SAND2021-9943C

Lessons From Examining Repetitive Job Behavior and I/O Performance Variability on a Production HPC System

Emily Costa
Northeastern University, USA

Tirthak Patel
Northeastern University, USA

Benjamin Schwaller
Sandia National Laboratory, USA

James Brandt
Sandia National Laboratory, USA

Devesh Tiwari
Northeastern University, USA

ABSTRACT

As I/O demand of scientific applications increases, identifying, predicting, and analyzing I/O behaviors is critical to ensure parallel storage systems are efficiently utilized. This paper investigates I/O behavior and performance variability on a large-scale high-performance computing (HPC) system using novel methodology that identifies recurrent job I/O behaviors. We show how our unique methodology can be used to perform temporal and feature analyses to detect interesting I/O patterns in production HPC systems, and its implications operating/managing a state-of-art system.

1 INTRODUCTION

Large-scale high-performance computing (HPC) systems have been known to suffer from I/O performance bottleneck and variability as the applications that run on them perform a large amount of shared distributed I/O, be it in the form of data analysis output, loading data visualization applications, checkpointing and restarting an application state, and reading and writing large machine/deep learning models [4, 11, 15, 27, 33, 39, 51]. The increase in the complexity of large-scale workloads and file systems, only exacerbates the I/O bottleneck and variability condition [3, 9, 24, 29]. In fact, as exascale computing projects emerge, the HPC community has become more aware of the importance of investigating I/O trends extensively [3, 8, 23].

Thus, characterizing I/O trends of an HPC system has become critical in scaling I/O-intensive scientific applications on modern large-scale HPC systems. While previous studies have characterized and analyzed I/O workload performance and behavior on HPC systems, their analysis has been based on traditional I/O beliefs. These beliefs include HPC scientific workloads perform unique and repetitive I/O, have large and periodic I/O bursts, and I/O characteristics are similar for read and write I/O [6, 9, 27, 34]. However, our detailed I/O behavior study on large-scale production HPC systems reveals that some of these conventional beliefs might no longer be applicable. We develop and present a methodology to identify and quantify emerging I/O behaviors. In this paper, we make the following **contributions**:

(1) We provide a machine learning-based clustering methodology to detect I/O-intensive applications and their unique, yet repetitive I/O behaviors. We demonstrate the use of our methodology in identifying different I/O characteristics and mitigating I/O challenges.

(2) We find that I/O-intensive applications have more unique read behaviors than write, but write behaviors are more predictable. This makes it easier for I/O schedulers to predict, manage, and absorb write I/O bursts. However, systems operations administrators should devise read I/O management policies using our methodology.

(3) Conventional beliefs on HPC applications being repetitive hold true, but an application may have multiple unique I/O behaviors. We show these unique behaviors last for a short while and may overlap with each other for the same application. We suggest refraining from scheduling policies that rely on inter-arrivals regularity for I/O scheduling as it can lead to suboptimal outcomes.

(4) Performance variation has been critical for HPC I/O [23, 24, 27, 34]. We focus on understanding I/O characteristics that cause or worsen performance variability. We find that runs with similar I/O behavior observe significant performance variation, especially for read I/O. System operators should note that some applications observe more performance variation than others. The mitigation strategies should consider read and write I/O separately using our methodology even for the same application.

(5) For jobs that run repetitively, but perform a low amount of I/O, we suggest they be carefully managed by HPC centers in terms of performance variation mitigation. We show these applications are prone to observing the highest amount of variation which results in the most number of complaints.

(6) We find that to encourage and cultivate best I/O practices, users need to be educated about the behavior that having fewer files can help deliver more stable performance. One solution is to consolidate I/O data in one shared file, as opposed to having multiple unique files. However, the trade-off between observed performance variation and file striping needs to be carefully considered by the system operators.

(7) Our analysis reveals that clusters that run on weekends tend to observe some of the highest performance variations. This can be addressed by providing incentive to users to run I/O intensive jobs on different days.

(8) Our analysis shows that there are separate and disjoint time zones during which different applications experience high and low performance variations. System administrators can leverage our clustering methodology to detect and manage periods of high performance variation without performing any additional instrumentation or probing and without deploying new high-overhead ML-based prediction models.

We will be making our developed toolset available as a package so that other HPC systems can easily perform similar studies and test

the wider applicability of presented findings using this systematic methodology.

2 BACKGROUND AND METHODOLOGY

2.1 Overview of the System

This study is based on a production-level system, Blue Waters. Blue Waters is a Cray XE/XK based supercomputer with approximately 27,000 compute nodes, with reported peak performance of over $\approx$ 13 Pflop/s. Blue Waters supports a wide variety of science domain – major applications that routinely run on this system include but are not limited to geospatial, physics, astronomy, material science, uncertainty quantification, machine learning, and weather modeling domains. The system features three Cray Lustre parallel file systems- Lustre Home, Lustre Projects, and Lustre Scratch. Home and Projects have 2.2 PB of storage with 36 Lustre Object Storage Targets (OSTs) each, while the Scratch system has 22 PB of storage with 360 OSTs. The total raw storage in the file system is 34.0 PB, including 25.0 PB of disk storage, with a I/O bandwidth peak performance of 1 TB/s.

2.2 Data Collection

In this study, we utilize a rich I/O data set collected on Blue Waters[46]. The data collection and analysis period for this study is approximately six months (Jul-Dec 2019). Our study focuses on job runs using the POSIX (Portable Operating System Interface) I/O interface since the vast majority of I/O is done using POSIX ($\approx$ 90.4%) on this system. This study considers $\approx$ 150 thousand runs for analysis, each of these runs have complete and accurate I/O information captured by Darshan. The selected runs provide the I/O characteristics of most of the I/O-dominant workloads on the system.

Darshan is the primary I/O monitoring tool used to monitor and collect the I/O data for application runs. Darshan was deployed on Blue Waters because it provides application-level I/O tracing capability with low-overhead suited for production usages. Due to its rich collection of data and light-weight deployment, Darshan has been used to perform insightful analysis of I/O trends and characterizations [11, 24, 29, 35]. However, we acknowledge that Darshan does not capture server-side information (e.g. OST-level monitoring. compute and storage overhead, etc.) which is not required for this study and cannot be easily correlated with application-level behavior.

We noticed that the same application displayed unique read and write I/O behavior (e.g., read from the same shared file, but write to multiple different files). Previous works note this divergence [36, 42]. Therefore, when we study the repetitive I/O behavior from the same application and its corresponding performance variation, we consider read and write I/O separately. Furthermore, we carefully consider what constitutes an application. The same executable might be run by multiple users, but they might exhibit different I/O behavior due to users inputting potentially varying parameters or using different functions of the executable. Therefore, we consider them as different applications. Throughout our analysis, we distinguish between applications by providing a unique executable name and user ID pair.

2.3 Grouping Applications With Similar I/O

Why are we grouping application runs? In our study, we applied a clustering methodology to group applications. The motivation behind clustering runs is to identify the repetitive pattern of applications with similar I/O behavior. Using clustering methodology provides us the statistical confidence to differentiate between runs of the same applications with different I/O characteristics. Furthermore, clustering methodology allows us to identify runs with similar I/O behavior and perform analysis on their observed I/O performance.

What are the inputs and outputs of our clustering methodology? The inputs to our clustering methodology are I/O features collected from the Darshan logs of all application runs. At the end of clustering, we output clusters that consist of runs that belong to the same application and have similar I/O behavior. As noted earlier, the clusters are different for read and write I/O behavior.

Clustering process details. Using the Darshan tool, we collect I/O metrics of the runs, normalize the metrics, then cluster using hierarchical clustering from the scikit-learn API [38]. Our clustering methodology uses information about the I/O behavior of each run – the major I/O characteristics include I/O amount, I/O request size histogram, number of shared and unique files. The I/O amount done in a run is given in bytes. The I/O request sizes are represented in a histogram format such that the number of requests done in a certain range (e.g. 100-1K bytes) is provided. A total of 10 I/O request size ranges are given which are used as separate clustering metrics. A file accessed during the run is categorized as shared if more than one rank accesses it and unique if it is only accessed by one rank. A total of thirteen metrics from the Darshan logs were found to be most relevant for clustering and affected the clustering outcomes, and hence, those were used as the input features for clustering.

As a part of preprocessing the data for clustering, we normalize the parameters such that the distribution of the values have a normal distribution with an expected value of 0 and standard deviation of 1, or $\mu = 0$ and $\sigma = 1$. This is effective in accounting for the varying I/O behaviors in the applications. Since the clustering algorithm, hierarchical clustering, uses Euclidean distance to cluster the data, normalization prevents the algorithm from being partial to an input. This is due to Euclidean distance being sensitive to the scale and magnitude of parameters [32]. Hence, as the I/O metrics numerically scale differently, normalization is necessarily applied. Additionally, I/O metrics are weighted differently when compared to the same I/O metric in other runs in the respective clusters. Therefore, normalizing the I/O metrics avoids inter- and intra-application clustering biases.

