Hands-on Breakout: Darshan

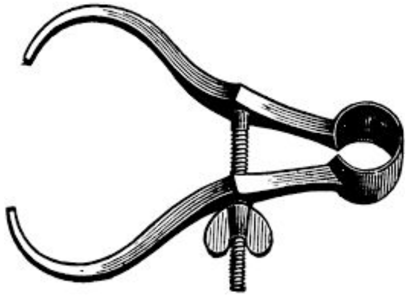
Shane Snyder
Argonne National Laboratory

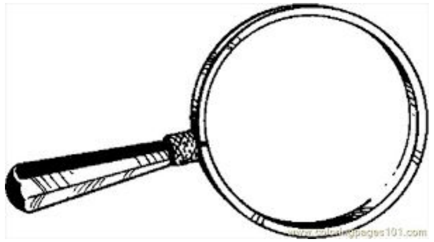
ALCF Hands-on HPC Workshop, Day 3
October 12, 2023

Understanding I/O problems in your application

Example questions:

- ❑ How much of your run time is spent reading and writing files?
- ❑ Does it get better, worse, or is it the same as you scale up?
- ❑ Does it get better, worse, or is it the same across platforms?
- ❑ How should you prioritize I/O tuning to get the most bang for your buck?





We recommend using a tool called **Darshan** as a starting point.

In this hands-on session, we'll cover:

1. Darshan background
2. Darshan usage on HPC systems (e.g., ALCF Polaris)
3. Darshan analysis tool insights
4. General HPC I/O best practices and tuning considerations

What is Darshan?

Darshan is a scalable HPC I/O characterization tool. It captures a concise picture of application I/O behavior with minimal overhead.

★ Widely available
- Deployed at most large supercomputing sites
- Including most systems at ALCF, OLCF, and NERSC

★ Easy to use
- No changes to code or development process
- Negligible performance impact: just “leave it on”

★ Produces a *summary* of I/O activity for every job
- This is a great starting point for understanding your application’s data usage
- Includes counters, timers, histograms, etc.

How does Darshan work?

Two primary components:

1. **Darshan runtime library**

   ○ Instrumentation modules: lightweight wrappers (interposed at link or run time) intercept application I/O calls and record statistics about file accesses
     – File records are stored in bounded, compact memory on each process

   ○ Core library: aggregate statistics when the application exits and generate a log file
     – Collect, filter, compress records and write a single summary file for the job

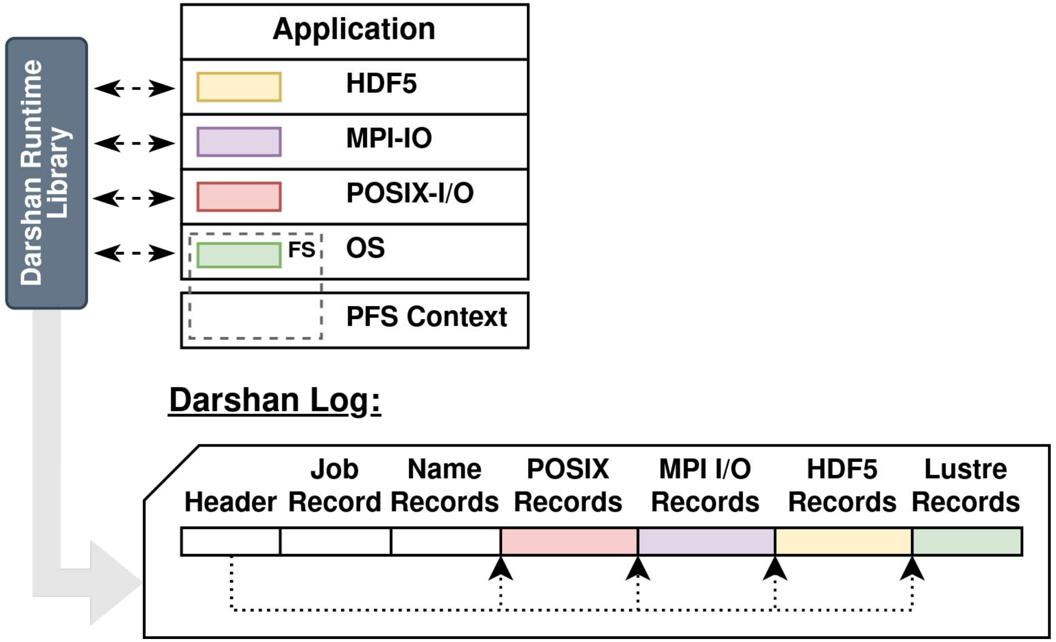


Figure courtesy Jakob Luettgau (UTK)

How does Darshan work?

Two primary components:

1. **Darshan runtime library**

NOTE: Though traditionally restricted to MPI apps, recent Darshan versions can often be made to work in non-MPI contexts. **More on this later…**

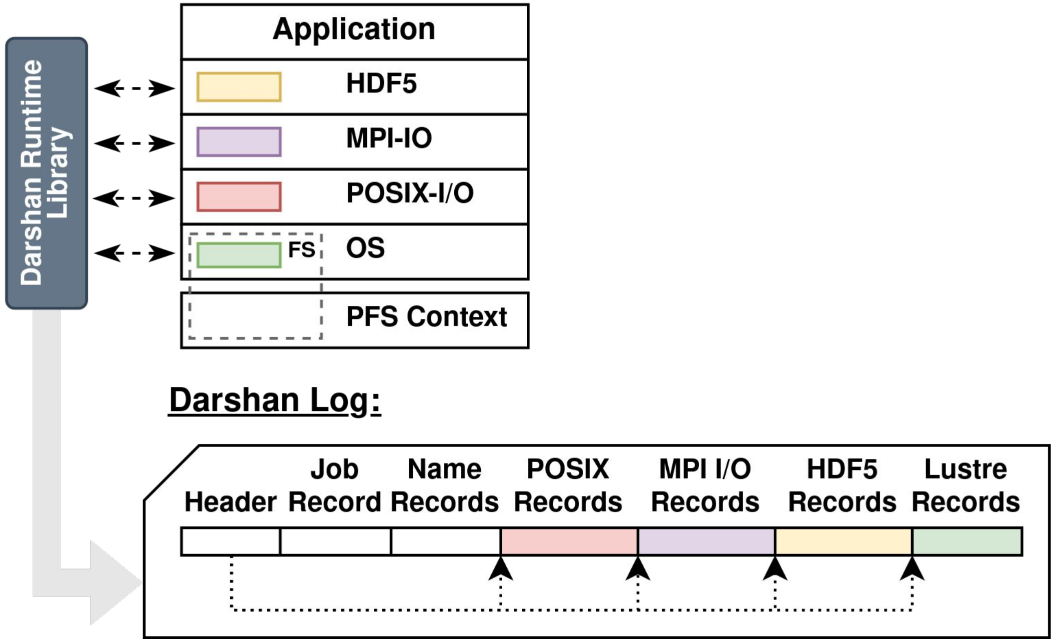


Figure courtesy Jakob Luettgau (UTK)

How does Darshan work?

Two primary components:

2. Darshan log analysis tools

- Tools and interfaces to inspect and interpret log data
  - PyDarshan command line utilities like the new job summary tool
  - Python APIs for usage in custom tools, Jupyter notebooks, etc.
  - Legacy C-based tools/library

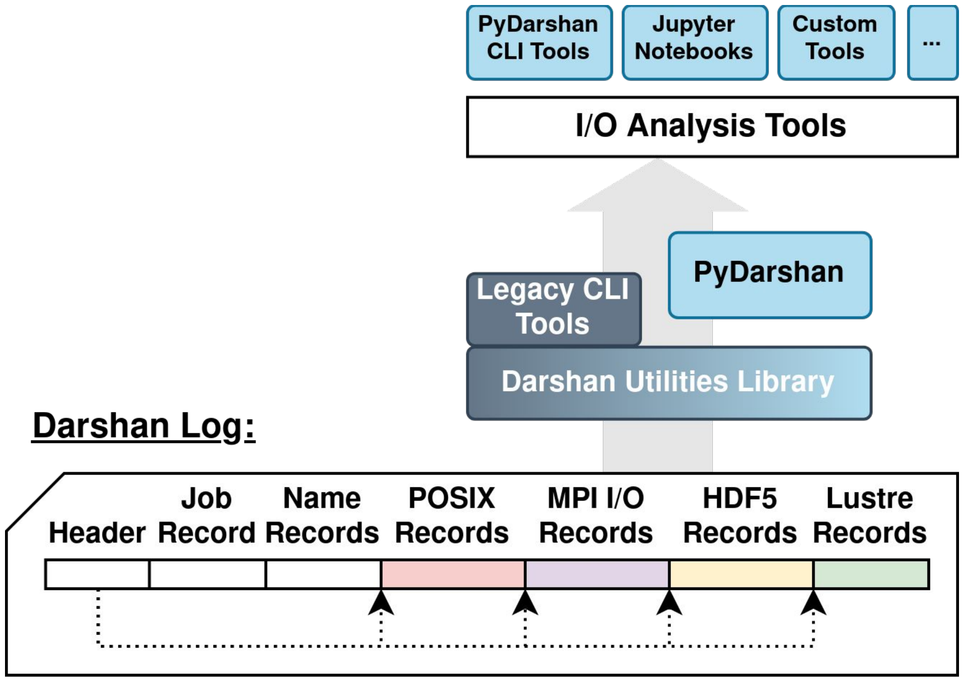


Figure courtesy Jakob Luettgau (UTK)

Using Darshan

Using Darshan

- We'll consider ALCF Polaris as an example in the following slides.
- The workshop repo includes Darshan hands on examples that are configured for use on Polaris.
  - https://github.com/argonne-lcf/ALCF_Hands_on_HPC_Workshop
  - See the **darshan-hands-on** directory
- Darshan deployments on other HPC systems are very similar. The most likely differences are:
  - Location of log files (where to find data after your job completes)
  - Analysis utility availability (usually easiest to just copy logs to your workstation to analyze)
  - Loading the Darshan module (if it's not already there by default)
- We'll briefly cover differences on other notable DOE systems after the Polaris example.

Using Darshan on Polaris: load the software

```
snyder@polaris-login-01> cd ALCF_Hands_on_HPC_Workshop/darshan-hands-on/
snyder@polaris-login-01>
snyder@polaris-login-01> source polaris-setup-env.sh
snyder@polaris-login-01>
```

The **darshan-hands-on** directory in the workshop GitHub repo includes a script to configure your environment with the tools needed for Darshan analysis.

NOTE: This additional setup script is manually loading the Darshan module, which is not yet enabled by default on Polaris – we are working on making this automatic!

Using Darshan on Polaris: load the software

```
snyder@polaris-login-01> cd ALCF_Hands_on_HPC_Workshop/darshan-hands-on/
snyder@polaris-login-01>
snyder@polaris-login-01> source polaris-setup-env.sh
snyder@polaris-login-01>
snyder@polaris-login-01> module list

Currently Loaded Modules:
  1) craype-x86-rome             9) cray-pmi/6.1.2
  2) libfabric/1.11.0.4.125     10) cray-pmi-lib/6.0.17
  3) craype-network-ofi         11) cray-pals/1.1.7
  4) perftools-base/22.05.0     12) cray-libpals/1.1.7
  5) nvhpc/21.9                 13) PrgEnv-nvhpc/8.3.3
  6) craype/2.7.15              14) craype-accel-nvidia80
  7) cray-dsmml/0.2.2           15) darshan/3.4.3
  8) cray-mpich/8.1.16          16) cray-python/3.9.12.1
```

The **darshan-hands-on** directory in the workshop GitHub repo includes a script to configure your environment with the tools needed for Darshan analysis.

Use "`module list`" to see a list of software loaded in your environment.

Darshan 3.4.3 should now be loaded.

Using Darshan on Polaris: load the software

```
snyder@polaris-login-01> cd ALCF_Hands_on_HPC_Workshop/darshan-hands-on/
snyder@polaris-login-01>
snyder@polaris-login-01> source polaris-setup-env.sh
snyder@polaris-login-01>
snyder@polaris-login-01> module list

Currently Loaded Modules:
  1) craype-x86-rome           9) cray-pmi/6.1.2
  2) libfabric/1.11.0.4.125   10) cray-pmi-lib/6.0.17
  3) craype-network-ofi       11) cray-pals/1.1.7
  4) perftools-base/22.05.0   12) cray-libpals/1.1.7
  5) nvhpc/21.9               13) PrgEnv-nvhpc/8.3.3
  6) craype/2.7.15            14) craype-accel-nvidia80
  7) cray-dsmml/0.2.2         15) darshan/3.4.3
  8) cray-mpich/8.1.16        16) cray-python/3.9.12.1
```

These steps are similar, and often cases easier, on other HPC platforms where Darshan is deployed:

- Theta/Summit: Darshan module loaded by default
- Perlmutter: Darshan can be manually loaded with ‘**`module load darshan`**’

Always check facility documentation!

Using Darshan on Polaris: instrument your code

Compile and run your application!

```
cc -o helloworld helloworld.c
```

```
qsub helloworld.qsub
```

That's all there is to it; Darshan does the rest.*

From the **helloworld** example in the **darshan-hands-on** directory in the workshop GitHub repo

* Well, almost. There is one caveat: in the default Darshan configuration, your application must call MPI_Init() and MPI_Finalize() to generate a log.

Argonne NATIONAL LABORATORY

Using Darshan on Polaris: find your log file

All Darshan logs are placed in a central location. The '**`darshan-config --log-path`**' command will provide the log directory location.

```
snyder@polaris-login-04:~> darshan-config --log-path
/lus/grand/logs/darshan/polaris
snyder@polaris-login-04:~>
snyder@polaris-login-04:~> cd /lus/grand/logs/darshan/polaris/2023/10/10/
```

Go to subdirectory for the year / month / day your job executed.

Be aware of time zone (or just check adjacent days)!
Polaris, for example, uses the GMT time zone and will roll over to the next day at 7pm local time.

Argonne NATIONAL LABORATORY

Using Darshan on Polaris: find your log file

All Darshan logs are placed in a central location. The '**`darshan-config --log-path`**' command will provide the log directory location.

```
snyder@polaris-login-04:~> darshan-config --log-path
/lus/grand/logs/darshan/polaris
snyder@polaris-login-04:~>
snyder@polaris-login-04:~> cd /lus/grand/logs/darshan/polaris/2023/10/10/
snyder@polaris-login-04:/lus/grand/logs/darshan/polaris/2023/10/10> ls | grep snyder
snyder_helloworld_id1126258-60233_10-10-72928-17597401521854260646_1.darshan
```

File name includes your username, app name, and job ID.

For convenience, users often copy logs somewhere else to save/analyze.

Argonne NATIONAL LABORATORY

Using Darshan on Polaris: analyze log

After locating your log, users can utilize Darshan log analysis tools for gaining insights into application I/O behavior. PyDarshan tools likely aren't available everywhere, but traditional tools like `darshan-parser` should be.

```
shane@shane-x1-carbon: darshan-parser ./log.darshan | grep POSIX_BYTES_WRITTEN | sort -nr -k 5
POSIX   387     6966057185861764086     POSIX_BYTES_WRITTEN     5413869452      /projects/radix
POSIX   452     6966057185861764086     POSIX_BYTES_WRITTEN     5413865644      /projects/radix
POSIX   197     6966057185861764086     POSIX_BYTES_WRITTEN     5413857652      /projects/radix
POSIX   5       6966057185861764086     POSIX_BYTES_WRITTEN     5413852168      /projects/radix
POSIX   451     6966057185861764086     POSIX_BYTES_WRITTEN     5413844532      /projects/radix
POSIX   64      6966057185861764086     POSIX_BYTES_WRITTEN     5413823236      /projects/radix
POSIX   68      6966057185861764086     POSIX_BYTES_WRITTEN     5413788992      /projects/radix
POSIX   195     6966057185861764086     POSIX_BYTES_WRITTEN     5413663132      /projects/radix
POSIX   323     6966057185861764086     POSIX_BYTES_WRITTEN     5413658668      /projects/radix
POSIX   132     6966057185861764086     POSIX_BYTES_WRITTEN     5413648628      /projects/radix
```

If you know what you're looking for, `darshan-parser` can be a quick way to extract important I/O details from a log, e.g., the 10 most heavily written files, but it is not super user friendly…

Using Darshan on Polaris: generate summary report

The Polaris environment setup script in the **darshan-hands-on** directory in the workshop GitHub repo also enables support for PyDarshan analysis tools.

Generate an HTML summary report with PyDarshan using the following command: ‘**python -m darshan summary <log_path>**’.

```
snyder@polaris-login-04> source ~/ALCF_Hands_on_HPC_Workshop/darshan-hands-on/polaris-setup-env.sh
snyder@polaris-login-04>
snyder@polaris-login-04> python -m darshan summary helloworld.darshan
/opt/cray/pe/python/3.9.12.1/lib/python3.9/site-packages/scipy/__init__.py:138: UserWarning: A NumPy version >=1.16.5 a
nd <1.23.0 is required for this version of SciPy (detected version 1.25.1)
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion} is required for this version of "
Report generated successfully.
Saving report at location: /home/snyder/ALCF_Hands_on_HPC_Workshop/darshan-hands-on/helloworld/helloworld_report.html
```

Using Darshan on Polaris: generate summary report

The Polaris environment setup script in the **darshan-hands-on** directory in the workshop GitHub repo also enables support for PyDarshan analysis tools.

Generate an HTML summary report with PyDarshan using the following command: '**python -m darshan summary <log_path>**'.

```
snyder@polaris-login-04> source ~/ALCF_Hands_on_HPC_Workshop/darshan-hands-on/polaris-setup-env.sh
snyder@polaris-login-04>
snyder@polaris-login-04> python -m darshan summary helloworld.darshan
/opt/cray/pe/python/3.9.12.1/lib/python3.9/site-packages/scipy/__init__.py:138: UserWarning: A NumPy version >=1.16.5 a
nd <1.23.0 is required for this version of SciPy (detected version 1.25.1)
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion} is required for this version of "
Report generated successfully.
Saving report at location: /home/snyder/ALCF_Hands_on_HPC_Workshop/darshan-hands-on/helloworld/helloworld_report.html
```

If successful, the tool should generate an HTML report matching the input log file name.

