GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

HPS

Seminar Report

I/O Analysis using Darshan

Zoya Masih

MatrNr: 19034321

Supervisor: Prof. Dr. Julian Kunkel

Georg-August-Universität Göttingen
Institute of Computer Science

April 3, 2023

Abstract

The need for powerful, on demand, scalable storage systems in the area of HPC, has been increasingly growing. To meet such a requirement, profiling tools with the capability of capturing detailed application-level behavior are essential. Darshan is such a tool which simplify the task of understanding and tuning I/O behavior. It helps to get an accurate picture of application I/O by showing access patterns, sizes, number of operations etc. with minimum overhead.

In this report, Darshan profiling tool is introduced, and the setup steps are described. IO500 benchmark is also shortly introduced and instrumented for Darshan. The results of capturing details of the benchmark are then presented and analyzed. In this experiment, IO500 is run on two working nodes of GWDG clusters, and the Darshan log files are presented for both regular and extended tracing (DXT) configurations.

Contents

List of Tables

List of Figures

List of Listings

List of Abbreviations

DXT Darshan eXtended Tracing

GWDG Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

HPC High-Performance Computing

IO Input/Output

MPI Message Passing Interface

POSIX Portable Operating System Interface for uniX

1 Introduction

In the field of High Performance Computing (HPC), we deal with powerful computing systems and software applications to solve complex computational problems that require significant amounts of processing power, memory, and storage. The applications in this field demand robust storage systems. Storage systems must meet the concurrent I/O requirements, e.g., accessing to hundreds of thousands of compute elements. In such a situation, analyzing application characteristics is an important aspect of HPC because it helps to identify the computational requirements of an application and ensure that the hardware and software infrastructure can meet those requirements. In other words, understanding storage access characteristics of computational science applications is critical in storage optimization to increase the efficiency of the storage system. In HPC, applications can be highly parallel and may require large amounts of memory, storage, and network bandwidth. Therefore, analyzing the characteristics of an application is essential to select the appropriate hardware and software infrastructure to support the application's need.

In 2009, in the areas of memory and communication subsystem behavior, many analyzing tools, Jumpshot [WD99] as an example, were available that provided insight into how an application is interacting with the subsystem. Unlikely, similar tools were not available for I/O. In fact, there were an overall lack of understanding of how computational science applications interact with the storage system. This shortage was a main motivation for providing an I/O characterization tool named Darshan, developed at Argonne National Lab [LR09]. Darshan is designed to capture an accurate picture of application I/O behavior, including properties such as patterns of access within files, with the minimum possible overhead. This characterization can shed important light on the I/O behavior of applications at extreme scale. Darshan also can enable researchers to gain greater insight into the overall patterns of access exhibited by such applications, helping the storage community to understand how to best serve current computational science applications and better predict the needs of future applications. Darshan captures MPI-IO routines using the profiling (PMPI) interface to MPI. POSIX routines are captured by inserting wrapper functions via the GNU linker's `- -wrap` argument. Darshan invokes no communication or storage routines until the end of the job. It therefore reduces the scope of the scalability challenge to a single shutdown routine.

The current report is describing all the procedures done for a project in the seminar course 'Newest Trends in High-Performance Data Analytics'. The goals which were expected to be fulfilled during this project are as follows.

1. Setting up Darshan
2. Instrumenting IO500 benchmark
3. Obtaining text and PDF results of profiling IO500 for regular and DXT configuration
4. Analyzing the results

The outline of the report is as follows. In Section two, the process of setting up Darshan on the compute nodes of GWDG and instrumenting the benchmark IO500 is defined. The test system then is defined in section 3. Finally, The outputs of Darshan are analyzed

in section 4. At the end of some subsections, challenges, and some points are discussed under the title Note. As usual, the report is closed with a conclusion and references.

2 Introduction to Darshan

In this section, we will explain how to set up Darshan and how to instrument the benchmark we are going to analyze.

2.1 Installation

For the first steps, one can start with downloading.

1. `wget https://ftp.mcs.anl.gov/pub/darshan/releases/darshan-3.4.1.tar.gz`
2. `tar -xvzf darshan-3.4.1.tar.gz`
3. `cd darshan-3.4.1/`
4. `./prepare.sh`

The Darshan source tree is divided into two parts. The first one is darshan-runtime, which is installed on systems where you intend to instrument MPI applications. To install that, one need the following steps.

