A Deep Look Into the Temporal I/O Behavior of HPC Applications

Francieli Boito*, Luan Teylo*, Mihail Popov*, Theo Jolivel†, François Tessier†, Jakob Luettgau†, Julien Monniot†, Ahmad Tarraf ‡, André Carneiro§, and Carla Osthoff §

*Univ. Bordeaux, CNRS, Bordeaux INP, Inria, LaBRI, UMR 5800, F-33400, Talence, France

†Univ Rennes, Inria, CNRS, IRISA, Rennes, France

‡Department of Computer Science, Technical University of Darmstadt, Darmstadt, Germany

§LNCC, Petrópolis - RJ, 25651-075, Brazil

{first_name}.{last_name}@inria.fr, ahmad.tarraf@tu-darmstadt.de, {andrerc,osthoff}@lncc.br

***Abstract*—The increasing gap between compute and I/O speeds in high-performance computing (HPC) systems imposes the need for techniques to improve applications' I/O performance. Such techniques must rely on assumptions about I/O behavior in order to efficiently allocate I/O resources such as burst buffers, to schedule accesses to the shared parallel file system or to delay certain applications at the batch scheduler level to prevent contention, for instance. In this paper, we verify these common assumptions about I/O behavior, specifically about temporal behavior, using over 440,000 traces from real HPC systems. By combining traces from diverse systems, we characterize the behaviors observed in real HPC workloads. Among other findings, we show that I/O activity tends to last for a few seconds, and that periodic jobs are the minority, but responsible for a large portion of the I/O time. Furthermore, we make projections for the expected improvement yielded by popular approaches for I/O performance improvement. Our work provides valuable insights to everyone working to alleviate the I/O bottleneck in HPC.**

***Index Terms*—high-performance computing, parallel file systems, workload characterization, temporal I/O behavior**

I. Introduction

In high-performance computing (HPC) platforms, applications, usually submitted as batch jobs, rely on a parallel file system (PFS) to access persistent data. The PFS is deployed over a set of dedicated resources, separate from compute resources, which are shared by all concurrent jobs. Over the years, the gap between computing performance and the ability of storage systems to store and retrieve data has widened considerably: while supercomputers have become increasingly efficient for computing tasks, storage systems have become relatively less efficient [51]. At the same time, new, non-traditional, HPC applications such as data analytics and machine learning have shifted the workload toward more data intensity [32]. Therefore, I/O has been an important bottleneck for performance and efficient system usage at scale.

Many techniques have been proposed to improve I/O performance, often focusing on mitigating contention in concurrent accesses. Common examples are the allocation of resources such as burst buffers or I/O nodes [4], [33], [34], [48], [63], [67], [69], [73], [75], [78], scheduling concurrent accesses to the shared PFS [6], [13], [26], [40], [44], adapting applications (by changing when and how much data they write, for example) according to current load [62], [64], [67], [72] or I/O-aware batch scheduling [11], [35], [49], [71]. With rare exceptions, these techniques rely on some assumptions about the workload characteristics. As a result, information about applications' temporal I/O behaviors is an important building block to allow for smart decision-making for I/O optimization.

In this paper, we verify such assumptions by extensively studying the temporal I/O behavior of HPC applications over a decade. While many efforts have described other I/O characteristics such as request size and used I/O API [9], [53], [61], [66], [77], the temporal behavior has been the focus of less work. Moreover, assumptions about it are often based on experience (e.g., periodic writes can be due to checkpointing, which is something applications often need [22], [67]) and on knowledge from specific applications or domains [21], [23], [29], [79]. By studying large sets of traces from real HPC systems, we aim at characterizing the workload we can expect in practice and therefore provide valuable tools for anyone working to improve HPC I/O. For this analysis, we gather traces of I/O activity from four different systems, including both large and small scale systems and facilities, covering a diverse group of user communities. Through this in-depth study, we answer the following questions:

a) When do applications perform I/O? While work on resource allocation usually considers steady activity [10], [57], [58], which may waste resources [3], [67], efforts to overlap I/O with compute time (dedicated core to perform I/O operations [25], use of an ephemeral file system [73]) rely on operations being more sparse, with sufficient compute time in between. While answering this question, we also verify the common belief that HPC applications read at the beginning

This work has received funding from multiple sources. The authors gratefully acknowledge the support from the Exa-DoST project of the NumPEx PEPR program (ANR-22-EXNU-0004), as part of the France 2030 initiative, and the NCSA-Inria-ANL-BSC-JSC-Riken-UTK Joint-Laboratory on Extreme Scale Computing (JLESC). Furthremore, the authors acknowledge the German Federal Ministry of Education and Research (BMBF) and the Hessian Ministry of Science and Research, Art and Culture (HMWK) for supporting this work as part of the NHR funding. Finally, the authors acknowledge the support of BMBF and the European Commission under the EuroHPC Programme ADMIRE (GA No. 956748, BMBF funding No. 16HPC006K). This work was also conducted within the DecoHPC joint team between Inria and LNCC.

and write at the end of their execution.

b) Are applications periodic? Another assumption about HPC applications is that they perform I/O periodically. This motivates I/O improvement techniques where the behavior of an I/O phase is used to predict the behavior of the next ones [28], [42], [46], [68]. Information about the frequency of I/O phases has also been used for I/O scheduling [13].

c) How long are I/O intervals? We investigate for how long I/O activities usually last. This information is important because, in the absence of prior knowledge about applications' behaviors, many techniques were proposed to characterize them based on recent accesses [37], [44]. Such approaches require some observation time before being able to improve I/O performance, hence the importance of knowing the impact of the observation window length.

d) Do applications always present the same temporal I/O behaviors? An alternative to approaches where recent accesses are observed in order to make predictions about an application's behavior is to rely on knowledge obtained from previous executions of the same application [10], [24], [78]. Answering this question is therefore of paramount importance to evaluate the viability of such proposals.

e) Are there usage behavior patterns for users and projects? They are widely assumed to exist [78], [81]. We study the "campaign duration" of users and projects/accounts and categorize typical usage patterns. Answering these questions about I/O interactions helps the optimization of data staging and hierarchical management [22], [23].

f) How common are concurrent accesses to the shared I/O infrastructure? Our goal with this is twofold: first, we look for evidence of the global I/O performance being degraded due to contention, which motivated the proposal of many contention-avoidance techniques [20], [36]. Secondly, we also quantify how often these concurrent accesses actually happen, and hence how much benefit can be obtained from a technique such as I/O scheduling.

After answering these questions, we propose a *general classification of temporal application I/O behaviors*. We quantify the prevalence of different classes, which allows for the creation of realistic workloads for evaluation of new proposed techniques. Moreover, this classification provides insights for the design of new I/O improvement techniques. Our contributions also include *publishing two large datasets* of I/O traces that allow for further analysis on temporal I/O behaviors. Indeed, the fact that such datasets are extremely rare is one of the reasons why temporal I/O behavior has not been studied so much before.

This paper is organized as follows. After presenting the datasets in Section II, we answer the questions a) to f), listed above, in Sections III to VIII. In Section IX we present our classification of temporal I/O behaviors. Related work is discussed in Section X, followed by final remarks in Section XI.

II. Datasets of job I/O activity

In this paper, we extract insights about temporal I/O behaviors observed in practice by HPC systems. Therefore, we base our study on datasets of job I/O activity obtained from real systems. To focus on realistic workloads, we gathered traces from jobs running over a period of time instead of profiling a set of chosen applications ourselves. Furthermore, we study traces from diverse systems, covering different scales, facilities, user communities and parallel file systems. Our datasets, listed below, cover different periods of time over 11 years. Finally, we have two types of traces: system-side, obtained from the parallel file system, and application-side, from the Darshan profiling tool [19], [54]. Each type has its own limitations and benefits, hence, having the two makes our analyses more complete and robust. The file systems traces, collected by us, have been made available [15] together with the code we developed for the analysis. This allows for reproducibility and extensions of our results.

a) PlaFRIM is an experimental platform from Inria Center at the University of Bordeaux. It has 192 nodes and a BeeGFS with two OSS, each with four OST, default stripe count of 4, and a 100 Gbps network. Its peak I/O performance is of $\approx$12 GiB/s. We collected data over a period of 26 months (from May 2022 to July 2024). The `beegfs-ctl` command was used to obtain bandwidth (grouped by user) every second.

b) SDumont is located at the National Laboratory for Scientific Computing (LNCC), in Brazil, has 36,472 cores on 1,134 nodes, and is capable of 5.1 petaflops. Its Lustre is deployed on 10 OSS, each with one OST. It has a default stripe count of 1 and a peak performance of 30 GiB/s. `collectl` was used every 15 seconds from each compute node to obtain information from the PFS, from January 2020 to December 2020 (12 months). The workload of this machine has been studied by Carneiro et al. [16] and Bez et al. [8]

c) Intrepid was a system from the Argonne National Laboratory, composed of 40,960 nodes. It offered two parallel file systems, PVFS and GPFS, sharing the same 128 file servers [47]. We obtained one year (2013) of Darshan traces, which were downloaded a few years ago from the ALCF I/O Data Repository [17], when they were still publicly available online. This I/O infrastructure has been studied by Lang et al. [47], and two months of traces from 2010 were studied by Carns et al. [18] (however, not regarding temporal behavior).

d) Blue Waters was a 13.3 petaflops system in Illinois, USA, decommissioned in 2021. It featured more than 26,000 compute nodes. The `scratch` Lustre partition was distributed across 360 OSSs and 1440 OSTs. The report by Jones et al. [43] studied general I/O characteristics in the machine. For this work, we used publicly available Darshan traces from 2019 [1], the peak year for machine usage. A preliminary analysis of this data was conducted by Jolivel et al. [42]

A. Filters and final dataset sizes

To answer most of our questions, we are interested in observing the I/O behavior of HPC applications. For this reason, in the file system traces, we excluded all jobs that did not access the PFS or that did not correspond to a full application execution: jobs that completed unsuccessfully, interactive jobs, and jobs that were impacted by temporary

TABLE I: Number of jobs before and after the different filters for file system traces

	Jobs	Concurrent from the same user	Concurrent on the same nodes	Do not use the PFS	Not completed batch jobs	In a blind spot	Too short	I/O benchmarks	Final number of jobs
PlaFRIM	1,434,330	163,400	-	200,273	470,950	1,832	486,547	94,854	**16,474**
SDumont	380,840	-	2,191	141,193	116,182	-	55,560	-	**65,714**

issues with the monitoring tool (the "blind spots"). Moreover, due to the granularity of monitored data, we could not obtain traces from concurrent jobs by the same user (in PlaFRIM) and sharing compute nodes (in SDumont). Additionally, we excluded from PlaFRIM jobs from a large campaign of I/O benchmark executions. Finally, we only considered jobs that executed for at least a certain time (4 seconds in PlaFRIM and 60 seconds in SDumont), because otherwise the time series were not long enough for any meaningful analysis. Table I presents the number of jobs filtered out at each step. We are left with 16,474 jobs for PlaFRIM and 65,714 for SDumont.

