An Empirical Method for Processing I/O Traces to Analyze the Performance of DL Applications*

Edixon Parraga[1][0000−0002−5038−7827], Betzabeth Leon[1][0000−0003−1778−0237], Sandra Mendez[2][0000−0002−5793−1928], Dolores Rexachs[1][0000−0001−5500−850X], Remo Suppi[1][0000−0002−0373−8292], and Emilio Luque[1][0000−0002−2884−3232]

[1] Computer Architecture and Operating Systems Department. Universitat Autònoma de Barcelona, Campus UAB, Edifici Q, 08193 Bellaterra, Barcelona, Spain
{edixon.parraga, betzabeth.leon, dolores.rexachs, remo.suppi, emilio.luque}@uab.es
https://webs.uab.cat/hpc4eas/

[2] Computer Sciences Department. Barcelona Supercomputing Center (BSC). Barcelona, 08034, Spain.
sandra.mendez@bsc.es

Abstract. The exponential growth of data handled by Deep Learning (DL) applications has led to an unprecedented demand for computational resources, necessitating their execution on High Performance Computing (HPC) systems. However, understanding and optimizing Input/Output (I/O) of the DL applications can be challenging due to the complexity and scale of DL workloads and the heterogeneous nature of I/O operations. This paper addresses this issue by proposing an I/O traces processing method that simplifies the generation of reports on global I/O patterns and performance to aid in I/O performance analysis. Our approach focuses on understanding the temporal and spatial distributions of I/O operations and related with the behavior at I/O system level. The proposed method enables us to synthesize and extract key information from the reports generated by tools such as Darshan tool and the seff command. These reports offer a detailed view of I/O performance, providing a set of metrics that deepen our understanding of the I/O behavior of DL applications.

Keywords: DL · I/O Analysis · HPC · I/O behavior patterns.

1 Introduction

Deep learning (DL) is one of the most popular computational approaches in the field of machine learning (ML) and has been shown to improve performance in areas such as natural language processing [1], computer vision [2], and computational biology [3]. However, the exponential growth of data handled by DL

* This research has been supported by the Agencia Estatal de Investigación (AEI), Spain, and the Fondo Europeo de Desarrollo Regional (FEDER) UE, under contract PID2020-112496GB-I00 and partially funded by the Fundacion Escuelas Universitarias Gimbernat (EUG). The authors thankfully acknowledge RES resources provided by CESGA in FinisTerrae III to RES-DATA-2022-1-0014.

applications has led to an unprecedented demand for computational resources, necessitating their execution on High Performance Computing (HPC) systems [4].

A critical and often challenging aspect of running DL applications in HPC systems is efficiently managing file Input/Output (I/O) operations. These operations are essential for loading data and storing results. Optimal handling of these operations is important, especially when the application needs to deal with datasets containing thousands of samples (i.e., thousands of small images) processed by the parallel file system of the HPC systems. These operations present I/O patterns that can become bottlenecks, limiting the overall performance of DL applications [5].

To understand the impact of I/O on the performance of DL applications, it is necessary to comprehend their I/O behavior in HPC systems. However, due to the complexity of the I/O software stack for these applications, there is a need for a method to process and depict I/O metrics and patterns in a structured way that can guide users in the analysis of I/O performance.

In this context, we propose an I/O traces processing method that simplifies the generation of reports about global I/O patterns and performance by utilizing specific outputs from monitoring and profiling tools. Our approach focuses on analyzing temporal and spatial I/O patterns and their distribution on the parallel filesystem, which can be correlated with the obtained performance metrics. As the authors note in [6], analyzing and understanding an application's I/O access patterns provides key insights into how an application's I/O behavior affects its performance on different systems. This allows us to understand the I/O behavior on the HPC system and identify and data access and distribution patterns that minimize their impact on application I/O performance. Therefore, our paper aims to contribute in the following ways:

- Introducing an I/O trace processing method that extracts and synthesizes critical information from outputs generated by I/O monitoring and profiling tools in HPC systems.
- Providing insights to aid in the identification of potential I/O bottlenecks within the output of compute nodes and/or storage nodes.

The structure of this article is presented as follows: Section 2 highlights the usefulness of understanding I/O patterns by using profiling, monitoring and tracing tools and it also provides a review of related works. Section 3 presents our I/O trace processing method step by step. Section 4 applies our approach to a case study. Finally, in Section 5, we present our conclusions and future work.

2 Background

In this section we present a brief review on the importance of using I/O tracing, monitoring, and profiling tools to understand the I/O behavior of DL applications. Furthermore, we review some works related to our approach.

2.1 Analysis from data traces and Monitoring Tools

Understanding the I/O behavior in DL applications requires a meticulous and structured approach to capturing and analyzing meaningful data. This section breaks down into two main components that our approach needs: 1) Patterns of Data Access in Files and 2) Profiling, Tracing and Monitoring tools to capture these access patterns at the different levels of the I/O software stack.

Patterns of Data Access in Files: According to the authors of [7], data access characteristics in DL workloads differ markedly from traditional workloads, with unique patterns in memory and random accesses to large files. Tracing and analysing I/O operations provide a detailed insight into data access and allow measurement of the impact on application performance. This is essential for understanding and optimizing the I/O of the DL application. Traces collect important information about resource usage and I/O operations behavior. These traces help in identifying bottlenecks and potential issues, and they are essential for validating and reproducing experiments.

