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

ATPESC 2022

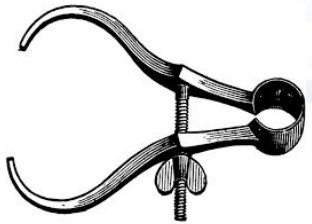
Shane Snyder
ssnyder@mcs.anl.gov
Mathematics and Computer Science Division
Argonne National Laboratory

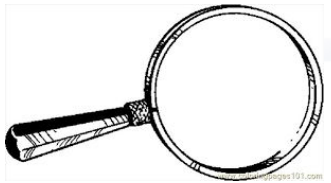
August 5, 2022

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 the same as you scale up?
- Does it get better, worse, or 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.

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

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 histograms, timers, counters, etc.

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

How does Darshan work?

Darshan is primarily intended for MPI applications.* It inserts lightweight instrumentation when your program is compiled or executed.

- Intercepts I/O calls and records statistics about file accesses
  - File records are stored in bounded, compact memory at each rank
- Aggregates statistics when the application exits
  - Collect, filter, compress records and write a single summary file for the job
- Provides command line tools to inspect and interpret statistics
  - Usually start by generating a summary PDF that plots metrics of interest

* You can also instrument non-MPI applications; we'll cover this later in the afternoon.

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

Using Darshan

- We'll use Theta as an example in the following slides.
- The hands on exercises also include examples that are set up for use on Theta.
  - 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 Theta example

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

Using Darshan on Theta: make sure the software is loaded

These steps are similar on other platforms; check your site documentation!

```
snyder@thetalogin6:~> module list |& tail -n 10
 15) rca/2.2.20-7.0.2.1_2.87__g8e3fb5b.ari
 16) atp/3.6.4
 17) perftools-base/20.06.0
 18) PrgEnv-intel/6.0.7
 19) craype-mic-knl
 20) cray-mpich/7.7.14
 21) nompirun/nompirun
 22) adaptive-routing-a3
 23) darshan/3.3.0
 24) xalt
```

Use “**module list**” to see a list of software loaded in your environment.

Darshan is probably already loaded by default. Darshan 3.3.0 is the current version on Theta

```
snyder@thetalogin6:~>
snyder@thetalogin6:~> module load darshan
```

If not, just run “**module load darshan**” to get it.

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

Using Darshan on Theta: instrument your code

Compile and run your application!

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_Initialize() and MPI_Finalize() to generate a log.

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

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

Using Darshan on Theta: find your log file

All Darshan logs are placed in a central location. On newer Darshan versions, '`darshan-config --log-path`' command will provide the log directory location. **Otherwise, check your site documentation!**

```
snyder@thetalogin6:~> darshan-config --log-path
/lus/theta-fs0/logs/darshan/theta
snyder@thetalogin6:~> cd /lus/theta-fs0/logs/darshan/theta/2022/7/22
snyder@thetalogin6:/lus/theta-fs0/logs/darshan/theta/2022/7/22> ls | grep snyder
snyder_fidgetspinnerA_id611125_7-22-73770-15971764111489070954_1.darshan
snyder_helloworld_id611104_7-22-66844-15971764111489070954_1.darshan
snyder_warpdriveB_id611115_7-22-70100-2565962767823139646_1.darshan
snyder@thetalogin6:/lus/theta-fs0/logs/darshan/theta/2022/7/22>
snyder@thetalogin6:/lus/theta-fs0/logs/darshan/theta/2022/7/22> cp snyder_* ~/tmp/
snyder@thetalogin6:/lus/theta-fs0/logs/darshan/theta/2022/7/22>
```

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

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

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

For convenience, copy your logs somewhere to analyze.

