNL)*, whose performance is about 200 PF. Figure 2 depicts the architecture of *Summit*. The supercomputer contains 4068 AC922 compute nodes, each having two IBM POWER9 (P9) processors and NVIDIA Tesla V100 (Volta) GPUs. Moreover, each node features 512 GB DDR4 memory, and an NVLink 2.0 bus interconnects each P9 CPU to 3 v100 GPUs. An InfiniBand EDR network with a fat-tree topology connects the nodes. Each compute node has an installed 1.6 TB NVMe device, the so-called burst buffer, providing an aggregate of 7.4 PB raw capacity, with 26.7 TB/s and 9.7 TB/s as the maximum read and write bandwidths, respectively. In addition, *Summit* is connected to *Alpine*, a 250 PB IBM Spectrum Scale (GPFS) file system. This file system is equipped with 154 GPFS Network Shared Disk (NSD) servers for maintaining file data in parallel. *Alpine* peak bandwidth is estimated to be 2.5 TB/s in aggregate under a large sequential write I / O access pattern. *Alpine* is a central-wide file system and can be directly accessed by all other *Oak Ridge National Laboratory (ORNL)*'s resources.

3 Related Work

Several prior works have studied profiling the applications' I/O patterns at different levels of storage system. This characterization identifies the interaction of I/O layers and the parameters influencing application and system performance. It also reveals opportunities to modify applications or enhance system I/O utilities to improve HPC I/O efficiency. The prior works in this area usually fall into two categories: 1) *Application-level* profiling 2) *System-level* profiling. The goal of profiling at the *application level* is to explain the relationship between the I/O performance of an application and the high-level I/O statistics collected using I/O characterization tools such as Darshan. On the other hand, *system level* profiling is usually agnostic to applications.

3.1 Application level I/O profiling

Using the information from I/O tracing tools like Darshan, which provides a fine-grained summary of the I/O activities, the workloads types can be approximately characterized. There have been several prior research works on application profiling using Darshan. Bang *et al.* [7] take a 4-month real-world trace from the Cori system in NERSC and select a set of features using a novel feature selection method. It picks the Darshan features which are correlated the most with the amount of write throughput. The results show that there are 3 natural clusters to which all the jobs belong. Pavan *et al.* [32] use unsupervised learning to characterize I/O applications by parsing Darshan logs into a set of I/O phases that have only one access pattern. It uses four months Darshan logs from Interpid at Argonne National Laboratory. The work demonstrates that most of the accesses are through POSIX and small requests, and most of the patterns access unique files. Ng *et al.* [28] train a classifier on a set of applications, which are selected using features in Darshan logs submitted on Mira from April 2013 to October 2015. The work tries to predict the write throughput of the Lustre file system.

Other prior works leverage the information provided by tracing the lower-level API calls, or the runtime environment profilers. They seek to optimize the I/O, using the knowledge they have from the history of runs of the same application, or the applications running with similar properties. Kim *et al.* [23] adjust the underlying storage configuration based on previous runs of a similar application. There has also been prior work on gathering information from different libraries like MPI-IO, and POSIX [14] by injecting a code into the application runtime. In addition, Tang *et al.* [36] develop a benchmark capable of profiling a job submitted to Summit, with a varied configuration over time across the I/O sublayers, and then propose SCTuner, which exploits the insights gained from the profiler and dynamically tunes the HDF5 hyperparameters very efficiently.

3.2 System level I/O profiling

In this approach, a feature set of the system-level information is selected. Examples of such features include the average write latency to the underlying storage system within a particular time period, or the number of users and processes running on the system, and the peak I/O throughput. Wan *et al.* [38] collect information from the system-level traces, and extract features which are agnostic of any particular application. Using certain system-specific features, it predicts the amount of I/O throughput. The work yields 75% accuracy when applied. Both application and system level profiles are considered in [5], which uses knowledge from the workload domain to be able to map jobs into the system resources so that the storage system is efficiently utilized. The same route is adopted by [4] which determines the best parameters for an application. Particularly, they consider a combination of MPI-IO level and Lustre level features on top of Cori supercomputers and develop a Bayesian model to predict a performance score like bandwidth. In addition, Patel *et al.* bridge the gap between the accessibility of the HPC community and the I/O behavior of various workloads on top of such a system. They extract some low-level features like the number of cores, the amount of wall clock time, the amount of wait time for a job, etc. using which they showcase some temporal patterns including the burstiness of the submissions at certain points during the week. In addition, they predict the running time of the jobs submitted.

Most of the literature and the aforementioned studies fail to deliver satisfactory performance on certain workloads. This limitation arises because prior jobs of the application's submitter and their role in shaping job's I/O behavior are not considered in their evaluations. We establish the imperative of adopting an alternative approach to studying the I/O runtime of diverse workloads through user-based profiling. Profiling user behavior at the application level, utilizing finer-grained features extracted from Darshan logs, facilitates a comprehensive understanding of variations in the I/O profiles of an application. Our investigation elucidates a correlation between users' historical activities and their subsequent submissions, demonstrating a pronounced predictability in the application I/O behavior when considering the antecedent job configurations of the submitter.

