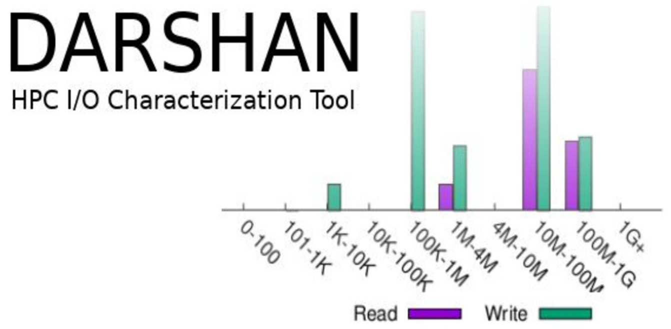
HPE Cray User Group

May 8, 2025

Expanding Community Access to Real-world HPC Application I/O Characterization Data Using Darshan



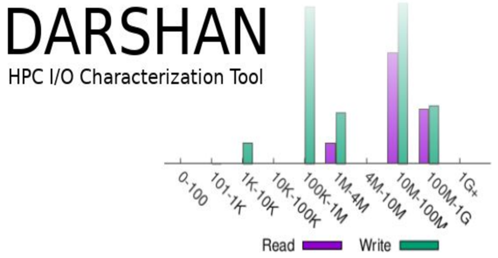
Shane Snyder, **Philip Carns**, Kevin Harms, Robert Latham, Robert Ross

Argonne National Laboratory

https://www.mcs.anl.gov/research/projects/darshan/

Roadblocks to understanding HPC I/O

- Application I/O performance is remarkably difficult to interpret.
  - Applications are diverse, systems are complex, and platform specifications are unrealistic reference points.
- The **Darshan** I/O characterization tool seeks to address this problem by providing insight into application-level behavior in production.
  - It transparently captures concise information about file use, access patterns, I/O time, and more.
- Despite big strides in this area, we still have significant problems:
  - **For CS researchers**: how can I identify trends, perform comparisons, or model behaviors that extend beyond my own personal experiments?
  - **For facility operators**: how can I translate data into practical, actionable, productive optimizations?



Problem statement and objective

Traditionally there were two ways to explore a broad collection of Darshan logs:

1. Work at (or with) a facility using internal data sets.
   - Manpower is limited and collaboration is difficult.
2. Study public community log collections and repositories.
   - (e.g., ALCF, NCSA, OLCF, JGU Mainz, and others)
   - These provide full access, but log sets are limited or quickly out of date.

This presentation presents a third, complimentary option: automated continuous workflows to anonymize, aggregate, and publish Darshan logs, along with public tools for aggregate analysis.

We invite researchers and facility operators not only to reproduce our results, but to find novel uses for the data!

Experimental platform

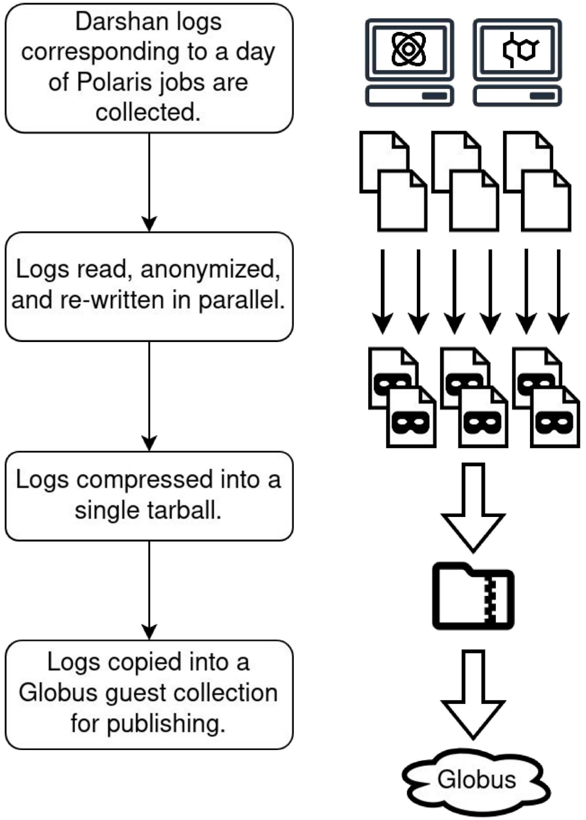
- **Polaris** is a 560 node production HPC platform at the Argonne Leadership Computing Facility with 4 GPUs per node and a Lustre file system.
- Darshan logs (one per application execution) are archived on Polaris in a shared /year/month/day directory structure.
- Users can only access their own logs, but the administrators can access the entire population. This is a typical and recommended deployment configuration.
- At the time this paper was written, over 700,000 Darshan logs had been collected on Polaris.

Darshan log collection workflow

Darshan log collection workflow

We deployed an automated workflow to collect, anonymize, and publish Darshan logs on ALCF Polaris:

- ❖ Implemented via pipelines on ALCF's GitLab CI instance, which uses Jacamar CI to provide Polaris runners
- ❖ Runs nightly to process previous days' log data
- ❖ Resilient to both system and GitLab downtime
- ❖ CI pipeline provides informative status and error reporting for monitoring workflow health over time

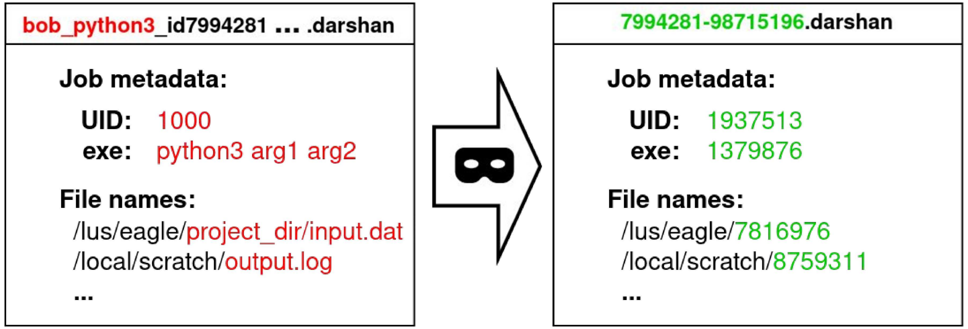

Darshan logs corresponding to a day of Polaris jobs are collected.
Logs read, anonymized, and re-written in parallel.
Logs compressed into a single tarball.
Logs copied into a Globus guest collection for publishing.
Globus

