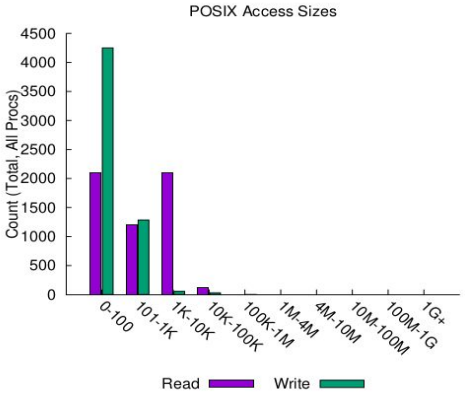
April 22, 2020

Argonne NATIONAL LABORATORY

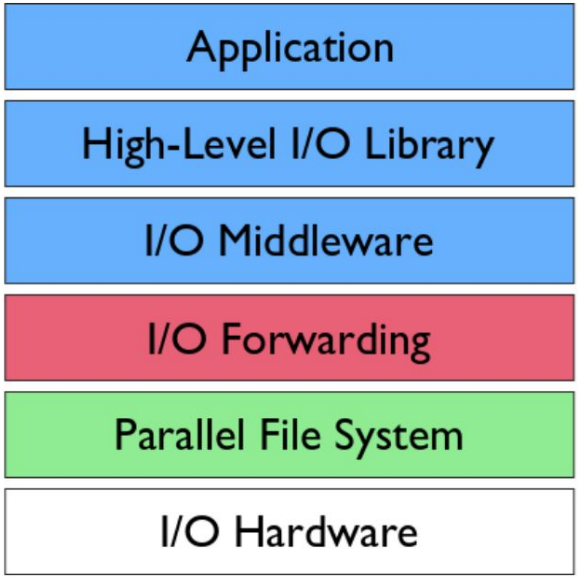
HPC I/O characterization with Darshan



Shane Snyder
Argonne National Laboratory

Motivation

- I/O performance has long been a critical obstacle to scientific productivity for HPC apps
- I/O systems have resorted to increasingly complex designs to keep up with app I/O needs
  - Deep stacks of I/O libraries and middleware to optimize workloads
  - Growing storage hierarchies on the backend offering conventional (HDD) and emerging (NVM) storage devices
- Effective I/O characterization tools can help users and system admins navigate this complexity to better understand HPC I/O behavior



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

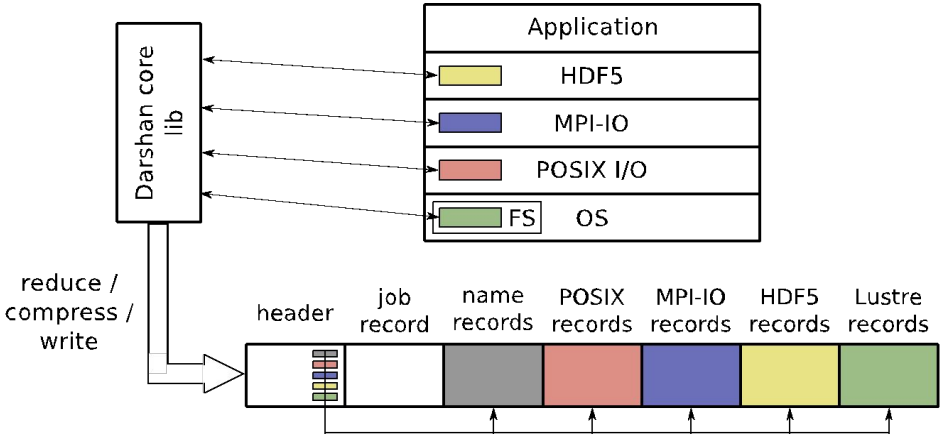
How does Darshan work?

- 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

How does Darshan work?