4 Profiling I/O behavior of Users

In the preceding section, we emphasized the importance of recognizing the role played by users in shaping the I/O patterns of HPC applications, as well as understanding the latent factors within an application that can enhance the I/O performance of both individual applications and the overall system. To address this, we construct an I/O profile for each application's job submission by employing engineering techniques and extracting features from the corresponding Darshan log. This profile serves as a representation of the I/O behavior exhibited during the application's individual submission. Utilizing the compiled profiles, we demonstrate the potential for making inferences based on the historical behavior of the users, which we have observed across a significant number of samples, and their respective submissions In the meantime, we endeavor to explain the means by which the users may contribute to the behavior difference across the submissions of an application. The remainder of this section fully describes our methodology for attaining the mentioned objectives, along with the process of extracting and crafting the necessary features essential for carrying out these analyses.

4.0.1 ***Building I/O Profiles.*** For the development of Input/Output (I/O) profiles, we leverage counters extracted from Darshan logs. Each application execution on the Summit supercomputer is delineated as a job, consisting of tasks instantiated by 'jsrun' commands. Every 'jsrun' invocation produces a corresponding Darshan log file. These logs encapsulate various metrics such as the volume of bytes read and written, cumulative I/O time, and metadata operations associated with POSIX, MPI-IO, and STDIO [2]. Summit employs IBM's Cluster System Management (CSM) as its batch job scheduler [3]. Darshan logs encompass these metrics for every file accessed throughout the application's execution. Additionally, the logs facilitate the determination of the number of processes interacting with a single file, which provides valuable insights into I/O contention. Furthermore, the metadata included within a Darshan log, such as the executable name utilized by 'jsruns', user identification, and the start and end times, allows for the amalgamation of logs from multiple 'jsrun' instances. This metadata is instrumental in feature engineering and extraction, culminating in the construction of job-level I/O profiles.

A subset of counters is selected from the provided Darshan logs, which encompass approximately 200 counters. Additionally, a distinct set of features is synthesized from the Darshan logs. The amalgamation of both extracted and engineered features constitutes an I/O profile for each job, as delineated in Table 1. Feature engineering is deemed indispensable, as Darshan logs encompass an extensive array of statistics, with a majority of features exhibiting sparsity in most instances. This inherent sparsity introduces a potential threat to the fidelity of the machine learning models designated for analyzing the submissions. It is imperative that the organized feature sets semantically preserve the critical I/O characteristics of the job submissions. The ensemble of derived features collectively forms a comprehensive profile for each job. The derived job I/O profiles enhance job clustering accuracy by reflecting the predominant types of access performed by applications. Each profile records features related to I/O accesses for each interface, the file system, and the type of access (see Table 1), effectively representing the job's I/O activity over time.

As discussed, the features we report in Table 1 are calculated for each I/O interface (POSIX, MPI-IO, STDIO), the file system receiving the I/O operations (GPFS or Burst Buffer), the file access methods (FPP, PSF or SSF), and the transfer size category (KB, MB, GB) as described below.

- **I/O interfaces (POSIX, MPI-IO, STDIO)**. It is noteworthy that MPI-IO routines utilize POSIX to perform their I/O. This is why part of the POSIX I/O may have resulted from calls made through MPI-IO. To further realize the amount of contention coming from MPI-IO calls, we separate the I/Os coming from MPI-IO from the ones arising from the job's direct calls to POSIX API. Such discrimination enables us to recognize how MPI-IO high-level calls are absorbed by the layers underneath the I/O stack. As a result, this distinction aids the analysts to have an estimation of the job's I/O access pattern. Also, studying MPI-IO metrics allows us to understand what types of high-level I/O calls the applications make, and how lower layers transform the queries into more efficient ones. Also, as per what is presented in [11], in some of the science domains, the majority of I/Os are performed via STDIO, which makes them a treasurable source for learning about the users in the system, and accordingly to leverage this knowledge to increase the I/O performance.
- **File system requests (burst buffers (NVMe) vs GPFS)**. Previous studies show that the usage of burst buffers is not common across the users in HPC systems, mainly because users do not know about the speedup they would achieve by directing their I/Os to burst buffers. Therefore, the classification of features in such terms yields invaluable information to us from the overall contention on the GPFS, and how we can balance the contention across different layers of storage.
- **Files access method (File access type)**. We name a file accessed by all the processes as *single shared files (SSF)*. If more than one (not all) processes access the file but not all processes access that file, we call it *part shared file (PSF)*, and *file per process (FPP)* if only one process accesses the file. Prior works demonstrate that most of the applications favor independent I/Os rather than collective accesses as

Features	***total number of files***, ***number of write-intensive files***, ***number of read-intensive files*** ***number of read-write files***, ***number of metadata files***, total bytes read/written, total number of read/write calls, total number of metadata operations
Interface (Library)	POSIX (Not from MPI-IO), POSIX (From MPI-IO), MPI-IO, STDIO
File System	parallel file system (GPFS for *Summit*), burst buffer (node-local NVMe for *Summit*)
File Access Type	Single-shared file (SSF), Part-shared file (PSF), File-per-process (FPP)
File Transfer Size Category	KB (1 KB - 999 KB), MB (1 MB - 999 MB), GB (1 GB and above)

Table 1: Features derived from a Darshan log. Each feature is reported for every I/O interface, file system and file type. The features in bold represent the number of files of a certain category, which is determined by Algorithm 1.

stated above, lying in the fact that a lot of users in the HPC community are mostly not familiar with the primitives different libraries provide. The contention may emerge from the fact that the underlying parallel file system fails to transform independent I/O calls into sequential collective I/O calls. Such grouping provides us with more information about the slowdowns due to the users mostly doing independent I/O operations.