General log collection workflow

Darshan log collection workflow

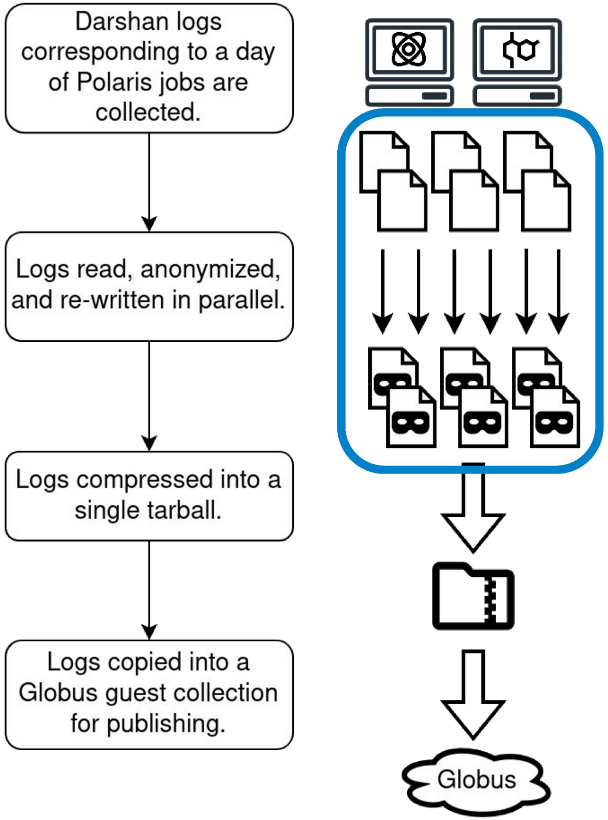
Log anonymization is accomplished by masking sensitive, user-identifying information in the log file name and within the log itself:

- username/UID
- executable + cmd line args
- file name suffixes (i.e., file name beyond the mount point)



```
bob_python3_id7994281 ... .darshan

Job metadata:
  UID:  1000
  exe:  python3 arg1 arg2

File names:
  /lus/eagle/project_dir/input.dat
  /local/scratch/output.log
  ...
```

```
7994281-98715196.darshan

Job metadata:
  UID:  1937513
  exe:  1379876

File names:
  /lus/eagle/7816976
  /local/scratch/8759311
  ...
```



1. Darshan logs corresponding to a day of Polaris jobs are collected.
2. Logs read, anonymized, and re-written in parallel.
3. Logs compressed into a single tarball.
4. Logs copied into a Globus guest collection for publishing.

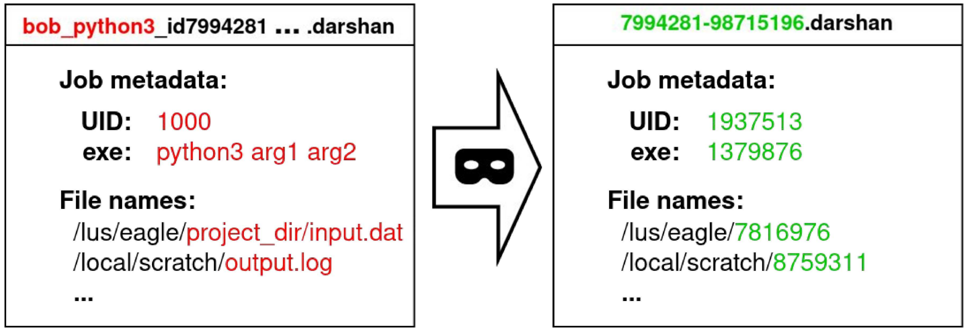
General log collection workflow

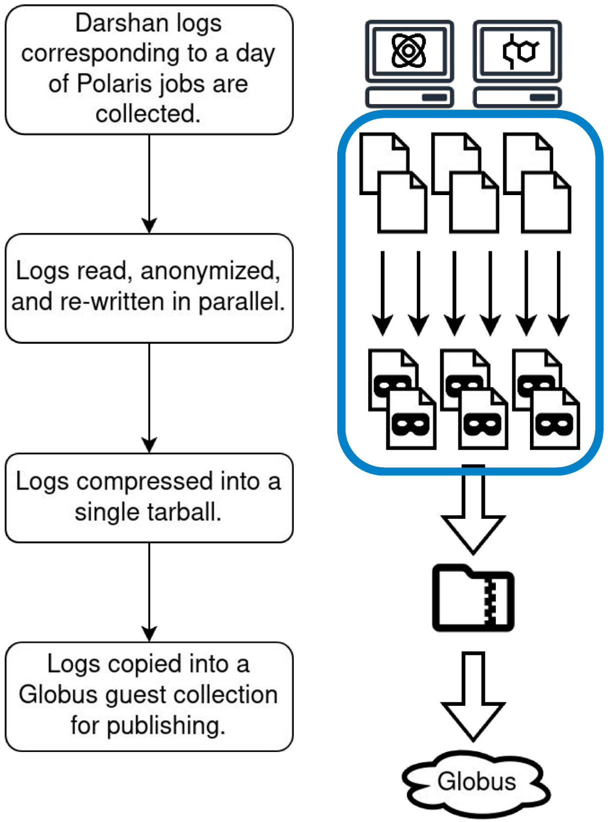
This is done with the `darshan-convert` command line tool.

