Uncovering Access, Reuse, and Sharing Characteristics of I/O-Intensive Files on Large-Scale Production HPC Systems

Tirthak Patel, *Northeastern University;* Suren Byna, Glenn K. Lockwood, and Nicholas J. Wright, *Lawrence Berkeley National Laboratory;* Philip Carns and Robert Ross, *Argonne National Laboratory;* Devesh Tiwari, *Northeastern University*

https://www.usenix.org/conference/fast20/presentation/patel-hpc-systems

This paper is included in the Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST '20)

February 25–27, 2020 • Santa Clara, CA, USA

978-1-939133-12-0

Open access to the Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST '20) is sponsored by

Uncovering Access, Reuse, and Sharing Characteristics of I/O-Intensive Files on Large-Scale Production HPC Systems

Tirthak Patel
Northeastern University

Suren Byna, Glenn K. Lockwood, Nicholas J. Wright
Lawrence Berkeley National Laboratory

Philip Carns, Robert Ross
Argonne National Laboratory

Devesh Tiwari
Northeastern University

Abstract

Large-scale high-performance computing (HPC) applications running on supercomputers produce large amounts of data routinely and store it in files on multi-PB shared parallel storage systems. Unfortunately, storage community has a limited understanding of the access and reuse patterns of these files. This paper investigates the access and reuse patterns of I/O-intensive files on a production-scale supercomputer.

1 Introduction

High-performance computing (HPC) applications running on large-scale facilities routinely perform TBs of I/O. Consequently, significant efforts have been made to study the I/O behavior of HPC systems and workloads in the recent past. Previous studies have attempted to characterize the I/O of workloads based on application-level traces [10,11,17,39], present experimental analysis of factors affecting I/O [35,56–58], and provide guidance for I/O storage systems [29,32–34,54,59]. However, there is limited understanding about how different files produced by HPC systems are re-accessed and re-used, from the same application and across applications. This is primarily because it is fundamentally challenging to measure and collect file-based I/O information across multiple executions as it requires tracing all executions of an application and the affected files which imposes high overhead and hence, is unsuitable for production HPC systems. The benefits of such a study are multi-fold, including understanding the nature of file-specific I/O, uncovering file reuse patterns, studying the effect of I/O variability on I/O performance, and optimizing file placement decisions. However, the costs of conducting such a study are prohibitively high for production systems [4,8,44]. This is one of the major reasons why the community has lacked such an understanding so far.

To the best of our knowledge, this is the first work to perform in-depth characterization and analysis of access, reuse, and sharing characteristics of I/O-intensive files. In particular, this is the first work to characterize (1) whether HPC files are ready-heavy, write-heavy, or both; (2) inter-arrival times for re-access and type of re-access across runs; (3) sharing of a file across multiple applications. Furthermore, our file-based I/O timing analysis also reveals key sources of inefficiencies that cause I/O variability within and across runs.

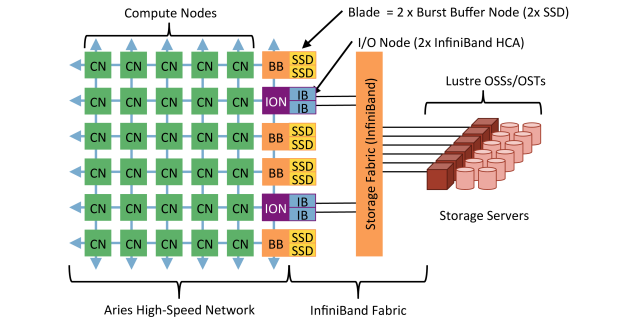


Figure 1: Architecture of the Cori supercomputer [7].

This study was carried out using a lightweight Darshan I/O monitoring tool to trace application I/O on Cori, a leading top 500 supercomputer, for a period spanning four months (Oct'17-Jan'18) during production - covering ≈36 million node-hours of operational system time.

Next, we briefly describe Cori and our methodology.

2 Background and Methodology

Brief Overview of the System. This study is based on a Cray XC40 supercomputer, Cori, ranked at #13 in the Top-500 supercomputers list. Cori achieves the peak computational performance of ≈27 Pflop/s. Cori contains 9,688 Intel Xeon Phi and 2,388 Intel Haswell processors. Fig. 1 shows Cori's network and storage structure. Cori features a disk-based Lustre file system which is composed of ≈10,000 disks organized as 248 Lustre Object Storage Targets (OST). Each OST is configured with GridRAID and has a corresponding Object Storage Server (OSS) for handling I/O requests. The total size of the file system is ≈30 PB with a peak I/O bandwidth of 744 GB/s. During the data collection period of this study, the file system was shared with Edison, an older Cray XC30 system which was near the end of its lifetime (retired in May'19). Edison was comparatively much smaller system (only 2 Pflop/s of peak performance) and generated much lesser I/O traffic compared to Cori as it was also near the end of its lifetime. As Edison was recently decommissioned, we only focus on Darshan logs collected on the Cori system. Cori also has a SSD-based Cray DataWarp burst-buffer storage layer. We note this study does not focus on burst-buffer I/O activities as they are limited (5-15%) and the shared file system observes almost all of the I/O traffic as per Darshan data.

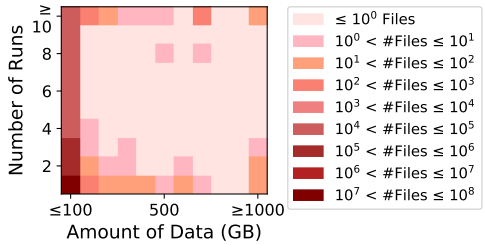
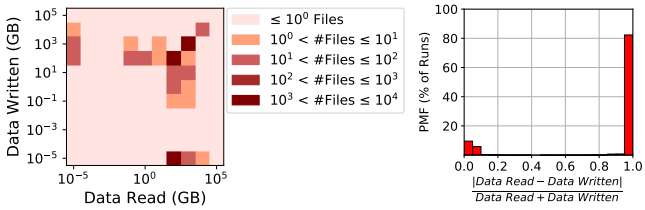


Figure 2: Over 99% of ≈52 million files transferred < 1 GB data and were accessed only once during the study period.



(a) (b)

Figure 3: (a) Files can be divided into groups: read-heavy (RH), write-heavy (WH), or both, read- and write- heavy (RW). (b) Difference in data read and written per run shows >82% of runs either perform only read I/O or only write I/O.