- **File Transfer Size Category**. This categorization is essential as it can determine the mechanics of potential underlying low-level I/O operations. To this end, we consider 3 categories to discretize the transfer size: 1) KB, which corresponds to the I/O operations transferring 1 KB or less 2) MB for transfers ranging from 1 MB to 999 MB), GB for 1 GB transfers and above.

Algorithm 1: Categorization of files as write-intensive, read-intensive, metadata or read-write.

```
Input: Total bytes written into: $bytes\_written$, Total bytes read from: $bytes\_read$
Output: Category of the file as being write-intensive, read-intensive, read-write or
        metadata
begin
    if $bytes\_read + bytes\_written == 0$ then // No I/O is done; Label it as
      metadata
        return METADATA
    $ratio$ = (bytes_read - bytes_written) / (bytes_read + bytes_written)
    if $ratio >= -1$ && $ratio <= -0.5$ then
        return WRITE_INTENSIVE
    else if $ratio >= 0.5$ && $ratio <= 1$ then
        return READ_INTENSIVE
    else
        return READ_WRITE
```

To generate the profiles, we first aggregate all Darshan logs of the jsruns within a single job to derive a job-wise summary of the I/O features for each job submission of an application. We then need to aggregate the values of the features for different runs of an application by a single user on a particular scale. Thus, we get one file with the features introduced in Table 1. Algorithm 2 explains how we obtain the user I/O profile for a single application for a specific scale. In line 7, we derive the features listed in table 1 for the given jsrun. The next step is to aggregate these vectors. The aggregation involves the summation of the features described in table 1. We name this aggregate vector as the job-level submission I/O profile. Next, we aggregate all of these job-level I/O profiles into a single coarse-grained I/O profile by averaging all the job-level profiles. We suppose that the coarse-grained I/O profile could represent the average behavior of the user when submitting an application at a scale.

Algorithm 2: Generating user I/O profile for one application for a specific scale.

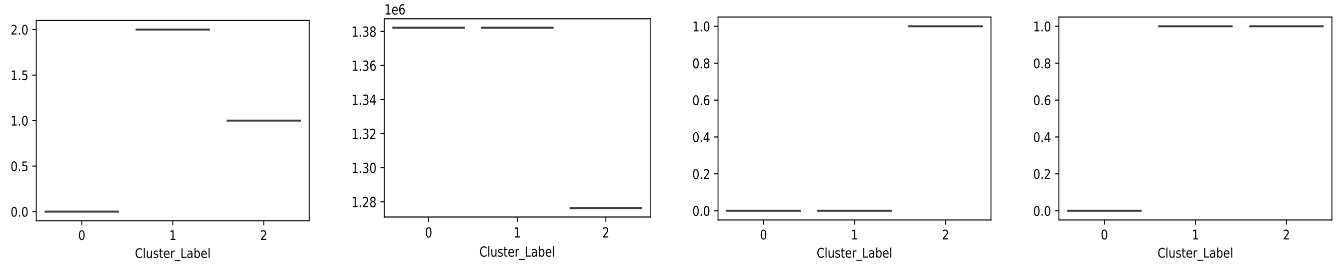
```
   Output: profiles generated for every user for each application and scale
1  begin
2      for app in uid_apps do
3          for scale in uid_apps do
4              set jobs_features as empty list
5              for job in app do
6                  for jsrun in job do
7                      //Extract the features in table 1 from the Darshan statistics
8                      jsrun_feature_vector = calculate_features(job_statistics)
9                      add jsrun_feature_vector to jsrun_feature_vectors
10                 //Compile the features vectors for each jsrun into an individual
                     job-level I/O profile
11                 joblevel_profile = aggregate(jsrun_feature_vectors)
```

5 Analysis

In this section, we leverage the profiles produced for the collection of executions we maintain to model the I/O activity of an application as a function of the configurable knobs for the application, and the user's attitude towards tweaking the settings. To this end, we need to quantitatively distinguish the job submissions whose I/O patterns contrast and to identify the causes of jobs differentiated in an interpretable way. To this end, we first leverage an unsupervised learning mechanism to group the submissions by the I/O patterns, and then in the next subsection, we associate the configurable knobs of the applications with the discovered I/O patterns, and devise a modeling strategy to find out the I/O pattern of an application under submission.

5.1 Categorizing submissions by I/O pattern

We cluster the job submissions in our dataset in an unsupervised manner so that the jobs with distinct I/O characteristics are separated from each other, and to have a representation of the I/O patterns the jobs yield. The procedure for this categorization is detailed in Algorithm 3, where we group the submissions by the submitter (*uid*) and the executed application (*app*), and apply the clustering algorithm for each group. This step helps us reduce the the random variation of the data, which might have otherwise confuse the clustering algorithm. Following this, we standardize the features of the corresponding profile collection using equation 1, scaling them down to attain a mean of 0 and a standard deviation of 1. Subsequently, we eliminate features with a predetermined



(a) Number of write intensive files receiving MB transfers under STDIO with file per process access.

(b) Number of GB-sized write transfers under STDIO with file per process access.

(c) Number of write intensive files receiving GB transfers under STDIO with file per process access.

