ARGONNE ATPESC 2023 EXTREME-SCALE COMPUTING

Introduction to Darshan

How to learn more about the I/O behavior of your application

Shane Snyder
ssnyder@mcs.anl.gov
Argonne National Laboratory

August 10, 2023

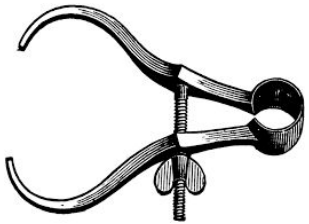
Understanding I/O problems in your application

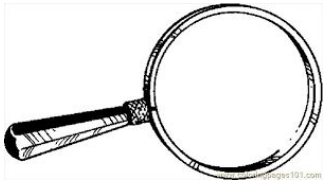
Example questions:

- ❑ 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.

This presentation is an introduction; we'll see more detailed Darshan examples later today.





Hands on exercises: https://github.com/radix-io/hands-on

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 ALCF, OLCF, and NERSC systems used for ATPESC training

★ 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

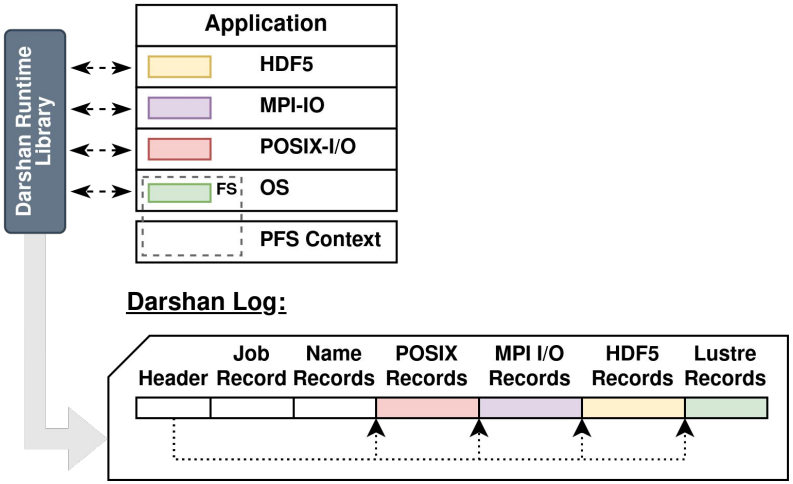


Figure courtesy Jakob Luettgau (UTK)

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 in the afternoon**…

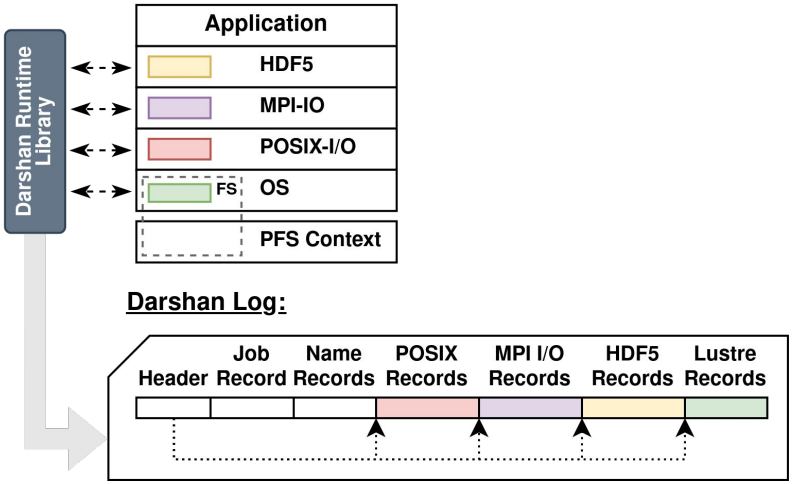


Figure courtesy Jakob Luettgau (UTK)

