Darshan-util installation and usage

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Introduction

This document describes darshan-util, a collection of tools for parsing and summarizing log files produced by Darshan instrumentation. The darshan-util package can be installed and used on any system regardless of where the logs were originally generated. Darshan log files are platform-independent.

More information about Darshan can be found at the Darshan web site.

Requirements

Darshan-util has only been tested in Linux environments, but will likely work in other Unix-like environments as well.

HARD REQUIREMENTS

- C compiler
- zlib development headers and library (zlib-dev or similar)

OPTIONAL REQUIREMENTS

- libbz2 development headers and library (libbz2-dev or similar)
- Perl
- pdflatex
- gnuplot 4.2 or later
- epstopdf

Compilation and installation

Configure and build example

```
tar -xvzf darshan-<version-number>.tar.gz
cd darshan-<version-number>/darshan-util
./configure
make
make install
```

The darshan-util package is intended to be used on a login node or workstation. For most use cases this means that you should either leave CC to its default setting or specify a local compiler. This is in contrast to the darshan-runtime documentation, which suggests setting CC to mpicc because the runtime library will be used in the compute node environment.

You can specify `--prefix` to install darshan-util in a specific location (such as in your home directory for non-root installations). See `./configure --help` for additional optional arguments, including how to specify alternative paths for zlib and libbz2 development libraries. darshan-util also supports VPATH or "out-of-tree" builds if you prefer that method of compilation.

The `--enable-shared` argument to configure can be used to enable compilation of a shared version of the darshan-util library.

Analyzing log files

Each time a darshan-instrumented application is executed, it will generate a single log file summarizing the I/O activity from that application. See the darshan-runtime documentation for more details, but the log file for a given application will likely be found in a centralized directory, with the path and log file name in the following format:

```
<YEAR>/<MONTH>/<DAY>/<USERNAME>_<BINARY_NAME>_<JOB_ID>_<DATE>_<UNIQUE_ID>_<TIMING>.darshan
```

This is a binary format file that summarizes I/O activity. As of version 2.0.0 of Darshan, this file is portable and does not have to be analyzed on the same system that executed the job. Also, note that Darshan logs generated with Darshan versions preceding version 3.0 will have the extension `darshan.gz` (or `darshan.bz2` if compressed using bzip2 format). These logs are not compatible with Darshan 3.0 utilities, and thus must be analyzed using an appropriate version (2.x) of the darshan-util package.

darshan-job-summary.pl

You can generate a graphical summary of the I/O activity for a job by using the `darshan-job-summary.pl` graphical summary tool as in the following example:

```
darshan-job-summary.pl carns_my-app_id114525_7-27-58921_19.darshan.gz
```

This utility requires Perl, pdflatex, epstopdf, and gnuplot in order to generate its summary. By default, the output is written to a multi-page pdf file based on the name of the input file (in this case it would produce a `carns_my-app_id114525_7-27-58921_19.pdf` output file). You can also manually specify the name of the output file using the `--output` argument.

An example of the output produced by darshan-job-summary.pl can be found HERE.

NOTE: The darshan-job-summary tool depends on a few LaTeX packages that may not be availalble by default on all systems, including: lastpage, subfigure, and threeparttable. These packages can be found and installed using your system's package manager. For instance, the packages can be installed on Debian or Ubuntu systems as follows: `apt-get install texlive-latex-extra`

darshan-summary-per-file.sh

This utility is similar to darshan-job-summary.pl, except that it produces a separate pdf summary for every file accessed by an application. It can be executed as follows:

```
darshan-summary-per-file.sh carns_my-app_id114525_7-27-58921_19.darshan.gz output-dir
```

The second argument is the name of a directory (to be created) that will contain the collection of pdf files. Note that this utility probably is not appropriate if your application opens a large number of files.

If you would like to produce a summary for a single specific file, then you can run the following command to produce a quick list of the files opened by an application and the amount of time spent performing I/O to each of them:

```
darshan-parser --file-list carns_my-app_id114525_7-27-58921_19.darshan.gz
```

Once you have identified a specific file of interest, then you can produce a summary for that specific file with the following commands:

```
darshan-convert --file HASH carns_my-app_id114525_7-27-58921_19.darshan.gz interesting_file ↩
    .darshan.gz
darshan-job-summary.pl interesting_file.darshan.gz
```

The "HASH" argument is the hash of a file name as listed in the `darshan-parser --file-list` output. The `interesting_file.darshan.gz` file produced by darshan-convert is like a normal Darshan log file, but it will only contain instrumentation for the specified file.

darshan-parser

You can use the `darshan-parser` command line utility to obtain a complete, human-readable, text-format dump of all information contained in a log file. The following example converts the contents of the log file into a fully expanded text file:

```
darshan-parser carns_my-app_id114525_7-27-58921_19.darshan.gz > ~/job-characterization.txt
```

The format of this output is described in the following section.

