ARGONNE
ATPESC2024
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 8, 2024

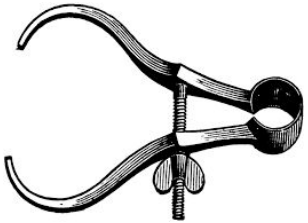
Argonne NATIONAL LABORATORY

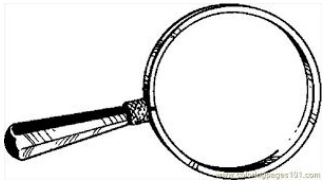
Understanding I/O problems in your application

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.





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

★ 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.

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

How does Darshan work?

Two primary components:

1. Darshan runtime library

- Instrumentation modules: lightweight wrappers intercept application I/O calls and record per-file statistics about access patterns
  - File records 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

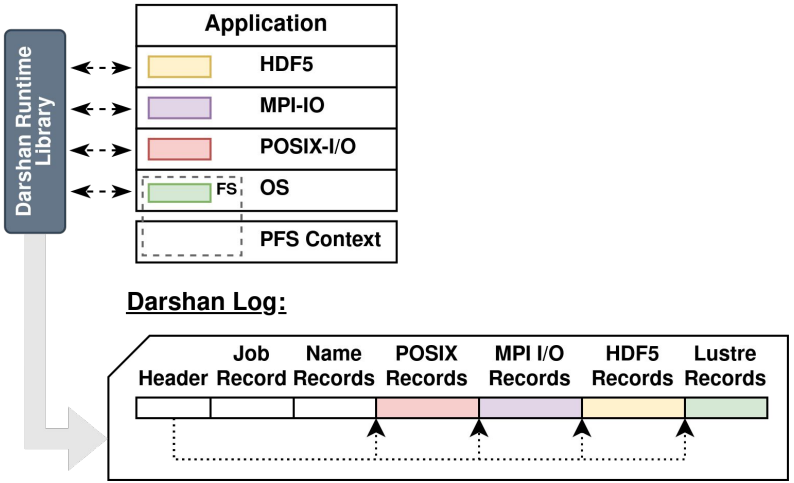


Figure courtesy Jakob Luettgau (Inria)

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

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.

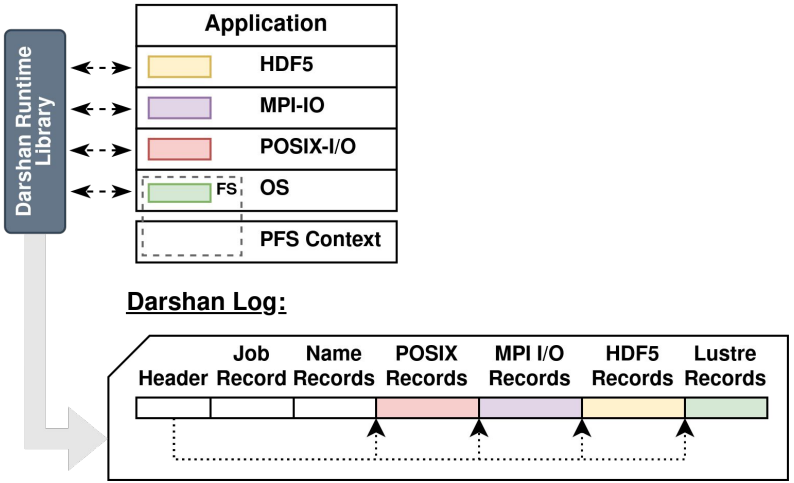


Figure courtesy Jakob Luettgau (Inria)

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

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
  - Python APIs for usage in custom tools, Jupyter notebooks, etc.
  - Legacy C-based tools/library

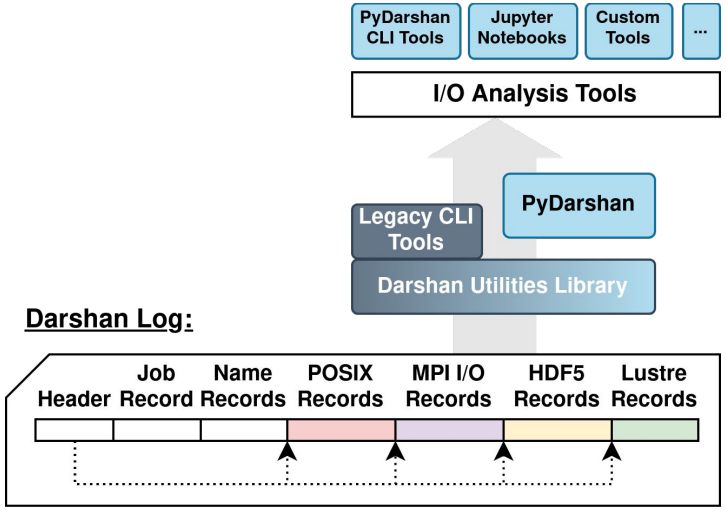


Figure courtesy Jakob Luettgau (Inria)

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

Darshan hands on exercise

- We'll start by collectively working through a hands on exercise demonstrating how to use the Darshan toolchain on ALCF Polaris.
- Usage on other systems is 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 this Polaris example.

