MOSAIC: Detection and Categorization of I/O Patterns in HPC Applications

Théo Jolivel, François Tessier, Julien Monniot, Guillaume Pallez

MOSAIC: Detection and Categorization of I/O Patterns in HPC Applications

Théo Jolivel, François Tessier, Julien Monniot, Guillaume Pallez
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
{*first_name*}.{*last_name*}@*inria.fr*

***Abstract*—With the gap between computing power and I/O performance growing ever wider on HPC systems, it is becoming crucial to optimize how applications perform I/O on storage resources. To achieve this, a good understanding of application I/O behavior is an essential preliminary step. In this paper, we introduce MOSAIC, a method for categorizing applications according to their I/O behavior. We first propose an abstraction for characterizing I/O operations in terms of periodicity, temporality and metadata access. We then present a set of segmentation-based techniques for quickly and automatically detecting meaningful data access patterns. In the end, MOSAIC is able to characterize a full set of real-world I/O traces from the Blue Waters supercomputer with 92% accuracy.**

***Index Terms*—I/O, characterization, analysis, traces**

I. INTRODUCTION

While the ratio of I/O performance to computing power has declined by a factor of 10 in the last decade [1], the volume of data generated by scientific workflows and applications has significantly grown. In some supercomputing centers for instance, this volume has increased almost 40-fold in ten years [2]. This has made access to storage resources a major bottleneck to scaling up applications.

Several levers exist along the data path to mitigate this burden. For example, optimizations can be applied at the I/O library level [3]–[5] or within the application source code [6] to improve I/O performance. At the job scheduler level, decisions can be taken when allocating resources to avoid I/O interference between jobs [7]–[9]. However, all these optimizations require a good upstream understanding of application I/O behavior. While very detailed information is both hard to obtain and not necessary for making rapid and efficient decisions [10]–[12], an abstraction able to describe an application's overall I/O behavior is a good tradeoff. Tools exist to trace application execution and monitor I/O operations but they lack a way to analyze and depict them rapidly.

In this paper, we propose MOSAIC, an approach to categorize execution traces and give information about the general behavior of applications from an I/O perspective. To demonstrate our method, we analyze a full year of I/O execution traces of Blue Waters, a supercomputer that is now decommissioned. From these traces, we determine a set of non-exclusive categories to describe the I/O behavior of jobs, including the temporality and the periodicity of the accesses and the metadata overhead. We then propose a set of algorithms based on event fusion and trace segmentation to automatically categorize the I/O traces and implement them in a tool called MOSAIC (Merging Operations and SegmentAtion for I/o Categorization). For example, a numerical simulation performing regular checkpoints throughout its execution and writing a final result before finishing will be identified as *periodic* and *write_on_end* by MOSAIC. Over more than 24,000 I/O traces, we verified that our method has an accuracy of 92% in categorizing data access patterns. Based on these results, we present a post-mortem analysis of I/O behavior on Blue Waters, and we highlight some significant correlations between categories that could drive scheduling algorithms.

Our contributions are as follows:

- An analysis of I/O execution traces to determine a set of common and redundant features
- A method for categorizing jobs according to their I/O behavior
- An analysis of this categorization on a year of I/O traces

In the rest of this paper, we first present an overview of relevant related work. Section III then describes both the categories we determined to characterize data access patterns and MOSAIC, our categorization algorithm. We show in Section IV the result produced by MOSAIC, run on one year of Blue Waters I/O traces. We also discuss here some implementation details and the accuracy of our method evaluated with sampling.

II. RELATED WORK

A. I/O Monitoring and Analysis

Tracing the I/O behavior of applications running at large scale is a topic that has been studied for years. Several approaches have been proposed, from very low-level monitoring tools like Recorder [13] or EZTrace [14], which collect a large volume of information at each execution, to higher-level abstractions like Darshan [15], which partially aggregates I/O operations between the opening and closing of a file. Overall, a trade-off must be found between the overhead a user is prepared to face and the precision of the information needed.

In this ecosystem, Darshan [15] is one of the most widely used tools and the source of the largest publicly available I/O-enabled execution trace datasets. It is able to trace the I/O behavior of an application running on supercomputers with very low overhead and without modifying the application's source code. Darshan supports multiple I/O APIs, such as

POSIX, MPI-IO, or STDIO. An additional module, called DXT (Darshan eXtended Traces) [16], allows gathering additional information at a higher level of detail but at the cost of a higher overhead. However, no large DXT-enabled I/O trace datasets are publicly available to the best of our knowledge.

