An Unsupervised Learning Approach for I/O Behavior Characterization

Pablo J Pavan, Jean Luca Bez, Matheus S Serpa, Francieli Zanon Boito, Philippe O.A. Navaux

An Unsupervised Learning Approach for I/O Behavior Characterization

Pablo J. Pavan[1], Jean Luca Bez[1], Matheus S. Serpa[1], Francieli Zanon Boito[2], Philippe O. A. Navaux[1]
[1]Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS) — Porto Alegre, Brazil
[2]Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, 38000 Grenoble, France
{pjpavan, jean.bez, msserpa, navaux}@inf.ufrgs.br, francieli.zanon-boito@inria.fr

***Abstract*—I/O operations are the bottleneck of several applications due to the difference between processing and data access speeds. Hence, understanding the I/O behavior is vital to find problems and propose solutions. Thus, identifying and characterizing the I/O access pattern is important, since it reflects directly on applications' performance. With this premise, we propose an I/O characterization approach that uses unsupervised learning to cluster jobs with similar I/O behavior, using information from high-level aggregated traces. As a case study, we apply our approach on four months of activity — a total of $28,938$ jobs — from the Intrepid supercomputer located at Argonne Laboratory. Our experimental results show that nine access patterns represent the I/O behavior in 73% of the clusters. From these nine patterns, we learn some aspects about the I/O such as the most accesses patterns are made using POSIX and small requests, also, the most patterns are accessing unique files. Lastly, analyzing the I/O workload over four months, we can notice that it is composed by several applications that spend a short time on I/O activity, but when compared to the others, the total I/O time represents a greater portion of the overall system.**

***Index Terms*—Application I/O Behavior, Parallel I/O, I/O workload characterization, unsupervised learning, clustering algorithms**

I. Introduction

I/O operations in High-Performance Computing (HPC) systems are made to a shared Parallel File System (PFS), where dedicated machines act as servers [1], [2]. These I/O operations are a bottleneck for a growing number of applications due to the difference between the processing speed and the data access speed [3]. The way those applications access their data is characterized by the I/O access patterns and are directly related to the overall applications' performance.

Features commonly considered in the literature when characterizing the access pattern are the size of the requests, the operation type (write or read) and the spatiality of requests (contiguous or 1D-strided access) [4]. The access pattern represents the way an application access its data, and it has a direct impact on I/O performance. Therefore, it must be taken into account when developing applications and configuring the file system. Consequently, identifying and characterizing the I/O behavior is important, as it can be used by HPC developers to improve current and future applications' performance [5], [6].

It is possible to investigate I/O operations by using profiles generated by specialized tools. These tools intercept all the I/O events to gather information. Usually, the data provided by them contains only aggregated information regarding the job's I/O requests as fine-grained tracing often imposes a burdensome overhead. Despite using aggregation, large scale systems still generate huge amounts of data. One way to handle this data to extract knowledge about the I/O operations is by using unsupervised learning.

Unsupervised learning is a subdomain of machine learning that discovers unknown patterns from data. Clustering algorithms, dimensionality reduction, and density estimation are examples of this technique [7], [8]. These algorithms can assist in I/O characterization by summarizing vast amounts of data in useful knowledge. For instance, clustering algorithms can be used to group elements of the dataset that are more similar to each other, resulting in different classes. Additionally, no prior information about the dataset is required [9], [10]. Therefore, such an approach can be used to attain an overview of the I/O workload.

In this way, this paper describes an approach for I/O workload characterization on supercomputers. We use an unsupervised learning technique — the K-means clustering algorithm — to identify and characterize the main I/O behaviors observed in a machine over time. Our approach uses high-level application traces, that contain aggregated I/O statistics, from the Intrepid Blue Gene/P supercomputer at Argonne, USA. The information obtained from such characterization can guide researchers to develop future I/O optimizations directly on applications or on runtime systems. Furthermore, it can provide researchers a better idea of what a real-life scenario looks like, allowing them to evaluate their proposed techniques, ensuring that tests are covering patterns truly relevant to supercomputers.

We follow a reproducible and open methodology in our investigation. The companion material of this work is publicly available at https://gitlab.com/pjpavan/sbac-2019 and it contains all the code and analysis of this paper.

The remainder of this paper is organized as follows. Section II discuss the Related work. Section III describes the workflow to identify the I/O phases from the jobs. Section IV presents the approach to cluster similar jobs' I/O behavior using unsupervised learning. The case study of data from the Intrepid machine is demonstrated in Section V. Section VI concludes this paper, summarizing our findings and pointing future work perspectives.

II. Related Work

In the HPC context, a wide range of factors can negatively impact I/O performance. These factors were evaluated in several studies as in [11], [12], [13], [14], [15]. Moreover, understanding and characterizing a platform can provide extra insights on how the applications should perform I/O operations to obtain the best performance and thus, mitigate these negative factors.

Zoll et al. [16] studied a set of application-side I/O traces from the cluster of the Advanced School for Computing and Imaging (ASCI). Their traces were obtained with the *strace* tool, ranged from tens of seconds to half an hour, and included two scientific applications and three benchmarks generated with IOR benchmark. They concluded that a Markov model could not represent the request arrival rate of applications' I/O streams because they present self-similarity. They proposed a stochastic model to predict I/O arrival rate. Wang et al. [17] studied request size behavior from the same traces and showed that the most applications performed large numbers of small requests (from a few bytes to 1 MB) in small time intervals.