Data Collection. We use Darshan, a light-weight I/O monitoring tool which provides application-level I/O tracing capability [11] to collect file I/O access patterns. Darshan V3.10 was enabled by default on Cori for all users during the study period. Darshan reports key information including user id, job id, application (executable) id, start timestamp, end timestamp, and number of processes (ranks). Darshan also traces key statistical metrics for each file at the I/O-software-stack-level for different types of I/O interfaces including POSIX (Portable Operating System Interface) I/O, MPI (Message Passing Interface) I/O, and STD (Standard) I/O. These metrics include amount of read/write data, aggregate time for read/write/meta operations, rank id of I/O performing rank(s), and variance of I/O size and time among different application ranks. Lastly, Darshan also collects Lustre-file-system-level metrics such as stripe width and OST IDs over which a file is striped. However, Darshan does not report actual file sizes, only the size of the data transferred. Over the period of this study, ≈84 million logs (one per execution) were collected with information spanning ≈52 million unique files, 8489 applications, 651 users, and 12.8 PB of data transfer (6.9 PB read data and 5.9 PB write data).

Explanation of Analysis Figures. We now briefly describe the format of the analysis figures used for our study.

Heatmaps. These plots are used to show the significance of a specific relationship between two metrics. The intensity of a heatmap box color indicates the number of files which exhibit the corresponding relationship between the two metrics.

CMF Plots. We use CMF (Cumulative Mass Function) plots to show the cumulative distribution of a metric. A vertical dotted blue line is used to indicate the mean of the distribution. Some CMF plots show the distribution of the CoV (Coefficient of Variation $(\%) = \frac{\text{standard deviation}}{\text{mean}} \times 100$) of a metric to highlight the normalized variability observed by the metric.

Violin Plots. These plots are used to show the density (in terms of the number of files) for different values of a metric in a vertical format. A horizontal solid blue line is used to indicate the mean of the density distribution.

Next we describe how we select I/O-intensive files, classify these files, and classify the runs which access them.

2.1 Selecting I/O-Intensive Files

As mentioned previously, Cori's Darshan logs contain information about ≈52 million files. However, our analysis shows that a large majority of these files perform very little I/O during the study period. Fig. 2 shows a heatmap of the aggregate amount of data transferred to/from a file vs. the number of runs during which a file is accessed. Most of the files experience less than 100 GB of I/O during the study period and are accessed by only one run. In fact, over 99% of these files transfer less than 1 GB data. Note that this does not mean that the actual file size is less than 1 GB; but the data transfer to/from the file amounts to less than 1 GB.

Therefore, a majority of such files may not be helpful in establishing representative characteristics related to dominant I/O patterns of HPC applications. These files include user notes, scripts, executables, non-I/O-intensive-application outputs, and error logs. Therefore, *our study focuses on a class of "I/O-intensive" files which individually experience data transfer of at least 100 GB during the study period and are accessed by at least 2 runs - to capture the most dominant and representative I/O patterns. From here on, we refer to these I/O-intensive files as "files" simply*. This methodology streamlines our analysis to useful Darshan logs spanning ≈400k runs, 791 applications, 149 users, 8.5k files, and 7.8 PB of data transfer (4.7 PB read data and 3.1 PB write data). We ensured that our analysis is not skewed by only a handful of users performing most of the I/O to these files. In fact, over 70% of selected users perform I/O to more than 2 files, with each user performing I/O to 57 files on average.

2.2 File Classification

Next, we classify I/O-intensive files in terms of the type of I/O they perform. This helps us derive type-specific insights for different types of files in Sec. 3. We study the aggregate amount of read and write data transferred per file. Fig. 3(a) shows a heatmap of the amount of read data transfer vs. the amount of write data transfer. We observe that files can be classified into three distinct clusters. The lower right cluster consists of 22% of the files which transferred mostly read data during the four months. We refer to these as read-heavy or RH files. The upper left cluster consists of 7% of the files which transferred only write data (write-heavy or WH files). Lastly, the cluster in the top right corner with the largest percentage of files (71%), consists of files which are both, read- and write-heavy (referred to as RW files).

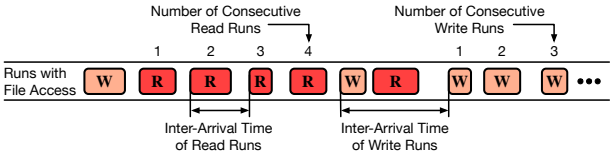


Figure 4: Visual representation of inter-arrival times and number of consecutive runs for both read (R) and write (W) runs.

Finding 1. *HPC files can be classified as read-heavy (RH), write-heavy (WH), or read- and write- heavy (RW). For the first time, we quantify that a significant fraction of the files are read-heavy (22%) and 7% of files are write-heavy - these 7% files are constantly written to but not read, which may indicate unread checkpoint/analysis data. 71% of HPC files are RW files (i.e., both read- and write- heavy). These files may include checkpoint/analysis files which do get read. Such a file classification can be used for file placement decisions in a multi-tier storage system including burst buffers, where each tier is suitable for different kind of I/O operations.*

2.3 Run Classification

While the files can be cleanly classified into three clusters, they can be accessed by multiple "application runs" (simply, referred to as "runs") and can perform both read and write I/O. A run refers to a job running on multiple compute nodes and consisting of multiple MPI processes/ranks and possibly shared-memory threads within a node. We found that a vast majority of runs perform either mostly-read or mostly-write I/O. To demonstrate this, we calculate the difference in the amount of read and write data for each run using the formula: $\frac{|\text{data read}-\text{data written}|}{\text{data read}+\text{data written}}$. The value of this formula ranges from 0 to 1: 1 indicates that all of the data transacted by the run is either exclusively read or exclusively write and 0 indicates equal amount of read and write data transfer. Fig. 3(b) shows that over 82% of all runs have a value very close to 1, i.e., they are either read-intensive or write-intensive. In the context of I/O, we refer to read-intensive runs as simply *"read runs"* and write-intensive runs as *"write runs"*. We found that 69% of all runs are read runs and 31% are write runs. RH files are mostly read by read runs, WH files are mostly written by write runs, and both read and writes runs operate on RW files. This classification helps us establish a producer-consumer relationship among runs in Sec. 3.1.