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

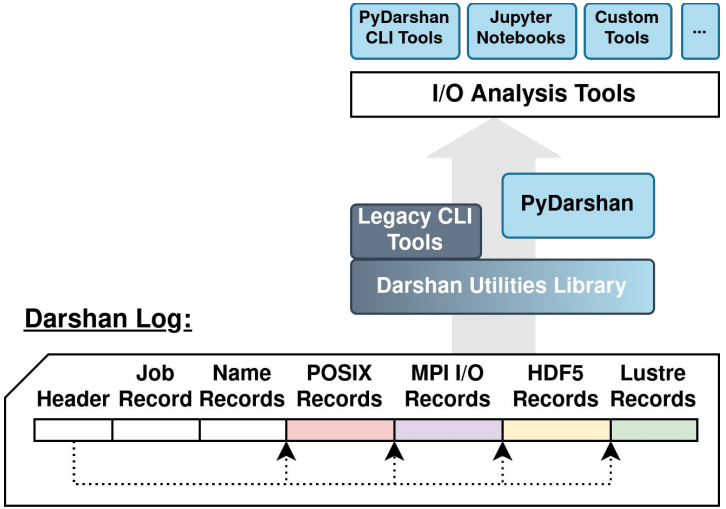


Figure courtesy Jakob Luettgau (UTK)

Hands on exercises: https://github.com/radix-io/hands-on

Using Darshan

- We'll use ALCF Polaris as an example in the following slides.
- The hands on exercises also include examples that are set up for use on Polaris.
  - https://github.com/radix-io/hands-on
- Other systems are very similar, though. 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 DOE systems after the Polaris example.

Using Darshan on Polaris: make sure the software is loaded

```
snyder@polaris-login-04:~> cd atpesc-io/hands-on/
snyder@polaris-login-04:~/atpesc-io/hands-on>
snyder@polaris-login-04:~/atpesc-io/hands-on> source polaris-setup-env.sh
```

The atpesc-io **hands-on** exercise repository 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!

Hands on exercises: https://github.com/radix-io/hands-on

Using Darshan on Polaris: make sure the software is loaded

```
snyder@polaris-login-04:~> cd atpesc-io/hands-on/
snyder@polaris-login-04:~/atpesc-io/hands-on>
snyder@polaris-login-04:~/atpesc-io/hands-on> source polaris-setup-env.sh

snyder@polaris-login-04:~/atpesc-io/hands-on> module list

Currently Loaded Modules:
  1) craype-x86-rome          11) cray-pals/1.1.7
  2) libfabric/1.11.0.4.125   12) cray-libpals/1.1.7
  3) craype-network-ofi       13) PrgEnv-nvhpc/8.3.3
  4) perftools-base/22.05.0   14) craype-accel-nvidia80
  5) nvhpc/21.9               15) cray-parallel-netcdf/1.12.2.3
  6) craype/2.7.15            16) cray-hdf5-parallel/1.12.1.3
  7) cray-dsmml/0.2.2         17) cray-netcdf-hdf5parallel/4.8.1.3
  8) cray-mpich/8.1.16        18) darshan/3.4.3
  9) cray-pmi/6.1.2           19) cray-python/3.9.12.1
 10) cray-pmi-lib/6.0.17
```

The atpesc-io **hands-on** exercise repository 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: make sure the software is loaded

```
snyder@polaris-login-04:~> cd atpesc-io/hands-on/
snyder@polaris-login-04:~/atpesc-io/hands-on>
snyder@polaris-login-04:~/atpesc-io/hands-on> source polaris-setup-env.sh

snyder@polaris-login-04:~/atpesc-io/hands-on> module list

Currently Loaded Modules:
  1) craype-x86-rome          11) cray-pals/1.1.7
  2) libfabric/1.11.0.4.125   12) cray-libpals/1.1.7
  3) craype-network-ofi       13) PrgEnv-nvhpc/8.3.3
  4) perftools-base/22.05.0   14) craype-accel-nvidia80
  5) nvhpc/21.9               15) cray-parallel-netcdf/1.12.2.3
  6) craype/2.7.15            16) cray-hdf5-parallel/1.12.1.3
  7) cray-dsmml/0.2.2         17) cray-netcdf-hdf5parallel/4.8.1.3
  8) cray-mpich/8.1.16        18) darshan/3.4.3
  9) cray-pmi/6.1.2           19) cray-python/3.9.12.1
 10) cray-pmi-lib/6.0.17
```

