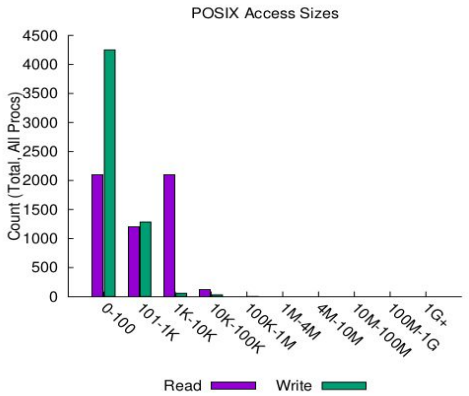
February 4, 2020

Argonne NATIONAL LABORATORY

I/O Performance Addicts



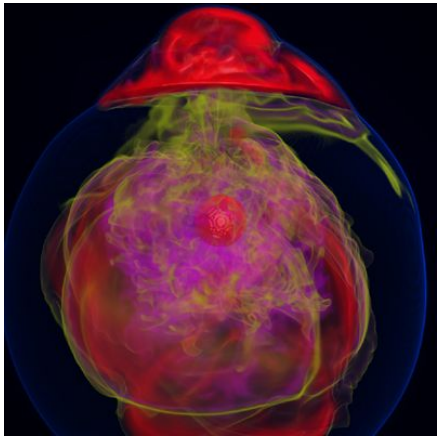
Shane Snyder
Argonne National Laboratory

ECP Annual Meeting ’20
Houston, TX

Why are we here?

Because I/O performance is addicting!

- Modern scientific computing applications access increasingly large and complex datasets to enable productive insights
- To support the diverse I/O needs of these applications, HPC systems are embracing deeper storage hierarchies and more elaborate layers of I/O libraries
- I/O analysis tools are of great help for navigating the complexity of HPC storage systems



Visualization of entropy in Terascale Supernova Initiative application. Image from Kwan-Liu Ma (UC Davis)

IBM Summit (OLCF)

Darshan: An application I/O characterization tool for HPC

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 typically enabled by default!) at many HPC facilities relevant to ECP
- 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?

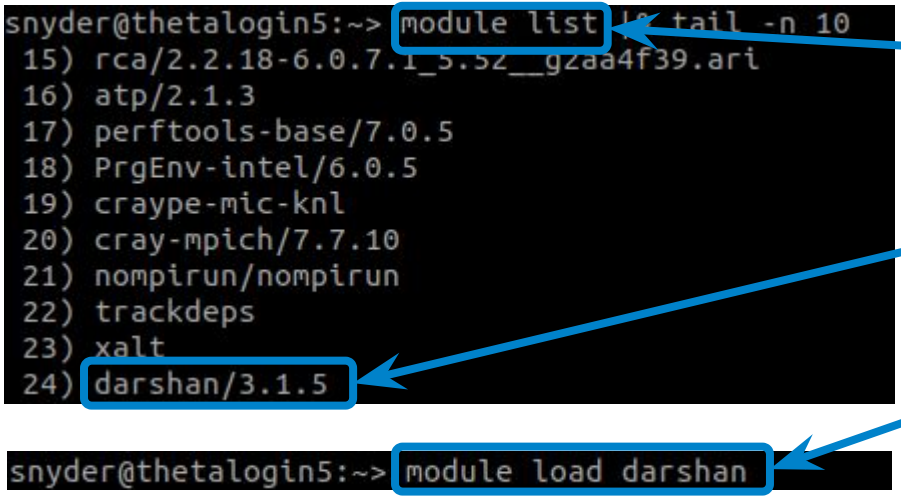
- Darshan inserts application I/O instrumentation at link-time (for static executables) or at runtime (for dynamic executables)
  - Darshan instrumentation traditionally only compatible with MPI programs*
- As app executes, Darshan records file access statistics for each process
  - Per-process memory usage is bounded to limit runtime overheads
- At app shutdown, collect, aggregate, compress, and write log data
  - Lean on MPI to reduce shared file records to a single record and to collectively write log data
- With a log generated, Darshan offers command line analysis tools for inspecting 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

* More on this later

Using Darshan on ECP platforms

Using Darshan on Theta (ALCF)

- Theta is a Cray XC40 system that uses static linking by default*
  - Static instrumentation enabled using Cray software module that injects linker options when compiling application



Use '**`module list`**' to confirm Darshan is actually loaded

Darshan 3.1.5 current default version available on Theta

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

* More on this shortly

Using Darshan on Theta (ALCF)

- OK, Darshan is loaded...now what?
  - Just compile and run your application!
  - Darshan inserts instrumentation directly into executable

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

```
snyder@thetalogin5:~> cd /lus/theta-fs0/logs/darshan/theta/
snyder@thetalogin5:/lus/theta-fs0/logs/darshan/theta> cd 2020/1/22
snyder@thetalogin5:/lus/theta-fs0/logs/darshan/theta/2020/1/22> ls |
grep snyder
snyder_mpi-io-test_id403177_1-22-74255-16539625359987666393_1.darshan
```

