2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)

A Zoom-in Analysis of I/O Logs to Detect Root Causes of I/O Performance Bottlenecks

Teng Wang[1], Suren Byna[1], Glenn K. Lockwood[1], Shane Snyder[2], Philip Carns[2], Sunggon Kim[3], Nicholas J. Wright[1],

[1] Lawrence Berkeley National Laboratory, [2] Argonne National Laboratory, [3] Seoul National University
Email: {tengwang, sbyna, glock, njwright}@lbl.gov, {ssnyder, carns}@mcs.anl.gov, skim@dcslab.snu.ac.kr

***Abstract*—Scientific applications frequently spend a large fraction of their execution time in reading and writing data on parallel file systems. Identifying these I/O performance bottlenecks and attributing root causes are critical steps toward devising optimization strategies. Several existing studies analyze I/O logs of a set of benchmarks or applications that were run with controlled behaviors. However, there is still a lack of general approach that systematically identifies I/O performance bottlenecks for applications running "in the wild" on production systems. In this study, we have developed an analysis approach of "zooming in" from platform-wide to application-wide to job-level I/O logs for identifying I/O bottlenecks in arbitrary scientific applications. We analyze the logs collected on a Cray XC40 system in production over a two-month period. This study results in several insights for application developers to use in optimizing I/O behavior.**

I. Introduction

Parallel I/O remains a critical tool for reading and writing large volumes of data produced by scientific simulations, experiments, and observations. Simulation codes such as fusion [1], climate [2], and cosmology [3] produce tens to hundreds of terabytes of data. Similarly, experimental and observational instruments such as light sources [4] and astronomy observations [5] produce terabytes of data per day. Efficient I/O on parallel file systems of high-performance computing (HPC) systems is critical for accelerating science.

Identifying and attributing root causes of poor I/O performance are the crucial steps in optimizing I/O performance. Several monitoring tools have been developed toward this goal. Among them, Darshan [6], which has been deployed on several HPC systems, characterizes application-level performance [6]. File system tools such as the Lustre Monitoring Toolkit (LMT) [7] for Lustre and *ggiostat* [8] for GPFS characterize system-level performance.

Several analyses of I/O logs have identified system-level factors such as I/O traffic and metadata server load as contributing to poor I/O performance. For example, Yildiz et al. [9], Lockwood et al. [10], and Lofstead et al. [11] have analyzed the impact of file systems using a selected group of benchmarks and applications. While system-level analysis identifies common contributing factors affecting broad subsets of jobs[1], they do not identify contributing factors within the applications themselves.

[1]We refer a single application execution as a 'job'.

At the application level, several researchers, including Devendran et al. [12] and Li et al. [13], analyzed I/O behavior and identified bottlenecks by ingesting the profiling code inside the application code. This approach requires a nontrivial amount of manual intervention, however, and is impractical for studying on a wide spectrum of applications.

In our previous work [8], we performed a long-running holistic analysis to investigate how system state impacts application I/O performance. We executed a set of benchmarks periodically for an entire year with controlled I/O behaviors (i.e., process count, I/O patterns, and file system stripe settings) and studied how I/O performance of different runs varies across both the long-term and short-term time windows. We then attributed performance variation to system "weather" changes. We have continued this work by studying Darshan logs collected from the general production job population over a two-month period. A preliminary analysis revealed that sometimes the same application may use different numbers of processes, choose different I/O patterns, and select diverse file system configurations from execution to execution. Consequently, performance of these applications varies widely. Identifying and attributing the application-level settings that cause poor I/O performance are challenging tasks because of the diverse configurations.

In this study, we built on our previous work by developing techniques to systematically identify I/O bottlenecks in arbitrary applications running "in the wild" in addition to applications and benchmarks executed in a controlled fashion, and we investigate the impact of system-level and application-level factors on the I/O performance. We devised a systematic "zoom-in" analysis strategy for identifying applications' I/O performance bottlenecks, and we conducted a comprehensive study of I/O logs collected over a two-month period on a production supercomputing system.

In our analysis, we first summarize a list of common I/O factors that impact I/O performance. These factors come from either the system *weather* (e.g., storage server load) or applications' I/O behavior (e.g., percentage of small I/O). We then analyze how these factors globally impact all the jobs running on the system. To locate the key factors that impact the individual applications' I/O performance, we group together the jobs belonging to the same application (identified by name). For each application, we leverage

978-1-7281-0912-1/19/$31.00 ©2019 IEEE
DOI 10.1109/CCGRID.2019.00021

parallel coordinates plot to cluster together jobs bottlenecked by the same contributing factors. We then "zoom in" on the jobs whose performance bottleneck is indistinguishable from the parallel coordinates plot and find out their performance bottleneck by an in-depth analysis of their detailed I/O behavior. With this zoom-in analysis, we are able to identify the common performance bottlenecks that impact I/O performance of a group of jobs or of an individual job. We have created a set of tools that are reusable for analyzing I/O logs collected by Darshan, the Slurm job scheduler [14], and Lustre file system monitoring. The overall contributions of this paper are as follows.

- Propose a top-down analysis approach for systematically zooming in on applications' I/O bottlenecks. Using this approach, we comprehensively analyze ≈88,000 I/O logs produced during a two-month period. Our analysis identifies the common performance bottlenecks for any application of interest and also uncovers several important I/O performance contributing factors that had previously received minimal attention.
- Demonstrate that none of the system-level factors could be identified as prominent to poor I/O performance in analyzing a large set of Darshan logs, where the applications were run without same configurations.
- Demonstrate that application-level configurations—such as a single process undertaking exceptionally heavy I/O, a single storage server storing all data, or unbalanced I/O from application processes and to servers—are typical causes of poor I/O performance.
- Develop a set of tools that are reusable for this analysis. These tools are able to visualize the I/O behavior of each job, as well as the impact of I/O contributing factors to the I/O bandwidth of a cluster of jobs belonging to any selected applications and users.

The remainder of the paper is organized as follows. We briefly discuss the I/O monitoring tools and techniques used in this study (§II). We then discuss our analysis strategy (§III) and present an analysis of logs collected over two months on a production system (§IV). We discuss the related work in system-level and application-level analysis (§V) and briefly summarize our conclusions (§VI).

II. Background

In this section, we first provide a high-level overview of the I/O instrumentation tools used in this study. We then present the techniques we used for our analysis.

A. Overview of existing I/O instrumentation tools

Darshan, Slurm, and the LMT are widely adopted instrumentation tools on HPC systems. These tools collect job-level I/O statistics, job-level resource utilization, and file-system-level I/O traffic, respectively.