Guide to darshan-parser output

The beginning of the output from darshan-parser displays a summary of overall information about the job. Additional job-level summary information can also be produced using the `--perf`, `--file`, `--file-list`, `--file-list-detailed`, or `--total` command line options. See the Additional summary output section for more information about those options.

The following table defines the meaning of each line in the default header section of the output:

output line	description
"# darshan log version"	internal version number of the Darshan log file
"# exe"	name of the executable that generated the log file
"# uid"	user id that the job ran as
"# jobid"	job id from the scheduler
"# start_time"	start time of the job, in seconds since the epoch
"# start_time_asci"	start time of the job, in human readable format
"# end_time"	end time of the job, in seconds since the epoch
"# end_time_asci"	end time of the job, in human readable format
"# nprocs"	number of MPI processes
"# run time"	run time of the job in seconds

Log file region sizes

The next portion of the parser output displays the size of each region contained within the given log file. Each log file will contain the following regions:

- header - constant-sized uncompressed header providing data on how to properly access the log
- job data - job-level metadata (e.g., start/end time and exe name) for the log
- record table - a table mapping Darshan record identifiers to full file name paths
- module data - each module (e.g., POSIX, MPI-IO, etc.) stores their I/O characterization data in distinct regions of the log

All regions of the log file are compressed (in libz or bzip2 format), except the header.

Table of mounted file systems

The next portion of the output shows a table of all general purpose file systems that were mounted while the job was running. Each line uses the following format:

```
<mount point> <fs type>
```

Format of I/O characterization fields

The remainder of the output will show characteristics for each file that was opened by the application. Each line uses the following format:

```
<module> <rank> <record id> <counter name> <counter value> <file name> <mount point> <fs ↩
    type>
```

The <module> column specifies the module responsible for recording this piece of I/O characterization data. The <rank> column indicates the rank of the process that opened the file. A rank value of -1 indicates that all processes opened the same file. In that case, the value of the counter represents an aggregate across all processes. The <record id> is a 64 bit hash of the file path/name that was opened. It is used as a way to uniquely differentiate each file. The <counter name> is the name of the statistic that the line is reporting, while the <counter value> is the value of that statistic. A value of -1 indicates that Darshan was unable to collect statistics for that particular counter, and the value should be ignored. The <file name> field shows the complete file name the record corresponds to. The <mount point> is the mount point of the file system that this file belongs to and <fs type> is the type of that file system.

I/O characterization fields

The following tables show a list of integer statistics that are available for each of Darshan's current instrumentation modules, along with a description of each. Unless otherwise noted, counters include all variants of the call in question, such as `read()`, `pread()`, and `readv()` for POSIX_READS.

Table 1: POSIX module

counter name	description
POSIX_OPENS	Count of how many times the file was opened
POSIX_READS	Count of POSIX read operations
POSIX_WRITES	Count of POSIX write operations
POSIX_SEEKS	Count of POSIX seek operations
POSIX_STATS	Count of POSIX stat operations
POSIX_MMAPS	Count of POSIX mmap operations
POSIX_FSYNCS	Count of POSIX fsync operations
POSIX_FDSYNCS	Count of POSIX fdatasync operations
POSIX_MODE	Mode that the file was last opened in
POSIX_BYTES_READ	Total number of bytes that were read from the file
POSIX_BYTES_WRITTEN	Total number of bytes written to the file
POSIX_MAX_BYTE_READ	Highest offset in the file that was read
POSIX_MAX_BYTE_WRITTEN	Highest offset in the file that was written
POSIX_CONSEC_READS	Number of consecutive reads (that were immediately adjacent to the previous access)
POSIX_CONSEC_WRITES	Number of consecutive writes (that were immediately adjacent to the previous access)
POSIX_SEQ_READS	Number of sequential reads (at a higher offset than where the previous access left off)
POSIX_SEQ_WRITES	Number of sequential writes (at a higher offset than where the previous access left off)
POSIX_RW_SWITCHES	Number of times that access toggled between read and write in consecutive operations
POSIX_MEM_NOT_ALIGNED	Number of times that a read or write was not aligned in memory
POSIX_MEM_ALIGNMENT	Memory alignment value (chosen at compile time)
POSIX_FILE_NOT_ALIGNED	Number of times that a read or write was not aligned in file
POSIX_FILE_ALIGNMENT	File alignment value. This value is detected at runtime on most file systems. On Lustre, however, Darshan assumes a default value of 1 MiB for optimal file alignment.
POSIX_MAX_READ_TIME_SIZE	Size of the slowest POSIX read operation
POSIX_MAX_WRITE_TIME_SIZE	Size of the slowest POSIX write operation
POSIX_SIZE_READ_*	Histogram of read access sizes at POSIX level
POSIX_SIZE_WRITE_*	Histogram of write access sizes at POSIX level
POSIX_STRIDE[1-4]_STRIDE	Size of 4 most common stride patterns