Next, we cluster the read and write runs using agglomerative hierarchical clustering. This technique recursively merges pairs of clusters based on a linkage distance and forms a varying number of clusters per application instead of a fixed number. We chose this clustering methodology as it clusters based on a set threshold of similarity [41]. The 13-dimensional, as 13 parameters are used, Euclidean distance between the I/O metrics of each run is used for linkage distance. Agglomerative hierarchical clustering can also be done by clustering into a certain number of clusters for each group, but we used distance threshold in order to allow groups to cluster

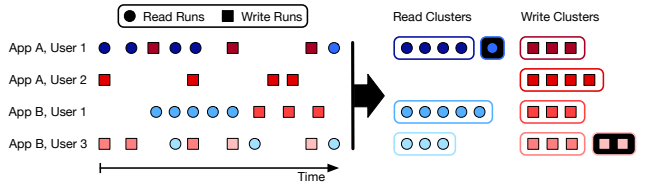

Figure 1: Visual representation of the clustering methodology. Different colors represent different I/O behaviors.

into different numbers of clusters based on how many distinct I/O behaviors exist within them.

After the clusters are formed, we set a threshold for the minimum number of runs in a cluster required for a cluster to be included in our study. While other studies have used lower thresholds [16, 35], we instated a higher threshold to ensure that our study draws statistically significant conclusions (e.g., clusters with only a handful runs are not very useful for learning the common behavior of applications). However, setting a very high threshold can automatically eliminate most clusters. To balance between competing trade offs, we use a threshold of forty runs in a cluster since we found that it was the minimum number of runs required to achieve statistical significance (number of runs per cluster) and it also resulted in a sufficient number of read/write clusters (497 read clusters and 257 write clusters). Higher thresholds can be chosen and similar conclusions will be obtained, but we aimed at using the maximum number of clusters possible while maintaining statistical significance.

Summary of frequently used terms and definitions. "Application" refers to a unique pair of executable and user ID. Applications are run multiple times during the course of this study. Application "runs" with similar I/O behavior – as determined by our clustering methodology based on the I/O and execution characteristics (e.g., Darshan tool) – are grouped together. These groups are referred to as "clusters". Note that an application can have multiple clusters. Each cluster can have a different number of "runs" and may span over different times. That is, one cluster can have 200 runs spanning over three months while another cluster with 500 runs may span only two weeks. Since runs within a cluster have similar I/O behavior, they are expected to achieve similar performance – although that may not always be the case. Each application can have multiple active clusters at a given time and the temporal density of runs within a cluster may be different. These concepts and terms are summarized visually in Fig. 1.

2.4 Workloads

After our clustering processes, our study is done with 497 read clusters and 257 write clusters with $\approx$ 80 thousand runs for read clusters and $\approx$ 93 thousand for write clusters. The runs included in our study used the executables `Vasp`, `Quantum Espresso (QE)`, `MoSST Dynamo (mosst)`, `SpEC`, and `Weather Research and Forecasting Model (WRF)`. As previously mentioned, applications are executable separated by the user running the executable. Note that applications are represented throughout this paper by an executable short-hand along with a number that distinguishes between users (e.g. `vasp0`, `vasp1`).

The type of workloads in our study include geospatial, weather modeling, quantum-mechanical simulation, and benchmark applications. Note that our study does not explicitly include the full extent of workloads on Blue Watersbecause they may not be run repeatedly enough and may not perform significant I/O. The variety of workloads included in our study capture the representative I/O behavior on the system.

2.5 Result Metrics

In this section, we lay out the frequently used metrics in our analysis.

I/O performance. It is as reported by the Darshan tool in terms of I/O throughput (amount of I/O performed per unit time).

Coefficient of Variation (CoV). When quantifying the relative variation of metrics within clusters, we used Coefficient of Variation (CoV). This statistical measure normalizes the standard deviation, σ, to the average, μ, given I/O behavior exhibited in each cluster. CoV provides a standardized means of quantifying variation, the dispersion of data about the mean, and is given as a percentage with the following formula:

$$CoV = \frac{\sigma}{\mu} * 100$$

Z-score. While CoV gives a standardized measurement to compare cluster behavior, z-score provides a standardized means to compare job behavior. The z-score for each job provides how many standard deviations a given metric is from the average of the jobs in its respective cluster. This enables us to identify I/O trends by quantifying the discrepancy of individual job I/O behaviors in relation to jobs with similar characteristics. A z-score, Z, with $-1 < Z < 1$ is within one standard deviation of the metric average and means the given job I/O behavior does not vary significantly from others in the same cluster. Additionally, a z-score with $1 < |Z| < 2$ displays high deviation and $|Z| > 2$ should be considered an outlier of the data distribution. The following is the formula for z-score:

$$Z = \frac{x - \mu}{\sigma}$$

Cumulative Distribution Functions (CDFs). We use CDF to show the cumulative distribution of a metric. Vertical draws are added to show the median values of the metrics.

Box plots and violin plots. These plots are used to show the density and distribution for values of a metric in a vertical format. In each violin in the violin plots, the median is given by a solid horizontal line while the 25th and 75th percentile are indicated with horizontal dashed lines. In each box in the box plots, the median is also given by a solid horizontal line while the 25th and 75th percentiles are represented by the ends of the box.

3 INVESTIGATING GENERAL TRENDS IN JOB I/O BEHAVIOR

In this section, we present the general I/O behavior results of different applications and their read and write clusters.

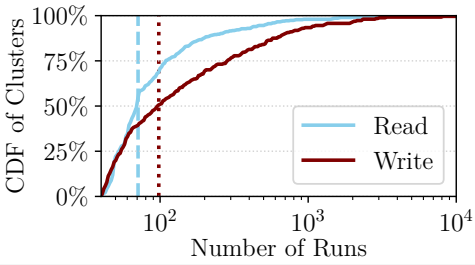

Figure 2: Write clusters have more runs than read clusters: the median size of a read cluster is 70, and that of a write cluster is 98.

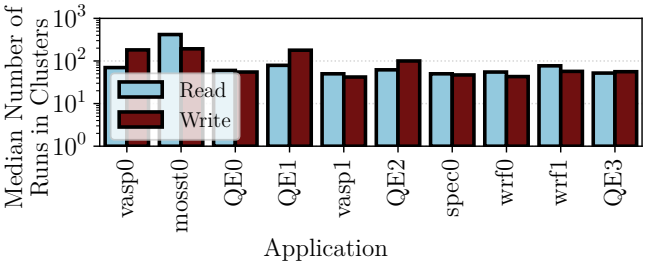

Figure 3: While on average, write clusters tend to have more runs than read clusters, many applications do not exhibit this behavior.

Operation with a Higher Median Number of Runs by Application	
Read	mosst0, QE0, vasp1, spec0, wrf0, wrf1
Write	vasp0, QE1, QE2, QE3

Table 1: The I/O operation, read or write, that has a higher median number of runs in the clusters of an application.

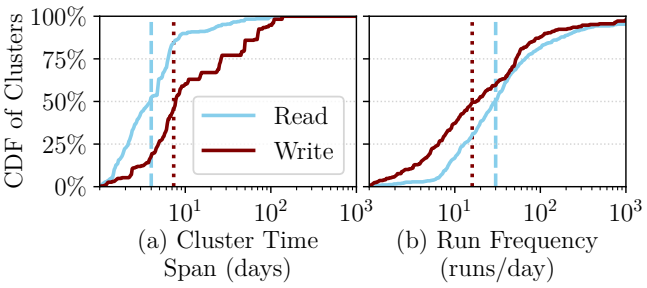

Figure 4: (a) 80% of read clusters span <10 days and 40% of write clusters span <10 days. (b) The median frequency of runs is 58 runs per day for read clusters and 38 runs per day for write clusters.

3.1 Characterizing Cluster Behaviors

We begin by looking at the characteristics of the read and write clusters generated using our clustering methodology. We ask the following question:

RQ 1. Do applications exhibit different repetitive I/O behavior in terms of read and write?

Divergence in Read and Write I/O Behaviors. Interestingly, our methodology produces almost twice the number of read clusters as write clusters – that is, the read I/O behaviors are more unique in number (hence, more clusters). More than 70% of the applications demonstrating more distinct read behaviors than write behaviors. For instance, the `vasp0` application has 406 read clusters, but only 138 write clusters. One could hypothesize that this trend is an artifact of that there are more read runs than write. But, on the contrary, our study covers ≈ 13k more write runs than read. The number of unique read I/O behaviors are relatively more. But, how frequently do these behavior occur (i.e., number of runs within a cluster)?