1) Darshan instrumentation: Darshan is a lightweight, application-level I/O characterization tool designed to collect compact histograms, cumulative timers, and statistics that are representative of a job's I/O behavior. In a parallel program, each process independently records its per-file statistics, such as the total bytes written/read, start and end times of the write/read activity, and total read/write/metadata time on each file. The per-file statistics from each process are later aggregated before the application terminates, during which the statistics of accessing shared files (accessed by all or a portion of processes), private files (files accessed by one process), and the entire job (e.g., total bytes read/written by a job) are calculated and recorded as well.

2) Slurm logs: Slurm is a resource manager that allocates compute node resources for jobs submitted to a batch queue. It maintains a database that records the resource utilization of each job. We use the node count allocated to each job, since this information is not available in Darshan logs.

3) LMT log: The Lustre Monitoring Tool (LMT) monitors the I/O activity on the Lustre file system servers (e.g., Object Storage Servers (OSS), Object Storage Targets (OST), and MetaData Servers (MDS)). LMT collects I/O statistics at 5-second intervals, such as the CPU utilization of OSSes, MDSes, and the bytes read/written on the OSTs.

B. Sweep-line-based analysis

Darshan contains I/O statistics for both private and shared files accessed during a job's execution, such as the start and end times of the write/read activity on each file. Since large-scale simulation or analysis may access tens of thousands of files, finding I/O bottlenecks is a time-consuming process.

To tackle this, we introduced sweep-line analysis in IOMiner [15] to select a minimal set of files whose I/O time covers the I/O time of all file accesses in the job. We refer to this "minimal set" as the *IO covering set*. In Figure 1, six files are accessed in different periods, and the total I/O time is 18 seconds. The entire I/O time is covered by the *I/O accesses* of File1, File2, and File5. The non-overlapped portion of these three files is marked in red and selected as the IO covering set. The performance bottlenecks of accessing these files now represent the bottleneck for the entire job, since their I/O time covers this job's I/O time.

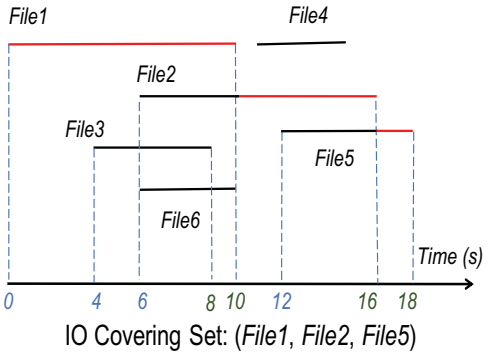


Figure 1. Overview of sweep-line-based analysis [15].

Table I
KEY FACTORS THAT AFFECT I/O BANDWIDTH

1	Data size (Datasize)	total number of bytes written/read by all MPI processes
2	Process count (nprocs)	# of MPI processes
3	Node count (nnodes)	# of compute nodes
4	Metadata overhead (Mtime)	% of CPU cycles spent on metadata operations over total CPU cycles spent on I/O
5	Collective I/O (Col)	use of collective buffering
6	Storage server count (OST #)	# of storage servers used by a job
7	Small I/O percent (Small (%))	% of small I/O (<1KB) among all the number of reads/writes
8	Sequential I/O percent (Seq (%))	% of sequential I/O requests among all the number of reads/writes
9	Data size distribution among processes (MaxRankIO)	distribution of bytes written/read among all the processes
10	Request count distribution among processes	distribution of number of read/write requests among processes
11	Data size distribution among storage servers	distribution of bytes written/read among all the storage servers
12	Request count distribution among storage servers	distribution of read/write requests issued to the storage servers
13	OSS CPU utilization (ossAvgCPU (%))	average CPU utilization of job's storage servers 5-second before it starts
14	MDS CPU utilization (mdsAvgCPU(%))	average CPU utilization of the metadata server 5-second before job starts
15	OST I/O traffic (ostAvgIO (MB))	average bytes read/written on job's storage servers 5-second before it starts

III. ZOOM-IN ANALYSIS APPROACH FOR I/O BOTTLENECK DETECTION AND ATTRIBUTION

In this section, we present our approach for I/O bottleneck analysis. We first summarize the factors contributing to I/O performance, then outline the workflow and tools developed for our proposed approach.

A. Key factors contributing to I/O performance

In Table I, we summarize an extensive list of factors that affect I/O performance. The factors numbered 1 to 8 (shown in gray) are the those widely considered as important in previous studies. For instance, factors 1 to 6 have been frequently discussed in previous literature [16], [17], [18]. Factors 7 and 8 are those that I/O tuning techniques aim to optimize [19], [20]. Factors 9 to 12 (shown in blue) are the factors identified as important during this study. Factors 13 to 15 (in yellow) are the key factors that show the current status ("I/O weather") of the system, identified as important in our previous study [8]. In this study, we consider a job's "weather" 5-second before it starts.

B. Zoom-in analysis on the performance impact of different contributing factors

We investigate the performance impact of the factors in Table I in different scopes. At the system or platform level, we identify the factors that have high impact across all the jobs running on the entire computing system. Optimizing these factors can potentially have platform-wide I/O performance improvement. We build a correlation matrix based on the contributing factor values and I/O bandwidth of each job, for measuring the global correlation of these factors with I/O bandwidth; and we visualize this matrix using a heatmap.

At the application level, our analysis identifies the most significant factors that impact the individual application's I/O bandwidth, enabling application developers to optimize these factors. Since jobs belonging to the same applications can be launched by different users with various configurations (e.g., process count, data size, I/O pattern), their bandwidth often falls into different categories (e.g., (0, 1] GB/s, (1, 10] GB/s, (10, 100] GB/s). To cluster the jobs based on their key performance-impacting factors (i.e., performance bottlenecks), we use parallel coordinates plots to cluster jobs first based on their users, since the same users often run applications with similar configurations, resulting in groups of jobs with similar I/O bandwidth and key performance-impacting factors. For each user, our parallel coordinates plot further clusters its jobs based on their bandwidth categories and performance-contributing factor values. This gives us insights into how these factors impact jobs in each bandwidth category.

Job-level analysis is needed when the performance bottleneck is indistinguishable from the parallel coordinates plot. We visualize I/O activity of jobs using sweep-line analysis (e.g., Figure 1) and investigate potential bottlenecks of the I/O accesses to the job's IO covering set files.

C. Tools developed for the zoom-in analysis

We have developed several tools for our top-down analysis, including ones to generate heatmaps and parallel coordinates plots for platform- and application-wide analysis, respectively, and a tool to visualize the sweep-line method for job-level analysis. Furthermore, we have developed a tool to extract important statistics for IO covering set files.