Finding 2. *Somewhat surprisingly, modern HPC applications largely tend to perform only one type of I/O during a single run: either read or write. This is in contrast to the commonly-held assumption that HPC applications have both read and write I/O phases during the same run [16, 20, 21, 28, 36, 46, 49, 60]. This finding indicates the potential rise of scientific workflows instead of traditional monolithic scientific applications [6, 40, 45]. The presence of non-monolithic applications provides the opportunity to better schedule different components of a large workflow to avoid I/O contention among different workflows.*

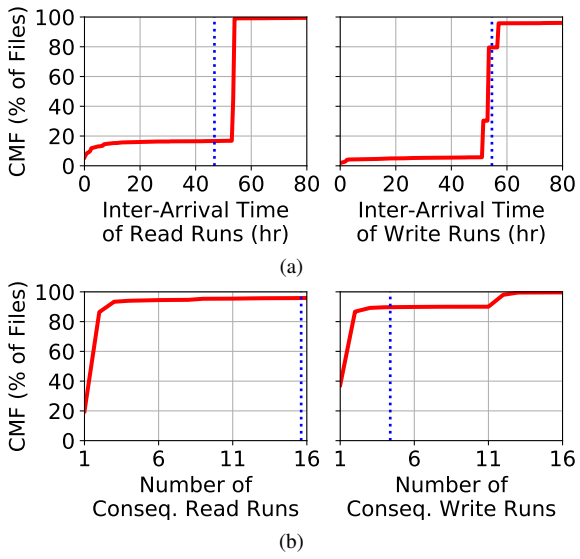


Figure 5: (a) Most of the read and write runs have inter-arrival times of 50-55 hours per file (file re-access interval). (b) The mean number of consecutive read runs in 13 and the mean number of consecutive write runs is 3.

3 Result Discussion and Analysis

In this section, we explore HPC file behavior concerning multi-run reuse and multi-application sharing (Sec. 3.1), and we study I/O data characteristics pertaining to load imbalance and intra- and inter- run I/O variability (Sec. 3.2).

3.1 File Reuse Characteristics

Run Inter-Arrival Times. In Sec. 2.3, we showed that a run can be classified as either read run or write run, and found that the total number of read runs are more than 2x the number of write runs. Now, we study the inter-arrival times of these different runs to understand the avgerage time taken to reuse the same file (inter-arrival time is defined as shown in Fig. 4). Fig. 5(a) shows that the mean inter-arrival time of read runs experienced by a file is 47 hours, while that of write runs is 55 hours. But, on an average, 80% of files are re-accessed only after 50-55 hours for both read and write runs. We note that the average inter-arrival time is much longer than the average runtime of jobs on Cori (e.g., >80% of HPC jobs on these systems finish in less than 2 hours) [3, 43].

Finding 3. *Read and write runs have similar inter-arrival times of over 2 days for 80% of the files. For the first time we find that most files get re-accessed after a relatively long period (>50 hours) - much longer than the runtime of jobs. This enables opportunity for data compression [18] of files which are expected to remain inactive for some time and also leverage transparent burst-buffer prefetching and caching [9, 47] for files expected to be accessed in a short while.*

Consecutive Runs of the Same I/O Type. Read and write runs having similar inter-arrival times motivates us to test

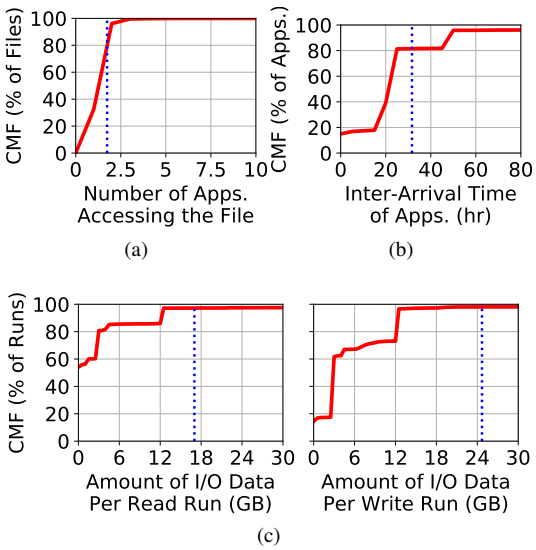


Figure 6: (a) Over 65% of files are accessed by at least 2 applications. (b) The average inter-arrival time of each application to perform I/O to a specific file is 31 hours. (c) On average, read runs transfer 17 GB of data per run, while write runs transfer 25 GB of data per run.

if read and write runs are scheduled back to back, and if so, how long do these sequences last. We calculate the average number of consecutive read runs and write runs for each file (as shown in Fig. 4) and plot the distribution in Fig. 5(b). Over 80% of files experience 2 or more consecutive read runs and over 65% of files experience 2 or more consecutive write runs. A majority of files experience 2 consecutive read runs (65%) and 2 consecutive write runs (50%). This suggests that files get accessed in alternating phases of multiple read runs and multiple write runs - consistent with our observation that RW files dominate the population (71%). However, there are many files which experience a large number of consecutive read runs (due to RH files). In fact, the mean number of consecutive read runs experienced by a file is over 14, while the mean number of consecutive write runs is < 4. There are only 2.2× as many read runs as write runs (Sec. 2.3), but mean number of consecutive read runs is 4.3× the number consecutive write runs. This indicates that data is produced a few times, and then consumed many times over, true for most RW files. This observation suggests that scientific simulations often produce data during certain runs, which is then used as a driver input by several subsequent runs to explore different potential paths or analyze a simulated phenomena in detail. We note that consecutive write runs does not imply that all the previously written data is rewritten/lost. Some scientific workflows could append a file over two consecutive write runs and then, read a part of the file in the subsequent run.

Finding 4. *HPC files experience a few consecutive write runs and a long string of consecutive read runs on average. This insight can help leverage MPI "hints" [38] to guide the system about the type of I/O about to be executed. Partitioning of I/O servers [25] to separately serve RH files (which perform many consecutive reads) and RW files (for read and write runs) can boost I/O performance.*

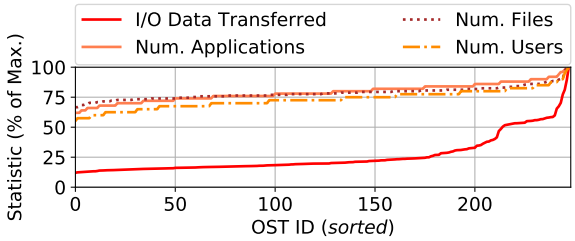


Figure 7: The amount of I/O data transferred by each OST is largely unequal, even though the number of files, applications, and users are more balanced due to capacity balancing.