For the Darshan traces, we only filtered out corrupted Darshan files and jobs that did not access the PFS. 42,066 jobs were left for Intrepid. For Blue Waters, 316,399 Darshan traces remained, which actually correspond to a smaller number of individual jobs. In the rest of this paper, for simplicity, we refer to these traces as "jobs".

B. General characteristics of the datasets

The extended version of this paper [14, Section 2.2] presents detailed statistics from the datasets. In summary, SDumont jobs present in general low I/O performance (mean of 87 MiB/s), significantly below the system's theoretical peak. Moreover, PlaFRIM and SDumont had similar fractions of time spent on I/O (on average 38 and 44%, median 25 and 37%), despite them being overestimated, due to the resolution of the traces, at different extents. That indicates that this estimation is not that far from reality, or at least that short I/O tends to happen repeatedly in relatively long periods of time. We further study such I/O intervals in Section V.

III. When do applications perform I/O?

The general perception of HPC applications is that they often read at the beginning of their execution, then write periodically throughout the run and/or at the end [27], [31]. The well-known exception is machine learning codes, which issue read operations during most of their executions [21], [29], [32]. In this section, we focus on the question of *when* I/O happens at a high level. Later, in Section IX, we further characterize temporal I/O behavior. The aspect regarding periodic I/O behavior is examined in Section IV.

We first break job executions into four equal-size segments, followed by classifying them into write (`W`), read (`R`), both (`B`), or no I/O (`0`). To mitigate noise from small operations, we discard segments with peak bandwidth below 1 MiB/s and then filter out all `0000` jobs. Additionally, to avoid excessively breaking I/O intervals into segments, we filter jobs shorter than 12 seconds (3 seconds per segment, based on median I/O interval duration as shown in Section V). For SDumont, we actually filter all jobs shorter than 180 seconds, so they have at least three points per segment. After the filters, we classify 10,916 jobs from PlaFRIM, 18,231 from SDumont, 40,519 from Intrepid, and 172,452 from Blue Waters.

Fig. 1 shows the temporal behaviors that were most common in the datasets. We further discuss them below and summarize their prevalence in Table II. For Intrepid and Blue Waters, we add smaller datasets composed of the 516 and 1095 (respectively) longest and largest jobs. They were created by taking the jobs with execution time and number of compute nodes higher than the 90% quartiles (332 and 145 minutes, 4,096 and 123 nodes).

Read at the beginning: This behavior is indeed very common by the prevalence of classes such as `R000`, `R00W`, `B000`, `B00B`, among others. Most jobs read at the beginning of their execution (97% in Intrepid, 86% in PlaFRIM, 78% in Blue Waters and 51% in SDumont). On the other hand, a smaller portion read *only* at this moment, with the exception of the `-large` datasets (see Table II).

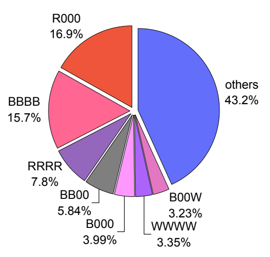
Read throughout the execution: If we count all classes where there are at least 3 segments with read operations (R or B), they account for more jobs (39%) in PlaFRIM than in all others (up to 23%).

Write at the end Although approximately 40% of the jobs in PlaFRIM, SDumont and Intrepid (96% in Blue Waters) write at the end, only between 7% and 37% write *only* at the end. This behavior is even more rare among the largest jobs. Moreover, the anticipated `R00W` class only accounts between 1% and 20% of the jobs (with `R0WW` accounting for less than 3% on all machines). Interestingly, Blue Waters has many jobs with I/O at the beginning and at the end only, but those include writes in the first quarter (`B00B`, `B00W`, etc).

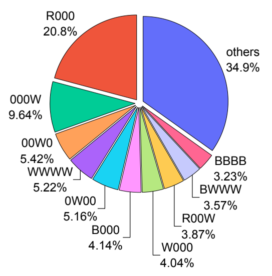
Write throughout the execution Between 20% and 34% of the jobs have writes (`W` or `B`) in at least three segments. Again, the "stereotype" for HPC applications represented by the `RWWW`, `BWWW`, `RRWW`, and `RBWW` classes account together for a small portion of jobs: up to 8% (in Intrepid). Nonetheless, when considering only the largest jobs, these classes account for 65% on Intrepid and 46% on Blue Waters.

TABLE II: When do applications perform I/O?

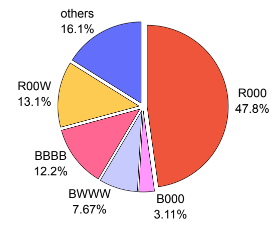
	read			write		
	beginning	throughout	others	end	through out	others
PlaFRIM	32%	39%	29%	7%	34%	59%
SDumont	36%	10%	54%	15%	20%	65%
Intrepid	79%	17%	4%	15%	24%	61%
Intrepid-large	75%	23%	2%	3%	95%	2%
Blue Waters	69%	2%	29%	37%	21%	42%
BW-large	68%	5%	27%	10%	76%	14%



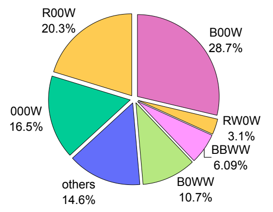
(a) PlaFRIM



(b) SDumont



(c) Intrepid



(d) Blue Waters

Fig. 1: Temporal I/O behavior. "Others" combine all the classes that account for less than 3% of the jobs.

> **When do applications perform I/O?**
>
> Most applications read at the beginning of their execution. However, that is the *only* moment most read only for the oldest system and among the largest jobs. Indeed, the widely accepted description of HPC applications — reading at the beginning and then writing throughout and/or at the end of their execution — does not seem to be that common among shorter jobs, which can write anytime. Finally, in practice a large number of jobs is expected to not fall into any of the described behaviors. Therefore, system-wide I/O improvement techniques should avoid relying on assumptions about when write operations happen.

IV. Are applications periodic?

Due to common patterns in HPC, like checkpointing, many applications perform their I/O periodically. In this section, we examine this statement closer. For that, we used the existing tool FTIO [68], which captures periodic I/O phases. We used its API to develop a parallel extension[1] that quickly processes an entire dataset and merges the results into a single file. For example, the execution for all 16,474 PlaFRIM traces takes 120.96 seconds with 30 processes on a PlaFRIM (Bora) node.

Darshan traces contain aggregated counters per accessed file. To estimate a time series for them, we use the timestamps of first and last read/write to each file. Therefore, we filtered out jobs that accessed a single file, as they would be seen as having a single I/O phase, leaving 41,013 and 246,699 jobs from Intrepid and Blue Waters in this section. As in Section III, we create the Intrepid-large and BW-large datasets of size 519 and 1,077, respectively.

Upon inspecting many jobs, we noticed varying values for the confidence metric. This metric gauges the confidence in the obtained results, but does not allow us to distinguish between periodic and non-periodic behavior. Thus, most jobs for which a dominant frequency was detected had a high confidence value (over 0.5). Initially we consider the I/O phases periodic for all jobs for which FTIO could find a dominant frequency. Hence, we further limit this definition to include jobs for which FTIO detects at least three phases, so it better fits the notion of "periodic I/O". Table III summarizes our findings. We can see the periodic jobs with more than two I/O phases are the minority, between 7 and 26% of the studied jobs. Nonetheless, they are more common among larger jobs (between 31 and 52%). Moreover, we found they are responsible for a more significant part of the resource usage: periodic writes account for over a third of the write time in PlaFRIM and SDumont, and of the total written data in all systems. The extended version [14, Table 5] includes more details of this analysis.