To analyze, it's likely easiest to copy the report to your own workstation to view in a browser.

Argonne NATIONAL LABORATORY

Using Darshan on Polaris: generate summary report

The Polaris environment setup script in the **darshan-hands-on** directory in the workshop GitHub repo also enables support for PyDarshan analysis tools.

Generate an HTML summary report with PyDarshan using the following command: ‘**python -m darshan summary <log_path>**’.

```
snyder@polaris-login-04> source ~/ALCF_Hands_on_HPC_Workshop/darshan-hands-on/polaris-setup-env.sh
snyder@polaris-login-04>
snyder@polaris-login-04> python -m darshan summary helloworld.darshan
/opt/cray/pe/python/3.9.12.1/lib/python3.9/site-packages/scipy/__init__.py:138: UserWarning: A NumPy version >=1.16.5 a
nd <1.23.0 is required for this version of SciPy (detected version 1.25.1)
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion} is required for this version of "
Report generated successfully.
Saving report at location: /home/snyder/ALCF_Hands_on_HPC_Workshop/darshan-hands-on/helloworld/helloworld_report.html
```

NOTE: Ignore these Python warnings about version requirements, they should not cause any issues with report generation

What about other HPC systems?

- **Perlmutter** (NERSC):
  - How to enable: ‘`module load darshan`’
  - Log directory: /pscratch/darshanlogs/
- **Summit** (OLCF):
  - How to enable: automatic
  - Log directory: /gpfs/alpine/darshan/summit

If Darshan is not available on a system, it can be installed via Spack or directly from source. Darshan is provided as 2 separate packages in Spack:

- **darshan-runtime** - library for instrumenting apps
- **darshan-util** - tools for analyzing Darshan log files

PyDarshan is available on PyPI (e.g., ‘`pip install darshan`’) and also in Spack

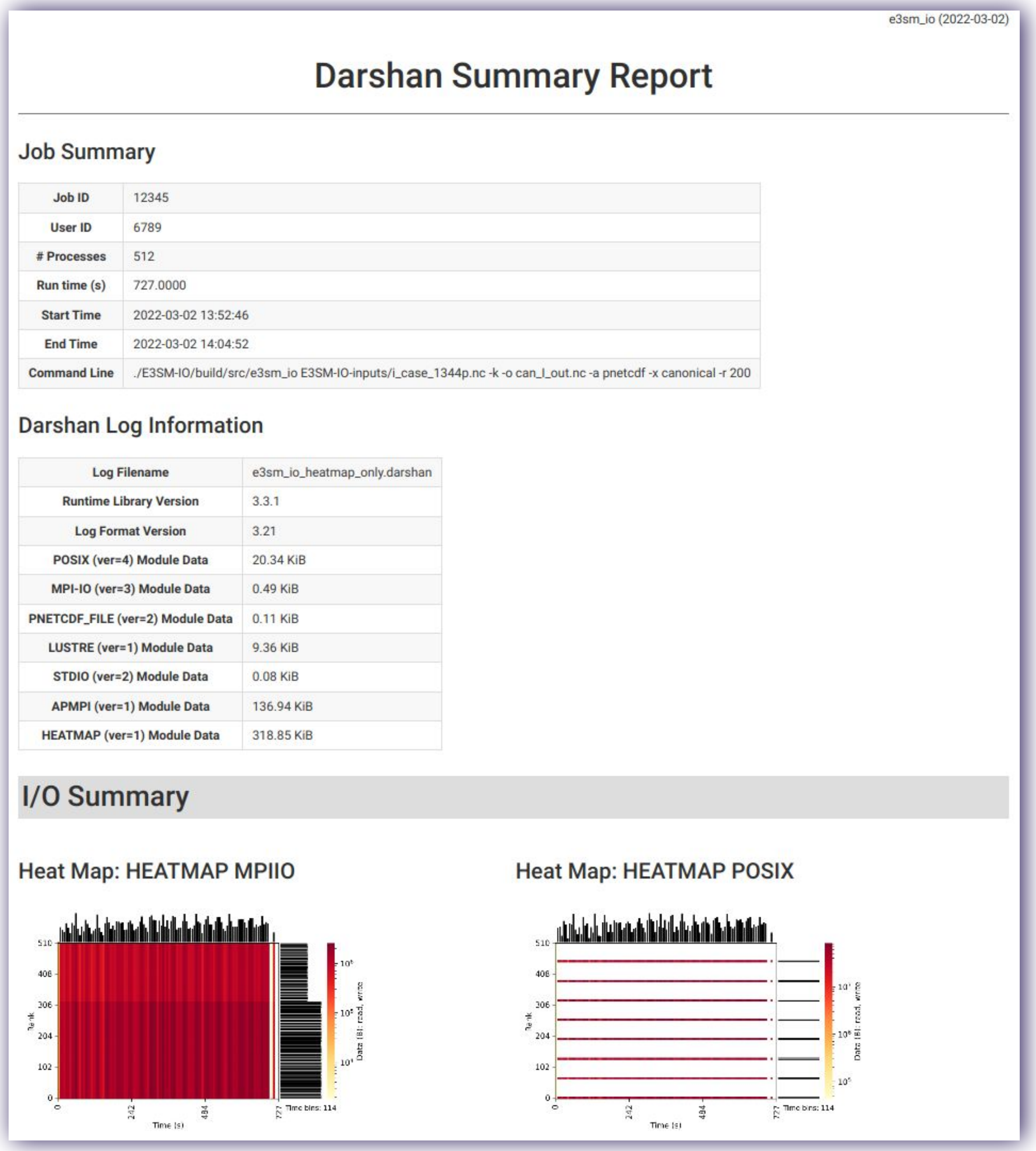
See our website for more details:
https://www.mcs.anl.gov/research/projects/darshan

Analyzing Darshan logs

Job analysis example

The PyDarshan job summary tool generates an HTML report containing graphs, tables, and performance estimates characterizing the I/O workload of the application

We will summarize some of the highlights in the following slides


e3sm_io (2022-03-02)

Darshan Summary Report

Job Summary

Job ID	12345
User ID	6789
# Processes	512
Run time (s)	727.0000
Start Time	2022-03-02 13:52:46
End Time	2022-03-02 14:04:52
Command Line	./E3SM-IO/build/src/e3sm_io E3SM-IO-inputs/i_case_1344p.nc -k -o can_I_out.nc -a pnetcdf -x canonical -r 200

Darshan Log Information

Log Filename	e3sm_io_heatmap_only.darshan
Runtime Library Version	3.3.1
Log Format Version	3.21
POSIX (ver=4) Module Data	20.34 KiB
MPI-IO (ver=3) Module Data	0.49 KiB
PNETCDF_FILE (ver=2) Module Data	0.11 KiB
LUSTRE (ver=1) Module Data	9.36 KiB
STDIO (ver=2) Module Data	0.08 KiB
APMPI (ver=1) Module Data	136.94 KiB
HEATMAP (ver=1) Module Data	318.85 KiB

I/O Summary

Heat Map: HEATMAP MPIIO

Heat Map: HEATMAP POSIX

Argonne NATIONAL LABORATORY

Job analysis: high-level job info

e3sm_io (2022-03-02)

Executable name and job date

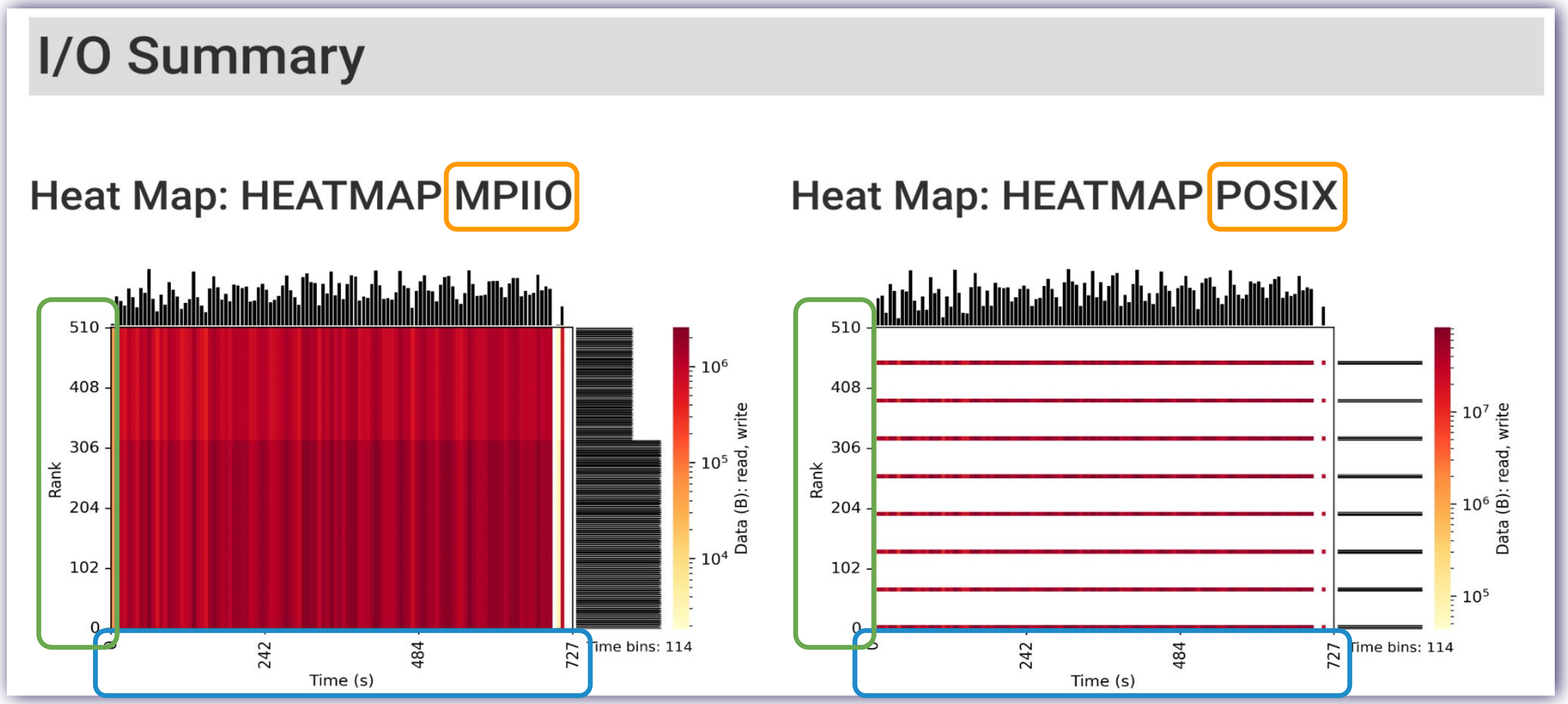
Darshan Summary Report

Job Summary

Detailed job metadata

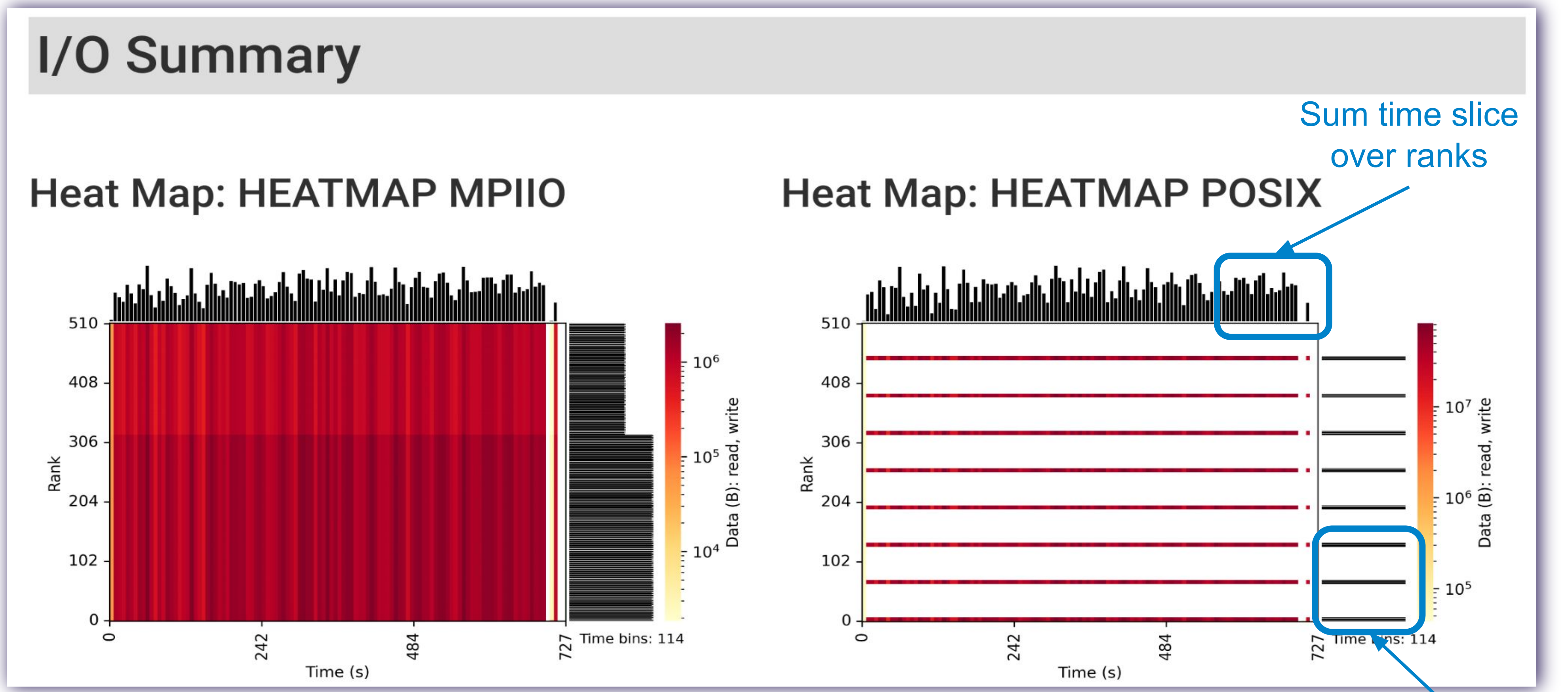
Job ID	12345
User ID	6789
# Processes	512
Run time (s)	727.0000
Start Time	2022-03-02 13:52:46
End Time	2022-03-02 14:04:52
Command Line	./E3SM-IO/build/src/e3sm_io E3SM-IO-inputs/i_case_1344p.nc -k -o can_I_out.nc -a pnetcdf -x canonical -r 200

Job analysis: I/O heatmaps



Heatmaps showcase application I/O intensity (r+w volume) across **time**, **ranks**, and **interfaces** – helpful for identifying hot spots, I/O and compute phases, etc.

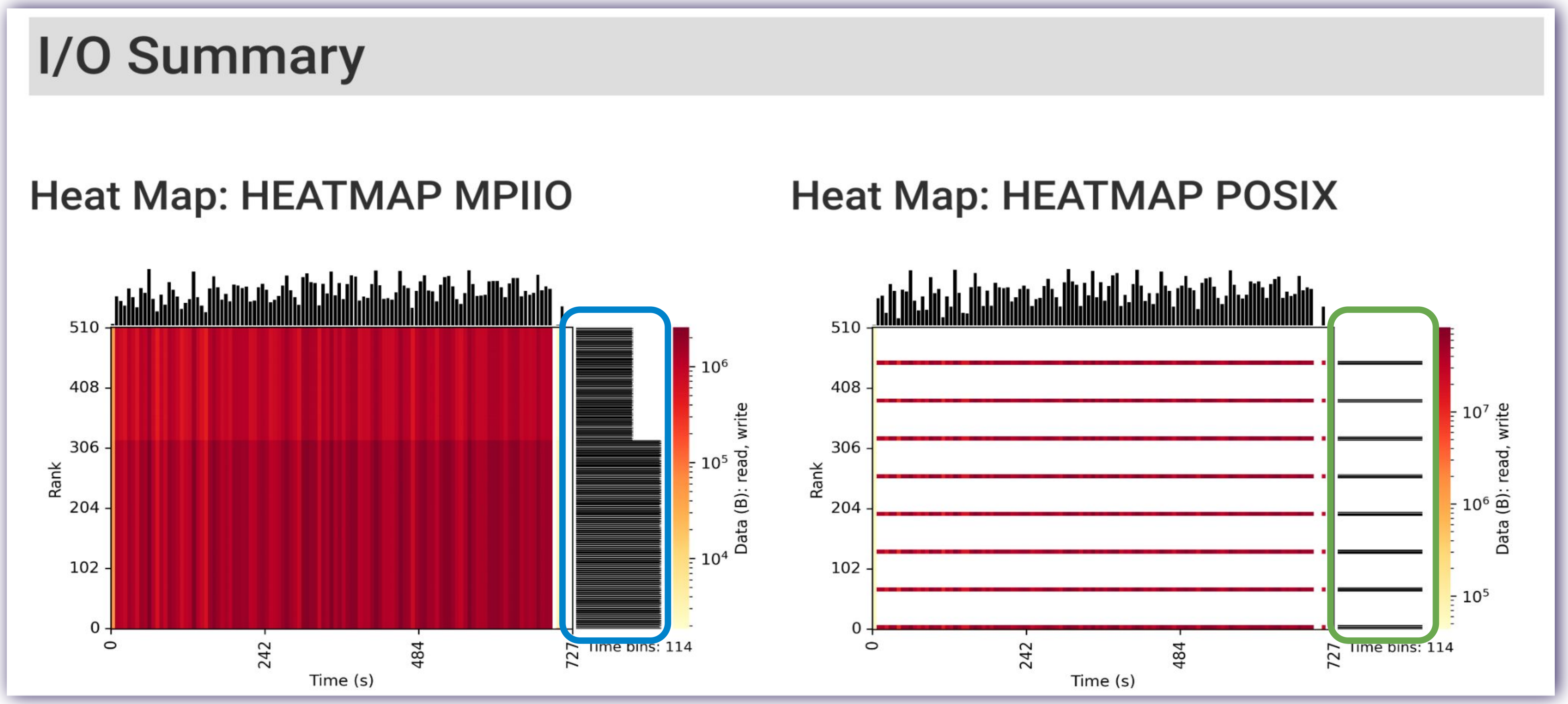
Job analysis: I/O heatmaps



Heatmaps showcase application I/O intensity (r+w volume) across time, ranks, and interfaces – helpful for identifying hot spots, I/O and compute phases, etc.

