Darshan: Enabling Insights into HPC I/O Behavior

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

Nik Awtrey, Tyler Reddy
Los Alamos National Laboratory

ECP Community BoF Days
May 12, 2022
BoF overview

• The ability to characterize and understand application I/O workloads is critical to ensuring efficient use of an evolving and 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
BoF overview

In this BoF, we provide an overview of the Darshan I/O characterization tool and hear from guest speakers on exciting new Darshan instrumentation/analysis activities.

Agenda:

- Darshan overview (Shane Snyder, ANL)
  - Background
  - Usage on ECP platforms
  - New and upcoming features
- AutoPerf instrumentation modules (Kevin Harms, ALCF)
- PyDarshan interface and tools (Nik Awtrey, LANL)
- Understanding I/O behavior with interactive Darshan log analysis (Jean Luca Bez, LBL)
- Free time: Q/A, audience-driven discussion
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 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

https://www.mcs.anl.gov/research/projects/darshan/
https://github.com/darshan-hpc/darshan
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 ECP platforms
Using Darshan on Cori (NERSC) and Theta (ALCF)

MPI applications

➢ Darshan is already installed and enabled by default on Cori and Theta Cray XC40 systems
  ○ Instrumentation enabled using software module that injects Darshan linker options when compiling MPI applications using Cray compiler wrappers (cc, CC, etc.)

Use `module list` to confirm Darshan is actually loaded

Darshan 3.3.1 is the default version on Cori (3.3.0 on Theta)

If Darshan not loaded, you can always load manually using `module load`
Using Darshan on Cori (NERSC) and Theta (ALCF)

MPI applications

➢ OK, Darshan is loaded...now what?
  ○ Just compile and run your application!
  ○ Darshan inserts instrumentation directly into executable, regardless of linking mode

➢ 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

```bash
ssnyder@cori09:~$ module show darshan

conflict         darshan
module-whatis    a scalable HPC I/O characterization tool
setenv          DARSHAN_BASE_DIR /global/common/cori_cle7up03/software/darshan/3.3.1
prepend-path     PATH /global/common/cori_cle7up03/software/darshan/3.3.1/bin
prepend-path     LD_LIBRARY_PATH /global/common/cori_cle7up03/software/darshan/3.3.1/lib
prepend-path     PKG_CONFIG_PATH /global/common/cori_cle7up03/software/darshan/3.3.1/lib/libconfig

ssnyder@cori09:~$ export LD_PRELOAD=/global/common/cori_cle7up03/software/darshan/3.3.1/lib/libdarshan.so
```

Manually set LD_PRELOAD to point to Darshan's shared library before running your application.
Using Darshan on Summit (OLCF)

MPI applications

➢ Summit is an IBM Power9-based system that uses dynamic linking by default
  ○ Like Cori/Theta, software modules used to enable Darshan instrumentation
  ○ LD_PRELOAD mechanism used to interpose Darshan instrumentation libraries at runtime – no recompile needed!

```
[ssnyder@login4.summit ~]$ module list
Currently Loaded Modules:
  1) xl/16.1.1-10
  2) spectrum-mpi/10.4.0.3-20210112
  3) lsf-tools/2.0
  4) hsi/5.0.2.p5
  5) darshan-runtime/3.3.0-lite
  6) xalt/1.2.1
  7) DefApps
```

Summit also provides ‘module list’ command

Darshan 3.3.0 is the default version on Summit, ‘-lite’ just means this module has no HDF5 support

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

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

Darshan logs typically stored in a central directory for all users for system-wide deployments, use 'darshan-config --log-path' to find

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

Log file name starts with the following pattern: ‘username_exename_jobid…’
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 this mode

**NOTE:** With Darshan’s non-MPI environment set, log files will be generated for every invoked process

- Instrumentation scope can be limited to specific commands using: ‘env DARSHAN_ENABLE_NONMPI=1 cmd <args>’
- Darshan runtime configuration parameters (which we will cover later) can help limit instrumentation to specific application names
Analyzing Darshan logs
Analyzing Darshan logs

