

March 2, 2021

Darshan: Enabling Application I/O Understanding in an Evolving HPC Landscape

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

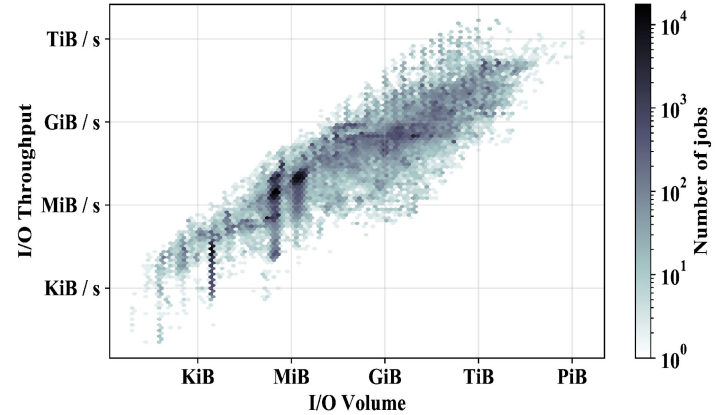


Figure courtesy of Mihailo Isakov (TAMU)

NERSC Data Seminar

Understanding and improving HPC I/O

- ❖ The ability to characterize and understand application I/O workloads is critical to ensuring efficient use of an evolving and increasingly complex HPC I/O stack
 - Deep layers of coordinating I/O libraries and entirely new-to-HPC storage paradigms (e.g., object storage)
 - Emerging storage hardware (e.g., PMEM) and storage architectures (e.g., burst buffers)
- ❖ I/O analysis tools are invaluable in helping to navigate this complexity and to better understand I/O
 - Characterize I/O behavior of individual jobs to inform tuning decisions
 - Characterize job populations to better understand system-wide I/O stack usage and optimize deployments



Darshan: An application I/O characterization tool for HPC



Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.

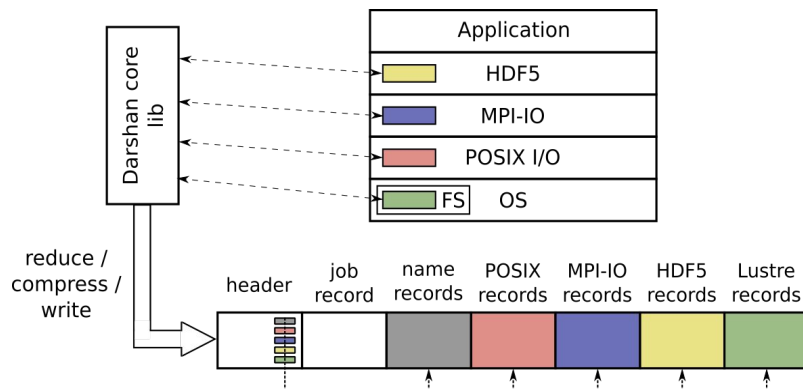


What is Darshan?

- ❖ Darshan is a lightweight I/O characterization tool that captures concise views of HPC application I/O behavior
 - Produces a summary of I/O activity for each instrumented job
 - Counters, histograms, timers, & statistics
 - Full I/O traces (if requested)
- ❖ Widely available
 - Deployed (and commonly enabled by default!) at many HPC facilities around the world
- ❖ Easy to use
 - No code changes required to integrate Darshan instrumentation
 - Negligible performance impact; just “leave it on”
- ❖ Modular
 - Adding instrumentation for new I/O interfaces or storage components is straightforward

How does Darshan work?

- ❖ Darshan can insert application I/O instrumentation at link-time (for static and dynamic executables) or at runtime using LD_PRELOAD (for dynamic executables)
 - Starting in version 3.2.0, Darshan supports instrumentation of any dynamically-linked executable (MPI or not) using the LD_PRELOAD method
- ❖ Darshan records file access statistics for each process as app executes
- ❖ At app shutdown, collect, aggregate, compress, and write log data
- ❖ After job completes, analyze Darshan log data
 - `darshan-job-summary` - provides a summary PDF characterizing application I/O behavior
 - `darshan-parser` - provides complete text-format dump of all counters in a log file
 - `pydarshan` - Python analysis module for Darshan logs



Using Darshan on Cori



Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.



Using Darshan on Cori

MPI applications

- ❖ Darshan is already installed and on by default on NERSC's Cori system
 - Instrumentation enabled using Cray software module that injects Darshan linker options when compiling MPI applications using Cray compiler wrappers (cc, CC, etc.)

```
ssnyder@cori05:~> module list
- Package -----
Currently Loaded Modulefiles:
modules/3.2.11.4
altd/2.0
darshan/3.2.1
craype-network-aries
intel/19.0.3.199
craype/2.6.2

ssnyder@cori05:~> module load darshan
```

Use 'module list' to confirm Darshan is actually loaded

Darshan 3.2.1 current default version available on Cori

If Darshan not loaded, you can always load manually using 'module load'

Using Darshan on Cori

HDF5 applications

- ❖ Note that in addition to the default darshan/3.2.1 module, there is a special darshan/3.2.1-hdf5 that enables instrumentation of HDF5 APIs*
 - Offered as a separate module to prevent non-HDF5 applications from inheriting Darshan's HDF5 library dependency

```
ssnyder@cori07:~> module avail -l 2>&1 | grep darshan | cat
darshan/3.1.7                                2021/03/03
darshan/3.1.7-test                          2021/03/03
darshan/3.2.1                                default 2021/03/03
darshan/3.2.1-hdf5                          2021/03/03
ssnyder@cori07:~>
ssnyder@cori07:~> module switch darshan/3.2.1 darshan/3.2.1-hdf5
```

Use 'module switch' to switch to the non-default Darshan module built with HDF5 support

***Note:** this module is only compatible with cray-hdf5 versions 1.10+

Using Darshan on Cori

- ❖ OK, Darshan is loaded...now what?
 - Just compile and run your application!
 - Darshan inserts instrumentation directly into executable
- ❖ LD_PRELOAD is another option for dynamically-linked executables:
 - This method is necessary for Python environments (i.e., mpi4py, h5py)
 - Also helpful for applications that cannot be recompiled