These steps are similar, and often cases easier, on other DOE platforms:

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

Check your site 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.*

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

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-01:~> darshan-config --log-path
/lus/grand/logs/darshan/polaris
snyder@polaris-login-01:~>
snyder@polaris-login-01:~> cd /lus/grand/logs/darshan/polaris/2023/8/3
```

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.

Using Darshan on Polaris: find your log file

```
snyder@polaris-login-01:~> darshan-config --log-path
/lus/grand/logs/darshan/polaris
snyder@polaris-login-01:~>
snyder@polaris-login-01:~> cd /lus/grand/logs/darshan/polaris/2023/8/3
snyder@polaris-login-01:/lus/grand/logs/darshan/polaris/2023/8/3> ls | grep snyder
snyder_helloworld_id565589-8006_8-3-67004-19711592040693242000_1.darshan
```

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

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

Hands on exercises: https://github.com/radix-io/hands-on

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     6960057185861764086     POSIX_BYTES_WRITTEN     5413869452      /projects/radix
POSIX   452     6960057185861764086     POSIX_BYTES_WRITTEN     5413865644      /projects/radix
POSIX   197     6960057185861764086     POSIX_BYTES_WRITTEN     5413857652      /projects/radix
POSIX   5       6960057185861764086     POSIX_BYTES_WRITTEN     5413852168      /projects/radix
POSIX   451     6960057185861764086     POSIX_BYTES_WRITTEN     5413844532      /projects/radix
POSIX   64      6960057185861764086     POSIX_BYTES_WRITTEN     5413823236      /projects/radix
POSIX   68      6960057185861764086     POSIX_BYTES_WRITTEN     5413788992      /projects/radix
POSIX   195     6960057185861764086     POSIX_BYTES_WRITTEN     5413663132      /projects/radix
POSIX   323     6960057185861764086     POSIX_BYTES_WRITTEN     5413658668      /projects/radix
POSIX   132     6960057185861764086     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 atpesc-io **hands-on** repository 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-01:~/atpesc-io/hands-on> source polaris-setup-env.sh
snyder@polaris-login-01:~/atpesc-io/hands-on> cd darshan/helloworld/
snyder@polaris-login-01:~/atpesc-io/hands-on/darshan/helloworld> 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
 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/atpesc-io/hands-on/darshan/helloworld/helloworld_report.html
```


Hands on exercises: https://github.com/radix-io/hands-on

Using Darshan on Polaris: generate summary report

The Polaris environment setup script in the atpesc-io **hands-on** repository 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-01:~/atpesc-io/hands-on> source polaris-setup-env.sh
snyder@polaris-login-01:~/atpesc-io/hands-on> cd darshan/helloworld/
snyder@polaris-login-01:~/atpesc-io/hands-on/darshan/helloworld> 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
 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/atpesc-io/hands-on/darshan/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.