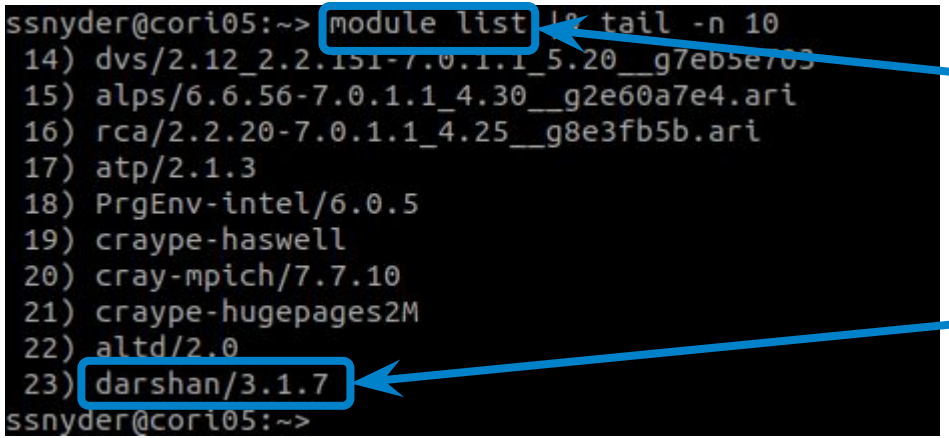
Darshan logs stored in a central directory -- **check site documentation for details**.

Logs further indexed using '*year/month/day*' the job executed. Pay attention to time zones to ensure you're looking in the right spot.

Log file name starts with the following pattern: '*username_exename_jobid*…'

Using Darshan on Cori (NERSC)

- Cori is also a Cray XC40 that has traditionally used static linking by default*
  - Using Darshan on Cori is essentially identical to to the process used on Theta



Use '**`module list`**' to confirm Darshan is actually loaded

Darshan 3.1.7 current default version available on Cori

* More on this shortly

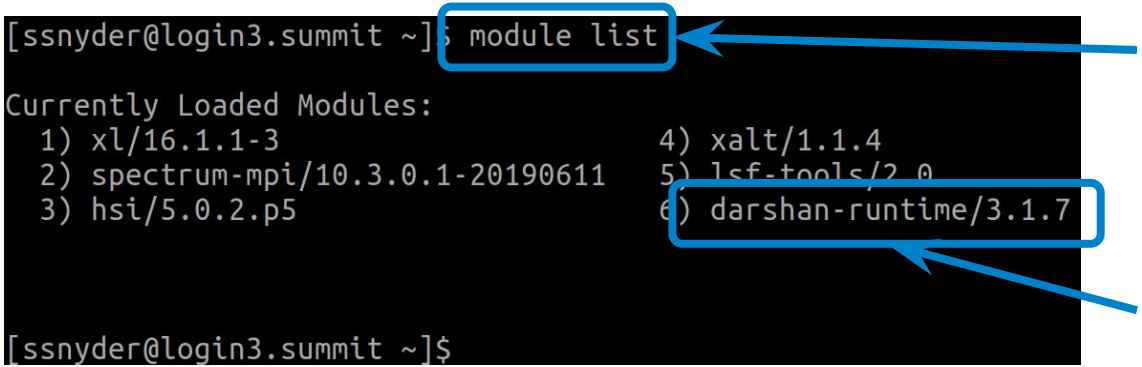
Using Darshan on Cori (NERSC)

- After compiling and running your application, look for your log files:

```
ssnyder@cori04:~> cd /global/cscratch1/sd/darshanlogs/
ssnyder@cori04:/global/cscratch1/sd/darshanlogs> cd 2020/1/28/
ssnyder@cori04:/global/cscratch1/sd/darshanlogs/2020/1/28>
ssnyder@cori04:/global/cscratch1/sd/darshanlogs/2020/1/28> ls ssnyder*
ssnyder_mpi-io-test_id27701820_1-28-41326-15632543236112513392_1.darshan
```

Using Darshan on Summit (OLCF)

- Summit is an IBM Power9-based system that uses dynamic linking by default
  - LD_PRELOAD mechanism used to interpose Darshan instrumentation libraries at runtime
  - Like Cori/Theta, software modules used to enable Darshan instrumentation



Summit also provides '**`module list`**' command

Darshan 3.1.7 is the default version on Summit.