Numerous studies have used Darshan traces to analyze the overall I/O behavior of a supercomputer [17], [18], to guide the design of storage systems [19] or to improve the way applications perform I/O operations [6]. Luu et al. [20] used Darshan traces from three machines to gather insights about how applications generally access data on the PFS. Karimit et al. [21] also used Darshan data to compare the behavior in I/O performance of machine learning workloads from their scientific domains. Given this large volume of information, a natural idea is to take advantage of it to better characterize I/O behavior of scientific applications.

B. I/O Characterization

Several approaches have been proposed to analyze and categorize executions according to their I/O activities [22], [23]. Recent work, for example, uses frequency techniques such as discrete Fourier transforms to detect periodicity in I/O traces [24]. However, this approach fails to distinguish between two intricate periodic behaviors and does not cover other types of behaviors. Other works present a categorization based on aggregate statistics (total data volume, number of processes involved in I/O, etc.) rather than on individual I/O operations [25]. This type of categorization only makes it possible to establish very high-level patterns that do not provide temporal information. Another approach consists of predicting activity spikes on the parallel file-system from Darshan files to understand the global load and anticipate when the system will likely experience higher stress than usual [26]. However, this work does not allow to describe the global I/O behavior of an application, which could be used for job scheduling, for example.

Finally, rather than adopting an application view, Boito et al. [10], [27] focused on analyzing the logs of a parallel file-system to determine classes of I/O behavior and study the file system's configuration in the face of these access patterns. The categorization here is very high-level and does not allow to describe the temporality or periodicity of accesses.

In MOSAIC, we use a clustering algorithm that allows better flexibility in identifying mixed patterns. We also analyze the temporality of accesses to determine when an application needs resources the most, and the load on the metadata server.

III. APPROACH

We present here our approach for assigning applications into representative categories based on their I/O behavior. We first propose an abstraction able to describe at a high-level the data access pattern of an application. Then we introduce MOSAIC, an algorithm for automatically detecting and categorizing these patterns from I/O traces.

	Categories
Temporality	{read_, write_}: on_start, on_end, after_start, before_end, after_start_before_end, steady, insignificant
Periodicity	periodic, periodic_second, periodic_minute, periodic_hour, periodic_day_or_more, periodic_low_busy_time, periodic_high_busy_time
Metadata	high_spike, high_density, multiple_spikes, insignificant_load

TABLE I: Categories for characterizing I/O behavior

A. Category Definition

The main goal of our method is to provide a high-level, fast-to-detect and representative description of I/O behavior in large-scale applications. Therefore, the categories need to be wide enough to identify the general motifs encountered. A preliminary analysis from the literature [28] shows that applications mainly exhibit intermittent I/O behavior (writing final results, reading input data) and periodic behavior (checkpointing, periodically reading files). These operations lead to metadata access (opening/closing files, searching for offsets). Our categorization is intended to be representative of these behaviors. We have thus identified three different aspects:

- Periodicity: These categories describe periodic behaviors that generally correspond to checkpointing phases. Several labels give an order of magnitude of the period or the volume of data involved.
- Temporality: The corresponding categories describe when read or write operations have occurred. The most common labels are, for example: *read_on_start* or *write_on_end*.
- Metadata impact: This category class describes the impact of I/O on the parallel file-system metadata server.

Table I lists the categories we have defined for each class of behavior (periodicity, temporality, metadata). Most of these categories, though evocatively named, are described in more detail in Section IV when detected. The "insignificant" categories are used to exclude applications that are not I/O-intensive from characterization. Therefore, we estimate that applications reading or writing less than 100MB or carrying out fewer metadata operations than the number of ranks fall into those categories. These thresholds have been determined experimentally for the dataset processed, but do not cover certain cases (massive library loading at application start, for example). Future work will investigate advanced methods for determining them. MOSAIC evaluates these criteria (below or above the threshold) independently for read and write operations. A trace can be both categorized as *read_insignificant* and get labels highlighting an important write activity. The abovementioned threshold can be modified in MOSAIC to extend or narrow the amount of I/O activities to categorize.

In total, our categories describe 98% of a year's worth of Blue Waters' Darshan traces.