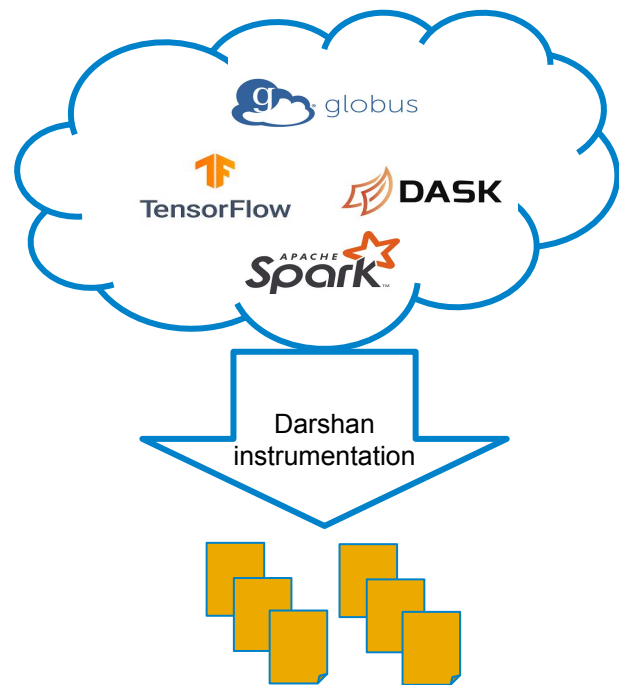
```
ssnyder@nid00010:/global/cscratch1/sd/ssnyder/darshan-test> module show darshan
-----
/global/common/software/nersc/cle7/extra_modulefiles/darshan/3.2.1:
module-whatis      a scalable HPC I/O characterization tool
prepend-path       PE_PKGCONFIG_LIBS darshan-runtime
prepend-path       PKG_CONFIG_PATH /usr/common/software/darshan/3.2.1/lib/pkgconfig
prepend-path       LD_LIBRARY_PATH /usr/common/software/darshan/3.2.1/lib
prepend-path       PATH /usr/common/software/darshan/3.2.1/bin
-----
ssnyder@nid00010:/global/cscratch1/sd/ssnyder/darshan-test> export LD_PRELOAD=/usr
/common/software/darshan/3.2.1/lib/libdarshan.so
```

Manually set LD_PRELOAD to point to Darshan's shared library before running your application

Using Darshan on Cori

non-MPI applications

- ❖ Starting in version 3.2.0, Darshan supports instrumentation of non-MPI applications using LD_PRELOAD (i.e., dynamically-linked binaries)
 - Users must additionally export 'DARSHAN_ENABLE_NONMPI=1' to enable Darshan in this case
- ❖ Note: there is a bug in the Darshan 3.2.1 module on Cori that prevents non-MPI mode from working properly
 - Users can install an explicit non-MPI version of Darshan themselves to workaround for now (e.g., 'spack install darshan-runtime~mpi+slurm')



Using Darshan on Cori

- ❖ After the application terminates, look for your log files:

Darshan logs stored in a central directory for all Cori users, use `'darshan-config --log-path'` to find

```
ssnyder@cori01:~> darshan-config --log-path
/global/cscratch1/sd/darshanlogs
ssnyder@cori01:~> cd /global/cscratch1/sd/darshanlogs
ssnyder@cori01:/global/cscratch1/sd/darshanlogs>
ssnyder@cori01:/global/cscratch1/sd/darshanlogs> cd 2021/3/4
ssnyder@cori01:/global/cscratch1/sd/darshanlogs/2021/3/4> ls | grep snyder | cat
ssnyder_mpi-io-test_id40245367_3-4-50083-3517743081787486417_1614894884.darshan
```

Logs further indexed using `'year/month/day'` the job executed.

Log file name starts with the following pattern:
`'username_exename_jobid...'`

- ❖ Note: Spack-installed Darshan versions simply put log files directly in your home directory by default -- override by setting `DARSHAN_LOG_DIR_PATH` environment variable

Analyzing Darshan logs



Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.



Analyzing Darshan logs

- ❖ After generating and locating your log, use Darshan analysis tools to inspect log file data:

```
ssnyder@cori07:.../4> cp ssnyder_mpi-io-test_id40245367_3-4-50083-35177430817874  
86417_1614894884.darshan ~/tmp-analysis/  
ssnyder@cori07:.../4> cd ~/tmp-analysis/
```

```
ssnyder@cori07:~/tmp-analysis> darshan-parser ssnyder_mpi-io-test_id40245367_3-4-  
-50083-3517743081787486417_1614894884.darshan
```

```
#<module> <rank> <record id> <counter> <value> <file name>  
POSIX -1 4212256932904785913 POSIX_OPENS 16 /global/csc  
POSIX -1 4212256932904785913 POSIX_FILENOS 0 /global/csc  
POSIX -1 4212256932904785913 POSIX_DUPS 0 /global/csc  
POSIX -1 4212256932904785913 POSIX_READS 4 /global/csc  
  
MPI-IO -1 4212256932904785913 MPIIO_INDEP_OPENS 8 /gld  
MPI-IO -1 4212256932904785913 MPIIO_COLL_OPENS 0 /gld  
MPI-IO -1 4212256932904785913 MPIIO_INDEP_READS 4 /gld  
MPI-IO -1 4212256932904785913 MPIIO_INDEP_WRITES 4 /gld
```

Copy the log file
somewhere else for
analysis

Invoke darshan-parser
(already in PATH on Cori)
to get detailed counters

Modules use a common format
for printing counters, indicating
the module, rank, record ID,
counter name, counter value,
filename, etc. -- here sample
counters are shown for both
POSIX and MPI-IO modules

Analyzing Darshan logs

- ❖ But, darshan-parser output isn't so accessible for most users... use darshan-job-summary tool to produce summary PDF of app I/O behavior
 - Due to LaTeX and Perl dependencies, it may be easier just to copy Darshan logs to a personal workstation for analysis

Invoke darshan-job-summary on log file to produce PDF

```
shane@shane-x1-carbon ~/tmp-analysis $ darshan-job-summary.pl ssnyder_mpi-io-test_id40245367_3-4-50083-3517743081787486417_1614894884.darshan
shane@shane-x1-carbon ~/tmp-analysis $
shane@shane-x1-carbon ~/tmp-analysis $ ls
ssnyder_mpi-io-test_id40245367_3-4-50083-3517743081787486417_1614894884.darshan
ssnyder_mpi-io-test_id40245367_3-4-50083-3517743081787486417_1614894884.darshan.pdf
```

