Drishti: Guiding End-Users in the I/O Optimization Journey

Jean Luca Bez, Hammad Ather, Suren Byna
* Lawrence Berkeley National Laboratory
jlbez@lbl.gov, hather@lbl.gov, sbyna@lbl.gov

***Abstract*—The complex software and hardware I/O stack of HPC platforms makes it challenging for end-users to obtain superior I/O performance and to understand the root causes of I/O bottlenecks they encounter. Despite the continuous efforts from the community to profile I/O performance and propose new optimization techniques and tuning options for improving the performance, there is still a translation gap between profiling and tuning. In this paper, we propose *Drishti*, a solution to guide scientists in optimizing I/O in their applications by detecting typical I/O performance pitfalls and providing recommendations. We illustrate its applicability in two case studies and evaluate its robustness and performance by summarizing the issues detected in over a hundred thousand Darshan logs collected on the Cori supercomputer at the National Energy Research Scientific Computing Center (NERSC). *Drishti* can empower end-users and guide them in the I/O optimization journey by shedding some light on everyday I/O performance pitfalls and how to fix them.**

***Index Terms*—insights, I/O optimization, HPC, Darshan**

I. Introduction

Using the HPC I/O stack deployed in large-scale computing centers efficiently has been a tricky problem. The inter-dependencies among layers of I/O software, including high-level parallel I/O libraries (e.g., HDF5 [1], PnetCDF [2]) interfaces (e.g., MPI-IO, POSIX, STDIO), and file systems (e.g., Lustre [3], GPFS [4]) makes this a non-trivial task that often requires an I/O expert in the loop. Despite numerous efforts from the community to collect I/O profiles or traces and propose new optimization and tuning solutions to improve performance, there is still a gap between profiling and tuning [5]. The main challenge in *transforming I/O profiles into meaningful information is to detect root causes of I/O bottlenecks and mapping them into actionable items so an end-user can receive guidance on tuning I/O performance.*

In this paper, we prototype a solution, called *Drishti*, towards closing this gap. The term '*Drishti*' comes from Sanskrit, meaning "eyesight" or "vision" to cultivate insights[1]. Our goal is to provide insights of collected I/O profiles or traces and to guide users to take action on fixing the most common I/O performance bottlenecks. In this study, we have used I/O profiles collected by Darshan [6] for exploring insights and suggesting optimizations. We demonstrate Drishti's applicability with two case studies by automatically identifying common I/O bottlenecks and providing the end-user with recommendation and solution snippets. We also summarize the most common I/O performance bottlenecks detected in more than 100,000 Darshan logs.

[1]Meaning of *Drishti*: https://www.yogapedia.com/definition/5286/drishti

II. Background

A. Cori Supercomputer

Cori is a Cray XC40 supercomputer deployed at NERSC. It has $2,388$ Intel Xeon Haswell processor nodes and $9,688$ Intel Xeon Phi Knight's Landing (KNL) compute nodes, all connected to a Lustre parallel file system with $\approx$ 30 PB of storage capacity and a peak I/O bandwidth of 744 GB/s. Lustre provides a single POSIX namespace with five Metadata Servers (MDSes) and 244 Object Storage Servers (OSSes).

B. Darshan I/O Profiling Toolset

Darshan [6] is a popular tool available in several production supercomputers to collect I/O profiling information from applications. It aggregates I/O profile information to provide I/O performance statistics without adding overhead or perturbing application behavior. Darshan also provides an extended tracing module (DXT) [7] that captures fine-grain metrics on write and read operations using POSIX and MPI-IO. In this paper, we use the regular Darshan log files to analyze and identify I/O performance issues and to map those to various optimization strategies.

Darshan 3.3 and higher provides *pyDarshan* [8], a Python utility to interact with Darshan log records of HPC applications. In this work, we rely on such features to extract meaningful data to build insights and provide recommendations.

III. Related Work

A plethora of related work [9]–[14] has exposed some of the most common root causes of I/O performance bottlenecks originating from the application down to the file system. Though some issues are intrinsic to how the application was modeled and coded, others reflect the complexity of correctly tuning the exposed parameters or the interplay between factors in the I/O software and hardware stack [5], [15]–[18].