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

Darshan hands on exercise: make sure the software is loaded

```
snyder@polaris-login-02:~> module list

Currently Loaded Modules:
  1) nvhpc/23.9             8) craype-x86-milan
  2) craype/2.7.30          9) PrgEnv-nvhpc/8.5.0
  3) cray-dsmml/0.2.2      10) libfabric/1.15.2.0
  4) cray-mpich/8.1.28     11) craype-network-ofi
  5) cray-pmi/6.1.13       12) perftools-base/23.12.0
  6) cray-pals/1.3.4       13) darshan/3.4.4
  7) cray-libpals/1.3.4
```

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

Darshan should already loaded by default. Darshan 3.4.4 is the current version on Polaris.

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

Darshan hands on exercise: make sure the software is loaded

```
snyder@polaris-login-02:~> module list

Currently Loaded Modules:
  1) nvhpc/23.9            8) craype-x86-milan
  2) craype/2.7.30         9) PrgEnv-nvhpc/8.5.0
  3) cray-dsmml/0.2.2     10) libfabric/1.15.2.0
  4) cray-mpich/8.1.28    11) craype-network-ofi
  5) cray-pmi/6.1.13      12) perftools-base/23.12.0
  6) cray-pals/1.3.4      13) darshan/3.4.4
  7) cray-libpals/1.3.4
```

```
snyder@polaris-login-02:~> module load darshan
```

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

Darshan should already loaded by default. Darshan 3.4.4 is the current version on Polaris.

If not, just run "**`module load darshan`**" to get it.

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

Darshan hands on exercise: build/instrument the helloworld example

- Setup a working directory and checkout the **hands-on** repo.

```
mkdir atpesc-io
cd atpesc-io
git clone https://github.com/radix-io/hands-on.git
cd hands-on
```

- Move to the **darshan/helloworld** example directory and build the example code.

```
cd darshan/helloworld
cc -o helloworld helloworld.c
```


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

Darshan hands on exercise: build/instrument the helloworld example

- Setup a working directory and checkout the **hands-on** repo.

```
mkdir atpesc-io
cd atpesc-io
git clone https://github.com/radix-io/hands-on.git
cd hands-on
```

- Move to the **darshan/helloworld** example directory and build the example code.

```
cd darshan/helloworld
cc -o helloworld helloworld.c
```

With the Darshan software module loaded, Polaris compilers will automatically link in the Darshan library. **No other user intervention is required!**

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

Darshan hands on exercise: build/instrument the helloworld example

- Setup a working directory and checkout the **hands-on** repo.

```
mkdir atpesc-io
cd atpesc-io
git clone https://github.com/radix-io/hands-on.git
cd hands-on
```

- Move to the **darshan/helloworld** example directory and build the example code.

```
cd darshan/helloworld
cc -o helloworld helloworld.c
```

With the Darshan software module loaded, Polaris compilers will automatically link in the Darshan library. **No other user intervention is required!** ★

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

Darshan hands on exercise: run the job

- Submit the **helloworld** job script to the scheduler.

```
qsub helloworld.qsub
```

- Use the **`qstat -u <username>`** tool to help determine when your job is complete. (If no qstat output, there are no queued/running jobs).

```
snyder@polaris-login-02:~/atpesc-io/hands-on/darshan/helloworld> qstat -u snyder

polaris-pbs-01.hsn.cm.polaris.alcf.anl.gov:
                                                                  Req'd  Req'd   Elap
Job ID                  Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
----------------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
2051077.polaris-pbs* snyder   R2035676 helloworl*    --    4 256    --  00:1[ Q   --
```


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

Darshan hands on exercise: 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-02:~> darshan-config --log-path
/lus/grand/logs/darshan/polaris
snyder@polaris-login-02:~>
snyder@polaris-login-02:~> ls /lus/grand/logs/darshan/polaris/2024/7/16/ | grep snyder
```

Check the subdirectory for the year / month / day your job executed.

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

Darshan hands on exercise: 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-02:~> darshan-config --log-path
/lus/grand/logs/darshan/polaris
snyder@polaris-login-02:~>
snyder@polaris-login-02:~> ls /lus/grand/logs/darshan/polaris/2024/7/16/ | grep snyder
snyder_helloworld_id2022038-3465522_7-16-66592-17487606729000876424_1.darshan
```

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

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

Darshan hands on exercise: generate job summary report

The **hands-on** repo includes a Polaris environment setup script that 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-02:~/atpesc-io/hands-on> source polaris-setup-env.sh
snyder@polaris-login-02:~/atpesc-io/hands-on>
snyder@polaris-login-02:~/atpesc-io/hands-on> python -m darshan summary /lus/grand/logs/darshan/
polaris/2024/7/16/snyder_helloworld_id2022038-3465522_7-16-66592-17487606729000876424_1.darshan
Report generated successfully.
Saving report at location: /home/snyder/atpesc-io/hands-on/snyder_helloworld_id2022038-3465522_7
-16-66592-17487606729000876424_1_report.html
```


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

Darshan hands on exercise: generate job summary report

The **hands-on** repo includes a Polaris environment setup script that 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-02:~/atpesc-io/hands-on> source polaris-setup-env.sh
snyder@polaris-login-02:~/atpesc-io/hands-on>
snyder@polaris-login-02:~/atpesc-io/hands-on> python -m darshan summary /lus/grand/logs/darshan/
polaris/2024/7/16/snyder_helloworld_id2022038-3465522_7-16-66592-17487606729000876424_1.darshan
Report generated successfully.
Saving report at location: /home/snyder/atpesc-io/hands-on/snyder_helloworld_id2022038-3465522_7
-16-66592-17487606729000876424_1_report.html
```