IV. BOTTLENECK ANALYSIS ON I/O LOGS

In this section, we first conduct a platform-wide analysis of the contributing factors listed in Table I, and then investigate how they impact the individual applications and jobs. We end the section with a summary of the key discoveries.

We perform our analysis on all available Darshan logs collected from runs on the Cori supercomputer hosted at the National Energy Research Scientific Computing Center (NERSC) during October 2017 and November 2017. Overall, there were 1.7 million Darshan logs, which capture 74 million core-hours and write/read traffic of 11.5 PB data.

Correlating the Darshan logs with the Slurm job scheduler logs reveals that these Darshan logs cover 39% of the total core-hours. In the remainder of the analysis, we exclude the jobs that write/read <1 GB of data or those that use a single process, because their I/O bandwidth is limited by the small data size or process count. We also exclude jobs with a run time of smaller than 5 minutes, because I/O time is important for jobs running across a reasonable timespan. These two constraints leave ≈88,000 jobs for further analysis.

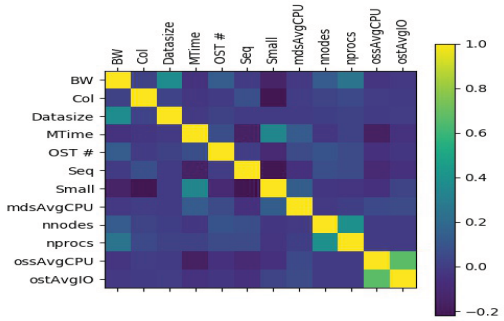


Figure 2. Heatmap showing the impact of different contributing factors to jobs' I/O bandwidth (I/O bandwidth is abbreviated here as BW).

A. Platform-wide impact of all the contributing factors

At the platform-wide level, we analyzed contributing factors 1–8 and 13–15 from Table I. The remaining factors were calculated based on applications' internal behavior and are used in the fine-grained application-level analysis. In Figure. 2, we show the correlation heatmap of all factors with I/O bandwidth. Among the 8 application-level contributing factors (1–8), 5 have a correlation coefficient either >0.1 (positive impact) or < -0.1 (negative impact). Data size and nprocs show the highest impact, with a correlation coefficient of 0.38 and 0.25, respectively. In contrast, the three system weather factors mdsAvgCPU, ossAvgCPU, and ostAvgIO have a correlation coefficient of only -0.02, -0.03, and -0.02, respectively. These observations suggest that contributing factors have different impacts on I/O bandwidth. However, none of them alone have significant platform-wide impact on all the jobs, because of the large number of jobs and the distinct I/O behaviors. We have also observed that a few contributing factors exhibit correlations with each other. For instance, the correlation between nprocs and nnodes is 0.39, and the correlation between Small and MTime is 0.34. Understanding the correlation between different factors is helpful when the applications' I/O bandwidth is limited by multiple correlated factors. For instance, when both Small and MTime are high, long MTime may be caused by the seek overhead from too many small I/O operations.

B. Overview of top applications consuming CPU-hours

To investigate the impact of key factors contributing to the I/O performance of individual applications, we grouped the ≈88,000 jobs based on their binary executable names and sorted them based on the total number of CPU-hours used by each executable group. Figure 3 shows the top 15 applications with the percentage of total CPU-hours consumed by each application. We have anonymized these applications to conceal the identity of the users by showing the science area of an application and adding an integer suffix when there are multiple applications in the same area, for example, Cosmology1 and Cosmology2. Overall, these 15 applications account for 70% of the total Darshan log count among the ≈88,000 logs and consume 74% of the total CPU-hours.

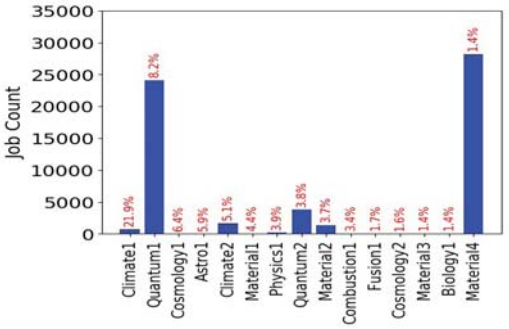


Figure 3. Top 15 CPU core-hour-consuming applications. The % value on top of each bar refers to the ratio of total core-hours of all the jobs in an application to the total core hours of all the 88k jobs.

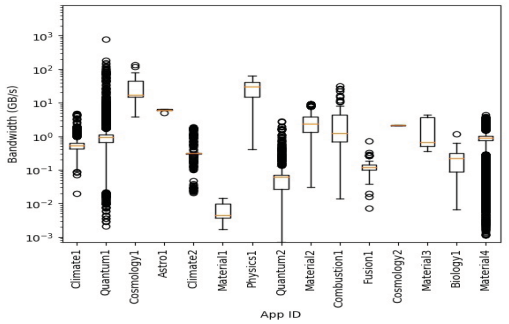


Figure 4. Distribution of the bandwidth of top core-hour-consuming applications among the selected jobs.

In Figure 4 we plot the I/O bandwidth distribution of the 15 applications. The I/O performance varies by multiple orders of magnitude for the same application. We studied the root causes of this variation and identified factors contributing to the poor I/O performance. Because of the page limit, we select to show Cosmology1, Combustion1, Cosmology2, Climate1, and Quantum1 applications for further analysis because they either consume a large fraction of CPU-hours or are highlighted by the supercomputing facility.

We choose a subset of I/O impacting factors in Table I and include them in a parallel coordinates plot, as shown in Figure 5 for the Cosmology1 application. Parallel coordinates are a typical visualization tool for showing multiple variable quantities in the same plot [21]. The plot lines are color coded based on users; in Figure 5, the red and blue lines represent two distinct users. We use MaxRankIO to

reflect the impact of the skewed data distribution of factor 9 in Table I, defined as Equation 1, where S_{max} is bytes written/read by the MPI rank[2] with maximum I/O, because we observed that one rank often writes/reads exceptionally more data than do the others; S_i is the I/O size of each rank; and nprocs is the number of MPI ranks in a job.

$$MaxRankIO = S_{max}/\sum_{i=0}^{nprocs-1} Si \quad (1)$$

While other factors outside those in a parallel coordinates plot could also become the performance bottleneck, they could be further identified by using the sweep-line analysis.