From a user's perspective, merely having access to performance metrics from profilers and traces (e.g., Darshan [6], Recorder [19], TAU [20]) is insufficient as there is no translation between those, the bottlenecks that exist, and the actions users could perform to mitigate the bottlenecks. These are left to the end-user to find out. Efforts such as IOMiner [21], UMAMI [12], and TOKIO [22] provide a framework to analyze combined metrics seeking to detect root causes of I/O problems. DXT Explorer [5] approaches the problem visually and interactively but does not provide guidance on how to fix the issues. On the other hand, auto-tuning approaches [15],

DOI 10.1109/PDSW56643.2022.00006

[17], [18], [23] attempt relieving the burden, on users, of finding and fixing performance issues due to mis-configuration, but they are application-specific and time-consuming.

Despite having an expressive set of high-level libraries, high-performance middleware, tunable parameters, and optimization techniques, it becomes cumbersome for end-users to know when to apply each method if there is no mapping from metrics to problems and from those to a set of solutions. Our work attempts to close this gap with *Drishti* by analyzing I/O profiles and providing a set of actionable items intended to fix common I/O bottlenecks in applications.

IV. Moving I/O Issues to the Spotlight

In Table I, we summarize a few common root causes of I/O performance bottlenecks and whether or not they can be detected from metrics collected by a profiler or tracer or if they require additional logs. For instance, unlike its Extended Tracing module (i.e., DXT), Darshan's profiling only keeps track of the timestamp of the first and last operations to a given file, effectively hiding what happens in between such as different behaviors or I/O phases. Furthermore, gathering insights on file system usage relies on having access to OST-related counters or external system logs. Such profilers do not provide metrics to analyze bandwidth limitation, unbalanced workloads, high-peak file system usage, or contention.

In this study, we focus on metrics available in a regular Darshan log, so we center the comparison on its currently available features. We build upon those common issues and, through *Drishti*, provide actionable items for the end-user.

Drishti takes as input a Darshan profiling log and optional arguments to report the various (file-based and overall) performance issues it detects through analysis and actionable tasks for potentially resolving the issues. A verbose mode shows sample snippets of code to guide end-users to fix the detected I/O performance issues. We implemented *Drishti* as a Python-based command-line tool that harnesses the *pyDarshan* API and the Rich library[2]. Rich provides support for writing rich text (with color and style) to the terminal and for displaying advanced content such as tables, markdown, and syntax highlighted code, making command-line applications visually appealing and presenting data in a more readable way. *Drishti*

TABLE I
Root causes of I/O performance bottlenecks from related work and feasibility to identify them from metrics.

Root Causes	Darshan	DXT	System
Too many I/O phases [9]	✓	✓	✗
Bandwidth limited by a single OST I/O bandwidth [9], [10]	✗	✗	✓
Limited by the small data size [9]	✓	✓	✗
Unbalanced I/O workload among ranks [9]	✓	✓	✗
Large number of small I/O requests [9]	✓	✓	✗
Unbalanced I/O workload on OSTs [9], [11]	✓	✓	✓
Bad file system weather [9], [12]	✗	✗	✓
Redundant/overlapping I/O accesses [24], [25]	✓	✓	✗
I/O resource contention at OSTs [13], [14]	✗	✗	✓
Heavy metadata load [10]	✓	✗	✗

[2] https://rich.readthedocs.io

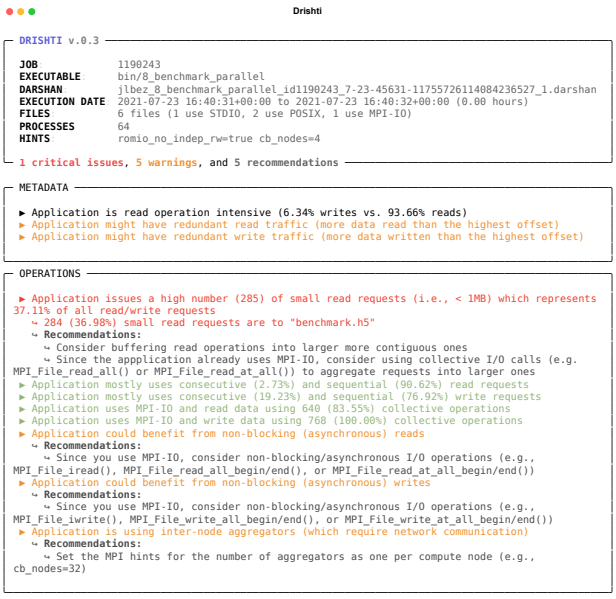

Fig. 1. Sample I/O insights report generated by *Drishti*.

TABLE II
I/O Insights are grouped according to impact.