Kim et al. [18], [19] characterized the scientific workload of the Spider (Lustre) HPC storage cluster from Oak Ridge Leadership Computing Facility (OLCF). They considered the system utilization, the demands of read and write operations, idle time, and the distribution of read requests to write requests. The study demonstrated that the bandwidth usage and the inter-arrival time of requests could be modeled as a Pareto distribution.

To motivate their work on cross-application coordination, Dorier et al. [20] used data from the Parallel Workload Archive, from the period between January and September 2009. Through a simple, optimistic model, they used the distribution of some concurrent jobs to show that there was a high probability of having multiple applications concurrently performing I/O operations, even when applications spent as little as 5% of their execution time on I/O.

Luu et al. [21] analyzed the Darshan logs of a million jobs executed during 2013 over two supercomputers. They demonstrated that almost a third of the jobs had an aggregated throughput of no more than 256 MB/s. Additionally, despite the existence of high-level parallel libraries, three-quarters of the jobs used only POSIX to perform I/O.

Liu et al. [22] proposed AID (Automatic I/O Diverter) a system that performs automatic application I/O characterization and I/O-aware job scheduling. AID analyzes existing I/O traffic and batch job history logs, without any prior knowledge on applications or user/developer involvement. They evaluated AID on Titan supercomputer, using real applications, and they confirmed that AID is able to (1) identify I/O-intensive applications and their detailed I/O characteristics, and (2) significantly reduce these applications' I/O performance degradation/variance by jointly evaluating outstanding applications' I/O pattern and real-time system l/O load.

Xie et al. [23], the authors presented a characterization of the storage performance of the Cray XK6 Jaguar supercomputer while examining the implications of those results for application performance. They observed and quantified limitations from competing traffic, concurrency, interference and stragglers of write operations on shared files.

Similarly to the related work discussed so far, we seek to better understand the HPC application's behavior from their I/O operations. Differently from Zoll et al. [16] and Wang et al. [17], we used as a basis for our analysis the traces obtained transparently by the Darshan tool. Also, we present an approach about the I/O characterization, not only a analyze about the data.

On the other hand, some studies used clustering techniques to understand the applications' behavior, but not with I/O scope. Mishra et al. [24] clustered similar applications executed over Google Cloud Backend using K-means algorithm. They show that the tasks execution time is bimodal in that tasks either have a short duration or a long duration, most tasks have a short duration, and most resources are consumed by a few tasks with a long duration that has large demands for CPU and memory.

Similar to the aforementioned work, Di et al. [25] characterized Google applications, based on a one-month Google trace with over $650k$ jobs running across over 12000 heterogeneous hosts. They classify applications using K-means and shows the number of applications in the clustering sets follows a Pareto-similar distribution.

Calzarossa et al. [26] presents a survey of workload characterization with a focus on web workloads as well as on the workloads associated with online social networks, video services, mobile apps, and cloud computing infrastructures. They discussed the peculiarities of these workloads and presented the methodological approaches like clustering and modeling techniques applied for their characterization.

As these studies, we used unsupervised learning to clustering similar jobs. However, we use this approach to clustering jobs with similar I/O behavior and characterized them by access pattern using I/O phases concept.

III. Identifying I/O phases

In this paper, we consider the access pattern as characterized by the interface, type of operation, spatiality, and request size, all reported by Darshan I/O profiling tool [27]. Darshan describes a representation of the job's I/O behavior, reporting aggregated I/O statistics such as the access pattern, time and number of operations, among other information.

Because we are using aggregated information about the jobs, such as average request sizes and total amount of accessed data, it is not straightforward to split the execution into temporal I/O phases described by their access patterns. That is the case because the temporal behavior is partially lost when aggregating statistics about the execution. However, generating finer-grained traces is not considered to be feasible in practice, due to higher imposed overhead to applications and large amount of data being used for traces.

We use the concept of **I/O phases** to identify intervals where I/O operations are made using a particular access pattern. Since we are using the timestamps reported by Darshan, which

TABLE I
AVAILABLE FEATURES ABOUT THE I/O PHASES REPORTED BY DARSHAN

Interfaces	Operations	Spatiality	Request-size
MPI-IO	read/write	collective/indepedent unique/shared	yes
POSIX	read/write	consecutive/sequential unique/shared	yes
STDIO	read/write	unique/shared	no

express the time of the first and the last operations of a given pattern, we cannot assure that throughout that entire period I/O operations are indeed happening, but we can be sure that if any I/O operations are happening those patterns will characterize them. Table I details the features collected by Darshan regarding the I/O phases.

The available interfaces in our characterization of the access pattern are MPI-IO, POSIX, and STDIO. For both, read and write operations can be performed. For the MPI-IO's operations of access, spatiality can be collective or independent, accessing a unique or shared file. For POSIX, the spatiality can be consecutive or sequential accesses; accessing a unique file or a shared file. The difference between consecutive and sequential accesses is that consecutive occur when multiple accesses are immediately adjacent, while sequential happen at a higher offset than all the previous accesses. For STDIO, the spatiality of access only features to access to a single file or a shared file. Lastly, the request-size are available to MPI-IO and POSIX interfaces.