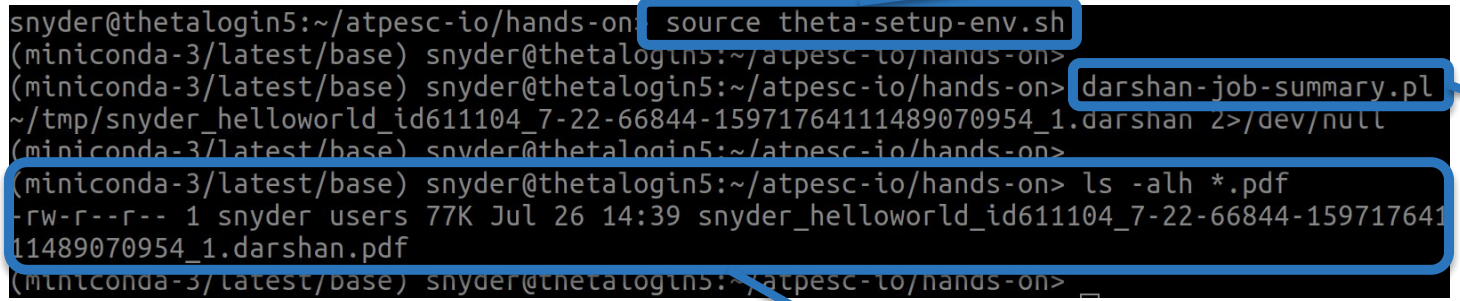
Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

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

Using Darshan on Theta: generate summary

At this point you could scp the log to another system to analyze (the logs are portable). This example shows how to generate a summary using tools on Theta.

The atpesc-io hands-on exercise repository includes a script to configure your environment with the tools needed for Darshan analysis.



Process your log with darshan-job-summary.pl.

It produces a PDF file that (by default) has the same name as your original log, plus a .pdf extension.

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

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

What about other systems?

- Cori:
  - How to enable: automatic
  - Log directory: /global/cscratch1/sd/darshanlogs/
- Summit:
  - How to enable: automatic
  - Log directory: /gpfs/alpine/darshan/summit

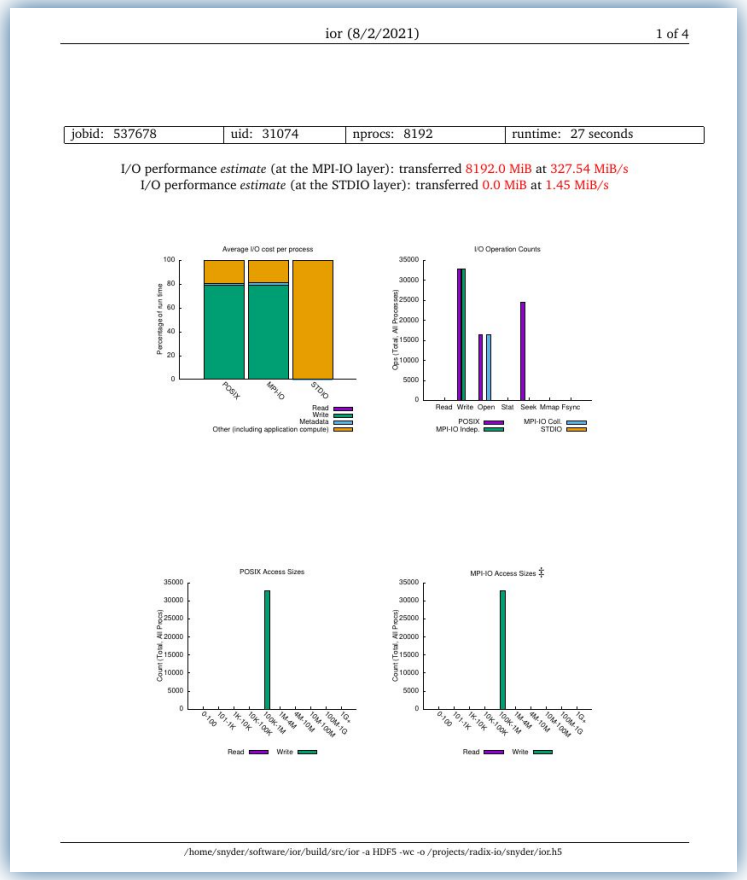
On each of these systems, you can use darshan-parser to inspect logs in text format, but you will need to copy the logs to another system to generate the pdf summary. Install the “darshan-util” package from Spack or the darshan-util portion of the Darshan source from:

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

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

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

Job analysis example



`darshan-job-summary` tool generates a multi-page PDF containing graphs, tables, and performance estimates characterizing the I/O workload of the application

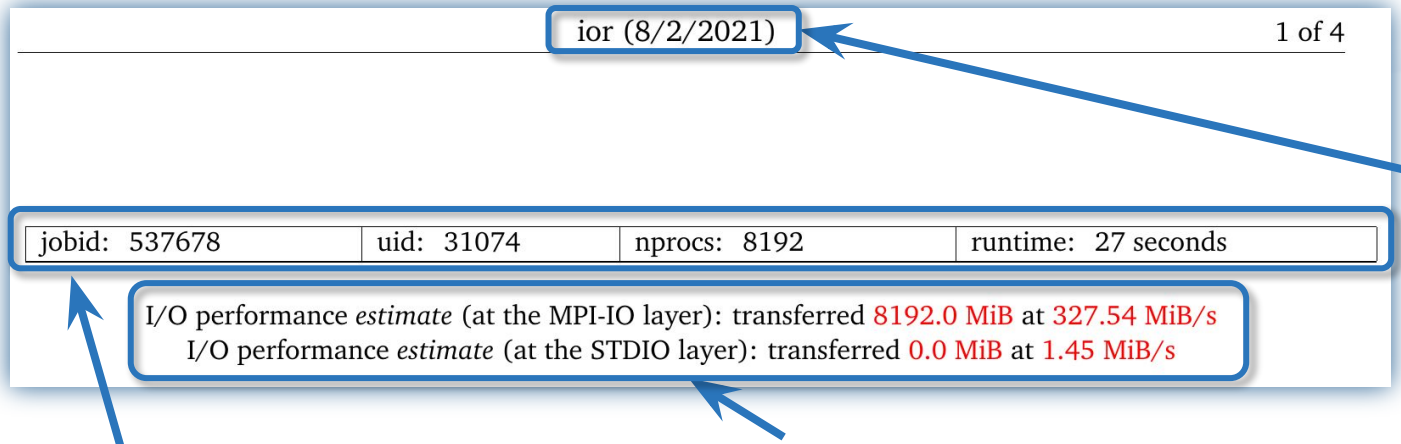
We will summarize some of the highlights in the following slides

Argonne NATIONAL LABORATORY

ECP EXASCALE COMPUTING PROJECT

Job analysis example

PDF header contains high-level job information



Executable name and job date

Scheduler job ID, user ID, number of processes, total app runtime

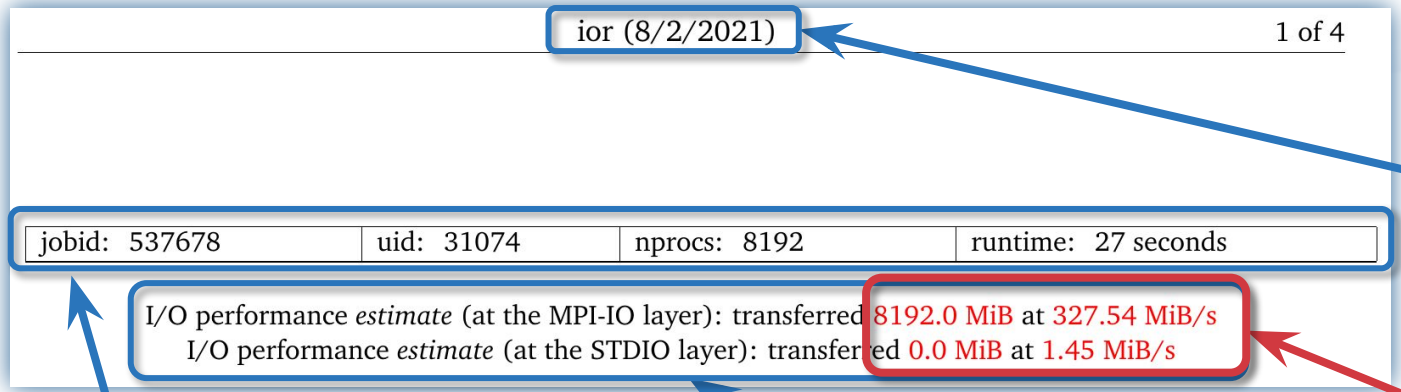
I/O performance estimates for different interfaces:

- MPI-IO (more on this soon)
- POSIX (open/close/read/write)
- STDIO (fopen/fclose/fread/fwrite)

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

Job analysis example

PDF header contains high-level job information



Executable name and job date

Note performance disparity between MPI-IO/STDIO, with the former used for bulk app I/O and latter used only for a small config file

Scheduler job ID, user ID, number of processes, total app runtime

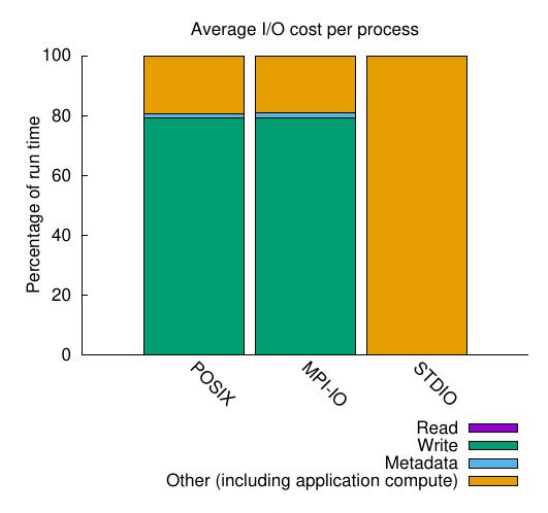
I/O performance estimates for different interfaces:

- MPI-IO (more on this soon)
- POSIX (open/close/read/write)
- STDIO (fopen/fclose/fread/fwrite)

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

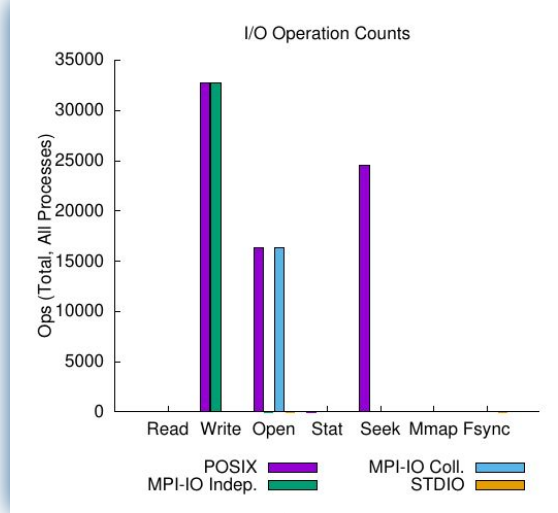
Hands on exercises: https://github.com/radix-io/hands-on

Job analysis example



Across main I/O interfaces, how much time was spent reading, writing, doing metadata, or computing?

If mostly compute, limited opportunities for I/O tuning



What were the relative totals of different I/O operations across key interfaces?