Darshan log collection workflow

Job scheduler IDs are not anonymized, to facilitate correlation with publicly available ALCF job scheduler datasets for Polaris.

https://reports.alcf.anl.gov/data/polaris.html





Darshan logs corresponding to a day of Polaris jobs are collected.
Logs read, anonymized, and re-written in parallel.
Logs compressed into a single tarball.
Logs copied into a Globus guest collection for publishing.
Globus

General log collection workflow

Darshan log collection workflow

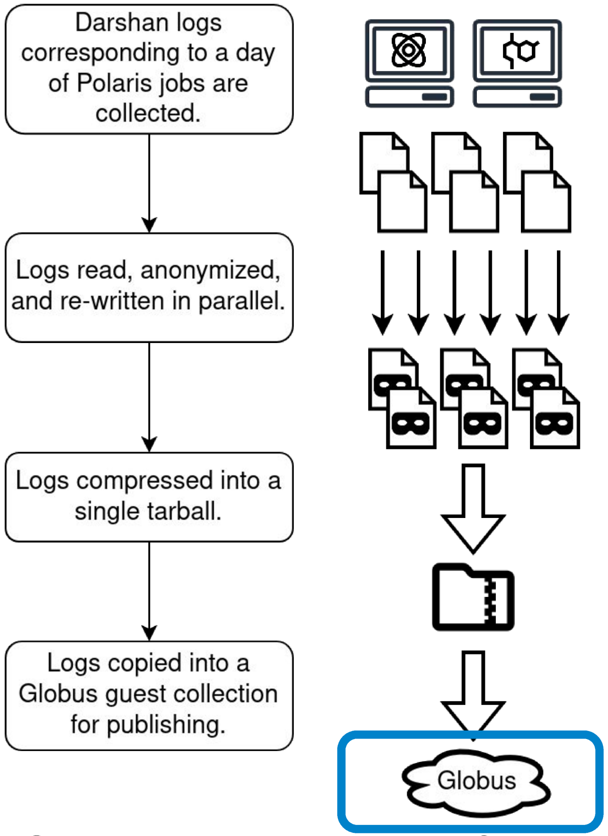
The log collection is published using Globus. Daily collections of logs are combined into a single tar file.

The log collection documentation contains a link to the Globus guest collection and an overview of the data contained in it.

zenodo.org/records/15052603

We've also performed an initial analysis of the logs that have been published so far (next slides). The scripts to reproduce that analysis are available on GitHub.

https://github.com/darshan-hpc/polaris-log-collection-scripts-cug25


Darshan logs corresponding to a day of Polaris jobs are collected.
Logs read, anonymized, and re-written in parallel.
Logs compressed into a single tarball.
Logs copied into a Globus guest collection for publishing.
Globus

General log collection workflow

Example analysis: Instrumentation coverage

Darshan coverage of Polaris jobs

Note that Darshan does not provide full coverage of all job I/O activity on Polaris.

- The Polaris Darshan log collection therefore offers insight into a smaller subset of jobs running on the system.

Darshan can fail to capture instrumentation data for a number of different reasons:

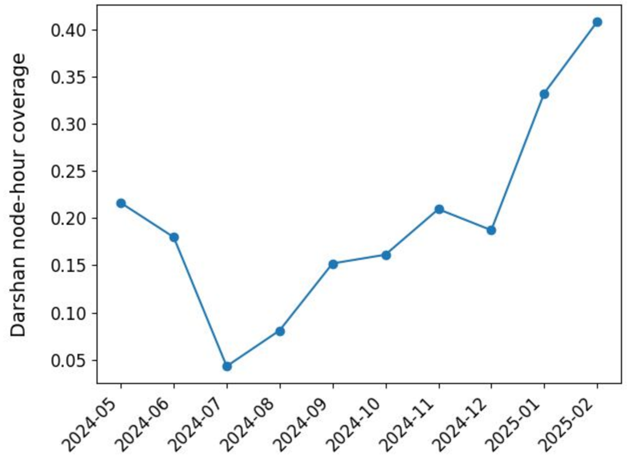
- Jobs that explicitly disable Darshan at build or run time
- Jobs that don't use MPI
  - Darshan traditionally relies on hooks into `MPI_Init()`/`MPI_Finalize()` to start/stop instrumentation.
- Jobs that hit their wall time
  - When hitting wall time, jobs are abruptly terminated and don't have a chance to shutdown cleanly.

Darshan coverage trends

In order to better understand job coverage, we broke it down into finer granularity:

- What does coverage look like from month to month? (see figure at right)
- What does coverage look like per project allocation?

Key finding: Darshan job coverage primarily depends on the level of instrumentation in high-impact projects during the observed timeframe (e.g., large-scale scientific campaigns with high computational demands).



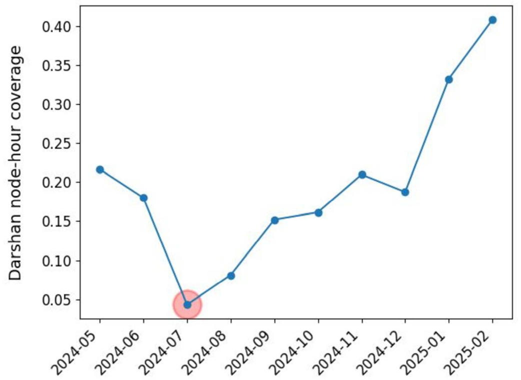
Darshan coverage of Polaris jobs over the 10-month period analyzed in this work

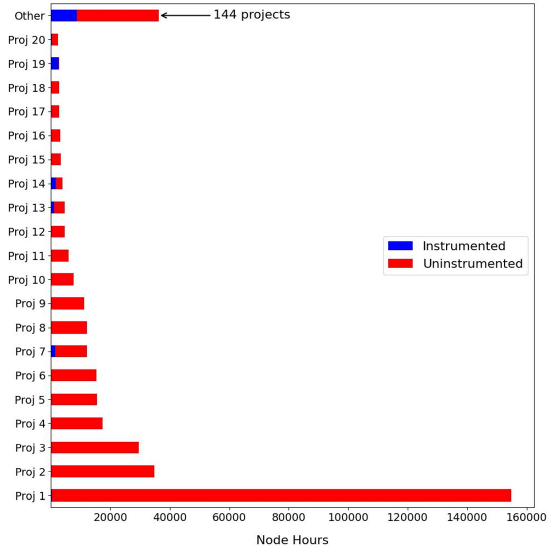
Example month with low coverage

We observed unusually low coverage of **<5%** in July '24.