Profiling, Tracing and Monitoring tools: To analyze the I/O behavior of the DL applications addressed in this study, the Darshan tool [6] has been selected. Darshan is a tool designed to investigate the I/O performance of HPC applications at large scale. Darshan is composed of different modules that allow it to capture the I/O operations at different levels. Furthermore, Darshan can be used for profiling, tracing, or monitoring of the file system operations, but on the client side. That is, this tool is deployed in the same environment where the DL application is running, providing direct, real-time monitoring of its I/O performance.

Darshan logs contains detailed information about I/O operations, timing, performance, I/O counters and so on. Synthesizing and organizing this extensive information is essential for practical analysis, which requires a processing method that condenses and clarifies the data. Using as input Darshan's logs, our proposed method simplifies the information and facilitates the visualization of the results of I/O behavior.

2.2 Related work

The behavior of I/O operations in DL is very important, being the focus of studies such as [8] and [9], which examine some existing analytical tools and compare some DL models. In [10], the authors use DXT Explorer to optimize these I/O operations. The DLIO tool, introduced in [11], replicates I/O behavior in scientific DL workloads, which is helpful for DL performance evaluation and improvement. Additionally, the research in [12] focuses on the evolution of I/O evaluation, while the research in [13] deals with the emulation of I/O behavior in scientific workflows. These studies highlight the influence of new workloads on HPC systems and the need to understand data transfer between modules.

Our work distinguishes itself by developing a method that processes I/O traces, providing a detailed analysis of I/O behavior and identifying potential improvements, thus contributing to research on I/O performance in DL applications.

3 Proposed Trace Analysis Method

This section describes our proposed method with the essential steps to perform the I/O trace processing of a DL application. The objective is to accurately obtain the required information, which is fundamental for analyzing the DL application I/O. The proposed method is composed by three main stages: Input, Processing and Output.

3.1 Stage 1: Input

In this stage, we set the right environment for the different tools used to capture the information related to I/O activities.

I/O Data Acquisition Tool Deployment: In the initial phase, we enable the DXT Darshan[6] module to obtain traces of the I/O operations at POSIX-IO level. This detail is needed to identify and model the I/O spatial and temporal pattern of the application. Darshan is loaded by using the `LD_PRELOAD` environment variable and this is exported in job submission script before the command to run the application in the HPC system. If the application runs without any problems, a Darshan log file is generated per each job identifier. Furthermore, the information related with counters and performance is provided by Darshan parser and perf.

As we deploy this tool in HPC systems, we implement scripts to take the job identifier of each execution of the application to relate with each Darshan log generated. This allows us to process several Darshan logs at the same time. It is useful when users need to analyze scaling issues.

Additionally, to see CPU and memory usage for a job and evaluate if the I/O is having impact on these resources we use the seff command output of the Slurm Workload Manager.

3.2 Stage 2. Processing.

In this stage, we initiate monitoring of the application by submitting the job to the `SLURM` queuing system (using the command `sbatch script.sh`), ensuring that the monitoring configuration established in the previous stage is fully active. To guarantee the accuracy and reliability of the data obtained, it is essential to perform each experiment several times. During this process, we verify that our Job is running correctly (command `squeue`) to confirm that the monitoring system is working correctly and that the binary file *.darshan* is being generated completely and consistently. This step ensures the integrity of the collected data and provides a basis for subsequent analysis.

Data Preparation: It involves organizing and preparing the collected data for analysis, including cleaning, filtering out irrelevant data, and transforming it into a format suitable for analysis. The data selected from the different reports are presented in Table 1. This table describes the relevant information extracted from the original `Seff` and Darshan reports, which were obtained directly without any additional processing. Additionally, it details the specific element extracted from each report and explains its usefulness in the context of our work.

Table 1. Data selected for the traces I/O analysis

Report	Information provided	Element extracted from the report	Utility
Seff	Job Metadata	Job ID, cluster, user/group, status	Allows you to track and audit resource usage. Provides context for the execution of the job.
	CPU and Memory Efficiency	CPU used, CPU efficiency, Core-walltime, memory efficiency.	Identifies resource utilization efficiency, helping to optimize CPU and memory allocation.
	Computing and Memory Resources	Nodes, cores per node, job clock time, memory used.	Provides data on the processing capacity used. Helps evaluate execution time in relation to allocated resources.
Darshan DXT Report	General Job Metadata and File System	Job ID, nprocs, runtime, file id, fs type, Lustre stripe size, Lustre OST obdidx.	Allows mapping I/O operations to specific resources.
	POSIX Module Data	File name, rank, hostname, write count, read count, Lustre stripe count, Module.	Provides data on the volume of I/O generated. Helps to understand how the workload is distributed.
	Temporal and Spatial Tracking Metrics	Wt/Rd, Segment, Offset, Length, Start(s), End(s), OST.	Provides specific details about I/O operations, essential for identifying and resolving bottlenecks.
Darshan Parser Report	Identification and Context of Work	Job ID, nprocs, run time, record_id, Module, fs type, file name	Defines the context and uniqueness of the analyzed work.
	POSIX Performance Metrics	POSIX READS/WRITE, POSIX BYTES, POSIX MAX BYTE, POSIX SIZE, POSIX F READ, POSIX F WRITE	Provides a detailed view of the performance of POSIX operations.
	Performance Analysis in LUSTRE	Number of OSTs, stripe parameters, POSIX time	Provides a detailed understanding of how I/O resources are configured and used.
Darshan Perf Report	General Job Metadata	Job ID, nprocs, run time, total bytes	Helps understand the scale of operations and I/O intensity.
	I/O Performance Metrics	I/O timing for unique files (seconds), I/O timing for shared files (seconds), Aggregate performance	Allows you to evaluate the impact of concurrent operations on overall performance.