1. `cd darshan-runtime/`
2. `./configure - -with-log-path=/path/file/ - -with-jobid-env=SLURM _JOB _ID -prefix=/scratch/users/zmasih/darshan/ CC=mpicc`
3. `make & make install`

The second one is darshan-util, which is installed on systems where you intend to analyze log files produced by darshan-runtime.

1. `cd ../darshan-util/`
2. `configure - -prefix=/scratch/users/zmasih/darshan/`
3. `make & make install`

2.1.1 Note

After the first installation, the following error occurred. `Error: WARNING: The POSIX module contains incomplete data! This happens when a module runs out of memory to store new record data.` Solutions to this problem could be by changes to allow the Darshan per-module memory limit configurable using the following methods:

1. at runtime, using the 'DARSHAN_MODMEM' environment variable, which should be set to the desired memory limit in MiB The commits for this enhancement are 98c93e0f (darshan-runtime changes) and 40d1dc03 (darshan-util changes).
2. `-with-max-records=<max _record _count>'` to change the default maximum number of records to track (2048 currently).

In my experience, adding `-with-mod-mem=500` to the configuration line solved the problem.

2.2 Instrumenting IO500

When the installation is complete, one may need to instrument the application which needs to be analyzed. In the following experiment, the IO500 benchmark is instrumented in the following steps.

1. Add `CC =/path/to/darshan-3.4.1/mpicc.darshan` to Makefile in io500 directory
2. `ldd io500`
3. `export PATH=$ PATH:/darshan/bin/path`
4. make a directory for the log files, in the path you defined in the configuration time.
5. `export DARSHAN_LOGPATH=/log file/path`
6. in the submission script file, put `export DARSHAN _CONFIG _PATH=darshan.config`
7. `make`

After the aforesaid steps are done, running the application as normal generates a logfile which is used for analyzing the performance of the instrumented application.

3 The Test System

In this section we will describe the system of the experiment, and also will briefly describe the application which is instrumented for Darshan.

3.1 The Hardware

The compute cluster in GWDG is divided into frontends and compute nodes. To run a program on one or more of the compute nodes, it is needed to interact with the batch system, or scheduler, Slurm. Nodes are also grouped in fat, medium and gpu partitions. In this experiment, we ran IO500 benchmark on the SCC clusters of GWDG, with 10 tasks per node, and set the job to run for 10 minutes in medium partition, see fig. 1.

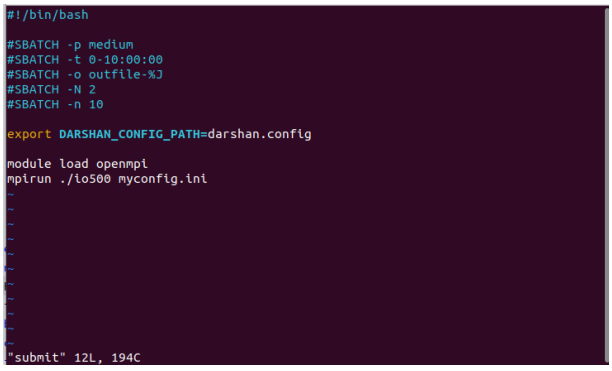


Figure 1: The IO500 job

3.2 Benchmark IO500

In this project, the application which is instrumented for Darshan is IO500.
The IO-500 benchmark consists of data and metadata benchmarks to identify performance boundaries for optimized and suboptimal applications. IO500 provides five main measurement scenarios using IOR and mdtest as follows [KT21].

- IOR Easy: For the applications with well optimized I/O patterns. Free to tune IOR parameters. Typically, file-per-process, large, aligned chunks to get the best possible bandwidth performance
- IOR Hard: For the applications that require a random workload- Limited options to tune. Forced to use small unaligned I/O to a single shared file for the worst possible bandwidth performance.
- MDtest Easy: For Metadata and small objects. Free to tune mdtest parameters with zero size files in separate directory per process to represent best case scenario for metadata rate
- MDtest Hard: For Small files (3901 bytes) in a shared directory- Limited options to tune. Forced all processes to write on a single shared directory. Representing worst case scenario for metadata rate
- Find: Finding a specific subset of files from those created by the four scenarios.