In order to extract the I/O phases, we used data from different jobs, collected using the Darshan tool that stores it in a binary format. In order to extract and transform the necessary data for our analysis, we used the `darshan-parser` tool, available with Darshan. It transforms the binary file into a text file, which contains all the counters related to the I/O operations collected by the profiler.

The text-format file generated by the `darshan-parser` is organized into two sections as Fig. 1 depicts. At the beginning of the file, stored in key-value pairs, there is data describing the execution: application name, job identifier, and the execution time. The remainder of the file contains the measurements of each counter as captured by Darshan, relative to each file opened by the application. Each line presents a tabular format. The counters represent the number of operations (read, write, seek, stat), using multiple interfaces (POSIX, MPI-IO, STDIO), histograms of access sizes, and cumulative time spent in each operation.

We implemented a script in Python to extract the counters relevant to our analysis and save them in a compressed JSON format. Those counters are in two classes: the first class refers to timestamp counters, these report the opening and closing times for each file, the number of operations of a given type and interface, and the cumulative time spent in I/O operations. The second class is performance counters, these indicate the fastest and slowest rank to execute a given operation, and the size of those accesses. The request sizes are available in bins, whereas only the four most common access sizes and the number of times they were observed are presented in exact values.

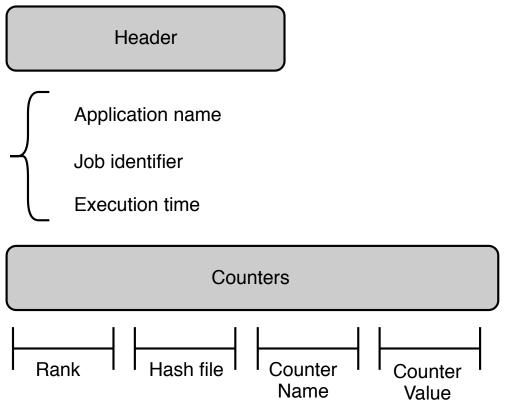

Fig. 1. Text-format file generated by the darshan-parser

Using the previously extracted information saved in compressed JSON, we collect each access pattern detected for each job, with start and end times, and save it in a CSV file. At this point, we have all the initially extracted I/O phases of each job, with their access pattern characterization. As these phases can overlap in time, we use the GenomicRanges [28] R library to combine overlapping periods into new phases, characterized by the union of their concurrent behaviors.

Fig. 2 shows the workflow to convert the Darshan binary file to the I/O phases. As explained in this section, there were several steps to arrive at the final result of the I/O phases. Each step was an important choice, either to reduce storage space or to increase the speed of data processing.

The final result of the I/O phases identification workflow can be seen in Fig. 3. It depicts the first 250 seconds of the execution of a job, with its I/O phases characterized with the approach described so far. It is possible to see intervals that represent different combinations of access patterns. For instance, the three I/O phases in purple demonstrate that during

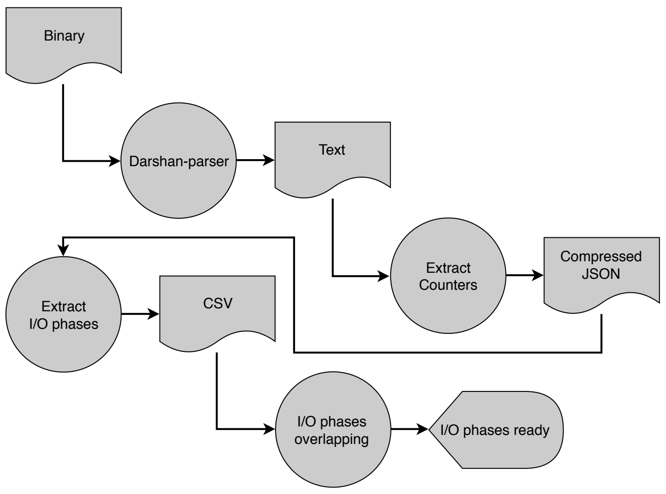

Fig. 2. Workflow to convert the Darshan binary file to the I/O phases

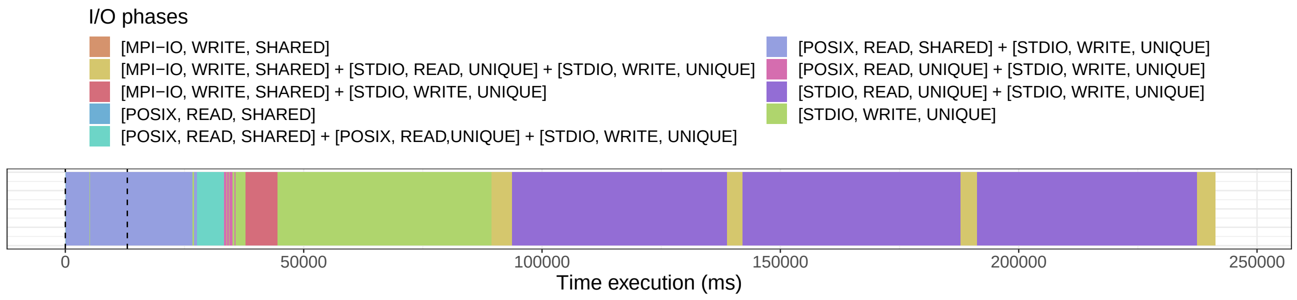

Fig. 3. Overview of I/O phases from a job.

that period, it was observed read and write operations to individual files using the STDIO interface. Furthermore, that access pattern repeats itself and appears in equally spaced intervals interleaved with additional MPI-IO write operations to a shared file, in yellow intervals.