Argonne NATIONAL LABORATORY

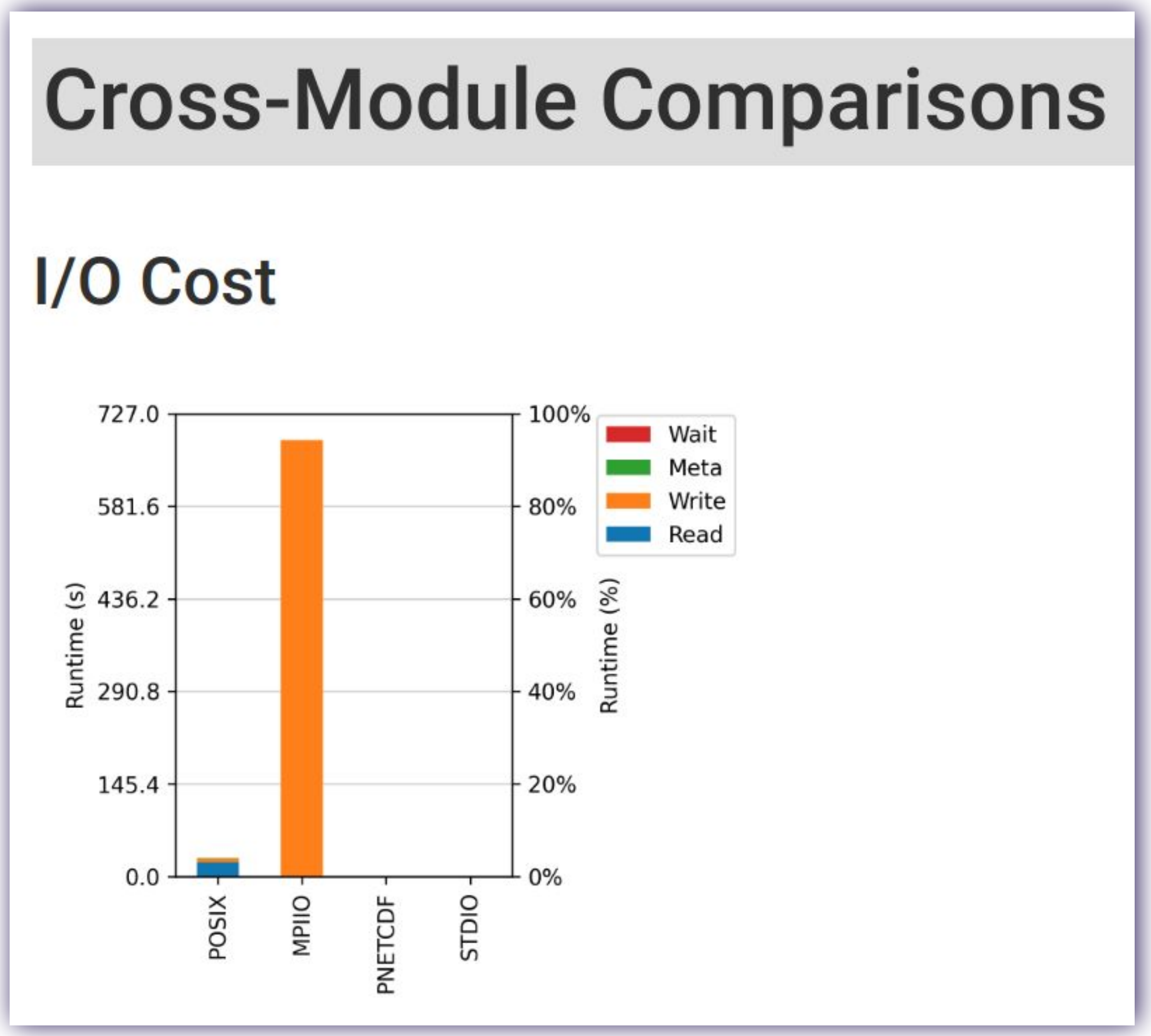
Job analysis: I/O heatmaps



This application demonstrates a couple of notable I/O characteristics:

- **I/O imbalance across MPI processes**
- **Collective MPI-IO accesses transformed to subset of “aggregator” ranks at POSIX level**

Job analysis: I/O cost



I/O cost indicates how much time on average was spent reading, writing, and doing metadata across different I/O interfaces

If I/O cost is a small portion of application runtime, tuning efforts are likely to have a relatively small impact

Job analysis: Per-interface statistics

Per-Module Statistics: POSIX

Overview

files accessed	3
bytes read	24.53 MiB
bytes written	283.74 GiB
I/O performance estimate	1023.84 MiB/s (average)

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

	number of files	avg. size	max size
total files	3	99.74 GiB	297.71 GiB
read-only files	1	11.18 MiB	11.18 MiB
write-only files	2	149.60 GiB	297.71 GiB
read/write files	0	0	0

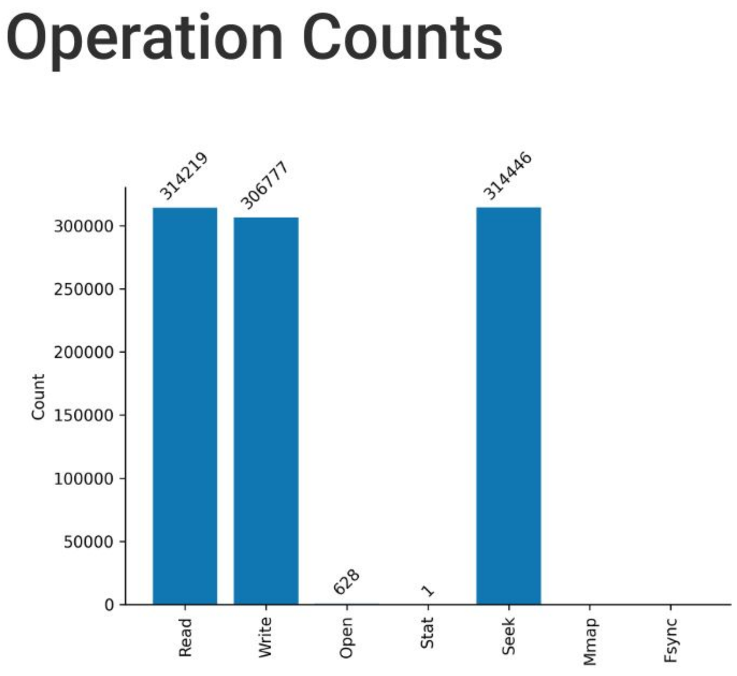
Stats available for various I/O APIs: POSIX, MPI-IO, STDIO, HDF5, PnetCDF

Aggregate stats for interface, as well as a **performance estimate**

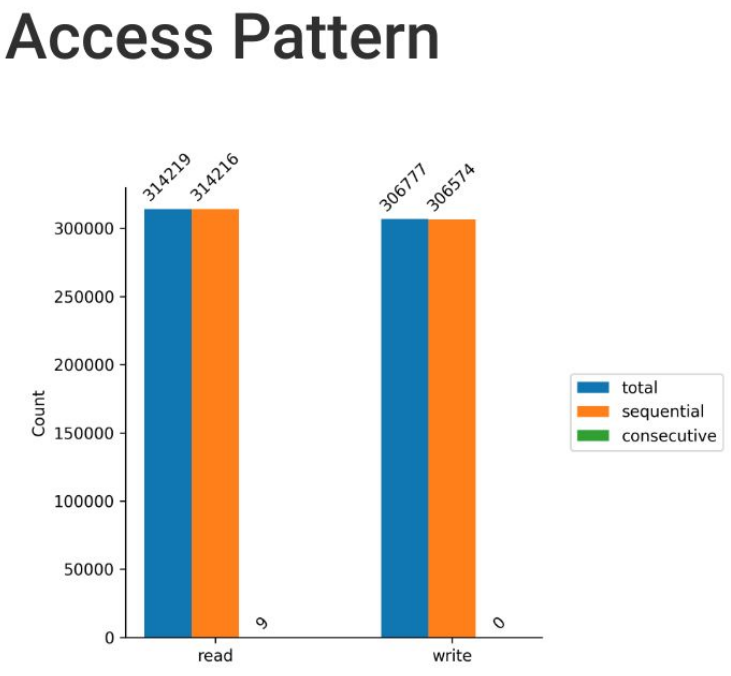
Number of files of different types (total, read-only, read/write, etc.) recorded by Darshan

Argonne NATIONAL LABORATORY

Job analysis: Per-interface statistics



★ sequential: greater than previous offset
consecutive: immediately following previous offset



Operation counts provide the relative totals of different types of I/O operations

Lots of metadata operations (open, stat, seek, etc.) could be a sign of poorly performing I/O

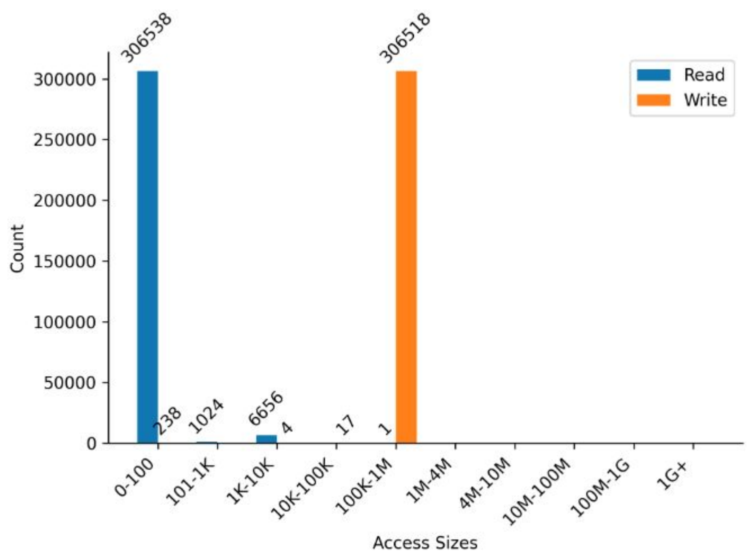
Access pattern indicates whether read/write operations progress sequentially or consecutively★ through the file

More random access patterns can be expensive for some types of storage

Argonne NATIONAL LABORATORY

Job analysis: Per-interface statistics

Access Sizes



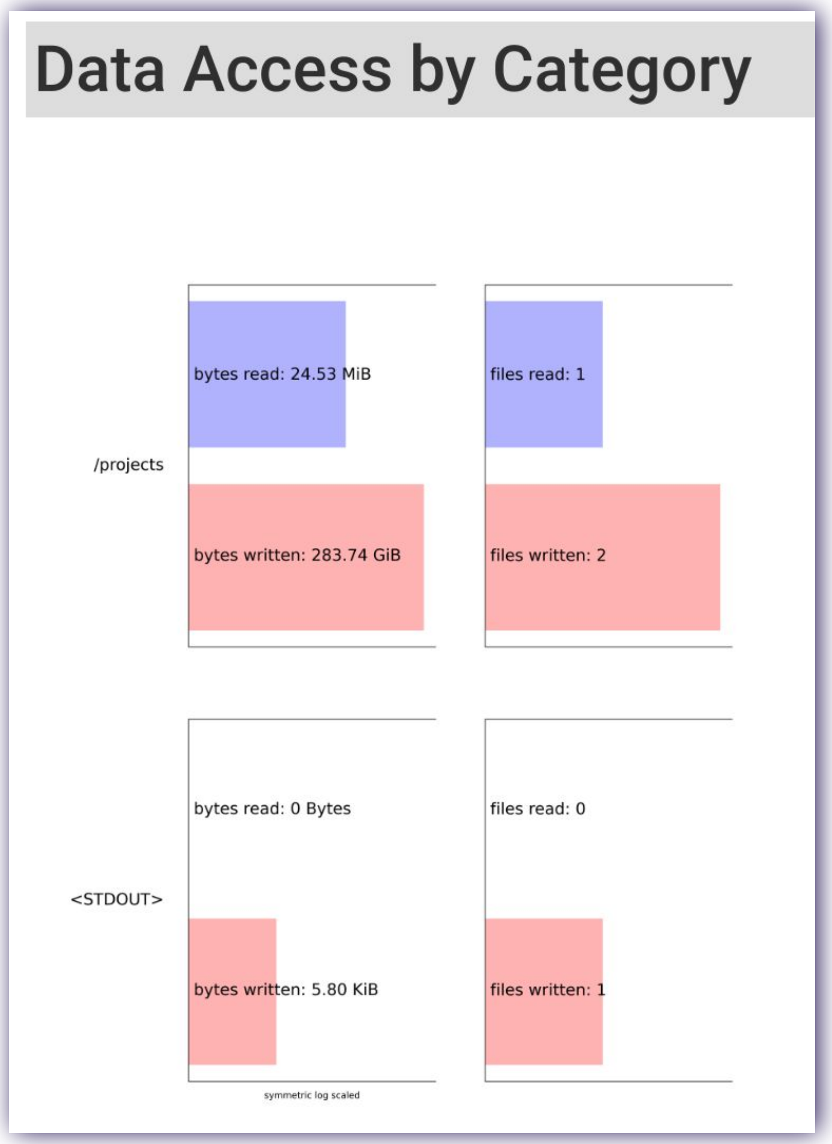
Common Access Sizes

Access Size	Count
800964	66221
1048576	36500
5376	2560
1048568	589

Details on access sizes are provided to better understand granularity of application read/write accesses

In general, larger access sizes (e.g., O(MiBs)) perform better with most storage systems

Job analysis: Data access by category



Data accesses, in terms of total files read/written and total bytes read/written, binned by different categories:

- FS mount points (e.g., `/home`, `/scratch`)
- standard streams (e.g., `STDOUT`)
- object storage pools
- etc.

Inform on job's general usage of different storage resources

Job analysis: additional help

e3sm_io (2022-03-02)

Darshan Summary Report

Job Summary

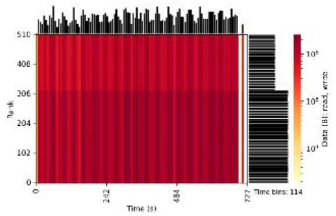
Job ID	12345
User ID	6789
# Processes	512
Run time (s)	727.0000
Start Time	2022-03-02 13:52:46
End Time	2022-03-02 14:04:52
Command Line	./E3SM-IO/build/src/e3sm_io E3SM-IO-inputs/i_case_1344p.nc -k -o can_I_out.nc -a pnetcdf -x canonical -r 200

Darshan Log Information

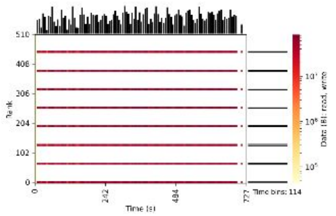
Log Filename	e3sm_io_heatmap_only.darshan
Runtime Library Version	3.3.1
Log Format Version	3.21
POSIX (ver=4) Module Data	20.34 KiB
MPI-IO (ver=3) Module Data	0.49 KiB
PNETCDF_FILE (ver=2) Module Data	0.11 KiB
LUSTRE (ver=1) Module Data	9.36 KiB
STDIO (ver=2) Module Data	0.08 KiB
APMPI (ver=1) Module Data	136.94 KiB
HEATMAP (ver=1) Module Data	318.85 KiB

I/O Summary

Heat Map: HEATMAP MPIIO



Heat Map: HEATMAP POSIX



Remember to contact facility support staff for help! The Darshan job summary can be a good discussion starter if you aren't sure how to proceed with performance tuning or problem solving.

Argonne NATIONAL LABORATORY

Darshan: a quick recap

- These slides have thus far covered some basic Darshan usage and tips.
- Key takeaways:
  - Tools are available to help you understand how your application accesses data.
  - The simplest starting point is Darshan.
  - It’s likely already instrumenting your application, or can quickly be made to do so.
  - You will probably start with an HTML report generated using PyDarshan.
- Refer to documentation and support channels provided by the Darshan team and/or facilities staff.
  - darshan-io.slack.com

Hands-on Intermission 1

Are there any initial questions/comments about Darshan or how to use it?

As a starting point, you can try running an example program with Darshan instrumentation enabled to ensure the toolchain works as expected

- Follow instructions at **darshan-hands-on/README.md** to setup environment, compile and run examples, find Darshan output, and run analysis tools
  - You'll have to copy generated HTML reports to your own workstation to view in a browser using `scp` command
- Use the **helloworld** example or try it with an application of your own

- ❑ How many files did the application open?
- ❑ How much data did it read, and how much data did it write?
- ❑ What approximate I/O performance did it achieve?