Report Generation and Analysis: the collected traces are processed to generate reports summarizing the performance and efficiency of I/O operations. This includes descriptive statistics, visualizations, and the detection of initial patterns.

A- Report: Utilized Resources by the job. The utilized resources per job can be obtained by the `seff` command line. `seff`'s output presents mainly information related to the memory utilization and CPU efficiency for completed jobs. The data obtained includes the following:

1. Left Column - Job Metadata: `Job ID`: This is the unique identifier assigned to the job by the cluster management system. `Cluster`: The specific cluster in which the job was run that is relevant to understanding the operational context. `User/Group`: The identity of the user or user group that submitted the job is critical for auditing and tracking resource utilization. `State`: The final status of the job (e.g., successfully completed, failed, canceled), providing an immediate view of the job's outcome without going into specific performance details.

2. Center Column - CPU and Memory Usage Efficiency Metrics: `CPU Utilized`: is the total CPU time the job has utilized. `CPU Efficiency`: A measure that reflects how effectively the allocated CPU time has been used, calculated as the ratio of CPU time to total available CPU time. `Core-walltime`: The product of the wall time and the number of cores, providing a composite measure of processing time consumed. `Memory Efficiency`: Similar to CPU efficiency, this metric evaluates the efficiency of memory usage allocated to the job.
3. Right Column - Compute and Memory Resources Used: `Nodes`: The total number of compute nodes assigned to the job. `Cores per node`: The number of processor cores available per node, which is essential to understanding the computing capacity per node. `Job Wall-clock time`: The total time from start to completion. `Memory Utilized`: The memory used during the job execution.

Fig. 1 shows in orange the data from the original report and in sky-blue the data extracted for the performance analysis in the next steps of our method. The three main columns represent the different categories of data to be extracted for the report.

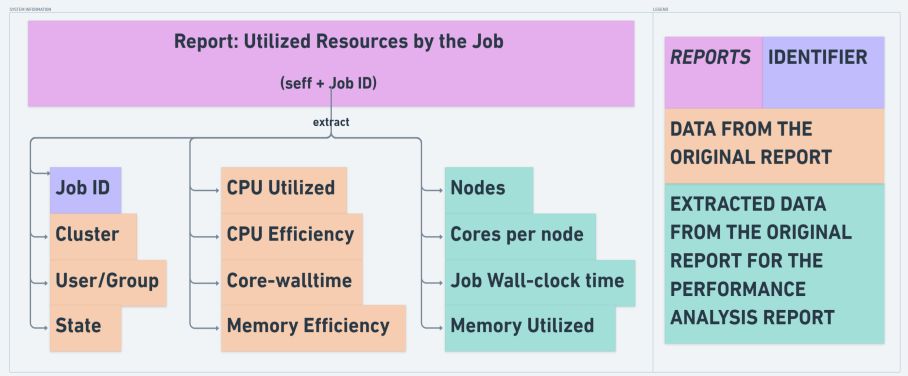


Fig. 1. Utilized resources by the Job

B- Darshan Log Files: We run our scripts, which process the log files generated by Darshan while monitoring the application, to generate the `DXT`, `PARSER`, and `PERF` reports. These reports are then used to apply our method for analyzing the I/O performance of the application. Our method focuses on the I/O done by the application on the datasets.

Darshan DXT Report Processing: This processing consists of extracting and organizing the relevant information from the DXT reports. Fig. 2 shows the main information into three columns, representing different strata of metadata and metrics:

1. General Metadata of the Job and the File System (Left Column): `Job ID, nprocs, run time`: These attributes define the operational context of the

computational job, including the unique job identification, process concurrency, and duration. `file id, fs type`: These are the identifiers that specify the file and the type of file system used, essential for mapping I/O operations to a specific storage resource. `Lustre stripe size, Lustre OST obdidx`: These are parameters related to the configuration of LUSTRE, a parallel file system, where the 'stripe size' and 'OST obdidx' are critical to understanding data distribution and storage location.

2. POSIX DXT Module Data (Central Column): `file name, rank, hostname`: These fields directly reference the file and execution context, including the rank of the MPI process and the compute node. `write count, read count`: Quantitative metrics of read and write operations provide insight into the volume of I/O that a specific job generates. `Lustre stripe count, Module`: Lustre's stripe count and the identification of the Darshan module used indicate the parallelization configuration and the active tracking subsystem.
3. Temporal and Spatial Tracking Metrics (Right Column): `Wt/Rd, Segment, Offset, Length, Start(s), End(s), OST`: These parameters outline the I/O profile at a granular level, detailing whether the operations are read or write, the specific location within of the file (segment and offset), the amount of data involved (length), and the timing of the operations (start and end), as well as the Object Storage Targets (OSTs).