Level	Description
HIGH	High probability of harming I/O performance.
WARN	Detected issues that could cause a significant negative impact on the I/O performance. The confidence of these recommendations is low as available metrics might not be sufficient to detect application design, configuration, or execution choices.
OK	Best practices have been followed.
INFO	Relevant information regarding application configuration.

is a command-line tool that can also export a report into multiple formats (e.g., standalone HTML file, SVG image, and text file). In Fig. 1, we show a sample *Drishti* report. These capabilities allow it to be used as part of the day-to-day workflow when submitting jobs or as a service in science gateways such as the Spin container-based platform[3].

Based on counters available in Darshan profiling logs, *Drishti* detects common bottlenecks and classifies the insights into four categories as shown in Table II. This classification reflects the impact of the triggered issues and the certainty of the provided recommendation since the available metrics might not be enough to detect particular application design, configuration, or execution choices.

In the current prototype of *Drishti*, we evaluate a Darshan log for a group of 30 triggers corresponding to given application behaviors. We show the complete list and description of each trigger in Table III. Some triggers have thresholds, which system administrators can tune based on each platform's characteristics. We describe some of the triggers below.

1) I/O Intensiveness: We define if an application is write- or read-intensive under two facets: the number of issued

[3] https://www.nersc.gov/systems/spin

operations or the total transfer size. This insight is provided as **INFO** and could be useful to determine whether or not an application could benefit from caching [26], burst buffers [27], or any other SSD-based file system [28]. *Drishti* focuses on the POSIX layer counters to report this based on operation count (`POSIX_READS` and `POSIX_WRITES`) or total transfer (`POSIX_BYTES_READ` and `POSIX_BYTES_WRITTEN`).

2) Small I/O Operations: Large numbers of small writes often result in poor performance [9], [21]. Common solutions involve aggregating or buffering of write/read operations or relying on MPI-IO's collective buffering. *Drishti* uses a threshold (defaults to 10%) to identify small write/read requests (considering all files and, in particular shared-filed, where collective operations could be easily implemented if MPI-IO is used). We use Darshan's histogram counters (e.g., `POSIX_SIZE_WRITE_`$\star$ and `POSIX_SIZE_READ_`$\star$) to observe the distribution of request sizes, where we define small operations as those that are smaller than 1 MB.

3) Redundant Traffic: Applications that read in total more bytes of data from the file system than were present in the file (i.e., re-reads, no caching, or all ranks read the full file) [24]. Even with caching, such jobs can cause disruptive I/O network traffic [25]. Aggregating requests or using collective I/O are possible solutions. Detecting this with Darshan might cause false positives as the profiler does not track the total file size. To detect this, we compare the total write/read transfer size for a given file using all interfaces with the maximum offset accessed by the application (`POSIX_MAX_BYTE_`$\star$).

4) Mis-aligned I/O: As HPC applications rely on a shared parallel file system (e.g., Lustre, GPFS) to store and retrieve data, and those use a technique called *data stripping* to increase performance, block/striped aligned I/O requests are important to avoid lock contention (false data sharing). As stripes can be located in different servers, unaligned requests can require multiple servers to complete an operation and introduce inefficiencies [29], [30]. Darshan exposes two POSIX counters to capture mis-aligned memory (`POSIX_MEM_NOT_ALIGNED`) and file (`POSIX_FILE_NOT_ALIGNED`) accesses.

5) Spatiality of Accesses: Spatiality refers to the location of requests in a file (contiguous, strided, or random), which directly impacts performance. Random writes inherently have lower performance than sequential writes [30]. Furthermore, avoiding random reads can promote better cache usage, helping prefetching and read-ahead [31]. *Drishti* evaluates `POSIX_CONSEC_`$\star$ and `POSIX_SEQ_`$\star$ to determine the spatiality. Any access that is not consecutive (immediately adjacent) nor sequential (at a higher offset) is taken as random.

6) Imbalance: Drishti detects imbalance in time (stragglers) and in transfer size. While the first might be caused by external factors such as network or file system contention, the latter is often related to how an application handles its data [32]–[34]. A common root cause of performance degradation is by having a single process, often rank 0, be responsible for writing and reading data. *Drishti* uses a threshold (by default 15% difference) to identify imbalance in combination with `POSIX_FASTEST_RANK_`$\star$ and `POSIX_SLOWEST_RANK_`$\star$ bytes and time Darshan counters.

7) Collective Operations: The MPI-I/O interface exposes collective operations that provide a big picture of the overall data movement across processes. It allows optimization techniques such as collective buffering and data sieving [35] to improve performance by building larger and contiguous accesses to the underlying storage system. When MPI-I/O is used by the application Darshan tracks the number of individual and collective operations that *Drishti* uses to compute the ratio.