20 projects account for 90% of node-hour coverage, with 144 smaller projects accounting for remaining 10% of node-hours.

The Darshan coverage was skewed by the largest of these projects (the single largest project accounts for 40% of all node-hours for this month).



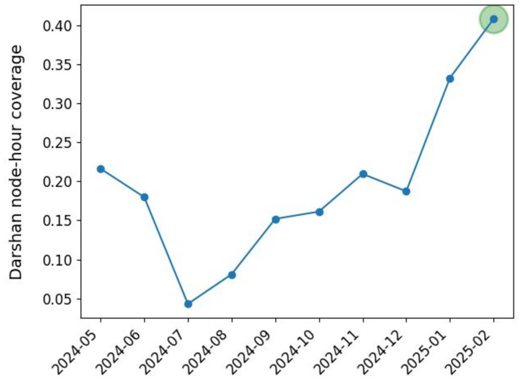


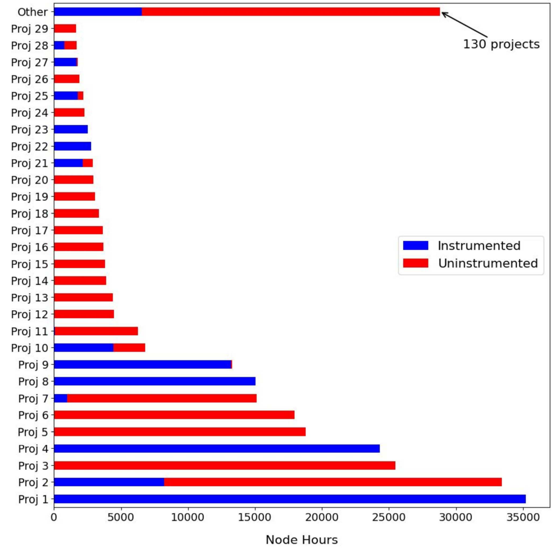
Example month with high coverage

We observed unusually high coverage of **>40%** in February ‘25.

29 projects account for 90% of node-hour coverage, with 130 smaller projects accounting for remaining 10% of node-hours.

In contrast with July ‘24, many of the largest projects did use Darshan (and the overall node hour usage is not as skewed).





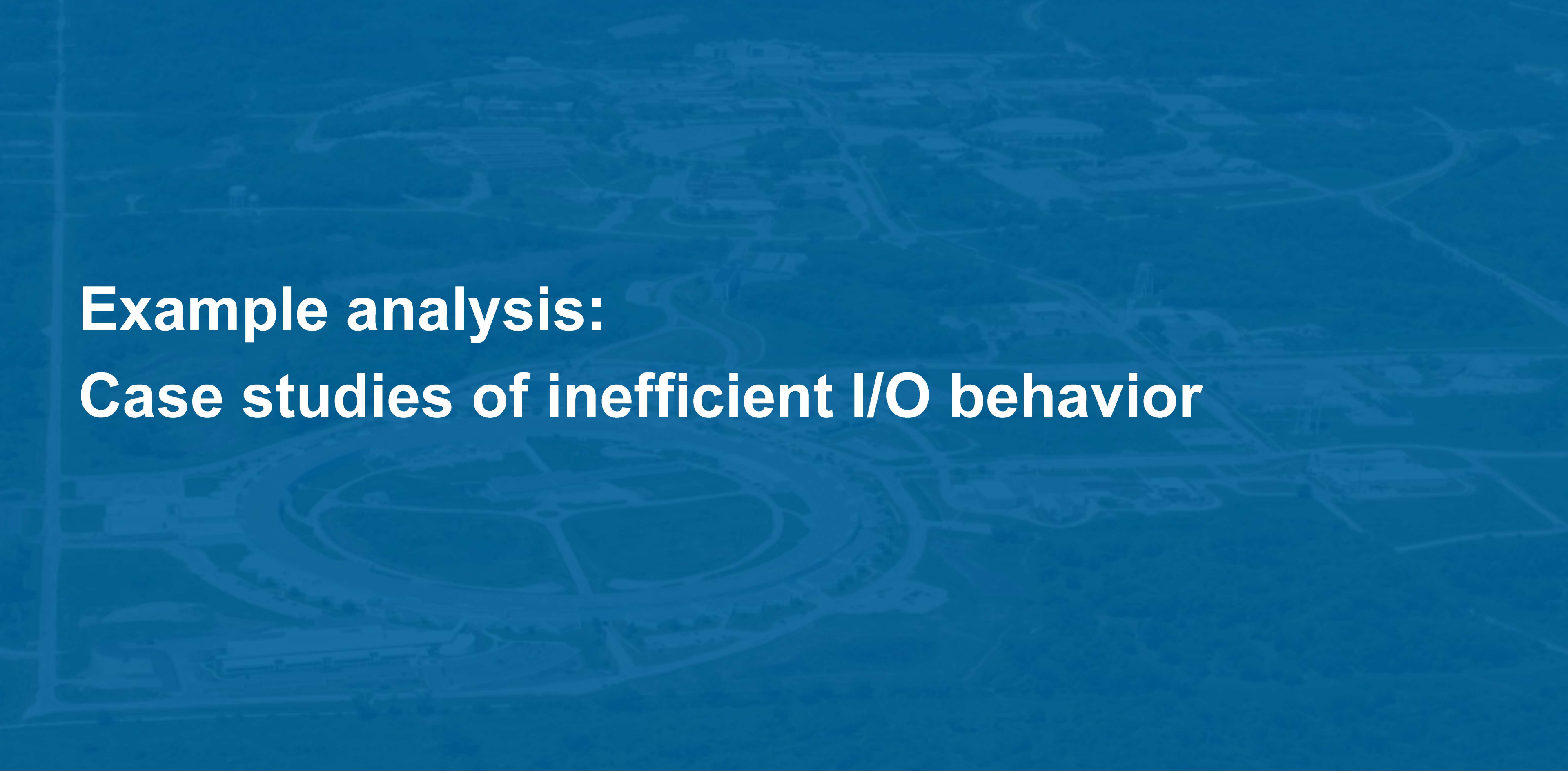
Methods to improve Darshan's coverage

Darshan does have features that can help improve coverage with additional deployment effort, but they have not been utilized in production yet.