Fig. 2 shows a cumulative distribution function (CDF) of cluster sizes, i.e., the number of runs in the read and write clusters. The vertical lines show the corresponding medians. The medians show that there are generally more write than read runs in the clusters – implying that *write behavior are more repetitive than read behavior, although the number of unique write behaviors are lesser.*

Write clusters have a median of 98 runs, while read clusters have a median of 70 runs. Furthermore, the 75^{th} percentile is 288 for write clusters, but only 111 for read clusters. For example, application `vasp0` has a median read cluster size of 70 and a median write cluster size of 182. However, this behavior varies from one application to another. In Fig. 3, the median read and write cluster sizes of an application are shown. Additionally, Table 1 categorizes the applications by whether their clusters typically experience more read or write runs. We observe that, although the number of runs in write clusters tends to be higher than read clusters for several applications, this is not always the case. For example, application `mosst0` has a median read cluster size of 417, but a median write cluster size of 193. Nonetheless, in general, applications tend to have more distinct read behaviors (more clusters), and more consistent write behaviors (more runs per clusters).

Lesson Learned 1. *I/O-intensive applications tend to have more unique read behaviors than write ones, but write behaviors are more repetitive and hence, more predictable.*

Implication. This makes it easier for I/O schedulers to predict, manage, and absorb write I/O bursts. On the other hand, systems operations administrators should devise read I/O management policies using our clustering methodology to ensure that unique read I/O characteristics get detected and carefully handled to reduce I/O performance degradation and variation, which are well-documented phenomena on shared HPC storage systems [12, 25, 35].

We now look at how the cluster behaviors vary in time for different applications. Recall that runs within a single cluster can occur over time and runs within a single cluster represent the same I/O behavior repeating itself over time. That is, we ask:

RQ 2. How long does a typical repetitive I/O behavior last? How frequently do repetitive I/O behavior runs of the same type occur?

Temporal Spans of Clusters. First, we study the "time span", which is the amount of time between the start of the first run in a cluster to the end of the last run in the cluster. Fig. 4(a) shows the CDF of the time spans of clusters. We observe that write clusters tend to span longer than read clusters — that is, repetitive write I/O behavior tends to last longer than repetitive read behavior on average.

The median read cluster spans ≈ 4 days, while the median write cluster spans ≈ 10 days. Over 80% of read clusters span <10 days, whereas only 40% of write clusters span 10 days. In general, the cluster spans are short even though our study covers a time span of six months. For example, the median time spans of the clusters of

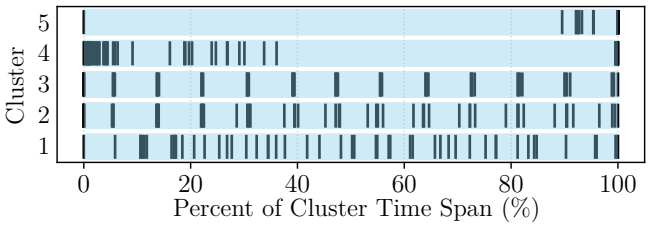

Figure 5: Different read clusters of the `vasp0` application have different inter-arrival patterns, with some clusters having more periodic and less irregular behavior than others.

the four applications with the most number of clusters are between 8 and 10 days. In addition, because write runs tend to span longer, Fig. 4(b) shows that write runs in a cluster occur at a lower frequency than read runs, in spite of write clusters tending to have more runs. Even though read clusters tend to have fewer runs, the runs in the clusters tend to occur more frequently. This indicates that unique I/O behavior of applications, especially read I/O behavior, lasts for a relatively short period of time, making it difficult to establish long-term (months) or even medium-term (weeks) patterns.

Lesson Learned 2. *Long-held beliefs about HPC applications being repetitive hold true, but a single application may have multiple unique I/O behavior. But, in contrast to conventional wisdom on HPC I/O [10, 13, 34, 51], we find unique I/O behaviors of typical repetitive HPC applications do not last long-term. Surprisingly, this is the case even for runs within write clusters, which have more runs.*

Implication. Designing I/O policy based on infrequent or "once in a while" profiling of an application's I/O behavior needs to be revisited, and a more dynamic approach needs to be adopted. This is because applications, while repetitive, change their behavior rather quickly. Stereotyping application I/O behavior toward I/O scheduling decisions can lead to suboptimal outcomes [4, 14, 17, 43].

Periodic Behavior of Runs. Next, we look at the temporal behavior of runs within a cluster. To motivate the analysis, in Fig. 5, we provide a visual representation of the normalized temporal distribution of run start times (vertical lines) of some clusters of the `vasp0` application. The x-axis is normalized to each cluster's time span for easier comparison. We observe that runs of different clusters of the same application can have very different inter-arrival patterns, even though they are run by the same user. For example, in cluster 5 we have a few close-by runs in the beginning and then no runs until the end. In cluster 3, we have several bursts of runs at regular intervals. In cluster 1, the application runs are almost randomly distributed from the first run to the last run. Note that we found that these behaviors are not correlated with the number of runs as all six clusters in this example have the same number of runs. In fact, we found that these behaviors are best correlated with the time a cluster spans (with a Pearson correlation coefficient of 0.75 in our example clusters).

In order to quantify these differences in behavior, we calculate the coefficient of variation (CoV), which represents the standard deviation of the inter-arrival times of runs within a cluster as a percentage of the mean inter-arrival time. The higher the CoV, the more irregular the inter-arrival times. To best understand this CoV, we plot it as a function of the time spans of different clusters in Fig. 6. One may expect that clusters that span longer times may

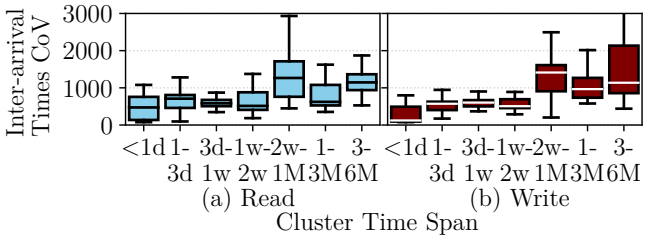

Figure 6: In general, the coefficient of variation (CoV) of inter-arrival times of runs in clusters increases with increasing time spans.

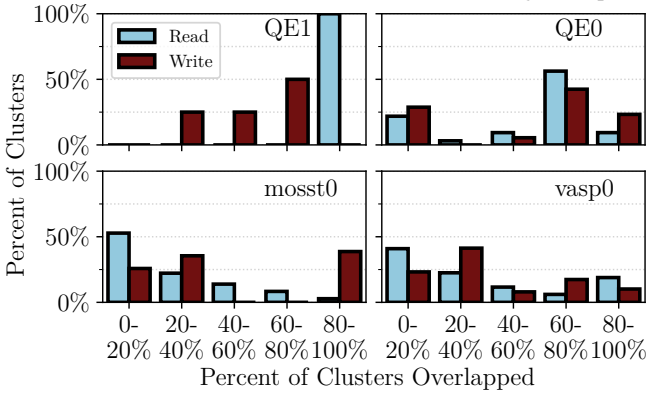

Figure 7: Applications have different behaviors in terms of runs from clusters with different I/O behaviors running simultaneously.

exhibit more irregularity in run inter-arrivals as inter-arrival patterns may change over a longer time period. This is in fact true. The figure shows that for both read and write clusters, in general, the CoV of inter-arrival times increased with the time span of the clusters. However, this CoV is quite high even for clusters spanning just a few days. For example, the median CoV of inter-arrival times is 514% and 506% for read and write clusters spanning 1-2 weeks, respectively. Thus, in general, application I/O clusters tend to have irregular inter-arrival times regardless of how long the clusters span.

Lesson Learned 3. *Read and write runs within a cluster are likely to have stochastic inter-arrival times regardless of the cluster span. Even clusters belonging to the same application and user may have different inter-arrival patterns. Unlike previous studies [2, 7, 51], our findings show that no assumption can be made about the inter-arrival patterns of different I/O behaviors of different applications.*

Implication. I/O behavior could be repetitive, but arrival patterns of runs belonging to a unique behavior can not be predicted trivially. System resource managers should refrain from policies that rely on regularity in inter-arrivals for I/O scheduling.

So far we have looked at the characteristics of clusters individually, next we study how they interact with each other temporally. A related inquiry is to understand how clusters (i.e., unique I/O behaviors) overlap with each other. In other words,

RQ 3. Can applications have multiple unique I/O behaviors active at the same time? Or, does one unique behavior (cluster) mostly start only after a previous one has finished?

Temporal Overlap of Clusters. In order to understand if applications express multiple unique I/O behaviors simultaneously, we look

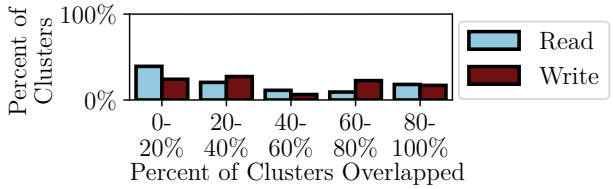

Figure 8: In general, an application can have overlapping clusters.

at the temporal concurrency of clusters from the same application. Fig. 7 shows the percent of clusters more than 50% with a given range of other clusters for the four applications with the most clusters. We present this in terms of percentage of clusters in order to normalize across applications and read or write operations, as each application and operation have varying numbers of clusters. We observe that applications `QE0` and `QE1` have temporal concurrency for read and write operations as a majority of clusters overlap with most of the other clusters in the application. On the other hand, application `mosst0` experiences less temporal concurrency for its write and, most notably, read clusters. This indicates that some write I/O behaviors occur concurrently, while read I/O behaviors occur at strictly unique time periods for this application. We also observe that application `QE1` has read clusters that overlap with all other read clusters from the application. Fig. 8 shows the overlap across all applications: in general, applications tend to have overlapping clusters as the majority of clusters do, in fact, overlap with at least one other cluster.