B. MOSAIC: I/O Pattern Categorization Algorithm

MOSAIC takes I/O traces in the format provided by Darshan as input, applies pre-processing (merging) and uses a

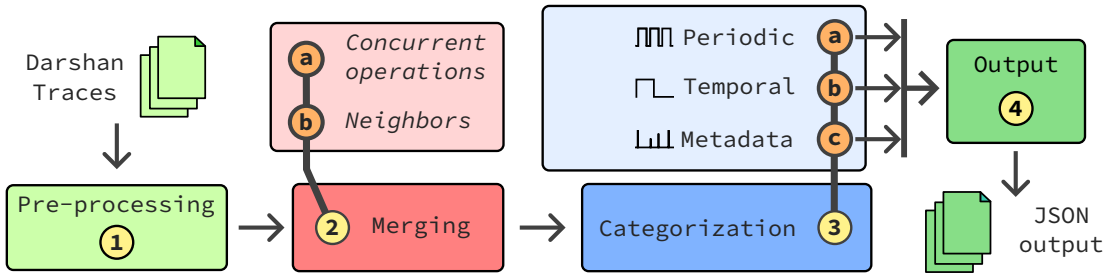

Fig. 1: MOSAIC workflow for processing an I/O trace

segmentation algorithm to detect data access patterns of the job. Figure 1 describes these steps.

1) Trace Pre-processing : MOSAIC begins by opening each Darshan trace file to check its validity (1). The corrupted entries (when a deallocation happens before the end of the application's execution for instance) are deleted.

The next step is to obtain a representative dataset without retaining all the executions. Indeed, it is very common on a supercomputer to run the same application several times (sometimes several hundred times), generating as many execution traces. Since we want to categorize application behavior, we assume that all executions of an application from a given user will belong to the same categories. Therefore, it allows us to drastically reduce the number of traces to categorize while ensuring a satisfactory diversity in observed I/O patterns. To validate this hypothesis, we looked at the most executed applications during one year from our input dataset. These applications correspond to typical HPC simulations such as LAMMPS, MILC, VASP or NEK5000. A study of the corresponding I/O traces shows that their I/O behavior is mostly stable regardless the execution parameters. For example, about 97% of the ≈12000 runs of LAMMPS are similarly categorized by MOSAIC while this percentage is 80% for NEK5000. For a set of executions, MOSAIC only analyzes the heaviest (i.e. the most I/O-intensive) trace.

2) Merging of I/O Operations: From this point, MOSAIC handles read and write operations independently. For each I/O trace kept from the previous step, MOSAIC applies two merging algorithms (2):

a) Concurrent operation merging (2) (a): if two I/O operations overlap, those are merged into a single one. This fusion has two objectives: manage process desynchronization so that, for example, several processes writing to the same file in a slightly desynchronized way will have their write operations merged as a single operation, and clarify the trace to enable the detection of periodic behavior (see Section III-B3).

b) Neighbor merging (2) (b): since we want to detect global application behaviors, MOSAIC merges nearby operations if the gap between them is negligible (less than 0.1% of the total execution time or less than 1% of the duration of the nearby merged operation). This second algorithm reduces the trace complexity by lowering the number of I/O operations to process and retain only the data necessary for a correct categorization. Again, it helps when processes are slowly desynchronized. If the desynchronization slowly slides operations until they are no longer overlapping, they can still be merged in the same operation if they are close enough.

3) Categorization:

a) Detection of Periodic Operations: (3) (a) Once the traces have been refined using previously described merging techniques, MOSAIC segments them by I/O operation. More concretely, a segment starts at the beginning of an I/O operation and ends at the beginning of the next one. The upper part of Figure 2 illustrates this division. Once this segmentation has been completed, our algorithm calculates the duration and volume of data read or written for each segment. A clustering algorithm based on Mean Shift [29] identifies all the segments that share comparable duration and data size, and groups them. A group with a size strictly greater than 1 formed by this algorithm corresponds to a periodic operation. In this way, several groups and therefore several periodic operations can be detected within a single application, which corresponds to real-life cases (for example, a numerical simulation performing both checkpointing and data reading at regular intervals). However, this technique requires defining the thresholds at which two segments are considered part of the same periodic operation. We empirically set and refined these thresholds on one month of traces until periodic operations were correctly identified, avoiding overfitting drawbacks from other methods. Then, we validated these thresholds on the whole dataset using a sampling method: we randomly selected 512 elements among all the categorized traces for one year of data to verify that the different patterns are handled correctly.