- ❖ Extensions enabling instrumentation of generic non-MPI applications
- ❖ `mmap`-based logging to ensure capture of instrumentation data in case of abrupt termination (e.g., job hitting wall time)

Working with facilities to enable additional Darshan features improve coverage of production applications.

Further, understanding why high-impact projects lack Darshan instrumentation will reveal previously unknown instrumentation blind spots.



Example analysis:
Case studies of inefficient I/O behavior

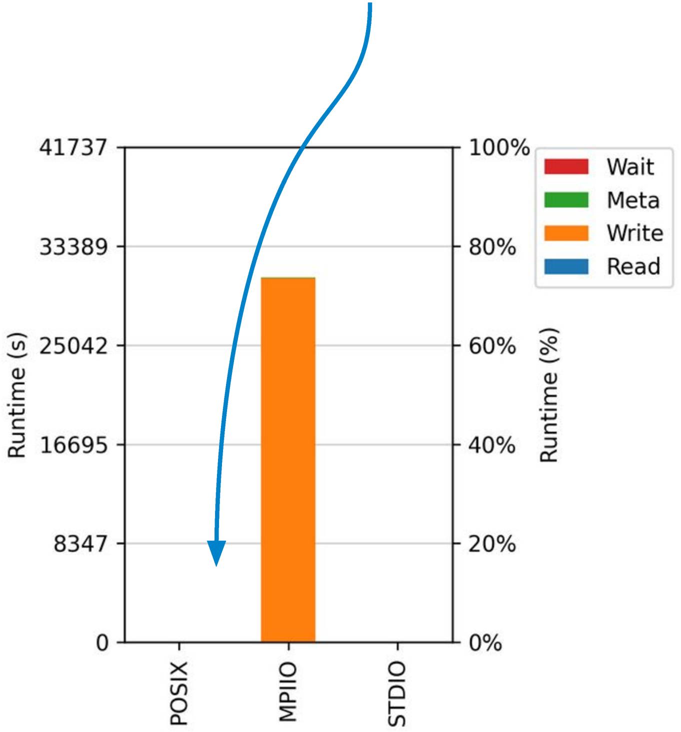
Case study: inefficient MPI-IO

Identified by ranking all MPI-IO based jobs in the dataset based on observed I/O performance.

The job in this case study was among the top 10 lowest performers and was selected for analysis because it was larger and ran for longer than other poor performers:

- 512 total processes (152 compute nodes)
- Over 11.5 hours total runtime
- 798 GiB of total write volume to 122 different shared files
- Sustained bandwidth of **~27 MiB/s**
  - 3 orders of magnitude slower than highest performers

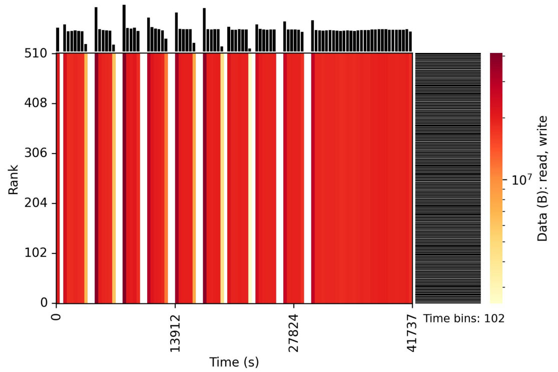
Unusual gap between MPI-IO and POSIX I/O time



Average cost of different I/O APIs and operations

Case study: inefficient MPI-IO

MPI-IO



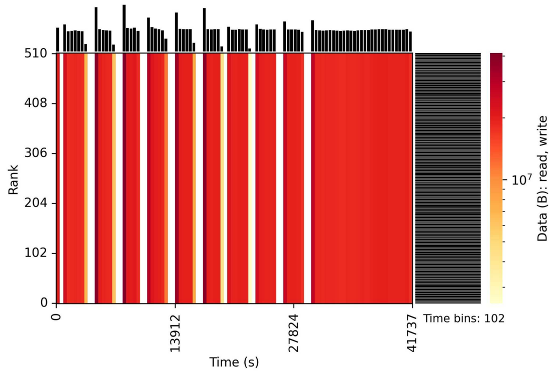
Darshan I/O heatmaps provide overview of I/O intensity over ranks, time, and interfaces.

In this case, application runtime is dominated by numerous, intense I/O phases.

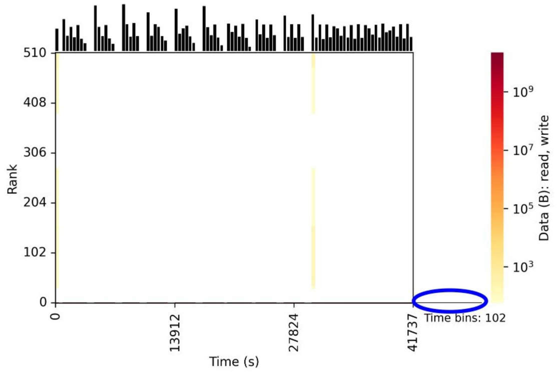
I/O is evenly distributed across ranks, as indicated by the marginal bars on the right y-axis. This looks like a reasonable I/O pattern.