8) Blocking I/O Accesses: I/O operations can be synchronous or asynchronous from the application's perspective. While each has pros and cons, asynchronous I/O is becoming increasingly popular to overlap computation or communication with I/O operations, which hides the cost associated with I/O and improves the overall performance [36], [37]. Both POSIX and MPI-IO interfaces offer blocking and non-blocking I/O calls. High-level libraries such as HDF5 have the Asynchronous I/O VOL Connector [38] to explore this feature. As Darshan only captures non-blocking calls at the MPI-IO level, *Drishti* uses `MPIIO_NB_READS` and `MPIIO_NB_WRITES` to suggest applications to explore such alternatives, if it makes sense to the application design and model.

9) High metadata load: Very high percentage of I/O times spent performing metadata operations such as `open()`, `close()`, `stat()`, and `seek()`. Regarding this, `close()` cost can be misleading due to write-behind cache flushing. These jobs are candidates for coalescing files and eliminating extra metadata calls. *Drishti* uses the `POSIX_F_META_TIME` to detect if a given rank has spent a cumulative time in metadata operations that is larger than a given threshold (by default 30s).

10) High STDIO Usage: Excessive use of STDIO for HPC applications might harm I/O performance [39], [40]. High-level interfaces such as MPI-IO are a better alternative. This issue is triggered based on a threshold (by default $> 10\%$) and the ratio of bytes transferred using STDIO compared to the total transfer size in all interfaces (e.g., $\star$`_BYTES_WRITTEN` and $\star$`_BYTES_READ`, where $\star$ is STDIO, POSIX, or MPI-IO).

V. I/O Performance Evaluation with *Drishti*

In this section, we apply *Drishti* to two use cases and evaluate its robustness and runtime with many Darshan logs.

A. I/O Insights in Practice

1) OpenPMD Use Case: OpenPMD [41] is an open metadata schema crafted to represent particle and mesh data from scientific simulations and experiments. Its library [42] provides a reference implementation of the standard using file formats such as HDF5 [1] and ADIOS [43]. In previous work [5], where we used openPMD-api 0.14.1 with HDF5 backend we noticed performance issues caused by a now-fixed bug in HDF5, which made collective metadata operations become individual (thus, issuing a lot of small requests), mis-aligned requests, and small data writes and reads. Fig. 2 depicts the *Drishti* report for this execution, and Fig. 3 confirms that the applied optimizations (refer to [5]) changed the application I/O behavior, avoiding common pitfalls.

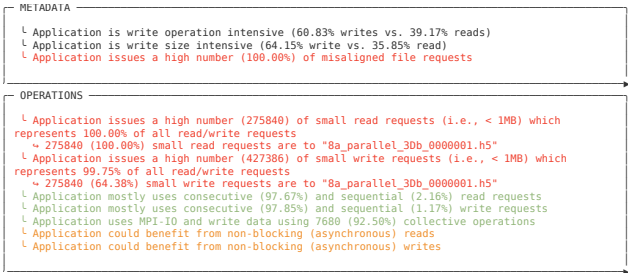

Fig. 2. *Drishti* report for the baseline OpenPMD in Cori.

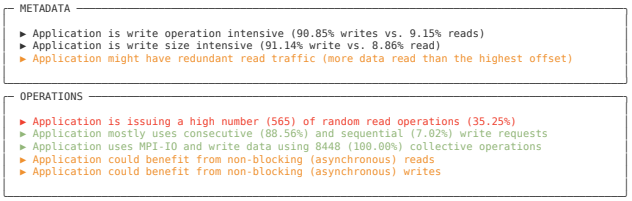

Fig. 3. *Drishti* report for the optimized OpenPMD in Cori.

2) Domain Decomposition Use Case: In previous work [5], we investigated the end-to-end (E2E) [44] domain decomposition I/O kernel. E2E uses NetCDF4, which internally relies on HDF5 to perform I/O operations.

The baseline executions were reported to perform very poorly. Applying *Drishti* to that Darshan log, we could have easily detected the write imbalance (99.90% in data transfer) caused by rank 0, as depicted by Fig. 4. As investigated in previous work, the culprit was the use of fill values for the subsequently overwritten datasets. After explicitly disabling this behavior (i.e., calling `nc_def_var_fill()` with the `NC_NOFILL` in NetCDF4), we achieved a speedup of $10\times$. Figure 5 depict the *Drishti* report of the optimized execution. It is important to notice that are still some mis-aligned accesses and redundant writes. Furthermore, *Drishti* correctly reports the large use of sequential operations and collective calls.