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

Darshan hands on exercise: analyze job summary report in a browser

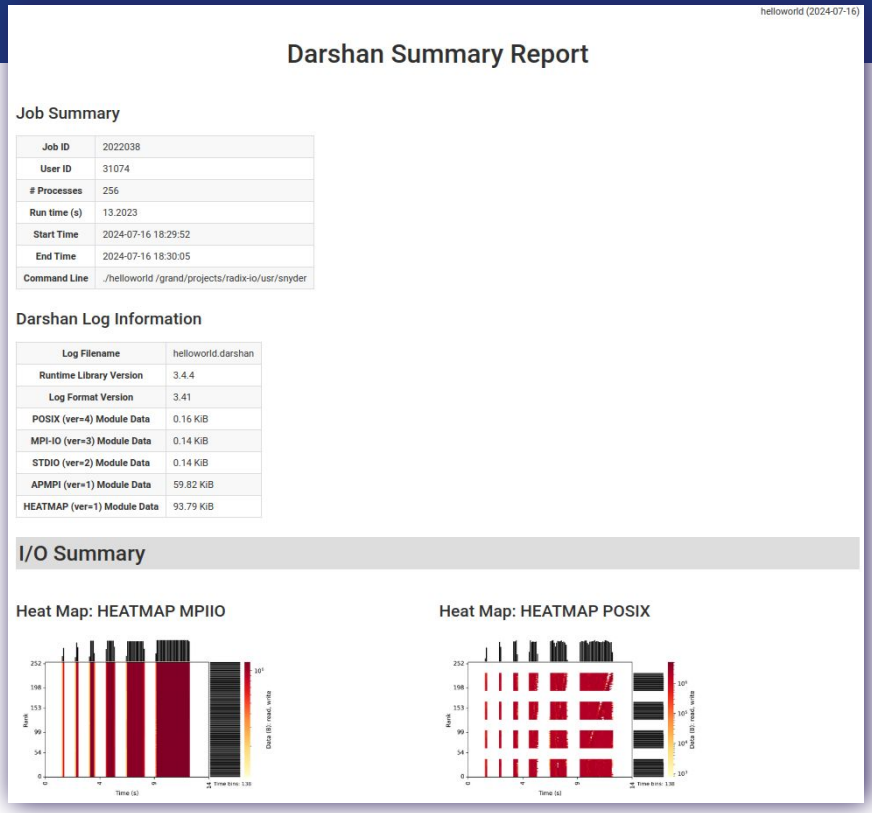
- First, use **`scp`** to copy the log to your own personal system.

```
scp snyder@polaris.alcf.anl.gov:/path/to/report.html \
    /local/path/to/report.html
```

- Next, open up the HTML report with your browser of choice and scan through the information it provides.

We will pause here to give everyone some time to catch up before moving onto interpreting the job summary report results.
Reach out to an instructor if you need help!

Job summary report example



helloworld (2024-07-16)

Darshan Summary Report

Job Summary

Job ID	2022038
User ID	31074
# Processes	256
Run time (s)	13.2023
Start Time	2024-07-16 18:29:52
End Time	2024-07-16 18:30:05
Command Line	./helloworld /grand/projects/radix-io/usr/snyder

Darshan Log Information

Log Filename	helloworld.darshan
Runtime Library Version	3.4.4
Log Format Version	3.41
POSIX (ver=4) Module Data	0.16 KiB
MPI-IO (ver=3) Module Data	0.14 KiB
STDIO (ver=2) Module Data	0.14 KiB
APMPI (ver=1) Module Data	59.82 KiB
HEATMAP (ver=1) Module Data	93.79 KiB