[1] https://github.com/tuda-parallel/FTIO/tree/main/ftio/api/trace_analysis

TABLE III: Proportion of jobs performing periodic I/O

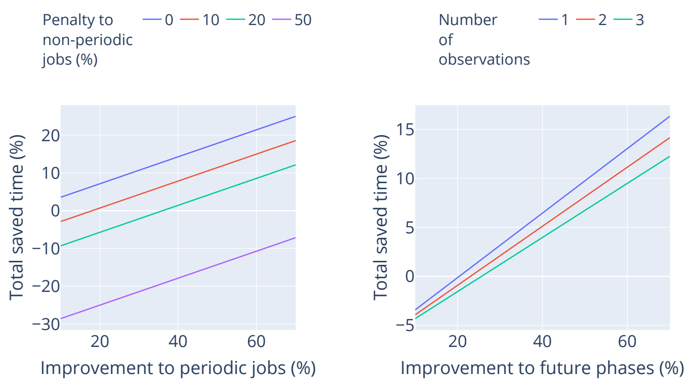
		Periodic read	**Periodic write**	**Periodic read AND write**	**Total periodic**
	PlaFRIM	43.3%	40.6%	27.4%	56.5%
	SDumont	13.5%	30.7%	9.4%	34.8%
	Intrepid	34.3%	24.5%	17.4%	41.4%
	Blue Waters	32.9%	38.1%	17.4%	53.6%
>2 phases	PlaFRIM	16.2%	16.5%	7.1%	25.6%
	SDumont	5.1%	13.8%	1.6%	17.3%
	Intrepid	1.8%	5.8%	0.3%	7.3%
	Intrepid-large	14.6%	17.1%	0.8%	30.9%
	Blue Waters	6.8%	3.4%	0.5%	9.7%
	BW-large	3.1%	50.8%	2.3%	51.6%

It is important to notice that the numbers in Table III are an underestimation. First, PlaFRIM and SDumont traces are limited by their resolution: by Nyquist, any frequency higher than 0.5 and 0.033 Hz, respectively, will appear as constant I/O activity. Second, in Darshan traces all periodic I/O involving a single file cannot be detected.

The extended version [14, Figure 2] includes an analysis of the found periods. In summary, we did not find any value or range that is particularly more common than the others.

A. Improving performance for periodic applications only

Many I/O scheduling algorithms model applications as periodic [13], [31]. Such techniques will improve performance by mitigating contention, but bad decisions (due to non-periodic applications) may harm it. In this section, we estimate the potential savings that could be achieved by I/O improvement techniques that are able to improve performance for periodic applications, at the cost of a penalty for non-periodic ones. Our estimate is a simplification, because we consider each job independently. If, for each job j, $t_{io}(j)$ gives the original I/O



(a) ...periodic applications, but impose a penalty to non-periodic ones (Eq. 2).

(b) ...I/O phases of periodic applications, after observing a number of phases (Eq. 3).

Fig. 2: Total read/write time savings (Eq. 1) potentially yielded by optimization techniques that improve performance for...

TABLE IV: Information about each job j

$t(j)$	Total execution time
$t_{io}(j)$	Total time spent on I/O operations
$n(j)$	Number of periodic I/O phases (given by FTIO)
$t_p(j)$	Time of each periodic I/O phase. Computed from $t_{io}(j)/n(j)$
$t^j_{int}(i)$	Duration of I/O interval i. Note that $t_{io}(j) = \sum t^j_{int}(i)$)
$t^j_{comp}(i)$	For interval i, the time since the previous (for reads) or until the next (for writes) I/O interval

time of the job and $t'_{io}(j)$ is the hypothetical I/O time after an optimization, we compute total savings in time as:

$$savings = (\sum t_{io} - \sum t'_{io}) / \sum t_{io} \tag{1}$$

Table IV lists all information we use about jobs throughout this paper. For this estimate, we define

$$t'_{io}(j) = \begin{cases} t_{io}(j) \times (1 - improvement) & \text{if j is periodic} \\ t_{io}(j) \times (1 + penalty) & \text{otherwise} \end{cases} \tag{2}$$

Results are presented in Fig. 2a. The estimations we present in this paper are optimistic because we ignore the interaction between jobs (improving I/O performance may be impossible in a saturated system), so they provide upper bounds to what can be expected in practice. Moreover, we only compute such estimates for PlaFRIM and SDumont, because Darshan traces do not allow for accurately measuring total I/O time by the jobs. Finally, only the SDumont result for writes is shown due to space limitations, and the other (similar) results can be found in the extended version [14, Figure 3].

We can see that, because periodic jobs are responsible for a large portion of the I/O activity, such techniques can still be beneficial as long as the penalty they impose to non-periodic jobs is not too high and/or the benefit to the periodic ones is high. To put these values into the system's perspective, the best savings in PlaFRIM translated into 13,923 node-hours, or 72 hours of the whole system in a period of 26 months, and in SDumont into 64.5 *days* in a period of 12 months.

Node-hours savings

Because most jobs are *not* I/O intensive, improvements in I/O performance may *not* be highly impactful from the system's point of view. They should be justified by speeding up I/O-intensive applications.

B. *Improving performance after observing some I/O phases*

We now estimate the impact of techniques that can improve I/O performance after observing a number of I/O phases of the job. Predicting future I/O phases is useful, for example, for managing buffers and adapting I/O behaviors (e.g., amount of data) to avoid contention [62]. To estimate it, we compute the time savings as per Eq. 1, assuming such techniques are only effective for periodic applications and that they need to observe N phases before being able to improve the next ones. If job j is periodic, we consider it has $n(j)$ I/O phases of duration $t_p(j) = t_{io}(j)/n(j)$ (we obtain $n(j)$ from FTIO). Thus, we define $t'_{io}(j)$ as follows:

$$t'_{io}(j) = \begin{cases} N \times t_p(j) \times \tau \,+\, (n(j) - N) \times t_p(j) \times \gamma \\ \qquad\qquad\qquad \text{if } j \text{ is periodic and } n(j) > N \\ t_{io}(j) \times \tau \qquad\quad \text{otherwise} \end{cases} \tag{3}$$

With:

$$\begin{aligned} \tau &= \; 1 + overhead \\ \gamma &= \; 1 - improvement \end{aligned} \tag{4}$$

For this we chose an overhead of 10%, a reasonable value based on related work that predicts future phases/accesses using grammars [27], DFT [68], etc., and on I/O profiling tools [18]. Fig. 2b shows the total savings that could be obtained. We can see that these techniques can be beneficial, even if they need to observe multiple phases to characterize applications, as long as they are able to improve performance for future phases by at least ≈30%.

Are applications periodic?

Despite it being widely accepted that HPC applications perform I/O periodically, we found that these are actually a minority of them (less than 26%). However, they account for a larger portion of the I/O time and node-hours. Therefore, it can still be worthy to propose I/O improvement techniques that are only good for periodic applications, even if they need to observe some I/O phases before starting to yield improvements. Still, it is important for such techniques to consider other patterns of I/O are the rule, not the exception.

V. How long are I/O intervals?

In this section, we look into I/O intervals (i.e. periods of time when jobs are doing I/O). Specifically, we are interested in their length, because it indicates how fast techniques that try to dynamically adapt to the current load must be in order to be useful. For each job, we measure periods of uninterrupted I/O activity. We focus on PlaFRIM traces only, because they are the ones with the best resolution. Darshan traces would overestimate I/O intervals length due to the way their temporal behavior is estimated from aggregated counters, as previously discussed, and in SDumont the minimum interval length would be 15 seconds. We found that most intervals (99.9%) last for more than one second, but only 8.9% last for 15 seconds or more. The median duration is 2 seconds. Furthermore, most jobs (70.5%) present at least two intervals, and 44.3% present at least four. Compared to read intervals, write intervals are in general shorter (average of 8 vs. 15 seconds) but jobs have more of them (on average 92 vs. 44). The extended version [14, Section 5] contains more details of these results.

A. Improving performance after observing recent accesses

In Section IV-B, we studied the impact of techniques that observe the application for a few I/O phases to improve performance for future I/O phases if they have similar behavior. Here, we conduct a similar analysis, but our new hypothetical technique does not focus on I/O phases, but rather observes a window of recent accesses in order to improve performance for the future ones. Detecting the current behavior is often used for auto-tuning strategies, for example [5]. We assume this will only be effective inside each I/O interval, since future I/O intervals do not necessarily present the same I/O behaviors. Moreover, we assume again a 10% overhead during the observation period.

For job j, we have $t^j_{int}(i)$ the duration of each I/O interval i (and hence $t_{io}(j) = \sum t^j_{int}(i)$). For interval i, we compute its new duration $t^{j\prime}_{int}(i)$ as:

$$t^{j\prime}_{int}(i) = \begin{cases} s \times \tau + (t^j_{int}(i) - s) \times \gamma & \text{if } t^j_{int}(i) > s \\ t^j_{int}(i) \times \tau & \text{otherwise} \end{cases} \quad (5)$$

With the observation time s, the *overhead* during observation, and the *improvement* after observation. These last two are again represented by τ and γ as per Eq. 4. Then we have:

$$t'_{io}(j) = \sum t^{j\prime}_{int}(i) \quad (6)$$

The total savings are estimated using Eq. 1. Fig. 3 shows the results of this analysis for write intervals. The similar read results can be found in the extended version [14, Figure 5]. We can see even observation periods of a few seconds can bring savings in read and write time. However, an observation window of 4s, for example, can improve performance for over 40% of the jobs while decreasing it for over 50% of them. The ideal window length seems to be between 1 and 2 seconds.

B. Potential for asynchronous I/O

We now investigate the potential for I/O operations to be made asynchronously. That can be done, for example, by using node-local devices for temporary storage, and by dedicating resources into either prefetching data from the PFS or flushing to it. For this study, we consider that operations can be done asynchronously if there is enough time without I/O operations until the next (write) or since the previous (read) I/O operation, and we allow operations to be partially asynchronous. This is therefore a pessimistic estimation, since multiple operations

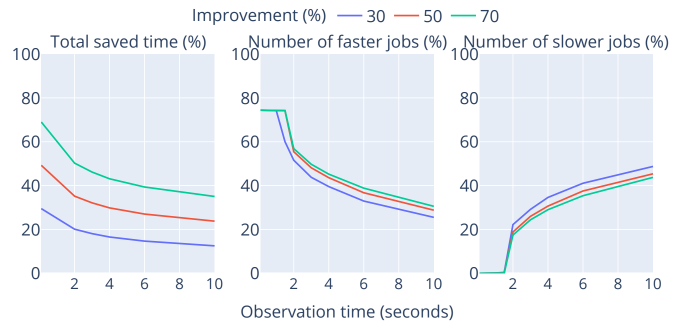


Fig. 3: Impact of performance-improving techniques after observing a job for a given time (x-axis) based on expected future access improvements (the colors). The impact on each job is estimated from Eq. 6, and total savings from Eq. 1

could be prefetched/flushed at the same time if capacity (temporary storage and/or available PFS bandwidth) allows.