Using Darshan on Polaris: generate summary report

The Polaris environment setup script in the atpesc-io **hands-on** repository 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-01:~/atpesc-io/hands-on> source polaris-setup-env.sh
snyder@polaris-login-01:~/atpesc-io/hands-on> cd darshan/helloworld/
snyder@polaris-login-01:~/atpesc-io/hands-on/darshan/helloworld> 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
 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/atpesc-io/hands-on/darshan/helloworld/helloworld_report.html
```

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

What about other 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

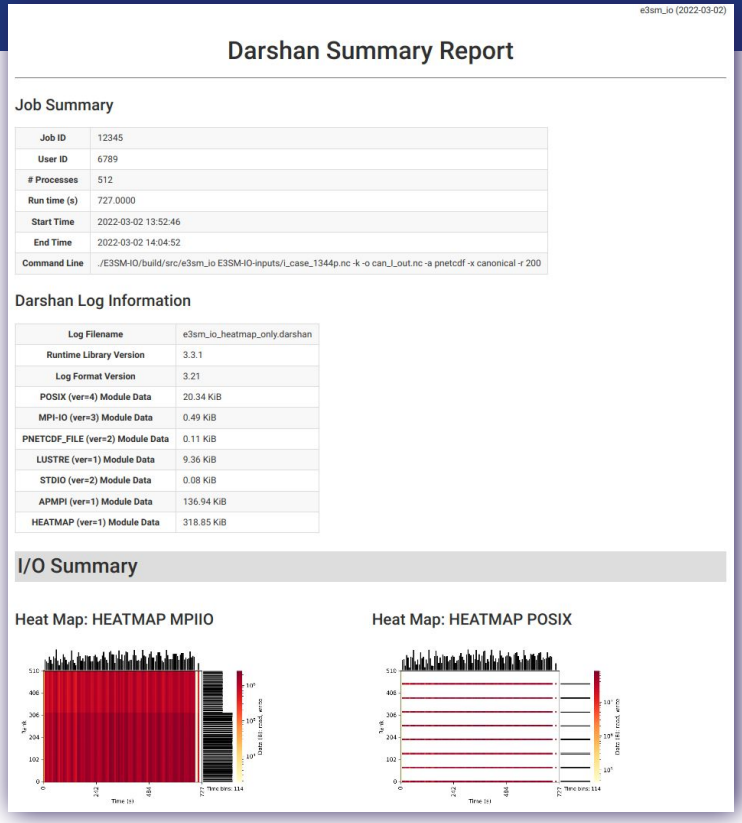
If Darshan is not available on a given system, it can either be installed via Spack or directly from source. Darshan is provided as 2 separate packages in Spack:

- **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/

Job analysis example