I/O Summary

Heat Map: HEATMAP MPIIO

Heat Map: HEATMAP POSIX

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

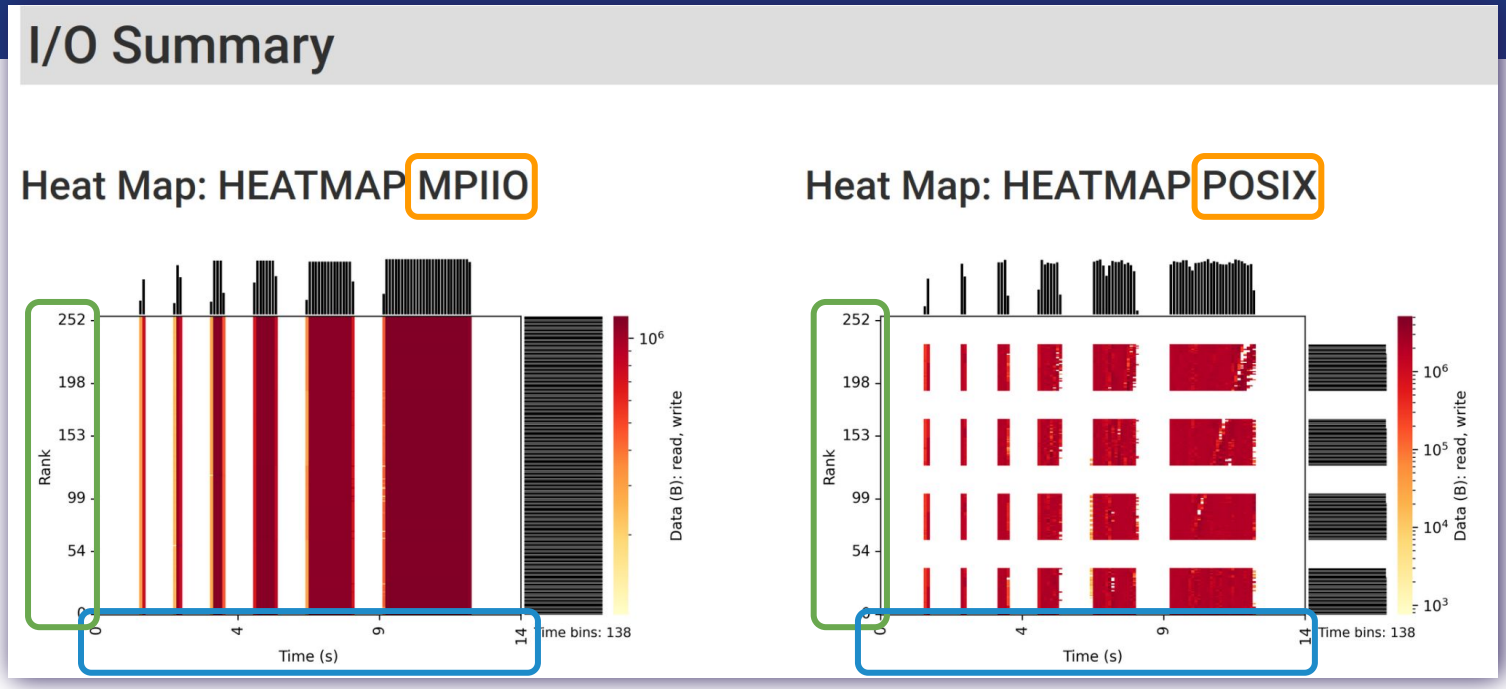
We will summarize some of the highlights from the **helloworld** example in the following slides.

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

Job summary report: high-level job info

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

Job summary report: I/O heatmaps



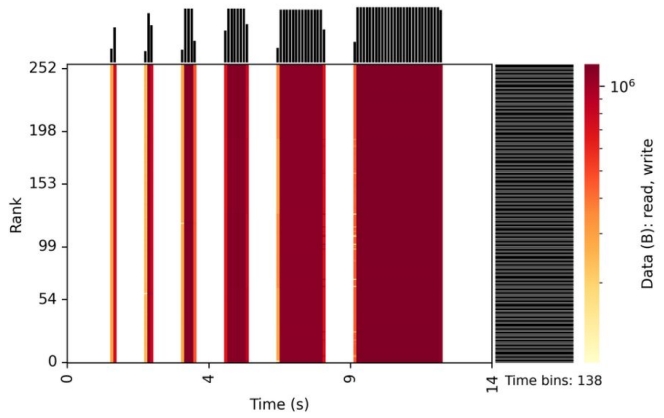
Heatmaps showcase application I/O intensity (read+write volume) across **time**, **ranks**, and **interfaces** – helpful for identifying hot spots, I/O and compute phases, etc.