For each interval i of job j, of duration $t^j_{int}(i)$, we define $t^j_{comp}(i)$ as the time since the previous I/O interval (for reads) or until the next one (for writes). Then, we compute its potentially asynchronous time as:

$$t^j_{int,async}(i) = min(t^j_{comp}(i), t^j_{int}(i)) \quad (7)$$

We then estimate the I/O time from each job j that can be made asynchronously as $t_{async}(j) = \sum t^j_{int,async}$, and report the total fraction of potentially asynchronous I/O time, obtained from:

$$AsyncRatio = \sum t_{async} / \sum t_{io} \quad (8)$$

Results are presented in Table V. This analysis is, of course, a simplification because some applications could be negatively impacted by dedicating resources for asynchronous I/O. Still, it shows there is great potential for operations to be done asynchronously, especially for writes.

TABLE V: Study of the potential for asynchronous I/O

	AsyncRatio (Eq. 8)	#Jobs with >90% asynchronous I/O	#Jobs with >99% asynchronous I/O
read	23.5%	41%	36.1%
write	74.5%	22.9%	12.7%

How long are I/O intervals?

Most jobs present a few — at least two — separate time intervals with I/O activity, each lasting for a few seconds. As a result, I/O behavior can be observed for a certain time, followed by taking actions to adapt the system and improve performance for future accesses. Nonetheless, such approaches must work on a short window, ideally under ≈1.5 seconds. We also observed that the intervals are often separated enough that I/O operations could be done asynchronously, especially for particularly (74% of the time).

VI. Do applications always present the same temporal I/O behaviors?

An alternative to gathering information about applications by observing them for a few I/O phases (Section IV-B) or for some time (Section V-A) is to use information about previous runs of the same application. For example, profiles obtained during previous executions are often used for resource management [10], [49]. In this section, we investigate at what extent runs from the same application present similar temporal I/O behaviors. For that, we use the Darshan traces (Intrepid and Blue Waters), which include the information of the executable. We have access to the binary name with the arguments used at runtime. However, this technique has limitations as it ignores other parameters, like environment variables for instance, and may lead to an overestimation of identical executions.

Definition 1. we consider multiple traces to come from the same *application* a if they are from the same user, have the same executable name and arguments. We found 9,932 individual applications for Intrepid and 14,622 for Blue Waters.

Detailed results can be found in the extended version [14, Tables 9 and 10]. We found that only 3.3% of Intrepid and 0.7% of Blue Waters individual applications were found to execute more than once. Nonetheless, they are responsible for most of the jobs: 80% and 96%, respectively. Repeating applications were executed on average 24 times on Intrepid (median of 2) and 134 times on Blue Waters (median of 5). Most of these runs (80% and 100%, respectively) were at the same scale.

Despite the large proportion of applications that always run at the same scale, only 17% and 18% were found to always access the same amount of data. For the others, the median difference was of 1 GiB (average of 180 GiB) in Intrepid and 52 KiB (average of 1 TiB) in Blue Waters. Furthermore, most (73% for Intrepid and 91% for Blue Waters) applications that executed more than once always presented the same class of temporal I/O behavior (as defined in Section III, Table II).

Finally, we found a high variability in execution time. Since I/O time is typically not the largest portion of execution time, that indicates an even higher I/O performance variability. The longest run time of each repeating application, divided by the shortest one, results in a median of 1.38 (average of 343) times for Intrepid and 2 (average of 2,419) for Blue Waters. Although less dramatic (average of 7 and 457, respectively), that variability is present even among runs where the same amount of data was accessed. Between those, 20% and 32% have runs that took at least twice the time of others, and 12% and 20% have a maximum difference of at least 3 times.

I/O performance variability

We found high I/O performance variability across application runs, even when handling the same amount of data. For between 20 and 32% of them, the slowest execution took at least twice the time of the fastest one.

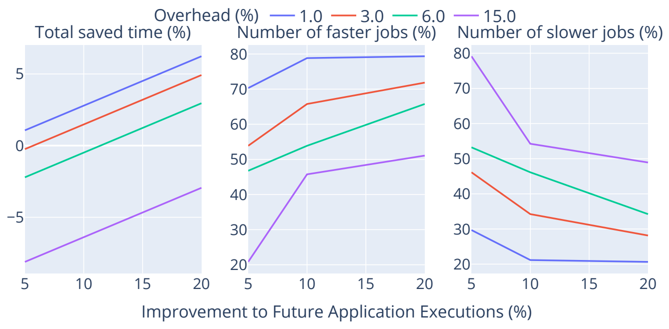


Fig. 4: Results achievable by techniques that use information from a previous run to improve performance in the next ones

We now estimate the impact of a hypothetical technique that gathers information from applications at their first run (imposing an overhead) and is then able to improve performance of the next runs, as long as they have similar behavior.

Definition 2. two jobs of the same application a have *similar behavior* if they have the same scale, same temporal I/O behavior (Table II), and the amounts of data they access do not differ by more than 20%.

For this analysis, we can *not* rely on I/O time, as we did in the previous sections, because the Blue Waters and Intrepid traces do not have a good estimate for total I/O time. Therefore, we use the total execution time instead, and lower the values we test for $overhead$ and $improvement$. For job j of duration $t(j)$, corresponding to an execution of an unique application a, we compute its new duration $t'(j)$, impacted by the I/O improvement technique, as follows. We use τ (for the overhead) and γ (for the improvement) as previously defined (Eq. 4), and consider all jobs for the analysis (including the applications that were executed only once).

$$t'(j) = \begin{cases} t(j) \times \gamma, & \text{if } j \text{ has similar behavior to the previous job of } a, \\ t(j) \times \tau, & \text{if } j \text{ has different behavior or is the first job of } a. \end{cases} \tag{9}$$

Similarly to what was done in Eq. 1, we compute fraction of time saved as:

$$savings = (\sum t - \sum t')/\sum t \tag{10}$$

Results are shown in Fig. 4, where the overhead is represented by the colors, and the improvement by the values in the x axes. We only show the result for Blue Waters, as the one from Intrepid was similar and can be seen in the extended version [14, Figure 6]. We can see improvements for up to 80% of the jobs are possible at a cost for at least 20%.

TABLE VI: Prevalence of user and group patterns. Multiple patterns can be assigned to the same user/group.

Pattern	**Total Users** (n=247)	**%**	**Total Groups** (n=124)	**%**
Mostly Write	55	22.3	39	31.5
Mostly Read	48	19.4	23	18.5
Mixed R/W	21	08.5	18	14.5
Activity Increases	10	04.0	10	08.1
Balance R/W	9	03.6	16	12.9
No Activity	113	45.7	33	26.6

Do applications always present the same temporal I/O behaviors?

Most jobs are runs of the same applications. Repeating applications tend to present similar temporal I/O behaviors across executions (over 70% of the cases). A consequence is that techniques that use information from previous runs to improve future ones seem to have good potential. However, their success depends on a low overhead.

VII. Are there usage behavior patterns for users and projects?

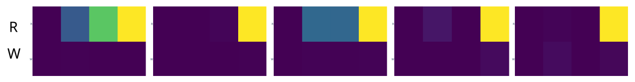
Behavior patterns are widely assumed to exist in the workflows of users across resources and in time. In this section, we use the Blue Waters dataset to investigate it. These jobs come from 247 users across 124 groups/projects accounts. Blue Waters reports both groups and accounts which largely coincide. More than half of the users spent less than 53 days using the system, and 75 % less than 189 days. Groups and accounts tend to be active for longer, at 50% of groups being active less than 107 days and 75% for less than 260 days.

We classify users and groups temporal behaviors, following a similar schema as outlined in Section III: we reduce and normalize the read and write activity of each user or group into a 2x4 matrix with the top row encoding read activity over time and the bottom row write activity. We identified five patterns, which are shown in Fig. 5. A similar figure, for users, can be seen in the extended version [14, Figure 8]. Table VI reports their frequency.

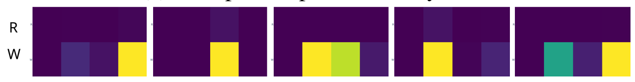
The two most common patterns show many users and projects predominantly read or write data, which aligns well with the idea of data producers and consumers. We further define two mixed patterns and distinguish where read and write activity are balanced and coincide in time, and where activity appears to be more randomly distributed over time. Finally, we notice a number of users and groups that increase their activity over time. Indeed, many teams tend scale up their experiments as the application matures (e.g. adding parallelization and middleware to take advantage of distributed architectures).

Are there usage behavior patterns for users and projects?

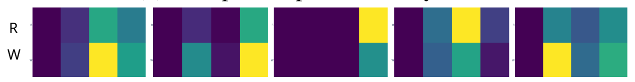
Users and group/patterns activity over time can be classified into a few patterns. It is interesting to notice a large portion predominantly read or write. These interactions give rise to I/O optimization opportunities that go beyond the I/O behavior in a particular run or job.