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

```bash
snyder@thetalogin5:/lus/theta-fs0/logs/darshan/theta/2020/1/22> cp snyder_mpi-io_test_id403177_1-22-74255-1653962535987666393_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_1-22-74255-1653962535987666393_1.darshan
```

<table>
<thead>
<tr>
<th>&lt;module&gt;</th>
<th>&lt;rank&gt;</th>
<th>&lt;record id&gt;</th>
<th>&lt;counter&gt;</th>
<th>&lt;value&gt;</th>
<th>&lt;file name&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSIX</td>
<td>-1</td>
<td>3675075178343058238</td>
<td>POSIX OPENS</td>
<td>16</td>
<td>/gpfs/mira-home</td>
</tr>
<tr>
<td>POSIX</td>
<td>-1</td>
<td>3675075178343058238</td>
<td>POSIX_READS</td>
<td>4</td>
<td>/gpfs/mira-home</td>
</tr>
<tr>
<td>POSIX</td>
<td>-1</td>
<td>3675075178343058238</td>
<td>POSIX_WRITES</td>
<td>4</td>
<td>/gpfs/mira-home</td>
</tr>
<tr>
<td>POSIX</td>
<td>-1</td>
<td>3675075178343058238</td>
<td>POSIX_SEEKS</td>
<td>6</td>
<td>/gpfs/mira-home</td>
</tr>
<tr>
<td>MPI-IO</td>
<td>-1</td>
<td>3675075178343058238</td>
<td>MPIIO_INDEP_OPENS</td>
<td>8</td>
<td>/gpfs/mira-home</td>
</tr>
<tr>
<td>MPI-IO</td>
<td>-1</td>
<td>3675075178343058238</td>
<td>MPIIO_COLL_OPENS</td>
<td>0</td>
<td>/gpfs/mira-home</td>
</tr>
<tr>
<td>MPI-IO</td>
<td>-1</td>
<td>3675075178343058238</td>
<td>MPIIO_INDEP_READS</td>
<td>4</td>
<td>/gpfs/mira-home</td>
</tr>
<tr>
<td>MPI-IO</td>
<td>-1</td>
<td>3675075178343058238</td>
<td>MPIIO_INDEP_WRITES</td>
<td>4</td>
<td>/gpfs/mira-home</td>
</tr>
<tr>
<td>MPI-IO</td>
<td>-1</td>
<td>3675075178343058238</td>
<td>MPIIO_COLL_READS</td>
<td>0</td>
<td>/gpfs/mira-home</td>
</tr>
</tbody>
</table>

- Copy the log file somewhere else for analysis
- Invoke `darshan-parser` to get detailed counters
- 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

Result is a multi-page PDF containing graphs, tables, and performance estimates characterizing the I/O workload of the application.
What’s new in Darshan?
HDF5 instrumentation modules

Available in Darshan 3.2.0+ versions

➢ HDF5 offers application scientists a convenient abstraction for large data collections
  ○ But, it can be difficult to understand interactions with lower layers of the I/O stack that most impact performance

➢ To 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 organized within files?
  ○ How are datasets accessed?
  ○ Do HDF5 accesses decompose efficiently to lower-level accesses? If not, do any optimizations make sense?

Number of elements accessed in each dataset dimension for the most common access for MACSio benchmark runs. For optimal performance, access dimensions should match any HDF5 chunking settings
Darshan runtime heatmaps

Available in Darshan 3.4.0+ versions

➢ Traditional Darshan instrumentation captures coarse-grained I/O timing info (start/end timestamps for open, read, write, close operations)
  ○ DXT tracing can be enabled for fine-grained details, but storing details of every I/O operation would be an expensive default mode for Darshan

➢ To address this, we have implemented runtime I/O heatmaps to capture per-process histograms of I/O activity
  ○ Histograms are a fixed size, with bins being resized as the app executes
  ○ Heatmaps for POSIX, MPI-IO, and STDIO interfaces are currently supported