Lesson Learned 4. *A widely-held belief is that HPC applications running on an HPC system express a unique I/O behavior, at least short-term [12, 24, 34]. However, our study shows that applications can express numerous I/O behaviors in the same time span.*

Implication. If a user experiences performance variation when running the same application multiple times simultaneously, our clustering methodology can be used to pinpoint the differences in the runs, since although the user may expect similar performance but these runs might belong to different unique behavior (i.e.,clusters). Hence, the user's expectation may not be well founded and our methodology uncovers that and could be used for better troubleshooting experience with the users.

4 INVESTIGATING TRENDS IN I/O PERFORMANCE VARIABILITY

In this section, we start by characterizing the performance variability characteristics leveraging our clustering methodology and the insights gained from the repetitive behavior analysis in the previous section. Then, we study how different factors such as the number of runs, time span, and I/O amount affect the performance variation. Thereafter, we investigate the I/O and temporal characteristics that differentiate the clusters with the highest and lowest performance variations. First we ask,

RQ 4. Do runs belonging to the same cluster (i.e., similar I/O behavior) experience different I/O performance?

Performance variations within a cluster. Recall that runs within each cluster exhibit similar I/O behaviors based on multiple distinctive I/O characteristics captured by Darshan (Sec. 5), and hence, our clustering methodology clusters them together. Two runs exhibiting

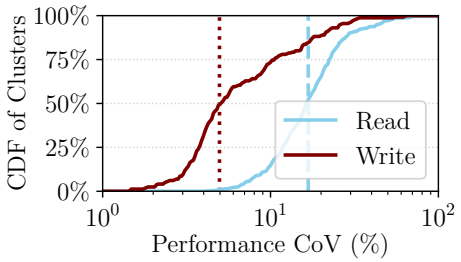

Figure 9: Runs within a cluster observe significant performance variation (CoV), especially for read clusters.

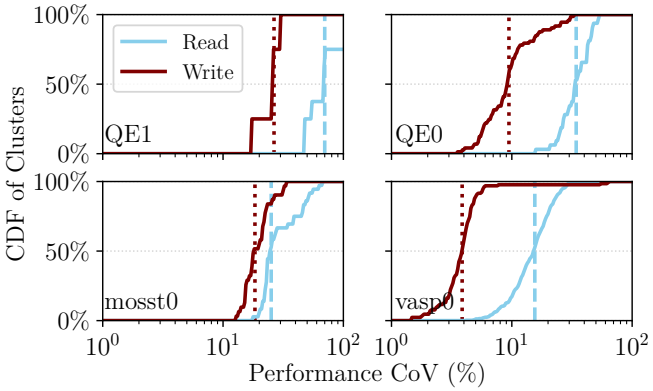

Figure 10: The I/O performance CoV CDFs of the four applications with the most number of clusters show high CoVs. The CoV for read clusters is notably higher than write clusters for each application.

similar I/O behaviors but run by different users and/or instances of different applications are put in *different* clusters.

Ideally, one may expect that runs that exhibit similar I/O behaviors (empirically less than 1% variation for all I/O characteristics) should experience similar I/O performances. However, Fig. 9 reveals that this is not true on our production system. Runs within the same cluster observe significant coefficient of variation (CoV, the standard deviation as a percent of the mean, $> 10\%$), even though these runs belong to the same application and are run by the same user.

While the magnitude of the performance variation is somewhat surprising, this trend by itself is not unexpected. Due to a variety of reasons, HPC applications may observe performance variation across their runs. However, a revealing finding is that performance variations across runs within the same cluster are almost consistently higher for read clusters than write clusters. The median CoV for read clusters is 16%, but 4% for write clusters. Applications performing similar read I/O behavior are likely to experience higher performance variation. Later in this section, we investigate different contributing factors that affect or are correlated with performance variation.

However, the first natural inquiry is whether this aggregate behavior is true only for some applications or for all applications. Fig. 10 confirms that these findings are indeed true across different applications, although the magnitude of performance variation and asymmetry between read and write performance variation may vary according to the application.

We note that the high amount of observed performance variation is not due to a methodological pitfall (e.g., a permanent performance change due to algorithmic improvement in application code being

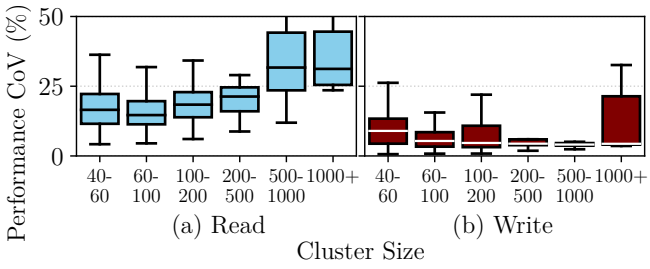

Figure 11: The performance CoVs of (a) read clusters and (b) write clusters do not change significantly and consistently with the cluster size (number of runs in the cluster).

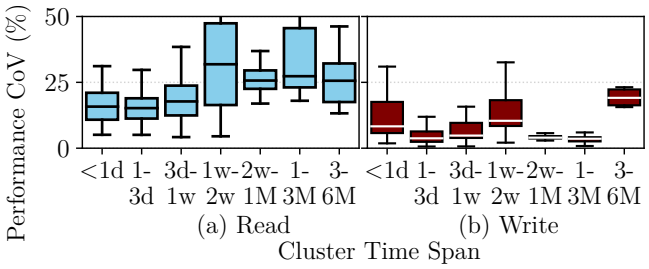

Figure 12: The performance CoVs of (a) read clusters and (b) write clusters generally increase with the time spans of the clusters.

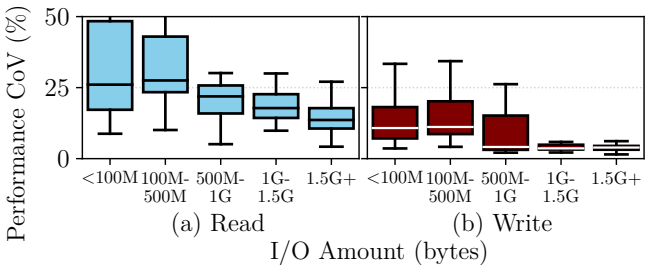

Figure 13: The performance CoVs of (a) read clusters and (b) write clusters generally decrease with the increase in the amount of I/O of the runs in the clusters.

mistakenly treated as performance variation). These variations are uncorrelated with chronological time across applications with different I/O characteristics.

Lesson Learned 5. *Runs within the same cluster (i.e., runs with similar I/O behavior) observe significant performance variation and the magnitude of variation is higher for read I/O. These trends are true consistently across different applications. While previous studies have observed I/O performance variation on production HPC systems [4, 11, 15, 24, 27, 29, 51], we show that read and write I/O are not equally susceptible to variation.*

Implication. System operators and user outreach teams should be aware that some applications (and their read I/Os) typically observe more performance variation than others. The mitigation strategies should consider read and write I/O behavior separately even though these runs might be coming from the same application and user.

Next, we investigate the performance variation of clusters based on their other characteristics such as number of runs in the cluster (size), time span of the cluster, and average amount of I/O performed by a run in a cluster.

RQ 5. Does performance variation correlate with number of runs in the cluster (size), time span of the cluster, and I/O amount?

Performance variations and cluster characteristics (size, span, and I/O amount). Fig. 11 shows the performance variation for different clusters grouped by their cluster size (i.e., number of runs in the cluster). We make two observations. First, as the cluster size increases, the performance variation may appear to increase in some cases, but the trend is not consistent. This indicates that simply because an application was run a number of times with similar I/O behavior does not automatically imply that it has a higher likelihood of experiencing a high performance variation. To confirm this, we performed a statistical test. The Spearman coefficient between cluster size and performance CoV is only 0.40 for read clusters and -0.12 for write clusters; this is a weak correlation. Second, we observe that irrespective of the cluster size, runs in a read cluster experience higher performance variation than write runs. This aligns with our previous finding, and reaffirms that this behavior is present in different types of read clusters of different sizes.