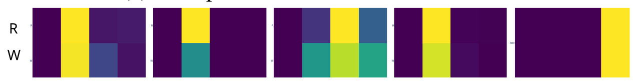
(a) Groups that predominantly read



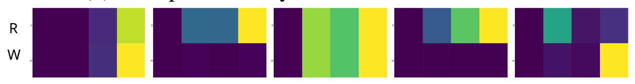
(b) Groups that predominantly write



(c) Groups with mixed R/W workloads



(d) Groups with very balanced R/W workloads



(e) Groups with increasing I/O activity over time

Fig. 5: Examples of group I/O activity over time

VIII. How common are concurrent accesses to the shared I/O infrastructure?

Several approaches to mitigate I/O interference are based on the assumption that parallel file systems are often accessed concurrently by multiple applications and that contention from these accesses impairs global I/O performance [12], [56], [59], [74], [80]. In this section, we investigate this assumption. We use the file system traces, since for Darshan traces we cannot say when exactly I/O operations happened. Moreover, we use all data, before any of the filters discussed in Section II-A. In the case of PlaFRIM, since the monitoring tool reports activity per *user*, it covers *all* activity to the file system, including accesses from the frontend and through `ssh`.

Fig. 6 shows the percentage of time at different concurrency levels. A figure showing a similar trend for PlaFRIM can be found in the extended version [14, Figure 10]. We found that, in both platforms, the PFS is idle for 17% of the time. In PlaFRIM, less than 1% of the time had more than seven concurrent users doing I/O, and in SDumont less than 3% of the time has accesses from more than 15 concurrent jobs.

Next, we examine the impact on bandwidth when multiple users (or jobs) perform I/O simultaneously. For this, we look at the average global bandwidth observed in scenarios with different numbers of concurrent accesses. Because some numbers correspond to only a few data points, we made groups of six. Fig. 7 shows results for SDumont. For both operations, average global I/O performance increases with more applications up to some point, from which additional concurrent applications harm performance. The differences are of factors between 2 and 7. On the other hand, in PlaFRIM results, available in the extended version [14, Figure 11], we see the increase for reads but not the decrease, and there are no differences for write performance. It is worth noting that PlaFRIM's I/O infrastructure has a peak performance between a third and half

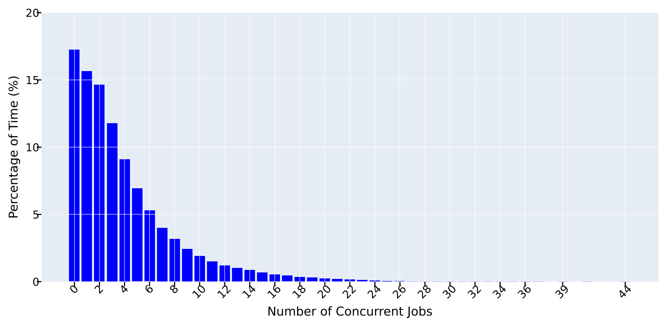


Fig. 6: Percentage of time with I/O accesses by concurrent jobs in SDumont. Percentages smaller than 0.1% were suppressed from the graph to improve readability.

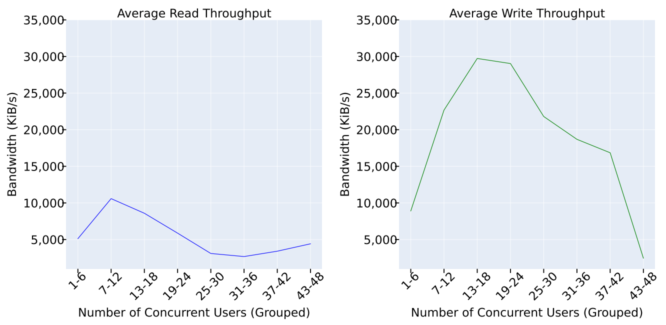


Fig. 7: Average bandwidth in SDumont for different degrees of concurrency. Y axes do not start at zero.

of that of SDumont, for almost six times less nodes. Therefore, it can more comfortably absorb its load. These results evidence that contention from concurrent accesses does harm global I/O performance.

Finally, we observed that global I/O performance surpassed 80% of the maximum observed performance (9.6 GiB/s and 24 GiB/s for PlaFRIM and SDumont, respectively) less than 1.7% of the time for PlaFRIM and even less for SDumont. Such behavior has been documented for larger systems as well [55], [78]. That means we can have performance degradation even when not reaching full system capacity.

How common are concurrent accesses to the shared I/O infrastructure?

Although concurrent accesses are not that common in the studied systems, our results evidence that concurrrent accesses can indeed harm global I/O performance, even when the imposed load is way below systems' capacity. That means other aspects than bandwidth sharing, such as network interference and access patterns interplays, should be the focus of interference mitigation techniques.

IX. A CLASSIFICATION OF APPLICATION TEMPORAL I/O BEHAVIORS

To complement the previous research questions, this section proposes a fine-grained classification of application temporal

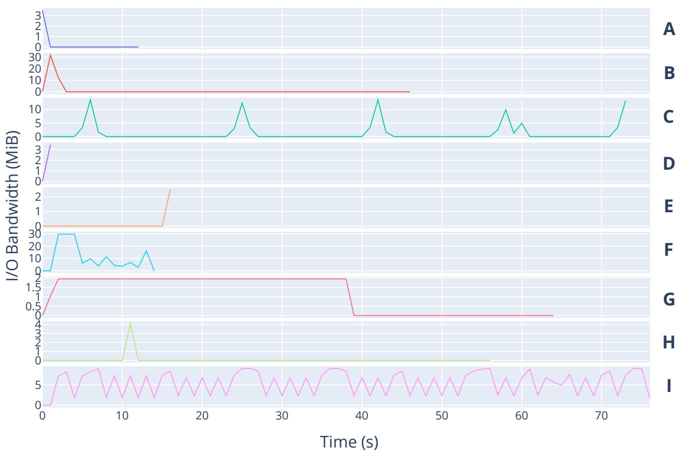


Fig. 8: An example job of each identified category. The x axis shows time, the y axis the bandwidth, and the letters identify the categories. All remaining jobs that do not fit in a category are contained in a heterogeneous set, not represented.

I/O behaviors through clustering of time series. Our goal is to identify common patterns exhibited by applications in practice. We selected SDumont for this analysis as it is both a large system and contains over 60,000 jobs. We only rely on file system traces as we manipulate time series and use I/O bandwidth as a classification criterion. From each job, we extract two dedicated time series: read and write. To remove noise and focus on key I/O activity, we further filter jobs with low peak I/O performance, below 1MiB/s, leaving 37% and 32% of all the read and write time series, respectively.

A. *Methodology for clustering I/O time series*

A challenge when comparing these time series and measuring similarity is that they have vastly different duration (from a few to 10^5 seconds) and amplitudes (the peak I/O performance, which varies within seven orders of magnitude). Therefore, we decided to divide them into groups of similar duration and peak I/O performance. The "duration", "peak I/O", and "ratio" columns from Table VII present these groups. The boundaries between categories were empirically defined and resulted in different values between reads and writes: we aimed both to gather time series with somewhat similar sizes and to avoid groups that are too large, as the resulting clustering cost might not be reasonable (more details below). Interestingly, job duration and peak I/O performance are not strongly correlated, as the whole range of I/O performance appears across all the possible execution times.

Jobs are represented by time series, ordered sets of values with a fixed time gap between samples. To compute a distance between couples of jobs, we use the state of the art method Dynamic Time Warping (DTW). We couple it with K-means clustering, both from the `tslearn` python package [70]. We sample 1,000 jobs per group and apply K-means using 10 clusters. We found that 1,000 is a large enough sample to characterize each group in reasonable time, while 10 clusters cover a large spectrum of behaviors. To further reduce the computation overhead, we also use the `tslearn TimeSeriesResampler` package [70] to reduce the size of long jobs to 80. While this may prune some information, it is a necessary step to scale our classification.

B. Categories of I/O behavior

By manually investigating and labeling each cluster across all the groups, we identified a set of commonly occurring categories. Table VII details all the patterns we identified along with their occurrence across all the groups. Table VIII describes the categories, while Fig. 8 shows an example of job for each. We observe a strong difference between reads and writes, consistently across the various groups. Read activity is often concentrated at the beginning of the jobs, as we observed in Section III, but longer jobs often present different behaviors. Write activity is more heterogeneous, however we can say writing at the beginning is more common for short jobs, while longer jobs more often write throughout the execution. Having a single burst of write operations at some point is also not uncommon. `C`, `F`, `G`, and `I` jobs are often periodic.

We present in Fig. 9 the most populated cluster, for each read and write group of duration 80, by displaying its centroid. We observe that read centroids experience a peak at the beginning of the job. On the opposite, write centroids present much more diverse behavior, illustrating the complex IO behaviors.

Patterns of temporal I/O behavior

The fact that we found a relatively small number of temporal I/O behavior patterns indicates it is possible to target specific behaviors for performance optimization. Moreover, the fact that there are dominating clusters in most groups says temporal I/O behavior is somewhat predictable.

Finally, we conclude clustering can be used to classify job traces. However, it is important to notice that in this paper we used manual labelization, and different methods could be required for an automatic solution. At the same time, a combination of the various approaches we used to answer the questions (for example, the segments analysis from Section III combined with FTIO output from Section IV) could provide a similar classification at a lower overhead than tracing and clustering time series.

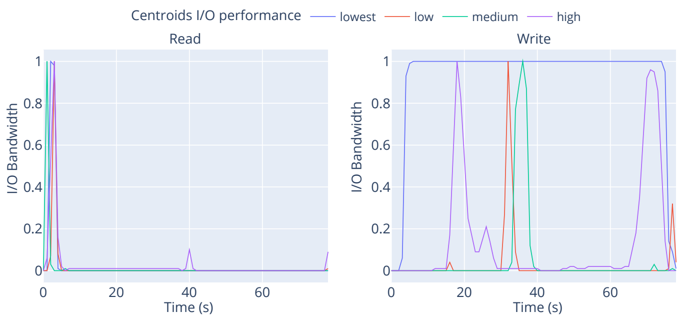


Fig. 9: Centroid of each dominant cluster per group across read and write time series of length 80. The x axis shows time, while the y axis the normalized bandwidth values of each centroid with respect to its maximum value, so that we can compare the trends between centroids.