B. Overview of Bottlenecks

To assert the robustness and performance of *Drishti*, we have applied it $112,612$ Darshan logs collected in Cori (from March 1st to March 5th, 2022). Table III summarizes the insights. Those represent $12,648$ unique jobs, as a job might

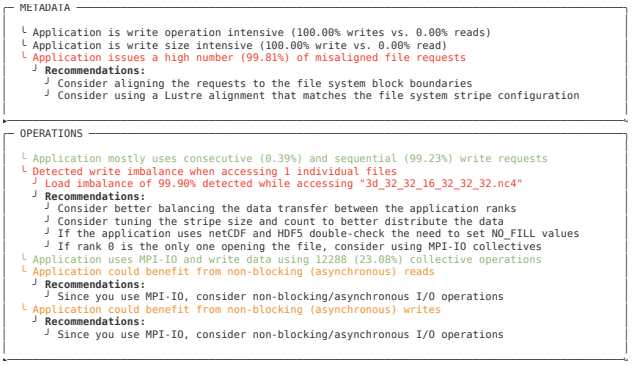

Fig. 4. *Drishti* report for the baseline E2E I/O kernels in Cori.

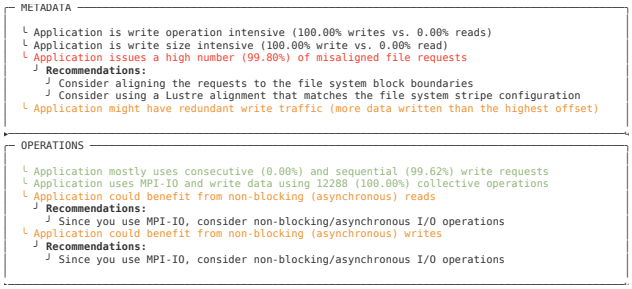

Fig. 5. *Drishti* report for the optimized E2E I/O kernels in Cori.

have multiple steps (we observed a median of 8.9 and up to $2,387$ steps in a job) and thus generate multiple Darshan files.

First, we noticed that 38.37% of the jobs in that period rely on STDIO for at least 10% of the total data size they transferred. Second, the vast majority of jobs (97.12%) do not use MPI-IO operation despite using MPI. It is important to notice that by default, in Cori, the system-provided versions of Darshan will only track applications that successfully called `MPI_Init()` and `MPI_Finalize()`. Thus, we can assume those represent MPI applications. Of those, we detected MPI-IO calls only in $3,483$ (2.88%).

We could also identify different bottlenecks for the POSIX interface using *Drishti*. We noticed that 33.84% and 57.26% of the jobs have a high number of sequential read and write operations, respectively. However, 23.60% of the jobs have a high random number of reads, which could degrade performance. Most concerning is the high occurrence of small requests (overall and in shared files), where we observed those small requests ($< 1MB$) comprising over than 10% of writes and reads. The redundant reads insight is also triggered by $14,518$ jobs. Furthermore, in over 40 thousand jobs *Drishti* detected data and time imbalance.

Among the jobs that relied on the MPI-IO interface, *Drishti* detected that 85.17% and 13.21% used write and read collective operations, respectively. A warning was issued for no non-blocking read and write in the logs in which MPI-IO calls are detected. The use of inter-node aggregators is also classified as high risk as many jobs unwillingly use such a setup, which could harm performance due to network communication costs and unbalanced workload among all ranks.

Finally, we measured the time taken to generate each of the $112,612$ reports. The total runtime depends on the size of the original Darshan profiling log and the number and complexity of the insights provided. For our current set of 30 checks described in Table III, *Drishti* takes a minimum of 0.02 seconds, a mean of 10.49 seconds, and a maximum of 134.98 seconds to generate a report. However, the third quartile (i.e., the value under which 75% of data points are found when arranged in increasing order) has a runtime of 17.77 seconds. This demonstrates that *Drishti* is a feasible solution to map metrics into well-known I/O bottlenecks and provide an initial set of recommendations.

TABLE III
SUMMARY OF INSIGHTS DETECTED BY DRISHTI USING MARCH 1 TO 5, 2022 DARSHAN LOGS FROM CORI.