Once a periodic behavior has been recognized, MOSAIC calculates the order of magnitude of its period from the length of a segment, the volume of data read or written by each operation, and the activity rate during the period.

b) Temporality Characterization: (3) (b) Independently, MOSAIC also characterizes the temporality of read and write accesses. To do so, it splits each trace into four equal chunks, each representing 25% of the execution time. The lower part of Figure 2 shows an example of this type of segmentation. The first and last chunks represent the beginning and end of the execution. For each chunk, MOSAIC sums the amount of bytes handled by the underlying I/O operations. It then compares these values to determine which one(s) contain(s) the

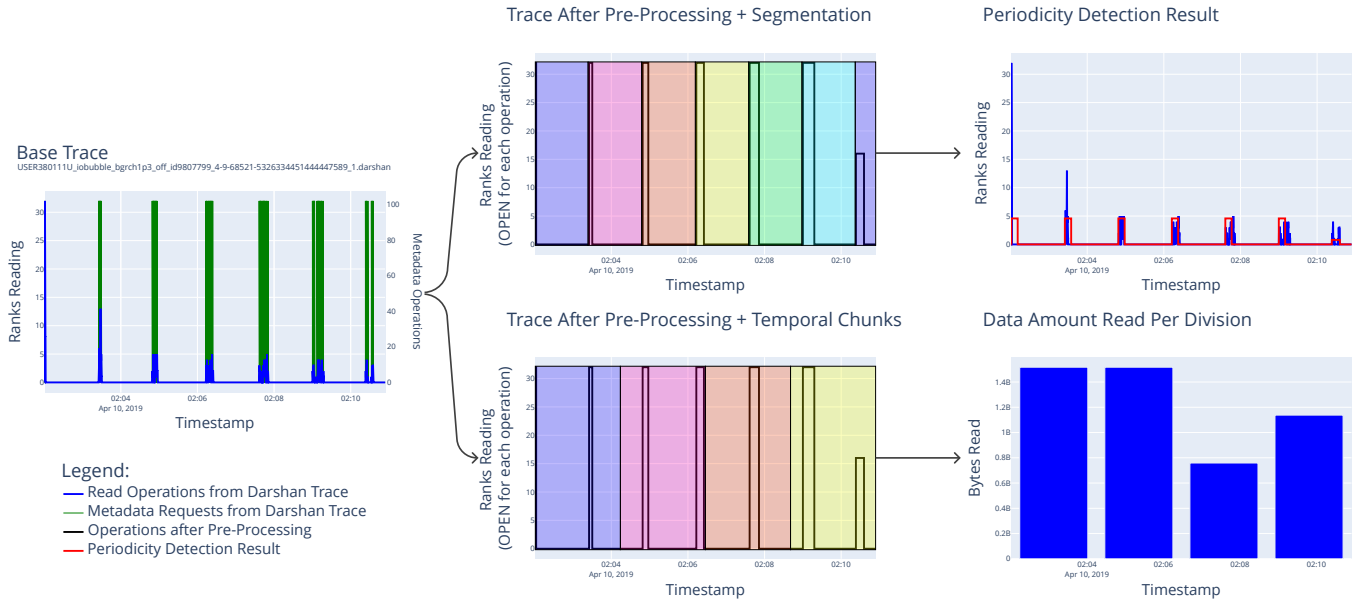

Fig. 2: Trace processing example

most impactful operations and assigns the associated category. For example, if the first chunk contains more than twice the amount of bytes operated in the other segments, the trace will be assigned to a {*read/write*}*_on_start* category. On the contrary, if all chunks contain the same amount of data (coefficient of variation under 25%), the trace will be characterized as {*read/write*}*_steady*.

c) Metadata Access: (3) (c) Finally, MOSAIC must be able to describe application behavior at the metadata server level when it is significant. To do so, it uses the number of *OPEN*, *CLOSE*, and *SEEK* requests emitted for each I/O operation. Because Darshan does not precisely trace *SEEK* operations, we assume these are co-located with each *OPEN* operation.

MOSAIC then assigns the application to categories to tell if a trace contains a high spike of requests, if multiple request spikes are present, and if there is a high density of requests at some point, i.e., multiple requests spread over several seconds. The number of metadata requests from which we consider an activity to have an impact is not obvious. Our thresholds are based on the work of JM Kunkel and GS Markomanolis [30], which benchmarked different metadata servers from supercomputers. The *Mistral* supercomputer at DKRZ is similar to Blue Waters, including its parallel file-system, and is saturated at roughly 3000 requests per second.

From that, MOSAIC considers a trace as having a high spike if more than 250 requests are emitted in one second, multiple spikes if there are at least 5 spikes of 50 requests or more, and a high density of requests if there are at least 5 spikes and an average of 50 requests or more per second throughout the execution.

4) Output: Once MOSAIC has processed a trace, it saves the assigned categories and the calculated values (period for instance) in a JSON file (4). It also outputs statistics about the global behavior of an application and patterns found in the traces. For example, MOSAIC provides the number of applications with a given behavior (i.e, belonging to the same category), based both on data from pre-processed traces and on raw data including multiple executions of the same application (see Section III-B1). The former data analyzes the behavior of the executed applications, while the latter gives information about the load on the parallel file system. A heatmap also gives information about recurrent associations of classes, with Jaccard index [31]. Section IV discusses this output data in greater detail.