X. Related Work

Given the widening gap between computing power and I/O performance, understanding I/O behavior on large-scale systems has been a fairly active field for several years [51]. It generally covers the way storage systems absorb I/O, on the one hand, and application access patterns, on the other.

On the storage system side, several studies have focused on analyzing the deployment of specific storage systems after a few months of production [9], [45], [52], [60]. The aim of these studies is generally to provide both users and administrators with optimization suggestions. These conclusions are aimed at a specific system, and are rarely generalizable to other architectures. In addition, this type of work requires a large volume of logs from several sources within the system (file system logs, application monitoring tools, etc.) [50]. Attempts have nevertheless been made to provide more global optimization paths that may be suitable for several systems, albeit limited to the configuration parameters of these systems [5], [12], [21], [65]. However, those approaches require privileged access to storage systems, which makes the task difficult.

On the application side, Darshan traces are often used to study a limited set of supercomputers. For instance, in [53], the authors analyze a year-long set of I/O traces collected from three platforms. The main objective of their study is to investigate how system state impacts the I/O performance of applications. Similarly, in [74], the authors focus on jobs in the ”wild” by studying the Darshan traces gathered from production jobs on a Cray XC40 machine over two months. Based on that, they propose a top-down analysis and apply it to more than 88,000 I/O traces, showing that their approach can identify common performance bottlenecks and uncover significant contributing factors for poor performance. In [59], the authors collect and analyze a year of traces from the storage system of a large-scale production system at NERSC. They are able to show that as of 2019, their data indicates that read operations are dominant (i.e., applications are reading more than writing) which is counter-intuitive. In another study, Yu *et al.* [80] collect and analyze I/O traces from two production HPC systems: Tianhe-1A and SC19, both located in China. In both systems, the traces are collected over a period of less than 30 days (in 2018 and 2019). Their focus is on the spatiality of bursty I/O operations, showing that traffic peaks often originate from a minority of compute nodes located close to each other. In [38], the authors study several years of Darshan logs (from 2017 to 2020) from the Theta supercomputer (at ALCF), with a main focus on understanding the causes of poor I/O throughput. Another work, by Yang et al. [79], proposes a metric to measure burstiness of I/O traces and apply it to conclude half the jobs in their cluster (which only runs CFD applications) are highly bursty. Finally, in [56], the authors study the Intrepid traces (although not from 2013) and observe high I/O performance variability.

read categorization													write categorization											
duration	peak I/O	ratio	A	B	C	D	E	F	G	H	I	J	peak I/O	ratio	A	B	C	D	E	F	G	H	I	J
1 - 20	1 - 4	*0.07*	0.73	0.13	0.01	0.06	0.00	0.00	0.00	0.00	0.00	0.07	1 - 5	*0.07*	0.00	0.31	0.05	0.00	0.11	0.11	0.30	0.00	0.00	0.12
1 - 20	4 - 20	*0.12*	0.48	0.35	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.17	5 - 40	*0.11*	0.00	0.62	0.02	0.00	0.05	0.05	0.19	0.00	0.00	0.08
1 - 20	20 - 100	*0.15*	0.87	0.00	0.00	0.03	0.03	0.00	0.00	0.00	0.00	0.07	40 - 150	*0.08*	0.00	0.51	0.00	0.00	0.10	0.03	0.32	0.00	0.00	0.03
1 - 20	100 - 10^6	*0.05*	0.42	0.43	0.00	0.00	0.00	0.11	0.00	0.00	0.00	0.04	150 - 10^7	*0.03*	0.19	0.35	0.00	0.00	0.00	0.34	0.00	0.00	0.00	0.12
20 - 80	1 - 4	*0.05*	0.87	0.00	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.07	1 - 5	*0.05*	0.00	0.00	0.13	0.00	0.00	0.37	0.12	0.29	0.00	0.09
20 - 80	4 - 20	*0.08*	0.79	0.10	0.00	0.00	0.04	0.00	0.04	0.00	0.00	0.03	5 - 40	*0.14*	0.00	0.00	0.06	0.00	0.06	0.00	0.31	0.35	0.06	0.16
20 - 80	20 - 100	*0.16*	0.70	0.00	0.00	0.00	0.24	0.00	0.00	0.00	0.00	0.06	40 - 150	*0.11*	0.00	0.00	0.11	0.00	0.00	0.07	0.77	0.00	0.00	0.06
20 - 80	100 - 10^6	*0.06*	0.47	0.00	0.00	0.00	0.00	0.47	0.00	0.00	0.00	0.06	150 - 10^7	*0.06*	0.00	0.00	0.00	0.23	0.23	0.23	0.00	0.23	0.00	0.06
80 - 10^5	1 - 4	*0.03*	0.37	0.10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.53	1 - 5	*0.09*	0.00	0.00	0.05	0.00	0.00	0.71	0.00	0.20	0.00	0.05
80 - 10^5	4 - 20	*0.1*	0.75	0.07	0.00	0.00	0.00	0.05	0.07	0.00	0.00	0.07	5 - 40	*0.08*	0.00	0.00	0.00	0.00	0.00	0.38	0.00	0.16	0.28	0.18
80 - 10^5	20 - 100	*0.07*	0.28	0.09	0.00	0.00	0.09	0.00	0.10	0.00	0.00	0.44	40 - 150	*0.06*	0.00	0.00	0.04	0.00	0.19	0.33	0.00	0.04	0.11	0.28
80 - 10^5	100 - 10^6	*0.07*	0.19	0.19	0.00	0.00	0.00	0.00	0.19	0.19	0.00	0.24	150 - 10^7	*0.10*	0.00	0.00	0.00	0.00	0.00	0.00	0.42	0.42	0.00	0.15

TABLE VII: **Jobs grouping and categorization**. Each entry presents a group of read/write time series according to both their duration and peak I/O performance. Each read/write group has a ratio indicating the proportion of time series that belong to this group. For instance, 15% of all the read time series last between 1 and 20 seconds, with I/O performance between 20 and 100 MiB/s. Then, for each homogeneous group, we identify the different categories of time series that occur. For instance, 51% of the write time series with duration between 1 and 20 seconds and peak I/O performance between 40 and 150 MiB/s fall in category `B`.

TABLE VIII: Categories description

category	description
A	Burst at the start of the job
B	Longer burst at the beginning of the job
C	Multiple, relatively short, I/O phases
D	Short job dominated by I/O
E	Ends with I/O activity
F	I/O activity dominates the job
G	Continuous I/O activity for a big part of the job
H	Short I/O burst but not at a specific moment
I	Dominating I/O with significant fluctuations
J	Hectic I/O activity

Among other results, those papers demonstrate the effectiveness of I/O traces in providing insights into a large-scale I/O system. However, although they provide a relevant perspective on I/O behavior, the temporal aspect is mostly omitted. Such temporal I/O behavior studies are challenging because datasets are limited either in terms of machines, or in terms of the volume of data collected and analyzed. Moreover, their analysis requires the use of adapted techniques such as clustering. On this subject, Betke and Kunkel [7] design and evaluate multiple approaches for clustering 1 million jobs. Their approach aims at covering multiple aspects of I/O behavior, including the temporal one, at once. That leads to a large number of clusters, which makes extracting information from them difficult. In contrast, we focused on clustering individual time series separately.

A straightforward and computationally intensive approach for clustering time series consists in using DTW distance with a clustering algorithm [2], [41], [70]. Alternative methods [30], [76] extract features that act as proxies to calculate the distances. While potentially cheap to apply, the key challenge of such approaches is to ensure that features capture the statistical properties that can discriminate in a meaningful way. Deep neural networks (DNNs) showed promising results in augmenting such features or by directly classifying series [39], [82]. We plan on exploring other methods towards a more automatic clustering of I/O traces as future work. In this paper, we fill a gap in the literature by proposing an in-depth study of temporal I/O behavior based on massive heterogeneous datasets.

XI. Conclusion

In this paper, we studied the temporal I/O behavior of over 440,000 jobs running on four HPC systems, all different in terms of infrastructure, scale, and users, covering several time periods over the last 11 years. The data we analyzed came either from parallel file systems (system-side traces) or from I/O monitoring tools (application-side traces). The aim of analyzing these traces is to provide an in-depth study of data accesses by HPC applications in the wild. We have thus identified and addressed a number of questions dealing with the temporality of I/Os, their periodicity, the existence and prevalence of certain patterns, I/O concurrency between applications or user practices. We also proposed a classification of temporal I/O behaviors, which shows a few patterns are able to represent a vast majority of jobs. Finally, we make two large datasets of traces available for the community. Overall, the results of this study provide relevant information for anyone working to improve high-performance I/O. They also serve as a basis for future research into both behavior detection tools and the use of trace analysis, particularly for scheduling and application optimization.

Acknowledgements

Computational resources and traces were provided by the Argonne Leadership Computing Facility (DOE contract DE-AC02-06CH11357), the PlaFRIM experimental testbed (Inria, CNRS, Université de Bordeaux, Bordeaux INP, and Région Nouvelle-Aquitaine, PlaFRIM), and the SDumont supercomputer at LNCC/MCTI, Brazil. The authors gratefully thank Brice Goglin, Julien Lelaurain, and the entire PlaFRIM team for their technical support in data acquisition.

References

[1] Blue waters data-sets. https://bluewaters.ncsa.illinois.edu/data-sets.