Level	Interface	Detected Behavior	Jobs	Total (%)	Relative* (%)
HIGH	STDIO	High STDIO usage ($>$ 10% of total transfer size uses STDIO)	43,120	38.29	52.10
OK	POSIX	High number of sequential read operations ($\geq$ 80%)	38,104	33.84	58.14
OK	POSIX	High number of sequential write operations ($\geq$ 80%)	64,486	57.26	98.39
INFO	POSIX	Write operation count intensive ($>$ 10% more writes than reads)	26,114	23.19	39.84
INFO	POSIX	Read operation count intensive ($>$ 10% more reads than writes)	23,168	20.57	35.35
INFO	POSIX	Write size intensive ($>$ 10% more bytes written then read)	23,568	20.93	35.96
INFO	POSIX	Read size intensive ($>$ 10% more bytes read then written)	40,950	36.36	62.48
WARN	POSIX	Redundant reads	14,518	12.89	22.15
WARN	POSIX	Redundant writes	59	0.05	0.09
HIGH	POSIX	High number of small ($< 1MB$) read requests ($>$ 10% of total read requests)	64,858	57.59	98.96
HIGH	POSIX	High number of small ($< 1MB$) write requests ($>$ 10% of total write requests)	64,552	57.32	98.49
HIGH	POSIX	High number of misaligned memory requests ($>$ 10%)	36,337	32.27	55.44
HIGH	POSIX	High number of misaligned file requests ($>$ 10%)	65,075	57.79	99.29
HIGH	POSIX	High number of random read requests ($>$ 20%)	26,574	23.60	40.54
HIGH	POSIX	High number of random write requests ($>$ 20%)	559	0.50	0.85
HIGH	POSIX	High number of small ($< 1MB$) reads to shared-files ($>$ 10% of total reads)	60,121	53.39	91.73
HIGH	POSIX	High number of small ($< 1MB$) writes to shared-files ($>$ 10% of total writes)	55,414	49.21	84.55
HIGH	POSIX	High metadata time (at least one rank spends $>$ 30 seconds)	9,410	8.36	14.35
HIGH	POSIX	Data transfer imbalance between ranks causing stragglers ($>$ 15% difference)	40,601	36.05	61.95
HIGH	POSIX	Time imbalance between ranks causing stragglers ($>$ 15% difference)	40,533	35.99	61.84
WARN	MPI-IO	No MPI-IO calls detected from Darshan logs	109,569	97.30	-
HIGH	MPI-IO	Detected MPI-IO but no collective read operation	169	0.15	5.55
HIGH	MPI-IO	Detected MPI-IO but no collective write operation	428	0.38	14.06
WARN	MPI-IO	Detected MPI-IO but no non-blocking read operations	3,043	2.70	100.00
WARN	MPI-IO	Detected MPI-IO but no non-blocking write operations	3,043	2.70	100.00
OK	MPI-IO	Detected MPI-IO and collective read operations	402	0.36	13.21
OK	MPI-IO	Detected MPI-IO and collective write operations	2,592	2.30	85.17
HIGH	MPI-IO	Detected MPI-IO and inter-node aggregators	2,496	2.22	82.02
WARN	MPI-IO	Detected MPI-IO and intra-node aggregators	304	0.27	9.99
OK	MPI-IO	Detected MPI-IO and one aggregator per node	29	0.03	0.95

** Relative percentage of jobs that triggered an insight within the context of each I/O interface.*

VI. CONCLUSION

In this paper, we proposed *Drishti*, a solution to guide end-users in optimizing their applications by detecting typical performance I/O pitfalls and providing a set of recommendations. As a command-line tool with multiple report formats, it can easily be integrated into the day-to-day workflow when submitting jobs or as a service in online science gateways. *Drishti* seeks to close the translation gap between I/O metric collection and optimization techniques and tuning solutions to improve performance by empowering end-users and guiding them in the I/O optimization journey.

We demonstrate how *Drishti* can be helpful to end-users with two case studies and evaluate its robustness and performance by summarizing the issues detected in over a hundred thousand Darshan logs collected in Cori. *Drishti* is open-source and can be downloaded from `github.com/hpc-io/drishti`. We also provide an online companion website with detailed reproducible instructions used in this paper at `jeanbez.gitlab.io/pdsw-2022`.

In the future, we plan to add fine-grain data from Darshan DXT logs to provide more precise and meaningful insights considering the changes in application behavior through time.

ACKNOWLEDGMENT

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This research used resources of the National Energy Research Scientific Computing Center under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] The HDF Group. (1997-) Hierarchical Data Format, version 5. [Online]. Available: http://www.hdfgroup.org/HDF5

[2] J. Li, W. keng Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, "Parallel netCDF: A High-Performance Scientific I/O Interface," in *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, ser. SC'03, 2003.