Multi-Application File Sharing. Taking the producer-consumer relationship one step further, it would be interesting to understand if the producer and the consumer are the same application or if they are different applications. From a methodological point of view, we note that all applications which access a file are run by the same user. So for any file, both producer and consumer applications belong to the same user. Also, a file is not considered to be shared by default among multiple users due to permission issues. Fig. 6(a) shows the CMF of the number of applications which access a file. Over 67% of files are accessed by at least 2 applications, thus indicating that files are often shared by multiple applications. Fig. 6(b) shows the CMF of the inter-arrival time of each application which performs I/O to a file. The mean inter-arrival time of each application is 31 hours, which is much lower than the mean inter-arrival time of individual read and write runs (>50 hours). Thus, for most files, 2 or more applications serve as the producer and the consumer, as opposed to a single application performing I/O to the file. This is consistent with our finding that a majority (86%) of files accessed by multiple applications are RW files (only 12% of these shared files are RH files and only 2% are WH files).
Finding 5. *HPC files are shared by multiple applications and each application performs both read and write I/O serving as both, the producer and the consumer. Inter-arrival times of these runs also indicate that the producer and the consumer are launched significantly apart in time - limiting the effectiveness of potential caching across applications.*

3.2 Characteristics of I/O Data Accesses

Per Run I/O Data Transfer. In Sec. 3.1, we studied how files get used over multiple runs. We now investigate how the data transaction characteristics change over these multiple runs. Fig. 6(c) shows a CMF of the amount of data transferred per run by read runs and write runs. We observe that on average, read runs transfer 17 GB of data per run, while write runs transfer 25 GB of data per run. In fact, 50% of read runs transfer less than 1 GB of data.

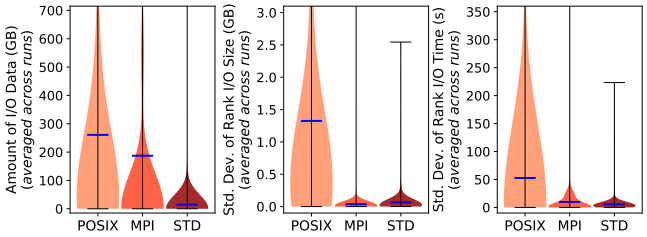


Figure 8: (a) POSIX and MPI I/O interfaces are used to transfer the most amount of data. (b) The variability in I/O size among different ranks of the same application is very small. (c) But, the variability in I/O time of individual ranks is large.

Finding 6. *While reads runs are more abundant than write runs and transfer more data in total, surprisingly, write runs transfer more amount of data than read runs per run. On average, write runs perform 1.4× the I/O of read runs per run. This finding can be exploited to manage I/O contention better at the system-level by limiting the number of concurrently executing write runs. Recall that our earlier finding indicates that HPC applications largely tend to perform only one type of I/O during one run and hence, "write runs" can easily be detected and classified.*

Spatial Load Imbalance. Now that we have found that different runs transfer different amount of data, the next question to investigate is how this difference affects the back-end OSTs. Fig. 7 shows the normalized I/O data transferred to/from each of the OSTs during the study period. Interestingly, there is a large spread in how much data is transferred by each OST. The least "active" OST is only 13% as active as the most active OST. On the other hand, when we look at the number of files on each OST, number of applications which use these files, and number of users which generate the files, we see that the spread is much lower.

Finding 7. *For the Lustre-based system studied in this work, OSTs are capacity-balanced to ensure approximately equal utilization at the file creation time, but that does not guarantee dynamic load-balance. Consequently, there is large inequality in terms of the amount of load (data transfer) which each OST observes over time - emphasizing the need for dynamic file migration (currently lacking in the Lustre file system), replication of read-only data, and caching.*

Intra-Run I/O Variability. Next, we look at how varying OST contention can affect the I/O time of concurrently running ranks (processes) within a run as these ranks could be performing I/O to different OSTs in parallel. For this analysis, we individually analyze the three different I/O interfaces used at Cori: POSIX I/O, MPI I/O, and STD I/O. First, we look at the amount of data transferred using each interface. Fig. 8(a) shows that POSIX is the most commonly used I/O interface transferring about 260 GB of data per run per file on average. Thereafter, MPI interface is used to transfer about 190 GB of

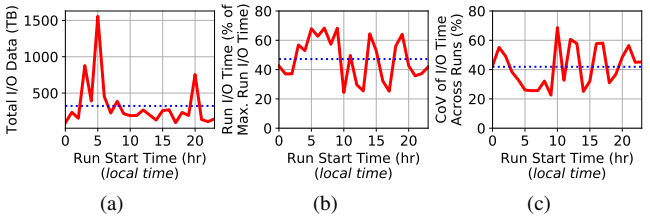


Figure 9: (a) A large amount of I/O data is transferred during 3am-5am local time. (b) Due to this, runs take the most amount of time to complete their I/O during the corresponding hours. (c) Also, variability in I/O time is lower when I/O time is higher and higher when I/O time is lower.

data per run per file on average. STD is the least commonly used interface, as is expected for parallel HPC applications.

Fig. 8(b) shows the standard deviation of the amount of data transferred across each rank performing I/O per run per file. On average, this standard deviation is very small across all three interfaces. For example, the average standard deviation of the amount of data transferred across POSIX I/O performing ranks is less than 1.5 GB, which is negligible compared to the average amount of data transferred using POSIX (260 GB). On the other hand, Fig. 8(c) shows the standard deviation of the I/O time across each rank performing I/O per run per file. This standard deviation is especially high for I/O performed using POSIX interface. This is because, typically when using the POSIX interface, each rank performs I/O to its own file, while when using the MPI I/O interface, all ranks perform I/O to a shared file. Because the default stripe width on the Cori supercomputer is 1, over 99% files are striped across only 1 OST. Therefore, if an application performs I/O to multiple files in parallel, they tend to perform I/O to multiple OSTs in parallel, as the files could be mapped to different OSTs. Thus, varying levels of resource contention at these OSTs can dramatically affect the I/O time of the individual ranks when using POSIX I/O.