C. Analysis of Cosmology1 I/O

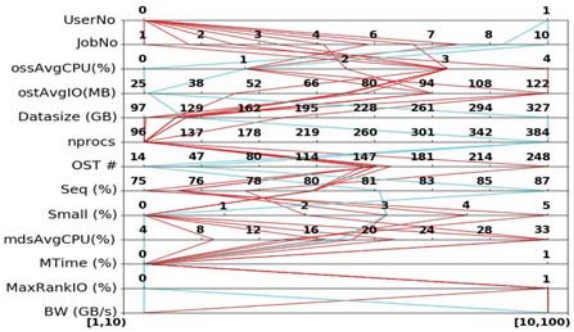
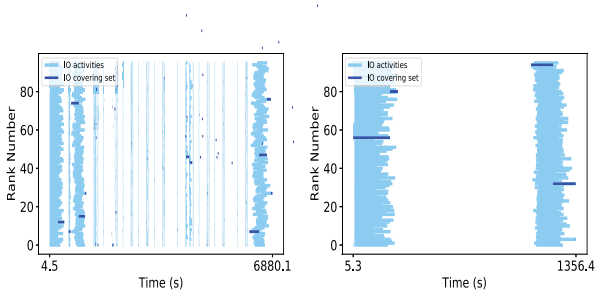


Figure 5. Impact of contributing factors to Cosmology1 I/O.

In Figure 5, we plot the representative contributing factor values of all the jobs of Cosmology1. Each job has a unique JobNo; its line spans across the axis of all the contributing factors, and its point on each axis represents its value on the corresponding contributing factor. UserNo represents the user that runs the job. Jobs run by different users are in different colors. In Figure 5, ten jobs (JobNo) belong to two users (UserNo). The bandwidth (BW) of all the jobs is above 1 GB/s. We can see that all the jobs use at least 14 OSTs (OST #), sequential I/O (Seq) dominates at least 75% of the total I/O requests, the small I/O percentage (Small) is constantly below 5%, and the percentage of CPU cycles on metadata operations (MTime) is within 1%. These factor values suggest that Cosmology1's I/O pattern is well formed. On the other hand, the average CPU utilization of the storage server 5 seconds before the job starts (ossAvgCPU) is below 4%, and the average CPU utilization of the metadata server 5 seconds before job starts (mdsAvgCPU) is below 33%. The average storage server I/O traffic 5 seconds before the job starts is below 122 MB (ostAvgIO), a tiny fraction of each storage server's maximum I/O traffic (15 GB). These weather factor values (mdsAvgCPU, ossAvgCPU, ostAvgIO) suggest that file system weather trivially impacts Cosmology1 I/O when the jobs start.

Despite the well-formed I/O behavior and friendly file system weather, the bandwidth of jobs run by both User

[2]A *rank* in our context refers to an MPI rank, which is a *process*.



(a) jobs with [1,10)GB/s (b) jobs with [10,100)GB/s

Figure 6. I/O activity of jobs for User 0 of analysis

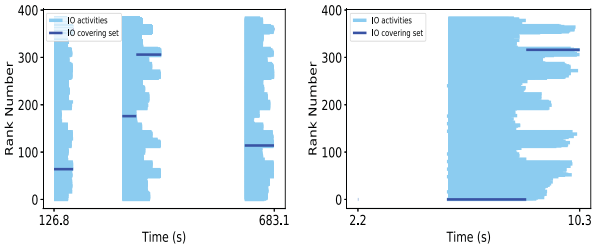
0 (red) and User 1 (light blue) falls into the range [1,10) GB/s and [10, 100) GB/s, respectively. The root causes are not distinguishable from the existing contributing factors in Figure 5.

To identify the root causes for this behavior, we 'zoomed in' on one job from each bandwidth category for each user. We plot the I/O sweep-line analysis of each job in Figures 6(a) and 6(b) for User 0 and in Figures 7(a) and 7(b) for User 1. The light blue lines in these figures are I/O activities of each rank writing/reading a file. The dark blue lines are the I/O activities on files of the IO covering set for the job. The x-axes in these plots show the execution time. In Figure 6(a), every rank is involved in many bursty I/O phases. *These frequent I/O phases are the key performance-limiting factors for the job referred to in Figure 6(a)*, since the I/O time of each I/O phase is bound by the slowest rank. In contrast, the bandwidth for the job referred to in Figure 6(b) is higher because each rank takes part in only two I/O phases.

In Figures 7(a) and 7(b), we show the I/O accesses of [1, 10) GB/s jobs and [10, 100) GB/s jobs for User 1, respectively. In Figure 7(a), the IO covering set includes four lines. Our investigation of these file accesses in the Darshan log reveals that these lines are generated by the I/O activities of writing eight files. The top six files with the longest I/O time range from 15 GB to 22 GB and are concurrently written by 64 processes, on average. However, all these files are stored on one Lustre OST using the default stripe count (1) set by the system. *The time for writing these files is constrained by the single OST's bandwidth, which we attribute as the reason for poor I/O performance*. In Figure 7(b), the IO covering set is dominated by the activity of reading two "RESTART" files of size 21 GB and 16 GB with 64 and 48 processes, respectively. Each file is stored on only one OST, similar to Figure 7(a). However, the aggregate bandwidth in reading these files far exceeds the single OST bandwidth, indicating that all or a fraction of these files are cached on the file system.

D. Analysis of Combustion1 I/O

In Figure 8, we show the parallel coordinates plot of Combustion1. There are 59 jobs belonging to three users. They exhibit diverse bandwidth spectra. For example, User



(a) jobs with [1,10)GB/s (b) jobs with [10,100)GB/s

Figure 7. I/O activity of jobs for User 1 of Cosmology1.

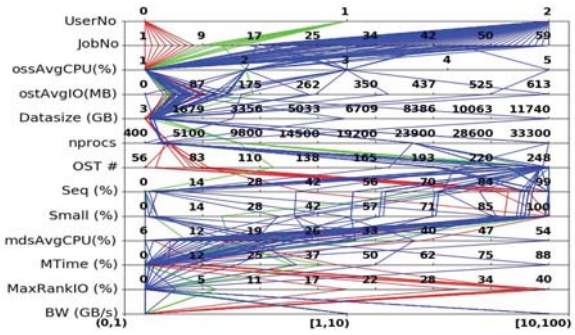
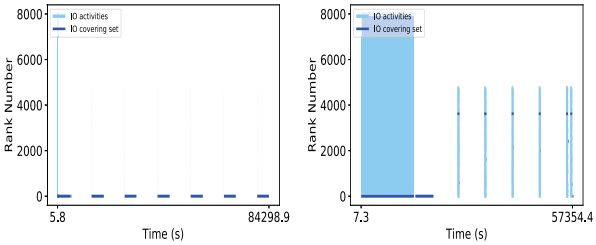


Figure 8. Impact of contributing factors to Combustion1 I/O.