(d) Number of KB-sized read transfers under STDIO with Single-shared file access.

Figure 3: The contrast between the distinctive features of user A clusters. The x axis represents the cluster label, while the y axis shows the violin plot values for each feature.

fraction of missing values (80% for all user and application pairs) or with a zero interquartile range (IQR) distance.

$$x_{standard} = \frac{x - \mu}{\sigma} \quad (1)$$

Algorithm 3: Job clustering grouped by user and application.

```
Input: profiles generated for each job run: $job\_run\_profiles$
Output: The job profiles labeled with their corresponding I/O cluster for each user and
        application
begin
    set $labeled\_job\_profiles$ to empty map
    for $uid, app$ in $uid\_app\_pairs$ do
        $uid\_app\_profiles = job\_run\_profiles[(uid, app)]$
        // Standardize features using equation 1
        $std\_profiles = standardize\_features(scale\_profiles)$
        // Remove the features which are zero at least 80% of the times, or with
           Inter-Quartile Range (IQR) distance of zero
        $std\_profiles = discard\_zero\_features(std\_profiles, 0.8)$
        // Runs HDBSCAN algorithm
        $labeled\_profiles = run\_hdbscan(std\_profiles, best\_k)$
        $labeled\_job\_profiles[(uid, app)] = labeled\_profiles$
    return $labeled\_job\_profiles$
```

Next, we perform clustering of job profiles for the designated user and application. We posit that an application, when executed under a particular user, manifests multiple patterns only if it exceeds 10 submissions. This threshold is established to prevent the clustering algorithm from aggregating all data points into a singular group. For this purpose, we employ the HDBSCAN algorithm [33]. Analogous to DBSCAN [19, 21], this method is an agglomerative, density-based, nonparametric clustering algorithm that functions by iteratively aggregating samples into clusters. However, DBSCAN consolidates data points into a single cluster if they reside within a manually tuned distance ϵ, rendering it suboptimal for data sets with varying density. In such scenarios, a large ϵ causes most data points to be assigned to a single cluster, whereas a small ϵ yields numerous small clusters, potentially fragmenting data points that could belong to the same cluster. HDBSCAN addresses this issue by incorporating a hierarchical clustering framework, wherein a comprehensive grid-search is performed across multiple ϵ values. Consequently, the determination of a secondary parameter, k, which delineates the minimum sample size for each cluster, becomes imperative. For our analysis, we set k to 2, based on the premise that at least two executions per user should form a cluster. Furthermore, empirical evidence suggests that the setting $k = 2$ produces a comparable partitioning compared to the higher values of k. Figure 4 elucidates the distribution of pairwise distances between clusters derived for various values of k. Evidently, elevated values of k exhibit approximately equivalent efficacy in differentiating clusters, thereby rendering this methodology less effective for certain datasets with larger k values. Generally, the minimum sample parameters can be optimally adjusted per user and application, contingent on the nature of the tasks and the users' historical I/O data. This adaptive clustering per user and application offers the distinct advantage of reduced computational time, enabling system schedulers to explore a broader spectrum of hyperparameters. Additionally, separating clustering by user and application improves the model's ability to accurately capture I/O patterns and distinguish between users, even when they run the same application.

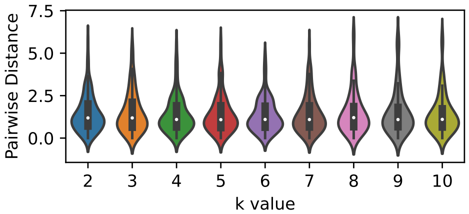


Figure 4: The distribution of the pairwise distances between the clusters for varying k values, where k is the min sample hyperparameter in HDBSCAN algorithm.

Our clustering methodology proves highly effective, yielding well-separated clusters for many user and application pairs. Figure 3 illustrates this for user A, who submitted jobs running `lmp_mpi` on August 20th and from mid-November to mid-December 2020. This binary represents the *LAMMPS* application [24], used to simulate molecular groups and provide chemists with insights into material dynamics.

Cluster 1 submissions tend towards write transfers measured in Megabytes, whereas Cluster 2 characterizes jobs that perform write transfers on the order of Gigabytes, particularly under the file-per-process access paradigm. Additionally, Clusters 1 and 2 generate single-shared read transfers at the Kilobyte scale, which Cluster 0 does not exhibit. Each cluster is represented by a range of values (or an exact value) per feature. For instance, Cluster 0 always features zero write intensive files with file per process at megabyte scale (figure 3a), a huge volume of gigabyte-sized write transfers (figure 3b), zero GB-sized write intensive files (figure 3c), and zero KB-sized single-shared read transfers (figure 3d). In contrast, Cluster 1 maximizes KB-sized single-shared read transfers (figure 3d), while Cluster 2 has zero GB transfers and a large number of GB-sized write intensive files (figures 3b and 3c), suggesting a more spread I/O across smaller files compared to Cluster 0.