Finding 8. *OST load imbalance leads to a high degree of variability in I/O time of ranks which are concurrently performing I/O, especially if the ranks are performing I/O to different OSTs, which is largely the case with POSIX I/O. This leads to the faster ranks having to wait for the slower ranks to finish I/O before they can resume computation, thus wasting precious compute cycles on the HPC system.*

Temporal Load Imbalance. Previously, we discovered that OST I/O imbalance and contention causes intra-run variability in I/O time. So the next step is to explore the temporal characteristics of I/O load. Fig. 9(a) shows the total amount of data which is transferred at different hours of the day. We observe that the largest amount of I/O activity is performed by runs which start between 3am and 5am local time. Note that Cori has users across the globe, so the specific local time (i.e., early morning) is not an indicator of when the local users are the most active. We plot the amount of data with respect to the

start time of the run which is sufficient for our analysis. We note that our following analysis does not necessarily establish a causal relationship between different factors, but instead attempts to explain the observed trends. In Fig. 9(b), we plot the I/O time of runs across different hours of the day. The I/O time of a run is plotted as percentage of the maximum I/O time among all runs which perform I/O to the same file to normalize it across files. However, we observe that runs started during 3am-5am and a few hours post 5am have the highest runtime due to the high I/O activity during this time. This is in spite of the fact that runs performing I/O to the same file have low variability in terms of the amount of data they transfer (as we will discuss later).

Interestingly, Fig. 9(c) shows that while the variability in I/O time is generally significantly high across all times (>20%), it is the lowest for runs which start during peak I/O activity periods. The CoV is calculated among runs belonging to the same file which start during the same hour of the day. The CoV of I/O time plot has a near opposite trend as that of the I/O time plot (Fig. 9(b)). In fact, the I/O time and CoV of I/O time have a Spearman Correlation Index of -0.94, which points to strong negative correlation. This indicates that when the I/O activity is highest, the variability in I/O time that the user can expect is slightly lower, i.e., if user A starts the same run every day during a high I/O activity period, they can expect less variability in the runs' I/O times (and therefore, runtimes) than user B who starts the same run every day during a low I/O activity period. Of course, the trade-off is that user A observes a higher I/O time on average than user B. This happens because when the I/O activity is high, the OSTs are heavily contended which may slow down all I/O. Hence, the effect of any variation in I/O time is small. However, when OSTs are not contended and I/O is faster, the effects of variation are more pronounced and noticeable.

Finding 9. *Temporal load imbalance causes I/O time of the same run to be different during different times of the day. Moreover, variability in I/O time is strongly negatively correlated with the I/O time during the time of the day. HPC systems need new techniques to mitigate the intra-run variability (i.e., ranks of the same application finishing at different times) which continues to have a considerable presence since the I/O variability is significant at all times (>20%).*

Inter-Run I/O Variability. The next question we address is that if there is temporal imbalance in storage system load, does it cause I/O time variability from one run to another? Note that the variability we addressed in Finding 9 was among runs starting during the same hour. Now we look at all runs accessing the same file regardless of their start times. First, we explore how much the amount of data transferred to/from the same file changes from one run to another. Fig. 10(a) shows the CMF of the CoV of the amount of I/O data transferred across runs for each file. Overall, more than 80% of files have a CoV of less than 5% which indicates a negligible

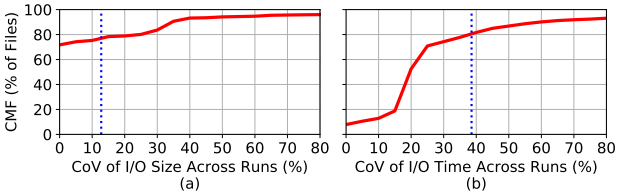


Figure 10: The change in the amount of data transferred across runs to read-only files is the smallest, but these files experience the highest variability across runs in terms of I/O time. Overall, the change in the amount of data is very small (mean CoV is 12%); however, the change in the amount of time it takes to transfer the data is much greater (mean CoV is 39%).

change in the amount of I/O data transferred from one run to another. This is especially true for RH files, and even true for RW files, which experience both, read runs and write runs, thus indicating that similar amount of data gets produced and consumed in a vast majority of cases. WH files exhibit the highest variability in the amount of data transferred (mostly write data in the case of WH files) with a mean CoV of 35% (results for different types of files are not shown for brevity).

Fig. 10(b) shows the CoV of I/O time for different runs for each file. Across all files, even though the amount of data does not change significantly from one run to another, the amount of time it takes to transfer this data experiences significant variability: the mean CoV of the I/O time across runs is 39%. RH files experience the most change in I/O time from one run to another with a mean CoV of 68%, even though they have the least change in the amount of data transferred. This is due to the fact that the OSTs experience different levels of contention at different times due to temporal load imbalance. In fact, because read runs transfer less amount of data on average than write runs (as we discussed in Finding 6), the effect of this load imbalance is especially prominent on their I/O time, which in turn has the largest impact on RH files.

Finding 10. *HPC files tend to experience similar amount of data transfer from one run to another, but they do experience a large variability in terms of the amount of time taken to transfer the data. This is especially true for ready-heavy files which have the least variability in I/O data, but the most variability in I/O time - indicating the need for special attention to RH files when mitigating I/O variability challenge.*

4 Scope of the Findings

While we have ensured that our results and insights are statistically significant, certain aspects of our study may limit the applicability and generalization ability of our analysis.

User Opt-Out. Cori users had the option to opt out of Darshan logging. However, the Darshan library is enabled by default for all users. Therefore, a large majority of users, especially the ones running I/O intensive applications, run Darshan during execution to understand their I/O behavior.

Time Period of Data Collection. Our study uses four months of data logs for analysis and is unable to detect trends longer

than four months. However, four months is a long period and all of the insightful findings such as read and write runs inter-arrival times, multiple application inter-arrival times, and temporal load imbalance are in the order of hours. We also note that the jobs on the Cori supercomputer do not exhibit significant seasonal behavior. That is, the I/O traffic remains relatively similar throughout the year, as also confirmed by previous studies [42]. Therefore, we do not expect our analysis and findings to be affected by the time period of the study.

Unavailable Information. Our study is restricted by the type of information traced by Darshan. Therefore, we are unable to study file size, file amendments/overwrites, number of nodes involved in I/O, and batch job I/O behavior. Information about random vs. sequential I/O type is available for POSIX I/O, but does not yield interesting results as we found that almost all of the I/O is sequential as is expected for HPC applications.

"What if?" Analysis. Our post-event analysis also bars us from posing "what if" questions such as what if a particular run is removed from analysis? How would it affect the I/O trends? Such questions are not possible to study retroactively in a parallel storage system as all concurrently running applications affect each other's I/O behavior in complex ways which cannot be decoupled easily.