Table 1: (continued)

counter name	description
POSIX_STRIDE[1-4]_COUNT	Count of 4 most common stride patterns
POSIX_ACCESS[1-4]_ACCESS	4 most common POSIX access sizes
POSIX_ACCESS[1-4]_COUNT	Count of 4 most common POSIX access sizes
POSIX_FASTEST_RANK	The MPI rank with smallest time spent in POSIX I/O
POSIX_FASTEST_RANK_BYTES	The number of bytes transferred by the rank with smallest time spent in POSIX I/O
POSIX_SLOWEST_RANK	The MPI rank with largest time spent in POSIX I/O
POSIX_SLOWEST_RANK_BYTES	The number of bytes transferred by the rank with the largest time spent in POSIX I/O
POSIX_F_*_START_TIMESTAMP	Timestamp that the first POSIX file open/read/write/close operation began
POSIX_F_*_END_TIMESTAMP	Timestamp that the last POSIX file open/read/write/close operation ended
POSIX_F_READ_TIME	Cumulative time spent reading at the POSIX level
POSIX_F_WRITE_TIME	Cumulative time spent in write, fsync, and fdatasync at the POSIX level
POSIX_F_META_TIME	Cumulative time spent in open, close, stat, and seek at the POSIX level
POSIX_F_MAX_READ_TIME	Duration of the slowest individual POSIX read operation
POSIX_F_MAX_WRITE_TIME	Duration of the slowest individual POSIX write operation
POSIX_F_FASTEST_RANK_TIME	The time of the rank which had the smallest amount of time spent in POSIX I/O (cumulative read, write, and meta times)
POSIX_F_SLOWEST_RANK_TIME	The time of the rank which had the largest amount of time spent in POSIX I/O
POSIX_F_VARIANCE_RANK_TIME	The population variance for POSIX I/O time of all the ranks
POSIX_F_VARIANCE_RANK_BYTES	The population variance for bytes transferred of all the ranks

Table 2: MPI-IO module

counter name	description
MPIIO_INDEP_OPENS	Count of non-collective MPI opens
MPIIO_COLL_OPENS	Count of collective MPI opens
MPIIO_INDEP_READS	Count of non-collective MPI reads
MPIIO_INDEP_WRITES	Count of non-collective MPI writes
MPIIO_COLL_READS	Count of collective MPI reads
MPIIO_COLL_WRITES	Count of collective MPI writes
MPIIO_SPLIT_READS	Count of MPI split collective reads
MPIIO_SPLIT_WRITES	Count of MPI split collective writes
MPIIO_NB_READS	Count of MPI non-blocking reads
MPIIO_NB_WRITES	Count of MPI non-blocking writes
MPIIO_SYNCS	Count of MPI file syncs
MPIIO_HINTS	Count of MPI file hints used
MPIIO_VIEWS	Count of MPI file views used
MPIIO_MODE	MPI mode that the file was last opened in
MPIIO_BYTES_READ	Total number of bytes that were read from the file at MPI level
MPIIO_BYTES_WRITTEN	Total number of bytes written to the file at MPI level
MPIIO_RW_SWITCHES	Number of times that access toggled between read and write in consecutive MPI operations
MPIIO_MAX_READ_TIME_SIZE	Size of the slowest MPI read operation
MPIIO_MAX_WRITE_TIME_SIZE	Size of the slowest MPI write operation
MPIIO_SIZE_READ_AGG_*	Histogram of total size of read accesses at MPI level, even if access is noncontiguous
MPIIO_SIZE_WRITE_AGG_*	Histogram of total size of write accesses at MPI level, even if access is noncontiguous