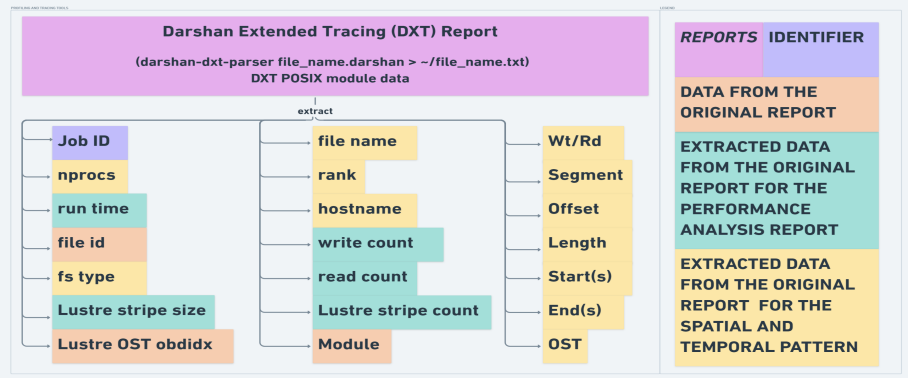


Fig. 2. Main I/O data extracted from DXT report

Darshan Parser Report Processing: `darshan parser` command provides detailed set of I/O performance metrics and counters for a specific job executed in a HPC system. Below, we summarize the main information extracted from the report to be used by our method:

1. Job Identification and Context: attributes such as Job ID, number of processes (nprocs), run time (run time), record identifier (record_id), module used (Module), file system type (fs type), and file name (file name) are essential to characterize the operational context and uniqueness of the analyzed job.

2. POSIX Performance Metrics: metrics related to the POSIX module, offering an overview of the number and volume of read and write operations (POSIX READS/WRITE), byte access (POSIX BYTES and POSIX MAX BYTE), and the sizes of the I/O operations (POSIX SIZE * - *). The specific entries for reading and writing files (POSIX F READ and POSIX F WRITE) offer a more focused look at individual file operations.
3. Performance Analysis in LUSTRE: The metrics associated with the LUSTRE file system provide detailed insight into the configuration and performance of a parallel file system. Metrics include the number of Object Storage Targets (OSTs), which directly influence the capacity and speed of I/O operations, and stripe parameters (size and width), which determine how data are distributed and accessed across multiple OSTs. Additionally, the time associated with metadata operations (POSIX F META TIME) is important to understanding the metadata management overhead.

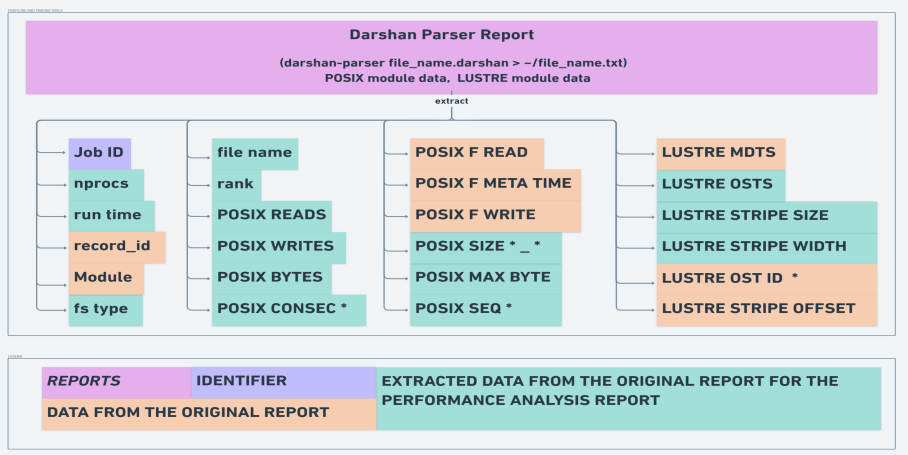


Fig. 3. Darshan Parser Report

In Fig. 3, it can be observed in sky-blue the selected information from the original report to be used to synthesize new analytical reports. These reports are important for understanding the performance of a specific job and for helping to identify potential I/O bottlenecks and optimization points.

Darshan Perf Report Processing: `darshan perf` command provides I/O performance metrics for shared and independent files accessed by application. As our focus is on performance evaluation, we extract the following data:

1. General Job Metadata (Left Column): `Job ID`: It is the identifier of the job, providing a unique reference to the set of supercomputing tasks in question. `nprocs`: Indicates the number of parallel processes running, which is critical to understanding the scale of parallelism involved. `run time`: Represents the duration of the supercomputing work, which is essential for evaluating time efficiency. `total bytes`: Quantifies the total volume of data handled, providing a metric of the I/O intensity of the work.

2. I/O Performance Metrics (Right Column): `I/O timing for unique files (seconds)`: Reflects the I/O time spent on unique files, essential for discerning file system performance when processes access their own files without sharing. `I/O timing for shared files (seconds)`: Measures the I/O timing for files shared between processes, which can indicate how concurrent operations affect performance. `Aggregate performance`: Displays aggregate performance metrics based on the slowest time recorded, which could identify the worst-case performance among all processes and provide insights for optimizations.

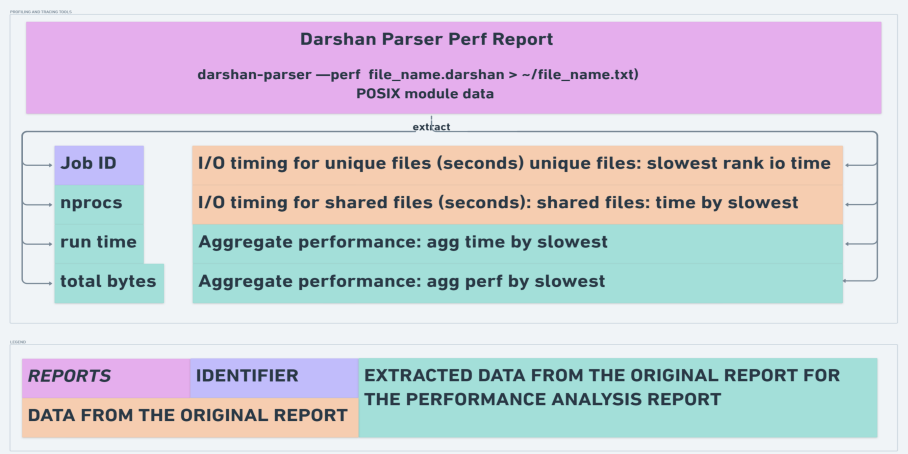


Fig. 4. Perf report

Fig. 4 presents data selected in sky-blue from the original report, which are used in performance analysis in our method.