Case study: inefficient MPI-IO

MPI-IO



POSIX

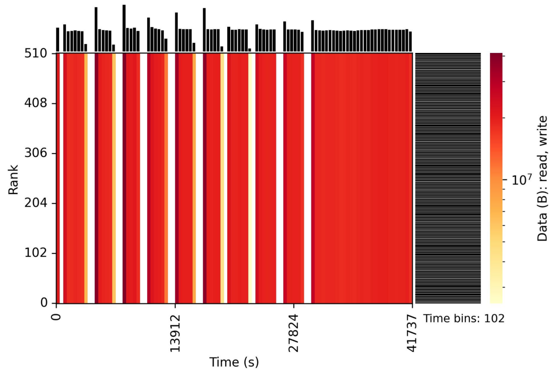


If we compare the MPI-IO heatmap to the POSIX heatmap, a new story emerges:

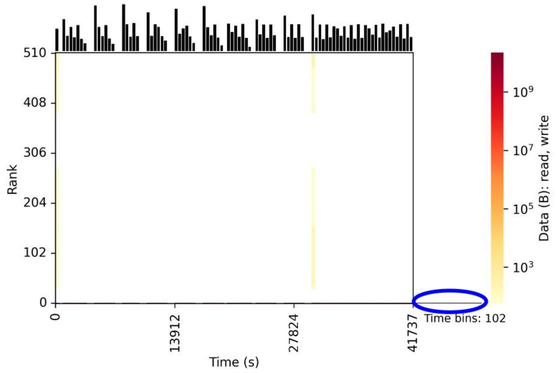
Rank 0 is solely responsible for essentially all of the write volume, creating a serious I/O bottleneck.

Case study: inefficient ~~MPI-IO~~ Lustre usage

MPI-IO



POSIX



This workload is likely the result of poor Lustre stripe settings for this kind of workload[1].

Cray MPICH's ROMIO implementation matches the number of "aggregators" (i.e., processes performing POSIX I/O to the storage system) to the number of OSTs the file is striped over.

The default stripe count on Polaris's Lustre file system is **1**, which can lead to this behavior if it is not changed.

1. Darshan has Lustre instrumentation that could verify this, but was not enabled on Polaris at the time these logs were captured.

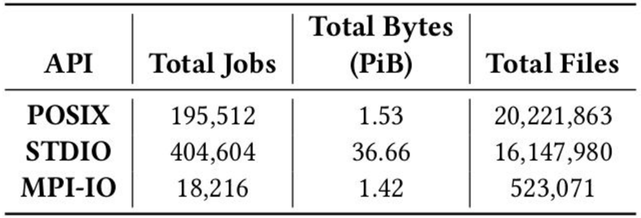
Case study: inefficient redundant I/O

This case study was inspired by the staggering amount of STDIO activity observed across the entire Polaris dataset.

- ❖ 36.6 PiB of STDIO activity, compared to 1.53 PiB of POSIX activity!
- ❖ This was incredibly surprising, but closer investigation revealed that it was heavily skewed by a small subset of jobs – *99% of STDIO activity was produced by 1% of jobs*!

For this case study, we analyze an extreme example job:

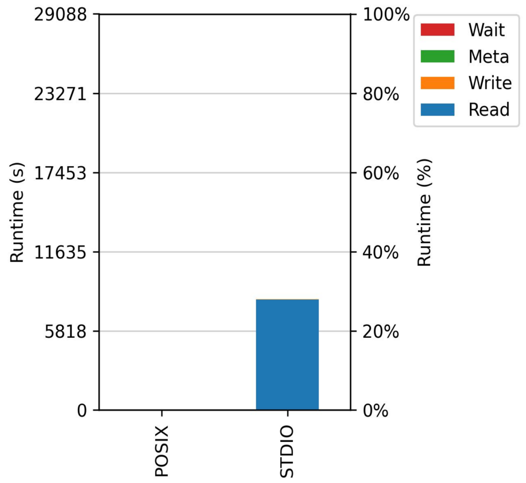
- ❖ Over 529 TiB of STDIO read activity, across just 8 application processes.



API	Total Jobs	Total Bytes (PiB)	Total Files
POSIX	195,512	1.53	20,221,863
STDIO	404,604	36.66	16,147,980
MPI-IO	18,216	1.42	523,071

Aggregate characteristics of Polaris jobs according to different I/O interfaces

Case study: inefficient redundant I/O



Average cost of different I/O APIs and operations

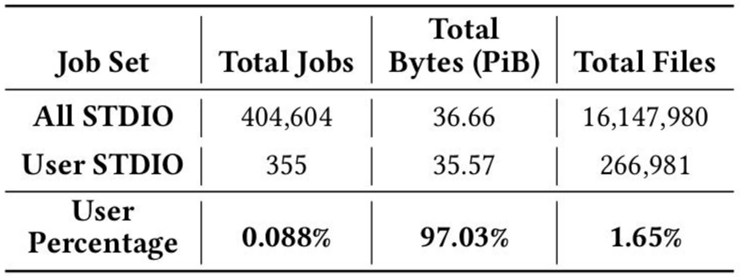
Nearly 30% of the 8-hour runtime spent on STDIO read operations.

Vast majority of read activity due to each process reading over 66 TiB of data from per-process files stored on a Lustre file system.

Detailed inspection of Darshan counters led to observation that each process first wrote ~1 GiB of data to these files before re-reading this data over 60,000 times each!

This is a good candidate for caching.

Case study: inefficient redundant I/O



Job Set	Total Jobs	Total Bytes (PiB)	Total Files
All STDIO	404,604	36.66	16,147,980
User STDIO	355	35.57	266,981
User Percentage	**0.088%**	**97.03%**	**1.65%**

Comparison of the STDIO usage from the user in this case study to aggregate data across our dataset

Most notable about this case study is the disproportionate impact jobs from this user have on aggregate systemwide usage of STDIO across the dataset.