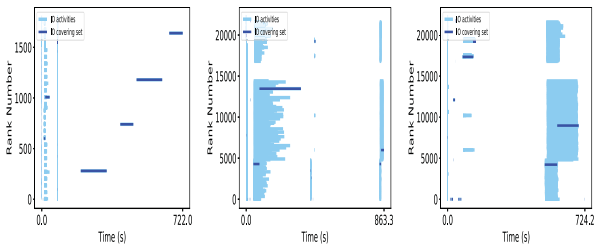
0's jobs (red) fall in (0, 1) GB/s. User 1's (green) job fall in [0, 1) GB/s and [1, 10) GB/s. User 2's (blue) jobs fall in [0, 1) GB/s, [1, 10) GB/s, and [10, 100) GB/s. The system weather (e.g., ossAvgCPU, ostAvgIO, mdsAvgCPU) trivially impacts I/O bandwidth, since the values mostly stay at low value similar to those of Cosmology1. Although the highest value of mdsAvgCPU is around 54%, it does not correlate with the high CPU cycles spent on metadata operations (e.g., MTime (%)).

In Figure 8, we can see that *User 0's jobs (red) are bottlenecked by a large percentage of small I/O jobs, since all its jobs' small I/O percent (Small %) is around 100%.* To identify the performance bottleneck for User 1's (green) jobs, we select two representative jobs from each of its bandwidth category and show their activities in Figures 9(a) and 9(b). In Figure 9(a), the IO covering set (dark blue) is dominated by seven of rank 0's I/O activities, *as rank 0 writes/reads much more data than other processes and straggles the entire job.* In Figure 9(b), the IO covering set is dominated by a long dark blue line. In its I/O activity, a shared file of size 65 GB is accessed by all the processes (as indicated by the large blue rectangular area). However, after investigating this file's Darshan records, we observe that *only one process performs the actual read operation, which straggles the entire job.*



(a) jobs with [0,1) GB/s (b) jobs with [1,10) GB/s

Figure 9. I/O activity of jobs for User 1 of Combustion1.



(a) jobs with [0,1) GB/s (b) jobs with [1,10) GB/s (c) jobs with [10,100) GB/s

Figure 10. I/O activity of jobs for User 2 of Combustion1.

Figure 10 shows the I/O activities of User 2's (blue in Figure 8) jobs that obtain bandwidths in the ranges of [0, 1) GB/s, [1, 10) GB/s, and [10, 100) GB/s. In Figure 10(a), the IO covering set is dominated by four I/O activities resulting from four ranks writing the same file with an aggregate of 7.2 million small write requests. *The large number of small writes are the bottleneck for this job's low I/O bandwidth.* In Figure 10(b), one large dark blue line dominates the covering set. During this activity, 72 ranks concurrently read a 73 GB shared file striped across 72 OSTs, and some of these OSTs are contended by other ranks. For instance, in Figure 11, we show another level of zooming in on this job's I/O workload on Lustre OSTs. The star-marked points represent this job's I/O traffic on the 72 OSTs. *Reading this shared file is bottlenecked by two spikes produced by the concurrent I/O activities from other ranks.* In Figure 10(c), the covering set is dominated by three dark blue lines resulting from I/O accesses to two files. Writing one file accounts 85% of the I/O time, during which 72 processes concurrently write a 73 GB shared file striped across 72 OSTs. However, these OSTs' I/O workloads are more balanced than those in Figure 11, as suggested by Figure 12. This explains the higher I/O bandwidth of this run than that shown in Figure 11.

E. Analysis of Cosmology2 I/O

In Figure 13, we show the parallel coordinates of Cosmology2 that contains three jobs of one user. All of them use 248 OSTs and 64K processes. Their bandwidth is within [1,10) GB/s. In Figure 14, we show the I/O activity of one Cosmology2 job. The I/O time is dominated by a blue rectangular region involving *all the ranks concurrently reading a shared file placed on only one OST because of the use of default stripe count (1), limiting its I/O bandwidth to 2.1 GB/s, below a single OST bandwidth (3 GB/s).*

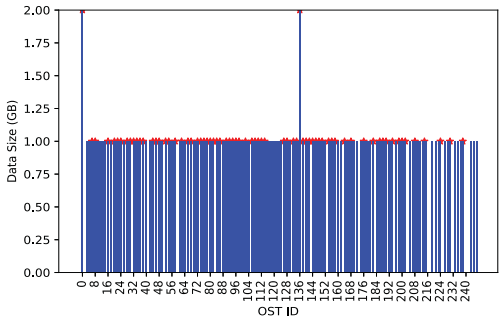


Figure 11. I/O workload on OSTs for a Combustion1 job corresponding to Figure 10(b).

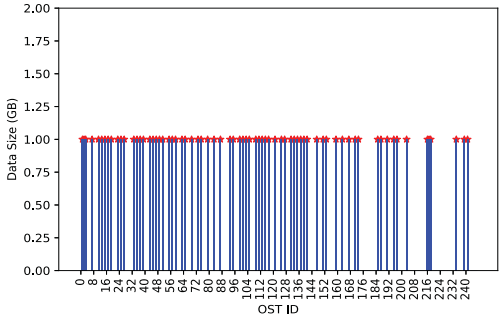


Figure 12. I/O workload on OSTs for a Combustion1 job corresponding to Figure 10(c).

F. Analysis of Climate1 I/O

In Figure 15, we plot the parallel coordinates plot of the Climate1 application with 693 jobs of four users. We focus on analyzing the performance of Users 2 and 3 since the contributing factor values of the other two users are entirely covered by these. Figure 16 and Figure 17 show the parallel coordinates plots for Users 3 and 2, respectively. In Figure 16, the I/O bandwidth of all the jobs is within (0,1) GB/s. This unanimously low bandwidth correlates with the contributing factor MaxRankIO, which is around 77% and 78% for all the jobs, meaning *data written/read by one rank dominates the total bytes written by all the ranks*.

In Figure 17, we plot the contributing factor values for User 2's jobs. Their I/O bandwidth fall in [0, 1) GB/s and

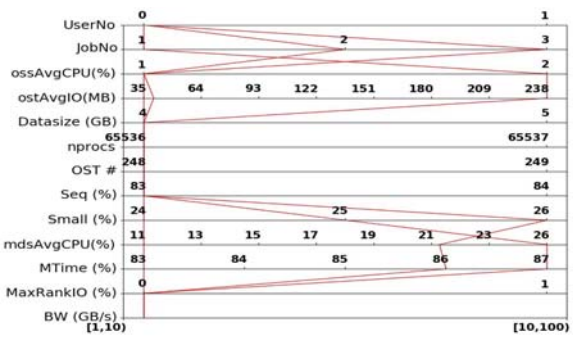


Figure 13. Impact of contributing factors to Cosmology2 I/O.