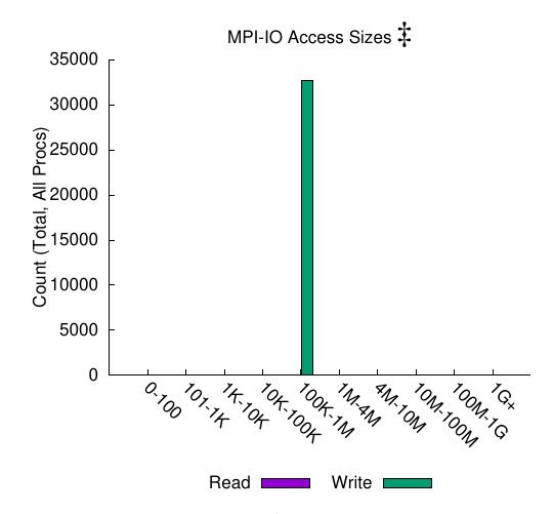
Lots of metadata operations (open, stat, seek, etc.) could be a sign of poorly performing I/O

Argonne NATIONAL LABORATORY

ECP EXASCALE COMPUTING PROJECT

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

Job analysis example



Histograms of POSIX and MPI-IO access sizes are provided to better understand general access patterns

In general, larger access sizes perform better with most storage systems

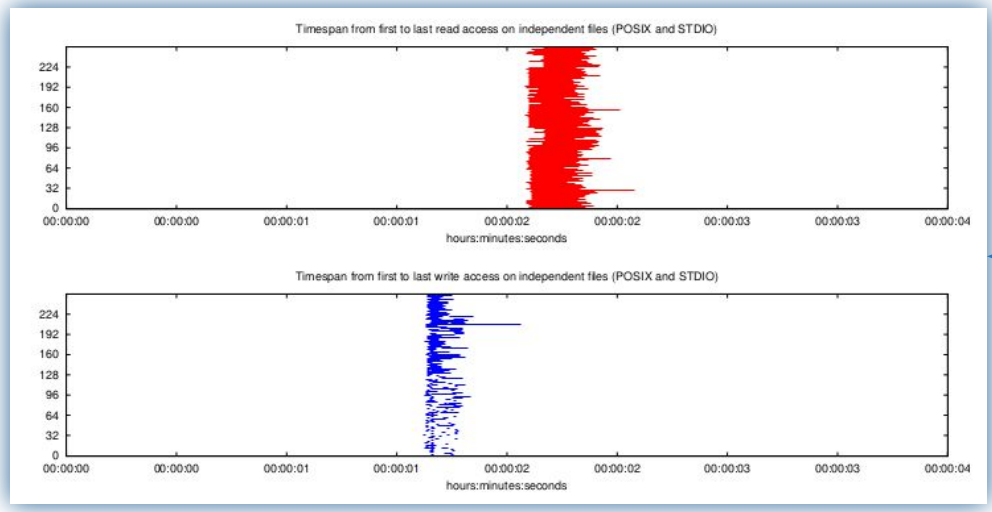
File Count Summary
(estimated by POSIX I/O access offsets)

type	number of files	avg. size	max size
total opened	2	4.1G	8.1G
read-only files	0	0	0
write-only files	2	4.1G	8.1G
read/write files	0	0	0
created files	2	4.1G	8.1G

Table indicating total number of files of different types (opened, read-only, read/write, etc.) recorded by Darshan

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

Job analysis example



Darshan can also provide basic timing bounds for read/write activity, both for independent file access patterns (illustrated) or for shared file access patterns

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

There are additional graphs in the PDF file not shown here. You can also dump all data from the log in human-readable text format using "darshan-parser".

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

Obtaining finer-grained details with Darshan

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

Shared file I/O details

First 2 panes indicate read (first) and write (second) activity for non-shared files

Last pane indicates both read and write activity for shared files (i.e. shared by all application processes)