Note: darshan-runtime and darshan-util are separate modules, with only darshan-runtime loaded by default

Using Darshan on Summit (OLCF)

- Since Summit uses `LD_PRELOAD`, there is no need to re-compile your application -- just run it and then look for your logs:

```
[ssnyder@login3.summit ~]$ cd /gpfs/alpine/darshan/summit/
[ssnyder@login3.summit summit]$ cd 2020/1/28/
[ssnyder@login3.summit 28]$
[ssnyder@login3.summit 28]$ ls ssnyder*
ssnyder_mpi-io-test_id857227_1-28-73628-59637088922742640 65_1.darshan
```

Note about dynamic linking on Cori/Theta

- In recent changes to the Cray programming environment, the default linking method was changed to dynamic
  - Cori adopted at the beginning of the year
  - Theta will be adopting soon
- We are working with ALCF and NERSC to accommodate these changes, focusing on a couple of options:
  - Use an `LD_PRELOAD` mechanism similar to that used on Summit
  - Use rpath mechanism to embed Darshan library path in dynamically-linked executable
- Goal is to rely on software modules on these systems to transparently enable/disable Darshan instrumentation regardless of the link method
  - In the meantime, may be necessary to use `LD_PRELOAD` manually to interpose Darshan

Analyzing Darshan logs

Analyzing Darshan logs

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

```
snyder@thetalogin5:/lus/theta-fs0/logs/darshan/theta/2020/1/22> cp snyder_
mpi-io-test_id403177_1-22-74255-16539625359987666393_1.darshan ~/tmp/
snyder@thetalogin5:/lus/theta-fs0/logs/darshan/theta/2020/1/22> cd ~/tmp/
snyder@thetalogin5:~/tmp> darshan-parser snyder_mpi-io-test_id403177_1-22-
74255-16539625359987666393_1.darshan
```

Copy the log file somewhere else for analysis

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

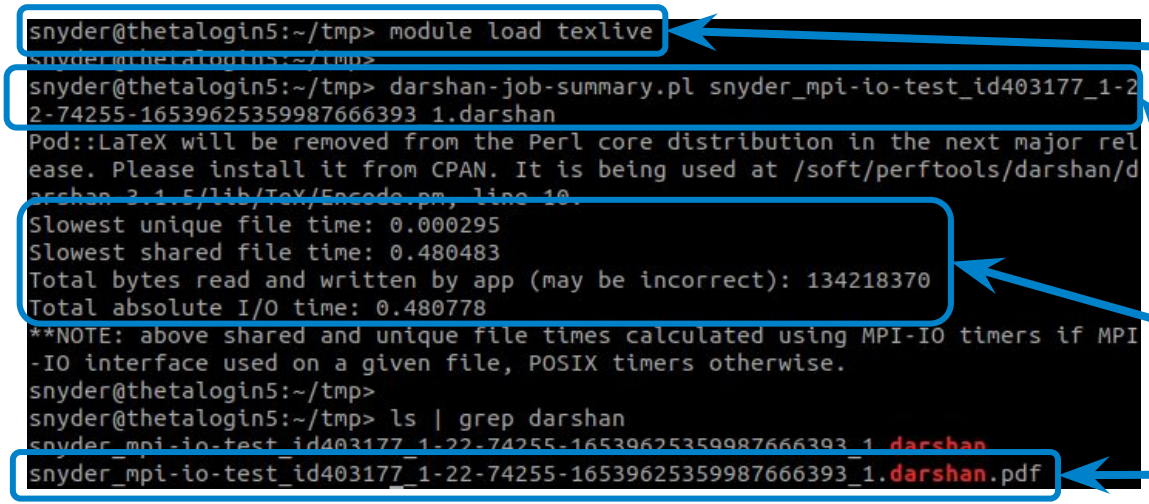
```
#<module>       <rank>  <record id>      <counter>       <value> <file name>
POSIX   -1      3675075178343058238     POSIX_OPENS     16      /gpfs/mira-home
POSIX   -1      3675075178343058238     POSIX_READS     4       /gpfs/mira-home
POSIX   -1      3675075178343058238     POSIX_WRITES    4       /gpfs/mira-home
POSIX   -1      3675075178343058238     POSIX_SEEKS     6       /gpfs/mira-home

MPI-IO  -1      3675075178343058238     MPIIO_INDEP_OPENS       8       /gpfs/mi
MPI-IO  -1      3675075178343058238     MPIIO_COLL_OPENS        0       /gpfs/mi
MPI-IO  -1      3675075178343058238     MPIIO_INDEP_READS       4       /gpfs/mi
MPI-IO  -1      3675075178343058238     MPIIO_INDEP_WRITES      4       /gpfs/mi
MPI-IO  -1      3675075178343058238     MPIIO_COLL_READS        0       /gpfs/mi
```

Modules use a common format for printing counters, indicating the corresponding module, rank, 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



On Theta, texlive module is needed for generating PDF summaries -- may not be needed on other systems

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

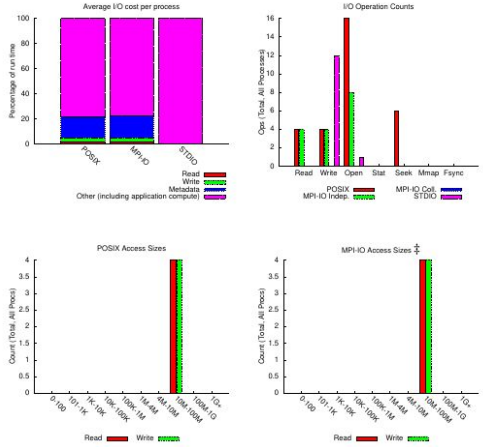
A few simple statistics (total I/O time and volume) are output on command line

Output PDF file name based on Darshan log file name

Analyzing Darshan logs

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



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.