HPC I/O insights with Darshan

A quick survey of the HPC I/O landscape

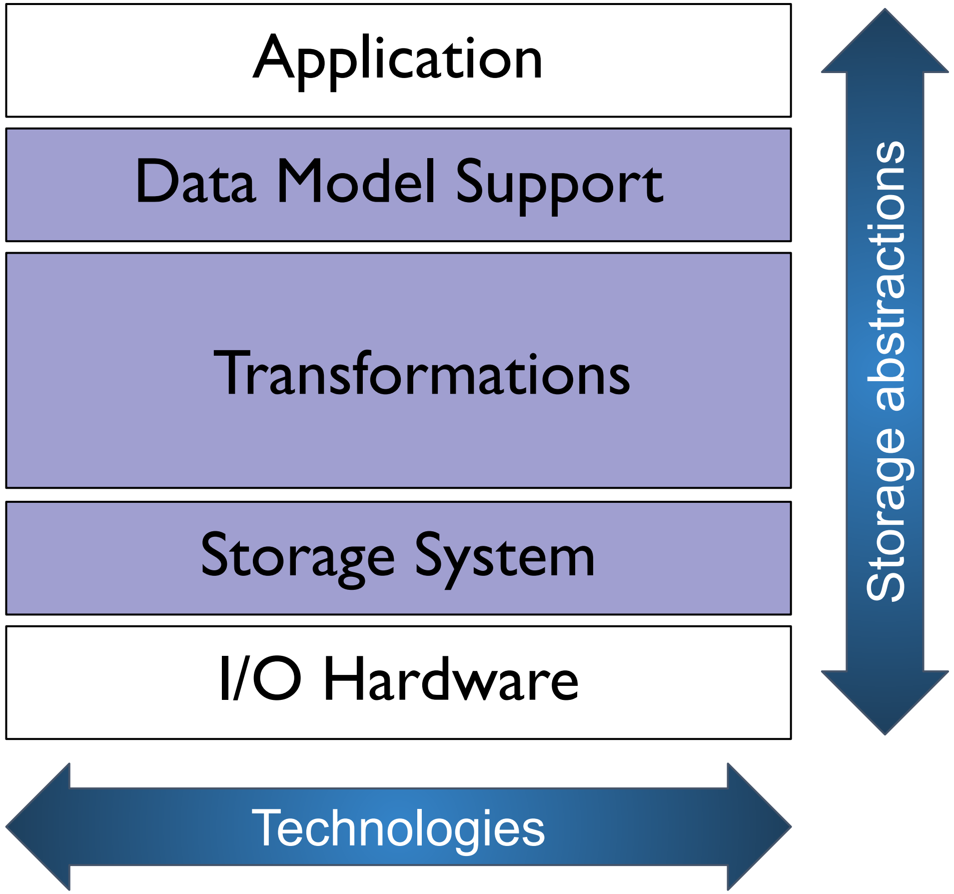
A complex data management ecosystem

As we discussed this morning, the HPC I/O landscape is deep and vast

- High-level data abstractions: HDF5, PnetCDF
- I/O middleware: MPI-IO
- Storage systems: Lustre, GPFS, DAOS
- Storage hardware: HDDs, SSDs, SCM

HPC applications themselves are evolving and encountering new data management challenges

Understanding I/O behavior in this environment is difficult, much less turning observations into actionable I/O tuning decisions



Argonne NATIONAL LABORATORY

A quick survey of the HPC I/O landscape

A complex data management ecosystem

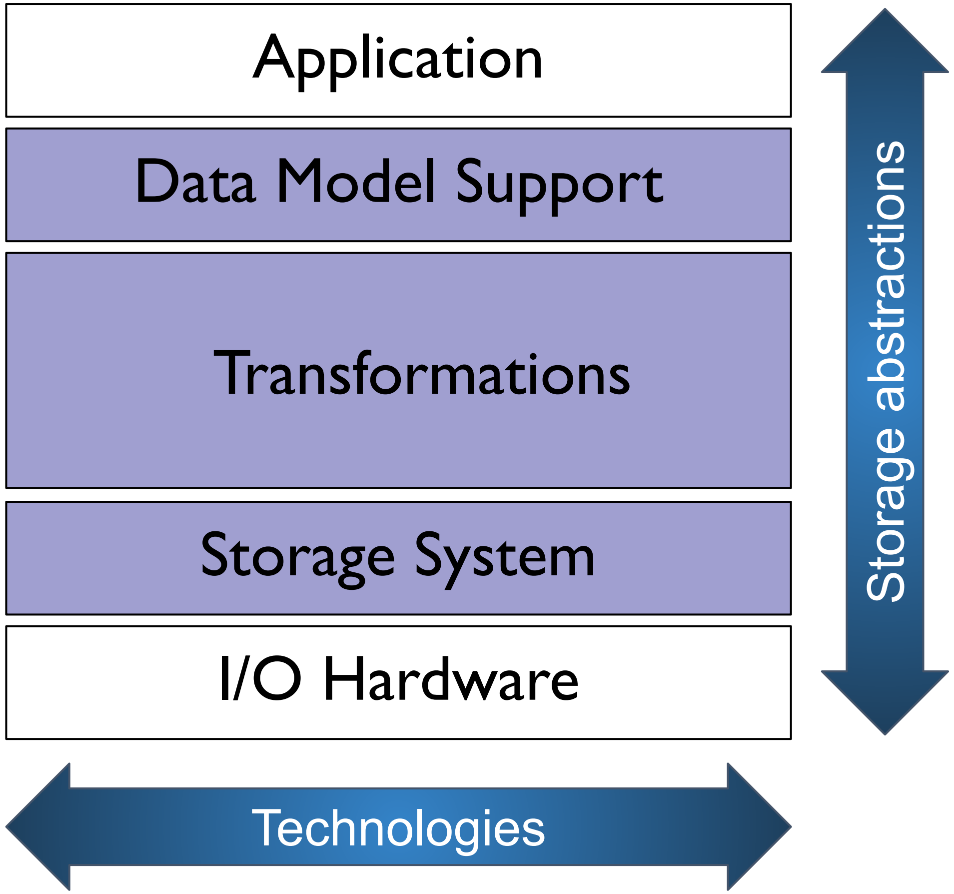
As we discussed this morning, the HPC I/O landscape is deep and vast

- High-level data abstractions: HDF5, PnetCDF
- I/O middleware: MPI-IO
- Storage systems: Lustre, GPFS, DAOS
- Storage hardware: HDDs, SSDs, SCM

HPC applications themselves are evolving and encountering new data management challenges

Understanding I/O behavior in this environment is difficult, much less turning observations into actionable I/O tuning decisions

Darshan can help navigate this complexity by characterizing I/O behavior across the I/O stack



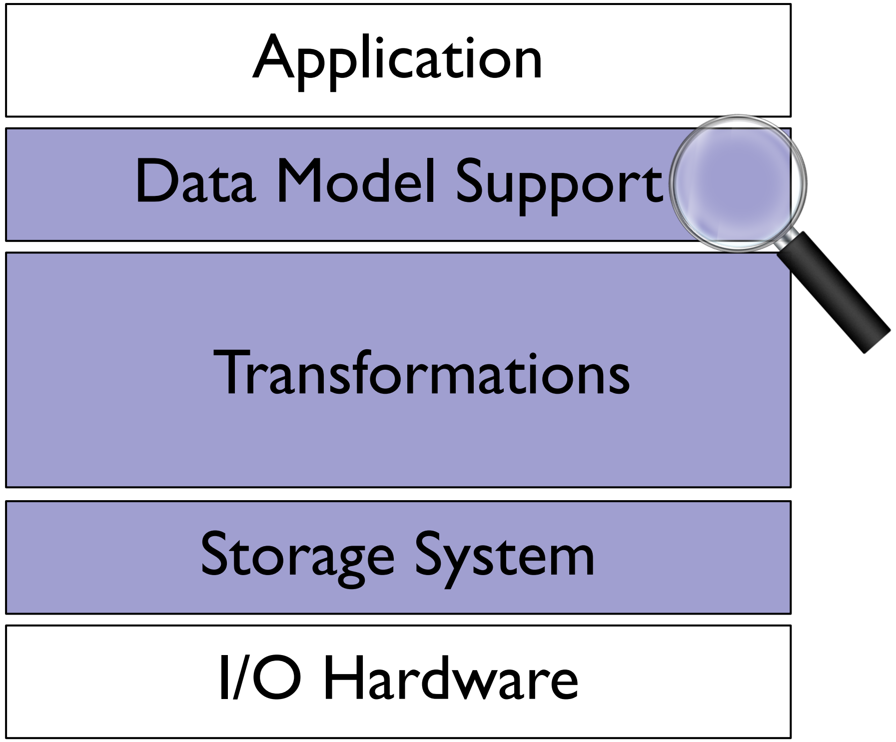
Argonne NATIONAL LABORATORY

Characterizing HPC I/O workloads with Darshan

A look under the hood of an HPC application

You have already heard some basics about Darshan, a powerful tool for users to better understand and tune their I/O workloads

Darshan provides many helpful stats across multiple layers of the I/O stack that are critical to understanding application I/O behavior



HDF5 stats*:

- Accessed files/datasets
- Operation counts
- Total read/write volumes
- Common access info (including details of hyperslab accesses)
- Chunking parameters
- Dataset dimensionality and size
- MPI-IO usage
- I/O timing

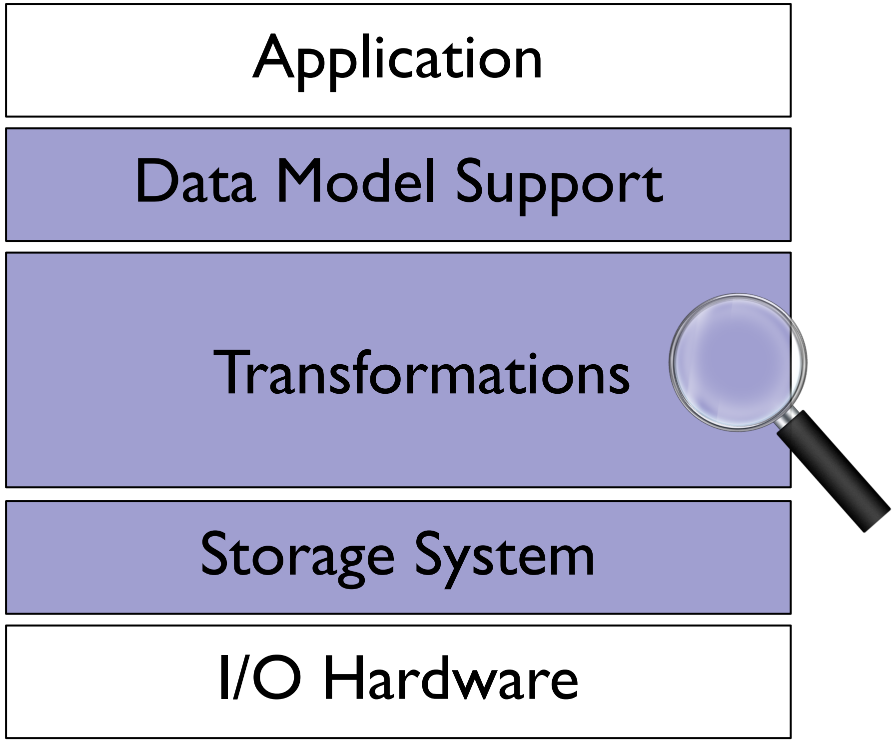
*Note: HDF5 instrumentation is not typically enabled for facility Darshan installs – you will need to install this version yourself

Characterizing HPC I/O workloads with Darshan

A look under the hood of an HPC application

You have already heard some basics about Darshan, a powerful tool for users to better understand and tune their I/O workloads

Darshan provides many helpful stats across multiple layers of the I/O stack that are critical to understanding application I/O behavior



MPI-IO stats:

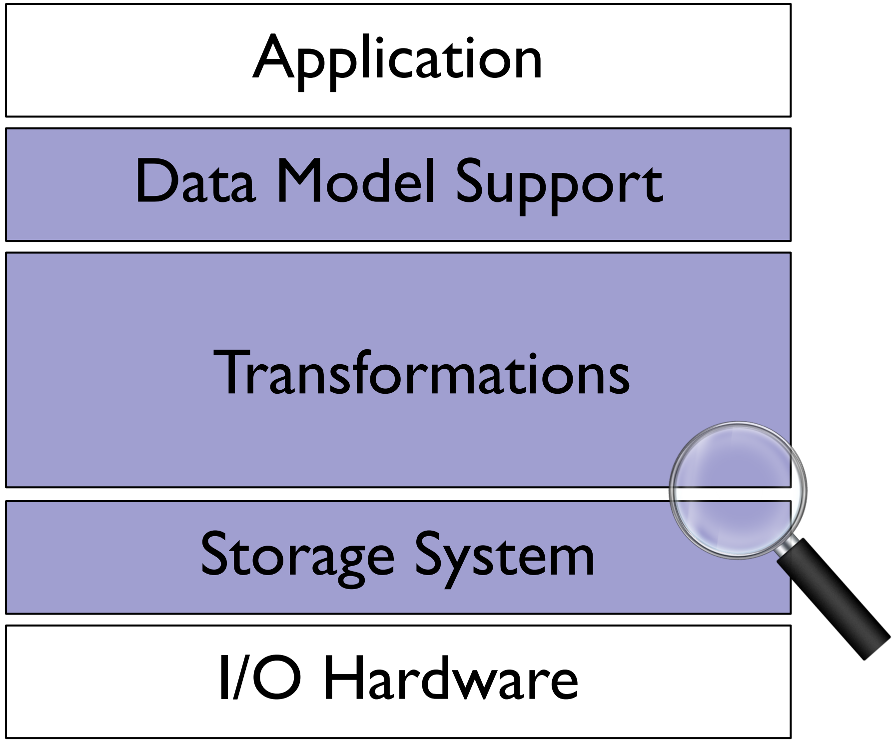
- Operation counts (open, read, write, sync)
- Collective and independent I/O usage
- Total read/write volumes
- Access size info
  - Common values
  - Histograms
- I/O timing

Characterizing HPC I/O workloads with Darshan

A look under the hood of an HPC application

You have already heard some basics about Darshan, a powerful tool for users to better understand and tune their I/O workloads

Darshan provides many helpful stats across multiple layers of the I/O stack that are critical to understanding application I/O behavior



POSIX stats:

- Operation counts (open, read, write, seek, stat)
- Total read/write volumes
- File alignment
- Access size/stride info
  - Common values
  - Histograms
- I/O timing

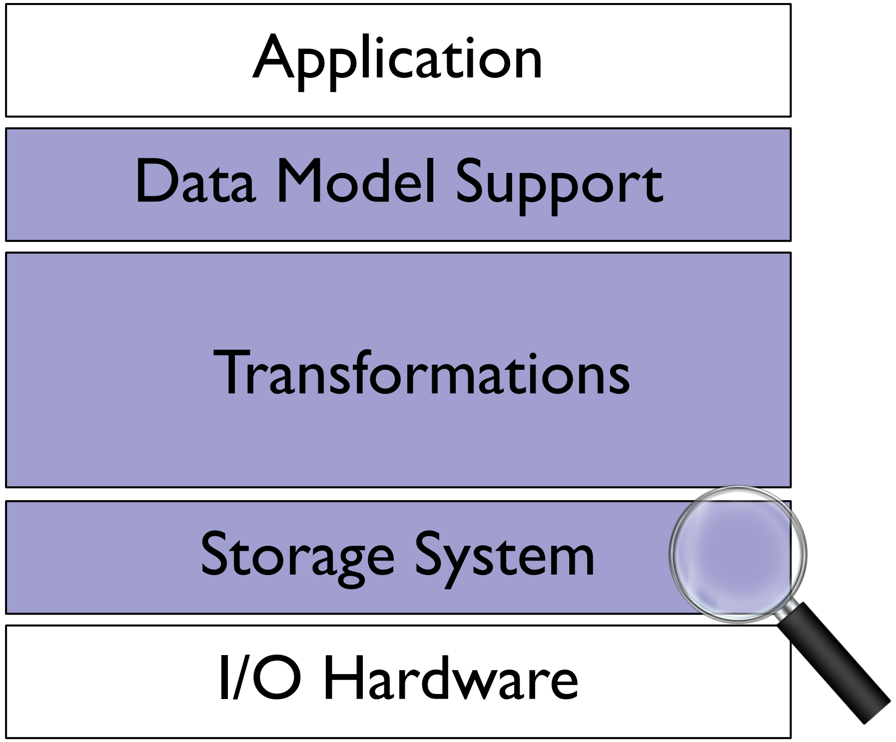
Argonne NATIONAL LABORATORY

Characterizing HPC I/O workloads with Darshan

A look under the hood of an HPC application

You have already heard some basics about Darshan, a powerful tool for users to better understand and tune their I/O workloads

Darshan provides many helpful stats across multiple layers of the I/O stack that are critical to understanding application I/O behavior



Lustre stats:

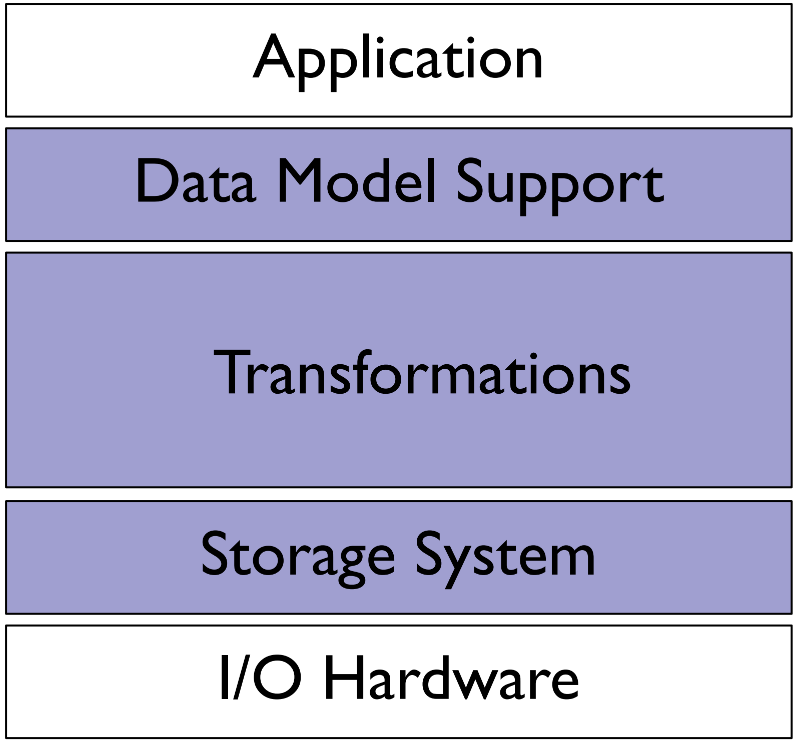
- Data server (OST) and metadata server (MDT) counts
- Stripe size/width
- OST list serving a file

Characterizing HPC I/O workloads with Darshan

A look under the hood of an HPC application

You have already heard some basics about Darshan, a powerful tool for users to better understand and tune their I/O workloads

Darshan provides many helpful stats across multiple layers of the I/O stack that are critical to understanding application I/O behavior



Let’s see how Darshan can be leveraged in some practical use cases that demonstrate some general best practices in tuning HPC I/O performance

Argonne NATIONAL LABORATORY

Tuning the storage system

Ensuring storage resources match application I/O needs

For some parallel file systems like Lustre, users have direct control over file striping parameters

Bad news: Users may have to have some knowledge of the file system to get good I/O performance

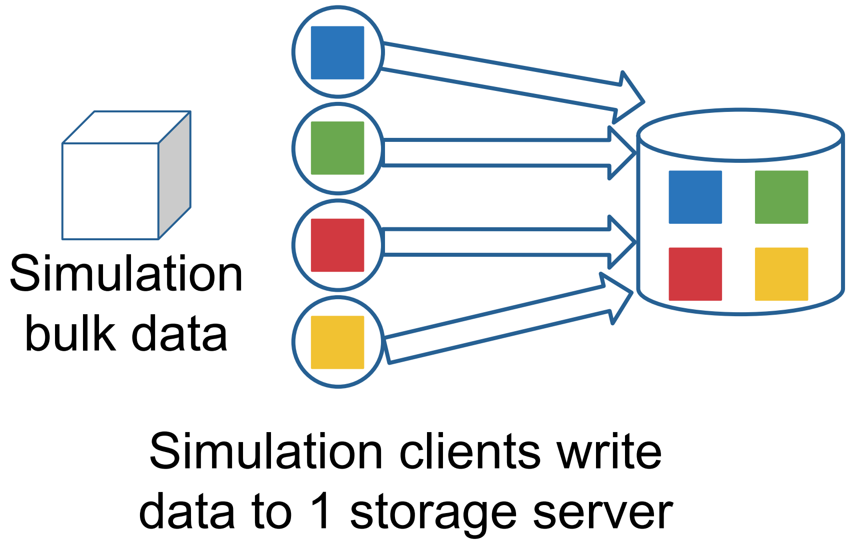
Good news: Users can often get higher I/O performance than system defaults with thoughtful tuning -- file systems aren't perfect for every workload!