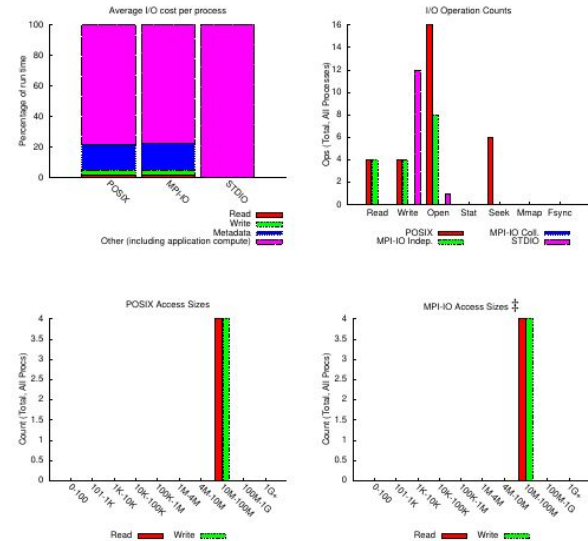
Output PDF file name based on Darshan log file name

Analyzing Darshan logs

I/O performance estimate (at the MPI-IO layer): transferred **642 MiB** at **266.40 MiB/s**
 I/O performance estimate (at the STDIO layer): transferred **0.0 MiB** at **2.08 MiB/s**

Result is 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



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

type	number of files	avg. size	max size
total opened	2	33M	64M
read-only files	0	0	0
write-only files	1	642	642
read/write files	1	64M	64M
created files	2	33M	64M

Most Common Access Sizes
(POSIX or MPI-IO)

	access size	count
POSIX	16777216	8
MPI-IO ‡	16777216	8

‡ NOTE: MPI-IO accesses are given in terms of aggregate datatype size.

Analyzing Darshan logs

PDF header contains some high-level information on the job execution



jobid: 403177	uid: 31074	nprocs: 4	runtime: 2 seconds
---------------	------------	-----------	--------------------

I/O performance *estimate* (at the MPI-IO layer): transferred **642 MiB** at **266.40 MiB/s**
I/O performance *estimate* (at the STDIO layer): transferred **0.0 MiB** at **2.08 MiB/s**



I/O performance estimates (and total I/O volumes) provided for MPI-IO/POSIX and STDIO interfaces

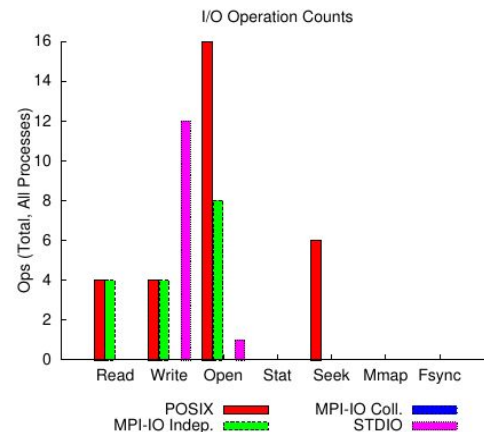
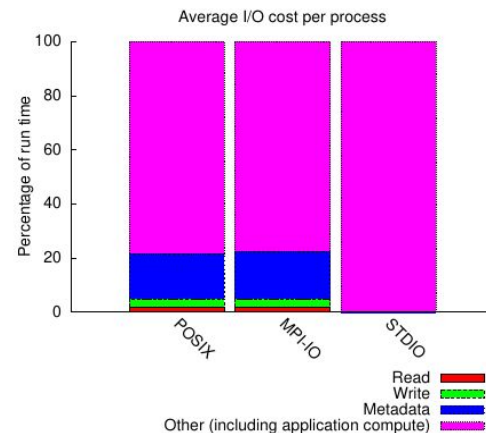
Analyzing Darshan logs

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



Analyzing Darshan logs

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

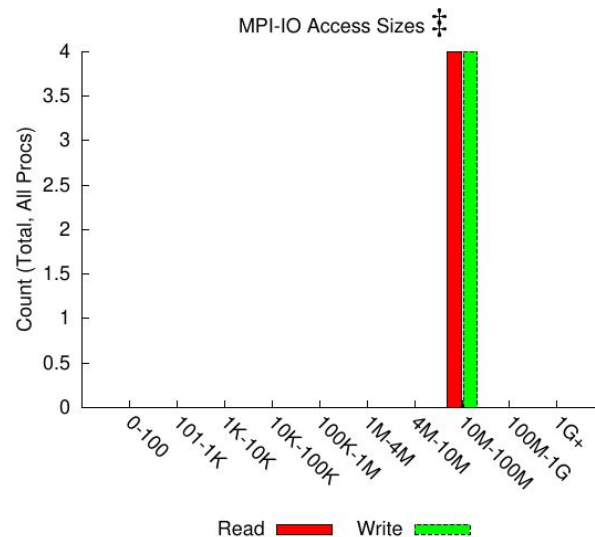


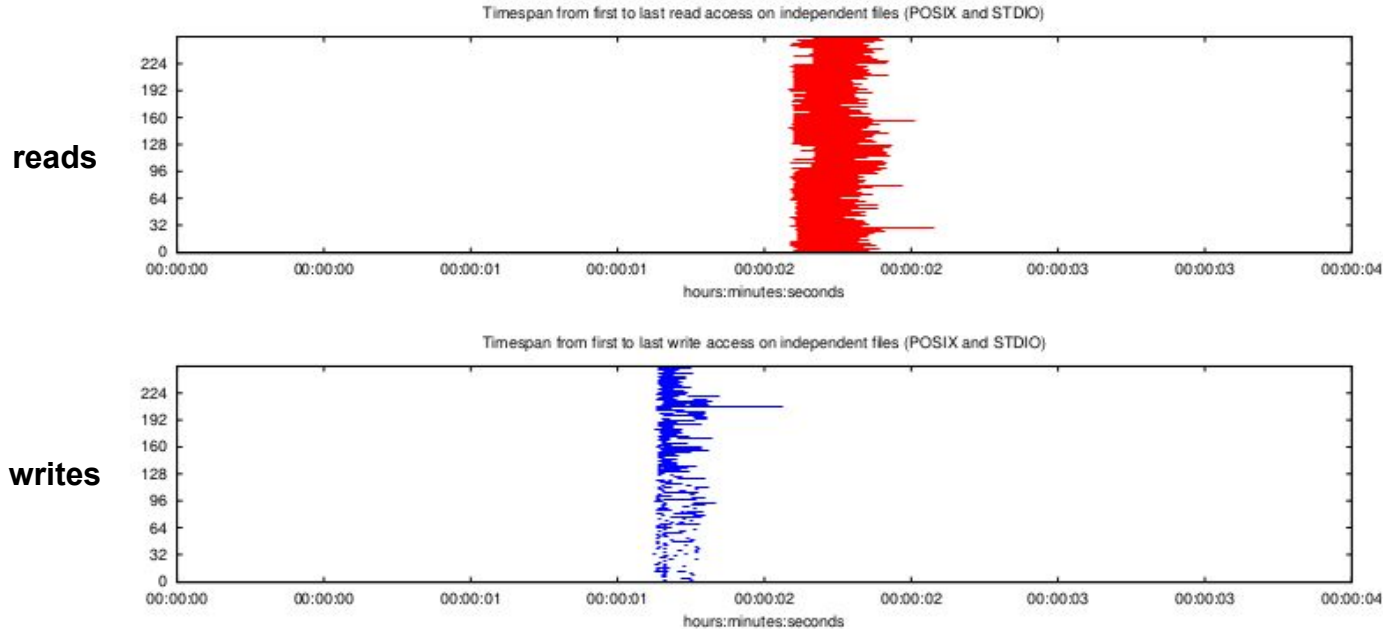
Table indicating total number of files of different types (opened, created, read-only, etc.) recorded by Darshan