IV. EVALUATION

To build and evaluate MOSAIC, we worked on the 2019 traces from the Blue Waters supercomputer. Blue Waters, which was decommissioned in 2021, was a 13.3 PFlops Cray XE/XK HPC system featuring more than 26,000 compute nodes interconnected within a 3D-torus network interconnect. In terms of storage, the machine was equipped with 26PB of storage space managed by a Lustre file-system. The "scratch" partition was distributed across 360 OSSs and 1440 OSTs. For several years, the I/O of applications running on Blue Waters were monitored by default with Darshan (DXT module disabled) unless explicitly stated by the user at compile time. These traces have been made publicly available [1]. We have selected 2019 as the peak year for machine usage. To the best of our knowledge, this is the most comprehensive dataset available. Other more recent machines, such as Theta at Argonne National Laboratory or Cori at NERSC, have also been monitored, but the traces, when available, are lossy aggregated versions of the Darshan traces.

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

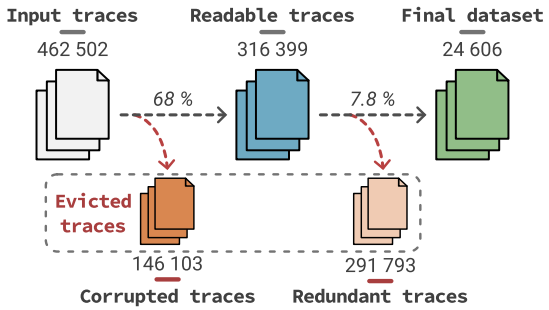

Fig. 3: Pre-processing of one year I/O traces from Blue Waters

The pre-processing phase of this dataset, as described in Section III-B1, is presented in Figure 3. It resulted in 32% of traces being corrupted and therefore evicted, and 8% of unique executions in the set of remaining valid traces. From an initial set of 462'502 traces, MOSAIC retained 24'606 entries for categorization.

In the remainder of this section, we present the results of MOSAIC on this dataset in terms of characterizing periodic behavior, access temporality and the impact of metadata operations. We then provide an analysis of significant correlations between certain behaviors and discuss the accuracy of our algorithm.

A. Periodicity

Execution	Non-Periodic	Periodic	
Single run	98%	2%	
		Min.	Hour
All runs	92%	6%	2%

TABLE II: Detection of periodic write operations

Table II shows how many applications have been categorized as performing periodic write operations from our dataset while considering unique executions of an application and the full set of executions. Therefore, 2% of the analyzed applications are periodic, representing 8% of the executions. The frequency of these periodic accesses fluctuates between a few minutes and a few hours.

However, this result needs to be treated cautiously. Indeed, one limitation of Blue Waters Darshan traces is that accesses are aggregated between the opening and closing of a file. In the case of an application that opens files at start time and keeps them open throughout the execution, Darshan will only provide a single entry in the trace set, signifying that several I/O operations have taken place during this interval without giving their temporal distribution. MOSAIC categorizes this behavior as {*read/write*}_*steady*. This category represents 37% of the write behaviors (see Section IV-B). It is likely that the majority of these behaviors are, in fact, periodic.

Periodic read accesses account for less than 2% of all executions and are subject to the same limitations as Darshan traces. The order of magnitude of these accesses is smaller, ranging from several seconds to several minutes.

	Studied distrib.	Insignificant	On start	Steady	Others
Read	Single run	85%	9%	2%	4%
	All runs	27%	38%	30%	5%
	Studied distrib.	Insignificant	On end	Steady	Others
Write	Single run	87%	8%	3%	2%
	All runs	47%	14%	37%	2%

TABLE III: Detection of temporality

B. Access Temporality

Table III shows the categorization of reads and writes in terms of temporality for the dataset reduced to single run for each application (highlighting the behavior of individual workloads), and for the complete dataset (emphasizing the global load on the parallel file-system).

These results first show that most applications perform either few reads (85%) or few writes (87%), the threshold being set at 100MB (see Section III-A). This categorization also shows that of the total number of application executions studied, 95% can be described by 6 categories: 3 for reading, 3 for writing. These categories highlight two expected phenomena: applications mostly read at the start of execution (38%) or during execution (30%), while they write either regularly (37%, probably periodic accesses related to checkpointing), or at the end of execution (14%). Finally, we also note that almost half of executions (47%) perform no or few writes, while only a quarter (27%) perform no or few reads.