IV. Unsupervised Learning Approach

The previous section described the methodology we used to identify and characterize each job by its I/O phases. Similarly, the entire supercomputer I/O workload can be characterized using such an approach, by applying it to each job individually. To be able to handle a massive amount of jobs, we apply an unsupervised learning technique to group the jobs that present similar behavior. The resultant groups are the result of a clustering algorithm; in our case, we employ K-means. The K-means algorithm aims to partition M data points in N dimensions into K clusters, seeking to minimize the within-cluster sum of squares (WSS) [29]. In our approach, each job with similar I/O phases is allocated to the closest cluster. The distance between each observation and a cluster is calculated using the Euclidean distance between the observation and the cluster's center. After each iteration, each center is updated as the mean of observations that belong to that cluster.

In our characterization method, M is the number of jobs, and N represents all the distinct phases detected in all the jobs considered in our analysis. In order to decrease the number of considered I/O phases, we characterized each job by its four most representatives I/O phases, which are the ones that account for most of its time spent in I/O. These four I/O phases represent about 90% of I/O time. The K parameter, the number of clusters, is found using the Elbow[30] and Silhouette [31], [32] methods. To create the $M \times N$ matrix, we match the jobs with their phases, where each cell accounts for the percentage each I/O phase represents of the time spent in I/O by that job. We used the K-means implementation from the Scikit-learn [33] library for Python 3.

The Elbow method shows the distortion for each K tested. This distortion is a result of the cost function that is the sum of the minimum values from the Euclidean distance between each value from M and the centroid of each cluster, divided by the number of observations. When the distortion makes an elbow shape in the chart, that indicates the optimal K, as adding more centroids would not reduce the distortion in a way to justify adding those new centroids.

The Silhouette reveals how close each observation in one cluster is to observations in a neighboring cluster. The Silhouette coefficient is measured in the range $[-1, 1]$. When this coefficient is near 1, it indicates that the observations are far away from a neighboring cluster. On the other hand, when this value is 0, it indicates that the observations are on or very close to the decision boundary between two neighboring clusters. Negative values indicate that those observations might have been assigned to the wrong cluster.

V. Case Study: I/O Characterization of the Intrepid Supercomputer

As a proof-of-concept of our proposal, we apply the method described in Section IV, to characterize the I/O workload of Intrepid Blue Gene/P supercomputer. This machine had its I/O profiled by Darshan, and all the data is available in the ALCF I/O Data Repository[1]. The repository contains a collection of anonymized log files from jobs executing scientific applications.

Our analysis uses data from four consecutive months (April to July 2012). This raw dataset has a total of 22 GB. After the process of extracting the I/O phases and clustering the jobs, we reduced this data to 66.8 MB. Table II details the number of jobs and unique I/O phases detected in each month. The unique phases are the N dimension used in K-means, and the jobs represent the M points.

We consider each month separately, aiming at detecting transient patterns and repeated ones. For each month, we applied the K-means algorithm with K ranging from 3 to 20, as we explained in Section IV. The Elbow and Silhouette methods were used to seek K value. Fig. 4 depicts the

[1]https://www.mcs.anl.gov/research/projects/darshan/data/

TABLE II
Number of jobs and unique phases for each studied month of 2012.

	April	May	June	July
Jobs	7275	8539	7107	6017
Unique Phases	542	607	594	421

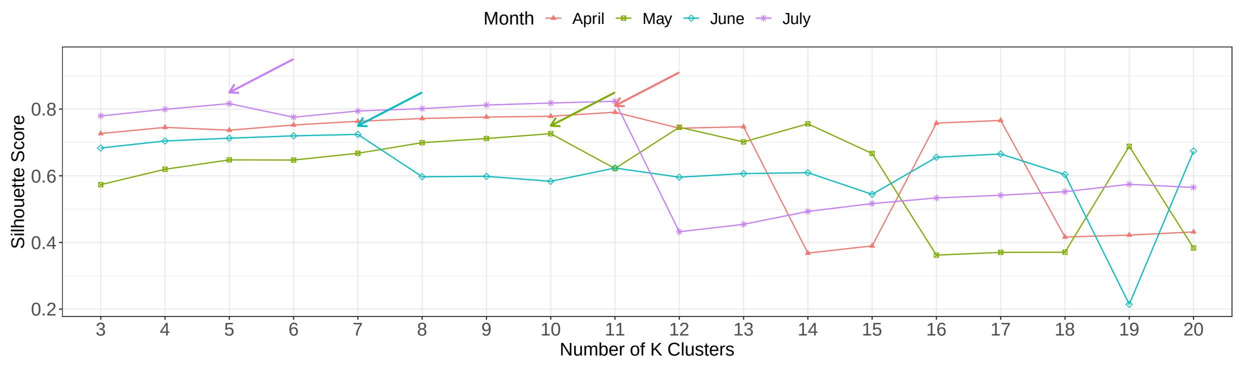

Fig. 4. Results of silhouette score in Intrepid dataset.