The previous finding naturally leads us to ask: is the performance variation correlated with the time span of a cluster? Recall that the cluster size simply refers to the number of runs within a cluster, these runs may be temporally spaced out in time or clustered in time. Time span of a cluster refers to the start of the first run in the cluster to the end of the last run in the cluster. Fig. 12 shows the performance variation for clusters with different time spans. We observe two interesting trends. First, read clusters typically have higher performance variation than write clusters for all clusters spanning different time periods — from less than one day to 3-6 months. Second, both types of clusters tend to observe higher performance variations if the clusters span longer, especially read clusters. These results indicate that when applications are run with similar I/O behavior, they tend to observe higher performance behavior over longer time spans. This is because there is a higher probability of different kinds of interference from different applications and the effects of system software and hardware upgrades.

Next, Fig. 13 shows the performance variation for clusters with different I/O sizes. The I/O amount per cluster is calculated as the average I/O performed by each run in the cluster (runs within a cluster have similar I/O amounts as a result of the clustering methodology). These results reveal two interesting trends: (1) Consistent with our previous findings, the read clusters observe more performance variation for any given I/O amount bin. (2) Runs with lower I/O amount tend to experience higher variation. For example, the median performance CoV for read clusters with less than 100MB is 26%, it is 14% for read clusters with more than 1.5GB. Similarly, the median performance CoV for write clusters with less than 100MB is 11%, while the same for write clusters with more than 1.5GB is 4%. Clusters lower I/O amount tend to spend relatively smaller amounts of time on I/O and hence, are more prone to be affected by transient interference from other applications, leading to a higher performance variance.

Lesson Learned 6. *Our results show that while the number of runs in a cluster does not have a significant impact on the performance CoV of the cluster, the performance CoV does generally increase*

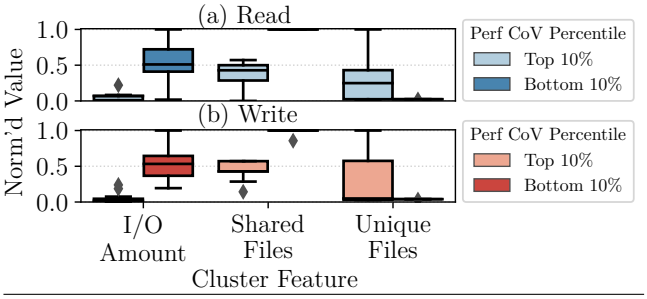

Figure 14: Along with the I/O amount, the number of shared and unique files used for I/O also affect the performance CoV of clusters.

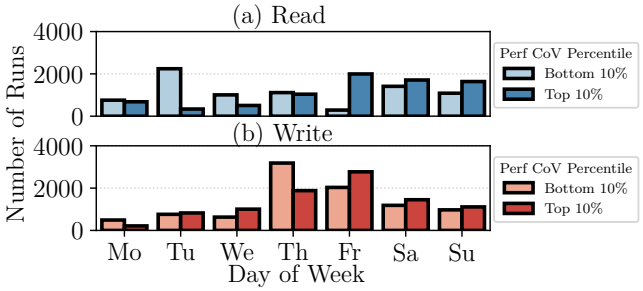

Figure 15: The runs in the read and write clusters that experience that highest variation in performance tend to run on the weekends.

with the increase in the time span and decrease with the increase in the I/O amount.

Implication. The I/O of applications that run repetitively, but perform a low amount of I/O, needs to be carefully managed by HPC centers in terms of performance variation mitigation. These applications are prone to observing the highest amount of variation and resulting in the most number of complaints. As a mitigation recommendation, the applications could be encouraged to perform their I/O to fewer storage targets and aggregate their I/O phases until they have a larger sum of I/O to perform (their computer and I/O phases might be too short).

The general performance variation characteristics across all clusters revealed interesting trends. To dig deeper, next we study the differences in behaviors of clusters with the highest and lowest performance variations to help discover the factors that affect performance variation.

Comparing Clusters with High and Low Performance Variation. To make the analysis simpler, we consider high and low and top 10% and bottom 10%, respectively. While the 10% threshold is configurable, we chose this threshold because it allowed us to perform our analysis on a statistically significant number of clusters and also capture sufficiently wide-behavior, without being affected by the clusters which show mediocre performance variation.

In particular, we consider all clusters from all applications and identify the 10% clusters that experience the most performance variations (top 10%) and the least performance variations (bottom 10%). *Note that we are purposely removing the application-user identifier from the clusters to understand if clusters exhibiting exceptionally high/low performance variation have temporal or spatial features that are common across applications.* We start our analysis by evaluating the general I/O characteristics of the clusters in the top and bottom 10% of performance CoVs. In particular, we ask:

RQ 6. How do I/O characteristics differ between clusters that observe highest and lowest performance variation?

Fig. 14 shows boxplots of I/O amount, number of shared files, and number of unique files of the high- and low-CoV clusters. We make several observations: (1) As expected, the top 10% high-CoV clusters have a much smaller I/O amount for both reads and writes, compared to the bottom 10% low-CoV clusters. This is consistent with our finding in Fig. 13, which shows that higher I/O amount results in lower performance CoV. (2) More surprisingly, low-CoV clusters tend to exclusively have only shared files (file is shared among all the ranks of the applications) and no unique files (each rank performs I/O to its own file). On the other hand, high-CoV clusters tend to read from many unique files. Having multiple unique files requires making a multitude of metadata requests to the metadata server, which tends to be a service bottleneck in the I/O pipeline as it is a single server shared across all files and applications. Thus, the number of files greatly affects the I/O performance CoV. Note that we also studied other I/O characteristics such as the number of different I/O requests sizes, but found no significant difference between high- and low-CoV clusters for those.

Lesson Learned 7. *On this production cluster, our methodology revealed that along with the I/O amount, I/O characteristics such as the number of shared and unique files can be useful in identifying applications with high performance variations.*

Implication. This finding emphasizes that, to encourage and cultivate best I/O practices, users need to be educated about the behavior that having fewer files can help deliver more stable performance. To get around this issue, one solution is to consolidate I/O data in one shared file, as opposed to having multiple unique files. Note that this does not necessarily lead to limiting the I/O parallelism as a shared file is still striped across multiple storage targets. However, there is an interesting trade-off between observed performance variation and file striping – that needs to be carefully considered by the HPC center operators.

Next, we examine the temporal characteristics of high-CoV and low-CoV clusters. In particular, we ask:

RQ 7. Is I/O performance variation correlated with day of the week, hour of the day, etc.?

Fig. 15 shows that higher performance variation is more likely to occur on the weekends, although not always. This trend is true for both read and write clusters. For example, the number of runs (read and write combined) belonging to the top 10% performance variation clusters on Fri-Sun is ≈ 11 thousand, while the same number is only ≈ 7 thousand for runs belonging to the bottom 10% performance variation clusters. Additionally, we observe that jobs launched during weekends do notably more I/O than during weekdays. This is because on many weekends, users tend to launch long-running I/O-intensive jobs to be finished during the weekend for post-processing/analysis in the following week. While not an uncommon behavior among many scientists, such behaviors can artificially induce more performance variation than normal. This is evident in

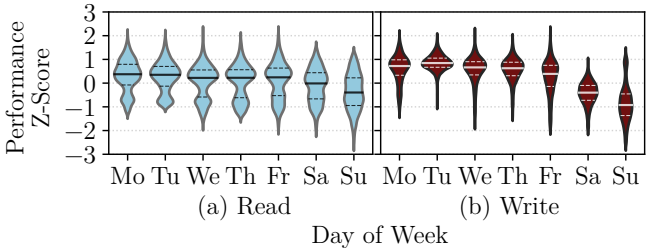

Figure 16: The median z-score of I/O performance of (a) read runs and (b) write runs decreases during the weekend days.

our data as the total amount of I/O increases an average of 150% on Saturday and Sunday.

The previous observation leads us to ask: *"how does the higher performance variation on weekends affect the average observed performance on weekends?"*. Surprisingly, we found that weekends also exhibit consistently lower average performance. In Figure 16, each run's performance is compared to runs within the same cluster by calculating the z-score of the performances in each cluster. The z-score is calculated on a per-cluster basis as the runs in each cluster have similar I/O behavior and use the same application. If a run has a low z-score, it indicates that performance is worse for that run in comparison to runs with similar I/O behaviors. We observe that runs tend to experience worse I/O performances when they are run on Fridays, Saturdays, or Sundays. This trend is especially strong on Sundays. In fact, for runs in write clusters, the median performance on Sundays is almost one standard deviation away from the average performance z-score of zero. We did a analysis for "time-of-the-day" and found that high CoV clusters and low CoV clusters do not exhibit different trends. On average, high CoV clusters are as likely to occur at a specific hour of day as low CoV clusters. Similarly, performance z-score distribution was similar across all hours of the day – suggesting that running an application during certain hours did not specifically make it more prone to lower performance.