Argonne NATIONAL LABORATORY

Tuning the storage system

Ensuring storage resources match application I/O needs

Tuning decisions can and should be made independently for different file types



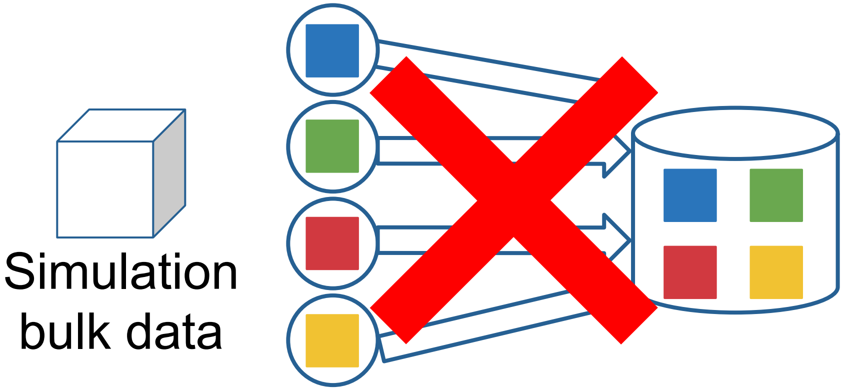
Argonne NATIONAL LABORATORY

Tuning the storage system

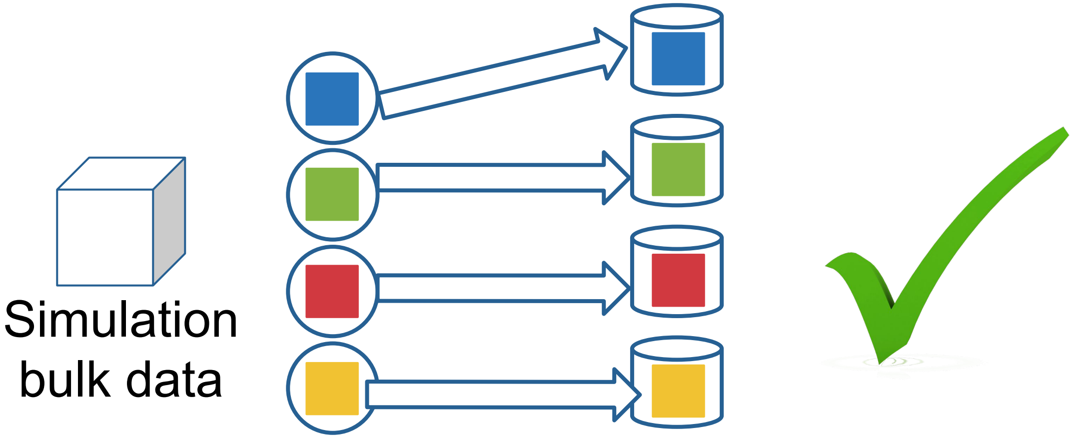
Ensuring storage resources match application I/O needs

Tuning decisions can and should be made independently for different file types

Large application datasets should ideally be distributed across as many storage resources as possible



Simulation clients write data to 1 storage server



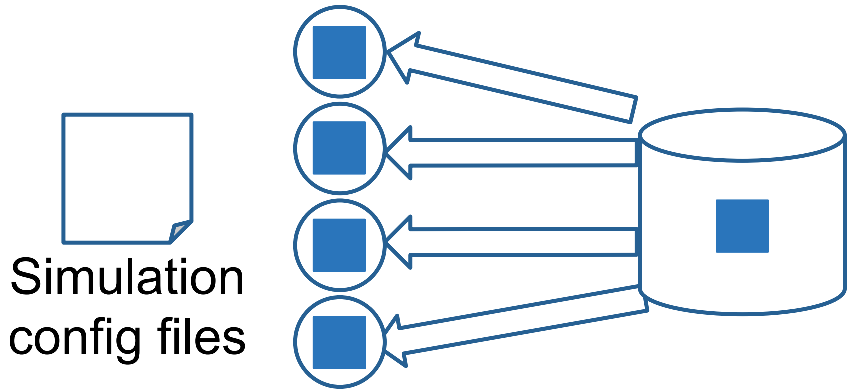
Simulation clients load balance writes across multiple servers

Tuning the storage system

Ensuring storage resources match application I/O needs

Tuning decisions can and should be made independently for different file types

On the other hand, smaller files often benefit from being stored on a single server



Simulation clients read config data from 1 storage server

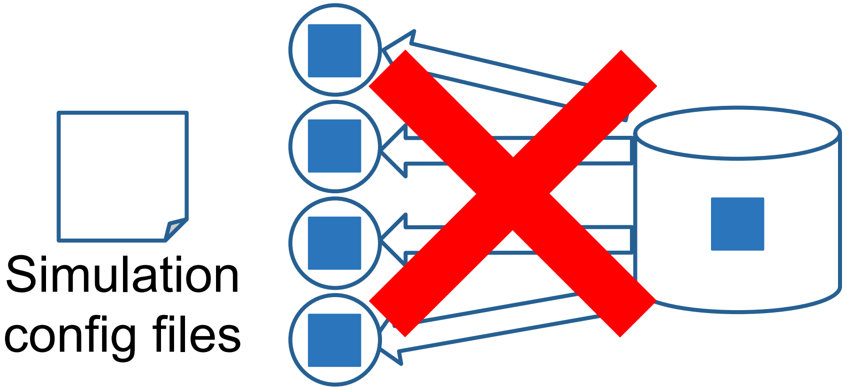
Argonne NATIONAL LABORATORY

Tuning the storage system

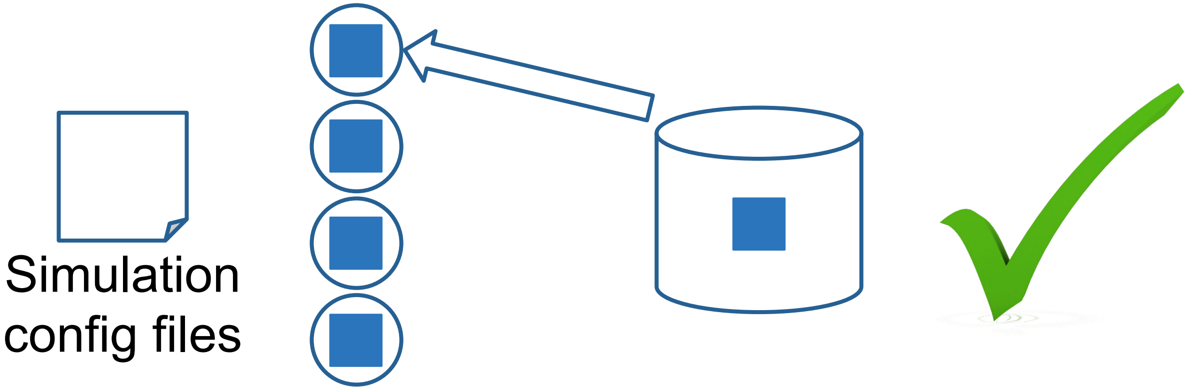
Ensuring storage resources match application I/O needs

Tuning decisions can and should be made independently for different file types

On the other hand, smaller files often benefit from being stored on a single server



Simulation clients read config data from 1 storage server



Better yet, limit storage contention by having 1 client read data and distribute using communication (e.g., MPI)

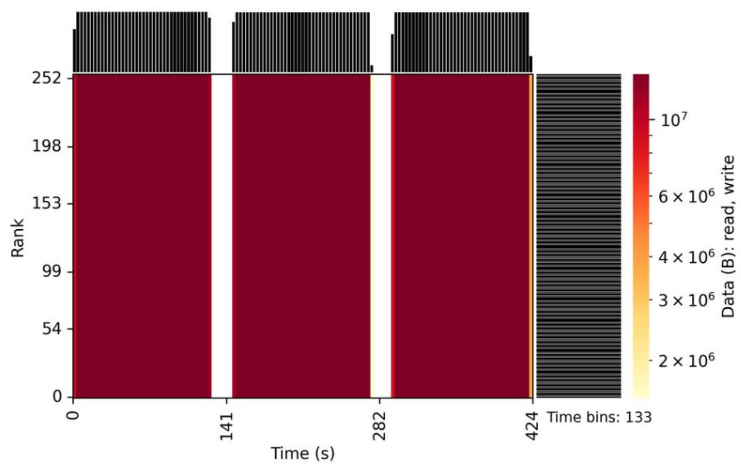
Tuning the storage system

Ensuring storage resources match application I/O needs

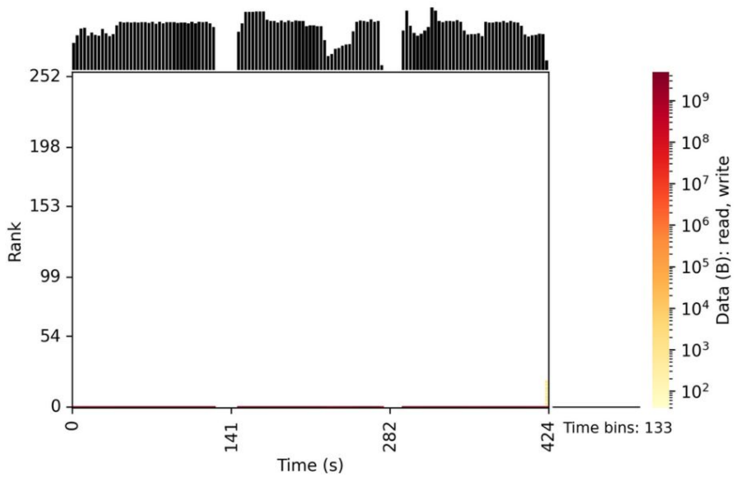
Be aware of what file system settings are available to you and don't assume system defaults are always the best… you might be surprised what you find

- ALCF Polaris/Theta and NERSC Perlmutter Lustre scratch file systems both have a default stripe width of 1 (i.e., files are stored on one server by default)

MPI-IO



POSIX



256 process (4 node) h5bench[1] runs on NERSC Perlmutter

h5bench contains lots of parameters for controlling characteristics of generated HDF5 workloads.

1. https://github.com/hpc-io/h5bench

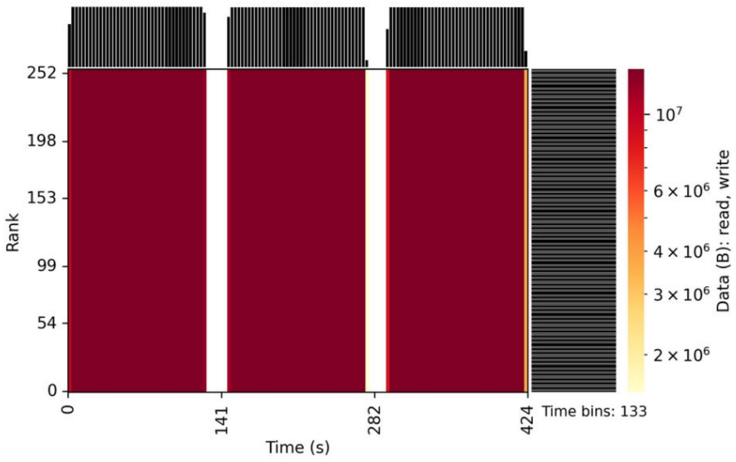
Tuning the storage system

Ensuring storage resources match application I/O needs

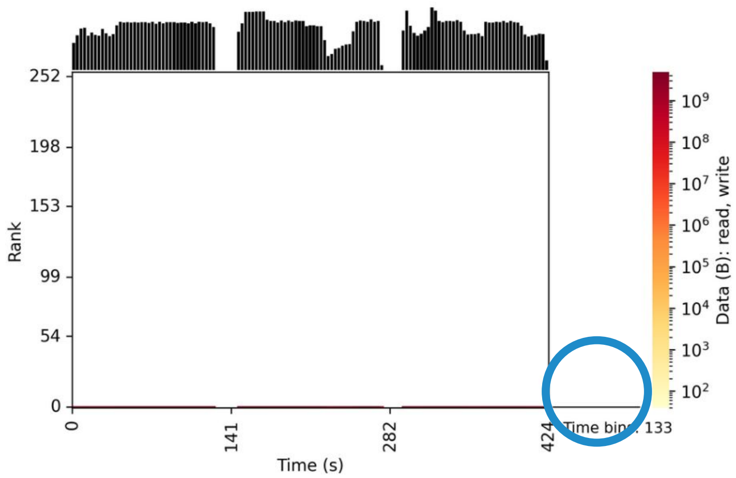
Be aware of what file system settings are available to you and don't assume system defaults are always the best… you might be surprised what you find

- ALCF Polaris/Theta and NERSC Perlmutter Lustre scratch file systems both have a default stripe width of 1 (i.e., files are stored on one server by default)

MPI-IO



POSIX

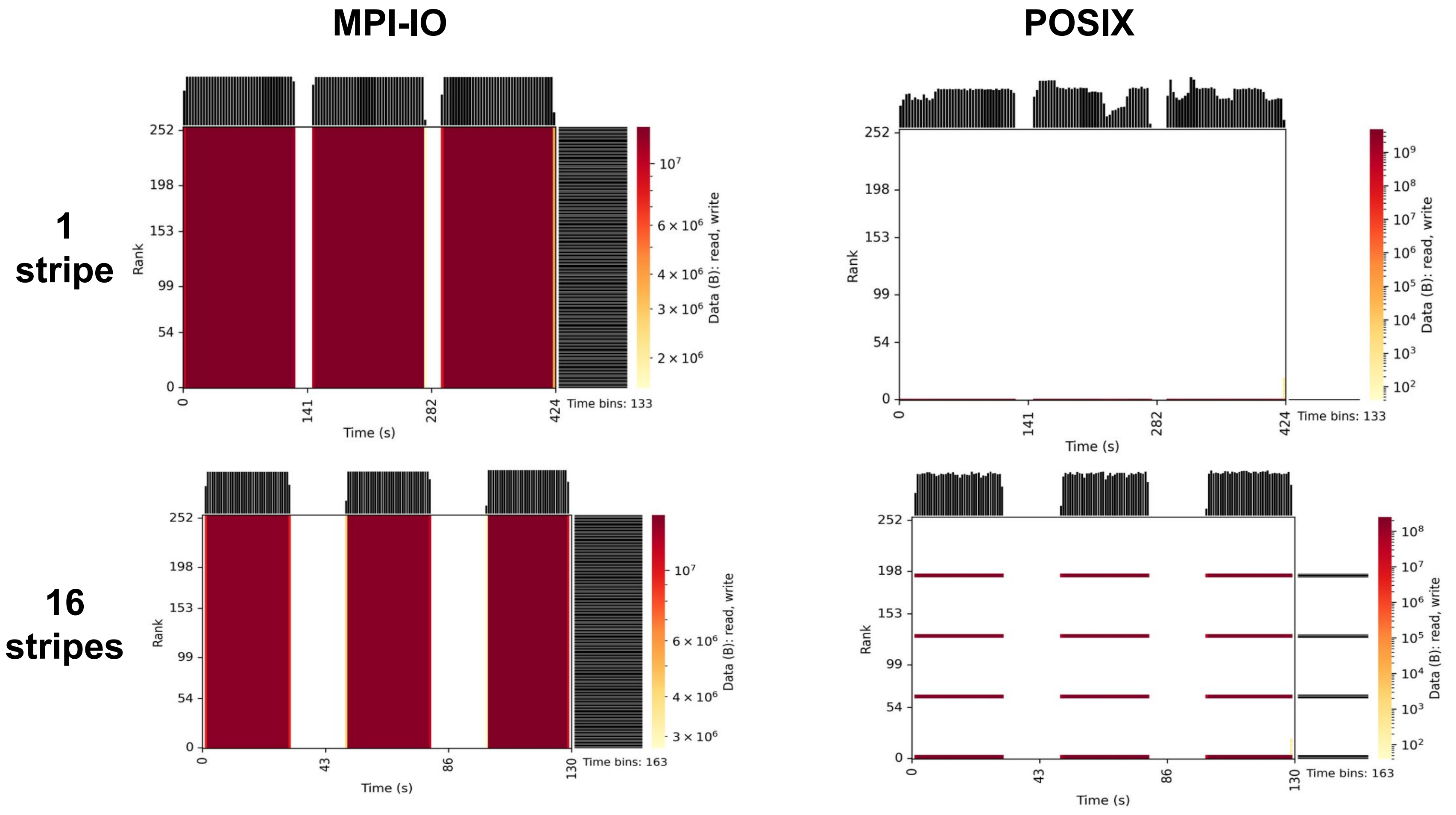


All I/O funneled through rank 0

MPI-IO collective I/O driver for Lustre assigns dedicated aggregators for each stripe, yielding a single aggregator for files of 1 stripe