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

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

Hands on exercises: https://github.com/radix-io/hands-on

Job analysis: high-level job info

e3sm_io (2022-03-02)

Darshan Summary Report

Executable name and job date

Job Summary

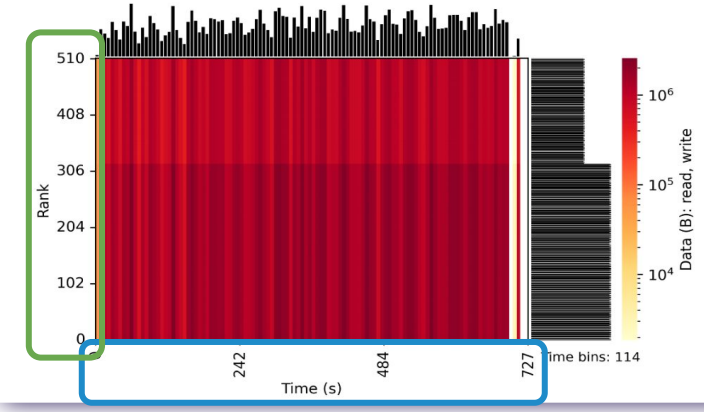
Detailed job metadata

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

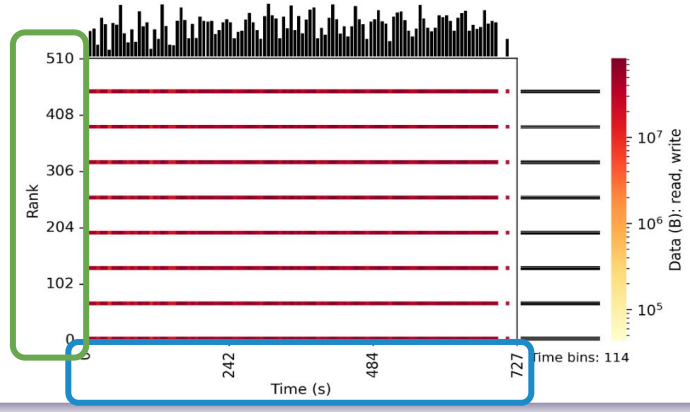
Job analysis: I/O heatmaps

I/O Summary

Heat Map: HEATMAP MPIIO

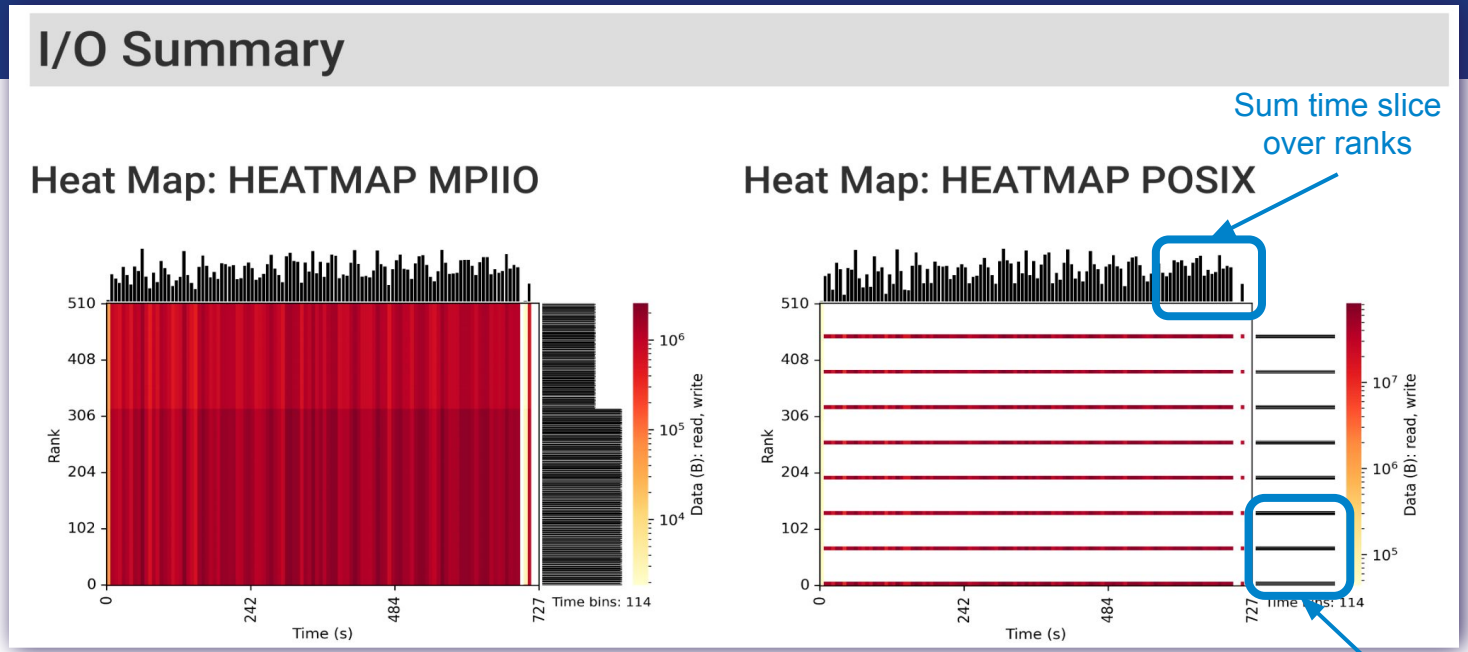


Heat Map: HEATMAP POSIX



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



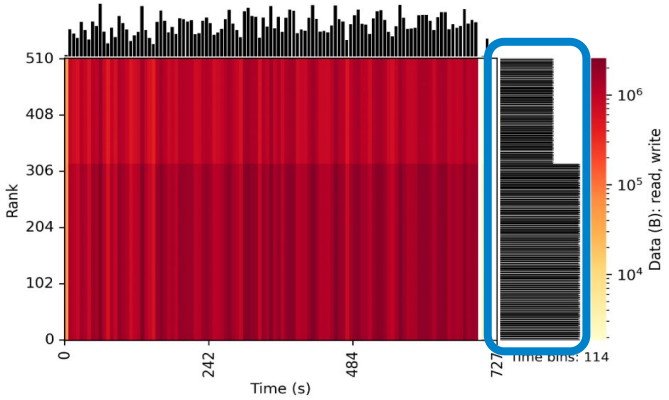
Heatmaps showcase application I/O intensity (r+w volume) across time, ranks, and interfaces – helpful for identifying hot spots, I/O and compute phases, etc.

Hands on exercises: https://github.com/radix-io/hands-on

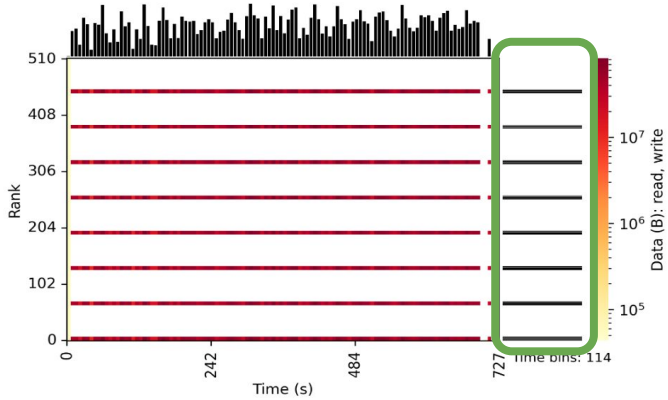
Job analysis: I/O heatmaps