5.2 User impact on the application's I/O behavior

In this subsection, we quantitatively correlate the behavior of an application with the past activities of the user when executing an application. As mentioned before, the users can alter the behavior of the application via the command line parameters they pass, and by adjustment of the application's surrounding runtime, e.g via directing the I/O traffic to the Burst Buffer rather than the slower file systems like GPFS or Lustre. In fact, this section finds an answer to the following questions:

(1) *Which parameters or settings can a user adjust to modify the I/O behavior of an application, and to what extent can these settings accurately predict the I/O behavior of a specific application execution?*
(2) *How much contribution do the settings of the past submissions of the same application have towards the I/O behavior of the next execution for the same user?*

Based on Darshan's input, the only adjustable parameter users have left to fine-tune would be the selection of command line options and the binary name submitted for each job. In our analysis, we focus on the 38 users with the maximum number of submissions within the system so that we are confident that our findings can be generalized to the whole system. Doing so leaves us with around 93000 job submissions. Out of this number of jobs, we focus on applications popular amongst the HPC community like LAMMPS and E3SM for which we have the highest number of records.

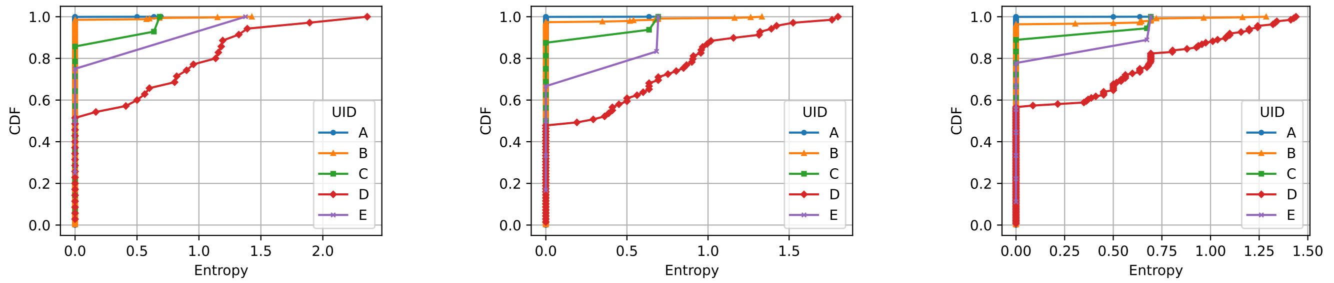
To measure the impact of the submitted command lines on the application's I/O pattern, we extract features from the command line submitted for each execution. To this end, we get the options and switches changing from one execution to another for each user and application, convert this data into nominal features, and quantify the predictability of the I/O patterns for an execution given the command line options submitted for that execution, as well as the past I/O patterns for executions of the same application under the same executor.

We then measure the ease of predicting the resultant pattern of an application when a set of command line arguments are submitted. To this end, we calculate the entropy of the I/O patterns grouped by the submitted command lines. In fact, entropy represents the uncertainty in the estimation of the value of a random variable. Henceforth, the lesser the value of the entropy, the lesser the multiplicity of the values for the random variable. The entropy is defined as follows for the random variable X:

$$entropy(X) = p_i * logp_i \tag{2}$$

Assuming X takes any of the values $x_1, x_2, \ldots, x_n$, we set p_i to be the probability that X becomes x_i. The entropy tends to zero in case there is a skewness towards a single value. In our case, a value close to zero suggests a single command line always leads to the same pattern. For the rest of this section, we plot the entropy variation within different time scales (In the overall dataset, weekly and daily).

LAMMPS is a classical molecular dynamics (MD) code that models ensembles of particles in a liquid, solid, or gaseous state. Its versatility extends to modeling atomic, polymeric, biological, solid-state (including metals, ceramics, and oxides), granular, coarse-grained, or macroscopic systems, employing diverse inter-atomic potentials (force fields) and boundary conditions. It can model 2d or 3d systems with sizes ranging from only a few particles up to billions [24], making it fit for HPC scale. Figures 5a to 5c illustrate the distribution of the entropy of the I/O patterns (clusters) for users who executed the highest number of commands per unique command line option in the entire dataset, within a week, and within a day respectively. According to these figures, different users executing the same application with the same options over time may observe different I/O patterns, while another set of users submitting the same options may see the same I/O pattern repeated. This implies the need for profiling each user different due to the unique nature of the temporal I/O patterns they may exhibit. This shows the I/O pattern of the LAMMP application is heavily reliant on the command line options submitted. For users like A, we find the LAMMP script they specify, alongside the presence of a unique set of simulation variables controlling the molecular simulation determining the I/O pattern, while for the other users, the command line only specifies the log files where the output is dumped, making it hard to explain the actual reason of variation. Moreover, Figure 6 illustrates the distribution of uncertainty in predicting the I/O patterns for the top 50 applications in the dataset, as submitted by a single user. High-performance computing applications like `lalibe` and `e3sm` show minimal variance in I/O patterns (near-zero daily entropy), indicating their predictability. However, `LAMMPS` demonstrates consistent behavior when the binaries `lmp_summit` and `lmp_g++` are used by users C and D, respectively, but exhibits increased uncertainty with `lmp_mpi` submitted by user E. These findings suggest that different users may require individualized monitoring. Figure 7 mirrors figure 6, examining executions within a day. These visualizations suggest that shorter analysis periods increase certainty in predicting execution I/O patterns, often resulting in singular patterns across more command lines. In the aforementioned plots (Figures 6 and 7), there are some binaries and applications for which the I/O pattern varies per command line. For a significant number of such applications, the command line takes no parameter, meaning the user intervention in the I/O behavior of the application cannot be explained using the command line. It is likely that such users



(a) Overall Entropy Distribution. **(b) Weekly Entropy Distribution.** **(c) Daily Entropy Distribution.**

Figure 5: The distribution of the entropy of LAMMP's I/O pattern when submitted with the same command line arguments for different users in the overall dataset, every week, and every day. The user IDs are anonymized.