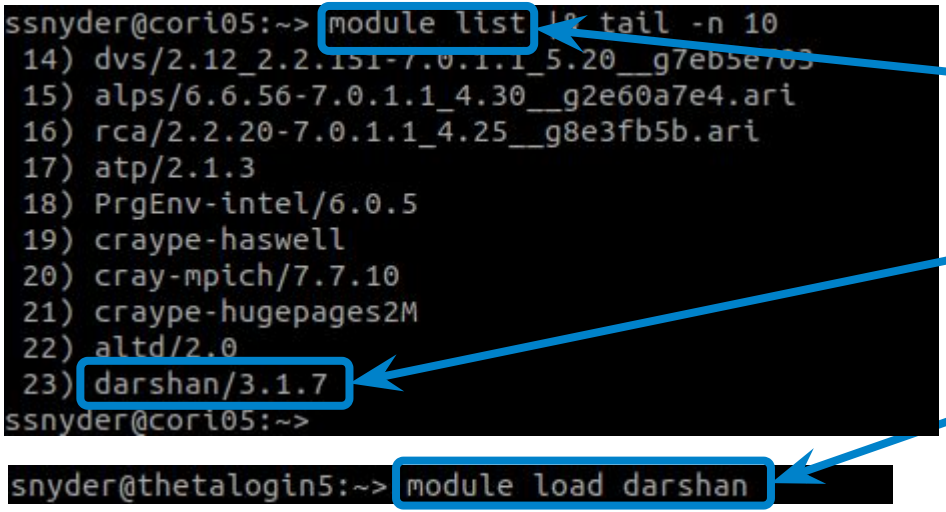
- Darshan's modular architecture centered around a *core library* and *instrumentation modules*:
  - *core library*: init/finalize library, coordinate with modules at runtime, reduce/compress/write log file
  - *instrumentation module*: captures data from some source, typically by defining wrappers for I/O functions of interest
- Self-describing file format to index and find data from different modules



Using Darshan on a production HPC system

Using Darshan on Cori (NERSC)

- Cori is a Cray XC40 system that has Darshan enabled by default
  - Darshan has traditionally integrated directly into Cray compiler wrappers using the software module system*, as shown below



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

Darshan 3.1.7 current default version available on Cori

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

* More on this shortly

Using Darshan on Cori (NERSC)

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

- After the application terminates, 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
```

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)

- Recent modifications to the Cray programming environment have resulted in a change in the default linking method from static to dynamic
  - Darshan integration into Cray compiler wrappers was traditionally specific to statically-linked executables
  - In response to this, we have re-worked our Cray software module to work in both static and dynamic linking cases and are working with Cray facilities to update existing deployments

- In the meantime, Darshan instrumentation can still be enabled manually at runtime using LD_PRELOAD
  - I.e., '`export LD_PRELOAD=/path/to/darshan/libdarshan.so`' prior to running the application

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

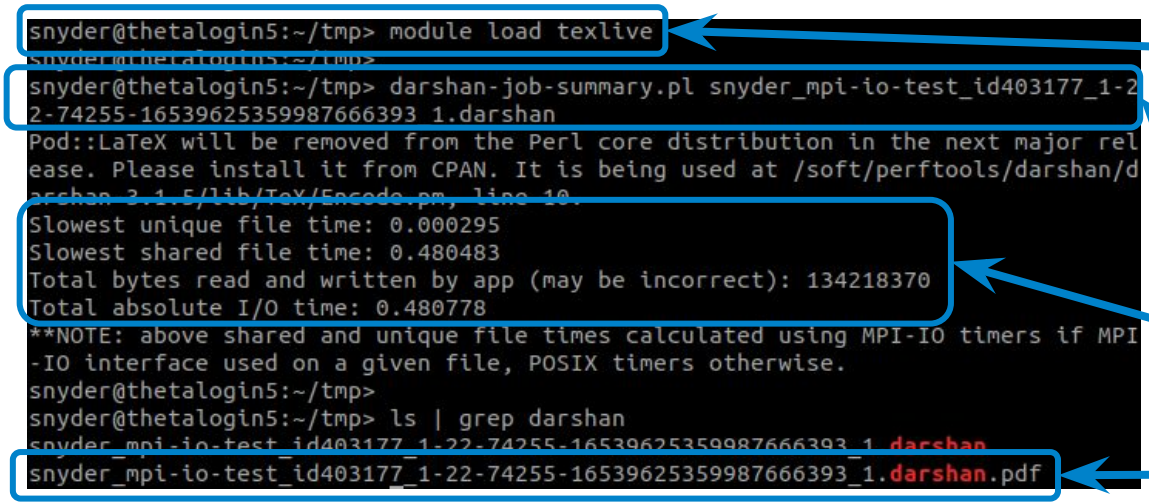
```
#<module>       <rank>  <record id>      <counter>       <value> <file name>
POSIX   -1      3675075178343058238     POSIX_OPENS     16      /gpfs/mira-home
POSIX   -1      3675075178343058238     POSIX_READS     4       /gpfs/mira-home
POSIX   -1      3675075178343058238     POSIX_WRITES    4  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