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

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

Analyzing Darshan logs

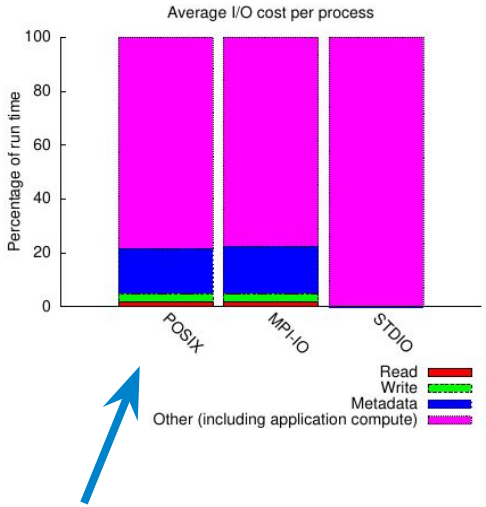
PDF header contains some high-level information on the job execution

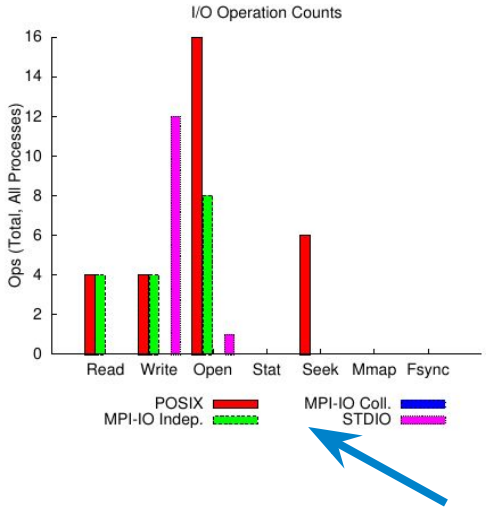
jobid: 403177	uid: 31074	nprocs: 4	runtime: 2 seconds

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





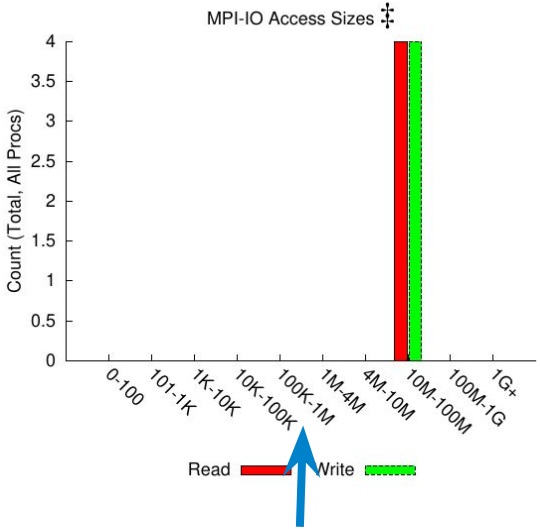
Across main I/O interfaces, how much time was spent reading, writing, doing metadata, or computing?

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

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

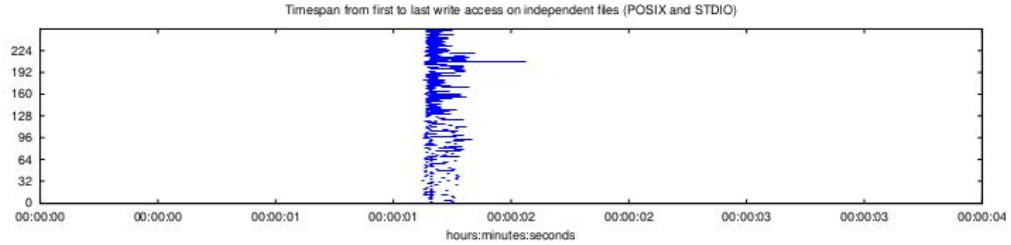
type	number of files	avg. size	max size
total opened	2	33M	64M
read-only files	0	0	0
write-only files	1	642	642
read/write files	1	64M	64M
created files	2	33M	64M