Performance analysis report: Based on the previous reports obtained in the base processing, our automated method processes the files and creates an overall report consolidating the previous reports, using the JobID as an index.

This global report summarizes the calculation of the information relevant to our analysis. Among the relevant data and its calculations are the following: Total Metadata Operations, Total Data Access Operations, Data Access Operations by Nodes, Data Access Process Operations, I/O Operations per second (IOPs), Bytes per Node, Bytes per Process, Total Operations. Fig. 5 depicts the different original reports and data selected from them to be used by our method. The selected data allows us to analyze the I/O performance of the DL application.

3.3 Stage 3. Output:

Taking all the information selected from the original reports two main outputs are available for the user: 1) Spatial and Temporal Pattern Analysis and 2) Performance Analysis.

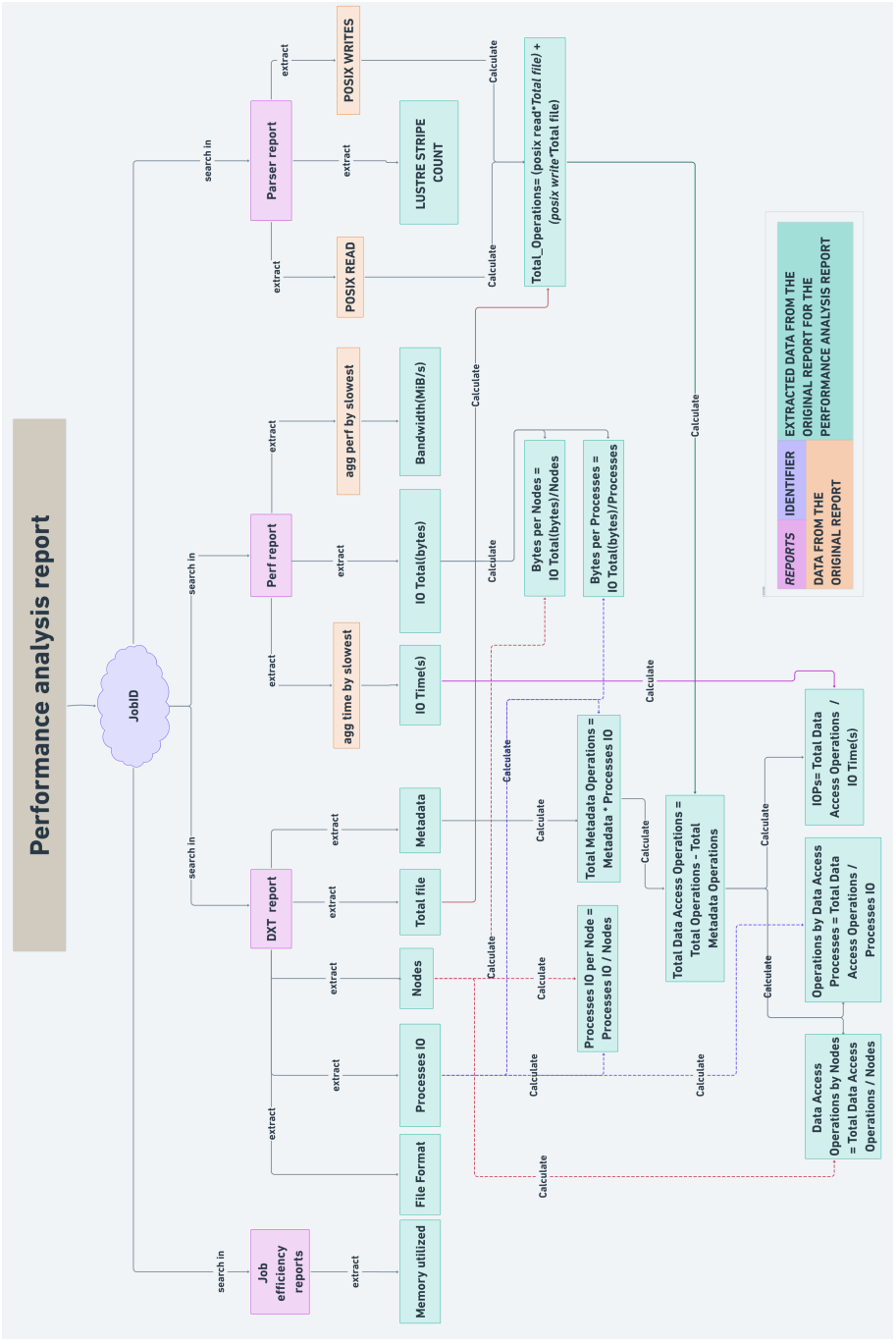


Fig. 5. Performance analysis report

Spatial and Temporal Pattern Analysis: An analysis of the I/O operations from DXT reports is carried out to identify temporal and spatial patterns. Users can plot these patterns and analyze the behavior of I/O events based on their order or timestamp. This analysis enables users to identify if there is a serialization of I/O operations by examining the sequence and timing of these operations. If a large number of I/O operations occur in a sequential manner without overlapping, this indicates serialization.