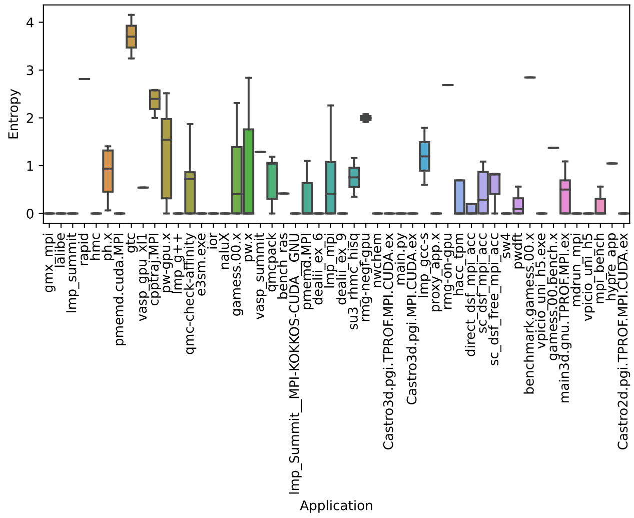


Figure 6: The average uncertainty of the application's I/O patterns when submitted by a user in the overall dataset. In this case, we treat the binaries with different namings which refer to the same application differently.

would most likely read files and settings which are hard-coded in the application code.

Next, we aim to quantify the interaction between prior and subsequent job submissions using the same binary. We measure the mutual information gain between past command-line arguments and the I/O pattern of the following job. To achieve this, we gather command lines associated with the identical binary as the job in question. For example, if the succeeding job employs `lmp_summit` as a binary, we exclusively consider submissions that execute `lmp_summit` for that user. .

Assuming X and Y are two random variables, the mutual information gain between the two, denoted by $I(X;Y)$ is defined as follows [6]:

$$I(X;Y) = H(X) - H(Y|X) \tag{3}$$

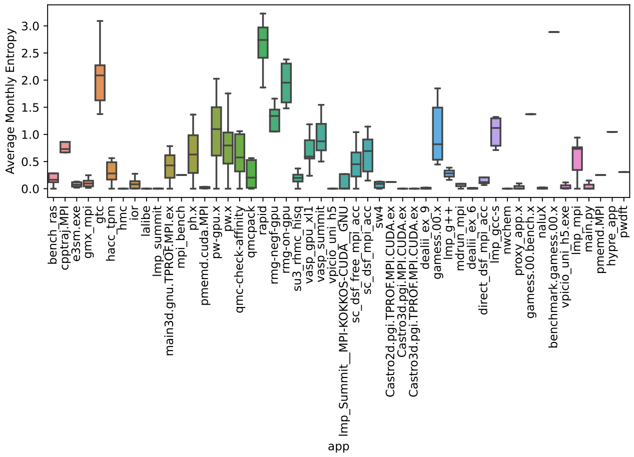


Figure 7: The average uncertainty of the application's I/O patterns when submitted by a user in within a day.

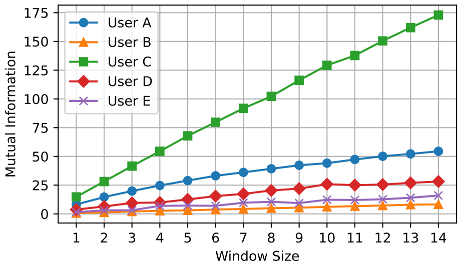


Figure 8: Mutual information between a command's label and the options of the command and the preceding commands, as determined by varying window sizes (1 to 4), for the top 5 users with the highest number of executions in the `LAMMP` application. The user IDs are anonymous.

Mutual information measures the reduction in uncertainty about one variable given knowledge of another. In our study, Y denotes the set of command-line options within a specific window (the number of preceding commands), while X represents the I/O pattern of the subsequent job using the same application. To measure the correlation between an application's I/O pattern and past command submissions, we take the union of all command line options across different submissions for a given binary. Missing options are filled with a default value, ensuring that each command line includes the complete set of options for consistent comparison. Figure 8 depicts the relationship between a command's label and its options, considering both current and preceding commands, for the LAMMP application among the top 5 users running the same binary with different options. Mutual information increases with window size, underscoring the importance of command line history. The varying slopes indicate unique user patterns, consistent with other binaries like Lalibe and previous figures, particularly for window sizes under 10, corresponding to 1 to 10 days of commands. Notably, the slope decreases for 4 users, suggesting that beyond a certain point, preceding commands may become irrelevant to the current job, a pattern also observed in other applications studied.

6 Discussion and Future Work

We discussed that the I/O patterns of HPC applications are highly influenced by historical configurations submitted by the same user for the same application. The analysis conducted supports this point. Our study shows that an application's I/O behavior can be modeled based on past I/O activity within a specific time frame and user-configured I/O settings via command line arguments. The findings indicate that I/O patterns for well-known HPC applications, such as E3SM and LAMMP, are predictable with low uncertainty within a short time frame but become more variable over longer periods. Furthermore, our results reveal a strong correlation between current and past command-line submissions made by users within 1 to 10 days, with the optimal time window being unique to each user.