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

Analyzing Darshan logs

reads

writes



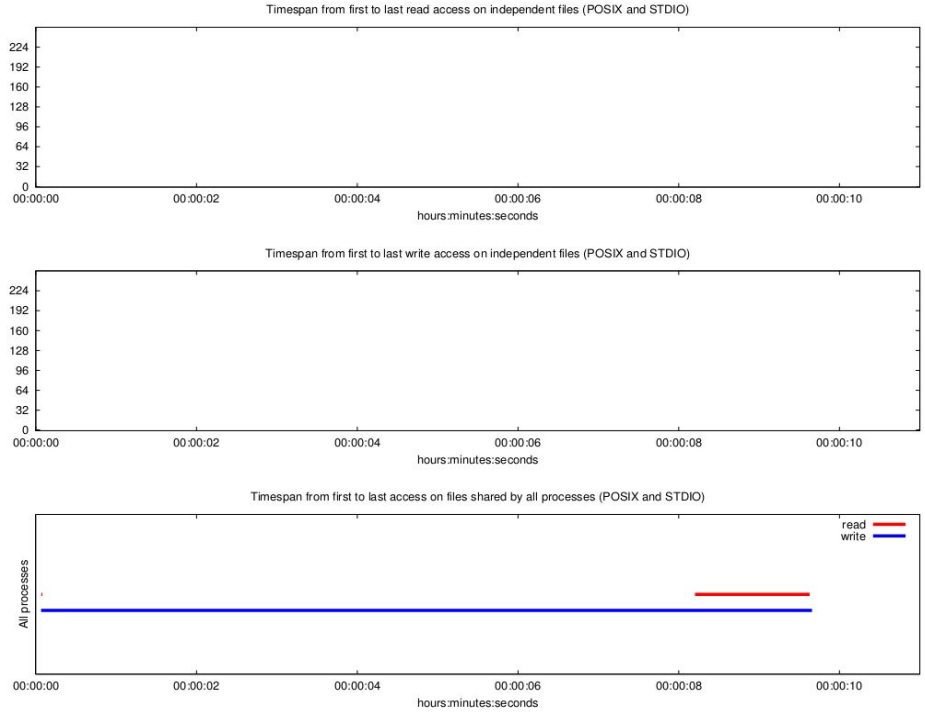
Darshan can also provide basic timing bounds for read/write activity, both for independent file access patterns (illustrated) or for shared file access patterns

What if we want more details?

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!

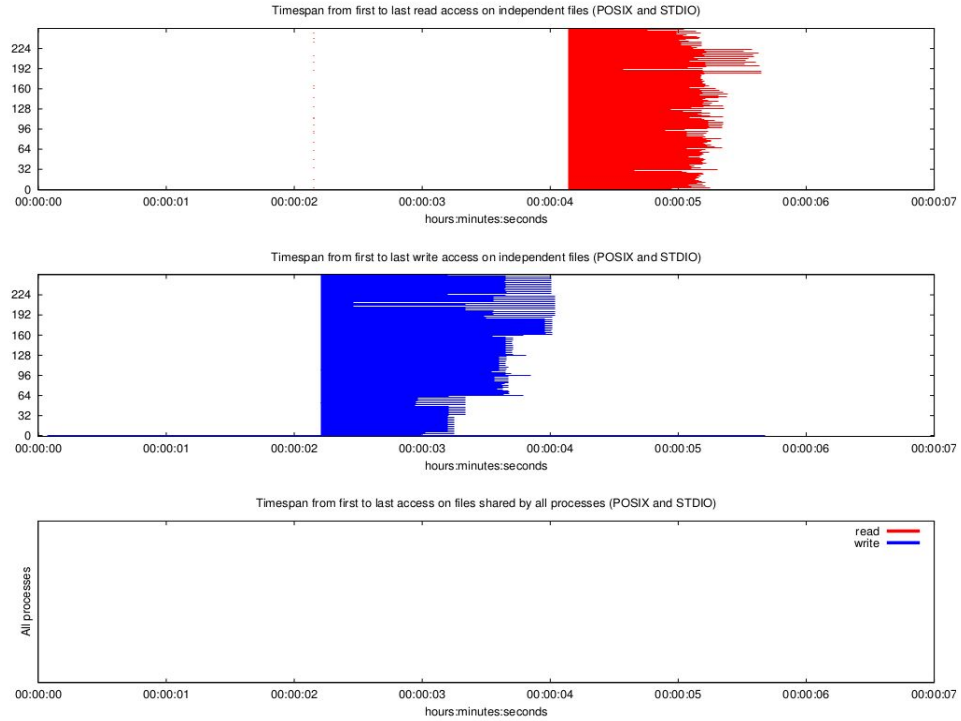
Disabling reductions of shared records



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

This is as a result of Darshan's decision to aggregate shared file records into a single file record representing all processes' access information