Lesson Learned 8. *Our analysis reveals that clusters that run on weekends tend to observe some of the highest performance variations. This is correlated with the trend that general I/O performance is worse on the weekends than on the weekdays. We caution that this finding is likely to be tied to a user-base that tends to run more I/O intensive jobs on weekends. This might be true at other centers as well, but requires confirmation.*

Implication. HPC centers need to proactively identify such trends and, if present, they need to be mitigated by monitoring I/O intensive applications and providing incentive to users to run I/O intensive jobs on different days. Note that, in this particular case, the spacing out of I/O intensive jobs does not need to be calibrated at fine timescales as no temporal trend is observed on an hour-of-the-day basis, only on a day-of-the-week basis.

In addition to the strong day-of-the-week temporal trends that we observed, next we investigate if there are larger temporal trends during the entire time period of our study. For example,

RQ 8. Do we observe that clusters that were run during a specific two-month period have a high performance variation?

To perform this analysis, we again take the clusters that are in the top 10% and the bottom 10% when all clusters across all applications

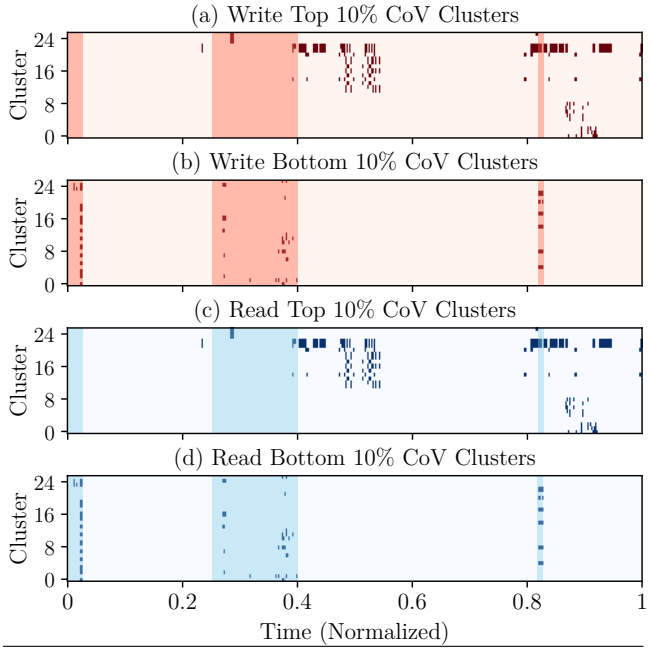

Figure 17: Temporal spectral of the runs belonging to the high and low performance CoV (a,b) read and (c,d) write clusters. The time periods during which the low-CoV runs are executed are largely disjoint from the periods during which the high-CoV runs are executed.

and users are sorted according to their performance variation. Fig. 17 shows the temporal spectral of the runs belonging to the high and low performance CoV (a) read and (b) write clusters. The x-axis shows the normalized chronological time from the start of our analysis period to the end. The y-axis shows the clusters in the top or bottom 10% CoV ranges. The dark dots represent the times during which the runs of the corresponding clusters were run. The shaded regions show the general periods during which the bottom 10% low-CoV runs were executed for read and write clusters. Note that the clusters belong to several different applications. For reads, the top 10% clusters are from applications `SpEC0` (cluster 0), `wrf0` (cluster 1), `QE1` (clusters 2-9), `QE0` (10-31), `mosst0` (32-43), and `vasp0` (44-49). The bottom 10% clusters are from `QE3` (0-1) and `vasp0` (2-49). For writes, the top 10% clusters are from applications `QE1` (clusters 0-2), `QE0` (3-10), `mosst0` (11-22), and `vasp0` (23-25). The bottom 10% clusters are all from `vasp0` (0-25). We make several observations.

The time periods during which the low-CoV runs are executed are different from the periods during which the high-CoV runs are executed. There are temporal zones when multiple applications experience high performance variation, for both read and write clusters. However, it is not straightforward to predict these temporal zones. For example, the `vasp0` application has read clusters in the top 10% performance CoVs (44-49), as well as the bottom 10% (2-49), and both types of clusters were run during different times, making it difficult to predict when the application will experience high performance variation. Nonetheless, it is possible to manage the I/O in the high and performance variation zones. One solution is to perform clustering so as to compute the base performance and detect variation from this base. This can open up opportunities for more effective I/O scheduling during high performance variation periods based on

congestion detection and control, i.e., using a reaction mechanism to detect performance variation and coordinate application I/O.

Fig. 17 also shows that the same application can have multiple incarnations (different clusters) that may be sensitive to performance variation differently. During the same time period, some incarnations experience significant variation, while other incarnations are not. The underlying reason is that different incarnations from the same application may have different I/O characteristics (e.g., amount of I/O, shared / unique files, etc.) and that may affect the degree of observed variation. Previous studies show that performance variation is a problem [15, 34, 40], but we are first one to show that the same application has incarnations with different sensitivity levels.

Lesson Learned 9. *Our analysis shows that there are separate and disjoint time zones during which different applications experience high and low performance variations. These zones are shared across different applications and clusters. However, not all applications experience a similar level of variation even during high performance variation zones. Our study shows that even a simple methodology using I/O monitoring data collected from Darshan can be useful in identifying these zones – without requiring additional system probing, benchmarking, high-overhead instrumentation. This can be achieved via (1) clustering applications based on their I/O behavior and (2) keeping track of their observed I/O performance (e.g., using Darshan). Keeping track of observed I/O performance helps us estimate the expected/reference I/O performance, and hence, hint at temporal zones with high variation.*

Implication. System administrators can leverage our clustering methodology to detect and manage periods of high performance variation without performing any additional instrumentation or probing and without necessarily requiring to deploy new high-overhead ML-based prediction models, although application of high-overhead ML models is useful and can further improve a HPC center's capability to detect variations [43, 47].

5 MISC. DISCUSSION

In this section, we discuss the the scope of our findings and system-specific factors that cannot be decoupled from our analysis.

System-specific environment and workloads. We acknowledge that our findings are ultimately a reflection of the system we are performing our analysis on. Anecdotally, while we believe that multiple other systems might share similar workload and user behavior, we do not claim that all findings are applicable as-is. Nevertheless, this study should encourage other centers to perform similar analysis and detect repetition/performance variation patterns. Our dominant workloads are widely-used and variants of those are run on other systems too. Note that the system under study uses a Lustre File System (LFS). Some of the findings related to performance variation might be related to LFS internals (e.g., metadata operation management, default striping) and should be tested on systems running different file systems (e.g., GPFS). However, we have ensured that file system updates and upgrades are not the source of consistent performance degradation during the course of our study.

In addition, we investigated the correlation between I/O variability and metadata operation intensity of a given application. For each run in each cluster, we collected the time spent on metadata and I/O performance then calculated the correlation between the two metrics.

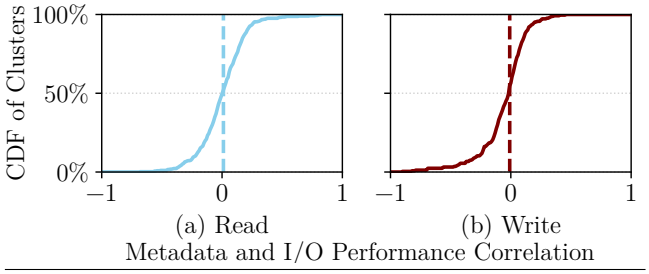

Figure 18: The Pearson correlation coefficients of the time spent on metadata and I/O performance per cluster shown as CDFs.

Fig. 18 shows the CDF of those correlation coefficients. We observe that the correlations are normally distributed around and the median coefficient is 0, which indicates there is a weak correlation between I/O variability and metadata intensity of the application, on average.

We note that more detailed server-side information is needed to better understand metadata and filesystem utilization correlations. For example, spatial OST-level load information is likely to exhibit better correlation. While we cannot establish such correlations, we caution that it is not a proof for non-existence. The lesson learned is that more fine-grained information is critical toward establishing such correlations, but such tools may sometimes impose unsuitable overhead on production systems.

Additional telemetry data and application-specific characteristics. We acknowledge that analyzing additional telemetry data and characteristics (e.g., data locality and migration) along with its interaction with the I/O variability could be useful in better understanding the repetitive execution patterns. Data movement patterns can be leveraged to reduce I/O variability caused via interference, or better coordinate I/O among multiple I/O-intensive applications. Some file systems (e.g., Ceph) may allow better migration capabilities for dynamic objects that can use additional telemetry data related to data locality and reuse access patterns.

Further, application domain (numerical simulation vs. AI applications) can have a significant impact on the observed repeatability and variance characteristics. This effect is already exhibited by our analysis where applications from different science domains exhibit different repeatability and variance characteristics. Emerging workloads such as deep learning training are not dominant I/O-resource consumers on this system. This is because most machine learning workloads are compute- and memory bandwidth-bound; they tend to cache the input training data and do not experience severe I/O bottlenecks after input fetching. However, but that is likely to change in the near future. As we adopt model-parallelism and bigger training models, these workloads will also experience I/O bottlenecks and their repeatability and variance characteristics should be compared with findings of this study.