Analyzing command-line options aids in storage system scheduling by revealing potential I/O behavior. For instance, reading from a .csv file versus a .hdf5 file can significantly impact file system access, influencing our modeling and predictions.

However, there are still a lot of aspects in terms of I/O characterization to be investigated. For example, many sample jobs lack command line options, containing only the executable name, which limits our analysis by treating all such submissions identically. However, executables, whether binary or high-level scripts, often include debugging symbols and source code. Schedulers can leverage Large Language Models (LLMs) to extract features from the source code and map them to configuration knobs, thereby enabling I/O pattern modeling.

7 Conclusion

I/O is a major part of the applications within HPC. A concurrent execution of these applications leads to a significant bottleneck and contention. Consequently, there is a huge need to identify the underlying causes of such issues, one of which is the nature of the applications themselves. This aspect of the I/O runtime is greatly studied by the previous state-of-the-arts. However, they have failed to provide accurate models of the I/O behavior. In particular, the behavior of applications can be influenced by various means and user-configurable parameters. An example includes the input they pass to their applications, which is produced by their previous submissions. In this work we quantitatively attribute the variation of the I/O performance to the user parameters set in a way specific to the users each. We cluster the executions of an application grouped by the users in an unsupervised manner. We then correlate the application execution's resultant I/O pattern (cluster) as a function of the history of the user activities when executing the same application. We employ conventional statistical and ML approaches to associate the I/O pattern of an application execution with the executor (user) of the application. Our study shows the resultant I/O pattern of an execution under a user is highly dependent on the past submitted command line options of the same application under the same user within a 1 to 10 day time window, which is unique to the submitter. This suggests that I/O schedulers have the potential to leverage knowledge about users' activities in the system, opening possibilities for optimization.

Acknowledgments

We thank our anonymous reviewers for their detailed feedback and valuable suggestions. This work is sponsored in part by the NSF under the grants: CSR-2106634, CCF-1919113/1919075, CNS-2045680, OAC-2004751, and OAC-2106446, Office of Science of the U.S. Department of Energy under the grant DE-AC05-00OR22725, SERB, Govt. of India Start-up Research grant SRG/2023/002445, BITS CRDF under grant C1/23/173 and BITS Pilani under the grants: BBF/BITS(G)/FY2022-23/BCPS-123/24-25/R1 and GOA/ACG/2022-2023/Oct/11. Results presented in this paper were obtained using the OLCF at Oak Ridge National Laboratory.

References

[1] [n. d.]. *Aurora Exascale Supercomputer.* https://www.anl.gov/aurora

[2] 2021. https://www.mcs.anl.gov/research/projects/darshan/docs/darshan-util.html

[3] 2021. HPC IBM-CSM. https://www.ibm.com/docs/en/spectrum-lsf/10.1.0

[4] Megha Agarwal, Divyansh Singhvi, Preeti Malakar, and Suren Byna. 2019. Active Learning-based Automatic Tuning and Prediction of Parallel I/O Performance. In *2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW)*. 20–29. https://doi.org/10.1109/PDSW49588.2019.00007

[5] Anthony Agelastos, Benjamin Allan, Jim Brandt, Ann Gentile, Sophia Lefantzi, Steve Monk, Jeff Ogden, Mahesh Rajan, and Joel Stevenson. 2015. Toward Rapid Understanding of Production HPC Applications and Systems. In *2015 IEEE International Conference on Cluster Computing*. 464–473. https://doi.org/10.1109/CLUSTER.2015.71

[6] A. Al-Ani and M. Deriche. 2002. Feature selection using a mutual information based measure. In *2002 International Conference on Pattern Recognition*, Vol. 4. 82–85 vol.4. https://doi.org/10.1109/ICPR.2002.1047405

[7] 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*. 33–40.

[8] Jiwoo Bang, Chungyong Kim, Kesheng Wu, Alex Sim, Suren Byna, Hanul Sung, and Hyeonsang Eom. 2021. An In-Depth I/O Pattern Analysis in HPC Systems. In *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. 400–405. https://doi.org/10.1109/HiPC53243.2021.00056

[9] Keith Bateman, Stephen Herbein, Anthony Kougkas, and Xian-He Sun. 2022. State of I/O in HPC 2020. (2022).

[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* (Austin, Texas) *(PDSW '15)*. Association for Computing Machinery, New York, NY, USA, 43–48. https://doi.org/10.1145/2834976.2834977

[11] 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*. 43–55.

[12] Philip Carns, Robert Latham, Robert Ross, Kamil Iskra, Samuel Lang, and Katherine Riley. 2009. 24/7 characterization of petascale I/O workloads. In *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, 1–10.

[13] Jesus Carretero, Emmanuel Jeannot, Guillaume Pallez, David E Singh, and Nicolas Vidal. 2020. Mapping and scheduling HPC applications for optimizing I/O. In *Proceedings of the 34th ACM International Conference on Supercomputing*. 1–12.

[14] Yan-Tyng Sherry Chang, Henry Jin, and John Bauer. 2016. Methodology and Application of HPC: I/O Characterization with MPIProf and IOT. In *2016 5th Workshop on Extreme-Scale Programming Tools (ESPT)*. 1–8. https://doi.org/10.1109/ESPT.2016.005

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

[16] DOE. 2021. Energy Exascale Earth System Model (E3SM). https://github.com/E3SM-Project/E3SM.