Furthermore, by comparing these patterns against expected parallel behavior, users can determine if the serialization is caused by the underlying I/O system's inability to handle parallel requests efficiently or by the I/O pattern generated by the application itself. Spatial and temporal access patterns are influenced by both the file type and the file system. While the file type dictates the logical organization of data, the file system's design and performance characteristics can affect how efficiently these patterns are handled, especially in parallel file systems.

Additionally, these patterns can be used to extrapolate the count of I/O operations and total bytes for different numbers of processes and I/O workloads, providing valuable insights for scaling and optimization.

Performance Analysis: Finally, the performance of the I/O operations is evaluated based on the previous analysis, as shown in Fig. 5. This step may involve comparing performance metrics to identify optimization opportunities. Fig. 5 shows the detailed procedure for monitoring and evaluating job performance in HPC systems. Using data selected from the seff command and the different reports of Darshan tool, we provide a comprehensive view of I/O performance that spans from memory utilization to distribution of I/O operations on the parallel file system. This comprehensive approach allows for pinpointing inefficiencies and understanding the interaction between application I/O patterns and the file system's capabilities.

4 Experimental Validation

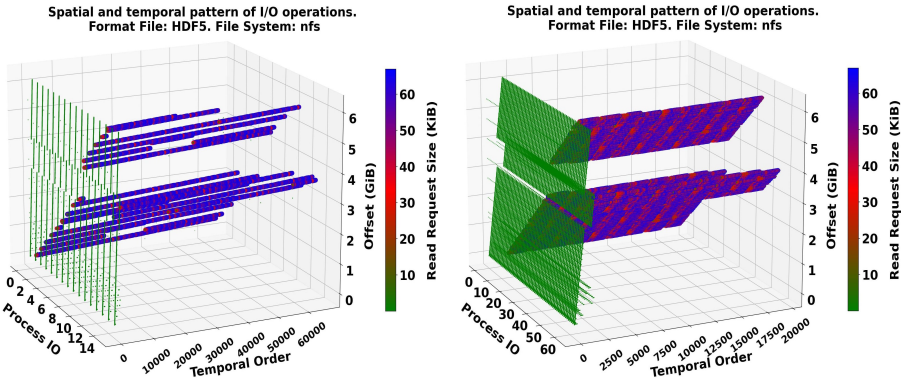
In this section, we apply our proposed method in a real DL application. We use Deep Galaxy application that aims to leverage the pattern recognition capability in modern DL to classify the properties of galaxy mergers [14]. The dataset contains 35,784 black and white images from simulations of galaxy mergers of different mass and size ratios. These images are stored in a compressed HDF5 dataset, with an internal structure of 36 folders each with 14 subfolders that represent the different positions of the camera and in each subfolder 71 images are stored with a resolution of 1024*1024 pixels each one, for a total dataset file size of 6.1 GiB. Below we show some graphical reporters designed to analyze DeepGalaxy I/O.

Figures 6 and 7 focus on the spatial and temporal pattern of the I/O operations. In Figure 6, which is a 3D representation, the X-axis, identified as

"Process IO", represents the different processes involved in the I/O operations; the Y-axis displays the **"Temporal Order"**, indicating the sequence in which the I/O operations occurred; and the Z-axis is the **"Offset (GiB)"**, representing the position in the file where the read or write operations take place, expressed in Gibytes (GiB). The legend on the right indicates the size of the read requests (**"Read Request Size (KiB)"**) with a color scale ranging from green, red and blue. In Figure 7, which is a 2D representation, the X-axis again represents "Process IO," while the Y-axis shows the "Offset (GiB)." The color scale remains the same, indicating the size of the read requests. To understand the underlying patterns and detect potential performance bottlenecks, we proceed to perform a comparative analysis of two graphs representing I/O operations.

4.1 Spatial and Temporal Pattern Analysis

The interaction of I/O operations within a distributed computing environment is very complex. This analysis aims to show these operations' spatial orientation and temporal progression. With this information, we aim to gain insight into the efficiency of data management and the effectiveness of resource utilization in HPC systems.



(a) 16 MPI processes (b) 64 MPI processes

Fig. 6. Temporal I/O pattern on a NFS file system. All processes access a single shared file at a different file offset. 4 MPI processes per compute node.

Fig. 6 shows the spatial and temporal pattern of DeeGalaxy when the dataset is read from an NFS file system. However, it is important to note that this pattern is similar on the Lustre file system. This is because the dataset is stored in an HDF5 file; therefore, the observed pattern is the same on both file systems. The only difference will be observed in the timing of the I/O operations, but not in their order.

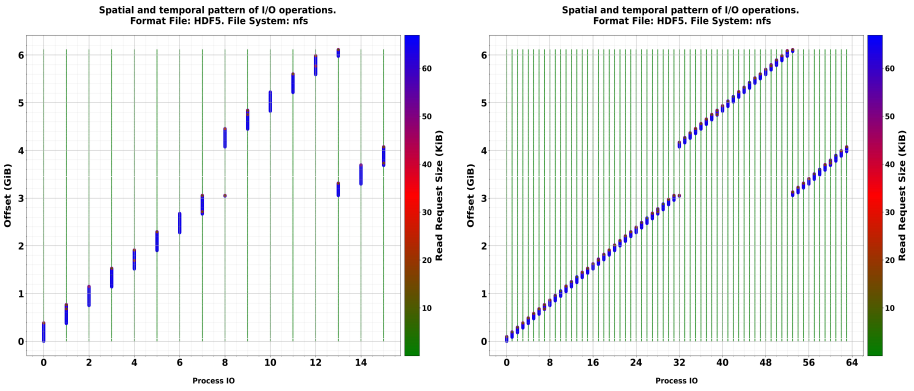
Fig. 6(a) and Fig. 7(a) depicts the variety in the size of I/O requests. The largest request is 66.91 KiB and the smallest request is noticeably smaller, at only

8 bytes. The average size of requests hovered at a median of 6.32 KiB. A total of 1,038,638 read-only I/O operations were performed. These read operations were spread over a set of 4 nodes and orchestrated by 16 I/O processes, following a shared data access pattern. Additionally, we observed a balanced number of I/O operations per process at POSIX-IO level. In terms of data volume, a total of 6.92 GiB is moved on the filesystem, which is 0.82 GiB more than the file size of the dataset.