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

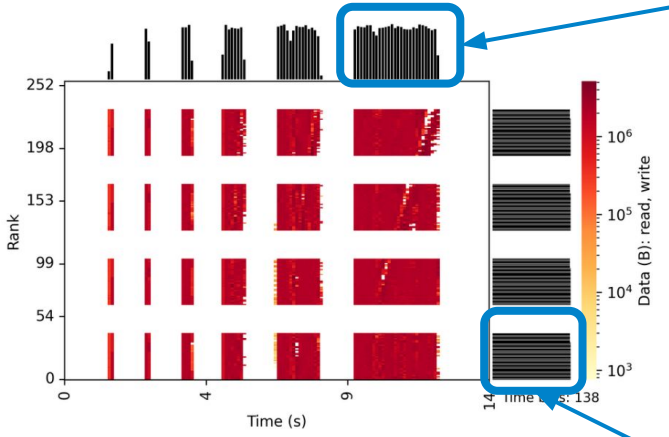
Job summary report: I/O heatmaps

I/O Summary

Heat Map: HEATMAP MPIIO



Heat Map: HEATMAP POSIX



Sum time slice over ranks

Sum rank over time slices

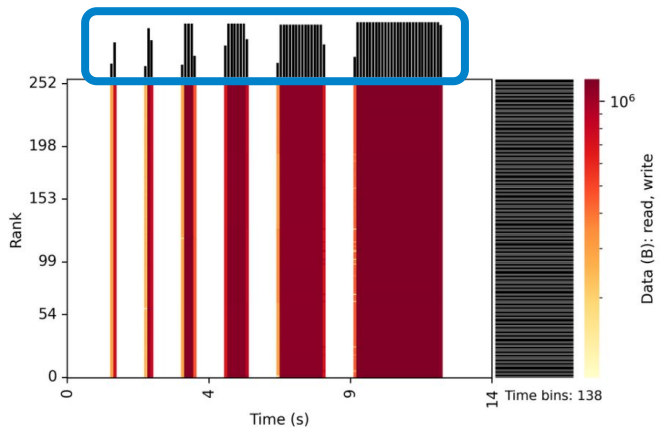
Heatmaps showcase application I/O intensity (read+write volume) across time, ranks, and interfaces – helpful for identifying hot spots, I/O and compute phases, etc.

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

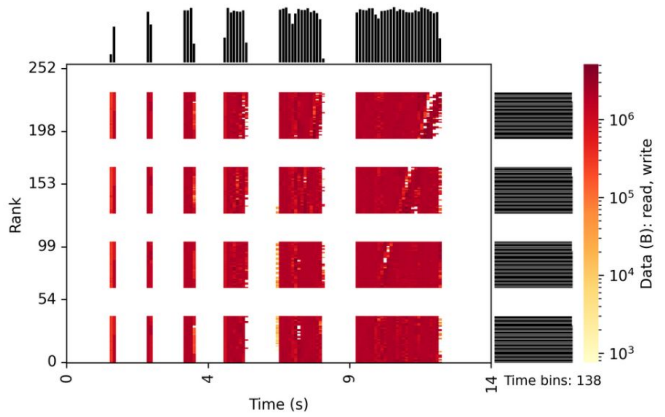
Job summary report: I/O heatmaps

I/O Summary

Heat Map: HEATMAP MPIIO



Heat Map: HEATMAP POSIX



This application demonstrates some notable I/O characteristics:

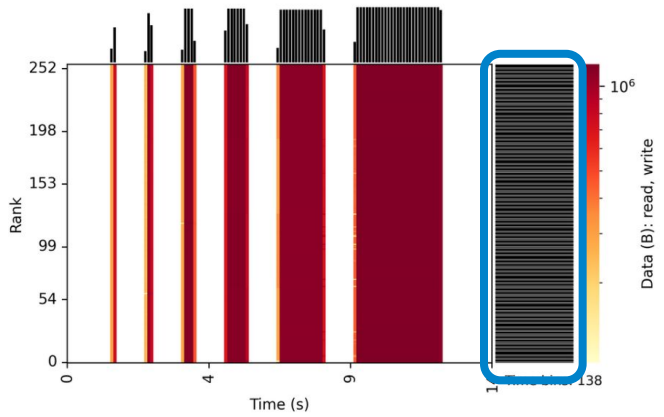
- Application I/O phases demonstrating increasing I/O intensity over time

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

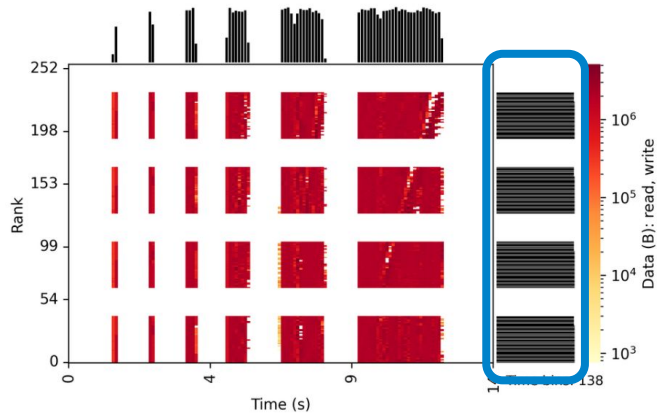
Job summary report: I/O heatmaps

I/O Summary

Heat Map: HEATMAP MPIIO



Heat Map: HEATMAP POSIX



This application demonstrates some notable I/O characteristics:

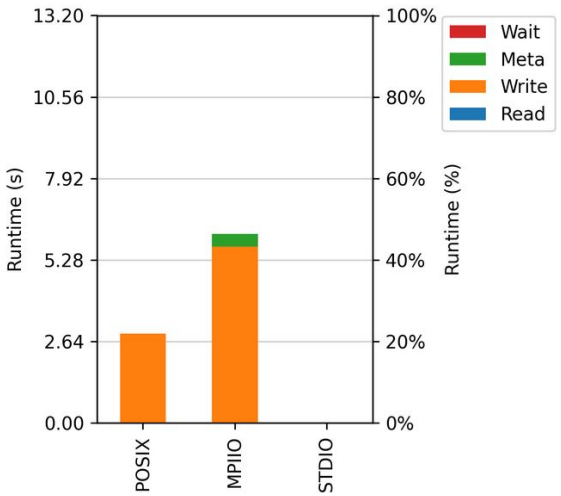
- Balanced, collective MPI-IO accesses transformed to subset of POSIX “aggregators”