[2] A. Abanda, U. Mori, and J. A. Lozano. A review on distance based time series classification. *Data Mining and Knowledge Discovery*, 33(2):378–412, 2019.

[3] A. Bandet, F. Boito, and G. Pallez. Scheduling distributed I/O resources in HPC systems. In *30th International European Conference on Parallel and Distributed Computing 26 - 30 August 2024 Madrid, Spain 30th International European Conference on Parallel and Distributed Computing*, Madrid, Spain, Aug. 2024.

[4] BeeGFS. Beeond: BeeGFS On Demand, 2018. https://www.beegfs.io/wiki/BeeOND.

[5] B. Behzad, S. Byna, Prabhat, and M. Snir. Optimizing I/O performance of HPC applications with autotuning. *ACM Trans. on Parallel Computing*, 5(4):15:1–15:27, Mar. 2019.

[6] A. Benoit, T. Herault, L. Perotin, Y. Robert, and F. Vivien. Revisiting I/O bandwidth-sharing strategies for HPC applications. Technical Report RR-9502, INRIA, Mar. 2023.

[7] E. Betke and J. Kunkel. The importance of temporal behavior when classifying job I/O patterns using machine learning techniques. In *High Performance Computing: ISC High Performance 2020 International Workshops, Frankfurt, Germany, June 21–25, 2020, Revised Selected Papers 35*, pages 191–205. Springer, 2020.

[8] J. Bez, A. Ramos Carneiro, P. J. Pavan, V. Soldera Girelli, F. Boito, B. A. Fagundes, C. Osthoff, P. Leite da Silva Dias, J.-F. Méhaut, and P. O. Navaux. I/O Performance of the Santos Dumont Supercomputer. *International Journal of High Performance Computing Applications*, 34(2):1–17, Sept. 2019.

[9] J. L. Bez, A. M. Karimi, A. K. Paul, B. Xie, S. Byna, P. Carns, S. Oral, F. Wang, and J. Hanley. Access patterns and performance behaviors of multi-layer supercomputer I/O subsystems under production load. In *Proc. of the 31st Int. Symp. on High-Performance Parallel and Distributed Computing*, page 43–55, Minneapolis MN USA, June 2022. ACM.

[10] J. L. Bez, A. Miranda, R. Nou, F. Boito, T. Cortes, and P. Navaux. Arbitration policies for on-demand user-level I/O forwarding on HPC platforms. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 577–586, 2021.

[11] R. Bleuse, K. Dogeas, G. Lucarelli, G. Mounié, and D. Trystram. Interference-aware scheduling using geometric constraints. In *Euro-Par'18*, pages 205–217. Springer, 2018.

[12] F. Boito, G. Pallez, and L. Teylo. The role of storage target allocation in applications' I/O performance with beegfs. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 267–277, 2022.

[13] F. Boito, G. Pallez, L. Teylo, and N. Vidal. IO-sets: Simple and efficient approaches for I/O bandwidth management. *IEEE Transactions on Parallel and Distributed Systems*, 34(10):2783–2796, 2023.

[14] F. Boito, L. Teylo, M. Popov, T. Jolivel, F. Tessier, J. Luettgau, J. Monniot, A. Tarraf, A. Carneiro, and C. Osthoff. A deep look into the temporal I/O behavior of HPC applications — extended version. Technical Report RR-9577, Inria & Labri, Univ. Bordeaux, Mar. 2025.

[15] F. Z. Boito, L. Teylo, M. Popov, T. Jolivel, F. Tessier, J. Luettgau, J. Monniot, A. Tarraf, A. R. Carneiro, and C. Osthoff. A deep look into the temporal I/O behavior of HPC applications [dataset], Mar. 2025. https://doi.org/10.5281/zenodo.14965920.

[16] A. R. Carneiro, J. L. Bez, C. Osthoff, L. M. Schnorr, and P. O. A. Navaux. Hpc data storage at a glance: The santos dumont experience. In *2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 157–166, 2021.

[17] P. Carns. ALCF I/O Data Repository. https://www.alcf.anl.gov/publications/alcf-io-data-repository.

[18] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross. Understanding and improving computational science storage access through continuous characterization. *ACM Trans. Storage*, 7(3), Oct. 2011.

[19] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley. 24/7 characterization of petascale I/O workloads. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10. IEEE, 2009.

[20] J. Carretero, E. Jeannot, G. Pallez, D. E. Singh, and N. Vidal. Mapping and scheduling hpc applications for optimizing I/O. In *Proceedings of the 34th ACM International Conference on Supercomputing*, ICS '20, New York, NY, USA, 2020. Association for Computing Machinery.

[21] F. Chowdhury, Y. Zhu, T. Heer, S. Paredes, A. Moody, R. Goldstone, K. Mohror, and W. Yu. I/O characterization and performance evaluation of beegfs for deep learning. In *Proceedings of the 48th International Conference on Parallel Processing*, ICPP '19, New York, NY, USA, 2019. Association for Computing Machinery.

[22] R. F. da Silva, R. M. Badia, V. Bala, and et al. Workflows Community Summit 2022: A Roadmap Revolution. Technical report, Zenodo, Mar. 2023.

[23] E. Deelman, T. Peterka, I. Altintas, C. D. Carothers, K. K. van Dam, K. Moreland, M. Parashar, L. Ramakrishnan, M. Taufer, and J. Vetter. The future of scientific workflows. *The International Journal of High Performance Computing Applications*, 32(1):159–175, Jan. 2018.

[24] H. Devarajan and K. Mohror. Mimir: Extending I/O interfaces to express user intent for complex workloads in HPC. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 178–188, 2023.

[25] M. Dorier, G. Antoniu, F. Cappello, M. Snir, R. Sisneros, O. Yildiz, S. Ibrahim, T. Peterka, and L. Orf. Damaris: Addressing performance variability in data management for post-petascale simulations. *ACM Trans. Parallel Comput.*, 3(3), Oct. 2016.

[26] M. Dorier, G. Antoniu, R. Ross, D. Kimpe, and S. Ibrahim. CALCioM: Mitigating I/O interference in HPC systems through cross-application coordination. In *IPDPS'14*, pages 155–164. IEEE, 2014.

[27] M. Dorier, S. Ibrahim, G. Antoniu, and R. Ross. Omnisc'IO: A grammar-based approach to spatial and temporal I/O patterns prediction. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 623–634, 2014.

[28] M. Dorier, S. Ibrahim, G. Antoniu, and R. Ross. Using formal grammars to predict I/O behaviors in hpc: The omnisc'IO approach. *IEEE Transactions on Parallel and Distributed Systems*, 27(8):2435–2449, 2016.

[29] N. Dryden, R. Böhringer, T. Ben-Nun, and T. Hoefler. Clairvoyant prefetching for distributed machine learning I/O. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[30] B. D. Fulcher and N. S. Jones. Highly comparative feature-based time-series classification. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):3026–3037, 2014.

[31] A. Gainaru, G. Aupy, A. Benoit, F. Cappello, Y. Robert, and M. Snir. Scheduling the I/O of hpc applications under congestion. In *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 1013–1022, 2015.

[32] A. Gainaru, D. Ganyushin, B. Xie, T. Kurc, J. Saltz, S. Oral, N. Podhorszki, F. Poeschel, A. Huebl, and S. Klasky. Understanding and leveraging the I/O patterns of emerging machine learning analytics. In J. Nichols, A. B. Maccabe, J. Nutaro, S. Pophale, P. Devineni, T. Ahearn, and B. Verastegui, editors, *Driving Scientific and Engineering Discoveries Through the Integration of Experiment, Big Data, and Modeling and Simulation*, pages 119–138, Cham, 2022. Springer International Publishing.

[33] J. Garcia-Blas, D. E. Singh, and J. Carretero. IMSS: In-memory storage system for data intensive applications. In *ISC High Performance 2022 International Workshops*, volume 13387 of *Lecture Notes in Computer Science*, pages 190–205. Springer International Publishing, Cham, 2022.

[34] F. Garcia-Carballeira, A. Calderon, J. Carretero, J. Fernandez, and J. M. Perez. The design of the expand parallel file system. *The International Journal of High Performance Computing Applications*, 17(1):21–37, 2003.

[35] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. *SIGCOMM Comput. Commun. Rev.*, 44(4):455–466, Aug. 2014.

[36] S. Herbein, D. H. Ahn, D. Lipari, T. R. Scogland, M. Stearman, M. Grondona, J. Garlick, B. Springmeyer, and M. Taufer. 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*, HPDC '16, page 69–80, New York, NY, USA, 2016. Association for Computing Machinery.

[37] F. Isaila, J. Carretero, and R. Ross. Clarisse: A middleware for data-staging coordination and control on large-scale HPC platforms. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 346–355, 2016.

[38] M. Isakov, E. Del Rosario, S. Madireddy, P. Balaprakash, P. Carns, R. B. Ross, and M. A. Kinsy. HPC I/O throughput bottleneck analysis with explainable local models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13. IEEE, 2020.

[39] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4):917–963, 2019.

[40] E. Jeannot, G. Pallez, and N. Vidal. Scheduling periodic I/O access with bi-colored chains: models and algorithms. *J. of Scheduling*, 24(5):469–481, 2021.

[41] Y.-S. Jeong, M. K. Jeong, and O. A. Omitaomu. Weighted dynamic time warping for time series classification. *Pattern recognition*, 44(9):2231–2240, 2011.

[42] T. Jolivel, F. Tessier, J. Monniot, and G. Pallez. Mosaic: Detection and categorization of I/O patterns in hpc applications. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1311–1319, 2024.