Heatmaps can quickly provide users insight into how I/O intensity varies over time, ranks, and I/O interfaces
Darshan runtime library configuration

Available in Darshan 3.4.0+ versions

➢ Motivated by persistent user reports of Darshan exhausting memory, new comprehensive runtime library configuration mechanisms have been implemented:
  ○ Environment variables
  ○ Config files

➢ Allow runtime control over the following:
  ○ Enabled/disabled modules
  ○ Number of records to allocate for each module
  ○ Record name exclusion/inclusion regexes
  ○ App name exclusion/inclusion regexes
  ○ Total Darshan memory for module data (MODMEM) and name records (NAMEMEM)

➢ Enable facilities to fine-tune Darshan behavior after it has been deployed

Example Darshan config file to modify Darshan’s memory usage and instrumentation scope

```
# enable DXT modules, which are off by default
MOD_ENABLE DXT_POSIX,DXT_MPIIO

# allocate 4096 file records for POSIX and MPI-I/O modules
# (darshan only allocates 1824 per-module by default)
MAX_RECORDS 4096 POSIX,MP1-I

# the '*' specifier can be used to apply settings for all modules
# in this case, we want all modules to ignore record names
# prefixed with "~/home" (i.e., stored in our home directory),
# with a superseding inclusion for files with a ".out" suffix
NAME_EXCLUDE */home *
NAME_INCLUDE .*out$ *

# bump up Darshan's default memory usage to 8 MiB
MODMEM 8

# avoid generating logs for git and ls binaries
APP_EXCLUDE git,ls
```
Other recent Darshan developments

➢ AutoPerf instrumentation modules (v3.3.0)
  ○ MPI communication characterization
  ○ Network counters for Cray XC systems
  ○ Contributions by Sudheer Chunduri

➢ Stable PyDarshan interface (v3.4.0)
  ○ Including first cut at a new job summary tool
  ○ Contributions by Nik Awtrey, Tyler Reddy, Jakob Luettgau

➢ Performance optimizations to improve overhead of Darshan wrappers, locks, and timers (v3.4.0)

➢ Adopted automake/libtool support for Darshan’s build (v3.4.0)
  ○ Contribution by Wei-Keng Liao

☐ More about this shortly
Next up on the Darshan roadmap
DAOS instrumentation modules

- DAOS offers a novel storage paradigm for HPC apps: object-based storage over a combo of SCM and SSD devices
  - libdfs: DAOS’s POSIX file system emulation API
  - libdaos: DAOS’s native object (key-val) API
- Understanding usage/performance characteristics can be critical to realizing full system I/O potential
  - Development of Darshan instrumentation modules for DAOS is in progress and should be ready for release soon

DFS module available for experimental use: https://github.com/darshan-hpc/darshan/pull/739

DAOS I/O stack for HPC applications and I/O middleware. Figure courtesy of Intel
DAOS instrumentation modules

➢ The file-based DFS interface is a natural fit for Darshan’s traditional instrumentation strategy (per-file I/O statistics)
   ○ Design follows closely that of Darshan’s POSIX module, as you would expect

➢ On the other hand, a Darshan instrumentation module for the DAOS object interface poses a major challenge:
   ○ How can Darshan succinctly capture access details for objects that are key-val stores?
   ○ Key access patterns will be critical to understanding how DAOS objects are used in the wild

DAOS object storage model. Objects can either be byte arrays or entire key-val stores.

Figure courtesy of Intel

We are still investigating what characteristics about key access patterns to capture and are seeking feedback from early DAOS users
Other next steps for Darshan

- Increase instrumentation coverage
  - Support more APIs (e.g., PnetCDF, currently in a PR)
  - Extend DXT tracing capabilities to new APIs (HDF5, STDIO, etc.)

- Iterate more on PyDarshan-based log analysis tools
  - Methods to extract actionable I/O insights from raw log data
    - “looks like your Lustre striping parameters were poorly chosen, consider doing X”
  - Workflow-aware log analysis utilities
    - Existing tools operate on single jobs, but workflows are foundational to HPC and warrant deeper understanding
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.