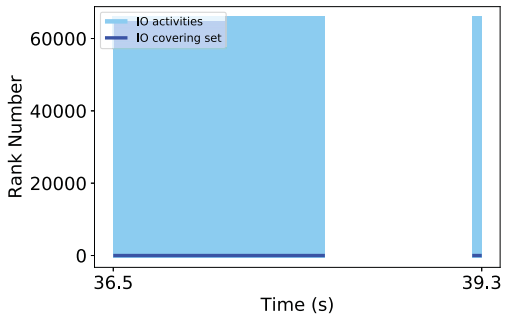


Figure 14. I/O activity of Combustion2 I/O.

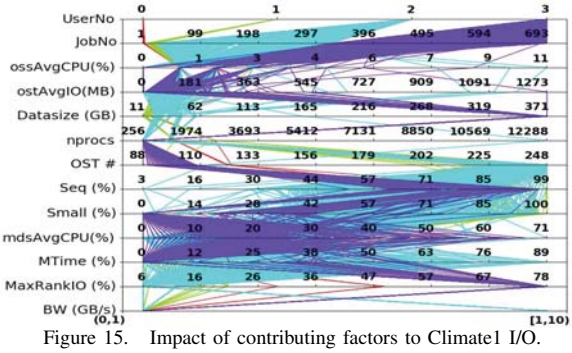


Figure 15. Impact of contributing factors to Climate1 I/O.

[1,10) GB/s. A detailed inspection reveals that all of these jobs' I/O bandwidths are within (0–1.5] GB/s. We select one of these jobs and plot its I/O sweep-line in Figure 18. We see multiple bursty I/O accesses involving all the ranks, followed by a number of I/O accesses that include a subset of processes (32 out of 1,024). The IO covering set is dominated by rank 0 writing 3 log files of 60.8 MB, 1.6 MB, and 0.2 MB. In writing these files, rank 0 issues 0.82M sequential writes. We have also observed that a long time was spent in writing these files from this user's other jobs, even though they were run on different days. *This long write time is probably due to some special treatment for writing logs (e.g., use of O_DIRECT)*. Further investigation into the application's I/O code is needed to confirm the root cause.

G. Analysis of Quantum1 I/O

In Figure 4, the range of bandwidth obtained by the Quantum1 application spans between a few megabytes per

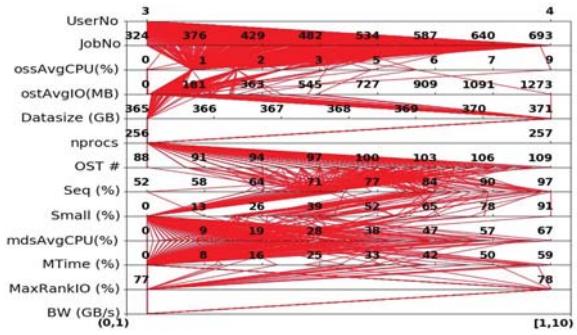


Figure 16. Impact of contributing factors to User 3's jobs of Climate1.

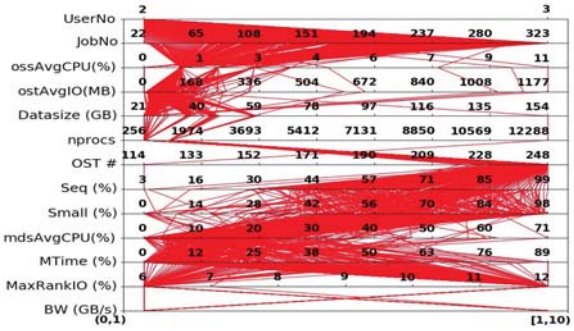


Figure 17. Impact of contributing factors to User 2's jobs of Climate1.

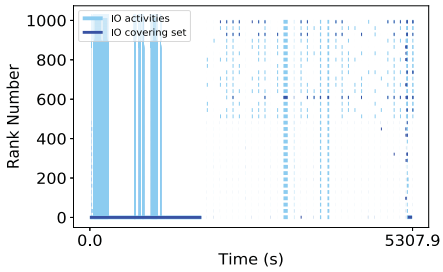


Figure 18. I/O activity of User 2's jobs of Climate1.

second to hundreds of gigabytes per second. This application has the second largest job count and spends the most CPU-hours in the analyzed logs (see Figure 3). Because of its large job count, we choose not to include its parallel coordinates plot and instead plot the bandwidth range for jobs of different users in Figure 19.

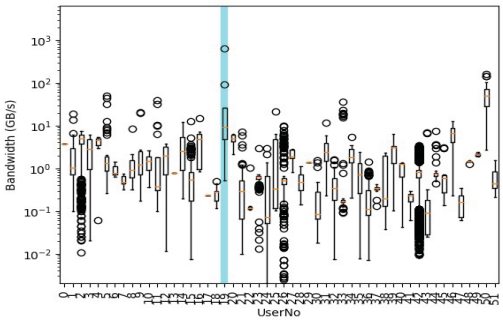


Figure 19. Bandwidth distribution of different users of the Quantum1 application.

We can see that the I/O bancwidth of User 19's jobs covers all the bandwidth ranges: (0, 1) GB/s, (1,10] GB/s, (10,100] GB/s, and >100 GB/s. Because of this wide range, we focus on analyzing User 19's jobs.

In Figure 20, we show the parallel coordinates plot for User 19's jobs. One job falls in (0,1) GB/s. This job writes/reads 1.2 GB of data. While the small data size is a bandwidth-limiting factor, we also plot its I/O accesses in Figure 21(a). Its IO covering set is dominated by a long I/O activity (dark blue), in which rank 39 writes a 4.8 MB file, accounting for 73% of this job's total I/O time. In writing this file, rank 39 issues eight sequential seeks and writes. This well-formed I/O pattern suggested that its long write

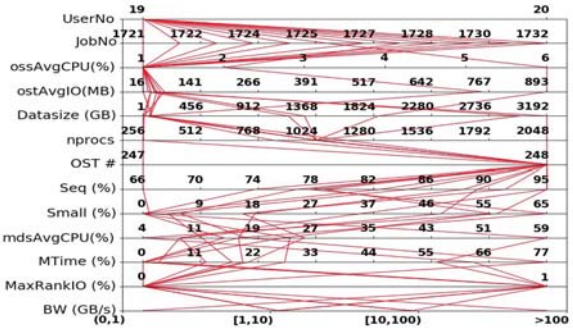
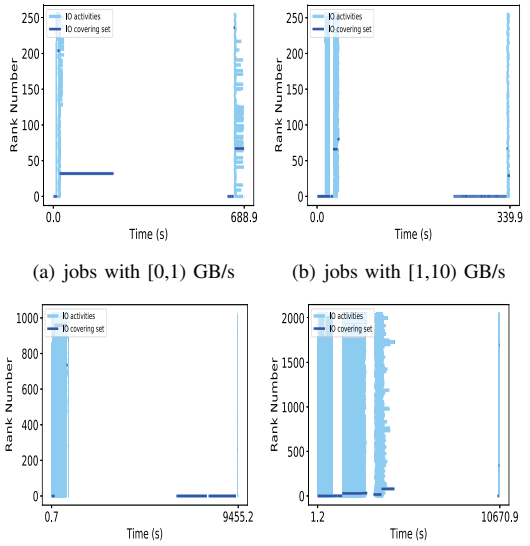


Figure 20. Impact of contributing factors on User 19's jobs of Quantum1.