Tuning the storage system

Ensuring storage resources match application I/O needs



Manually setting the stripe width to 16 yields more I/O aggregators and better performance:

```
> lfs setstripe -c 16 testFile
```

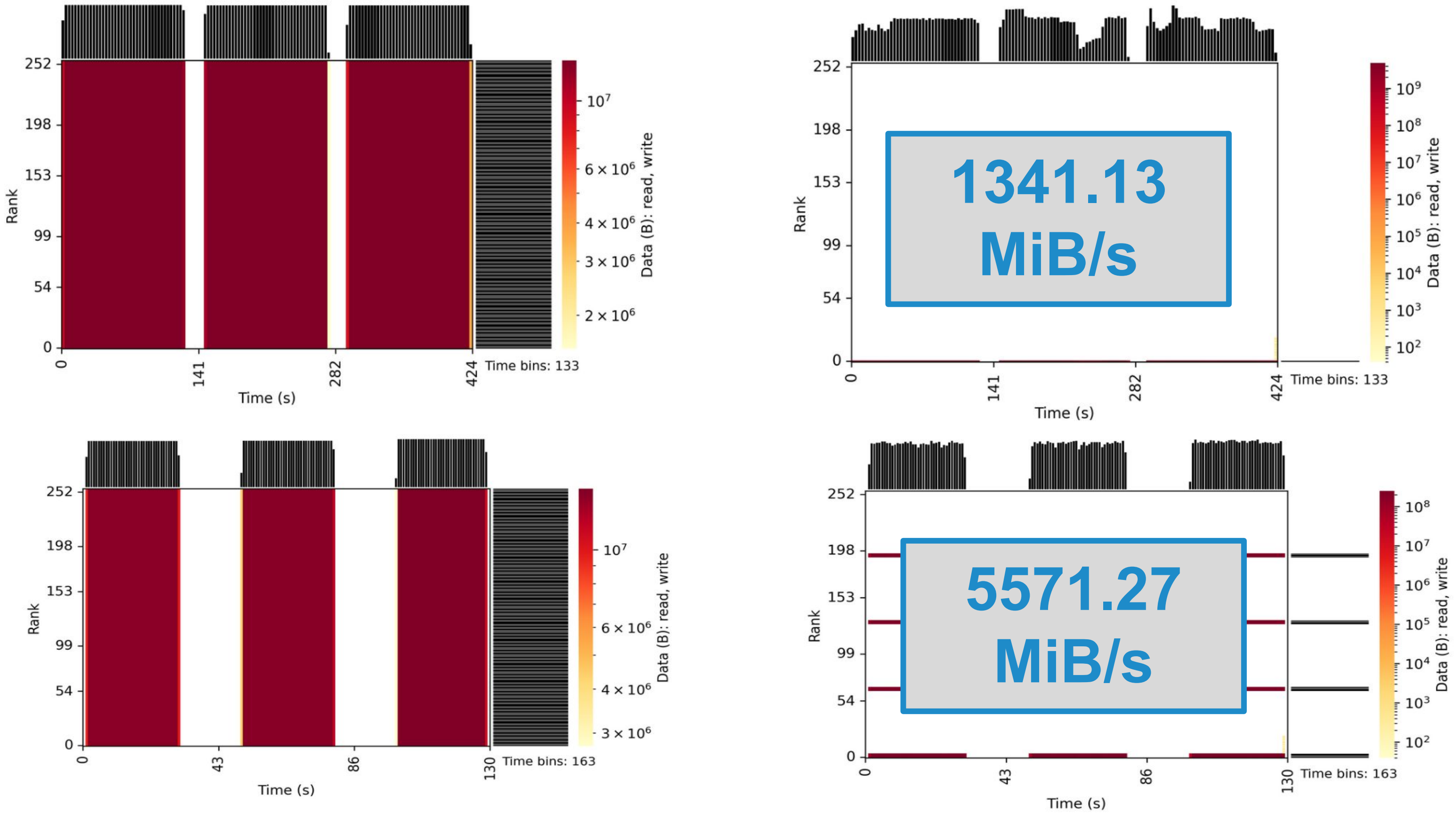
Tuning the storage system

Ensuring storage resources match application I/O needs

MPI-IO **POSIX**

1 stripe

16 stripes



Manually setting the stripe width to 16 yields more I/O aggregators and better performance:

```
> lfs setstripe -c 16 testFile
```

4x performance improvement!

Argonne NATIONAL LABORATORY

Tuning the storage system

Ensuring storage resources match application I/O needs

Consult facilities documentation for established best practice!

	Single Shared-File I/O	File per Process
File size	Command	Command
< 1 GB	keep default striping	keep default striping
1 - 10 GB	`stripe_small`	keep default striping
10 - 100 GB	`stripe_medium`	keep default striping
100 GB - 1 TB	`stripe_large`	keep default striping
> 1 TB	`stripe_large`	`stripe_large`

Perlmutter (NERSC) docs providing commands to set stripe params for various file types

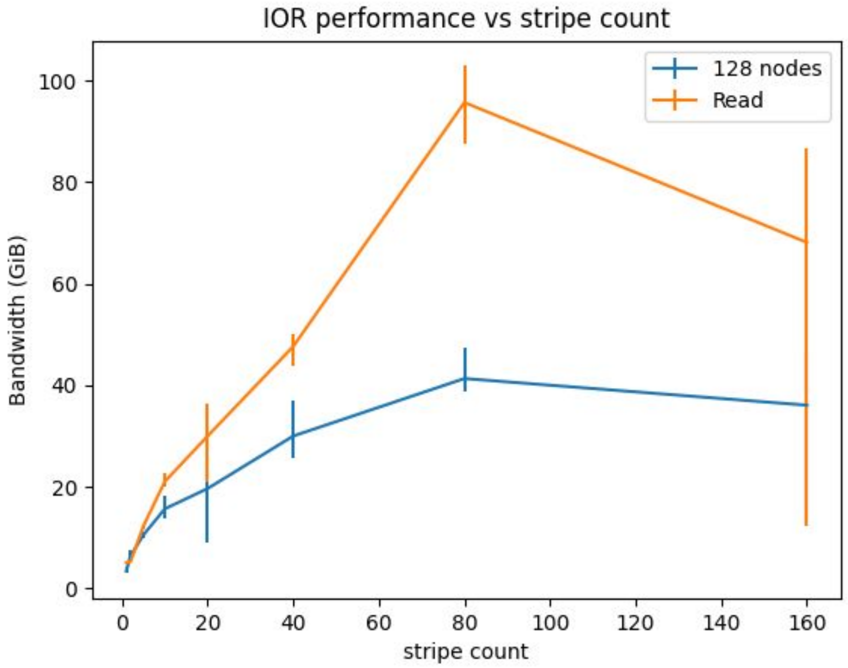
- The default striping set on Orion is targeted to work well for a variety of workloads
- In most cases, users should use this default striping. Though possible, manual striping should only occur after careful consideration and under collaboration with OLCF staff
- The default striping policy may change due to findings in production

OLCF presentation on Orion storage system detailing usage of Lustre's new progressive file layout mechanism

Tuning the storage system

Ensuring storage resources match application I/O needs

Consult facilities documentation for established best practice! Sometimes you may even need to experiment yourself.



128-node example of the IOR benchmark using various stripe counts on ALCF Polaris.

For more I/O intensive programs, it's typically better to err on the side of more storage servers. The following command stripes across all servers:

```
> lfs setstripe -c -1 testFile
```

https://github.com/radix-io/io-sleuthing/tree/main/examples/striping

Tuning low-level (POSIX) file I/O

Making efficient use of a no-frills I/O API

Users may also need to pay close attention to file system alignment when issuing I/O accesses to a file

- Accesses that are not aligned can introduce performance inefficiencies on file systems

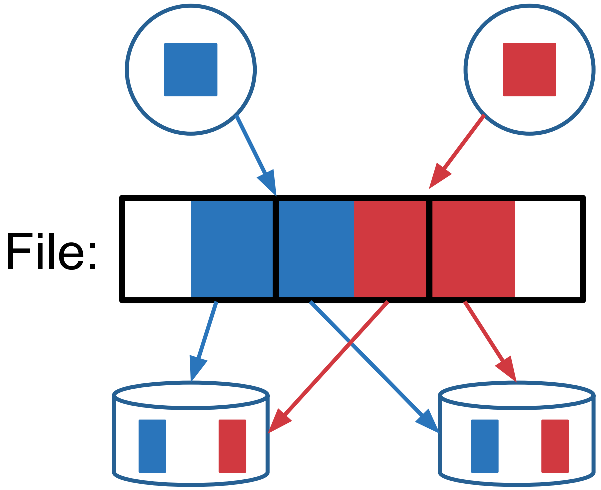
Tuning low-level (POSIX) file I/O

Making efficient use of a no-frills I/O API

Users may also need to pay close attention to file system alignment when issuing I/O accesses to a file

- Accesses that are not aligned can introduce performance inefficiencies on file systems

For Lustre, performance can be maximized by aligning I/O to stripe boundaries:



Unaligned I/O requests can span multiple servers and introduce inefficiencies in storage protocols

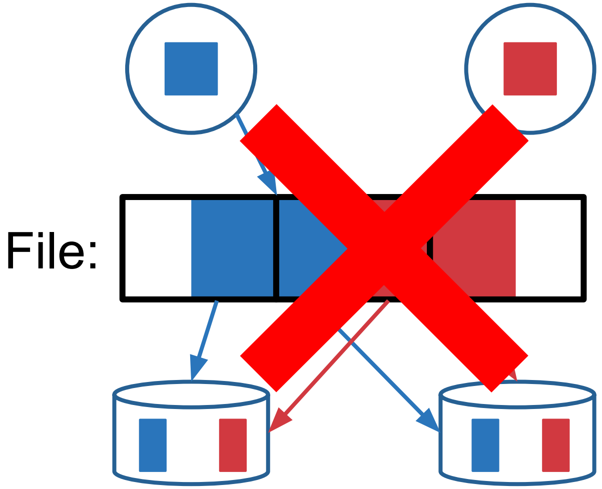
Tuning low-level (POSIX) file I/O

Making efficient use of a no-frills I/O API

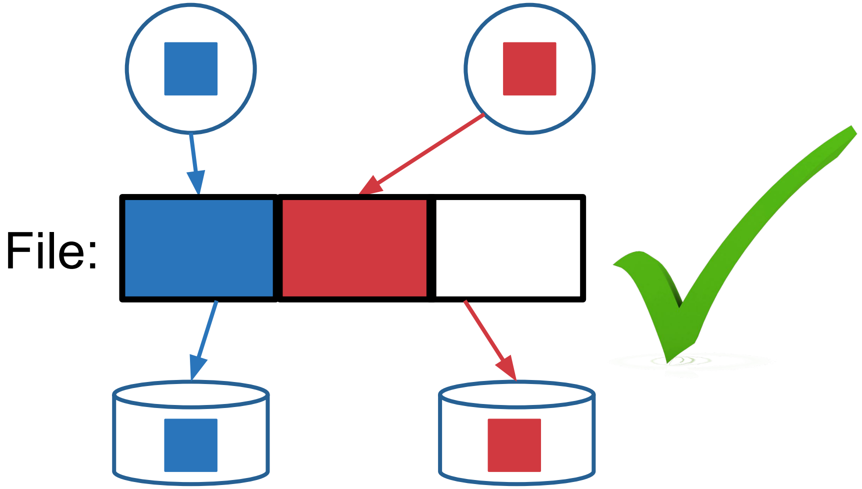
Users may also need to pay close attention to file system alignment when issuing I/O accesses to a file

- Accesses that are not aligned can introduce performance inefficiencies on file systems

For Lustre, performance can be maximized by aligning I/O to stripe boundaries:



Instead, ensure client accesses are well-aligned to avoid Lustre server contention



Tuning low-level (POSIX) file I/O

Making efficient use of a no-frills I/O API

Consider a simple 10-process (10-node) NERSC Cori example where processes write in an interleaved fashion to a single shared file

aligned

# Module	Rank	Wt/Rd	Segment	Offset	Length	Start(s)	End(s)	[OST]
X_POSIX	0	write	0	0	1048576	0.0054	0.0066	[197]
X_POSIX	0	write	1	10485760	1048576	0.0066	0.0073	[197]
X_POSIX	0	write	2	20971520	1048576	0.0073	0.0081	[197]
X_POSIX	0	write	3	31457280	1048576	0.0081	0.0088	[197]

Use Darshan's DXT tracing module to get details about each individual write access – **more details on DXT usage coming soon**

Tuning low-level (POSIX) file I/O

Making efficient use of a no-frills I/O API

Consider a simple 10-process (10-node) NERSC Cori example where processes write in an interleaved fashion to a single shared file

aligned

# Module	Rank	Wt/Rd	Segment	Offset	Length	Start(s)	End(s)	[OST]
X_POSIX	0	write	0	0	1048576	0.0054	0.0066	[197]
X_POSIX	0	write	1	10485760	1048576	0.0066	0.0073	[197]
X_POSIX	0	write	2	20971520	1048576	0.0073	0.0081	[197]
X_POSIX	0	write	3	31457280	1048576	0.0081	0.0088	[197]

Each access is aligned to the Lustre stripe size (1 MiB)

Each process interacts with a single Lustre server (OST)

Tuning low-level (POSIX) file I/O

Making efficient use of a no-frills I/O API

Consider a simple 10-process (10-node) NERSC Cori example where processes write in an interleaved fashion to a single shared file

unaligned

# Module	Rank	Wt/Rd	Segment	Offset	Length	Start(s)	End(s)	[OST]	
X_POSIX	0	write	0	524288	1048576	0.0065	0.0594	[32]	[197]
X_POSIX	0	write	1	11010048	1048576	0.0594	0.1268	[32]	[197]
X_POSIX	0	write	2	21495808	1048576	0.1268	0.2060	[32]	[197]
X_POSIX	0	write	3	31981568	1048576	0.2060	0.2069	[32]	[197]

Each access spans two Lustre stripes due to unaligned offsets

Each process interacts with two Lustre servers (OSTs)

Argonne NATIONAL LABORATORY

Tuning low-level (POSIX) file I/O

Making efficient use of a no-frills I/O API

Even in this small workload, we pay a nearly **20% performance penalty when I/O accesses are not aligned** to file stripes (1 MB)

aligned

# Module	Rank	Wt/Rd	Segment	Offset	Length	Start(s)	End(s)
X_POSIX	0	write	0	0	1048576	0.0054	0.0
X_POSIX	0	write	1	10485760	1048576	0.0066	0.0
X_POSIX	0	write	2	20971520	1048576	0.0073	0.0
X_POSIX	0	write	3	31457280	1048576	0.0081	0.0

310.14 MiB/s

unaligned

# Module	Rank	Wt/Rd	Segment	Offset	Length	Start(s)	End(s)	[OST
X_POSIX	0	write	0	524288	1048576	0.0065	0.0594	[
X_POSIX	0	write	1	11010048	1048576	0.0594	0.1268	[
X_POSIX	0	write	2	21495808	1048576	0.1268	0.2060	[
X_POSIX	0	write	3	31981568	1048576	0.2060	0.2069	[

380.28 MiB/s

Argonne NATIONAL LABORATORY

Tuning low-level (POSIX) file I/O

Making efficient use of a no-frills I/O API

Accounting for subtle I/O performance factors like file alignment can be a painstaking process…

High-level I/O libraries like HDF5 can help mask much of the complexity needed for transforming scientific computing I/O workloads into performant POSIX-level file system accesses – ***don't reinvent the wheel, use high-level I/O libraries wherever you can!***

Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

The HDF5 library provides a chunking mechanism to partition user datasets into contiguous chunks in the underlying file

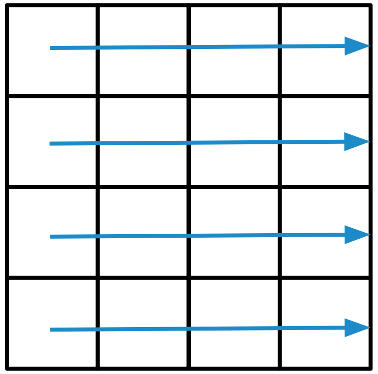
- Users can greatly improve performance of partial dataset I/O operations by choosing chunking parameters that match expected access patterns

Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

The HDF5 library provides a chunking mechanism to partition user datasets into contiguous chunks in the underlying file

- Users can greatly improve performance of partial dataset I/O operations by choosing chunking parameters that match expected access patterns



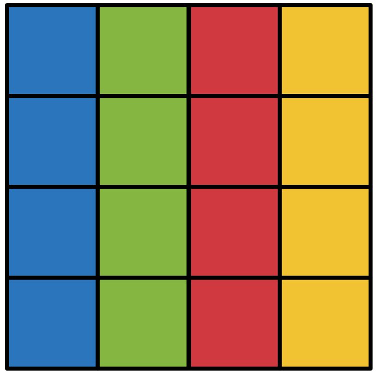
By default, HDF5 will store the dataset contiguously row-by-row (i.e., row-major format) in the file

Tuning high-level (HDF5) data access

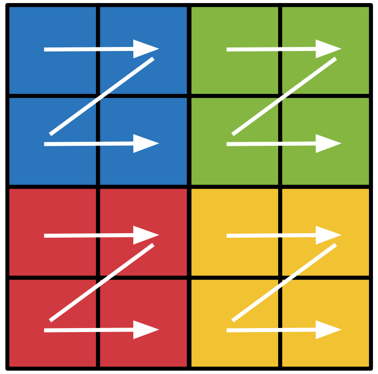
Optimizing application interactions with the I/O stack

The HDF5 library provides a chunking mechanism to partition user datasets into contiguous chunks in the underlying file

- Users can greatly improve performance of partial dataset I/O operations by choosing chunking parameters that match expected access patterns



column-based



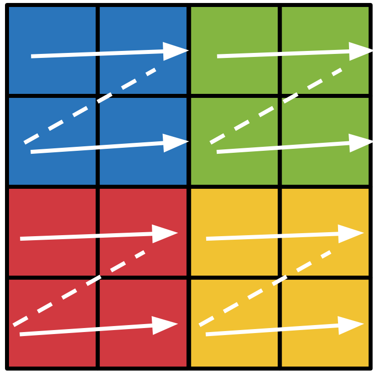
block-based

If dataset access patterns do not suit a simple row-major storage scheme, chunking can be applied to map chunks of dataset data to contiguous regions in the file

Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

Consider a 256-process (4-node) Polaris example where each process exclusively writes a 2048x2048 block of the dataset (32 MB per-process, 8 GB total)

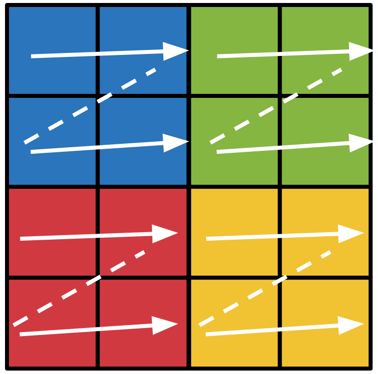


With no chunking, each process must issue many smaller non-contiguous I/O requests (solid lines) and seek around the file (dashed lines), yielding low I/O performance

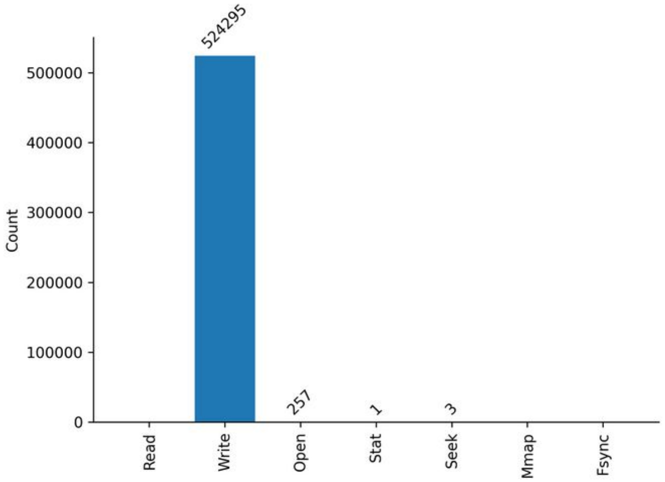
Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

Consider a 256-process (4-node) Polaris example where each process exclusively writes a 2048x2048 block of the dataset (32 MB per-process, 8 GB total)



I/O performance estimate	503.47 MiB/s (average)



Access Size	Count
16384	524288
96	2
328	1
544	1

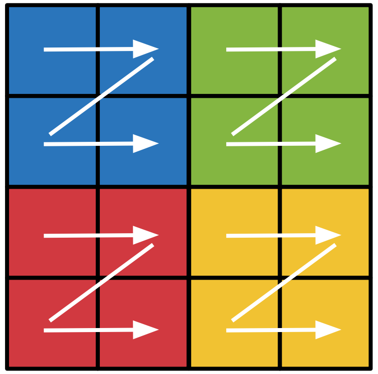
256 individual HDF5 writes (1-per-process) yields 500K+ POSIX writes

Argonne NATIONAL LABORATORY

Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

Consider a 256-process (4-node) Polaris example where each process exclusively writes a 2048x2048 block of the dataset (32 MB per-process, 8 GB total)

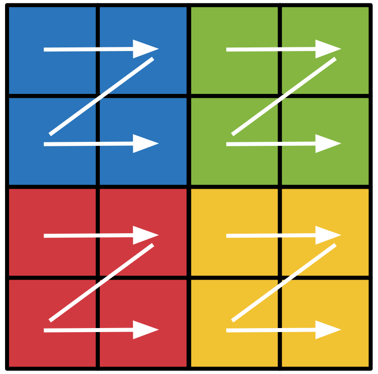


With chunking applied, each process can read their entire data block using one large, contiguous access in the file

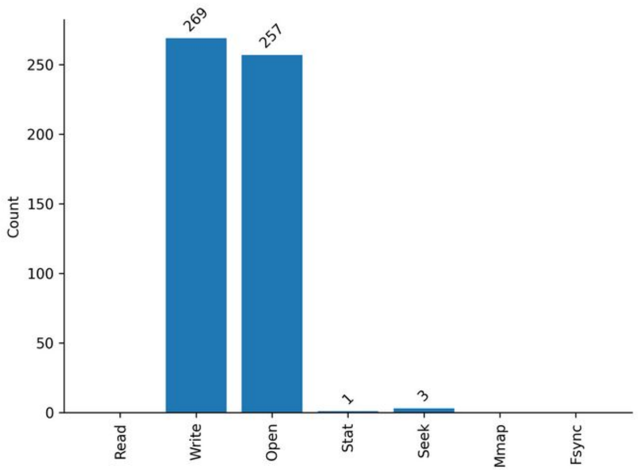
Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

Consider a 256-process (4-node) Polaris example where each process exclusively writes a 2048x2048 block of the dataset (32 MB per-process, 8 GB total)



I/O performance estimate	1450.57 MiB/s (average)



Access Size	Count
33554432	256
2616	6
96	2
544	1

Chunking results in a much more manageable POSIX workload

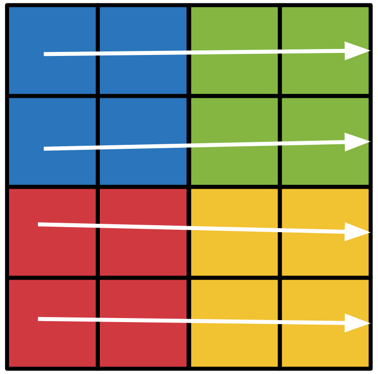
Nearly a 3x performance improvement!

Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

An alternative optimization forgoes chunking and uses collective I/O to improve the efficiency of this block-style data access

- Rely on MPI-IO layer collective buffering algorithm to generate contiguous storage accesses and to limit number of clients interacting with storage system



With collective I/O enabled, designated aggregator processes perform I/O on behalf of their peers, and communicate their data using MPI calls

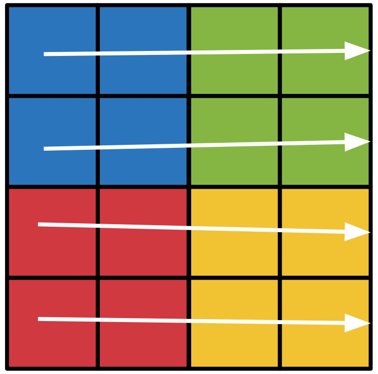
E.g., the green process sends its write data to the blue process (aggregator), who then writes both of their data in one big contiguous chunk

Argonne NATIONAL LABORATORY

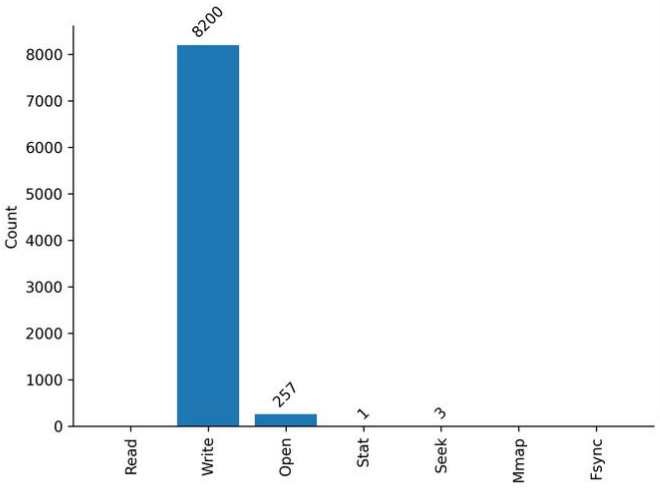
Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

Consider a 256-process (4-node) Polaris example where each process exclusively writes a 2048x2048 block of the dataset (32 MB per-process, 8 GB total)



I/O performance estimate	13124.01 MiB/s (average)



Access Size	Count
1048576	8191
96	2
2048	1
1046528	1

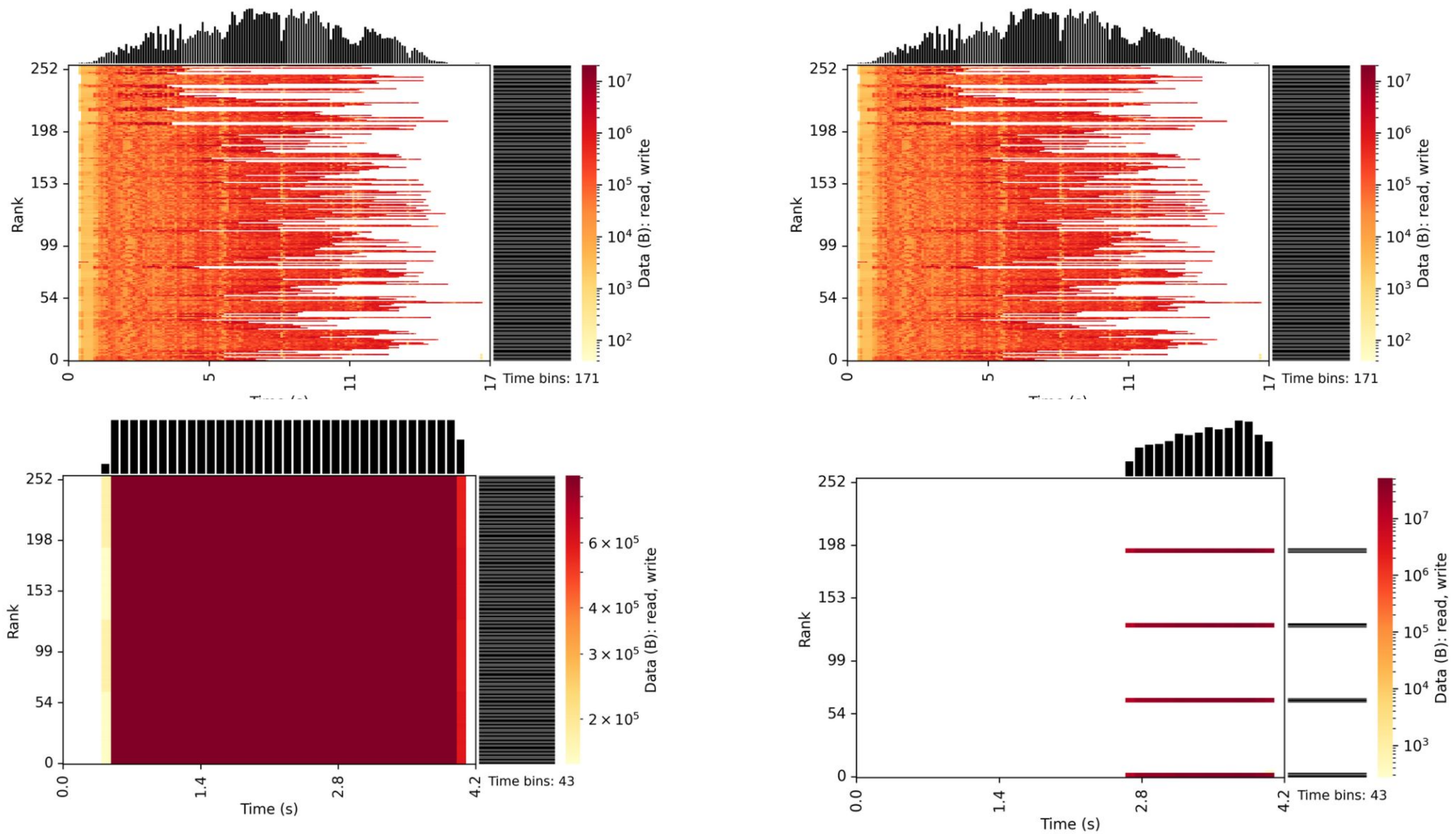
Collective I/O yields 26x improvement over no chunking, and 9x improvement over chunking!!!

Argonne NATIONAL LABORATORY

Tuning high-level (HDF5) data access

Optimizing application interactions with the I/O stack

MPI-IO **POSIX**



Darshan I/O activity heatmaps illustrate how different the I/O behavior is for the unoptimized independent configuration (**top**) and the most performant collective I/O configuration (**bottom**)

Summarizing I/O tuning options

As a user of I/O interface X, what tuning vectors do I have?

I/O Interface	Striping	Alignment	Collective I/O	Chunking
HDF5	✓	✓	✓	✓
MPI-IO	✓	✓	✓	X
POSIX	✓	✓-	X	X

Summarizing I/O tuning options

As a user of I/O interface X, what tuning vectors do I have?

I/O Interface	Striping	Alignment	Collective I/O	Chunking
HDF5	✓	✓	✓	✓
MPI-IO	✓	✓	✓	X
POSIX	✓	✓-	X	X

Automatically align application data and library metadata, if user requests so

Collective I/O can be automatically aligned

POSIX I/O requires manually aligning every access

Argonne NATIONAL LABORATORY

Summarizing I/O tuning options

As a user of I/O interface X, what tuning vectors do I have?

I/O Interface	Striping	Alignment	Collective I/O	Chunking
HDF5	✓	✓	✓	✓
MPI-IO	✓	✓	✓	X
POSIX	✓	✓-	X	X

Just another reminder that high-level I/O libraries are here to make your life easier

- I/O optimization strategies like collective I/O & chunking can net large performance gains, especially when combined with striping and alignment optimizations

Hands-on Intermission 2

In the **darshan-hands-on** directory there are 2 more examples with **A** & **B** versions: **warpdrive** & **fidgetspinner**

- Follow instructions at **darshan-hands-on/README.md** to setup environment, compile and run examples, find Darshan output, and run analysis tools
- Note: these examples will require at least some understanding of the MPI-IO library

See if you can spot the performance differences! Which version is faster? Why?

  - Use Darshan job summary tool to compare I/O behavior
  - Compare source code using diff to confirm

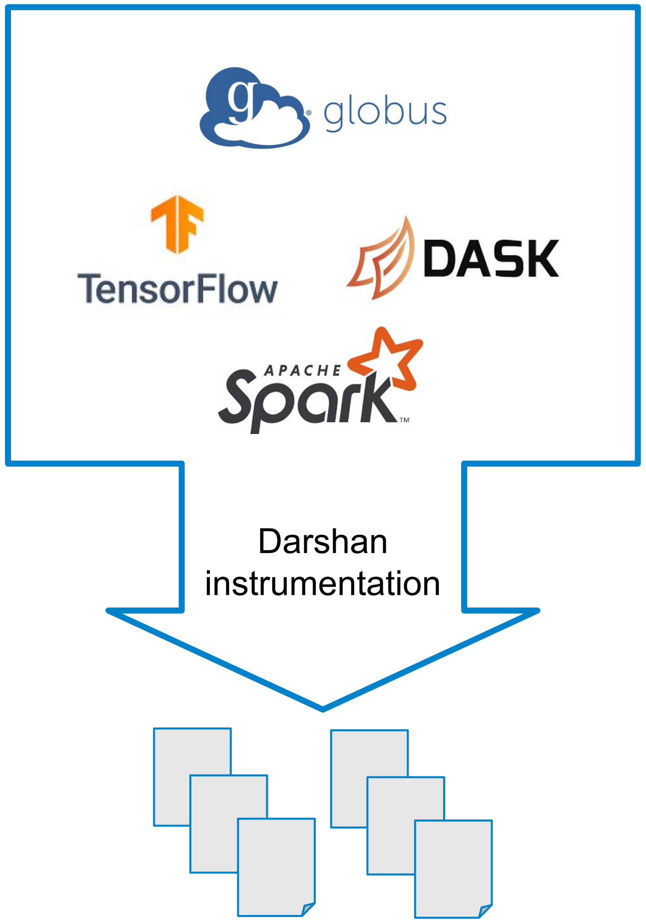
Argonne NATIONAL LABORATORY

Additional Darshan tips and tricks

Darshan instrumentation beyond MPI

- Historically, Darshan has only worked with MPI applications
  - `MPI_Init/MPI_Finalize` used to bootstrap/shutdown Darshan
- Darshan has been modified to use a secondary bootstrapping mechanism that enables its use outside of MPI
  - Based on GCC-specific library constructor/destructor attributes
  - **Only works for dynamically-linked executables!**
- To enable non-MPI mode, users must explicitly opt-in by setting the **`DARSHAN_ENABLE_NONMPI`** environment variable
  - A unique log will be generated for every process that executes
  - Often best to limit instrumentation scope to the target executable:

```
$ LD_PRELOAD=/path/to/libdarshan.so \
  DARSHAN_ENABLE_NONMPI=1 \
  ./exe <args>
```



Finer-grained details with Darshan: DXT tracing

- By default, Darshan captures a fixed set of counters for each file
- With DXT, Darshan additionally traces every read/write operation (for POSIX and MPI-IO interfaces)
- Enable by setting **DXT_ENABLE_IO_TRACE** env variable
- Finer grained instrumentation data comes at a cost of additional overhead and larger logs

```
export DXT_ENABLE_IO_TRACE=1

mpiexec -n 256 --ppn 64 ./helloworld
```

Finer-grained details with Darshan: DXT tracing

- By default, Darshan captures a fixed set of counters for each file
- With DXT, Darshan additionally traces every read/write operation (for POSIX and MPI-IO interfaces)
- Enable by setting **`DXT_ENABLE_IO_TRACE`** env variable
- Finer grained instrumentation data comes at a cost of additional overhead and larger logs

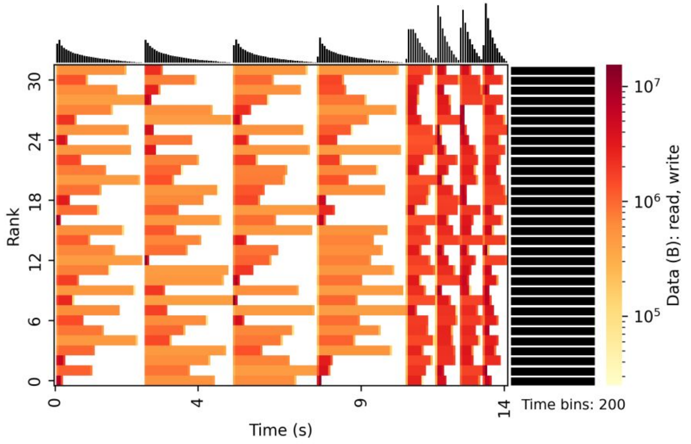
```
# DXT, file_id: 1163774110118722858, file_name: /grand/projects/ATPESC2023/usr/snyder/hello
# DXT, rank: 0, hostname: x3202c0s1b0n0
# DXT, write_count: 160, read_count: 0
# DXT, mnt_pt: /, fs_type: overlay
# Module    Rank  Wt/Rd  Segment          Offset       Length    Start(s)      End(s)
 X_POSIX       0  write        0               0      1048576      3.9347      3.9468
 X_POSIX       0  write        1       167772160      1048576      4.2503      4.2575
 X_POSIX       0  write        2       335544320      1048576      4.5495      4.5564
 X_POSIX       0  write        3       503316480      1048576      4.8632      4.8707
```

Trace includes the timestamp, file offset, and size of every I/O operation on every rank.