C. Metadata Access

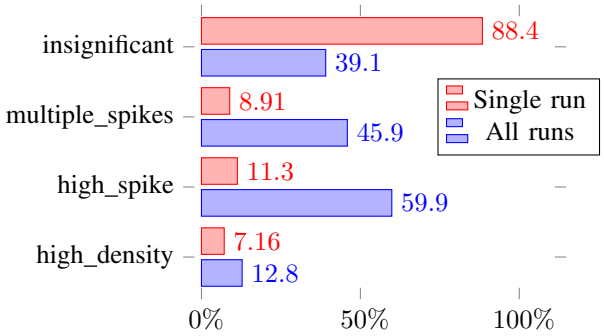

Fig. 4: Category distribution for metadata access

Figure 4 depicts the distribution of metadata categories for the traces we analyzed with MOSAIC.

The significant differences in values for the single run set or for all runs highlight that a small number of applications with a large number of executions are metadata-intensive. The most represented category in these results, *metadata_high_spike*, contains applications performing more than 250 metadata accesses per second at least once during their execution. 60% of the executions studied fall into this category, showing a heavy load on the metadata server. We also note that 45.9% of applications have several spikes (*metadata_multiple_spikes*), which is in line with the estimated percentage of applications with periodic writes (8% identified + 37% categorized as

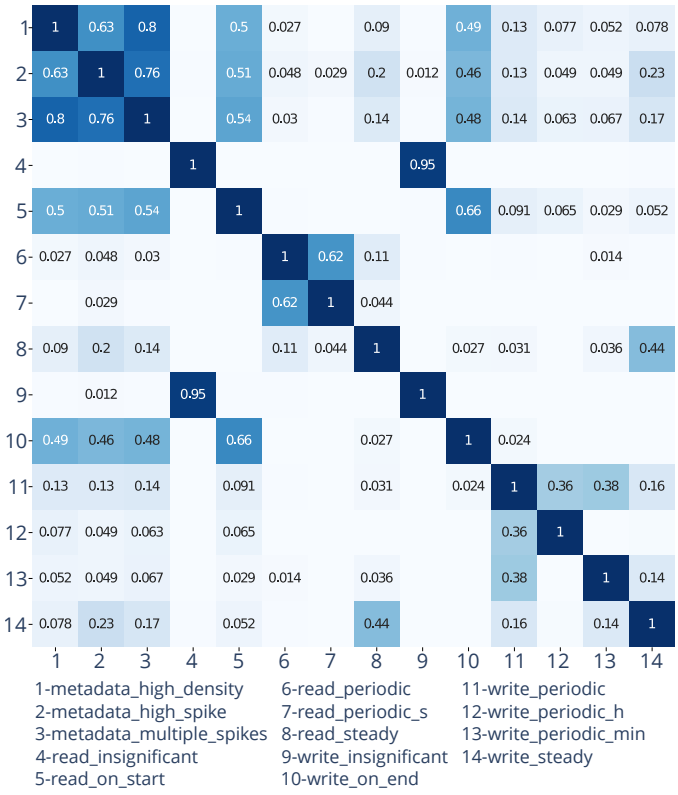

Fig. 5: Matrix of relevant Jaccard indices. Only values higher than 1% are shown

write_steady). Finally, just under 13% of executions are categorized as having a high metadata access density, characterized by an average of 50 requests per second throughout the execution.

D. Noteworthy Correlations

The Jaccard indices computed by MOSAIC, which compare similarity and diversity between samples, can be used to identify relevant correlations. These correlations can ultimately help in job scheduling choices to limit I/O interference between applications, for example. These correlations include the following:

- Applications with a high metadata request density and high metadata spikes are more likely to read on start and/or write on end.
- The large majority of applications (95%) having no significant read operations also have no significant write operation.
- 66% of applications reading on start writes on end. The pattern *read, compute, write* is well represented in the traces we analyzed.
- Almost all traces with periodic writes (96%) spend less than 25% of the time writing on the parallel file-system.

Figure 5 presents a subset of the generated Jaccard heatmap to present other interesting results.

E. MOSAIC Accuracy and Performance

We used the categorization results of one year of traces to estimate MOSAIC's accuracy. To do so, we applied a sampling method: we randomly selected a subset of 512 traces that we manually validated. We detected that 42 traces were incorrectly classified, mainly because of a sub-optimal detection of temporality in some cases where an operation is unequally spread across multiple chunks. This leads to an accuracy of 92% for MOSAIC.