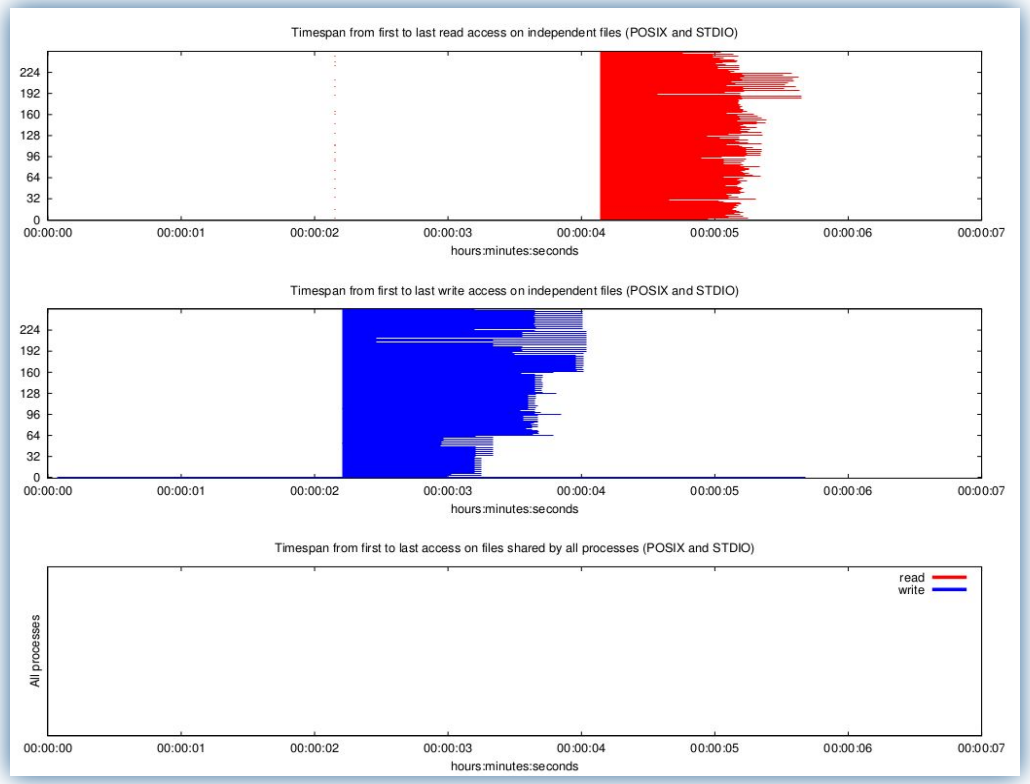
By default, Darshan condenses information on shared files from each rank into a single record to save space

This reduces the fidelity of Darshan's instrumentation a bit, masking per-process details related to file access timing (illustrated), read/write data volumes, etc.

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

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

Shared file I/O details




```
#!/bin/bash
#COBALT -t 30
#COBALT -n 128
#COBALT -q default
#COBALT -A radix-io

EXE_DIR=/home/snyder/software/ior/build
NODES=128
PPN=64

export DARSHAN_DISABLE_SHARED_REDUCTION=1

aprun -n $((NODES * PPN)) -N $PPN $EXE_DIR/src/ior -a POSIX
```

Optionally set the DARSHAN_DISABLE_SHARED_REDUCTION environment variable to skip the shared file reduction step, retaining per-process instrumentation details

This results in larger log files, but may be useful in better understanding underlying access patterns in collective workloads

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

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

Detailed trace data using DXT (Darshan eXtended Tracing)

```
#!/bin/bash
#COBALT -t 30
#COBALT -n 128
#COBALT -q default
#COBALT -A radix-io

EXE_DIR=/home/snyder/software/ior/build
NODES=128
PPN=64

export DXT_ENABLE_IO_TRACE=1

aprun -n $((NODES * PPN)) -N $PPN $EXE_DIR/src/ior -a POSIX
```