4 IO500 Analysis by Darshan

4.1 darshan-parser

Each time a darshan-instrumented application is executed successfully, a log file is automatically generated in the path defined by `with-log-path` in the configuration time. The command line utility `darshan-parser` can be used to obtain a human-readable, text-format output of all information contained in a log file.

```
zoya@zoya-Latitude-7420:~$ darshan-parser zmasih_io500_id15155848-497182_3-8-55773-7894066996975896480_1.darshan
# darshan log version: 3.41
# compression method: ZLIB
# exe: ./io500 myconfig.ini
# uid: 711947
# jobid: 15155848
# start_time: 1678285773
# start_time_asci: Wed Mar  8 15:29:33 2023
# end_time: 1678314671
# end_time_asci: Wed Mar  8 23:31:11 2023
# nprocs: 10
# run time: 28897.3787
# metadata: lib_ver = 3.4.1
# metadata: h = romio_no_indep_rw=true;cb_nodes=4
```

Figure 2: log file generated by darshan-parser, part 1

Figure 2 shows the first part of output of this command in the current experience. This part of the output displays a summary of overall job information. Additional options

can also be used to produce more information. For instance, `- -perf` generates job performance information. The second part of the output reports the size of each region contained within the given log file, Figure 3.

```
# log file regions
# ---------------------------------------------------------
# header: 1328 bytes (uncompressed)
# job data: 419 bytes (compressed)
# record table: 1482 bytes (compressed)
# POSIX module: 2619 bytes (compressed), ver=4
# STDIO module: 1172 bytes (compressed), ver=2
# HEATMAP module: 5828 bytes (compressed), ver=1
```

Figure 3: log file generated by darshan-parser, part 2

The next part shows a table of all general purpose file systems that were mounted while the job was running. Each line uses the format <mount point> <fs type>.
The remainder of the output will show characteristics for each file that was opened by the application. Each line uses the following format: <module> <rank> <record id> <counter name> <counter value> <file name> <mount point> <fs type>. These parts are not presented in this report due to their length.

4.1.1 Note

If the error `darshan-parser command not found` appears, `export PATH=$PATH:/bin/path/` is required.
In the previous versions, the log file was provided directly to the defined log directory, however, in the latest versions, in the log directory two directories with name 2023 and 2024 are generated, and inside directory 2023, you can follow the subdirectories according to the date of job execution.

4.2 PDF Output

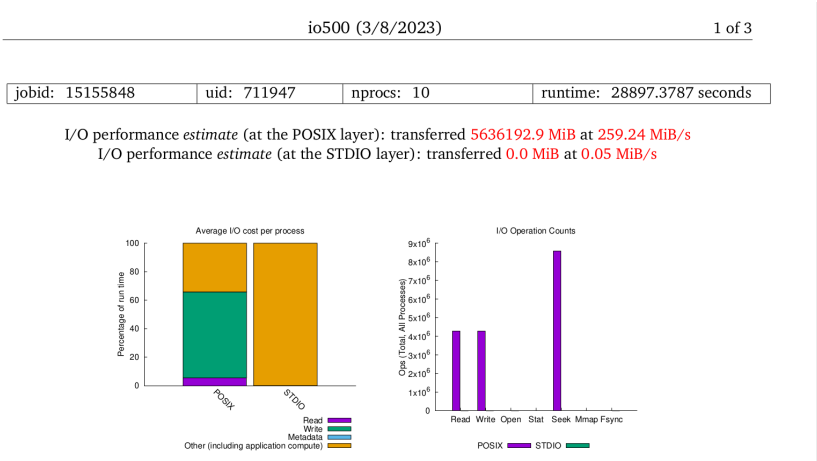


Figure 4: PDF output - part1

The tool `darshan-parser` generates a complete, human-readable output, however it is lengthy and challenging to understand and analyze. Therefore, a visualization may help

to analyze more conveniently. The command `darshan-job-summary.pl` generates a PDF result file.
As shown in Figure 4, total data transferred and I/O bandwidth observed at POSIX layer are 5636192.9 MiB and 259.24 MiB/s. In this experiment, read took smaller fraction of the total execution time rather than write at POSIX layer. From the right side picture also, we notice that read, write, and seek operations are performed using POSIX, and POSIX read operation count is equal to POSIX write operation count, and the number is far smaller than the seek operations.