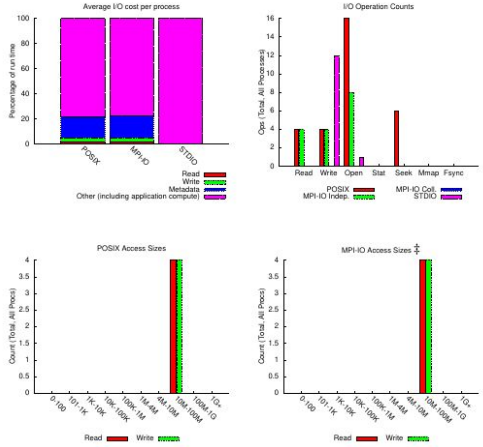
A few simple statistics (total I/O time and volume) are output on command line

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

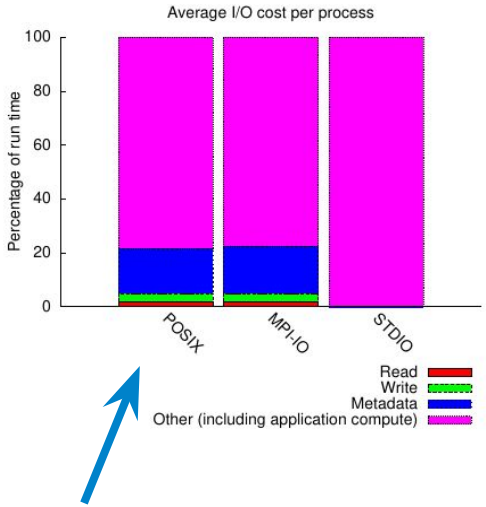
PDF header contains some high-level information on the job execution

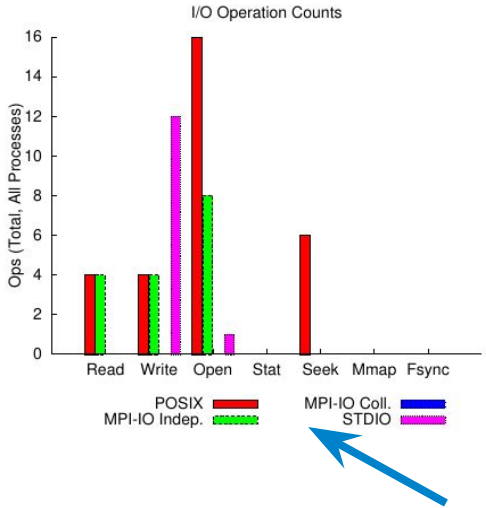
jobid: 403177	uid: 31074	nprocs: 4	runtime: 2 seconds

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





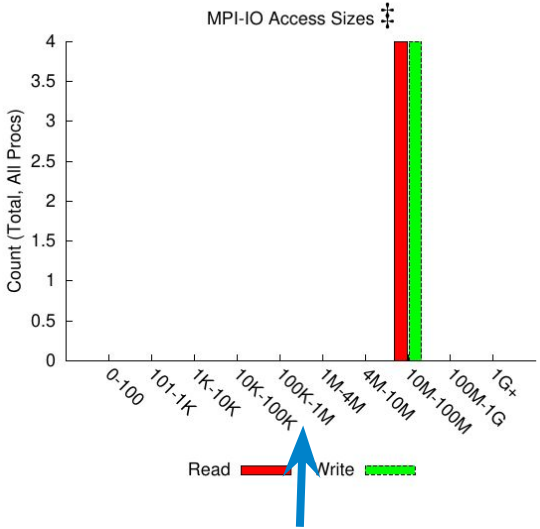
Across main I/O interfaces, how much time was spent reading, writing, doing metadata, or computing?

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)

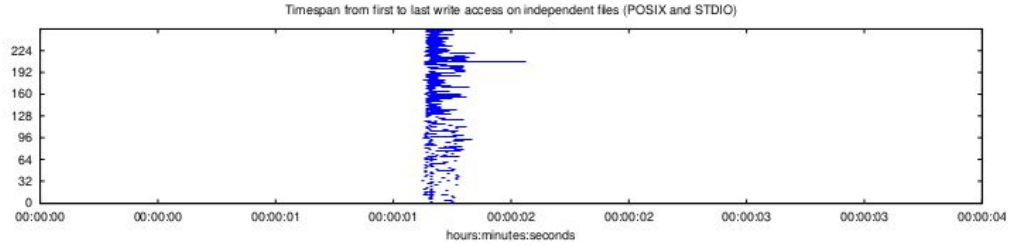
type	number of files	avg. size	max size
total opened	2	33M	64M
read-only files	0	0	0
write-only files	1	642	642
read/write files	1	64M	64M
created files	2	33M	64M