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

Job summary report: I/O cost

Cross-Module Comparisons

I/O Cost

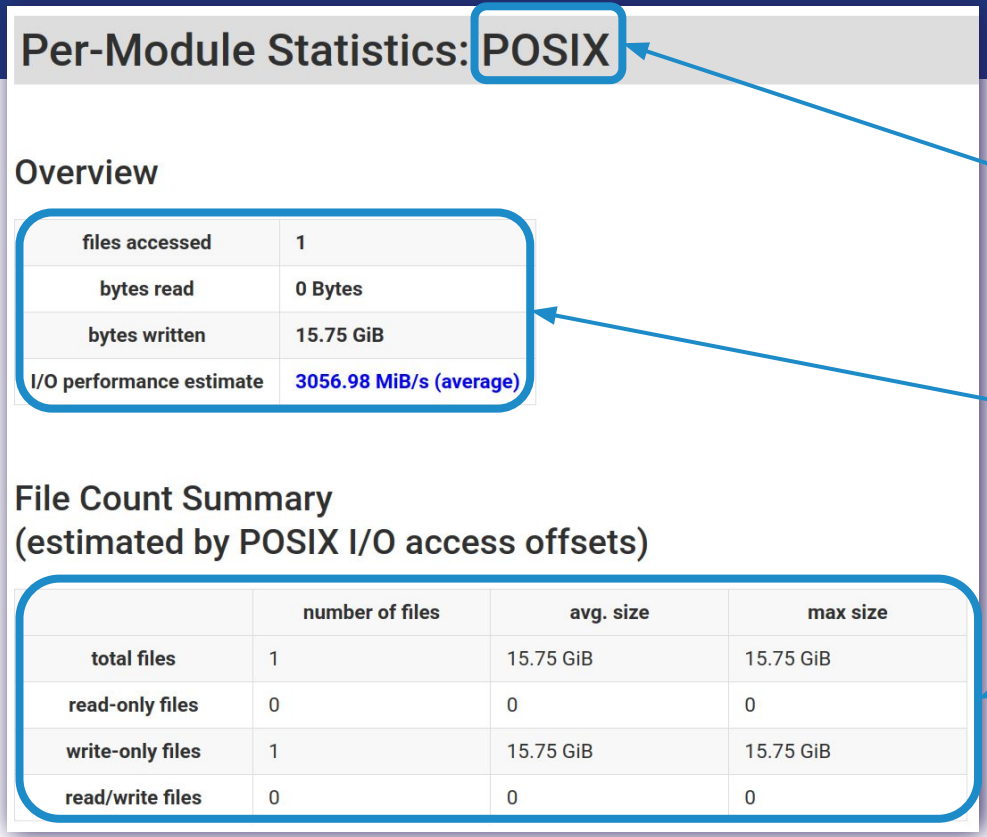


I/O cost indicates how much time on average was spent reading, writing, and doing metadata across different I/O interfaces.

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 summary report: Per-interface statistics



Per-Module Statistics: POSIX

Overview

files accessed	1
bytes read	0 Bytes
bytes written	15.75 GiB
I/O performance estimate	3056.98 MiB/s (average)

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

	number of files	avg. size	max size
total files	1	15.75 GiB	15.75 GiB
read-only files	0	0	0
write-only files	1	15.75 GiB	15.75 GiB
read/write files	0	0	0

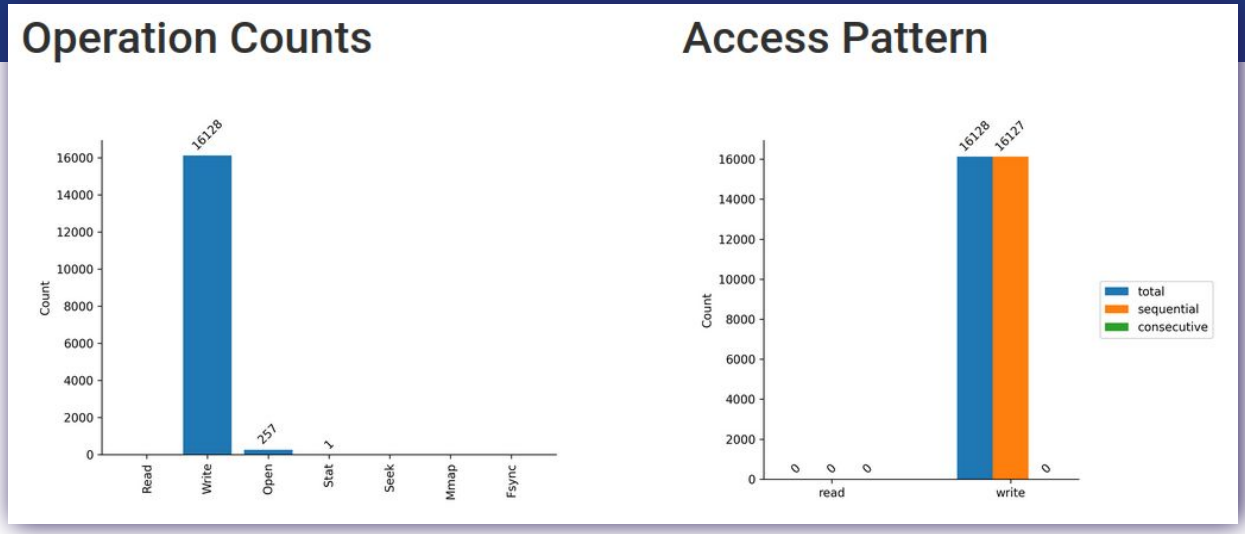
Stats available for various I/O APIs: POSIX, MPI-IO, STDIO, HDF5, PnetCDF

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 summary report: 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.

