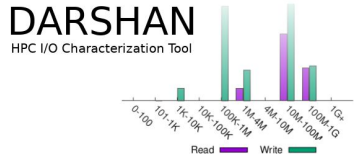
Interpreting Darshan Logs

Phil Carns, Kevin Harms, Rob Latham, Rob Ross, Shane Snyder
Argonne National Laboratory

ECP Annual Meeting

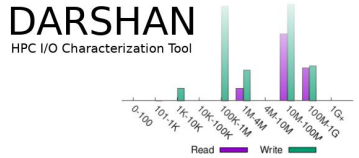
April 15, 2021

Purpose of this session



- In yesterday's BoF, we covered background on the Darshan I/O characterization tool and basics on how to use it on HPC systems
- Today, we focus primarily on how to interpret Darshan log data to help equip users with tools and best practices for understanding application I/O behavior
  - Traditional Darshan analysis tools, developed mostly using Darshan's C-based darshan-util library
  - PyDarshan, a recently developed Python interface to Darshan log files that allows for simpler development of Darshan log file analysis tools
- We have given attendees the option to provide Darshan logs of interest to our team so that we can analyze them in this session
  - We also have a few examples we can present that demonstrate how Darshan can be used to enable different I/O insights

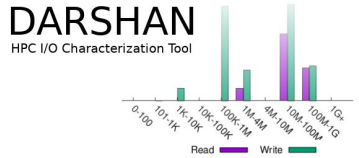
Session materials



- We have assembled materials for this session in a repo that may be of use to attendees:
  - Darshan-3.3.0-pre1 pre-release, containing up-to-date source code for Darshan software and PyDarshan log analysis package
  - Darshan log file analysis examples, including background details, job submission scripts, etc.
  - Jupyter notebooks demonstrating usage of PyDarshan for analyzing Darshan log files

https://github.com/darshan-hpc/ecpam-21-materials.git

Installing darshan-util and traditional Darshan log analysis utilities



Manual installation of darshan-util and log utilities:

(0a.) `tar -xzvf darshan-3.3.0-pre1.tar.gz`

(0b.) `git clone -b darshan-3.3.0-pre1 https://xgitlab.cels.anl.gov/darshan/darshan.git darshan-3.3.0-pre1`

1. `cd darshan-3.3.0-pre1/darshan-util`
2. `./configure --enable-shared --prefix=<install_prefix>`
3. `make install`

darshan-util library and corresponding binaries installed at given install prefix

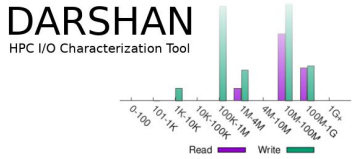
Spack installation of darshan-util and log utilities:

(0.) Make sure Spack repo is up-to-date

1. `spack install darshan-util@darshan-3.3.0-pre1`
2. `spack load -r darshan-util@darshan-3.3.0-pre1`

Darshan-util library and corresponding binaries installed at '`spack location -i darshan-util`', corresponding env vars set (`PATH`, `LD_LIBRARY_PATH`, etc.)

Installing PyDarshan log analysis package



PyPI installation of PyDarshan:

- Can install from PyPI repository, https://pypi.org/project/darshan/

1. `pip3 install --user darshan`

Manual installation of PyDarshan using `setup.py`:

- Run the following from the darshan-3.3.0-pre1 top-level directory

1. `cd darshan-util/pydarshan`
2. `pip3 install -r requirements.txt`
3. `python3 setup.py install --user`

Finding `libdarshan-util.so`:

- Python package must be able to find `libdarshan-util.so`

1. `module load darshan`
2. `export LD_LIBRARY_PATH=<install_prefix>/lib`
3. `export PKG_CONFIG_PATH=<install_prefix>/lib/pkgconfig`

Not necessary for PyPI wheel distributions

Traditional Darshan analysis tools

Overview

- "*Traditional Darshan analysis tools*" refers to the command utilities that have always been available in the darshan-util package
  - They produce static pdf summaries and column-oriented text statistics
- These are tried and true tools, but there is no interactive exploration unless you are comfortable manipulating text data yourself
- Step 1 (actually this probably applies to any analysis method): **find your log**
  - The logs themselves are not machine-dependent: copy them wherever you want for analysis
  - Sometimes it is easiest to analyze them on your own laptop
  - Darshan analysis utilities are backwards compatible for old logs

Finding your log file

The “darshan-config –log-path” command will show you where to look for logs on your system.

NOTE: it may report the name of an environment variable (e.g. $HOME) depending on your installation.