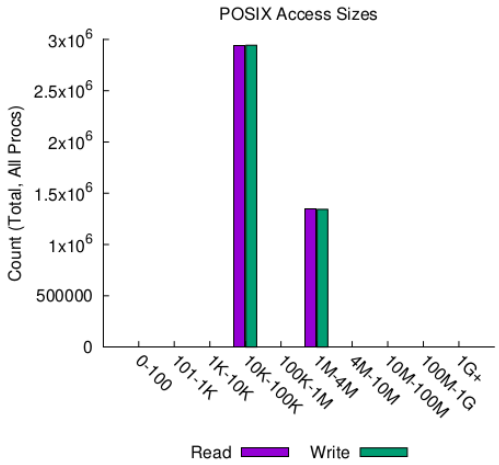


Figure 5: PDF output - part2

The next histogram, 5, represents the number of POSIX operations and their associated sizes. In our example, most of the I/O operations are of 10-100 kilo Bytes, and the reminder of the operations are of size 1-4 mega Bytes. In total, there are approximately 4.5×10^6 operations for both read and write.

Most Common Access Sizes (POSIX or MPI-IO)

	access size	count
POSIX	47008	5887820
	2097152	2686120

File Count Summary (estimated by POSIX I/O access offsets)

type	number of files	avg. size	max size
total opened	28	89GiB	263GiB
read-only files	9	234GiB	263GiB
write-only files	15	18GiB	263GiB
read/write files	3	43GiB	129GiB
created files	18	22GiB	263GiB

Figure 6: PDF output - part 3

In the next picture, 6 shows the most common I/O sizes at POSIX layer. There is also a second table which provides the number of created files, read/write-only and read/write files. Here we have 18 created files with the average size 22 GiB, whereas we have 15 write only files with avg/max size 18 GiB.
Figure 7 shows the timespan for individual files from the first to the last access. It is seen that each file was accessed for around 25 minutes. Figure 8 is the same as the previous one, however, it represents the timespan for shared files.

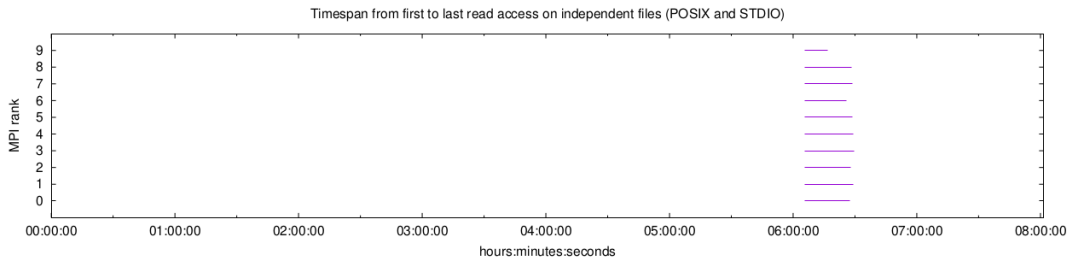


Figure 7: PDF output - part 4

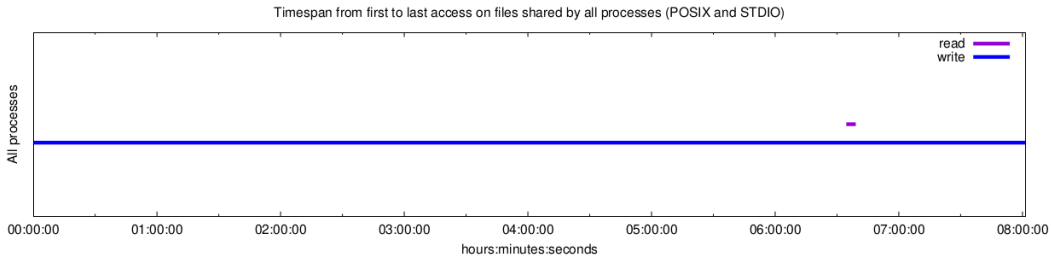


Figure 8: PDF output - part 5

Cumulative time spent and the amount of I/O functions are shown in the Figure 9. In this execution, for instance, we have 262.3 GiB independent reads with around 21.6 minutes timespan, whereas the numbers for shared file reads are 12.88 GiB and 4.6 minutes. Figure 10 shows the I/O pattern (type of offsets) for each file opened by the application. In our benchmark, for all 4.2 million operations, the offsets are sequential, and around 1.3 million of them have simultaneously consecutive access.