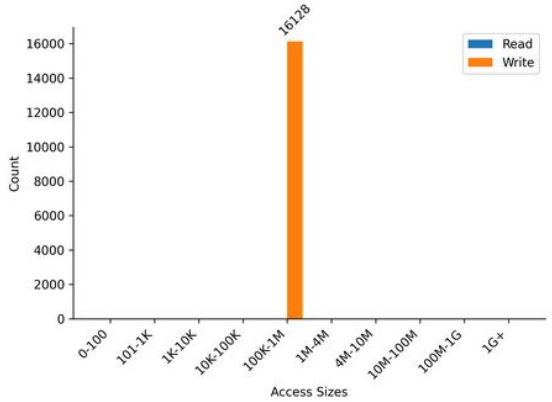
Access pattern indicates whether read/write operations progress sequentially or consecutively★ through the file.

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

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

Job summary report: Per-interface statistics

Access Sizes



Common Access Sizes

Access Size	Count
1048576	16128

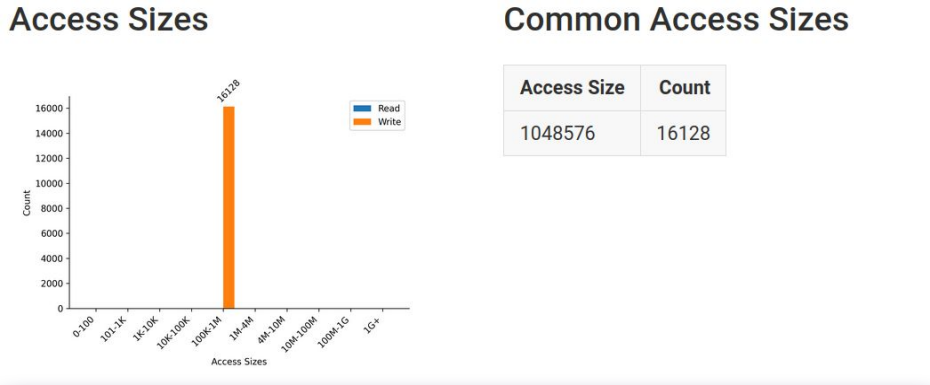
Details on access sizes are provided to better understand granularity of application read/write accesses.

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

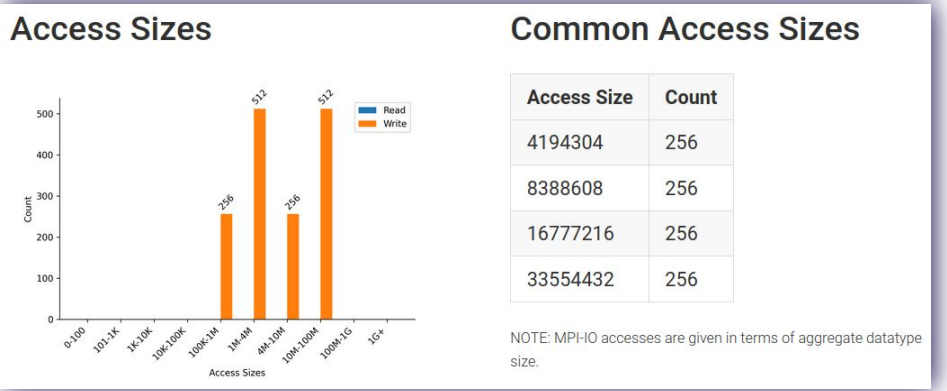
Hands on exercises:
https://github.com/radix-io/hands-on