Impact of Cori-specific environment and workloads. As expected, our findings are influenced by the nature of workloads which are executed at National Energy Research Scientific Computing Center (NERSC) and the NERSC system environment where Cori is hosted. Consequently, we caution that our findings cannot be generalized to other HPC systems as-is, but this work provides a methodological framework to conduct a study of this nature at other centers to confirm and refute the presented findings.

However, we also note that similarities between NERSC and other centers are likely since HPC users often tend to run workloads with similar characteristics [34]. Workloads running at NERSC are diverse in nature and correspond to a wide variety of scientific domains such as material science, cosmology, combustion, fluid dynamics, climate science, and quantum simulations. Prior studies have covered various aspects of these workloads [5, 31, 34, 53].

Increase in data analytics workloads may be the reason for read-heavy file I/O. Wide increase of such workloads on leading HPC centers has been observed in recent years [1, 12, 13]. NERSC has observed a rise in data analytics workloads in NERSC Exascale Science Applications Program (NESAP) [14]. Data and learning applications such as BD-CATS which run at NERSC are quite I/O-intensive. Interestingly, we also observed that some applications that generate large amounts of read data (QCD and quantum modeling of materials) do not necessarily come from the data analytics domain and have run at NERSC for many years. Finally, we note that the scope of this study is limited to only the NERSC system where the instrumentation was performed.

5 Related Work

In this section, we discuss and contrast some related work.

I/O Characterization Software. As HPC I/O has become more unstable and a bigger performance bottleneck over the last few years, much effort is geared toward developing I/O characterization tools for individual applications [10, 11, 22, 50, 55] and for the entire system [2, 23, 24, 51, 58]. Recent works focus on developing software for end-to-end characterization of I/O [15, 30, 31, 41, 48, 58]. These works deal with tool development and do not provide detailed analysis of I/O behavior, especially in terms of file access and reuse.

I/O Behavior Analysis. Most analysis works study the I/O behavior of individual applications and/or runs such as I/O periodicity, bandwidth characteristics, and inter/intra application execution I/O variability [19, 26, 29, 32–34, 37, 54, 56, 57, 59]. Variability and I/O characterization studies performed by some of these previous works are restricted to analyzing a few benchmarks as they do not have access to a system-level view of hundreds of concurrently running HPC applications. Apart from analyzing application-level I/O logs, works by Liu et al. and Madireddy et al. [27, 28, 35] also examine storage server logs to assess application I/O characteristics. In fact, many studies focus extensively on the storage system's I/O behavior [17, 21, 39, 51, 52] by exploring optimal file-system configurations or identifying system-level topology bottlenecks. Above works do not consider multi-fold interactions related to HPC files such as file re-access, multi-application file sharing, run classification and inter-arrival, spatial- and temporal- load imbalance, and intra- and inter- run variability.

6 Conclusion

Overall, our analysis of Darshan I/O logs on the Cori supercomputer reveals many previously unexplored and unexpected insights. We found that files which contribute the most to HPC I/O are not only re-accessed in more ways than one but are also shared across applications. They follow a producer-consumer relationship with runs which extensively write to the files and runs which extensively read them. We explored why and how these files have large intra- and inter-run variability not in terms of I/O size, but in terms of I/O time.

Acknowledgement. We are thankful to our shepherd, Avani Wildani, and anonymous reviewers for their constructive feedback. This work is supported in part by NSF Awards 1910601 and 1753840, Northeastern University, and Massachusetts Green High Performance Computing Center (MGHPCC). It is also supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under contract numbers DE-AC02-05CH11231 and DE-AC02-06CH11357. This work also used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy.

References

[1] NERSC 2017 Annual Report. https://www.nersc.gov/assets/Uploads/2017NERSC-AnnualReport.pdf, 2017.

[2] Anthony Agelastos, Benjamin Allan, Jim Brandt, Paul Cassella, Jeremy Enos, Joshi Fullop, Ann Gentile, Steve Monk, Nichamon Naksinehaboon, Jeff Ogden, et al. The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications. In *SC'14*, pages 154–165. IEEE, 2014.

[3] Gonzalo Pedro Rodrigo Alvarez, Per-Olov Östberg, Erik Elmroth, Katie Antypas, Richard Gerber, and Lavanya Ramakrishnan. Towards Understanding Job Heterogeneity in HPC: A NERSC Case Study. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 521–526. IEEE, 2016.

[4] George Amvrosiadis, Ali R Butt, Vasily Tarasov, Erez Zadok, and Ming Zhao. Data Storage Research Vision 2025 Report. *Technical Report*, 2019.

[5] Brian Austin, Tina Butler, Richard Gerber, Cary Whitney, Nicholas Wright, Woo-Sun Yang, and Zhengji Zhao. Hopper Workload Analysis. Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), 2014.

[6] Fayssal Benkhaldoun, Christophe Cérin, Imad Kissami, and Walid Saad. Challenges of Translating HPC Codes to Workflows for Heterogeneous and Dynamic Environments. In *2017 International Conference on High Performance Computing & Simulation (HPCS)*, pages 858–863. IEEE, 2017.

[7] Wahid Bhimji, Deborah Bard, David Paul, Melissa Romanus, et al. Accelerating science with the NERSC Burst Buffer Early User Program. In *Cray User Group (CUG)*, 2016.

[8] A Brinkmann, K Mohror, and W Yu. Challenges and Opportunities of User-Level File Systemsfor HPC. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2017.

[9] Surendra Byna, Yong Chen, Xian-He Sun, Rajeev Thakur, and William Gropp. Parallel I/O Prefetching using MPI File Caching and I/O Signatures. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 44. IEEE Press, 2008.

[10] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. Understanding and Improving Computational Science Storage Access through Continuous Characterization. *ACM Transactions on Storage (TOS)*, 7(3):8, 2011.

[11] Philip Carns, Robert Latham, Robert Ross, Kamil Iskra, Samuel Lang, and Katherine Riley. 24/7 Characterization of Petascale I/O Workloads. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10. IEEE, 2009.

[12] Steven WD Chien, Stefano Markidis, Vyacheslav Olshevsky, Yaroslav Bulatov, Erwin Laure, and Jeffrey S Vetter. TensorFlow Doing HPC. *arXiv preprint arXiv:1903.04364*, 2019.