Fig. 6(b) and Fig. 7(b) show variability in the size of I/O requests, with the maximum request size reaching 66.91 KiB and the minimum at only 8 Bytes. The average request size was approximately 5.91 KiB. Throughout the same period, there were 1,226,949 I/O operations performed, this time distributed over 16 nodes and managed by 64 I/O processes. The I/O aggregate data for these operations was also 6.92 GiB. In these pictures, we can also see whether there is an overlap in accesses or if each process read at a different file offset.

Fig. 6(a) and Fig. 6(b) depict how requests are distributed across time and file offset, which is critical for understanding access patterns and performance. Increasing from 4 to 16 nodes and from 16 to 64 I/O processes, suggesting that the NFS infrastructure can efficiently handle a growing number of operations without significantly degrading performance.

Figures 7(a) and 7(b) provide a 2D view of the I/O operations per process and their file offset, making it easier to identify whether the I/O processes access different offsets or if there are overlapping accesses to the same offset. In this case, each process reads its part of the file in parallel, so there is no overlap in accesses. This means that if we see any I/O serialization at runtime, it will be due to the underlying I/O system rather than application's I/O pattern.



(a) 16 MPI processes (b) 64 MPI processes

Fig. 7. Spatial I/O pattern on a NFS file system. All processes access a single shared file at a different file offset. 4 MPI processes per compute node.

4.2 DeepGalaxy Performance Analysis

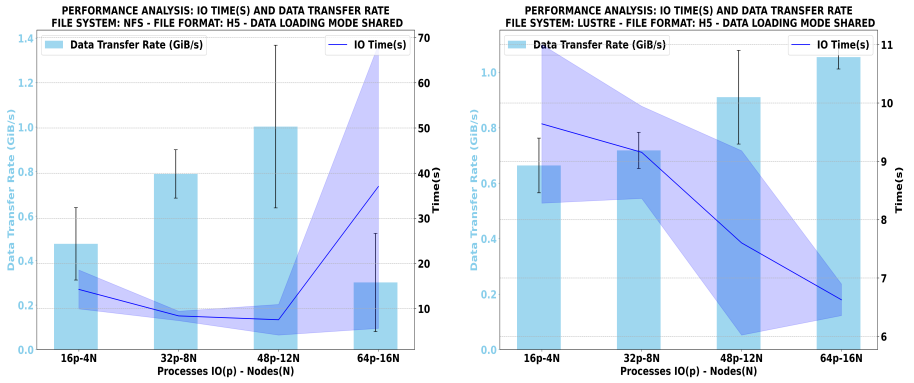
Extending our previous exploration of DeepGalaxy's I/O operations, we present four new plots for performance analysis, each showcasing results for a different number of processes. Furthermore, we executed the application reading dataset from a NFS and a Lustre filesystem. Fig. 8(a) and Fig. 8(b) present the I/O performance of the Deep Galaxy application, highlighting the influence of the count of processes and nodes. The x-axis represents the mapping used in each experiment. In all the cases, 4 processes are mapping in each compute node. So the label 16p-4N means a mapping of 16 processes distributed in 4 compute nodes. The y-axis correponts to time in seconds and the secondary y-axis displays the data transfer rate in GiB/s.

Fig. 8(a) indicates that NFS filesystem has significant performance fluctuations. At 16p-4N, the average I/O time was 14.26 seconds with a standard deviation of 4.30. The data transfer rate was relatively stable at 0.48 GiB/s (Std of 0.16). As the process count increased to 32 on 8 nodes, the average I/O time dropped to 8.39 seconds with a data transfer rate of 0.79 GiB/s. With 48 processes on 12 nodes, the I/O time slightly reduced to 7.54 seconds, though variability increased (Std of 3.39) with a peak data transfer rate of 1.00 GiB/s. However, at 64 processes on 16 nodes, the I/O time surged to 37.05 seconds, suggesting instability or potential bottlenecks and the data transfer rate significantly fell to 0.30 GiB/s with Std of 0.22.

In Fig. 8(a), it can be seen that despite initial reductions in I/O time, a significant spike and high variability in I/O time occur for the 64p-16N mapping. The data transfer rate increases up to 48p-12N, peaking at 1.00 GiB/s, before dropping sharply for 64p-16N. The high standard deviations in both I/O time and data transfer rate at 64p-16N indicate that NFS has problems managing the increasing number of small I/O operations for 64 I/O processes. This is an expected behavior because NFS is not designed to manage parallel I/O accesses.

Fig. 8(b) present the I/O performance of DeepGalaxy when reading dataset from a LUSTRE file system. For the 16p-4N mapping, an average I/O time of 9.64 seconds is reported with a stable data transfer rate of 0.66 GiB/s. With 32 processes on 8 nodes, I/O time decreases slightly to 9.15 seconds with an increased data transfer rate of 0.72 GiB/s. For 48 processes on 12 nodes, the I/O time further declines to 7.60 seconds and the data transfer rate climbs to 0.91 GiB/s. At the highest scale tested, 64 processes on 16 nodes, the average I/O time improves to 6.63 seconds with a data transfer rate peaking at 1.05 GiB/s.