(a) jobs with [0,1) GB/s (b) jobs with [1,10) GB/s

(c) jobs with [10,100) GB/s (d) jobs with $\geq$ 100 GB/s

Figure 21. I/O activity of jobs for User 19 of Quantum1.

time was likely due to the transient "weather" changes on the file system. *Further analysis revealed that the metadata server load during this job ("mdsAvgCPU" in Fig. 20) is 59%, which is significantly above that of the other jobs.*

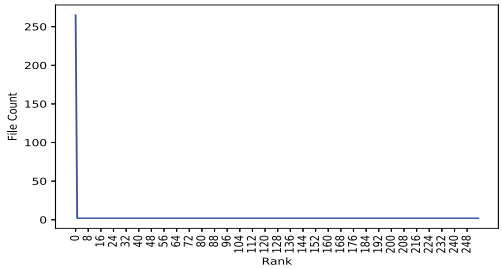


Figure 22. Distribution of the file count written by each rank of a Quantum1 job corresponding to Figure 21(b).

We have also plotted the I/O activities of User 19's jobs whose bandwidth is within [1, 10) GB/s, [10, 100) GB/s, and >100 GB/s in Figure 21(b), Figure 21(c) and Figure 21(d), respectively. In both Figure 21(b) and Figure 21(c) the covering set is dominated by rank 0's I/O. After further investigation into this job's Darshan log, we observe that

rank 0's I/O in Figure 21(b) is dominated by writing/reading numerous small configuration files (see Figure 22), which is the root cause of rank 0's long I/O time. In Figure 21(c), I/O activities are dominated by rank 0 writing/reading two files of size 6 GB each, and rank 0's data size is significantly above that of the other processes. However, the job's aggregate I/O bandwidth is still above that of Figure 21(b), since the processes write/read much more data (1.1 TB) than in Figure 21(b) (10 GB). Figure 21(d) exhibits the I/O behavior of the job with the highest I/O bandwidth. The I/O activity on the covering set is dominated by writing/reading four files. Further investigation of these files' records in the Darshan log reveals that they are first written and then read back by all the processes. *This read-after-write I/O pattern takes advantage of file system caching, which explains this job's high I/O bandwidth.*

H. Discussion

I/O performance evaluation of the five selected applications uncovers several performance-limiting factors, as shown in Table II. The gray/blue-colored cells signify negative/positive factors to I/O bandwidth, respectively. Some of them are seldom discussed in the literature, but our analysis indicates that they have significant impact on performance. In Table III, we summarize the applications/jobs and the factors impacting their I/O performance. For instance, we have observed unbalanced I/O workloads (#4 in Table II) among all the ranks in Combustion1, Climate1, and Quantum1, primarily because rank 0 undertakes exceptional I/O workloads. In addition, applications often adopt the default stripe count, which is 1 OST on Cori.

For applications that involve all or a subset of processes concurrently writing/reading a shared file, their aggregate I/O bandwidth can be limited by the default number of storage servers' I/O bandwidth (#2 in Table II). For example, in our analysis, the bandwidth of Cosmology1 and Cosmology2 is limited by the single OST bandwidth (see Table III). We have also observed that application bandwidth can be limited by the frequent synchronizations across many I/O phases (#1 in Table II on Cosmology1). On the other hand, high-performance jobs often exhibit some common characteristics, such as the use of the file-per-process I/O pattern (e.g., Cosmology1), read caching (e.g., Cosmology1 and Quantum1), and wide striping with balanced OST I/O (e.g., Combustion1).

Our analysis and the observations provide feedback to application developers, enabling them to carefully balance the I/O workload among different ranks and stay aware of different tuning options on the parallel file system. High-level I/O libraries (e.g., HDF5) and I/O middleware (e.g., MPI-IO) can use this feedback to automatically set the tuning parameters for achieving superior performance. Furthermore, our analysis shows that jobs of the same applications can have various I/O behaviors, resulting in distinct I/O bandwidth. I/O benchmark designers may benefit from considering the diversity of their applications' I/O behavior in mimicking their workloads.

Table II
INDEX OF ROOT CAUSES FOR JOBS' POOR I/O PERFORMANCE

1	Too many I/O phases
2	Bandwidth is limited by a single OST I/O bandwidth
3	Limited by the small data size
4	Unbalanced I/O workload among ranks
5	Too many small I/O requests
6	Unbalanced I/O workload on OSTs
7	Bad file system weather
8	Use of file-per-process pattern
9	Benefit from read caching
10	Balanced OST I/O workload with wide striping

Table III
SUMMARY OF KEY CONTRIBUTING FACTORS FOR DIFFERENT USERS AND APPLICATIONS

App/BW (GB/s)	(0, 1]	(1, 10]	(10, 100]	>100
Cosmology1 (User 0)		1	8	
Cosmology1 (User 1)		2	9	
Combustion1 (User 0)	5			
Combustion1 (User 1)	4	4		
Combustion1 (User 2)	5	6	10	
Cosmology2 (User 0)		2		
Climate1 (User 2)	4			
Climate1 (User 3)	4	4		
Quantum1 (User 19)	3&7	4		9

V. RELATED WORK

Existing efforts on I/O analysis can be classified into system-level and application-level analysis.

At the system level, much effort has been devoted to analyzing how the file system-side I/O activities impact applications' I/O performance. Yildiz [9] et al. used microbenchmarks to systematically analyze the root causes of the I/O interference on storage systems. Lofstead et al. [11] measured I/O performance variability based on the IOR benchmark. Lockwood et al. [10], [8] studied how the system "weather" (e.g., metadata load, file system fullness, and I/O contention) impact applications' I/O performance based on a set of benchmarks and applications. These works generally select a specific set of benchmarks or applications with well-formed I/O patterns (i.e., large sequential I/O) and run these applications in a controlled environment and fixed configuration (e.g., same process count) to observe how file system load impacts the performance in different runs. In contrast to these efforts, our analysis is performed on applications running "in the wild," with a focus on both the system-level impact and the impact from the I/O behavior of the application themselves.