silhouette scores, a simplified result of the Silhouette method. Each color represents a month. The x-axis represents the values of K, and the y-axis the score. Each arrow indicates the K chosen for each month as the best distribution between the clusters, i.e., 11, 10, 7, and 5. The closer to 1 the score of the tested number of clusters is, the better the jobs' distribution. But the choice of K that best represents the distribution in the case of Silhouette does not always present the best in the case of Elbow. Therefore, these K values are chosen from a visual analysis over the Elbow and Silhouette plots results. The **extra information** about the clustering in each month, included the Elbow and Silhouette plots and a table containing a summarize these results, can be accessed in the companion repository.

It is possible to notice that each month had a different value for K. Such difference illustrates that there are different I/O behaviors in each month and that there is not a unique value for K that could be used for all the months. Hence, the I/O behavior can change over a period of time.

After the execution of K-means for each month, we seek to characterize each cluster by its main I/O phase. We selected from each cluster the most representative phase by taking the highest average percentage of time spent in I/O. That resulted in 33 phases, one per generated cluster. From these, 24 clusters are characterized by only nine phases.

Table III details these nine I/O phases that are representative for more than one month. The first column presents a label for the phase characterization. The second column describes the characterization, and the last one presents the months when each phase was detected. We selected these nine phases because they occur in more of a cluster through the months, and that are the representative phases of the machine's workload. That means they would be a promising focus for optimization techniques.

From these nine phases, some aspects of the machine's I/O workload can be pointed. For instance, most of the representative patterns use the POSIX interface, the maximum observed request size is of 10 MB, and the minimum is close to zero bytes. Also, these representative patterns are most often to unique than to shared files.

Table IV shows the clustering results over the nine I/O phases in each month. The **Cluster** column shows the cluster label in the K-means' results. The column **Phase Characterization** following the same notation as in Table III. The **I/O percentage** column represents how many I/O time was spent by the jobs. The column **Jobs** indicates the number of jobs in that cluster, and the last column **Phases** shows the remain I/O phases that exist in the group.

We can notice that the cluster 0 in every month is characterized by **A**, these clusters have, on average 2.41% of the execution time spent in I/O.We can also notice a high number of jobs, and a high number of I/O phases that are part of this cluster but are least representative. These least representative I/O phases spend less I/O than the representative phases. Ultimately, these clusters represent the jobs that made fewer I/O operations.

The **A** I/O phase occurred one more time in the cluster 5 in April. In this case, 104 jobs were characterized by it; they spend 69.52% of the time performing this access pattern. More than one cluster of the same month with the same I/O phases characterization occurs because the most representative phase of each cluster is the one that accounts for most of its time spent in I/O. In these situations, two or more clusters are represented by the same phase, but with very different

TABLE III
NINE I/O PHASES AND THEIR OCCURRENCES IN THE FOUR MONTHS OF THIS STUDY.

	Phase Characterization	Month
A	`POSIX, FWRITE, CONSECUTIVE, UNIQUE, 0-100`	April – July
B	`POSIX, WRITE, UNIQUE, (1K-10K | 100K-1M)`	April – May
C	`POSIX, WRITE, SEQUENTIAL, UNIQUE, 0-1K`	April – May
D	`POSIX, WRITE, SEQUENTIAL, UNIQUE, 0-100`	April – June
E	`MPI-IO, WRITE, SHARED, 4M-10M`	April – June
F	`MPI-IO, WRITE, SHARED, 1M-4M`	April – July
G	`POSIX, WRITE, SHARED, 0-100`	April, June

H	`POSIX, FREAD SHARED, 10K-1M`
	`POSIX, READ, SHARED, 10K-1M`
	`POSIX, WRITE, CONSECUTIVE UNIQUE, 0-100`
	`POSIX, WRITE, UNIQUE, 10K-100K`
		May

I	`POSIX, READ, CONSECUTIVE, UNIQUE, 1K-100K`
	`POSIX, READ, SHARED, 1K-1M`
	`POSIX, WRITE, CONSECUTIVE, UNIQUE, 0-100K`
	`POSIX, WRITE, UNIQUE, 10K-100K`
		May – July