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

type	number of files	avg. size	max size
total opened	2	33M	64M
read-only files	0	0	0
write-only files	1	642	642
read/write files	1	64M	64M
created files	2	33M	64M

Analyzing Darshan logs



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

Obtaining finer-grained details with Darshan



Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.



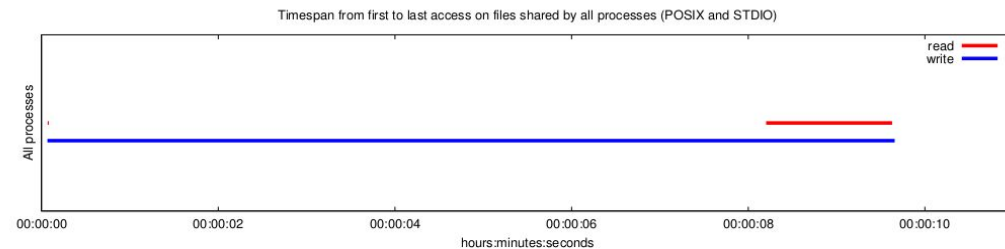
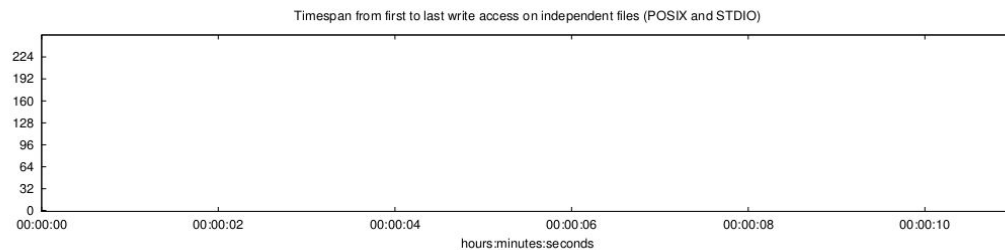
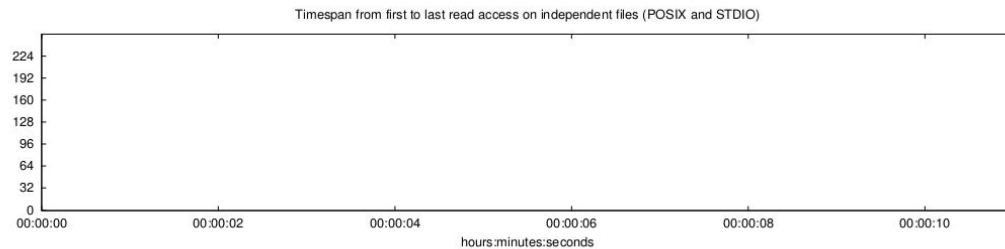
Focusing analysis on individual files

- ❖ If we want to focus Darshan analysis tools on a specific file, Darshan offers a couple of different options
 - `darshan-convert` utility can be used to create a new Darshan log file containing a specified file record ID (obtainable from `darshan-parser` output)
 - e.g., '`darshan-convert --file RECORD_ID input_log.darshan output_log.darshan`'
 - New log file can be ran through existing log utilities we have already covered
 - `darshan-summary-per-file` tool can be used to generate separate job summary PDFs for every file in a given Darshan log
 - Do not use if your application opens a lot of files!

Disabling reductions of shared records

You may notice that Darshan is unable to provide more detailed access information for shared file workloads, as illustrated here

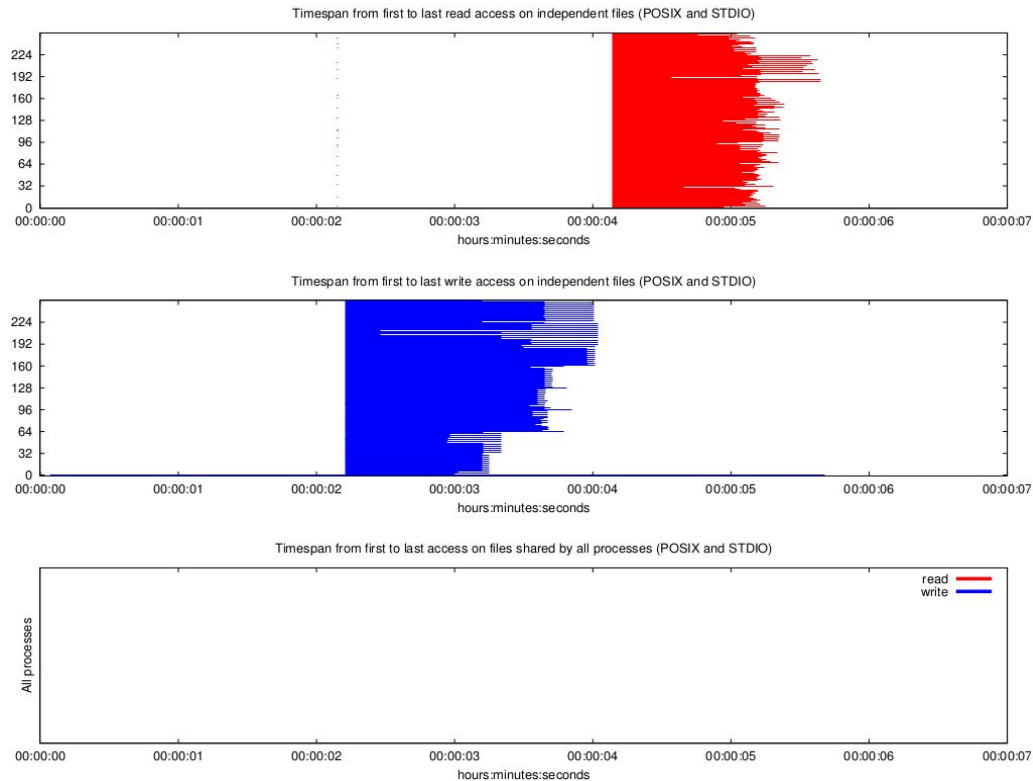
For shared files, information from each rank is combined into a single record to save space