An exhaustive number of application-level I/O studies have been carried out. Among them, Luu et al. [16] studied the I/O behavior of all the jobs running on the system, with a focus on I/O statistics such as distribution of jobs' process count and data size. This high-level analysis did not rely on a full picture of jobs' internal I/O behavior, however. Among

the manual profiling efforts, Devendran et al. [12] profiled the Chombo I/O benchmark and identified its performance bottleneck by ingesting the timer inside the applications. Similarly, Li et al. [13] and Liu et al. [22] analyzed the AMR and GEOS-5 applications. This type of profiling, however, requires significant manual intervention. Devarajan et aal. anlayzed the Montage application using PAT [23], a flexible I/O analysis framework. While PAT greatly simplifies the user's profiling effort, it does not reveal the applications' I/O behavior. Users still have to understand the applications' I/O in order to identify the pathological I/O behavior. In contrast, our sweep-line-based analysis visualizes a job's I/O behavior and performance bottlenecks based on their I/O traces. Our work also leverages parallel coordinates plots to cluster jobs with similar performance bottlenecks. This approach allows users to quickly find out the common I/O bottlenecks from a group of jobs.

VI. Conclusions

In this paper, we present an analysis approach to zoom in systematically to study the impact of I/O performance contributing factors at various scopes. At the platform wide, we have investigated the relative impact on all the jobs running across the platform. At the application level, our approach groups together jobs belonging to the same applications and clusters together jobs that are bottlenecked by the same contributing factors. At the job level, our approach identifies the I/O performance bottleneck for the individual jobs by an in-depth analysis on their I/O behavior. We use this zoom-in approach to analyze the I/O instrumentation data of ≈88,000 jobs, collected during a two-month period. Our analysis reveals several insightful conclusions that are useful for application developers, system administrators, and I/O library developers.

Acknowledgment

This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract numbers DE-AC02-05CH11231 and DE-AC02-06CH11357. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under contract number DE-AC02-05CH11231.

References

[1] C.-S. Chang, S. Ku, and H. Weitzner, "Numerical Study of Neoclassical Plasma Pedestal in a Tokamak Geometry," *Physics of Plasmas*, vol. 11, no. 5, pp. 2649–2667, 2004.

[2] D. Bader, W. Collins *et al.*, "Accelerated Climate Modeling for Energy (ACME) Project Strategy and Initial Implementation Plan," 2014.

[3] S. Habib, V. Morozov *et al.*, "HACC: Extreme Scaling and Performance across Diverse Architectures," in *SC13*. ACM, 2013, p. 6.

[4] A. MacDowell, D. Parkinson *et al.*, "X-Ray Micro-Tomography at the Advanced Light Source," in *Developments in X-Ray Tomography VIII*, vol. 8506. International Society for Optics and Photonics, 2012, p. 850618.

[5] "turbulence energetics in stably stratified geophysical flows: Strong and weak mixing regimes."

[6] S. Snyder, P. Carns, K. Harms, R. Ross, G. K. Lockwood, and N. J. Wright, "Modular HPC I/O Characterization with Darshan," in *Extreme-Scale Programming Tools (ESPT), Workshop on*. IEEE, 2016, pp. 9–17.

[7] *Lustre Monitoring Tool*, 2016, https://github.com/LLNL/lmt.

[8] G. K. Lockwood, S. Snyder, T. Wang *et al.*, "A Year in the Life of a Parallel File System," in *SC*. IEEE, 2018.

[9] O. Yildiz, M. Dorier, S. Ibrahim, R. Ross, and G. Antoniu, "On the Root Causes of Cross-Application I/O Interference in HPC Storage Systems," in *IPDPS*. IEEE, 2016, pp. 750–759.

[10] G. K. Lockwood, W. Yoo *et al.*, "UMAMI: A Recipe for Generating Meaningful Metrics through Holistic I/O Performance Analysis," in *PDSW*. ACM, 2017, pp. 55–60.

[11] J. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf, "Managing Variability in the IO Performance of Petascale Storage Systems," in *SC*. IEEE Computer Society, 2010, pp. 1–12.

[12] D. Devendran, S. Byna *et al.*, "Collective I/O Optimizations for Adaptive Mesh Refinement Data Writes on Lustre File System," in *CUG*, 2016.

[13] J. Li, W.-k. Liao, A. Choudhary, and V. Taylor, "I/O Analysis and Optimization for an AMR Cosmology Application," in *CLUSTER*. IEEE, 2002, pp. 119–126.

[14] *Slurm Workload Manager*, 2016, https://slurm.schedmd.com.

[15] T. Wang, S. Snyder, G. Lockwood *et al.*, "IOMiner: Large-Scale Analytics Framework for Gaining Knowledge from I/O Logs," in *CLUSTER*. IEEE, 2018, pp. 466–476.

[16] H. Luu, M. Winslett *et al.*, "A Multiplatform Study of I/O Behavior on Petascale Supercomputers," in *HPDC*. ACM, 2015, pp. 33–44.

[17] B. Xie, J. Chase *et al.*, "Characterizing Output Bottlenecks in a Supercomputer," in *SC*. IEEE Computer Society Press, 2012, p. 8.

[18] W. Yu, J. S. Vetter, and H. S. Oral, "Performance Characterization and Optimization of Parallel I/O on the Cray XT," in *IPDPS*. IEEE, 2008, pp. 1–11.

[19] R. Thakur, W. Gropp, and E. Lusk, "Data Sieving and Collective I/O in ROMIO," in *Frontiers of Massively Parallel Computation, 1999. Frontiers' 99. The Seventh Symposium on the*. IEEE, 1999, pp. 182–189.

[20] J. Bent, G. Gibson *et al.*, "PLFS: A Checkpoint Filesystem for Parallel Applications," in *SC*. ACM, 2009, p. 21.

[21] A. Inselberg, *Parallel Coordinates: Visual Multidimensional Geometry and Its Applications*. Berlin, Heidelberg: Springer-Verlag, 2009.

[22] Z. Liu, B. Wang *et al.*, "Profiling and Improving I/O Performance of A Large-Scale Climate Scientific Application," in *ICCCN*. IEEE, 2013, pp. 1–7.

[23] *Performance Analysis Tool*, 2018, https://github.com/intel-hadoop.