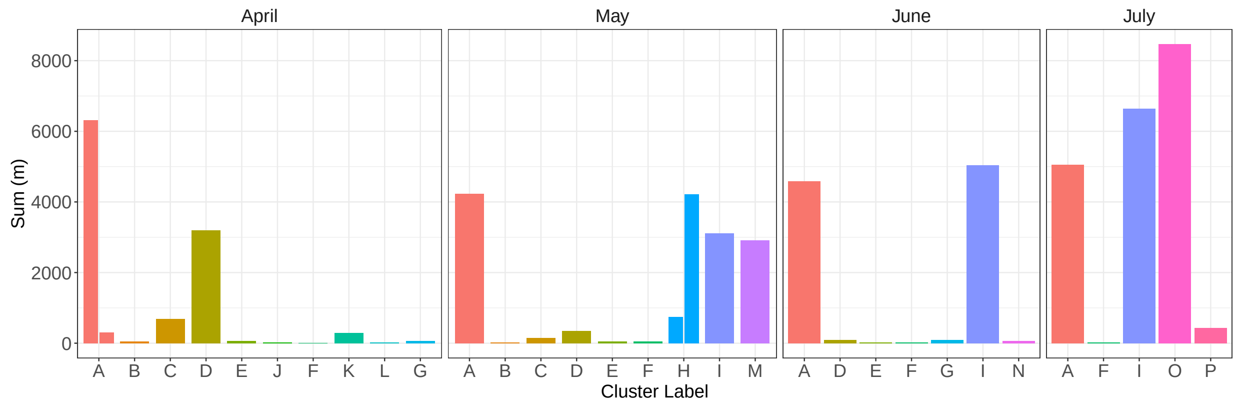

Fig. 5. Total time spent in I/O by each group of jobs of each month.

amounts of time spent in I/O by these phases. It happened twice for this dataset where two clusters of the same month were represented for the same I/O phase (**A** in April and **H** in May).

The number of jobs in each month follows a similar distribution: where each month has one cluster with the most number of jobs. These clusters represent 73.34%, 65.12%, $91,13\%$, and 95.92% of the total jobs in April, May, June, and July, respectively. Similar to the distribution of jobs, these clusters also have a high number of least representative I/O phases. We can notice that the jobs whose I/O behavior is not detected in other executions ended up grouped in this cluster.

The clusters characterized by **H** and **I** have a singular I/O phase characteristic. The singular I/O phases are the ones that have specific I/O characteristics when they are compared against another I/O phases. The singular I/O phases, in this case, includes four I/O access pattern that made reads and writes at the same time, this behavior occurs because these clusters contain only jobs from the same applications.

A. *I/O workload analysis by month*

The I/O workload was analyzed by month using the characterization of each cluster. To understand the distribution of the workload, we used the sum of the I/O time from each job present in each cluster. It provides an I/O overview by month. Fig. 5 shows the sum of the time spent in I/O by all jobs in each clusters obtained for each month. The x-axis shows the clusters represented by their most representative I/O phases, following the same notation as in Table III.

Among the identified behaviors, two phases (**A** and **F**) were representative in all studied months. Groups represented by **A** account for 59%, 26%, 46%, and 24% of the I/O time throughout the four months. On the other hand, **F** accounts less than 0.5% in each month.

Other relevant phases appear throughout the months. For instance, in April, group **D** with 29% of the I/O time. In May there are three groups **H**, **I**, and **M** with 30%, 19% and 18% respectively. In June, cluster **I** represents 50% of the accesses. In the last month, there are two clusters, **I** with 32%, and cluster **O**, with 41% of the I/O time. Table V, details the characterization of **M** and **O** phases, observed only in May and June, respectively.

M and **O** phases are a singular I/O characteristic like **H** and **I** phases. Also, in the same way, **M** and **O** are phases that occurred in a single application, i.e., these I/O phases represent I/O accesses from two applications, one of each month.

Fig. 6 shows the average of the time spent in I/O by all jobs in each clusters obtained for each month. The x-axis shows the clusters represented by their most representative I/O phases, following the same notation as in Table III. When we look at the average time spent in I/O by different groups, we have more information about these phases. Pattern **A** has an average

TABLE IV
CLUSTERING RESULTS OF EACH MONTH

Month	Cluster	Phase Characterization	I/O (%)	Jobs	Phases
April	0	**A**	2.89	5336	482
	1	**B**	80.65	1304	4
	2	**C**	94.77	88	15
	3	**D**	76.35	66	27
	4	**E**	99.31	168	3
	5	**A**	69.52	104	6
	7	**F**	99.80	32	3
	10	**G**	89.00	26	26
May	0	**A**	2.60	5561	570
	1	**H**	92.99	928	17
	2	**D**	89.02	192	29
	3	**B**	74.84	797	4
	4	**E**	99.20	144	3
	5	**F**	99.90	246	3
	6	**I**	90.08	79	4
	7	**C**	94.06	50	16
	8	**H**	51.11	356	13
June	0	**A**	1.49	6477	568
	1	**D**	88.80	315	29
	2	**I**	90.02	128	4
	3	**G**	98.83	26	8
	4	**F**	99.90	96	3
	5	**E**	99.17	48	3
July	0	**A**	2.69	5572	403
	2	**F**	99.90	128	3
	2	**I**	90.23	173	5