Despite accounting for a fraction of a percent of overall jobs, this user accounted for over 97% of data accessed using STDIO interfaces.

Automating detection of inefficient applications

These case studies highlight the potential impact automated detection of inefficient application I/O behavior could have at HPC facilities.

❖ Common I/O pitfalls (striping misconfiguration, redundant I/O, etc.) can be readily deduced from Darshan’s counters and encoded into detectors.
❖ Automated workflows regularly run these detectors on collected Darshan logs to identify applications/users/projects that could benefit from intervention.

Working with facilities to develop automated detection workflows could dramatically improve application runtimes and overall system efficiency.

The evidence of users/projects with abnormally low performance or high system utilization suggests that targeted outreach could have outsized impact in these efforts.

Closing remarks

Log collection: Analysis scripts:

Conclusions

- We were able to combine existing tools to provide a new service to the community: daily releases of new production logs for community analysis.
- This doesn’t replace the need for other types of curated repositories, but it does address a longstanding gap and offer new possibilities for CS research and facility collaborations.
- Our initial analysis suggests that coverage could be improved by investigating high-value projects.
- Our initial analysis also indicates that we need active methods to identify suboptimal performance, particularly for workload outliers. User education is fantastic, but not sufficient on its own.

Future work

- Work with JGU Mainz to contribute example log files to their trace repository. Their effort is complementary to this one. Our contributions will be more valuable if we develop methods to at least tag application types.
- Advocate for broader adoption of this continuous log publication approach.
  - Not just for Darshan logs at other facilities, but for other types of system logs. Ideally we can share anonymization methods for consistency.
- Translate analysis examples into active capabilities for the facility.
  - Imagine a monthly report of projects in terms of I/O usage and efficiency, or even simply investigating how to expand instrumentation coverage.
- Correlate Darshan logs with other types of storage telemetry.
- Work to systematically improve Darshan coverage.

Questions?

This research used resources of the Argonne Leadership Computing Facility, a U.S. Department of Energy (DOE) Office of Science user facility at Argonne National Laboratory and is based on research supported by the U.S. DOE Office of Science-Advanced Scientific Computing Research Program, under Contract No. DE-AC02-06CH11357.

Data used in this work was generated from resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

Backup slides

Darshan coverage of Polaris jobs

Coverage is essentially a measure of the fraction of system job activity that Darshan is able to instrument.

By correlating the Darshan log collection with ALCF PBS job scheduler datasets[1] we can better quantify the extent of Darshan "coverage" of Polaris jobs:

$$coverage = \frac{\Sigma\, node\_hours_{Darshan}}{\Sigma\, node\_hours_{PBS}}$$

1. https://reports.alcf.anl.gov/data/polaris.html

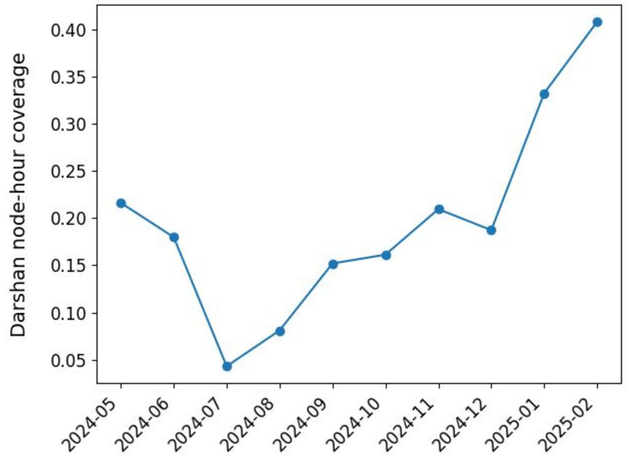
Darshan coverage of Polaris jobs

Coverage is essentially a measure of the fraction of system job activity that Darshan is able to instrument.

By correlating the Darshan log collection with ALCF PBS job scheduler datasets[1] we can better quantify the extent of Darshan "coverage" of Polaris jobs:

$$coverage = \frac{\Sigma\, node\_hours_{Darshan}}{\Sigma\, node\_hours_{PBS}}$$

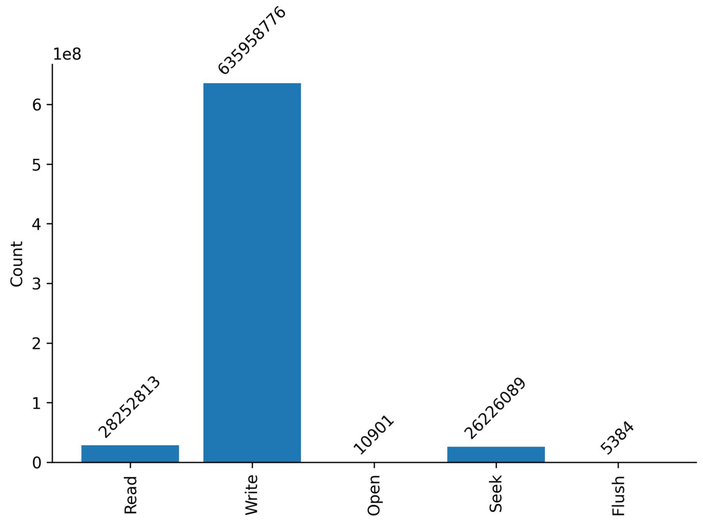
Coverage can vary wildly from month-to-month, but Darshan achieves a monthly average of around 20%.