Average I/O per process (POSIX and STDIO)

	Cumulative time spent in I/O functions (seconds)	Amount of I/O (MiB)
Independent reads	1295.845167	268612.000588322
Independent writes	797.9272159	268612.002606297
Independent metadata	0.3005687	N/A
Shared reads	274.1427077	13197.6434020996
Shared writes	16638.461706	13197.6435134888
Shared metadata	0.443241	N/A

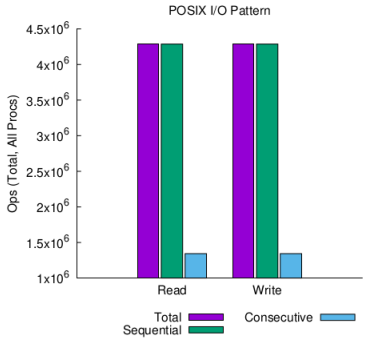
Data Transfer Per Filesystem (POSIX and STDIO)

File System	Write		Read	
	MiB	Ratio	MiB	Ratio
/scratch1	2818096.46008	1.00000	2818096.43990	1.00000
UNKNOWN	0.00111	0.00000	0.00000	0.00000

Figure 9: PDF output - part 6

4.2.1 Note

The `darshan-job-summary` tool depends on a few LaTeXpackages including lastpage, subfigure, and threeparttable. The packages are not currently available on the GWDG clusters, therefore one may need to copy the log file to another system, in which darshan-util is already installed on, to be able to generate the PDF outputs. To copy the log file,



sequential: An I/O op issued at an offset greater than where the previous I/O op ended.
consecutive: An I/O op issued at the offset immediately following the end of the previous I/O op.

Figure 10: PDF output - part 7

command `cp <log file name> destinationpath` is used.

It is possible that an older version of Darshan is not able to make the PDF for a log file generated by a newer version. In such a case, make sure that you are using the new version's library, to check that, the command `which darshan-parser` may be of a help.

4.3 DXT: Darshan eXended Tracing

Darshan's default characterization mechanism records information at a fixed granularity. In 2017, Darshan was augmented by proposing Darshan eXtended Tracing (DXT) for more detailed profiling of I/O software stacks [SC17]. DXT enables users and administrators to vary the level of fidelity captured by Darshan at run time without modifying or recompiling applications. This capability facilitates systematic analysis of the I/O behavior of applications and can provide useful application kernel I/O traces to help advance parallel I/O research.

DXT may exhibit higher runtime and memory overheads, so it is disabled by default in Darshan. By setting the environment to `export DXT_ENABLE_IO_TRACE=1` in the runtime, a DXT log file is generated. After the application is successfully executed, `darshan-dxt-parser` provides the DXT output.

4.3.1 DXT Explorer

DXT Explorer [Bez+21] is an interactive log analysis tool. Using this tool requires a Darshan log file collected with tracing data. To use DXT Explorer, Python 3 and R are also needed to be installed. In the first execution ever, DXT Explorer automatically downloads any missing, required R packages. the command `dxt-explore` then generates an explore.html file with an interactive plot that you can open in any browser to explore.

4.4 IO500 output

The output of IO500 job in the current experiment is shown in Figure 11. As we can see in the picture, the total time spending for ior-easy/hard write, is around 2000 seconds, whereas the time for ior-easy/hard read is 1700 seconds. The comparison supports the