Another important contributing factor is runtime configuration parameters. Runtime configuration parameters directly affect the observed repeatability and variance characteristics. Findings in our study hint that such trends already exist. For example, multiple application in our study (Vasp, Quantum Espresso, Weather Research and Forecasting Model) exhibits multiple different personalities (I/O behavior) and their repetitive patterns of such different behaviors also vary over time – potentially indicating the effect of configuration parameters that were used to run these applications during different

campaign runs by scientists. Including such telemetry data, although difficult to obtain and reason about accuracy, are valuable toward such analysis.

Post-hoc analysis. We acknowledge that our study is post-hoc in nature, so we cannot answer "what-if" questions (e.g., changing the schedule of applications, etc.). Nevertheless, our study demonstrates how practitioners can learn from our findings, applying similar analysis at their centers, and improve the state-of-practice. We also note that our performance variability analysis is fundamentally driven by our clustering methodology. In our approach, we implicitly assume that application runs with very similar I/O features and behavior should also observe similar I/O performance. While from our operational experience and user communications, we have found this expectation to be reasonable and valid in practice. However, in some cases, it might not be 100% accurate. Nevertheless, our study shows that rigorous and carefully designed clustering reduces such risks and indeed allows us to spot variations in observed I/O performance. We suspect that in practice, the variation might occur more frequently, but our methodology has made our estimates conservative.

Monitoring capability and granularity. We note that our analysis is limited by the granularity of the information provided by Darshan and other system monitoring tools. Due to overhead associated with production-level operations, these monitoring tools are not as intrusive as they could be in a non-production environment (e.g., latency for I/O operations in aggregated instead of recording latency for each I/O operation). Nevertheless, our analysis shows that one could draw useful operational insights and implications from these very low-overhead, production-suited monitoring / data collection. Also, we acknowledge that there are multiple other factors that might cause or be correlated with I/O variance (e.g., power-cooling conditions, co-scheduled applications, network congestion). However, instrumenting and collecting data from all sources to diagnose the problem in real-time on a production system is not practical. But, we show that even with production-suitable instrumentation, one can detect interesting application patterns and identify performance variation zones. We found that a Darshan-like tool can be leveraged to understand the varying I/O behavior of applications and an application's sensitivity toward I/O performance variability. However, there are a number of improvement areas (e.g., collecting more fine-grained phase-based information in Darshan and automatically performing clustering of applications). We will provide our developed toolset as a package so that other systems can leverage this tool in practice to gain insights, classify applications, and detect runs with high I/O performance variability. We are hopeful that this study paves the path for similar efforts at other centers and pushes the envelope of our monitoring capability on our HPC systems.

6 RELATED WORK

In this section, we discuss previous work relevant to our study.

Tools for I/O Characterization. I/O characterizations tools are increasingly deployed on HPC systems as analyzing I/O behavior is increasingly critical in understanding the health of a system. Several software tools have been proposed and developed to address (1) application-level [26, 30, 31, 53] and (2) system-level I/O monitoring [1, 20, 22, 28, 50]. The former may collect metrics on an individual application or collection of applications running on a system. In fact, some effort is being made to provide a framework to monitor I/O at both the system- and application-level [10, 25, 37]. TOKIO [25] is a notable framework that combines several of these tools, such as Darshan, the toolkit used in this study. Additionally, some of these frameworks propose combinations of I/O monitoring tools to provide a comprehensive understanding of high-performance systems [5].

Analysis of I/O. Characterization studies are typically done on (1) domain-specific workloads with application-level analysis and (2) system-specific HPC file system environment with application- and/or system-level analysis [6, 11, 12, 15, 19, 24, 29, 35, 40, 44, 45, 48, 49, 52]. The former gives insight to the I/O behavior of certain domains, such as machine learning, while the latter covers overall system I/O trends and characteristics such as I/O bandwidth, inter-arrival times of I/O requests and jobs, idle time, and I/O variability.

Implications of this Study. Works by Isakov et al. [16] and Pavan et al. [36] have used different grouping methodologies to predict I/O trends using additional machine learning techniques and to identify access patterns to take a retrospective approach in characterizing temporal and feature behaviors, respectively. In comparison, our study produced several findings with implications that will benefit future I/O bottleneck research and file system development. For example, a study by Koo et al. [21] proposes grouping I/O streams by users in burst buffers in order to improve scheduling policies and optimize I/O performance. Awareness of our findings can improve the stream-aware scheduling policy in a read and write-specific manner. Additionally, studies previously done by dividing jobs by only user application to analytically predict I/O performance, such as [18], might benefit by applying our clustering methodology and integrating relevant features identified by our study.

7 CONCLUSION

In this paper, we discussed a methodological framework and our experience in leveraging it for examining repetitive job behavior in large-scale file systems. Using this framework, we investigated several I/O patterns in a production HPC system and discussed the implications of our interesting findings. Our findings empower the development of new system features or policies to reduce I/O performance variation experienced by users running repetitive jobs. Our study proposes strategies to develop new features or policies to help reduce load imbalance and optimize I/O performance by being aware of workload I/O patterns used in an HPC system.

8 ACKNOWLEDGEMENTS

We thank the reviewers for their valuable comments that improved the paper. This research was supported in part by the U.S. Department of Energy and by Sandia National Laboratories [1] under contract No. 2055483. Any opinions, findings, and conclusions or recommendations ex-pressed in this material are those of the authors and do not necessarily reflect the views of SNL, DOE or the United States Government.

[1] Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. SAND : TO BE FILLED IN AFTER RA

REFERENCES

[1] 2018. Monitoring GPFS I/O performance with the mmpmon command. https://www.ibm.com/docs/en/spectrum-scale/5.0.0?topic=monitoring-gpfs-io-performance-mmpmon-command

[2] G. Aupy, A. Gainaru, V. Honoré, P. Raghavan, Y. Robert, and H. Sun. 2019. Reservation Strategies for Stochastic Jobs. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 166–175. https://doi.org/10.1109/IPDPS.2019.00027

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

[4] Babak Behzad, Surendra Byna, Prabhat, and Marc Snir. 2015. Pattern-Driven Parallel I/O Tuning. In *Proceedings of the 10th Parallel Data Storage Workshop* (Austin, Texas) *(PDSW '15)*. Association for Computing Machinery, New York, NY, USA, 43–48. https://doi.org/10.1145/2834976.2834977

[5] Eugen Betke and Julian Kunkel. 2017. Real-time I/O-monitoring of HPC applications with SIOX, elasticsearch, Grafana and FUSE. In *International Conference on High Performance Computing*. Springer, 174–186.

[6] Julian Borrill, Leonid Oliker, John Shalf, Hongzhang Shan, and Andrew Uselton. 2009. HPC global file system performance analysis using a scientific-application derived benchmark. *Parallel Comput.* 35, 6 (2009), 358–373.

[7] Louis-Claude Canon and Emmanuel Jeannot. 2009. Evaluation and optimization of the robustness of dag schedules in heterogeneous environments. *IEEE Transactions on Parallel and Distributed Systems* 21, 4 (2009), 532–546.

[8] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. 2011. Understanding and improving computational science storage access through continuous characterization. *ACM Transactions on Storage (TOS)* 7, 3 (2011), 1–26.

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

[10] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley. 2009. 24/7 Characterization of petascale I/O workloads. In *2009 IEEE International Conference on Cluster Computing and Workshops*. 1–10. https://doi.org/10.1109/CLUSTR.2009.5289150

[11] S. W. D. Chien, A. Podobas, I. B. Peng, and S. Markidis. 2020. tf-Darshan: Understanding Fine-grained I/O Performance in Machine Learning Workloads. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. 359–370. https://doi.org/10.1109/CLUSTER49012.2020.00046

[12] Devarshi Ghoshal, Brian Austin, Deborah Bard, Christopher Daley, Glenn Lockwood, Nicholas J Wright, and Lavanya Ramakrishnan. 2020. Characterizing Scientific Workflows on HPC Systems using Logs. In *2020 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*. IEEE, 57–64.

[13] Pilar Gomez-Sanchez, Sandra Mendez, Dolores Rexachs, and Emilio Luque. 2017. PIOM-PX: A Framework for Modeling the I/O Behavior of Parallel Scientific Applications. In *High Performance Computing*, Julian M. Kunkel, Rio Yokota, Michela Taufer, and John Shalf (Eds.). Springer International Publishing, Cham, 160–173.

[14] Jun He, John Bent, Aaron Torres, Gary Grider, Garth Gibson, Carlos Maltzahn, and Xian-He Sun. 2013. I/O acceleration with pattern detection. In *Proceedings of the 22nd international symposium on High-Performance Parallel and Distributed Computing*. 25–36.

[15] Dan Huang, Qing Liu, Jong Choi, Norbert Podhorszki, Scott Klasky, Jeremy Logan, George Ostrouchov, Xubin He, and Matthew Wolf. 2018. Can i/o variability be reduced on qos-less hpc storage systems? *IEEE Trans. Comput.* 68, 5 (2018), 631–645.