Darshan coverage of Polaris jobs over the 10-month period analyzed in this work

1. https://reports.alcf.anl.gov/data/polaris.html

Case study: inefficient redundant I/O



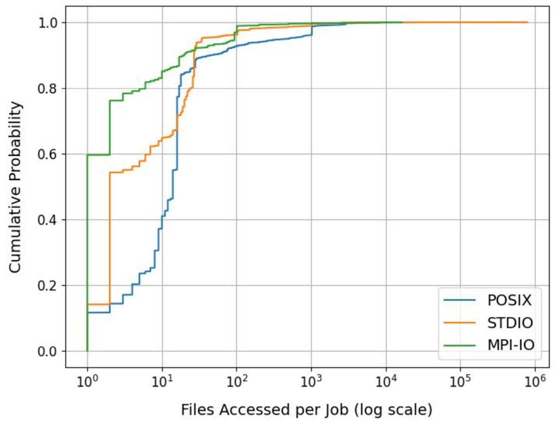
STDIO I/O operation counts

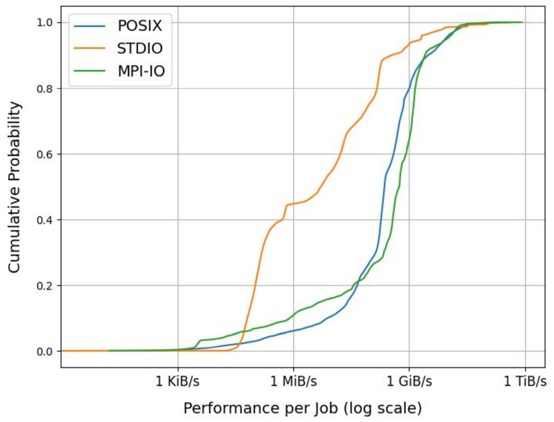
Interestingly, despite the application being dominated by reads (in both time and volume), many more write operations (635 million+) were issued than read operations (28 million+).

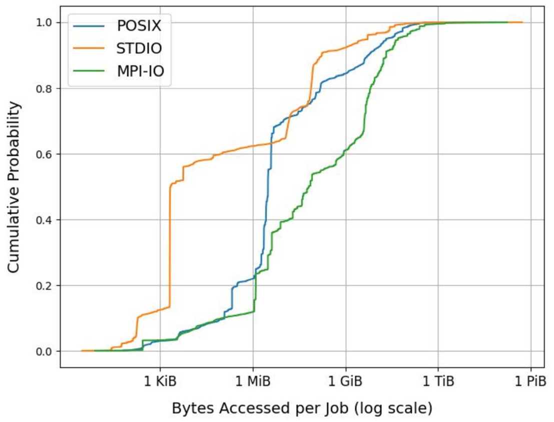
21.5 GiB of writes => *average access size of just 10 bytes*

This is typically a poor access pattern to Lustre, but a reasonable bandwidth of 150 MiB/s achieved likely due to aggressive write buffering in the STDIO library.

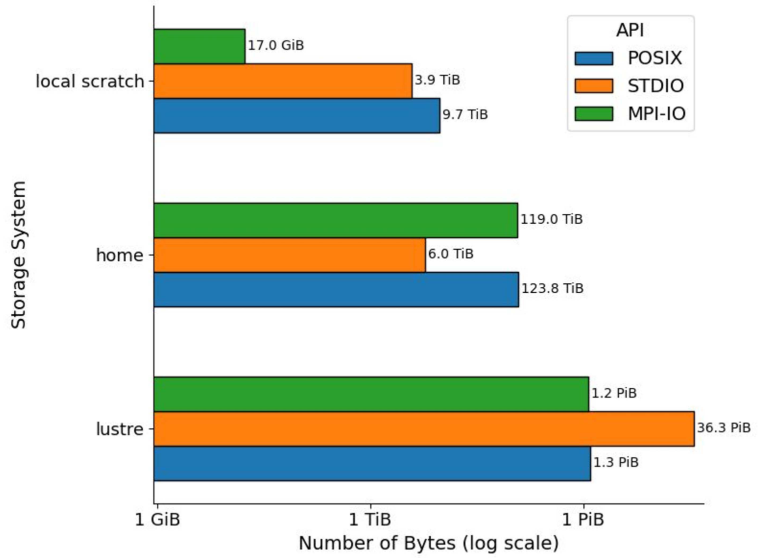
Backup - CDFs of I/O metrics across all jobs

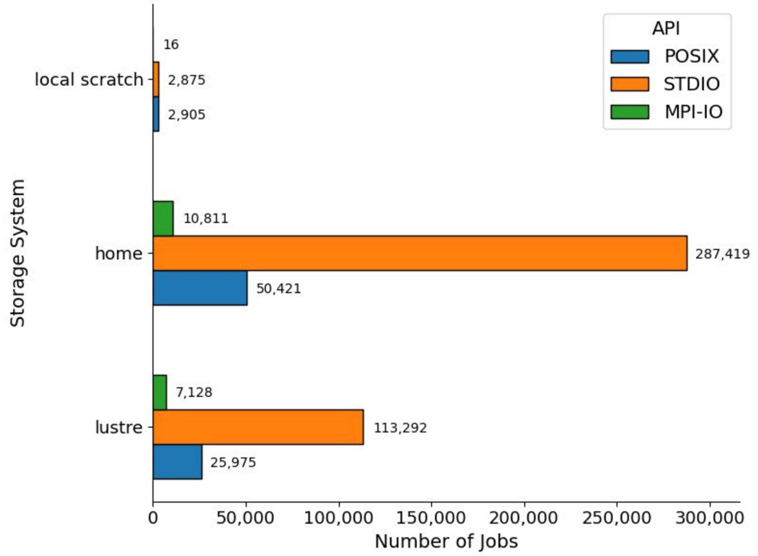






Backup - job usage of various interfaces/storage


API
POSIX
STDIO
MPI-IO
local scratch
home
lustre
Storage System
17.0 GiB
3.9 TiB
9.7 TiB
119.0 TiB
6.0 TiB
123.8 TiB
1.2 PiB
36.3 PiB
1.3 PiB
1 GiB
1 TiB
1 PiB
Number of Bytes (log scale)


API
POSIX
STDIO
MPI-IO
local scratch
home
lustre
Storage System
Number of Jobs