In terms of performance, the complete MOSAIC workflow described in Figure 1 is capable of processing the entire dataset (except for 2 files that take too long to load) in 165 minutes on a 64 cores AMD Zen2 EPYC 7702 processor. MOSAIC is written in Python 3.10.14 (1800+ lines of code) and uses the Dispy library to parallelize trace processing. The main bottleneck in our implementation is memory: 300 GB of RAM is required to process the dataset (dependent on the parallelization level). However, beyond analysis on a large set of traces, as is the case in this paper, MOSAIC can also be used for application-by-application categorization to provide information to a job scheduler, for example.

V. CONCLUSION

In this paper, we presented MOSAIC, an abstraction for describing the I/O behaviors of applications running on supercomputers, and an automatic categorization method using this abstraction. Our categorization method, based on the fusion and segmentation of I/O execution traces, can categorize 98% of traces from a real-world dataset with an accuracy rate of over 90%. This high-level and automatic fast-to-detect characterization of I/O behavior paves the way for job scheduling techniques that take data access patterns into account. For example, two jobs categorized as reading large volumes of data at the start of execution could be scheduled so as not to overlap.

Looking ahead, we are considering several ways of improving MOSAIC. First, some signal-processing-based techniques for periodic I/O detection have been shown to be effective [24]. In the short term, we plan to implement these techniques to improve the detection of this type of pattern. Secondly, category determination could be made more automatic using clustering methods. Finally, in the longer term, we plan to analyze the dataset in greater depth to detect I/O performance losses that could be attributed to concurrency. This way, we would like to be able to identify whether some categories are more conflicting than others, again in order to use this information to improve concurrency-aware job scheduling.

VI. ACKNOWLEDGMENT

As part of the "France 2030" initiative, this work has benefited from a State grant managed by the French National Research Agency (Agence Nationale de la Recherche) attributed to the Exa-DoST project of the NumPEx PEPR program, reference: ANR-22-EXNU-0004. This research has also been supported in part by the NCSA-Inria-ANL-BSC-JSC-Riken-UTK Joint-Laboratory on Extreme Scale Computing (JLESC).

REFERENCES

[1] "Top500 ranking," https://www.top500.org/.

[2] G. Lockwood, D. Hazen, Q. Koziol, R. Canon, K. Antypas, and J. Balewski, "Storage 2020: A Vision for the Future of HPC Storage," in *Report: LBNL-2001072*. Lawrence Berkeley National Laboratory, 2017. [Online]. Available: https://escholarship.org/uc/item/744479dp#author

[3] H. Tang, Q. Koziol, J. Ravi, and S. Byna, "Transparent asynchronous parallel i/o using background threads," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 891–902, 2022.

[4] H. Zheng, V. Vishwanath, Q. Koziol, H. Tang, J. Ravi, J. Mainzer, and S. Byna, "Hdf5 cache vol: Efficient and scalable parallel i/o through caching data on node-local storage," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2022, pp. 61–70.

[5] F. Tessier, P. Gressier, and V. Vishwanath, "Optimizing data aggregation by leveraging the deep memory hierarchy on large-scale systems," in *Proceedings of the 2018 International Conference on Supercomputing*, ser. ICS '18. New York, NY, USA: ACM, 2018, pp. 229–239. [Online]. Available: http://doi.acm.org/10.1145/3205289.3205316

[6] J. L. Bez, H. Ather, and S. Byna, "Drishti: Guiding end-users in the i/o optimization journey," in *2022 IEEE/ACM International Parallel Data Systems Workshop (PDSW)*, 2022, pp. 1–6.

[7] A. Gainaru, G. Aupy, A. Benoit, F. Cappello, Y. Robert, and M. Snir, "Scheduling the i/o of hpc applications under congestion," in *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2015, pp. 1013–1022.

[8] R. Bleuse, K. Dogeas, G. Lucarelli, G. Mounié, and D. Trystram, "Interference-aware scheduling using geometric constraints," in *Euro-Par 2018: Parallel Processing*, M. Aldinucci, L. Padovani, and M. Torquati, Eds. Cham: Springer International Publishing, 2018, pp. 205–217.

[9] W. Liang, Y. Chen, J. Liu, and H. An, "Cars: A contention-aware scheduler for efficient resource management of hpc storage systems," *Parallel Computing*, vol. 87, pp. 25–34, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016781911830382X

[10] F. Boito, G. Pallez, L. Teylo, and N. Vidal, "Io-sets: Simple and efficient approaches for i/o bandwidth management," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 10, pp. 2783–2796, 2023.