[3] "Lustre File System," www.lustre.org. Accessed: August 5, 2022.

[4] "IBM Spectrum Scale (GPFS)," ibm.com/products/spectrum-scale. Accessed: August 5, 2022.

[5] 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 Int. Parallel Data Systems Workshop (PDSW)*, 2021, pp. 15–22.

[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 Trans. Storage*, vol. 7, no. 3, Oct. 2011.

[7] C. Xu, S. Snyder, O. Kulkarni, V. Venkatesan, P. Carns, S. Byna, R. Sisneros, and K. Chadalavada, "DXT: Darshan eXtended Tracing," *Cray Users Group Meeting*, 1 2019.

[8] A. Darshan, "pyDarshan." [Online]. Available: https://github.com/darshan-hpc/darshan/tree/main/darshan-util/pydarshan

[9] T. Wang, S. Byna, G. K. Lockwood, S. Snyder, P. Carns, S. Kim, and N. J. Wright, "A Zoom-in Analysis of I/O Logs to Detect Root Causes of I/O Performance Bottlenecks," in *2019 19th IEEE/ACM Int. Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2019, pp. 102–111.

[10] 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 *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18. IEEE Press, 2018.

[11] J. Yu, G. Liu, W. Dong, X. Li, J. Zhang, and F. Sun, "On the load imbalance problem of I/O forwarding layer in HPC systems," in *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, 2017, pp. 2424–2428.

[12] G. K. Lockwood, W. Yoo, S. Byna, N. J. Wright, S. Snyder, K. Harms, Z. Nault, and P. Carns, "UMAMI: A Recipe for Generating Meaningful Metrics through Holistic I/O Performance Analysis," in *Proceedings of the 2nd Joint International Workshop on Parallel Data Storage and Data Intensive Scalable Computing Systems*, ser. PDSW-DISCS '17, 2017.

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

[14] H. Sung, J. Bang, A. Sim, K. Wu, and H. Eom, "Understanding Parallel I/O Performance Trends Under Various HPC Configurations," in *Proceedings of the ACM Workshop on Systems and Network Telemetry and Analytics*, ser. SNTA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 29–36. [Online]. Available: https://doi.org/10.1145/3322798.3329258

[15] B. Behzad, H. V. T. Luu, J. Huchette, S. Byna, R. Aydt, Q. Koziol *et al.*, "Taming Parallel I/O Complexity with Auto-Tuning," in *SC'13: Proceedings of the International Conf. on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, pp. 1–12.

[16] R. McLay, D. James, S. Liu, J. Cazes, and W. Barth, "A user-friendly approach for tuning parallel file operations," in *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 229–236.

[17] B. Behzad, S. Byna, Prabhat, and M. Snir, "Optimizing I/O Performance of HPC Applications with Autotuning," *ACM Trans. Parallel Comput.*, vol. 5, no. 4, Mar. 2019.

[18] M. Agarwal, D. Singhvi, P. Malakar, and S. Byna, "Active Learning-based Automatic Tuning and Prediction of Parallel I/O Performance," in *2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW)*, 2019, pp. 20–29.

[19] 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)*, May 2020, pp. 1–8.

[20] T. J. Sheehan, A. D. Malony, and S. Shende, "A Runtime Monitoring Framework for the TAU Profiling System," in *Proc. of the Third Int. Symposium on Computing in Object-Oriented Parallel Environments*, ser. ISCOPE '99. Berlin, Heidelberg: Springer-Verlag, 1999, p. 170–181.

[21] T. Wang, S. Snyder, G. Lockwood, P. Carns, N. Wright, and S. Byna, "IOMiner: Large-Scale Analytics Framework for Gaining Knowledge from I/O Logs," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, 2018, pp. 466–476.

[22] G. K. Lockwood, N. J. Wright, S. Snyder, P. Carns, G. Brown, and K. Harms, "TOKIO on ClusterStor: Connecting Standard Tools to Enable Holistic I/O Performance Analysis," *Cray User Group*, 1 2018. [Online]. Available: https://www.osti.gov/biblio/1632125

[23] A. Bağbaba, "Improving Collective I/O Performance with Machine Learning Supported Auto-tuning," in *IEEE Int. Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 814–821.

[24] S. Snyder, P. Carns, K. Harms, R. Ross, G. K. Lockwood, and N. J. Wright, "Modular HPC I/O Characterization with Darshan," in *Proceedings of the 5th Workshop on Extreme-Scale Programming Tools*, ser. ESPT '16. IEEE Press, 2016, p. 9–17.