Disabling reductions of shared records

Setting the
'DARSHAN_DISABLE_SHARED_REDUCTION'
environment variable will force Darshan
to skip the shared file reduction step,
retaining each process's independent
view of access information

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



Obtaining fine-grained traces with DXT

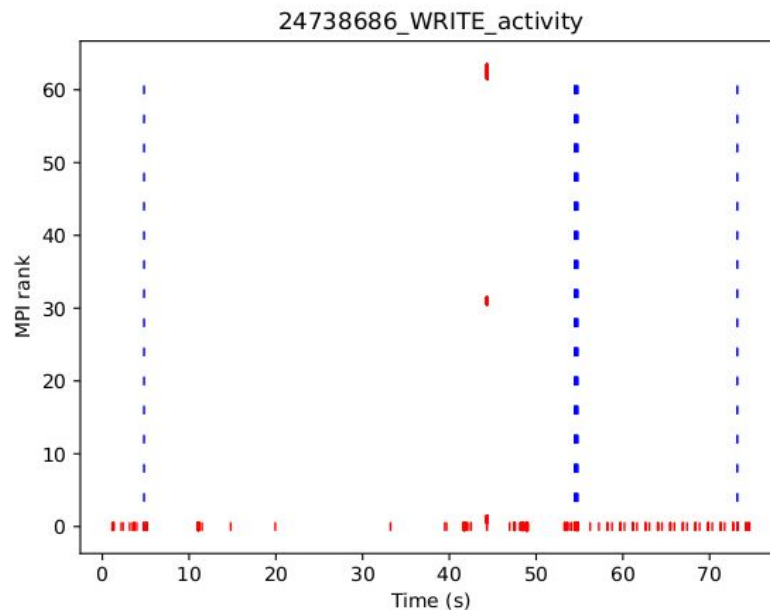
- ❖ Darshan's DXT module can be enabled at runtime for users wishing to capture detailed I/O traces for MPI-IO and POSIX interfaces
 - Fine-grained trace data comes at cost of larger per-process memory overheads
 - Set the `DXT_ENABLE_IO_TRACE` environment variable to enable
- ❖ `darshan-dxt-parser` can be then be used to dump text-format trace data:

```
# *****  
# DXT_POSIX module data  
# *****  
  
# DXT, file_id: 11542722479531699073, file_name: /global/cscratch1/sd/pcarns/ior/ior.dat - summa  
# 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]
```

Obtaining fine-grained traces with DXT

- ❖ `dxt_analyzer` Python script installed with `darshan-util` can be used to help visualize read/write trace activity:

Provides details on each I/O operation issued by each rank, providing a complete picture of which ranks are performing I/O and how long they are spending on I/O



New and upcoming Darshan features: HDF5 instrumentation



Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.



HDF5 instrumentation

Available in Darshan 3.2.0+

- ❖ HDF5 offers a convenient abstraction for large data collections, but it can be difficult to understand how it interacts with lower layers of the I/O stack that most impact performance
- ❖ To help better understand HDF5 usage and performance, we have developed Darshan instrumentation modules for HDF5 file (H5F) and dataset (H5D) APIs
 - What are file and dataset properties?
 - How are datasets accessed?
 - How are datasets organized within files?
 - Do HDF5 accesses decompose efficiently to lower-level (i.e., MPI-IO and POSIX) accesses? If not, do any optimizations make sense?

HDF5 instrumentation

Available in Darshan 3.2.0+

❖ H5F instrumentation highlights:

- Operation counts
 - open/create
 - flush
- MPI-IO usage
- Metadata timing

```
#<module>  <rank>  <record id> <counter>  <value> <file na
H5F -1  11831850109748558379  H5F_OPENS  8  /home/shane/
H5F -1  11831850109748558379  H5F_FLUSHES 0  /home/shane/
H5F -1  11831850109748558379  H5F_USE_MPIIO  1  /home/sh
H5F -1  11831850109748558379  H5F_F_OPEN_START_TIMESTAMP
H5F -1  11831850109748558379  H5F_F_CLOSE_START_TIMESTAMP
H5F -1  11831850109748558379  H5F_F_OPEN_END_TIMESTAMP
H5F -1  11831850109748558379  H5F_F_CLOSE_END_TIMESTAMP
H5F -1  11831850109748558379  H5F_F_META_TIME 0.019466
```

HDF5 instrumentation

Available in Darshan 3.2.0+

- ❖ H5D instrumentation highlights:
 - Operation counts:
 - open/create
 - read/write
 - flush
 - Total bytes read/written
 - Access size histograms
 - Dataspace selection types
 - Regular hyperslab
 - Irregular hyperslab
 - Points
 - Dataspace total dimensions, points
 - Chunking parameters
 - MPI-IO collective usage
 - Deprecated function usage
 - Read, write, and metadata timing

```
#<module> <rank> <record id> <counter> <value>
H5D -1 7600138186531619366 H5D_OPENS 8 /home/st
H5D -1 7600138186531619366 H5D_READS 16 /home/st
H5D -1 7600138186531619366 H5D_WRITES 16 /home/st
H5D -1 7600138186531619366 H5D_FLUSHES 0 /home/st
H5D -1 7600138186531619366 H5D_BYTES_READ 4194304
H5D -1 7600138186531619366 H5D_BYTES_WRITTEN 4194
H5D -1 7600138186531619366 H5D_RW_SWITCHES 4 /hor
H5D -1 7600138186531619366 H5D_REGULAR_HYPERSLAB_SE
xt4
H5D -1 7600138186531619366 H5D_IRREGULAR_HYPERSLAB_
xt4
H5D -1 7600138186531619366 H5D_POINT_SELECTS 0
H5D -1 7600138186531619366 H5D_MAX_READ_TIME_SIZE
H5D -1 7600138186531619366 H5D_MAX_WRITE_TIME_SIZE
H5D -1 7600138186531619366 H5D_SIZE_READ_AGG_0_100
H5D -1 7600138186531619366 H5D_SIZE_READ_AGG_100_41
```