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

Detailed I/O traces with DXT

Obtaining fine-grained traces with DXT

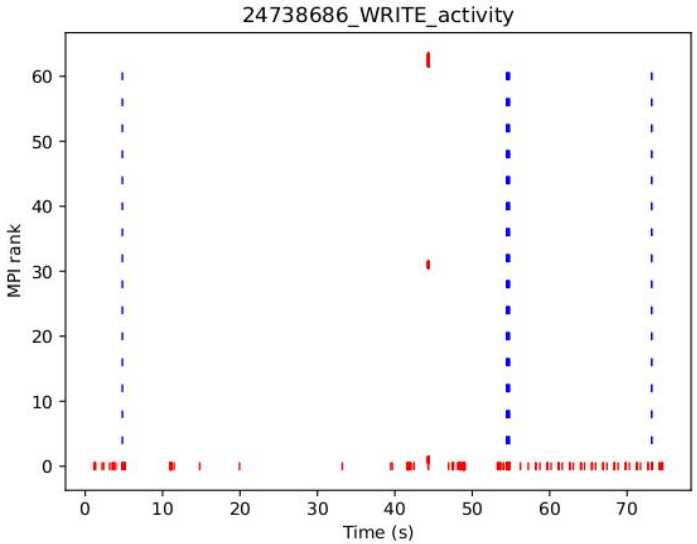
- Darshan's DXT module can be enabled at runtime for users wishing to capture detailed I/O traces for MPI-IO and POSIX interfaces
  - Fine-grained trace data comes at cost of larger per-process memory overheads
  - Set the `DXT_ENABLE_IO_TRACE` environment variable to enable
- `darshan-dxt-parser` can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



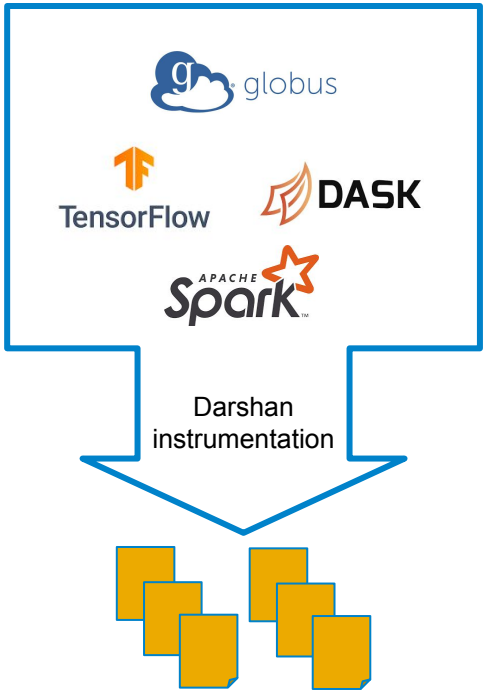
Provides details on each I/O operation issued by each rank, providing a complete picture of which ranks are performing I/O and how long they are spending on I/O

New happenings in Darshan: non-MPI support

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)

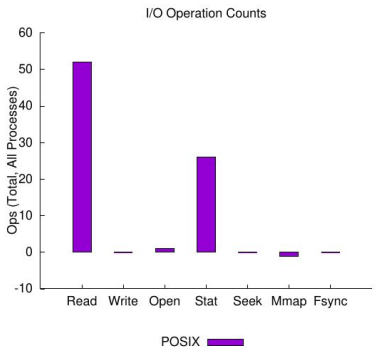
```
shane@shane-x1-carbon ~/software/spark$ ./bin/spark-submit examples/src/main/python/wordcount.py war-and-peace.txt
shane@shane-x1-carbon ~/software/spark$ ls ~/software/darshan/darshan-logs/2019/11/19/
shane_bash_id5167_11-19-33272-7035573431850780836_1574180073.darshan
shane_bash_id5218_11-19-33273-7035573431850780836_1574180074.darshan
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
```

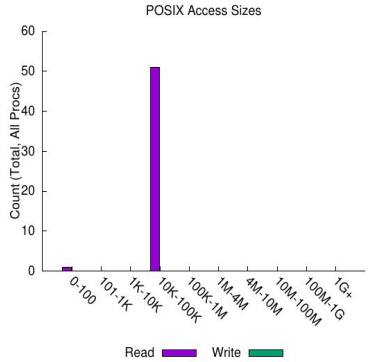
This simple Spark example generated a lot of logs!

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

Wrapping up

- We look forward to using Darshan to help understand I/O characteristics of HEP workflows and to help optimize their performance on upcoming HPC systems
  - ➢ Let us know how we can help!

- 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