Therefore, as the number of processes and nodes increases, both filesystems exhibit improved I/O times and similar data transfer rates. However, the variability in I/O times suggests that the Lustre filesystem is more appropriate for a larger number of I/O processes. The comparison clearly favors the Lustre filesystem, which consistently outperforms NFS in all metrics at the scales tested, underscoring its superior management capabilities in handling parallel I/O and load balancing on the data servers.



(a) Data Transfer Rate and IO Time. File System: NFS

(b) Data Transfer Rate and IO Time. File System: LUSTRE

Fig. 8. DeepGalaxy I/O Performance by using different number of processes.

5 Conclusions

We have presented our method for analyzing I/O traces of DL applications executed in HPC systems. Experimental validation has shown that our method provides useful information to guide users in analyzing the I/O behavior of DL applications. The temporal and spatial analysis of I/O patterns has offered a comprehensive understanding of I/O behavior, enabling the identification of potential bottlenecks and areas for performance improvement. Additionally, our approach has proven effective in synthesizing and extracting key information from reports generated by monitoring tools like Darshan and the `seff` command from SLURM.

Our research has shown that using a Lustre file system instead of NFS for specific workloads can significantly reduce I/O time and increase data transfer rates. In our experiments, the DeepGalaxy application demonstrated a reduction in I/O time from 37.05 seconds on NFS to 6.63 seconds on Lustre when the number of processes was increased from 16 to 64.

Understanding spatial and temporal I/O patterns is fundamental to explaining performance variations. For instance, significant differences in I/O performance were observed for the DeepGalaxy application when using different compute nodes and processes across various file systems. These differences highlight issues such as inefficient data access sequences and uneven distribution of I/O loads. By analyzing these patterns, we can identify and address these issues, leading to improved performance and efficiency.

For future work, we propose to apply the results of our method to address I/O optimization techniques such as caching and prefetching, distributing I/O operations across data servers, and identifying I/O intensity during peak demand periods to avoid I/O system saturation. Although I/O intensity largely depends on the application, it can be managed by implementing adaptive I/O

strategies that respond to varying load conditions. For example, during periods of high demand, dynamically reallocating I/O resources and prioritizing critical I/O operations can help mitigate saturation. Additionally, predictive modeling based on historical I/O patterns can forecast high-intensity periods, allowing for preventive adjustments in the I/O infrastructure. By implementing these optimization techniques, we aim to balance the load and improve the overall efficiency of the I/O system.

References

1. T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing [review article]," *IEEE Computational Intelligence Magazine*, vol. 13, no. 3, pp. 55–75, 2018.
2. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.
3. A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, 2017.
4. J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
5. D. Patil and P. Bhatia, "High performance computing for big data: methodologies and applications," *Journal of Parallel and Distributed Computing*, vol. 103, pp. 1–19, 2016.
6. P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, "Understanding and improving computational science storage access through continuous characterization," *ACM Transactions on Storage (TOS)*, vol. 7, no. 3, pp. 1–26, 2011. [Online]. Available: https://doi.org/10.1145/2027066.2027068
7. J. Lee and H. Bahn, "Analyzing data access characteristics of deep learning workloads and implications," in *2023 3rd International Conference on Electronic Information Engineering and Computer Science (EIECS)*, 2023, pp. 546–551.
8. R. Krishnamurthy, T. S. Heinze, C. Haupt, A. Schreiber, and M. Meinel, "Scientific developers v/s static analysis tools: Vision and position paper," in *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2019, pp. 89–90.
9. X. Wu, L. Qin, B. Yu, X. Xie, L. Ma, Y. Xue, Y. Liu, and J. Zhao, "How are deep learning models similar? an empirical study on clone analysis of deep learning software," in *2020 IEEE/ACM 28th International Conference on Program Comprehension (ICPC)*, 2020, pp. 172–183.
10. J. L. Bez, H. Tang, B. Xie, D. Williams-Young, R. Latham, R. Ross, S. Oral, and S. Byna, "I/o bottleneck detection and tuning: Connecting the dots using interactive log analysis," in *2021 IEEE/ACM Sixth International Parallel Data Systems Workshop (PDSW)*, 2021, pp. 15–22.
11. H. Devarajan, H. Zheng, A. Kougkas, X.-H. Sun, and V. Vishwanath, "Dlio: A data-centric benchmark for scientific deep learning applications," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2021, pp. 81–91.

12. S. Neuwirth and A. K. Paul, "Parallel i/o evaluation techniques and emerging hpc workloads: A perspective," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, 2021, pp. 671–679.
13. F. Chowdhury, Y. Zhu, F. Di Natale, A. Moody, E. Gonsiorowski, K. Mohror, and W. Yu, "Emulating i/o behavior in scientific workflows on high performance computing systems," in *2020 IEEE/ACM Fifth International Parallel Data Systems Workshop (PDSW)*, 2020, pp. 34–39.
14. M. X. Cai, J. Bédorf, V. A. Saletore, V. Codreanu, D. Podareanu, A. Chaibi, and P. X. Qian, "Deepgalaxy: Deducing the properties of galaxy mergers from images using deep neural networks," in *2020 IEEE/ACM Fourth Workshop on Deep Learning on Supercomputers (DLS)*, 2020, pp. 56–62.