Job summary report: Per-interface statistics

POSIX



MPI-IO

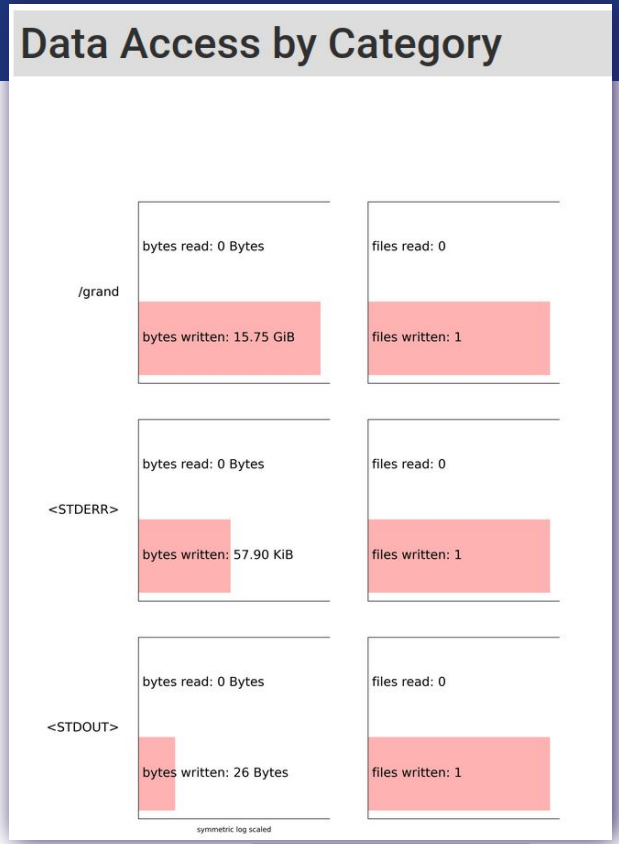


Note that the file access pattern issued by the application (i.e., using MPI-IO) can vary drastically from what is ultimately issued to the file system (i.e., using POSIX).

This application increases its access size each I/O phase, yet only 1 MiB operations are ever issued to the file system via POSIX calls.

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

Job summary report: Data access by category



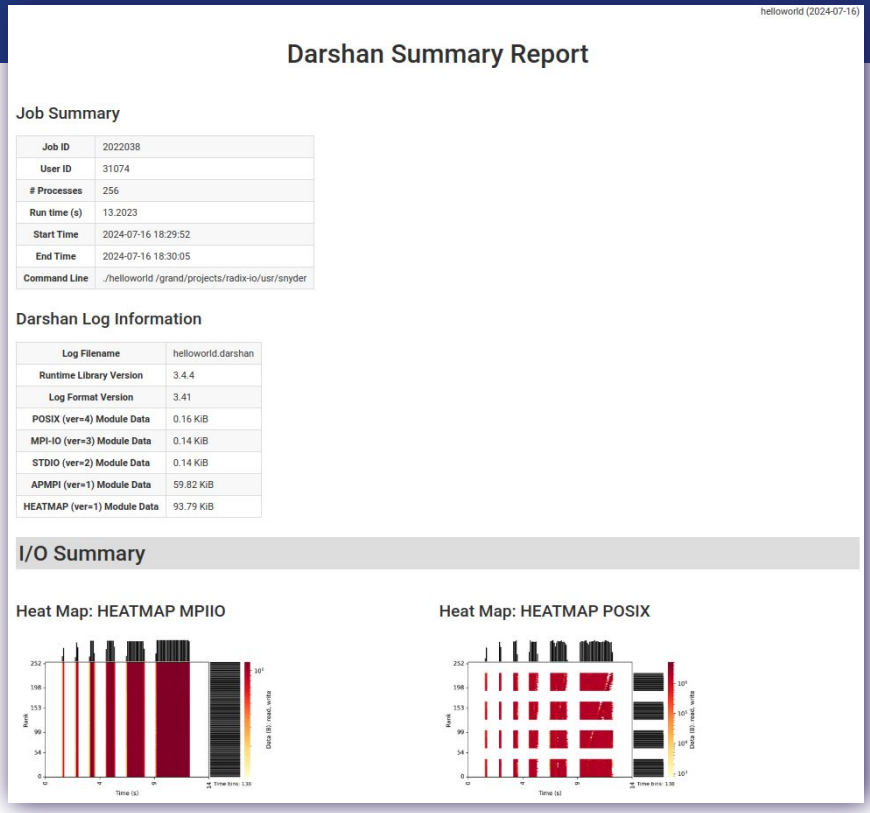
Data accesses, in terms of total files read/written and total bytes read/written, binned by different categories:

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

Informs on job's general usage of different storage resources.

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

Job summary report: additional help



helloworld (2024-07-16)

Darshan Summary Report

Job Summary

Job ID	2022038
User ID	31074
# Processes	256
Run time (s)	13.2023
Start Time	2024-07-16 18:29:52
End Time	2024-07-16 18:30:05
Command Line	./helloworld /grand/projects/radix-io/usr/snyder

Darshan Log Information

Log Filename	helloworld.darshan
Runtime Library Version	3.4.4
Log Format Version	3.41
POSIX (ver=4) Module Data	0.16 KiB
MPI-IO (ver=3) Module Data	0.14 KiB
STDIO (ver=2) Module Data	0.14 KiB
APMPI (ver=1) Module Data	59.82 KiB
HEATMAP (ver=1) Module Data	93.79 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.

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

What about using Darshan on 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 not available on a system, Darshan can either be installed via Spack or from source. It is provided as two separate packages in Spack:

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

Note that admin privileges are **not** required for installing/using Darshan tools on a system.

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/

Darshan: a recap

- These slides covered some basic Darshan 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 more Darshan use cases and features this afternoon.

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

More Darshan hands on exercises

- The hands-on repo contains additional Darshan examples (**warpdrive** and **fidgetspinner**) that you can try as time permits:
  - Each example has A and B versions that you can compare to spot the performance differences (and their cause).
  - These examples will be easier to understand after seeing this afternoon's MPI-IO presentation.
- We encourage you to try these exercises out and to check with the instructors to share what you find!
- **We also encourage you to try Darshan with your own applications to see what sorts of insights it can provide!**

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

Thank you!