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

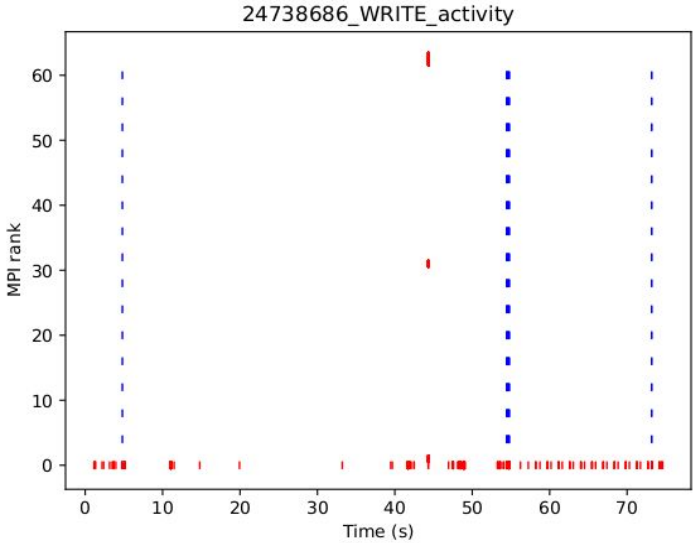
- Darshan's DXT module can be enabled at runtime for users wishing to capture detailed I/O traces for MPI-IO and POSIX interfaces
  - Fine-grained trace data comes at cost of larger per-process memory overheads
  - Set the `DXT_ENABLE_IO_TRACE` environment variable to enable
- `darshan-dxt-parser` can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

What's new with Darshan?

DXT trace triggers

Available in Darshan 3.1.8

- DXT traces can enable fine-grained insights into application I/O behavior, but at the cost of increased memory overheads
- To address this, we have integrated “trace triggers” into DXT to provide users with more control over which files Darshan will trace at runtime
  - *Static trace triggers*: use regex matching on static information related to file access to control whether a file is traced:
    - File name matching
    - Process rank matching
  - *Dynamic trace triggers*: use internal file access statistics gathered by Darshan to control whether a file is traced:
    - Frequent small I/O accesses
    - Frequent unaligned I/O accesses

DXT trace triggers

Available in Darshan 3.1.8

- Users inform Darshan about their desired trace triggers using a text file, which can specify 1 or more triggers to be used at runtime:

```
shane@shane-x1-carbon ~ $ export DXT_TRIGGER_CONF_PATH=dxt_triggers
shane@shane-x1-carbon ~ $
shane@shane-x1-carbon ~ $ cat dxt_triggers
FILE .h5$
FILE ^/scratch
RANK [1-2]
SMALL_IO .5
UNALIGNED_IO .5
```

Set this environment variable to inform Darshan about the trace triggers file

Text-based descriptions of each trigger, one per-line

DXT trace triggers

Available in Darshan 3.1.8

- Users inform Darshan about their desired trace triggers using a text file, which can specify 1 or more triggers to be used at runtime:

```
shane@shane-x1-carbon ~ $ export DXT_TRIGGER_CONF_PATH=dxt_triggers
shane@shane-x1-carbon ~ $
shane@shane-x1-carbon ~ $ cat dxt_triggers
FILE .h5$
FILE ^/scratch
RANK [1-2]
SMALL_IO .5
UNALIGNED_IO .5
```

Only trace files ending in prefix
‘.h5’ or with path prefix ‘/scratch’

DXT trace triggers

Available in Darshan 3.1.8

- Users inform Darshan about their desired trace triggers using a text file, which can specify 1 or more triggers to be used at runtime:

```
shane@shane-x1-carbon ~ $ export DXT_TRIGGER_CONF_PATH=dxt_triggers
shane@shane-x1-carbon ~ $
shane@shane-x1-carbon ~ $ cat dxt_triggers
FILE .h5$
FILE ^/scratch
RANK [1-2]
SMALL_IO .5
UNALIGNED_IO .5
```

Only trace files accessed by ranks 1-2

DXT trace triggers

Available in Darshan 3.1.8

- Users inform Darshan about their desired trace triggers using a text file, which can specify 1 or more triggers to be used at runtime:

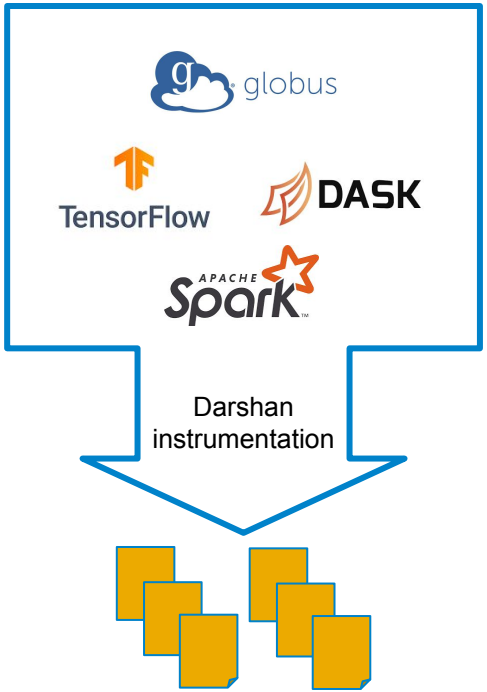
```
shane@shane-x1-carbon ~ $ export DXT_TRIGGER_CONF_PATH=dxt_triggers
shane@shane-x1-carbon ~ $
shane@shane-x1-carbon ~ $ cat dxt_triggers
FILE .h5$
FILE ^/scratch
RANK [1-2]
SMALL_IO .5
UNALIGNED_IO .5
```