HDF5 instrumentation

Available in Darshan 3.2.0+

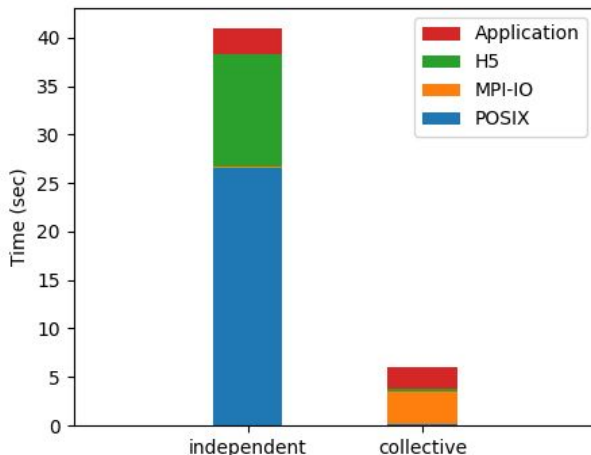
- ❖ Using the MACSio¹ HDF5 plugin, run a couple of simple examples demonstrating the types of insights HDF5 I/O instrumentation can enable
 - 60-process (5-node) single shared file, 3d mesh, write roughly 1 GiB of cumulative H5D data
 - Compare performance of collective and independent I/O configurations

b/w: ~30 MB/sec

POSIX I/O dominates, **H5** incurs non-negligible overhead forming this workload

Negligible time spent in **MPI-IO**

Average per-process time spent in I/O



b/w: ~290 MB/sec

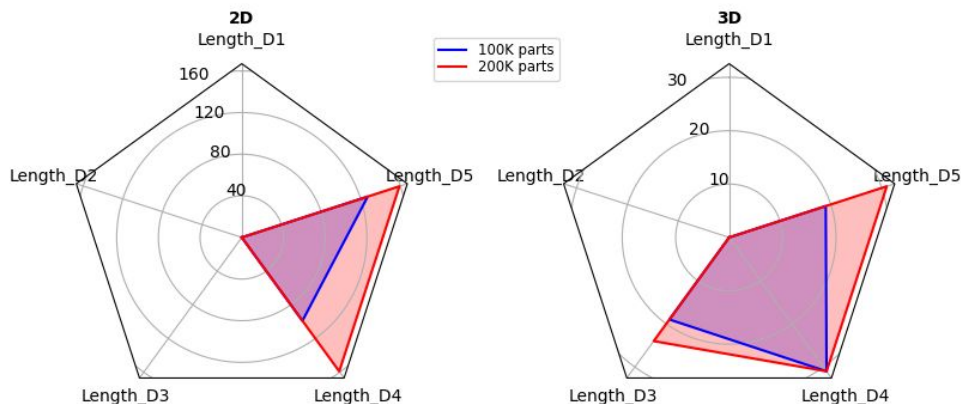
H5 and **POSIX** incur minimal overhead for this workload

MPI-IO collective I/O algorithm dominates

HDF5 instrumentation

Available in Darshan 3.2.0+

- ❖ Using the MACSio¹ HDF5 plugin, run a couple of simple examples demonstrating the types of insights HDF5 I/O instrumentation can enable
 - 60-process (5-node) single shared file, 3d mesh, write roughly 1 GiB of cumulative H5D data
 - Compare performance of collective and independent I/O configurations



Radar plots, or other methods, can be used to help visualize characteristics of HDF5 dataset accesses

Dataset access patterns could be used to help set/optimize chunking parameters to limit accesses to as few chunks as possible

New and upcoming Darshan features: DAOS instrumentation



Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.



DAOS instrumentation

Work in progress

- ❖ Intel's DAOS offers an exciting new storage paradigm for HPC apps, utilizing object-based storage interfaces over a combo of SCM and SSD devices
 - libdfs: DAOS's POSIX file system emulation API
 - libdaos: DAOS's native object (key-val) API
- ❖ Darshan will instrument libdaos and libdfs APIs to help provide insight into application and I/O middleware usage of DAOS
 - Legacy POSIX app usage and performance characteristics
 - Usage and performance characteristics of libdaos users (libdfs, HDF5 VOL, MPI-IO, etc.)

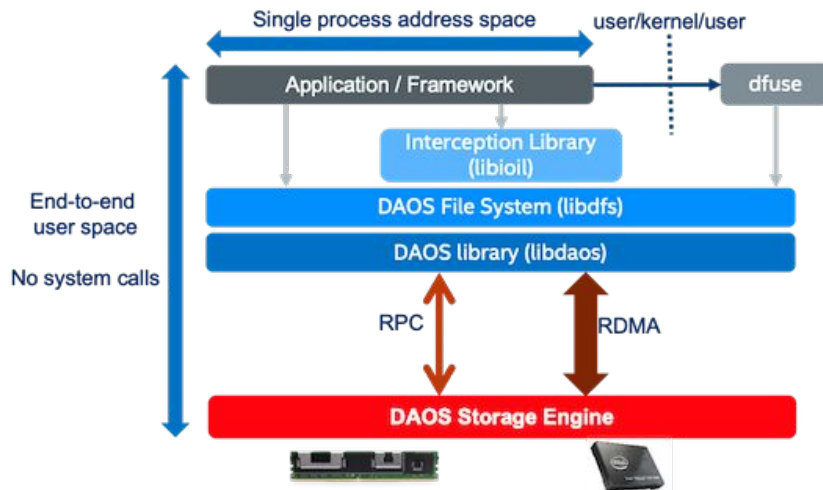


Figure courtesy of Intel

DAOS instrumentation

Work in progress

❖ libdfs instrumentation highlights:

- Operation counts:
 - open
 - read/write
 - punch
 - stat
- Total bytes read/written
- Access size histograms
- File chunk size
- DTX usage (strict consistency mode)
- Corresponding DAOS object record ID
 - Necessary to link Darshan's DFS records with native DAOS records
- Read, write, and metadata timing

```
#<module> <rank> <record id> <counter> <value>
DFS 0 8492698188889325809 DFS_OPENS 0 c3bcfc9a
DFS 0 8492698188889325809 DFS_GLOBAL_OPENS 0
DFS 0 8492698188889325809 DFS_LOOKUPS 1 c3bcfc9a
DFS 0 8492698188889325809 DFS_DUPS 0 c3bcfc9a
DFS 0 8492698188889325809 DFS_READS 0 c3bcfc9a
DFS 0 8492698188889325809 DFS_READXS 0 c3bcfc9a
DFS 0 8492698188889325809 DFS_WRITES 0 c3bcfc9a
DFS 0 8492698188889325809 DFS_WRITEXS 0 c3bcfc9a
DFS 0 8492698188889325809 DFS_NB_READS 0 c3bcfc9a
DFS 0 8492698188889325809 DFS_NB_WRITES 0 c3bcfc9a
DFS 0 8492698188889325809 DFS_GET_SIZES 0 c3bcfc9a
DFS 0 8492698188889325809 DFS_PUNCHES 0 c3bcfc9a
```

Note: Unsurprisingly, Darshan's DFS instrumentation module closely follows the design of the POSIX module

DAOS instrumentation

Work in progress

- ❖ libdaos instrumentation highlights:
 - Operation counts:
 - open
 - fetch (read) and update (write)
 - list (enumeration)
 - punch
 - Total bytes read/written
 - Access size histograms
 - Object class parameters
 - layout (static or dynamic striping)
 - redundancy (replication, erasure coding)
 - Container/pool UUIDs
 - Read, write, and metadata timing

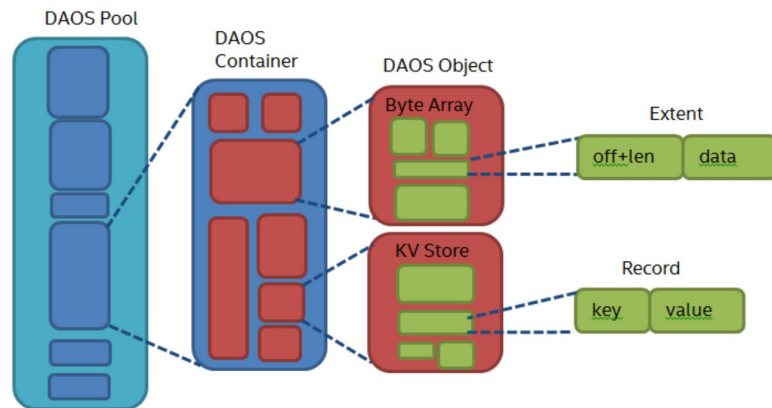


Figure courtesy of Intel

Note: Each DAOS object is multi-level key-val store, creating challenges for deciding what can be instrumented using a fixed set of Darshan counters -- we are still investigating what characteristics about key access patterns to capture

New and upcoming Darshan features: pydarshan



Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.



pydarshan log file analysis

Available in Darshan 3.3.0, coming soon!

* Thanks to **Jakob Luetzgau (DKRZ)** for contributing the majority of the pydarshan code, examples, and documentation

- ❖ Darshan has traditionally offered only the C-based darshan-util library and a handful of corresponding utilities to users
 - Development of custom Darshan analysis utilities is cumbersome, requiring users to either:
 - Develop analysis tools in C using the low-level darshan-util library
 - Perform an inconvenient conversion from darshan-parser text output
- ❖ pydarshan has been developed* to simplify the interfacing of analysis tools with Darshan log data
 - Use Python CFFI module to provide Python bindings to the native darshan-utils C API
 - Expose Darshan log data as dictionaries, pandas dataframes, and numpy arrays
- ❖ We are hopeful pydarshan will lead to a richer ecosystem for Darshan log analysis utilities

pydarshan log file analysis

Available in Darshan 3.3.0, coming soon!

- ❖ We've already found Jupyter notebooks to be an effective way of sharing pydarshan analysis examples (code, documentation, visualizations) with users, collaborators, etc.

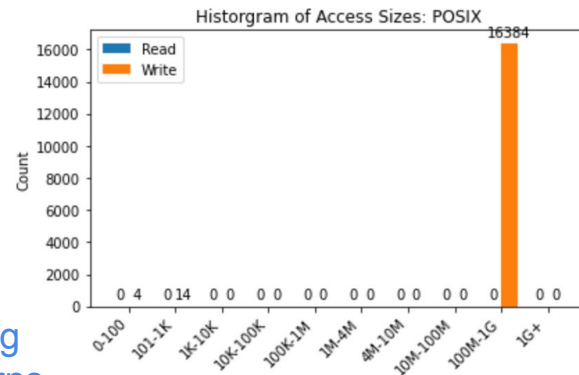
In [1]: `import darshan`

```
report = darshan.DarshanReport("example-logs/example.darshan", read_all=True) # Default
report.info()
```

```
Filename:      example-logs/example.darshan
Times:        2017-03-20 04:07:47 to 2017-03-20 04:09:43 (Duration 0:01:56)
Executeable:  /global/project/projectdirs/m888/glock/tokio-abc-results/bin.edison/vpici
rs/glock/tokioabc-s.4478544/vpicio/vpicio.hdf5 32
Processes:    2048
JobID:        4478544
UID:          69615
Modules in Log: ['POSIX', 'MPI-IO', 'LUSTRE', 'STDIO']
Loaded Records: {'POSIX': 1, 'MPI-IO': 1, 'STDIO': 1, 'LUSTRE': 1}
Name Records: 4
Darshan/Hints: {'lib_ver': '3.1.3', 'h': 'romio_no_indep_rw=true;cb_nodes=4'}
DarshanReport: id(140124449925824) (tmp)
```

In [3]: `# access histograms`
`plt = plot_access_histogram(report, 'POSIX')`
`plt.show()`

Summarizing... iohist POSIX



In just a few lines of code, users can read a Darshan log into memory and generate plots describing access patterns

pydarshan log file analysis

Available in Darshan 3.3.0, coming soon!

- ❖ A beta version of pydarshan is currently available in PyPI and ready for users to experiment with
 - Use `pip install darshan` to install the pydarshan module from PyPI on your system
 - Alternatively, pydarshan can be installed alongside darshan-util, by providing `--enable-pydarshan --enable-shared` configure options

darshan 0.0.7 ✓ Latest version

`pip install darshan`

Released: Jan 27, 2021

Python tools to interact with darshan log records of HPC applications.

Navigation

- Project description
- Release history
- Download files

Project description

Python utilities to interact with Darshan log records of HPC applications. pydarshan requires darshan-utils (3.2.2+) to be installed.