System installs usually have a year/month/day hierarchy in the log directory

In this example, I want to copy all of my logs (prefixed with username) for a specific day.

```
[carns@thetalogin5 ~]$
[carns@thetalogin5 ~]$ darshan-config --log-path
/lus/theta-fs0/logs/darshan/theta
[carns@thetalogin5 ~]$
[carns@thetalogin5 ~]$ cp `darshan-config --log-path`/2021/4/6/${USER}* .
[carns@thetalogin5 ~]$
[carns@thetalogin5 ~]$ ls *.darshan
carns_ior_id510256_4-6-49164-15971764111489070954_1617716366.darshan
carns_ior_id510260_4-6-49778-9578554294911421599_1617716983.darshan
[carns@thetalogin5 ~]$
```

user name

executable name

job ID

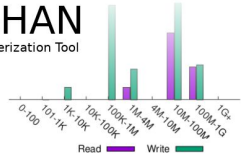
NOTE: the log date will be determined by the compute node time zone, not your time zone!

darshan-job-summary

- Darshan-job-summary is a typical starting point for understanding a Darshan log.
  - It produces a PDF that you can save/print/email etc: a good conversation starter!
  - Good for getting the general “feel” of what the I/O is like in an application
- Limitations:
  - Presentation is static
  - It requires a latex and gnuplot tool chain

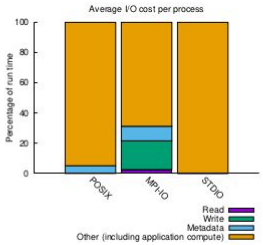
```
carns@carns-x1-7g ~> ls *.darshan
carns_ior_id510256_4-6-49164-15971764111489070954_1617716366.darshan
carns@carns-x1-7g ~>
carns@carns-x1-7g ~> darshan-job-summary.pl carns_ior_id510256_4-6-49164-1597176
4111489070954_1617716366.darshan
carns@carns-x1-7g ~>
carns@carns-x1-7g ~> ls *.darshan.pdf
carns_ior_id510256_4-6-49164-15971764111489070954_1617716366.darshan.pdf
```

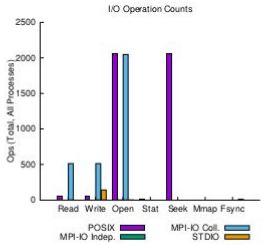
darshan-job-summary example

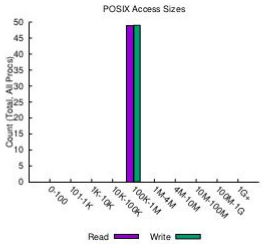


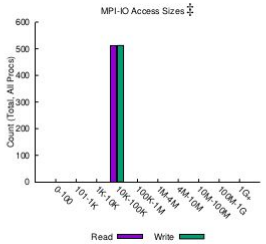
jobid: 510256	uid: 4279	nprocs: 512	runtime: 2 seconds

I/O performance *estimate* (at the MPI-IO layer): transferred 94.1 MiB at 148.75 MiB/s
I/O performance *estimate* (at the STDIO layer): transferred 0.0 MiB at 1.34 MiB/s









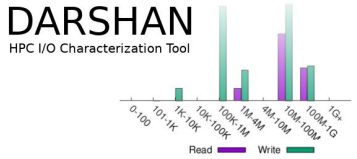
darshan-parser

- This tool extracts everything* from a Darshan log and displays it in text format.
  - Provides more information than you can find in job-summary, with per-file granularity
  - You can grep/sort/awk through it in text format, or make your own analysis scripts
- Limitations:
  - Parsing text isn't very fast, especially if you are mining many logs
  - Text parsing is a little fragile, and requires learning more details about Darshan counters

```
carns@carns-x1-7g ~>
carns@carns-x1-7g ~> darshan-parser example.darshan |wc -l
351
carns@carns-x1-7g ~>
carns@carns-x1-7g ~> darshan-parser example.darshan |head
# darshan log version: 3.21
# compression method: ZLIB
# exe: ./ior -w -r -o /projects/CSC250STDM12/carns/ior.dat -c -Z -b 1000000 -t 1
000000 -s 1 -a MPIIO
# uid: 4279
# jobid: 510260
# start_time: 1617716978
# start_time_asci: Tue Apr  6 09:49:38 2021
# end_time: 1617716982
# end_time_asci: Tue Apr  6 09:49:42 2021
# nprocs: 512
carns@carns-x1-7g ~>
carns@carns-x1-7g ~> darshan-parser example.darshan |grep POSIX_BYTES_WRITTEN
POSIX   -1      3588668995759596825     POSIX_BYTES_WRITTEN     512000000       /
projects/CSC250STDM12/carns/ior.dat     /       overlay
carns@carns-x1-7g ~>
```

* Well, pretty close anyway. We'll learn about traces in a minute.