Table 2: (continued)

counter name	description
MPIIO_ACCESS[1-4]_ACCESS	4 most common MPI aggregate access sizes
MPIIO_ACCESS[1-4]_COUNT	Count of 4 most common MPI aggregate access sizes
MPIIO_FASTEST_RANK	The MPI rank with smallest time spent in MPI I/O
MPIIO_FASTEST_RANK_BYTES	The number of bytes transferred by the rank with smallest time spent in MPI I/O
MPIIO_SLOWEST_RANK	The MPI rank with largest time spent in MPI I/O
MPIIO_SLOWEST_RANK_BYTES	The number of bytes transferred by the rank with the largest time spent in MPI I/O
MPIIO_F_OPEN_TIMESTAMP	Timestamp of first time that the file was opened at MPI level
MPIIO_F_READ_START_TIMESTAMP	Timestamp that the first MPI read operation began
MPIIO_F_WRITE_START_TIMESTAMP	Timestamp that the first MPI write operation begin
MPIIO_F_READ_END_TIMESTAMP	Timestamp that the last MPI read operation ended
MPIIO_F_WRITE_END_TIMESTAMP	Timestamp that the last MPI write operation ended
MPIIO_F_CLOSE_TIMESTAMP	Timestamp of the last time that the file was closed at MPI level
MPIIO_READ_TIME	Cumulative time spent reading at MPI level
MPIIO_WRITE_TIME	Cumulative time spent write and sync at MPI level
MPIIO_META_TIME	Cumulative time spent in open and close at MPI level
MPIIO_F_MAX_READ_TIME	Duration of the slowest individual MPI read operation
MPIIO_F_MAX_WRITE_TIME	Duration of the slowest individual MPI write operation
MPIIO_F_FASTEST_RANK_TIME	The time of the rank which had the smallest amount of time spent in MPI I/O (cumulative read, write, and meta times)
MPIIO_F_SLOWEST_RANK_TIME	The time of the rank which had the largest amount of time spent in MPI I/O
MPIIO_F_VARIANCE_RANK_TIME	The population variance for MPI I/O time of all the ranks
MPIIO_F_VARIANCE_RANK_BYTES	The population variance for bytes transferred of all the ranks at MPI level

Table 3: STDIO module

counter name	description
STDIO_OPENS	Count of how many times the file was opened using the stdio interface (e.g., `fopen()`)
STDIO_READS	Count of stdio read operations
STDIO_WRITES	Count of stdio write operations
STDIO_SEEKS	Count of stdio seek operations
STDIO_FLUSHES	Count of stdio flush operations
STDIO_BYTES_WRITTEN	Total number of bytes written to the file using stdio operations
STDIO_BYTES_READ	Total number of bytes read from the file using stdio operations
STDIO_MAX_BYTE_READ	Highest offset in the file that was read
STDIO_MAX_BYTE_WRITTEN	Highest offset in the file that was written
STDIO_FASTEST_RANK	The MPI rank with the smallest time spent in stdio operations
STDIO_FASTEST_RANK_BYTES	The number of bytes transferred by the rank with the smallest time spent in stdio operations
STDIO_SLOWEST_RANK	The MPI rank with the largest time spent in stdio operations
STDIO_SLOWEST_RANK_BYTES	The number of bytes transferred by the rank with the largest time spent in stdio operations
STDIO_META_TIME	Cumulative time spent in stdio open/close/seek operations
STDIO_WRITE_TIME	Cumulative time spent in stdio write operations
STDIO_READ_TIME	Cumulative time spent in stdio read operations
STDIO_*_START_TIMESTAMP	Timestamp that the first stdio file open/read/write/close operation began
STDIO_*_END_TIMESTAMP	Timestamp that the last stdio file open/read/write/close operation ended

Table 3: (continued)

counter name	description
STDIO_F_FASTEST_RANK_TIME	The time of the rank which had the smallest time spent in stdio I/O (cumulative read, write, and meta times)
STDIO_F_SLOWEST_RANK_TIME	The time of the rank which had the largest time spent in stdio I/O
STDIO_F_VARIANCE_RANK_TIME	The population variance for stdio I/O time of all the ranks
STDIO_F_VARIANCE_RANK_BYTES	The population variance for bytes transferred of all the ranks

Table 4: HDF5 module

counter name	description
HDF5_OPENS	Count of HDF5 opens
HDF5_F_OPEN_TIMESTAMP	Timestamp of first time that the file was opened at HDF5 level
HDF5_F_CLOSE_TIMESTAMP	Timestamp of the last time that the file was closed at HDF5 level

Table 5: PnetCDF module

counter name	description
PNETCDF_INDEP_OPENS	Count of PnetCDF independent opens
PNETCDF_COLL_OPENS	Count of PnetCDF collective opens
PNETCDF_F_OPEN_TIMESTAMP	Timestamp of first time that the file was opened at PnetCDF level
PNETCDF_F_CLOSE_TIMESTAMP	Timestamp of the last time that the file was closed at PnetCDF level