[11] E. Jeannot, G. Pallez, and N. Vidal, "Io-aware job-scheduling: Exploiting the impacts of workload characterizations to select the mapping strategy," *The International Journal of High Performance Computing Applications*, vol. 37, no. 3-4, pp. 213–228, 2023. [Online]. Available: https://doi.org/10.1177/10943420231175854

[12] A. Benoit, T. Herault, L. Perotin, Y. Robert, and F. Vivien, "Revisiting i/o bandwidth-sharing strategies for hpc applications," *Journal of Parallel and Distributed Computing*, vol. 188, p. 104863, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731524000273

[13] C. Wang, J. Sun, M. Snir, K. Mohror, and E. Gonsiorowski, "Recorder 2.0: Efficient parallel i/o tracing and analysis," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 1–8.

[14] M. I. Naas, F. Trahay, A. Colin, P. Olivier, S. Rubini, F. Singhoff, and J. Boukhobza, "Eziotracer: unifying kernel and user space i/o tracing for data-intensive applications," in *Proceedings of the Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems*, ser. CHEOPS '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3439839.3458731

[15] 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*, 2009, pp. 1–10.

[16] C. Xu, S. Snyder, V. Venkatesan, P. Carns, O. Kulkarni, S. Byna, R. Sisneros, and K. Chadalavada, "Dxt: Darshan extended tracing," Argonne National Lab.(ANL), Argonne, IL (United States), Tech. Rep., 2017.

[17] G. K. Lockwood, S. Snyder, T. Wang, S. Byna, P. Carns, and N. J. Wright, "A year in the life of a parallel file system," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2018, pp. 931–943.

[18] J. L. Bez, A. M. Karimi, A. K. Paul, B. Xie, S. Byna, P. Carns, S. Oral, F. Wang, and J. Hanley, "Access patterns and performance behaviors of multi-layer supercomputer i/o subsystems under production load," in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 43–55. [Online]. Available: https://doi.org/10.1145/3502181.3531461

[19] J. Monniot, F. Tessier, M. Robert, and G. Antoniu, "Supporting dynamic allocation of heterogeneous storage resources on hpc systems," *Concurrency and Computation: Practice and Experience*, vol. 35, no. 28, p. e7890, 2023. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.7890

[20] 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*, ser. HPDC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 33–44. [Online]. Available: https://doi.org/10.1145/2749246.2749269

[21] A. M. Karimi, A. K. Paul, and F. Wang, "I/o performance analysis of machine learning workloads on leadership scale supercomputer," *Performance Evaluation*, vol. 157-158, p. 102318, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166531622000268

[22] W. Yang, X. Liao, D. Dong, and J. Yu, "A quantitative study of the spatiotemporal i/o burstiness of hpc application," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2022, pp. 1349–1359.

[23] M. Isakov, E. d. Rosario, S. Madireddy, P. Balaprakash, P. Carns, R. B. Ross, and M. A. Kinsy, "Hpc i/o throughput bottleneck analysis with explainable local models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–13.

[24] A. Tarraf, A. Bandet, F. Zanon Boito, G. Pallez, and F. Wolf, "Capturing Periodic I/O Using Frequency Techniques," in *IPDPS 2024 - 38th IEEE International Parallel & Distributed Processing Symposium*, San Francisco, United States, May 2024, pp. 1–13. [Online]. Available: https://inria.hal.science/hal-04382142

[25] H. Devarajan and K. Mohror, "Extracting and characterizing i/o behavior of hpc workloads," in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, 2022, pp. 243–255.

[26] E. Saeedizade, R. Taheri, and E. Arslan, "I/o burst prediction for hpc clusters using darshan logs," in *2023 IEEE 19th International Conference on e-Science (e-Science)*, 2023, pp. 1–10.

[27] F. Boito, G. Pallez, and L. Teylo, "The role of storage target allocation in applications' i/o performance with beegfs," in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, 2022, pp. 267–277.

[28] J. L. Bez, S. Byna, and S. Ibrahim, "I/o access patterns in hpc applications: A 360-degree survey," *ACM Comput. Surv.*, vol. 56, no. 2, sep 2023. [Online]. Available: https://doi.org/10.1145/3611007

[29] K. Fukunaga and L. Hostetler, "The estimation of the gradient of a density function, with applications in pattern recognition," *IEEE Transactions on Information Theory*, vol. 21, no. 1, pp. 32–40, 1975.

[30] J. M. Kunkel and G. S. Markomanolis, "Understanding metadata latency with mdworkbench," in *High Performance Computing*, R. Yokota, M. Weiland, J. Shalf, and S. Alam, Eds. Cham: Springer International Publishing, 2018, pp. 75–88.

[31] S. Fletcher and M. Z. Islam, "Comparing sets of patterns with the jaccard index," *Australasian Journal of Information Systems*, vol. 22, Mar. 2018. [Online]. Available: https://journal.acs.org.au/index.php/ajis/article/view/1538