darshan-dxt-parser



- What if you want more than just statistics, but an actual trace of each I/O operation?
- Re-run your job with "export DXT_ENABLE_IO_TRACE=1" in your job script
  - This will capture the most detail possible with Darshan
  - Includes access sizes, offsets, and start and end of each I/O operation
- Limitations:
  - Not enabled by default (note in the above example that it is enabled via explicit runtime environment variable; this will make Darshan produce larger log files than a "normal" run)
  - Not many command line tools to visualize results

```
carns@carns-x1-7g ~>
carns@carns-x1-7g ~> darshan-dxt-parser example.darshan |wc -l
5087
```

```
# ************************************************
# DXT_POSIX module data
# ************************************************

# DXT, file_id: 3588668995759596825, file_name: /projects/CSC250STDM12/carns/ior.dat
# DXT, rank: 0, hostname: nid03824
# DXT, write_count: 489, read_count: 425
# DXT, mnt_pt: /, fs_type: overlay
# Module    Rank  Wt/Rd  Segment          Offset       Length    Start(s)      End(s)
 X_POSIX       0  write        0               0      1048576      0.1681      0.1730
 X_POSIX       0  write        1         1048576      1048576      0.1755      0.1794
 X_POSIX       0  write        2         2097152      1048576      0.1816      0.1857
 X_POSIX       0  write        3         3145728      1048576      0.1880      0.1916
 X_POSIX       0  write        4         4194304      1048576      0.1939      0.1977
 X_POSIX       0  write        5         5242880      1048576      0.2004      0.2040
```

dxt_analyzer

DARSHAN
HPC I/O Characterization Tool

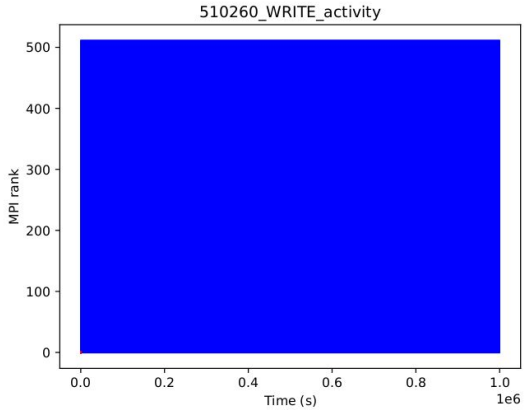
- dxt_analyzer can be used to visualize trace data
  - What ranks did I/O?
  - When exactly (in the job's run time) did they do it?
- Limitations:
  - Does not show all information captured by the trace (no offset information or individual response times)

```
carns@carns-x1-7g ~> darshan-dxt-parser example.darshan > example.darshan.dxt
carns@carns-x1-7g ~>
carns@carns-x1-7g ~> dxt_analyzer.py -i example.darshan.dxt
carns@carns-x1-7g ~>
carns@carns-x1-7g ~> ls *.pdf
dxt_plot.pdf
carns@carns-x1-7g ~>
```

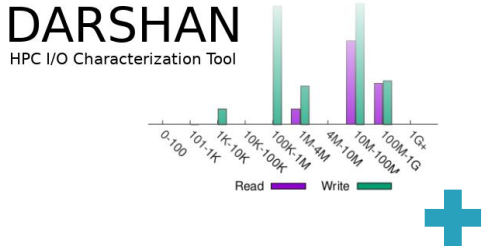
This example is boring!

It is a benchmark in which all 512 ranks wrote one big chunk at the same time, and then the program exited.

In a real application you would likely see phases of I/O, or ranks doing things at different times.



PyDarshan


DARSHAN
HPC I/O Characterization Tool
0-100
101-1K
1K-10K
10K-100K
100K-1M
1M-4M
4M-10M
10M-100M
100M-1G
1G+
Read
Write


python™

PyDarshan

- Darshan python module included as part of future Darshan release 3.3.0
- Vision that pydarshan will enable more users to write analysis code
  - No longer required to write in C
  - No longer required to parse ASCII output from darshan-parser
- Python module that
  - uses existing darshan-util code to load darshan log data similar to existing tools like darshan-parser
  - provides a low-level wrapper around darshan-util code
  - provides higher-level interface to access log data
  - supports multiple data formats
- Initial python interface!
  - Looking for feedback and experiences
  - Open to suggestions for improvement