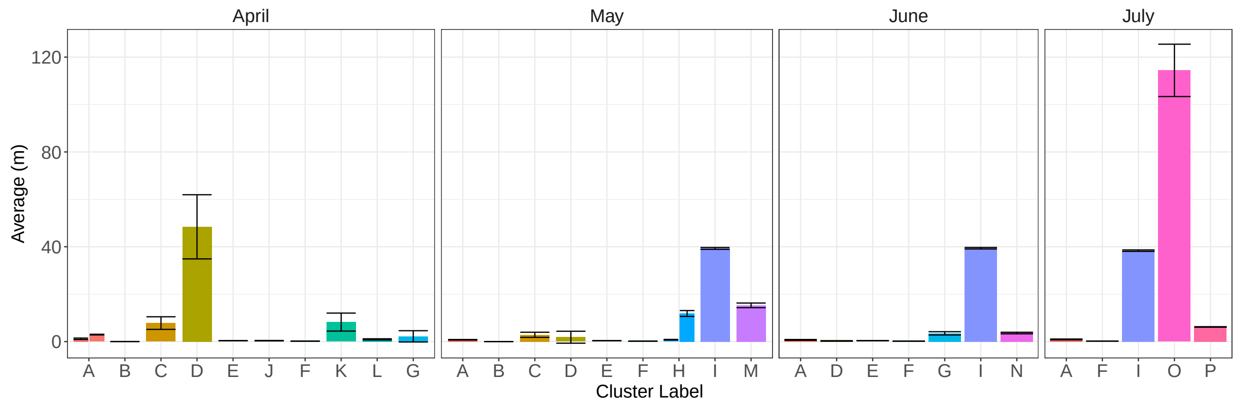

Fig. 6. Average time spent in I/O by each group of jobs of each month.

I/O time of only 2.41 seconds, despite being a representative phase in all months. It means that **A** phases tend to be very short, but this pattern is still representative because it happens in a large number of jobs (e.g. 5,336 jobs in April). In contrast, phases **D**, **I** and **O** have long average I/O times but happened for a smaller number of jobs (for instance, the group **I** contains 79, 128 and 173 jobs in May, June, and July).

If we interesting in apply optimizations to improve the performance of **A** phase in the system, our analysis points that individual applications would not observe large performance improvements. However, since these phases account for a large portion of the time spent in I/O in the machine, the small performance improvements would accumulate over time. However, optimizations targeting phases **D**, **I**, and **H** would impact applications performance more and system performance less.

We applied our proposed approach to a dataset from four months of activity of the Intrepid machine. By identifying jobs by their most representative I/O phases and clustering jobs with similar I/O behavior, we identified access patterns that are representative of the machine's I/O workload. From 33 clusters, 24 are represented by only nine access patterns. The characterization of these patterns shows that most accesses were done through POSIX with small requests. These patterns give potential targets for optimization techniques. Moreover, the analysis of the total and average time spent in I/O with different patterns paint a picture of the behavior of applications in this system, and of the improvements one could observe by applying optimizations.

TABLE V
THE REPRESENTATIVE PHASES CHARACTERIZATION IN MAY AND JULY.

	Phase Characterization	Month

M	`POSIX, READ, CONSECUTIVE, UNIQUE, 10K-100K`
	`POSIX, READ, SHARED, 1K-1M`
	`POSIX, WRITE, CONSECUTIVE, UNIQUE, 0-100K`
	`POSIX, WRITE, SEQUENTIAL, UNIQUE, 10K-100K`
		May

O	`MPI-IO, READ, INDEPENDENT, SHARED, 0-1K`
	`MPI-IO, WRITE, INDEPENDENT, SHARED, 0-100`
	`POSIX, READ, CONSECUTIVE, SHARED, 0K-100`
	`POSIX, READ, SHARED, 101-1k`
	`POSIX, WRITE, CONSECUTIVE, SHARED, 0-100`
		Jule

VI. Conclusion and Future Work

The I/O behavior is a significant aspect of HPC applications' overall performance. Furthermore, I/O is the performance limiting factor for many parallel applications. Hence, it is crucial to understand the I/O workload of real applications on HPC platforms. In this paper, we proposed a strategy for the I/O workload characterization of supercomputers applications using unsupervised learning. The characterization of the jobs I/O behavior is constructed using execution traces. These traces consist of aggregated events of different internal job operations that are manipulated to extract each one of the I/O phases. Next, a clustering algorithm (K-means) uses these data to group jobs of similar behavior, providing a characterization of the supercomputer job I/O. We used a larger dataset — four months of activity in the Intrepid supercomputer, 28.938 jobs — as a case study for our approach.

We identified nine access patterns that alone represent 24 out of 33 clusters; thus, these nine patterns represent 72.2% of I/O behavior in the clusters. The characterization gives us general insights about the machine I/O workload. For instance, we learn that most of these patterns use POSIX and small requests. Together with the information of total and average I/O time spent by the cluster of jobs, we can have an idea of the improvements one could expect by optimization techniques (and what patterns these techniques should target). Also, we can notice that it is composed by several applications that spend a short time on I/O activity, but when compared to the others, the total I/O time represents a greater portion of the overall system.

Another interesting point found using our approach was the reduction of the storage space needed to store the system workload information. Therefore, after running the approach, we reduced from 22GB of raw data to 66.8MB of I/O workload data ready to statistics analysis.

As we followed a reproducible and open methodology in our investigation. The companion material of this work is publicly available at https://gitlab.com/pjpavan/sbac-2019 and it contains all the code and analysis of this paper.

In addition to providing information to guide system improvements, such an analysis can provide valuable information to researchers from the parallel I/O field. Notably, knowing what are the representative patterns allowsfor an adequate evaluation of new proposed techniques, and is also useful for simulation of HPC systems. Future work includes the use of this approach in other data sets and the exploration of new clustering algorithms for the characterization.