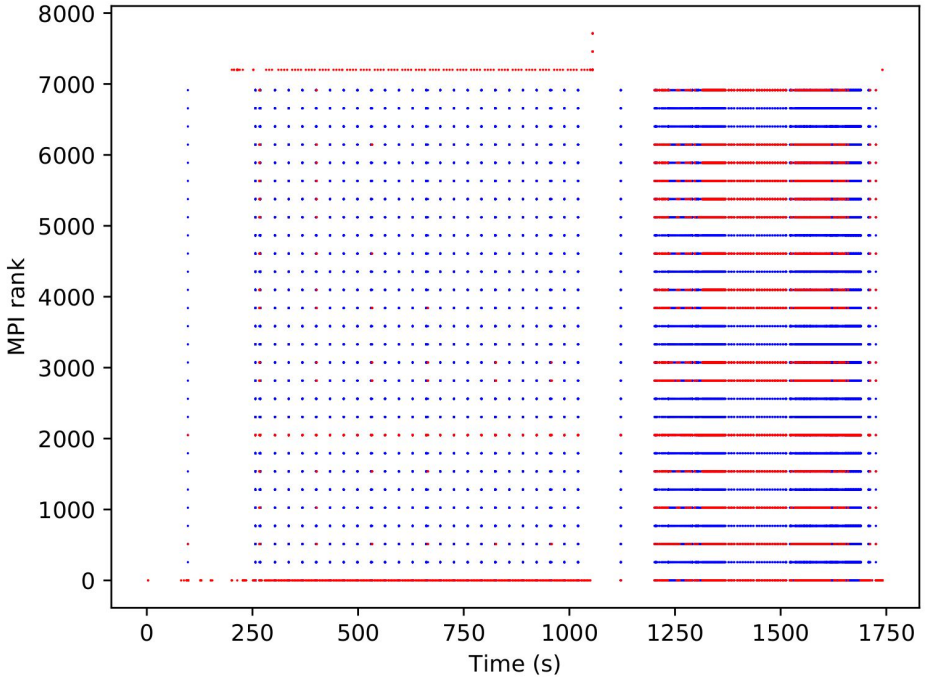
- What if that *still* isn't enough detail? You can also capture a full trace including the timestamp, file offset, and size of every I/O operation on every rank.
- Set the `DXT_ENABLE_IO_TRACE` environment variable in your job to enable this feature.
- This causes additional overhead and larger files, but captures precise access data.

```
# DXT, file_id: 11542722479531699073, file_name: /global/cscratch1/sd/pcarns/ior/ior.dat
# DXT, rank: 0, hostname: nid00511
# DXT, write_count: 16, read_count: 16
# DXT, mnt_pt: /global/cscratch1, fs_type: lustre
# DXT, Lustre stripe_size: 1048576, Lustre stripe_count: 24
# DXT, Lustre OST obdidx: 49 185 115 7 135 3 57 95 43 27 191 1 163 51 15 153 187 55 151 239 79
25 137 47
# Module    Rank  Wt/Rd  Segment          Offset       Length    Start(s)      End(s)  [OST]
 X_POSIX       0  write        0               0      1048576      0.7895      0.8267  [ 49]
 X_POSIX       0  write        1         1048576      1048576      0.8267      0.9843  [185]
 X_POSIX       0  write        2         2097152      1048576      0.9843      1.0189  [115]
 X_POSIX       0  write        3         3145728      1048576      1.0189      1.0250  [  7]
```

A full text dump of DXT trace data can be generated using the `darshan-dxt-parser` tool

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

Detailed trace data using DXT (Darshan eXtended Tracing)



- You can also plot DXT trace data using the "`dxt_analyzer.py`" script distributed with Darshan.

- Example on the left:
  - Looks similar to the timespan plots already provided by the Darshan job summary tool.
  - But, it plots each individual operation precisely, rather than just showing ranges of times that each process was performing I/O.
  - Can help users identify exactly when and where app I/O accesses were issued.
    - This example app clearly uses a subset of processes for performing read/write operations

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

Darshan: a recap

- These slides covered some basic usage and tips.
- Refer to facility documentation 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.
  - Refer to documentation and site support for help interpreting.
  - You will probably start with a PDF generated by darshan-job-summary.pl.
- We'll see additional Darshan use cases and features this afternoon.

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

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!

Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT


EXASCALE COMPUTING PROJECT


Argonne
NATIONAL LABORATORY

Thank you!


U.S. DEPARTMENT OF
ENERGY
Office of
Science


NNSA
National Nuclear Security Administration