[16] M. Isakov, E. Rosario, S. Madireddy, P. Balaprakash, P. Carns, R. B. Ross, and M. A. Kinsy. 2020. HPC I/O Throughput Bottleneck Analysis with Explainable Local Models. In *2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, Los Alamitos, CA, USA, 1–13. https://doi.org/10.1109/SC41405.2020.00037

[17] Sunggon Kim, Alex Sim, Kesheng Wu, Suren Byna, Yongseok Son, and Hyeonsang Eom. 2020. Towards HPC I/O Performance Prediction through Large-Scale Log Analysis. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing* (Stockholm, Sweden) *(HPDC '20)*. Association for Computing Machinery, New York, NY, USA, 77–88. https://doi.org/10.1145/3369583.3392678

[18] Sunggon Kim, Alex Sim, Kesheng Wu, Suren Byna, Yongseok Son, and Hyeonsang Eom. 2020. Towards hpc i/o performance prediction through large-scale log analysis. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*. 77–88.

[19] Y. Kim, R. Gunasekaran, G. M. Shipman, D. A. Dillow, Zhe Zhang, and B. W. Settlemyer. 2010. Workload characterization of a leadership class storage cluster. In *2010 5th Petascale Data Storage Workshop (PDSW '10)*. 1–5. https://doi.org/10.1109/PDSW.2010.5668066

[20] Andreas Knüpfer, Christian Rössel, Dieter an Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen Malony, et al. 2012. Score-P: A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir. In *Tools for High Performance Computing 2011*. Springer, 79–91.

[21] Donghun Koo, Jaehwan Lee, Jialin Liu, Eun-Kyu Byun, Jae-Hyuck Kwak, Glenn K Lockwood, Soonwook Hwang, Katie Antypas, Kesheng Wu, and Hyeonsang Eom. 2021. An empirical study of I/O separation for burst buffers in HPC systems. *J. Parallel and Distrib. Comput.* 148 (2021), 96–108.

[22] Julian M Kunkel, Michaela Zimmer, Nathanael Hübbe, Alvaro Aguilera, Holger Mickler, Xuan Wang, Andriy Chut, Thomas Bönisch, Jakob Lüttgau, Roman Michel, et al. 2014. The SIOX architecture–coupling automatic monitoring and optimization of parallel I/O. In *International Supercomputing Conference*. Springer, 245–260.

[23] Wei Liu, Kai Wu, Jialin Liu, Feng Chen, and Dong Li. 2017. Performance evaluation and modeling of HPC I/O on non-volatile memory. In *2017 International Conference on Networking, Architecture, and Storage (NAS)*. IEEE, 1–10.

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

[25] Glenn K Lockwood, Nicholas J Wright, Shane Snyder, Philip Carns, George Brown, and Kevin Harms. 2018. *TOKIO on ClusterStor: connecting standard tools to enable holistic I/O performance analysis*. Technical Report. Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States).

[26] Glenn K Lockwood, Wucherl Yoo, Suren Byna, Nicholas J Wright, Shane Snyder, Kevin Harms, Zachary Nault, and Philip Carns. 2017. 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*. 55–60.

[27] Jakob Lüttgau, Shane Snyder, Philip Carns, Justin M Wozniak, Julian Kunkel, and Thomas Ludwig. 2018. Toward understanding I/O behavior in HPC workflows. In *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS)*. IEEE, 64–75.

[28] J. Lüttgau, S. Snyder, P. Carns, J. M. Wozniak, J. Kunkel, and T. Ludwig. 2018. Toward Understanding I/O Behavior in HPC Workflows. In *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage Data Intensive Scalable Computing Systems (PDSW-DISCS)*. 64–75. https://doi.org/10.1109/PDSW-DISCS.2018.00012

[29] Sandeep Madireddy, Prasanna Balaprakash, Philip Carns, Robert Latham, Robert Ross, Shane Snyder, and Stefan M Wild. 2017. Analysis and correlation of application I/O performance and system-wide I/O activity. In *2017 International Conference on Networking, Architecture, and Storage (NAS)*. IEEE, 1–10.

[30] HuangSandra Méndez, Javier Panadero, Alvaro Wong, Dolores Rexachs del Rosario, and Emilio Luque Fadón. 2012. A new approach for analyzing I/O in parallel scienti. In *XVIII Congreso Argentino de Ciencias de la Computación*.

[31] Sandra Méndez, Dolores Rexachs, and Emilio Luque. 2012. Modeling parallel scientific applications through their input/output phases. In *2012 IEEE International Conference on Cluster Computing Workshops*. IEEE, 7–15.

[32] Glenn W Milligan and Martha C Cooper. 1986. A study of the comparability of external criteria for hierarchical cluster analysis. *Multivariate behavioral research* 21, 4 (1986), 441–458.

[33] Lena Oden, Christian Schiffer, Hannah Spitzer, Timo Dickscheid, and Dirk Pleiter. 2019. IO challenges for human brain atlasing using deep learning methods-an in-depth analysis. In *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 291–298.

[34] Tirthak Patel, Suren Byna, Glenn K Lockwood, and Devesh Tiwari. 2019. Revisiting I/O behavior in large-scale storage systems: The expected and the unexpected. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–13.

[35] Tirthak Patel, Suren Byna, Glenn K Lockwood, Nicholas J Wright, Philip Carns, Robert Ross, and Devesh Tiwari. 2020. Uncovering access, reuse, and sharing characteristics of i/o-intensive files on large-scale production {HPC} systems. In *18th {USENIX} Conference on File and Storage Technologies ({FAST} 20)*. 91–101.

[36] P. J. Pavan, J. L. Bez, M. S. Serpa, F. Z. Boito, and P. O. A. Navaux. 2019. An Unsupervised Learning Approach for I/O Behavior Characterization. In *2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. 33–40. https://doi.org/10.1109/SBAC-PAD.2019.00019

[37] SC PDSW-DISC. [n.d.]. Toward Understanding I/O Behavior in HPC Workflows. ([n. d.]).

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[39] Sarunya Pumma, Min Si, Wu-chun Feng, and Pavan Balaji. 2017. Towards scalable deep learning via I/O analysis and optimization. In *2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on*

Data Science and Systems (HPCC/SmartCity/DSS). IEEE, 223–230.

[40] David Skinner and William Kramer. 2005. Understanding the causes of performance variability in HPC workloads. In *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.* IEEE, 137–149.

[41] Michael Steinbach, George Karypis, and Vipin Kumar. 2000. A comparison of document clustering techniques. (2000).

[42] H. Tang, S. Byna, S. Harenberg, X. Zou, W. Zhang, K. Wu, B. Dong, O. Rubel, K. Bouchard, S. Klasky, and N. F. Samatova. 2016. Usage Pattern-Driven Dynamic Data Layout Reorganization. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 356–365. https://doi.org/10.1109/CCGrid.2016.15

[43] Ozan Tuncer, Emre Ates, Yijia Zhang, Ata Turk, Jim Brandt, Vitus J. Leung, Manuel Egele, and Ayse K. Coskun. 2017. Diagnosing Performance Variations in HPC Applications Using Machine Learning. In *High Performance Computing*, Julian M. Kunkel, Rio Yokota, Pavan Balaji, and David Keyes (Eds.). Springer International Publishing, Cham, 355–373.

[44] Feng Wang, Qin Xin, Bo Hong, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Tyce T McLarty. 2004. File system workload analysis for large scale scientific computing applications. In *Proceedings of the 21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies*. 139–152.

[45] T. Wang, S. Byna, G. K. Lockwood, S. Snyder, P. Carns, S. Kim, and N. J. Wright. 2019. 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)*. 102–111. https://doi.org/10.1109/CCGRID.2019.00021

[46] Blue Waters. 2019. Darshan Data. Available from Blue Waters... https://bluewaters.ncsa.illinois.edu/data-setsDarshan.

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

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

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

[50] Bin Yang, Xu Ji, Xiaosong Ma, Xiyang Wang, Tianyu Zhang, Xiupeng Zhu, Nosayba El-Sayed, Haidong Lan, Yibo Yang, Jidong Zhai, et al. 2019. End-to-end I/O monitoring on a leading supercomputer. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*. 379–394.

[51] O. Yildiz, M. Dorier, S. Ibrahim, R. Ross, and G. Antoniu. 2016. On the Root Causes of Cross-Application I/O Interference in HPC Storage Systems. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 750–759. https://doi.org/10.1109/IPDPS.2016.50

[52] Orcun Yildiz, Matthieu Dorier, Shadi Ibrahim, Rob Ross, and Gabriel Antoniu. 2016. On the root causes of cross-application I/O interference in HPC storage systems. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 750–759.

[53] Wucherl Yoo, Michelle Koo, Yi Cao, Alex Sim, Peter Nugent, and Kesheng Wu. 2016. Performance Analysis Tool for HPC and Big Data Applications on Scientific Clusters. In *Conquering Big Data with High Performance Computing*. Springer, 139–161.