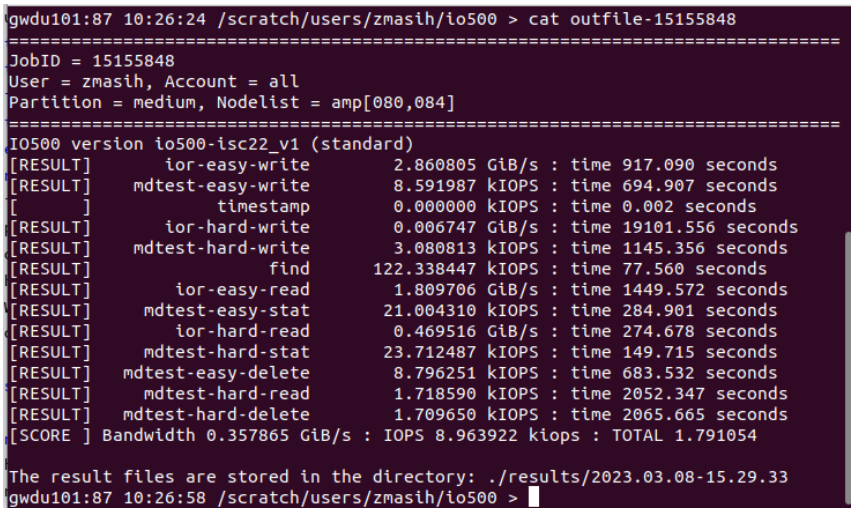


Figure 11: IO500 output

results in figure 4, which showed 60 percent of total runtime for write, in comparison to 5 percent for read.
More details on all IO500 scenarios can also be reached in the path `/io500/results`. In the current execution, the results on ior-easy write are shown in Figure 12. In this figure, for instance, we can see that the ordering in a file is sequential, which was already discussed in Figure 10 of Darshan output. We also noticed in Figure 5 that there is no file larger than 4 MB. In the support of that info, we can see that the xfersize is 2 MiB. This parameter refers to the size of the I/O operations that are being performed by the benchmark, and as regards to the point that ior-easy is referring to large files, we have the approval.

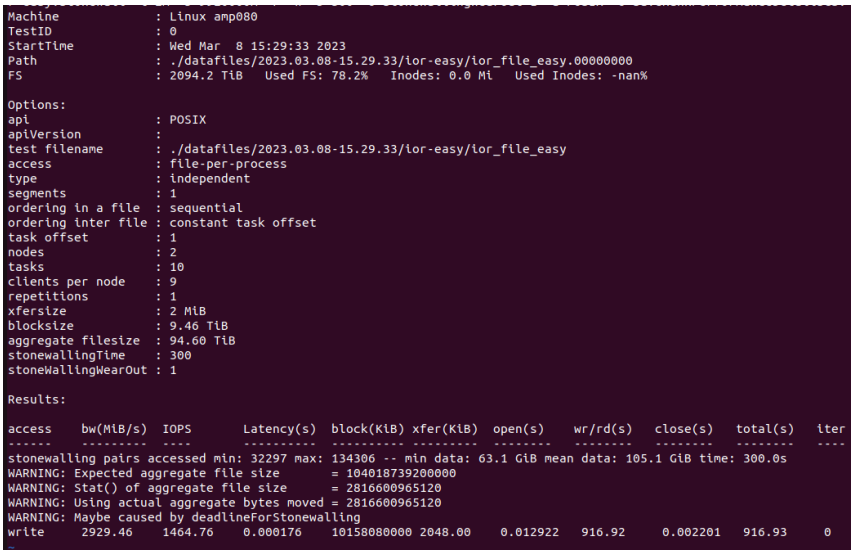


Figure 12: ior-easy-write

5 Conclusion

The profiling tool Darshan was introduced in 2009, in order to help to understand and tuning I/O behavior on extreme-scale systems. Darshan can produce a log file for each successful execution of an IO application. The log file then can be provided in a text or PDF format, for a regular or more detailed log file (DXT).
In this project, Darshan was installed on GCC cluster of GWDG and then IO500 benchmark was instrumented for it. The job was run on two compute nodes, and with 10 processes per node. This report described the installation and instrumentation steps, and then presented and analyzed the results.
Most of the operations in this experiment were of read, write, and seek in POSIX standard, and the majority of the run time, almost 60 percent, was related to the write operations. There were sequential access to all the files during the job was running, and all the files were of size 10-100 KB, or 1-4 MB.

References

[Bez+21] Jean Luca Bez et al. "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. DOI: 10.1109/PDSW54622.2021.00008.

[KT21] Radita Liem. Dmytro Povaliaiev. Jay Lofstead. Julian Kunkel and Christian Terboven. "User-centric system fault identification using IO500 benchmark". In: *2021 IEEE/ACM Sixth International Parallel Data Systems Workshop (PDSW)* (2021), pp. 35–40.

[LR09] Philip Carns. Robert Latham. Robert Ross. Kamil Iskra. Samuel Lang and Katherine Riley. "24/7 Characterization of Petascale I/O Workloads". In: *2009 IEEE International Conference on Cluster Computing and Workshops* 13.2 (2009), pp. 1–10.

[SC17] Cong Xu. Shane Snyder. Vishwanath Venkatesan. Philip Carns. Omkar Kulkarni. Suren Byna. Roberto Sisneros and Kalyana Chadalavada. "Dxt: Darshan extended tracing". In: *Argonne National Lab.(ANL), Argonne, IL (United States)* (2017).

[WD99] O.Zaki. E.Lusk. W.Gropp and D.Swider. "Toward scalable performance visualization with Jumpshot". In: *High Performance Computing Applications* 13.2 (1999), pp. 277–288.