I/O Summary

Heat Map: HEATMAP MPIIO



Heat Map: HEATMAP POSIX



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

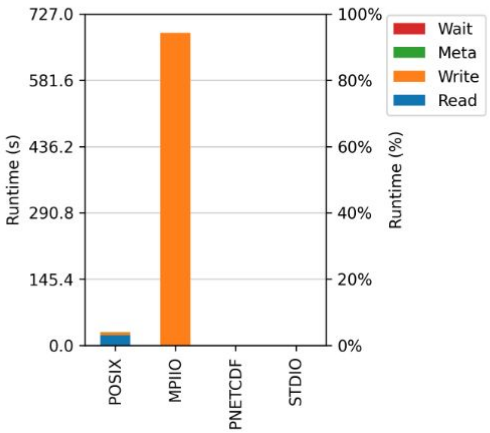
- **I/O imbalance across MPI processes**
- **Collective MPI-IO accesses transformed to subset of "aggregator" ranks at POSIX level**

Hands on exercises: https://github.com/radix-io/hands-on

Job analysis: I/O cost

Cross-Module Comparisons

I/O Cost

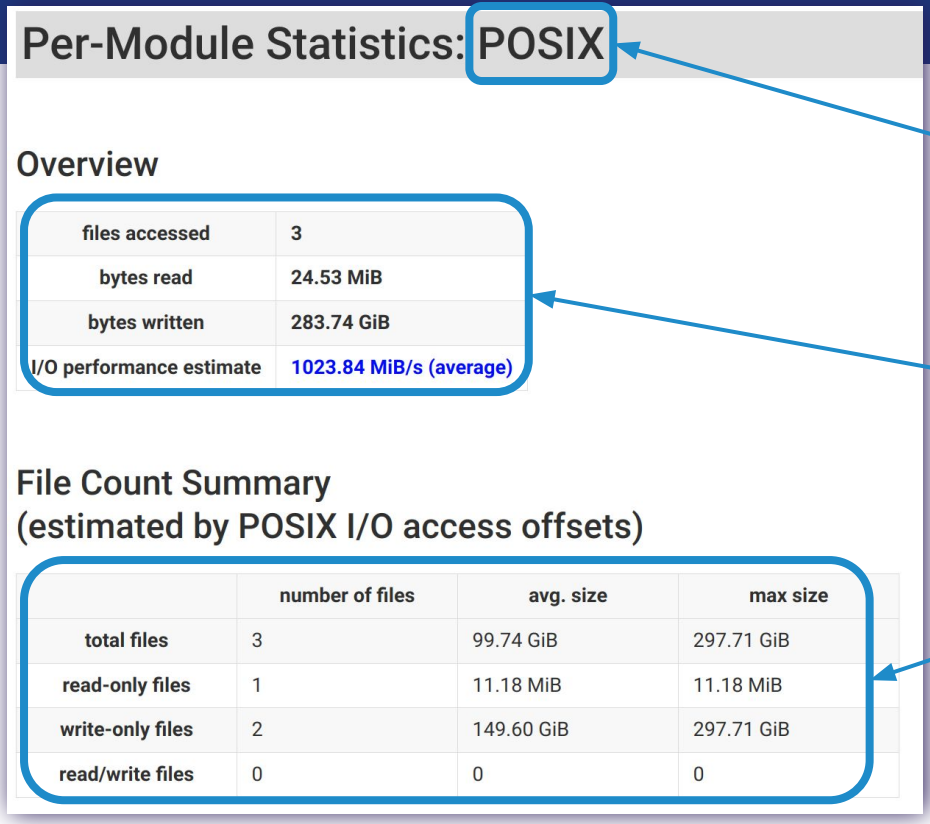


I/O cost indicates how much time on average was spent reading, writing, and doing metadata across different I/O interfaces

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

Hands on exercises: https://github.com/radix-io/hands-on

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

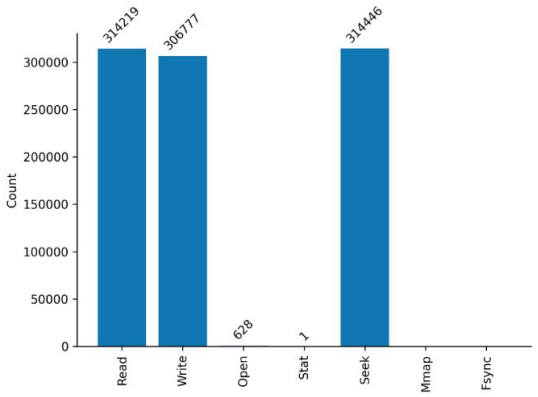
Aggregate stats for interface, as well as a **performance estimate**

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

Hands on exercises: https://github.com/radix-io/hands-on