[25] J. Carretero, E. Jeannot, G. Pallez, D. E. Singh, and N. Vidal, "Mapping and Scheduling HPC Applications for Optimizing I/O," in *Proceedings of the 34th ACM International Conference on Supercomputing*, ser. ICS '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3392717.3392764

[26] 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 Int. Symp. on Cluster, Cloud and Internet Computing (CCGrid)*, 2022, pp. 61–70.

[27] K. Hou, R. Al-Bahrani, E. Rangel, A. Agrawal, R. Latham, R. Ross, A. Choudhary, and W.-k. Liao, "Integration of Burst Buffer in High-level Parallel I/O Library for Exa-scale Computing Era," in *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS)*, 2018, pp. 1–12.

[28] G. K. Lockwood, K. Lozinskiy, L. Gerhardt, R. Cheema, D. Hazen, and N. J. Wright, "Designing an All-Flash Lustre File System for the 2020 NERSC Perlmutter System," *University of California*, 10 2021. [Online]. Available: https://www.osti.gov/biblio/1824335

[29] X. Zhang, K. Liu, K. Davis, and S. Jiang, "iBridge: Improving Unaligned Parallel File Access with Solid-State Drives," in *Parallel and Distributed Processing Symposium, International*. Los Alamitos, CA, USA: IEEE Computer Society, may 2013, pp. 381–392. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/IPDPS.2013.21

[30] S. Kim, A. Sim, K. Wu, S. Byna, T. Wang, Y. Son, and H. Eom, "DCA-IO: A Dynamic I/O Control Scheme for Parallel and Distributed File Systems," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2019, pp. 351–360. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CCGRID.2019.00049

[31] F. Z. Boito, E. C. Inacio, J. L. Bez, P. O. A. Navaux, M. A. R. Dantas, and Y. Denneulin, "A Checkpoint of Research on Parallel I/O for High-Performance Computing," *ACM Comput. Surv.*, vol. 51, no. 2, mar 2018. [Online]. Available: https://doi.org/10.1145/3152891

[32] B. Xie, J. Chase, D. Dillow, O. Drokin, S. Klasky, S. Oral, and N. Podhorszki, "Characterizing Output Bottlenecks in a Supercomputer," in *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 1–11.

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

[34] N. Tavakoli, D. Dai, and Y. Chen, "Client-Side Straggler-Aware I/O Scheduler for Object-Based Parallel File Systems," *Parallel Comput.*, vol. 82, no. C, p. 3–18, feb 2019. [Online]. Available: https://doi.org/10.1016/j.parco.2018.07.001

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

[36] H. Tang, Q. Koziol, S. Byna, J. Mainzer, and T. Li, "Enabling Transparent Asynchronous I/O using Background Threads," in *2019 IEEE/ACM Fourth Int. Parallel Data Systems Workshop (PDSW)*, 2019, pp. 11–19.

[37] B. Nicolae, A. Moody, E. Gonsiorowski, K. Mohror, and F. Cappello, "VeloC: Towards High Performance Adaptive Asynchronous Checkpointing at Large Scale," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2019, pp. 911–920. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/IPDPS.2019.00099

[38] 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.

[39] 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

[40] 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

[41] Huebl, Axel and Lehe, Remi and Vay, Jean-Luc and Grote, David P. and Sbalzarini, Ivo F. and Kuschel, Stephan and Sagan, David and Mayes, Christopher and Perez, Frederic and Koller, Fabian and Bussmann, Michael. (2015) openPMD: A meta data standard for particle and mesh based data. [Online]. Available: https://doi.org/10.5281/zenodo.1167843

[42] Koller, Fabian and Poeschel, Franz and Gu, Junmin and Huebl, Axel. (2019) openPMD-api 0.10.3: C++ & Python API for Scientific I/O with openPMD. DOI: 10.14278/rodare.209. [Online]. Available: https://doi.org/10.14278/rodare.209

[43] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, "Flexible IO and Integration for Scientific Codes through the Adaptable IO System (ADIOS)," in *CLADE*. NY, USA: ACM, 2008, pp. 15–24.

[44] J. Lofstead, M. Polte, G. Gibson, S. Klasky, K. Schwan, R. Oldfield, M. Wolf, and Q. Liu, "Six Degrees of Scientific Data: Reading Patterns for Extreme Scale Science IO," in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, ser. HPDC '11. New York, NY, USA: ACM, 2011, p. 49–60.