[13] Steven WD Chien, Stefano Markidis, Chaitanya Prasad Sishtla, Luis Santos, Pawel Herman, Sai Narasimhamurthy, and Erwin Laure. Characterizing Deep-Learning I/O Workloads in TensorFlow. In *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS)*, pages 54–63. IEEE, 2018.

[14] Jack Deslippe, Doug Doerfler, Brian Friesen, Yun Helen He, Tuomas Koskela, Mathieu Lobet, Tareq Malas, Leonid Oliker, Andrey Ovsyannikov, Samuel Williams, et al. Analyzing Performance of Selected NESAP Applications on the Cori HPC System. In *High Performance Computing: ISC High Performance 2017 International Workshops, DRBSD, ExaComm, HCPM, HPC-IODC, IWOPH, IXPUG, P^ 3MA, VHPC, Visualization at Scale, WOPSSS, Frankfurt, Germany, June 18-22, 2017, Revised Selected Papers*, volume 10524, page 334. Springer, 2017.

[15] Sheng Di, Rinku Gupta, Marc Snir, Eric Pershey, and Franck Cappello. Logaider: A Tool for Mining Potential Correlations of HPC Log Events.

[16] Hassan Eslami, Anthony Kougkas, Maria Kotsifakou, Theodoros Kasampalis, Kun Feng, Yin Lu, William Gropp, Xian-He Sun, Yong Chen, and Rajeev Thakur. Efficient disk-to-disk sorting: A case study in the decoupled execution paradigm. In *Proceedings of the 2015 International Workshop on Data-Intensive Scalable Computing Systems*, page 2. ACM, 2015.

[17] Raghul Gunasekaran, Sarp Oral, Jason Hill, Ross Miller, Feiyi Wang, and Dustin Leverman. Comparative I/O Workload Characterization of Two Leadership Class Storage Clusters. In *Proceedings of the 10th Parallel Data Storage Workshop*, pages 31–36. ACM, 2015.

[18] Jun He, John Bent, Aaron Torres, Gary Grider, Garth Gibson, Carlos Maltzahn, and Xian-He Sun. I/O Acceleration with Pattern Detection. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 25–36. ACM, 2013.

[19] Dan Huang, Qing Liu, Jong Choi, Norbert Podhorszki, Scott Klasky, Jeremy Logan, George Ostrouchov, Xubin He, and Matthew Wolf. Can I/O Variability Be Reduced on QoS-Less HPC Storage Systems? *IEEE Transactions on Computers*, 68(5):631–645, 2018.

[20] Ye Jin, Xiaosong Ma, Mingliang Liu, Qing Liu, Jeremy Logan, Norbert Podhorszki, Jong Youl Choi, and Scott Klasky. Combining Phase Identification and Statistic Modeling for Automated Parallel Benchmark Generation. *ACM SIGMETRICS Performance Evaluation Review*, 43(1):309–320, 2015.

[21] Youngjae Kim and Raghul Gunasekaran. Understanding I/O Workload Characteristics of a Peta-scale Storage System. *The Journal of Supercomputing*, 71(3):761–780, 2015.

[22] Michelle Koo, Wucherl Yoo, and Alex Sim. I/O Performance Analysis Framework on Measurement Data from Scientific Clusters. 2015.

[23] Julian M Kunkel, Michaela Zimmer, Nathanael Hübbe, Alvaro Aguilera, Holger Mickler, Xuan Wang, Andriy Chut, Thomas Bönisch, Jakob Lüttgau, Roman Michel, et al. The SIOX Architecture–Coupling Automatic Monitoring and Optimization of Parallel I/O. In *International Supercomputing Conference*, pages 245–260. Springer, 2014.

[24] Julian Martin Kunkel, Eugen Betke, Matt Bryson, Philip Carns, Rosemary Francis, Wolfgang Frings, Roland Laifer, and Sandra Mendez. Tools for Analyzing Parallel I/O. In *International Conference on High Performance Computing*, pages 49–70. Springer, 2018.

[25] Chih-Song Kuo, Aamer Shah, Akihiro Nomura, Satoshi Matsuoka, and Felix Wolf. How File Access Patterns Influence Interference Among Cluster Applications. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 185–193. IEEE, 2014.

[26] Qing Liu, Norbert Podhorszki, Jeremy Logan, and Scott Klasky. Runtime I/O Re-Routing+ Throttling on {HPC} Storage. In *Presented as part of the 5th {USENIX} Workshop on Hot Topics in Storage and File Systems*, 2013.

[27] Yang Liu, Raghul Gunasekaran, Xiaosong Ma, and Sudharshan S Vazhkudai. Automatic Identification of Application I/O Signatures from Noisy Server-Side Traces. In *FAST*, volume 14, pages 213–228, 2014.

[28] Yang Liu, Raghul Gunasekaran, Xiaosong Ma, and Sudharshan S Vazhkudai. Server-Side Log Data Analytics for I/O Workload Characterization and Coordination on Large Shared Storage Systems. In *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*, pages 819–829. IEEE, 2016.

[29] Glenn K Lockwood, Shane Snyder, Teng Wang, Suren Byna, Philip Carns, and Nicholas J Wright. A Year in the Life of a Parallel File System. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, page 74. IEEE Press, 2018.

[30] Glenn K Lockwood, Nicholas J Wright, Shane Snyder, Philip Carns, George Brown, and Kevin Harms. TOKIO on ClusterStor: Connecting Standard Tools to Enable Holistic I/O Performance Analysis. 2018.

[31] Glenn K Lockwood, Wucherl Yoo, Suren Byna, Nicholas J Wright, Shane Snyder, Kevin Harms, Zachary Nault, and Philip Carns. UMAMI: A Recipe for Generating Meaningful Metrics through Holistic I/O Performance Analysis. In *Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems*, pages 55–60. ACM, 2017.

[32] Uri Lublin and Dror G Feitelson. The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105–1122, 2003.

[33] Jakob Lüttgau, Shane Snyder, Philip Carns, Justin M Wozniak, Julian Kunkel, and Thomas Ludwig. Toward Understanding I/O Behavior in HPC Workflows. In *Proc. of Workshop in conjunction with ACM/IEEE Supercomputing Conference, Dallas, TX, USA*, 2018.

[34] Huong Luu, Marianne Winslett, William Gropp, Robert Ross, Philip Carns, Kevin Harms, Mr Prabhat, Suren Byna, and Yushu 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. ACM, 2015.