Only trace files that had greater than 50% small I/O accesses or greater than 50% unaligned I/O accesses

Non-MPI instrumentation support

WIP-ish (experimental version available in 3.2.0-pre1)

- To support an evolving HPC software landscape, we have broken Darshan's dependence on MPI to allow instrumentation in new contexts:
  - non-MPI computing frameworks (e.g., Spark, TensorFlow)
  - Inter- and intra-site file transfer utilities (e.g., Globus, cp)
  - General serial applications
- This required significant modifications to Darshan:
  - Build logic for detecting whether a compiler supports MPI
  - Refactoring of Darshan core functionality to make MPI optional
  - Definition of shared library constructor/destructor attributes to handle initialization/shutdown of the Darshan library*



* Side effect: this instrumentation method only works for dynamically linked executables

Non-MPI instrumentation support

WIP-ish (experimental version available in 3.2.0-pre1)

- To build Darshan with a non-MPI compiler (e.g., gcc), use the following arguments when configuring: '`--without-mpi CC=gcc`'
  - Other compilers (e.g., clang, llvm) possible, but gcc is recommended

- When running your app, you must set the `DARSHAN_ENABLE_NONMPI` environment variable (in addition to `LD_PRELOAD`):

```
shane@shane-x1-carbon ~/software/spark (master) $ export DARSHAN_ENABLE_NONMPI=1
shane@shane-x1-carbon ~/software/spark (master) $ export LD_PRELOAD=/home/shane/s
oftware/darshan/darshan-dev/install/lib/libdarshan.so
shane@shane-x1-carbon ~/software/spark (master) $ ./bin/spark-submit examples/src
/main/python/wordcount.py war-and-peace.txt
```

Non-MPI instrumentation support

WIP-ish (experimental version available in 3.2.0-pre1)

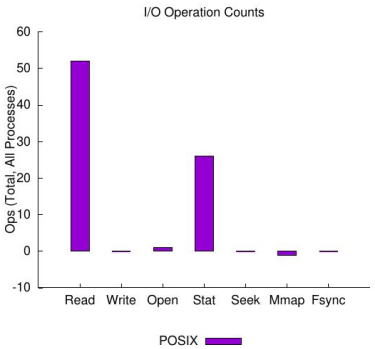
```
shane@shane-x1-carbon ~/software/spark$ ./bin/spark-submit examples/src/main/python/wordcount.py war-and-peace.txt
shane@shane-x1-carbon ~/software/spark$ ls ~/software/darshan/darshan-logs/2019/11/19/
shane_bash_id5167_11-19-33272-7035573431850780836_1574180073.darshan
shane_bash_id5218_11-19-33273-7035573431850780836_1574180074.darshan
shane_dirname_id5168_11-19-33272-7035573431850780836_1574180073.darshan
shane_dirname_id5171_11-19-33272-7035573431850780836_1574180073.darshan
shane_dirname_id5174_11-19-33272-7035573431850780836_1574180073.darshan
shane_git_id5164_11-19-33269-7035573431850780836_1574180070.darshan
shane_git_id5350_11-19-33280-7035573431850780836_1574180081.darshan
shane_java_id5167_11-19-33272-7035573431850780836_1574180081.darshan
shane_java_id5177_11-19-33272-7035573431850780836_1574180073.darshan
shane_python_id5224_11-19-33273-7035573431850780836_1574180081.darshan
shane_python_id5283_11-19-33277-7035573431850780836_1574180080.darshan
shane_rm_id5327_11-19-33279-7035573431850780836_1574180080.darshan
shane_rm_id5343_11-19-33279-7035573431850780836_1574180080.darshan
shane_rm_id5346_11-19-33280-7035573431850780836_1574180081.darshan
shane_rm_id5347_11-19-33280-7035573431850780836_1574180081.darshan
shane_rm_id5348_11-19-33280-7035573431850780836_1574180081.darshan
shane_sed_id5165_11-19-33269-7035573431850780836_1574180070.darshan
shane_sed_id5351_11-19-33280-7035573431850780836_1574180081.darshan
```

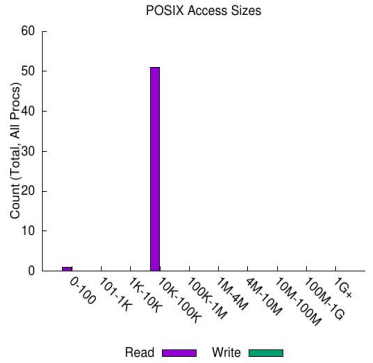
This simple Spark example generated a lot of logs!