Try it

- python3

  >>> import darshan

  >>>

```
>>> import darshan
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/ubuntu/.local/lib/python3.8/site-packages/darshan-0.0.6-py3.8.egg/darshan/__init__.py", line 14, in <module>
    from darshan.report import DarshanReport
  File "/home/ubuntu/.local/lib/python3.8/site-packages/darshan-0.0.6-py3.8.egg/darshan/report.py", line 10, in <module>
    import darshan.backend.cffi_backend as backend
  File "/home/ubuntu/.local/lib/python3.8/site-packages/darshan-0.0.6-py3.8.egg/darshan/backend/cffi_backend.py", line 24, in
<module>
    libdutil = find_utils(ffi, libdutil)
  File "/home/ubuntu/.local/lib/python3.8/site-packages/darshan-0.0.6-py3.8.egg/darshan/discover_darshan.py", line 200, in
find_utils
    raise RuntimeError('Could not find libdarshan-util.so! Is darshan-util installed? Please ensure one of the the following: 1)
export LD_LIBRARY_PATH=<path-to-libdarshan-util.so>, or 2) darshan-parser can found using the PATH variable, or 3)
pkg-config can resolve pkg-config --path darshan-util, or 4) install a wheel that includes darshan-utils via pip.')
RuntimeError: Could not find libdarshan-util.so! Is darshan-util installed? Please ensure one of the the following: 1) export
LD_LIBRARY_PATH=<path-to-libdarshan-util.so>, or 2) darshan-parser can found using the PATH variable, or 3) pkg-config
can resolve pkg-config --path darshan-util, or 4) install a wheel that includes darshan-utils via pip.
```

Structure

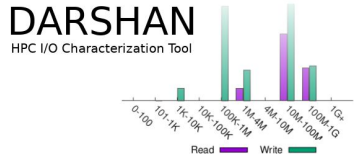
- Backend
  - Vision we might have more than one backend, but currently only one based on CFFI
  - CFFI is a python module that has limited interop with C data types and structures and C calling conventions
  - Wraps existing libdarshan-util functions for extracting data from logs
  - Probably don't start here… this is just informational
- >>> backend = darshan.backend.cffi_backend
- >>> dir(backend)
- ['API_def_c', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', '_log_get_lustre_record', '_structdefs', 'cffi', 'check_version', 'counter_names', 'ctypes', 'fcounter_names', 'ffi', 'find_utils', 'get_lib_version', 'libdutil', 'load_darshan_header', 'log_close', 'log_get_dxt_record', 'log_get_exe', 'log_get_generic_record', 'log_get_job', 'log_get_modules', 'log_get_mounts', 'log_get_name_records', 'log_get_record', 'log_lookup_name_records', 'log_open', 'logger', 'logging', 'np', 'pd']

- DarshanReport object
  - Provides easier data access, more "pythonic"
  - Loads entire log by default
  - Represents data as either numpy (default), pandas, or python dictionary
  - Convenience functions and data representations

Data Format

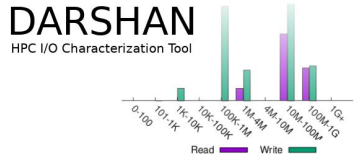
- Data is in the ‘records’ member of DarshanReport
- Records member is a dictionary of ‘DarshanRecordCollection’ objects, one for each module
  - Report.records[‘POSIX’] -> DarshanRecordCollection()
  - Derived from collections.abc.MutableSequence
  - [0] gives the first record which is dictionary-like
    - ‘id’ -> darshan record hash
    - ‘rank’ -> MPI rank the data is from or -1 if reduced from all ranks
    - ‘counters’ -> integer counters
    - ‘fcounters’ -> floating point counters
- counters and fcounters will be numpy arrays by default
  - Pandas dataframe and python dictionary are options

Notebook



- Example walkthrough
  - ecpam-21-materials/ecp-pydarshan-data-layout.ipynb
    - This notebook walks through data layout to explain the basics of accessing darshan data via pydarshan
    - We won't cover this today
  - ecpam-21-materials/ecp-pydarshan-log-analysis.ipynb
    - This notebook provides an initial basic analysis framework to look at what happens within the application I/O
    - Users can take the notebook and set the logfile to point to your own log and try it out

Resources



- Documentation
  - Darshan Webpage
  - Darshan Util documentation
- Repos
  - https://xgitlab.cels.anl.gov/darshan/darshan
  - https://xgitlab.cels.anl.gov/darshan/darshan/-/tree/master/darshan-util/pydarshan
- Training
  - ATPESC videos
- Examples
  - https://xgitlab.cels.anl.gov/darshan/darshan/-/tree/master/darshan-util/pydarshan/examples

Now, for the interactive portion of the session...

- Do any attendees have any logs they would like to share for us to analyze live during this session???

- If not, we have some prepared examples we can walkthrough to demonstrate how users can interpret Darshan log data -- do any attendees feel strongly about which examples we should prioritize?
  - PyDarshan log file analysis examples using Jupyter notebooks
  - Traditional Darshan analysis tools demonstrating interesting insights into example IOR workloads
  - Use of PyDarshan for analyzing a month's worth of logs on ALCF Theta to better understand MPI-IO behavior

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.