[35] Sandeep Madireddy, Prasanna Balaprakash, Philip Carns, Robert Latham, Robert Ross, Shane Snyder, and Stefan M Wild. Analysis and Correlation of Application I/O Performance and System-Wide I/O Activity. In *Networking, Architecture, and Storage (NAS), 2017 International Conference on*, pages 1–10. IEEE, 2017.

[36] Anirban Mandal, Paul Ruth, Ilya Baldin, Yufeng Xin, Claris Castillo, Mats Rynge, and Ewa Deelman. Evaluating i/o aware network management for scientific workflows on networked clouds. In *Proceedings of the Third International Workshop on Network-Aware Data Management*, page 2. ACM, 2013.

[37] Aleksander Maricq, Dmitry Duplyakin, Ivo Jimenez, Carlos Maltzahn, Ryan Stutsman, and Robert Ricci. Taming performance variability. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 409–425, 2018.

[38] John M May. *Parallel I/O for High Performance Computing*. Morgan Kaufmann, 2001.

[39] S. Oral et al. Best Practices and Lessons Learned from Deploying and Operating Large-Scale Data-Centric Parallel File Systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 217–228. IEEE, 2014.

[40] Suraj Pandey, Karan Vahi, Rafael Ferreira da Silva, Ewa Deelman, Ming Jiang, Cyrus Harrison, Al Chu, and Henri Casanova. Event-Based Triggering and Management of Scientific Workflow Ensembles. In *HPC Asia*, 2018.

[41] Byung H Park, Saurabh Hukerikar, Ryan Adamson, and Christian Engelmann. Big Data Meets HPC Log Analytics: Scalable Approach to Understanding Systems at Extreme Scale. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 758–765. IEEE, 2017.

[42] Tirthak Patel, Suren Byna, Glenn K Lockwood, and Devesh 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*, page 65. ACM, 2019.

[43] Gonzalo P Rodrigo, P-O Östberg, Erik Elmroth, Katie Antypas, Richard Gerber, and Lavanya Ramakrishnan. Towards Understanding HPC Users and Systems: A NERSC Case Study. *Journal of Parallel and Distributed Computing*, 111:206–221, 2018.

[44] Robert Ross, Lee Ward, Philip Carns, Gary Grider, Scott Klasky, Quincey Koziol, Glenn K Lockwood, Kathryn Mohror, Bradley Settlemyer, and Matthew Wolf. Storage Systems and I/O: Organizing, Storing, and Accessing Data for Scientific Discovery. Technical report, USDOE Office of Science (SC)(United States), 2019.

[45] Mats Rynge, Scott Callaghan, Ewa Deelman, Gideon Juve, Gaurang Mehta, Karan Vahi, and Philip J Maechling. Enabling Large-Scale Scientific Workflows on Petascale Resources using MPI Master/Worker. In *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the eXtreme to the campus and beyond*, page 49. ACM, 2012.

[46] Peter Scheuermann, Gerhard Weikum, and Peter Zabback. Adaptive load balancing in disk arrays. In *International Conference on Foundations of Data Organization and Algorithms*, pages 345–360. Springer, 1993.

[47] Seetharami Seelam, I-Hsin Chung, John Bauer, and Hui-Fang Wen. Masking I/O Latency using Application Level I/O Caching and Prefetching on Blue Gene systems. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–12. IEEE, 2010.

[48] Benjamin H Sigelman, Luiz Andre Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. Dapper, a Large-Scale Distributed Systems Tracing Infrastructure. 2010.

[49] Evgenia Smirni and Daniel A. Reed. Lessons From Characterizing the Input/Output Behavior of Parallel Scientific Applications. *Performance Evaluation*, 33(1):27–44, 1998.

[50] Shane Snyder, Philip Carns, Kevin Harms, Robert Ross, Glenn K Lockwood, and Nicholas J Wright. Modular HPC I/O Characterization with Darshan. In *2016 5th Workshop on Extreme-Scale Programming Tools (ESPT)*, pages 9–17. IEEE, 2016.

[51] Sudharshan S Vazhkudai, Ross Miller, Devesh Tiwari, Christopher Zimmer, Feiyi Wang, Sarp Oral, Raghul Gunasekaran, and Deryl Steinert. GUIDE: A Scalable Information Directory Service to Collect, Federate, and Analyze Logs for Operational Insights into a Leadership HPC Facility. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 45. ACM, 2017.

[52] Feiyi Wang, Sarp Oral, Saurabh Gupta, Devesh Tiwari, and Sudharshan S Vazhkudai. Improving Large-Scale Storage System Performance via Topology-Aware and Balanced Data Placement. In *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 656–663. IEEE, 2014.

[53] Teng Wang, Suren Byna, Glenn K Lockwood, Shane Snyder, Philip Carns, Sunggon Kim, and Nicholas 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, 2019.

[54] Teng Wang, Shane Snyder, Glenn Lockwood, Philip Carns, Nicholas Wright, and Suren Byna. IOMiner: Large-Scale Analytics Framework for Gaining Knowledge from I/O Logs. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 466–476. IEEE, 2018.

[55] Steven A Wright, Simon D Hammond, Simon J Pennycook, Robert F Bird, JA Herdman, Ian Miller, A Vadgama, Abhir Bhalerao, and Stephen A Jarvis. Parallel File System Analysis through Application I/O Tracing. *The Computer Journal*, 56(2):141–155, 2012.

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

[57] Bing Xie, Yezhou Huang, Jeffrey S Chase, Jong Youl Choi, Scott Klasky, Jay Lofstead, and Sarp Oral. Predicting Output Performance of a Petascale Supercomputer. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, pages 181–192. ACM, 2017.

[58] Bin Yang, Xu Ji, Xiaosong Ma, Xiyang Wang, Tianyu Zhang, Xiupeng Zhu, Nosayba El-Sayed, Haidong Lan, Yibo Yang, Jidong Zhai, et al. 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, 2019.

[59] Orcun Yildiz, Matthieu Dorier, Shadi Ibrahim, Rob Ross, and Gabriel Antoniu. On the Root Causes of Cross-Application I/O Interference in HPC Storage Systems. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 750–759. IEEE, 2016.

[60] Zhou Zhou, Xu Yang, Dongfang Zhao, Paul Rich, Wei Tang, Jia Wang, and Zhiling Lan. I/O-Aware Batch Scheduling for Petascale Computing Systems. In *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*, pages 254–263. IEEE, 2015.