`darshan-dxt-parser` utility can provide a raw text dump of the trace

Finer-grained details with Darshan: DXT tracing

- By default, Darshan captures a fixed set of counters for each file
- With DXT, Darshan additionally traces every read/write operation (for POSIX and MPI-IO interfaces)
- Enable by setting **`DXT_ENABLE_IO_TRACE`** env variable
- Finer grained instrumentation data comes at a cost of additional overhead and larger logs



Traces can be visualized using summary report heatmaps or other custom tools like DXT Explorer

Finer-grained details with Darshan: disabling shared file reductions

- To reduce log file size, globally shared file records are reduced into a single instrumentation record by default
  - However, this slightly masks per-rank contributions to I/O
- This behavior can be disabled by setting **DARSHAN_DISABLE_SHARED_REDUCTION** environment variable
- Allows for full accounting of per-rank contributions to shared files, if these details are important (e.g., for understanding collective I/O algorithms)

```
export DARSHAN_DISABLE_SHARED_REDUCTION=1
mpiexec -n 256 --ppn 64 ./helloworld $SCRATCHDIR
```


Argonne NATIONAL LABORATORY

Finer-grained details with Darshan: disabling shared file reductions

- To reduce log file size, globally shared file records are reduced into a single instrumentation record by default
  - However, this slightly masks per-rank contributions to I/O
- This behavior can be disabled by setting **DARSHAN_DISABLE_SHARED_REDUCTION** environment variable
- Allows for full accounting of per-rank contributions to shared files, if these details are important (e.g., for understanding collective I/O algorithms)

```
$ darshan-parser ./snyder_helloworld_id565659-63984_8-3-68717-7310522192037150959_1.dar
>    grep POSIX_BYTES_WRITTEN
POSIX   -1      1163774110118722858     POSIX_BYTES_WRITTEN     26214400000     /grand/
```

Rank -1 indicates a shared file record, with counters containing a reduced value access all ranks (e.g., ~24.5 GiB total bytes written across all ranks)

Argonne NATIONAL LABORATORY

Finer-grained details with Darshan: disabling shared file reductions

- To reduce log file size, globally shared file records are reduced into a single instrumentation record by default
  - However, this slightly masks per-rank contributions to I/O
- This behavior can be disabled by setting **DARSHAN_DISABLE_SHARED_REDUCTION** environment variable
- Allows for full accounting of per-rank contributions to shared files, if these details are important (e.g., for understanding collective I/O algorithms)

```
$ darshan-parser ./snyder_helloworld_id566419-55186_8-4-65788-3420894027405041227_1.dar
> grep POSIX_BYTES_WRITTEN
POSIX  0      1163774110118722858     POSIX_BYTES_WRITTEN     164626432       /grand/
POSIX  255    1163774110118722858     POSIX_BYTES_WRITTEN     0         /grand/projects
```

With shared reductions disabled, each rank retains their own record giving full insight into per-rank contributions (rank 0 writes 157 MiB and rank 255 writes nothing)

Argonne NATIONAL LABORATORY

Darshan runtime library configuration

- To bound memory overheads, Darshan imposes several internal memory limits (total memory usage, per-module record limits, etc.)
- For some workloads, default limits may be exceeded resulting in partial instrumentation data
- To offer user's more control over memory limits and instrumentation scope, Darshan provides a comprehensive runtime configuration system
  - Environment variables or config files

```
#   KEY                 VALUE              MODULES
NAME_EXCLUDE      ^/home                  *
NAME_EXCLUDE      .pyc$                   *
NAME_EXCLUDE      .so$                    *
NAME_INCLUDE      .h5$                    *

MODMEM      8

MAX_RECORDS       4000                    POSIX

MOD_ENABLE        DXT_POSIX,DXT_MPIIO

APP_EXCLUDE       git,ls,sed
```

Regular expressions can be specified to control whether matching record name patterns are included/excluded in Darshan instrumentation

Darshan runtime library configuration

- To bound memory overheads, Darshan imposes several internal memory limits (total memory usage, per-module record limits, etc.)
- For some workloads, default limits may be exceeded resulting in partial instrumentation data
- To offer user's more control over memory limits and instrumentation scope, Darshan provides a comprehensive runtime configuration system
  - Environment variables or config files

```
#   KEY                  VALUE              MODULES
NAME_EXCLUDE         ^/home                    *
NAME_EXCLUDE         .pyc$                     *
NAME_EXCLUDE         .so$                      *
NAME_INCLUDE         .h5$                      *

MODMEM         8

MAX_RECORDS          4000                  POSIX

MOD_ENABLE           DXT_POSIX,DXT_MPIIO

APP_EXCLUDE          git,ls,sed
```

Settings are also offered to control total per-process memory usage (8 MiB) and per-module maximum record counts (4000 POSIX records)

Darshan runtime library configuration

- To bound memory overheads, Darshan imposes several internal memory limits (total memory usage, per-module record limits, etc.)
- For some workloads, default limits may be exceeded resulting in partial instrumentation data
- To offer user's more control over memory limits and instrumentation scope, Darshan provides a comprehensive runtime configuration system
  - Environment variables or config files

```
#    KEY                    VALUE              MODULES
NAME_EXCLUDE        ^/home                        *
NAME_EXCLUDE        .pyc$                         *
NAME_EXCLUDE        .so$                          *
NAME_INCLUDE        .h5$                          *

MODMEM      8

MAX_RECORDS         4000                          POSIX

MOD_ENABLE          DXT_POSIX,DXT_MPIIO

APP_EXCLUDE         git,ls,sed
```

Additional settings allow control over enabled/disabled modules, as well as application names that should be included/excluded from instrumentation

Argonne NATIONAL LABORATORY

Thank you!

Bonus

Darshan-based analysis tools

Using Darshan as a starting point for developing new I/O analysis tools is attractive for a couple of reasons:

1. Darshan is commonly deployed in production at many HPC sites, making its I/O characterization data generally accessible to custom tools
2. Recent PyDarshan work has enabled much more agile development of Darshan-based I/O analysis tools in Python

We will start by considering a couple of Darshan-based I/O analysis tools: **DXT Explorer** and **Drishti**

DXT Explorer

- Darshan does not offer much in terms of DXT trace analysis tools beyond general I/O activity heatmaps
- **DXT Explorer★** is an interactive web-based trace analysis tool for DXT data that was developed to provide:
  - Combined views of MPI-IO and POSIX activity
  - Zoom in/out capabilities to focus on subsets of ranks or specific time slices
  - Contextual information about I/O calls
  - Views based on operation type, size, and spatiality
- Interactive trace analysis with DXT Explorer can enable interesting new insights into app I/O behavior

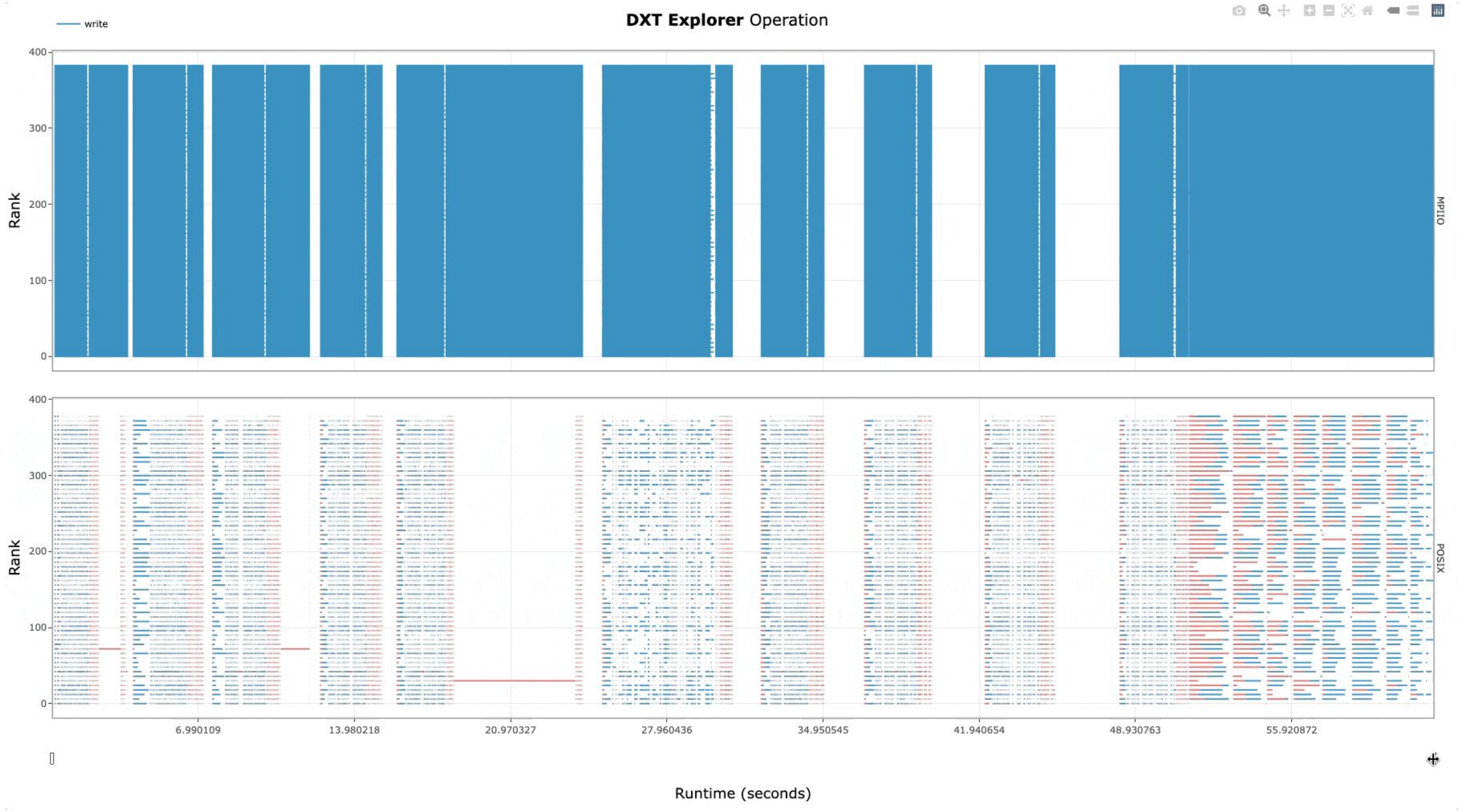
github.com/hpc-io/dxt-explorer

docker pull hpcio/dxt-explorer

★ DXT Explorer was developed by Jean Luca Bez (LBL). Slide content also provided courtesy of Jean Luca.

Bez, Jean Luca, et al. "I/O bottleneck detection and tuning: connecting the dots using interactive log analysis." *2021 IEEE/ACM Sixth International Parallel Data Systems Workshop (PDSW)*. IEEE, 2021.

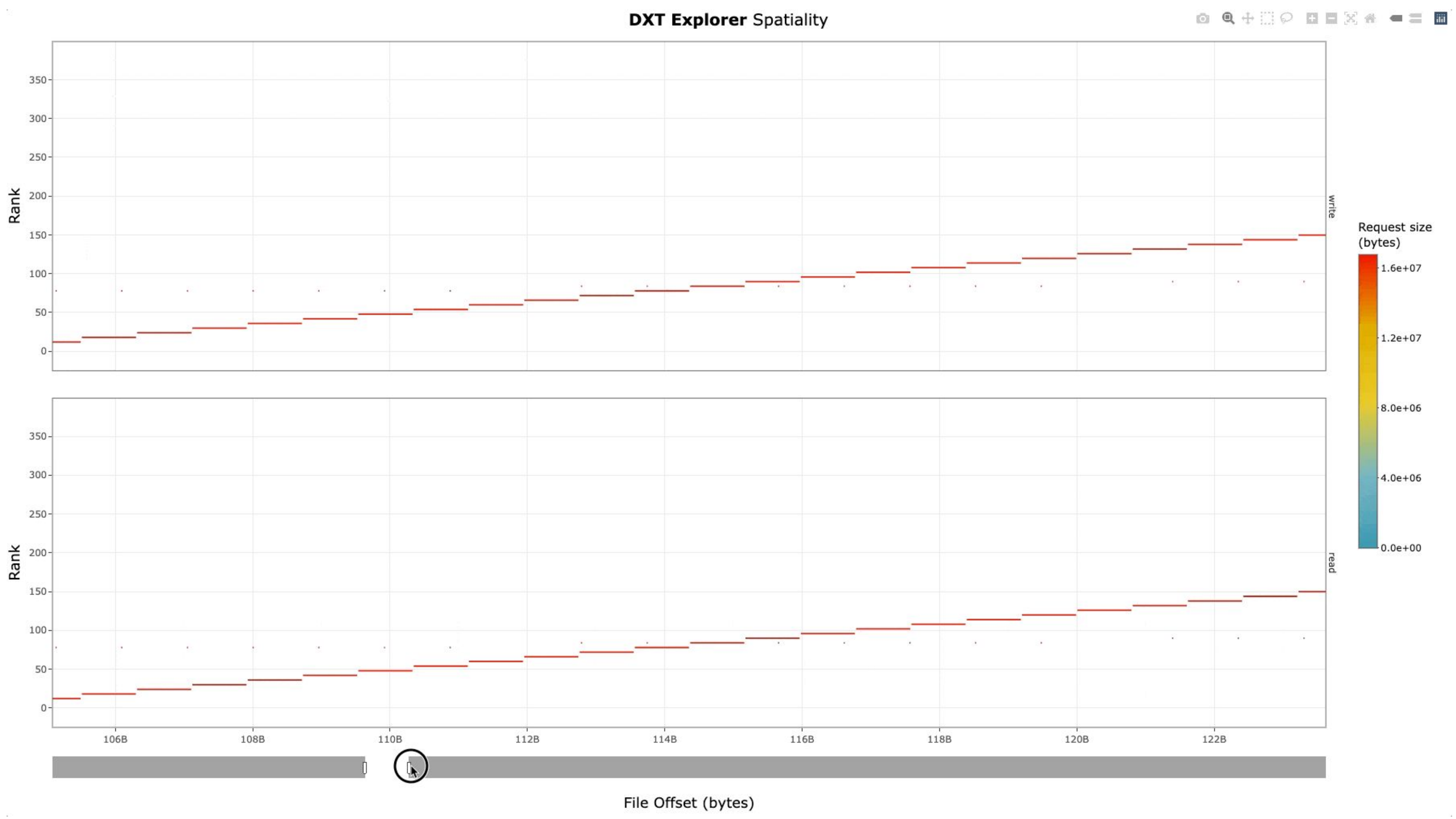
DXT Explorer



Explore the timeline by zooming in and out and observing how the MPI-IO calls are translated to the POSIX layer.

For instance, you can use this feature to detect stragglers.

DXT Explorer



Explore the spatiality of accesses in file by each rank with contextual information.

Understand how each rank is accessing each file.

Drishti

- Darshan can capture detailed I/O characterization data for an app, but translating this raw data to actionable tuning feedback is a significant challenge
- **Drishti★** is a command-line tool to guide end-users in optimizing I/O in their applications by detecting typical I/O performance pitfalls and providing a set of recommendations
- Drishti checks each given Darshan log against 30+ heuristic triggers for various I/O issues and suggests actions to take to resolve them
  - 4 levels of triggers: *high, warning, ok, info*

github.com/hpc-io/drishti-io

docker pull hpcio/drishti

★ Drishti was developed by Jean Luca Bez (LBL). Slide content also provided courtesy of Jean Luca.

Bez, Jean Luca, Hammad Ather, and Suren Byna. "Drishti: guiding end-users in the I/O optimization journey." 2022 IEEE/ACM International Parallel Data Systems Workshop (PDSW). IEEE, 2022.

Drishti

```
Drishti

┌─ DRISHTI v.0.3 ──────────────────────────────────────────────────────────────
│
│ JOB:            1190243
│ EXECUTABLE:     bin/8_benchmark_parallel
│ DARSHAN:        jlbez_8_benchmark_parallel_id1190243_7-23-45631-11755726114084236527_1.darshan
│ EXECUTION DATE: 2021-07-23 16:40:31+00:00 to 2021-07-23 16:40:32+00:00 (0.00 hours)
│ FILES:          6 files (1 use STDIO, 2 use POSIX, 1 use MPI-IO)
│ PROCESSES       64
│ HINTS:          romio_no_indep_rw=true cb_nodes=4
│
└─ 1 critical issues, 5 warnings, and 5 recommendations ──────────────────────

┌─ METADATA ───────────────────────────────────────────────────────────────────
│
│ ▶ Application is read operation intensive (6.34% writes vs. 93.66% reads)
│ ▶ Application might have redundant read traffic (more data was read than the highest read offset)
│ ▶ Application might have redundant write traffic (more data was written than the highest write offset)
│
└──────────────────────────────────────────────────────────────────────────────

┌─ OPERATIONS ─────────────────────────────────────────────────────────────────
│
│ ▶ Application issues a high number (285) of small read requests (i.e., < 1MB) which represents 37.11% of all
│ read/write requests
│   ↪ 284 (36.98%) small read requests are to "benchmark.h5"
│   ↪ Recommendations:
│     ↪ Consider buffering read operations into larger more contiguous ones
│     ↪ Since the appplication already uses MPI-IO, consider using collective I/O calls (e.g. MPI_File_read_all() or
│ MPI_File_read_at_all()) to aggregate requests into larger ones
```

Overall information about the Darshan log and execution

Number of critical issues, warning, and recommendations

Details on metadata and data operations

Critical issue and corresponding recommendation for `benchmark.h5`