Features

- Darshan Report Object Wrapper
- CFFI bindings to access darshan log files
- Plots typically found in the darshan reports (matplotlib)
- Auto-discover darshan-util.so (via darshan-parser in \$PATH)

Other upcoming Darshan features

- ❖ In addition to those covered today, the following features are also on the Darshan roadmap:
 - Autoperf instrumentation module (available in Darshan 3.3.0)
 - APMPI - MPI communication counters
 - CrayXC - compute and network counters for Cray XC systems
 - Thanks to **Sudheer Chunduri (ALCF)** for this contribution!
 - PnetCDF instrumentation module (work in progress)
 - Full instrumentation of PnetCDF blocking/nonblocking APIs
 - Thanks to **Claire Lee (NWU)** and **Wei-keng Liao (NWU)** for this contribution!
 - Fork handlers for non-MPI mode (work in progress, available in 'dev-fork-safe' branch)
 - Numerous updates to allow forked processes to generate their own Darshan logs
 - Enhanced analysis tools and report generation (???)
 - Building off of our pydarshan log utility bindings, we want to revamp our analysis tools and report generation using Python

Broader system-wide analysis using Darshan: Gauge



Argonne National Laboratory is a
U.S. Department of Energy laboratory
managed by UChicago Argonne, LLC.

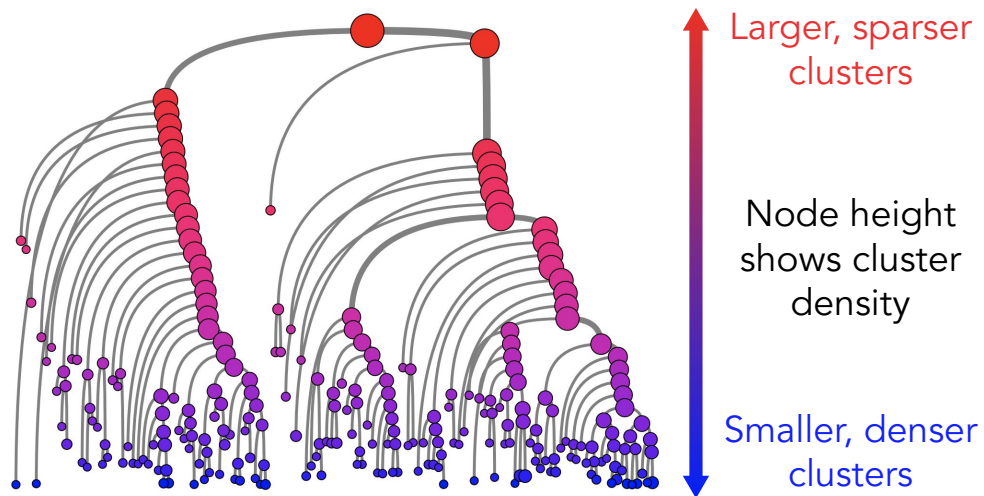


Special acknowledgement for the Gauge tool and the content of this slide goes to **Mihailo Isakov, et al. (TAMU)**

Gauge

I/O clustering and analysis tool for Darshan logs

- ❖ Allow system administrators and I/O experts to gain insights into classes of I/O workloads running on a computer
- ❖ HDBSCAN clustering used to organize Darshan logs into hierarchy of clusters that exhibit similar I/O characteristics
- ❖ Gauge users may select clusters to see more information on the clusters I/O characteristics
 - User and application details
 - Job features
 - Access patterns



Iskaov et al. "HPC I/O Throughput Bottleneck Analysis with Explainable Local Models", SC'20

del Rosario et al. "Gauge: An Interactive Data-Driven Visualization Tool for HPC Application I/O Performance Analysis", PDSW'20

Special acknowledgement for the Gauge tool and the content of this slide goes to Mihailo Isakov, et al. (TAMU)

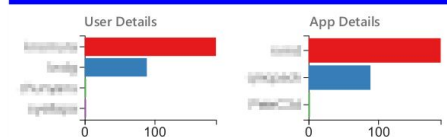
Gauge

I/O clustering and analysis tool for Darshan logs

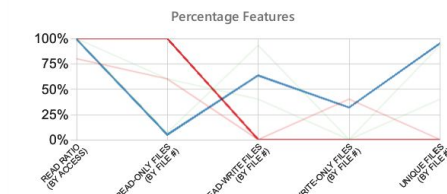
- ❖ Allow system administrators and I/O experts to gain insights into classes of I/O workloads running on a computer
- ❖ HDBSCAN clustering used to organize Darshan logs into hierarchy of clusters that exhibit similar I/O characteristics
- ❖ Gauge users may select clusters to see more information on the clusters I/O characteristics
 - User and application details
 - Job features
 - Access patterns



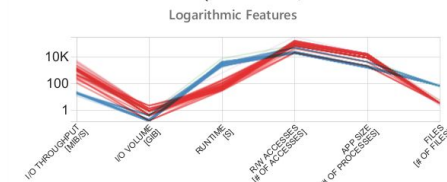
} Cluster name & options



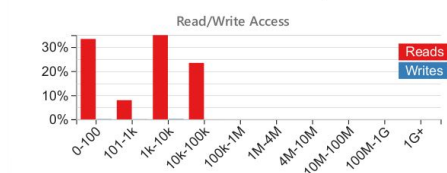
} User and application details: breakdown of users and applications



} Percentage features: job features represented as a ratio in 0-100 %



} Absolute features: job features that don't have a known range



} Breakdown of accesses by access size and read / write properties

Wrapping up

- ❖ We've covered a range of topics today demonstrating the role Darshan can play in developing a better understanding of HPC I/O behavior
 - Runtime characterization of application usage of a number of different I/O libraries
 - Traditional I/O libraries like POSIX, MPI-IO, HDF5, as well as upcoming libraries such as DAOS
 - I/O insights enabled via Darshan data, using existing utilities in the Darshan repo or using 3rd party tools like Gauge
 - pydarshan can greatly simplify the process of extending existing Darshan analysis tools or developing custom analysis tools
- ❖ Please reach out with any questions, comments, or feedback!
- ❖ Darshan website: <https://www.mcs.anl.gov/research/projects/darshan/>
- ❖ Darshan-users mailing list: darshan-users@lists.mcs.anl.gov
- ❖ Source code, issue tracking: <https://xggitlab.cels.anl.gov/darshan/darshan>
- ❖ NERSC documentation: <https://docs.nersc.gov/development/performance-debugging-tools/darshan>

Acknowledgement

This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. This research also used resources and data generated from resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.