Acknowledgment

This research received funding from the Petrobras project, grant n. 2016/00133-9. It was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brasil, and the LICIA International Laboratory. This research received funding from the European Unions Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 800144. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. The authors would like to thank Lucas L. Nesi, from UFRGS, for his valuable feedback about the paper.

This project is partially funded by the European Union.

References

[1] R. B. Ross, R. Thakur *et al.*, "Pvfs: A parallel file system for linux clusters," in *Proceedings of the 4th annual Linux showcase and conference*, 2000, pp. 391–430.

[2] S. Microsystems. (2007) LUSTRE file system - high-performance storage architecture and scalable cluster file system. [Online]. Available: http://science.energy.gov//media/ascr/ascac/pdf/reports/Exascalesubcommitteereport.pdf

[3] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, vol. 15, 2008.

[4] F. Z. Boito and et al., "A checkpoint of research on parallel I/O for high-performance computing," *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, p. 23, 2018.

[5] S. Li and et al., "A flattened metadata service for distributed file systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 12, pp. 2641–2657, 2018.

[6] Y. Tsujita and et al., "Improving collective MPI-IO using topology-aware stepwise data aggregation with I/O throttling," in *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*. ACM, 2018, pp. 12–23.

[7] M. Caron and et al., "Deep clustering for unsupervised learning of visual features," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 132–149.

[8] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint*, 2015.

[9] M. MacDonald, "Automated database workload characterization, mapping, and tuning through machine learning," 2018.

[10] C. Di Natale and E. Martinelli, "Data analysis," in *Breath Analysis*. Elsevier, 2019, pp. 81–94.

[11] R. Ross, R. Latham, W. Gropp, R. Thakur, and B. Toonen, "Implementing mpi-io atomic mode without file system support," in *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.*, vol. 2. IEEE, 2005, pp. 1135–1142.

[12] A. Nisar, W.-k. Liao, and A. Choudhary, "Scaling parallel i/o performance through i/o delegate and caching system," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 2008, p. 9.

[13] Y. Alforov, T. Ludwig, A. Novikova, M. Kuhn, and J. Kunkel, "Towards green scientific data compression through high-level i/o interfaces," in *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 2018, pp. 209–216.

[14] A. R. Carneiro, J. L. Bez, F. Z. Boito, B. A. Fagundes, C. Osthoff, and P. O. A. Navaux, "Collective i/o performance on the santos dumont supercomputer," in *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. IEEE, 2018, pp. 45–52.

[15] P. J. Pavan and et al., "Energy efficiency and I/O performance of low-power architectures," *Concurrency and Computation: Practice and Experience*, p. e4948.

[16] Q. Zoll, Y. Zhu, and D. Feng, "A study of self-similarity in parallel I/O workloads," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–6.

[17] F. Wang and et al., "File system workload analysis for large scale scientific computing applications," Lawrence Livermore National Lab.(LLNL), Livermore, CA, Tech. Rep., 2004.

[18] Y. Kim and et al., "Workload characterization of a leadership class storage cluster," in *2010 5th Petascale Data Storage Workshop (PDSW '10)*, 2010, pp. 1–5.

[19] Y. Kim and R. Gunasekaran, "Understanding i/o workload characteristics of a peta-scale storage system," *The Journal of Supercomputing*, vol. 71, no. 3, pp. 761–780, 2015.

[20] M. Dorier and et al., "Calciom: Mitigating i/o interference in hpc systems through cross-application coordination," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. IEEE, 2014, pp. 155–164.

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

[22] Y. Liu, R. Gunasekaran, X. Ma, and S. S. Vazhkudai, "Server-side log data analytics for i/o workload characterization and coordination on large shared storage systems," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 819–829.

[23] B. Xie, J. Chase, D. Dillow, O. Drokin, S. Klasky, S. Oral, and N. Podhorszki, "Characterizing output bottlenecks in a supercomputer," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, Nov 2012, pp. 1–11.

[24] A. K. Mishra and et al., "Towards characterizing cloud backend workloads: insights from google compute clusters," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 4, pp. 34–41, 2010.

[25] S. Di, D. Kondo, and F. Cappello, "Characterizing and modeling cloud applications/jobs on a google data center," *The Journal of Supercomputing*, vol. 69, no. 1, pp. 139–160, 2014.

[26] M. C. Calzarossa, L. Massari, and D. Tessera, "Workload characterization: A survey revisited," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, p. 48, 2016.

[27] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 characterization of petascale i/o workloads," in *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, 2009, pp. 1–10.

[28] M. Lawrence and et al., "Software for computing and annotating genomic ranges," *PLOS Computational Biology*, vol. 9, no. 8, pp. 1–10, 2013.

[29] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[30] D. S. Wilks, "Cluster analysis," in *International geophysics*. Elsevier, 2011, vol. 100, pp. 603–616.

[31] R. C. de Amorim and C. Hennig, "Recovering the number of clusters in data sets with noise features using feature rescaling factors," *Information Sciences*, vol. 324, pp. 126–145, 2015.

[32] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[33] F. Pedregosa and et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.