[43] M. D. Jones, J. P. White, M. Innus, R. L. DeLeon, N. Simakov, J. T. Palmer, S. M. Gallo, T. R. Furlani, M. T. Showerman, R. Brunner, A. Kot, G. H. Bauer, B. M. Bode, J. Enos, and W. T. Kramer. Workload analysis of blue waters. *CoRR*, abs/1703.00924, 2017.

[44] E. Karrels, L. Huang, Y. Kan, I. Arora, Y. Wang, D. S. Katz, W. Gropp, and Z. Zhang. Fine-grained policy-driven I/O sharing for burst buffers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, New York, NY, USA, 2023. Association for Computing Machinery.

[45] A. Khan, H. Sim, S. S. Vazhkudai, A. R. Butt, and Y. Kim. An analysis of system balance and architectural trends based on Top500 supercomputers. In *The Int. Conf. on High Performance Computing in Asia-Pacific Region*, page 11–22, Virtual Event Republic of Korea, Jan. 2021. ACM.

[46] S. Kim, A. Sim, K. Wu, S. Byna, Y. Son, and H. Eom. 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, page 77–88, New York, NY, USA, 2020. Association for Computing Machinery.

[47] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock. I/O performance challenges at leadership scale. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, New York, NY, USA, 2009. Association for Computing Machinery.

[48] Lawrence Livermore National Laboratory. UnifyFS, 2019. https://github.com/LLNL/UnifyFS.

[49] Y. Liu, R. Gunasekaran, X. Ma, and S. S. Vazhkudai. Server-Side Log Data Analytics for I/O Workload Characterization and Coordination on Large Shared Storage Systems. In *SC'16*, pages 819–829, Nov 2016.

[50] Z. Liu, R. Lewis, R. Kettimuthu, K. Harms, P. Carns, N. Rao, I. Foster, and M. E. Papka. Characterization and identification of HPC applications at leadership computing facility. In *Proc. of the 34th ACM Int. Conf. on Supercomputing*, page 1–12, Barcelona Spain, June 2020. ACM.

[51] G. Lockwood, D. Hazen, Q. Koziol, R. Canon, K. Antypas, and J. Balewski. Storage 2020: A Vision for the Future of HPC Storage. In *Report: LBNL-2001072*. Lawrence Berkeley National Laboratory, 2017.

[52] G. K. Lockwood, S. Snyder, S. Byna, P. Carns, and N. J. Wright. Understanding data motion in the modern HPC data center. In *2019 IEEE/ACM Fourth Int. Parallel Data Systems Workshop (PDSW)*, page 74–83, Nov. 2019.

[53] G. K. Lockwood, S. Snyder, T. Wang, S. Byna, P. Carns, and N. J. Wright. A Year in the Life of a Parallel File System. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 931–943, Nov. 2018.

[54] J. Luettgau, S. Snyder, T. Reddy, N. Awtrey, K. Harms, J. L. Bez, R. Wang, R. Latham, and P. Carns. Enabling Agile Analysis of I/O Performance Data with PyDarshan. In *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, SC-W '23, pages 1380–1391, New York, NY, USA, Nov. 2023. Association for Computing Machinery.

[55] H. Luu, M. Winslett, W. Gropp, R. Ross, P. Carns, K. Harms, M. Prabhat, S. Byna, and Y. Yao. A multiplatform study of I/O behavior on petascale supercomputers. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 33–44, 2015.

[56] S. Madireddy, P. Balaprakash, P. Carns, R. Latham, R. Ross, S. Snyder, and S. M. Wild. Analysis and correlation of application I/O performance and system-wide I/O activity. In *2017 International Conference on Networking, Architecture, and Storage (NAS)*, pages 1–10. IEEE, 2017.

[57] J. Monniot, F. Tessier, H. Casanova, and G. Antoniu. Simulation of Large-Scale HPC Storage Systems: Challenges and Methodologies . In *2024 IEEE 31st International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pages 232–242, Los Alamitos, CA, USA, Dec. 2024. IEEE Computer Society.

[58] J. Monniot, F. Tessier, M. Robert, and G. Antoniu. Supporting dynamic allocation of heterogeneous storage resources on HPC systems. *Concurrency and Computation: Practice and Experience*, 35(28):e7890, 2023.

[59] T. Patel, S. Byna, G. K. Lockwood, and D. Tiwari. Revisiting I/O behavior in large-scale storage systems: The expected and the unexpected. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2019.

[60] A. K. Paul, O. Faaland, A. Moody, E. Gonsiorowski, K. Mohror, and A. R. Butt. Understanding HPC application I/O behavior using system level statistics. In *2020 IEEE 27th Int. Conf. on High Performance Computing, Data, and Analytics (HiPC)*, pages 202–211, 2020.

[61] L. Pottier, R. F. da Silva, H. Casanova, and E. Deelman. Modeling the performance of scientific workflow executions on HPC platforms with burst buffers. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 92–103, 2020.

[62] Z. Qiao, Q. Liu, N. Podhorszki, S. Klasky, and J. Chen. Taming I/O variation on QoS-Less HPC storage: What can applications do? In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2020.

[63] F. J. Rodrigo Duro, F. Marozzo, J. García Blas, J. Carretero Pérez, D. Talia, and P. Trunfio. Evaluating data caching techniques in DMCF workflows using hercules. In *Proceedings of the Second International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2015)*, pages 95–106, Spain, 2015. Universidad Carlos III de Madrid.

[64] E. Saeedizade, R. Taheri, and E. Arslan. I/O burst prediction for HPC clusters using darshan logs. In *2023 IEEE 19th International Conference on e-Science (e-Science)*, pages 1–10, Los Alamitos, CA, USA, oct 2023. IEEE Computer Society.

[65] M. Seiz, P. Offenhäuser, S. Andersson, J. Hötzer, H. Hierl, B. Nestler, and M. Resch. Lustre I/O performance investigations on hazel hen: experiments and heuristics. *The J. of Supercomputing*, 77(11):12508–12536, Nov. 2021.

[66] K. Sivalingam and H. Richardson. Application IO analysis with lustre monitoring using LASSi for ARCHER. In *High Performance Computing: ISC High Performance 2020 International Workshops, Frankfurt, Germany, June 21–25, 2020, Revised Selected Papers*, page 255–266, Berlin, Heidelberg, 2020. Springer-Verlag.

[67] H. Sung, J. Bang, C. Kim, H.-S. Kim, A. Sim, G. K. Lockwood, and H. Eom. BBOS: Efficient HPC storage management via burst buffer over-subscription. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 142–151, 2020.

[68] A. Tarraf, A. Bandet, F. Boito, G. Pallez, and F. Wolf. Capturing periodic I/O using frequency techniques. In *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 465–478. IEEE, May 2024.

[69] O. Tatebe, K. Obata, K. Hiraga, and H. Ohtsuji. CHFS: parallel consistent hashing file system for node-local persistent memory. In *HPC Asia 2022: International Conference on High Performance Computing in Asia-Pacific Region*, pages 115–124, USA, 2022. ACM.

[70] R. Tavenard, J. Faouzi, G. Vandewiele, F. Divo, G. Androz, C. Holtz, M. Payne, R. Yurchak, M. Rußwurm, K. Kolar, et al. Tslearn, a machine learning toolkit for time series data. *Journal of machine learning research*, 21(118):1–6, 2020.

[71] T. T. Tran, M. Padmanabhan, P. Y. Zhang, H. Li, D. G. Down, and J. C. Beck. Multi-stage resource-aware scheduling for data centers with heterogeneous servers. *J. of Scheduling*, 21(2):251–267, Apr. 2018.

[72] D. Ucar and E. Arslan. Streaming file transfer optimization for distributed science workflows. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 187–197, 2020.

[73] M. Vef, N. Moti, T. Süß, T. Tocci, R. Nou, A. Miranda, T. Cortes, and A. Brinkmann. GekkoFS - A temporary distributed file system for HPC applications. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 319–324, USA, Sept. 2018. IEEE.

[74] T. Wang, S. Byna, G. K. Lockwood, S. Snyder, P. Carns, S. Kim, and N. J. Wright. A zoom-in analysis of I/O logs to detect root causes of i/o performance bottlenecks. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 102–111. IEEE, 2019.

[75] T. Wang, K. Mohror, A. Moody, K. Sato, and W. Yu. An ephemeral burst-buffer file system for scientific applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 807–818, USA, Nov. 2016. IEEE.

[76] Z. Xiao, X. Xu, H. Xing, S. Luo, P. Dai, and D. Zhan. Rtfn: A robust temporal feature network for time series classification. *Information sciences*, 571:65–86, 2021.

[77] B. Yang, X. Ji, X. Ma, X. Wang, T. Zhang, X. Zhu, N. El-Sayed, H. Lan, Y. Yang, J. Zhai, W. Liu, and W. Xue. End-to-end I/O monitoring on a leading supercomputer. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 379–394, Boston, MA, Feb. 2019. USENIX Association.

[78] B. Yang, Y. Zou, W. Liu, and W. Xue. An end-to-end and adaptive I/O optimization tool for modern HPC storage systems. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1294–1304, 2022.

[79] W. Yang, X. Liao, D. Dong, and J. Yu. A quantitative study of the spatiotemporal I/O burstiness of HPC application. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1349–1359, 2022.

[80] J. Yu, W. Yang, F. Wang, D. Dong, J. Feng, and Y. Li. Spatially bursty I/O on supercomputers: Causes, impacts and solutions. *IEEE Transactions on Parallel and Distributed Systems*, 31(12):2908–2922, 2020.

[81] W. Zhang, S. Byna, H. Sim, S. Lee, S. Vazhkudai, and Y. Chen. Exploiting user activeness for data retention in HPC systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[82] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu. Convolutional neural networks for time series classification. *Journal of systems engineering and electronics*, 28(1):162–169, 2017.