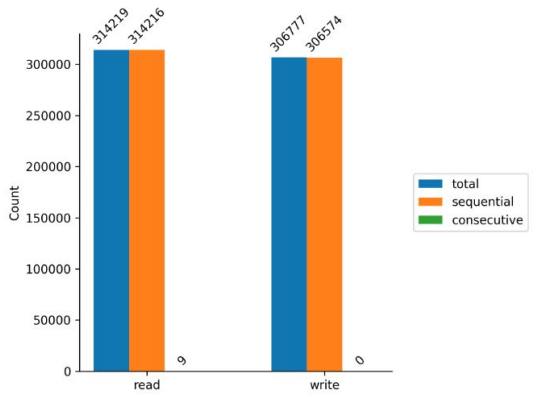
Job analysis: Per-interface statistics

Operation Counts



★ [sequential](): greater than previous offset
[consecutive](): immediately following previous offset

Access Pattern



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

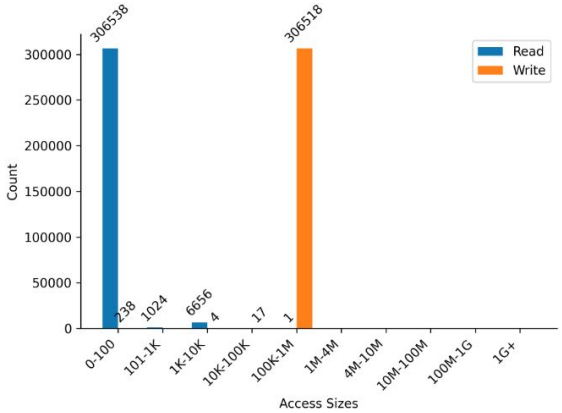
Lots of metadata operations (open, stat, seek, etc.) could be a sign of poorly performing I/O

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

Job analysis: Per-interface statistics

Access Sizes



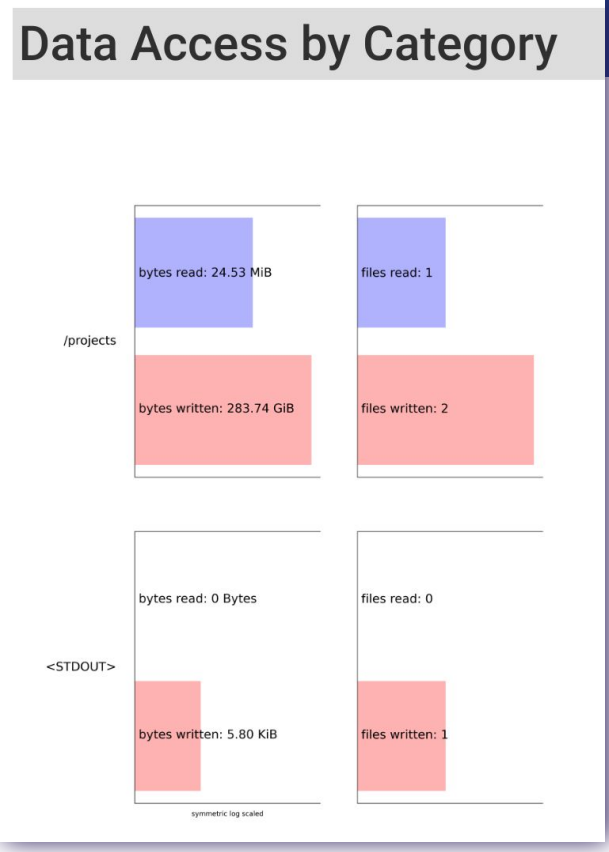
Common Access Sizes

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



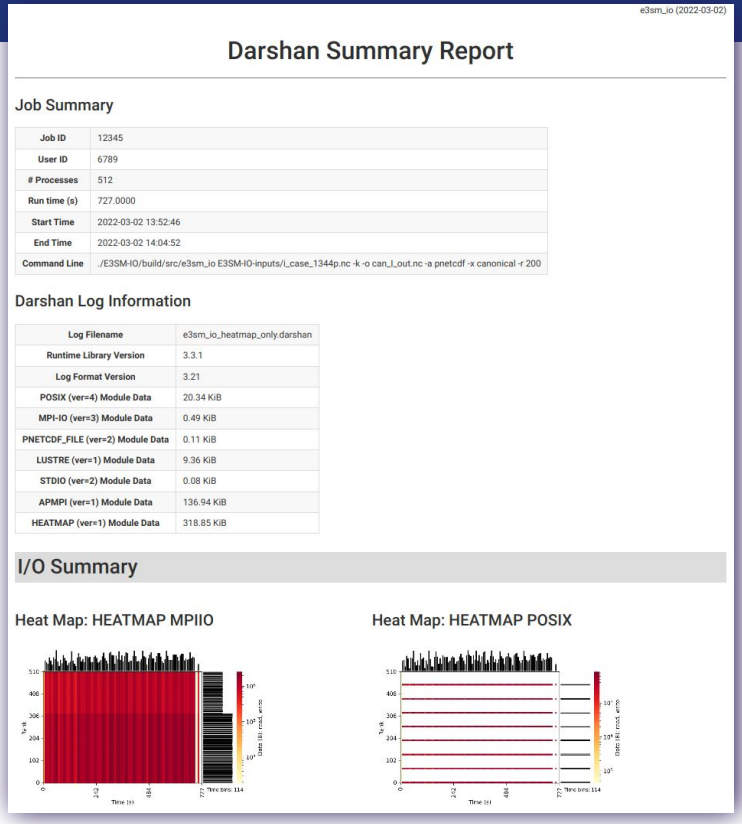
Data accesses, in terms of total files read/written and total bytes read/written, binned by different categories:

- 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

Hands on exercises: https://github.com/radix-io/hands-on

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 your site's support team 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.

Darshan: a recap

- These slides covered some basic usage and tips.
- Refer to facility documentation, support channels, or these slides when you need to.
- 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.
- We'll see additional Darshan use cases and features this afternoon.

Darshan hands on exercises

- The hands on exercises include 3 Darshan examples that you can try tonight or as time permits during the day:
  - **helloworld**: a simple application that you can run to test out the Darshan toolchain.
  - **warpdrive** and **fidgetspinner**: applications with A and B versions that you can compare to spot the performance differences (and their cause).

The warpdrive and fidgetspinner examples will be easier to understand after seeing the MPI-IO presentation later this morning.

Check with the instructors to share what you find!

Thank you!