[17] John L Hennessy and David A Patterson. 2011. *Computer architecture: a quantitative approach*. Elsevier.

[18] Stephen Herbein, Dong H Ahn, Don Lipari, Thomas RW Scogland, Marc Stearman, Mark Grondona, Jim Garlick, Becky Springmeyer, and Michela Taufer. 2016. Scalable I/O-aware job scheduling for burst buffer enabled HPC clusters. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*. 69–80.

[19] 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 *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–13.

[20] 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 *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–13. https://doi.org/10.1109/SC41405.2020.00037

[21] Kamran Khan, Saif Ur Rehman, Kamran Aziz, Simon Fong, and Sababady Sarasvady. 2014. DBSCAN: Past, present and future. In *The fifth international conference on the applications of digital information and web technologies (ICADIWT 2014)*. IEEE, 232–238.

[22] 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*. 77–88.

[23] 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 *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 351–360.

[24] LAMMPS Development Team. 2023. *LAMMPS Documentation*. https://docs.lammps.org/Intro_overview.html Accessed on: January 22nd 2024.

[25] Glenn K Lockwood, Shane Snyder, Teng Wang, Suren Byna, Philip Carns, and Nicholas J Wright. 2018. A year in the life of a parallel file system. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 931–943.

[26] Jay Lofstead, Fang Zheng, Qing Liu, Scott Klasky, Ron Oldfield, Todd Kordenbrock, Karsten Schwan, and Matthew Wolf. 2010. Managing variability in the IO performance of petascale storage systems. In *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–12.

[27] Sarah Neuwirth and Arnab K Paul. 2021. Parallel I/O Evaluation Techniques and Emerging HPC Workloads: A Perspective. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 671–679.

[28] Hong Wei Ng. 2018. Machine learning for selecting parallel I/O benchmark applications. (2018).

[29] Tirthak Patel and Suren Byna. 2020. Uncovering Access, Reuse, and Sharing Characteristics of I/O-Intensive Files on Large-Scale Production HPC Systems.. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies, 2020*.

[30] 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 *2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. 202–211. https://doi.org/10.1109/HiPC50609.2020.00034

[31] Arnab K Paul, Ahmad Maroof Karimi, and Feiyi Wang. 2021. Characterizing Machine Learning I/O Workloads on Leadership Scale HPC Systems. In *2021 29th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 1–8.

[32] Pablo J Pavan, Jean Luca Bez, Matheus S Serpa, Francieli Zanon Boito, and Philippe OA Navaux. 2019. An unsupervised learning approach for I/O behavior characterization. In *2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 33–40.

[33] Md Farhadur Rahman, Weimo Liu, Saad Bin Suhaim, Saravanan Thirumuruganathan, Nan Zhang, and Gautam Das. 2016. Hdbscan: Density based clustering over location based services. *arXiv preprint arXiv:1602.03730* (2016).

[34] Robert B Ross, Rajeev Thakur, et al. 2000. PVFS: A parallel file system for Linux clusters. In *Proceedings of the 4th annual Linux showcase and conference*. 391–430.

[35] David Schneider. 2022. The Exascale Era is Upon Us: The Frontier supercomputer may be the first to reach 1,000,000,000,000,000,000 operations per second. *IEEE spectrum* 59, 1 (2022), 34–35.

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

[37] TOP 500 Organization. 2022. TOP 500 Supercomputer Sites. https://www.top500.org/lists/top500/2022/11/.

[38] Lipeng Wan, Matthew Wolf, Feiyi Wang, Jong Youl Choi, George Ostrouchov, Jieyang Chen, Norbert Podhorszki, Jeremy Logan, Kshitij Mehta, Scott Klasky, et al. 2020. I/O Performance Characterization and Prediction through Machine Learning on HPC Systems. (2020).

[39] Michael R Wyatt, Stephen Herbein, Todd Gamblin, Adam Moody, Dong H Ahn, and Michela Taufer. 2018. Prionn: Predicting runtime and io using neural networks. In *Proceedings of the 47th International Conference on Parallel Processing*. 1–12.

[40] Bing Xie, Jeffrey Chase, David Dillow, Oleg Drokin, Scott Klasky, Sarp Oral, and Norbert Podhorszki. 2012. Characterizing output bottlenecks in a supercomputer. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–11.

[41] Bing Xie, Yezhou Huang, Jeffrey S Chase, Jong Youl Choi, Scott Klasky, Jay Lofstead, and Sarp Oral. 2017. Predicting output performance of a petascale supercomputer. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. 181–192.

[42] Bing Xie, Sarp Oral, Christopher Zimmer, Jong Youl Choi, David Dillow, Scott Klasky, Jay Lofstead, Norbert Podhorszki, and Jeffrey S Chase. 2020. Characterizing output bottlenecks of a production supercomputer: Analysis and implications. *ACM Transactions on Storage (TOS)* 15, 4 (2020), 1–39.

[43] Bing Xie, Zilong Tan, Philip Carns, Jeff Chase, Kevin Harms, Jay Lofstead, Sarp Oral, Sudharshan S Vazhkudai, and Feiyi Wang. 2019. Applying machine learning to understand write performance of large-scale parallel filesystems. In *2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW)*. IEEE, 30–39.