Non-MPI instrumentation support

WIP-ish (experimental version available in 3.2.0-pre1)

```
shane@shane-x1-carbon ~/software/spark$ ./bin/spark-submit examples/src/
shane@shane-x1-carbon ~/software/spark$ ls ~/software/darshan/darshan-lo
shane_bash_id5167_11-19-33272-7035573431850780836_1574180073.darshan
shane_bash_id5218_11-19-33273-7035573431850780836_1574180074.darshan
shane_dirname_id5168_11-19-33272-7035573431850780836_1574180073.darshan
shane_dirname_id5171_11-19-33272-7035573431850780836_1574180073.darshan
shane_dirname_id5174_11-19-33272-7035573431850780836_1574180073.darshan
shane_git_id5164_11-19-33269-7035573431850780836_1574180070.darshan
shane_git_id5350_11-19-33280-7035573431850780836_1574180081.darshan
shane_java_id5167_11-19-33272-7035573431850780836_1574180081.darshan
shane_java_id5177_11-19-33272-7035573431850780836_1574180073.darshan
shane_python_id5224_11-19-33273-7035573431850780836_1574180081.darshan
shane_python_id5283_11-19-33277-7035573431850780836_1574180080.darshan
shane_rm_id5327_11-19-33279-7035573431850780836_1574180080.darshan
shane_rm_id5343_11-19-33279-7035573431850780836_1574180080.darshan
shane_rm_id5346_11-19-33280-7035573431850780836_1574180081.darshan
shane_rm_id5347_11-19-33280-7035573431850780836_1574180081.darshan
shane_rm_id5348_11-19-33280-7035573431850780836_1574180081.darshan
shane_sed_id5165_11-19-33269-7035573431850780836_1574180070.darshan
shane_sed_id5351_11-19-33280-7035573431850780836_1574180081.darshan
```





Focusing analysis on the Java executable that does all of the I/O for this example

Detailed HDF5 instrumentation module

WIP-ish (available in branch dev-detailed-hdf5-mod, include in 3.2.0)

- Darshan has traditionally offered very little in the ways of HDF5 instrumentation, providing only basic statistics about HDF5 file open calls
- But, understanding and improving the I/O behavior of HDF5 workloads is critical to the performance of many current HPC applications
  - HDF5 provides a convenient abstract data model for scientific data, but it obscures how HDF5 storage constructs interact with lower layers of the I/O software stack (i.e., MPI-IO and POSIX levels)
- We have developed a new implementation of the HDF5 module that allows for better understanding of HDF5 I/O behavior from file- and dataset-level perspectives

Detailed HDF5 instrumentation module

WIP-ish (awaiting merge, will include in 3.2.0)

- We split the original HDF5 module into two instrumentation modules: H5F (for HDF5 files) and H5D (for HDF5 datasets), each independently recording instrumentation records

- H5F module 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
```

Detailed HDF5 instrumentation module

WIP-ish (awaiting merge, will include in 3.2.0)

- H5D module highlights:
  - Operation counts:
    - open/create
    - read/write
    - flush
  - Total bytes read/written
  - Access size histograms
  - Dataspace selection types
    - Points
    - Regular hyperslab
    - Irregular hyperslab
  - Dataspace total dimensions, points
  - 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/sh
H5D -1  7600138186531619366 H5D_READS   16  /home/sh
H5D -1  7600138186531619366 H5D_WRITES  16  /home/sh
H5D -1  7600138186531619366 H5D_FLUSHES 0   /home/sh
H5D -1  7600138186531619366 H5D_BYTES_READ  4194304
H5D -1  7600138186531619366 H5D_BYTES_WRITTEN   4194
H5D -1  7600138186531619366 H5D_RW_SWITCHES 4   /hom
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
```

darshan-util Python bindings

WIP (tentatively planning to include in 3.2.0)

- The only existing interface to Darshan logs is via the darshan-util C library
  - Non-C log file analysis tools require a costly conversion to text format (using darshan-parser) which the tool must then find a way to ingest
- To address this, we are developing Python bindings for the darshan-util library that simplify the interfacing of Darshan analysis tools with log data
  - Use Python CFFI module to provide Python bindings to the native darshan-utils C API
  - Organize Darshan log data using native Python constructs (e.g., dictionaries) to allow simple and efficient access to log data
- We are hopeful this will lead to more productive Darshan log file analysis tools that can be distributed with Darshan

Wrapping up

- We've covered a lot of ground in a short amount of time, but don't be overwhelmed...
  - No one is expected to be an expert at the end of this session!
  - Instead, we just want to equip you with resources you can consult to start to think about understanding and improving I/O performance using Darshan
  - Don't hesitate to reach out to us if you have questions, comments, or suggestions

- 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://xgitlab.cels.anl.gov/darshan/darshan

Thanks to all for attending!

All comments/questions are welcome!