Additional modules

Table 6: BG/Q module (if enabled on BG/Q systems)

counter name	description
BGQ_CSJOBID	Control system job ID
BGQ_NNODES	Total number of BG/Q compute nodes
BGQ_RANKSPERNODE	Number of MPI ranks per compute node
BGQ_DDRPERNODE	Size of compute node DDR in MiB
BGQ_INODES	Total number of BG/Q I/O nodes
BGQ_ANODES	Dimension of A torus
BGQ_BNODES	Dimension of B torus
BGQ_CNODES	Dimension of C torus
BGQ_DNODES	Dimension of D torus
BGQ_ENODES	Dimension of E torus
BGQ_TORUSENABLED	Bitfield indicating enabled torus dimensions
BGQ_F_TIMESTAMP	Timestamp of when BG/Q data was collected

Table 7: Lustre module (if enabled, for Lustre file systems)

counter name	description
LUSTRE_OSTS	number of OSTs (object storage targets) for the file system
LUSTRE_MDTS	number of MDTs (metadata targets) for the file system
LUSTRE_STRIPE_OFFSET	OST id offset specified at file creation time
LUSTRE_STRIPE_SIZE	stripe size for the file in bytes
LUSTRE_STRIPE_WIDTH	number of OSTs over which the file is striped
LUSTRE_OST_ID_*	indices of OSTs over which the file is striped

Additional summary output

The following sections describe addtitional parser options that provide summary I/O characterization data for the given log.

NOTE: These options are currently only supported by the POSIX, MPI-IO, and stdio modules.

Performance

Job performance information can be generated using the `--perf` command-line option.

Example output

```
# performance
# -----------
# total_bytes: 134217728
#
# I/O timing for unique files (seconds):
# ...........................
# unique files: slowest_rank_io_time: 0.000000
# unique files: slowest_rank_meta_only_time: 0.000000
# unique files: slowest_rank: 0
#
# I/O timing for shared files (seconds):
# (multiple estimates shown; time_by_slowest is generally the most accurate)
# ...........................
# shared files: time_by_cumul_io_only: 0.042264
# shared files: time_by_cumul_meta_only: 0.000325
# shared files: time_by_open: 0.064986
# shared files: time_by_open_lastio: 0.064966
# shared files: time_by_slowest: 0.057998
#
# Aggregate performance, including both shared and unique files (MiB/s):
# (multiple estimates shown; agg_perf_by_slowest is generally the most
# accurate)
# ...........................
# agg_perf_by_cumul: 3028.570529
# agg_perf_by_open: 1969.648064
# agg_perf_by_open_lastio: 1970.255248
# agg_perf_by_slowest: 2206.983935
```

The `total_bytes` line shows the total number of bytes transferred (read/written) by the job. That is followed by three sections:

I/O timing for unique files This section reports information about any files that were **not** opened by every rank in the job. This includes independent files (opened by 1 process) and partially shared files (opened by a proper subset of the job's processes). The I/O time for this category of file access is reported based on the **slowest** rank of all processes that performed this type of file access.

- unique files: slowest_rank_io_time: total I/O time for unique files (including both metadata + data transfer time)
- unique files: slowest_rank_meta_only_time: metadata time for unique files
- unique files: slowest_rank: the rank of the slowest process

I/O timing for shared files This section reports information about files that were globally shared (i.e. opened by every rank in the job). This section estimates performance for globally shared files using four different methods. The `time_by_slowest` is generally the most accurate, but it may not available in some older Darshan log files.

- shared files: time_by_cumul_*: adds the cumulative time across all processes and divides by the number of processes (inaccurate when there is high variance among processes).
  - shared files: time_by_cumul_io_only: include metadata AND data transfer time for global shared files
  - shared files: time_by_cumul_meta_only: metadata time for global shared files
- shared files: time_by_open: difference between timestamp of open and close (inaccurate if file is left open without I/O activity)
- shared files: time_by_open_lastio: difference between timestamp of open and the timestamp of last I/O (similar to above but fixes case where file is left open after I/O is complete)
- shared files: time_by_slowest : measures time according to which rank was the slowest to perform both metadata operations and data transfer for each shared file. (most accurate but requires newer log version)

Aggregate performance Performance is calculated by dividing the total bytes by the I/O time (shared files and unique files combined) computed using each of the four methods described in the previous output section. Note the unit for total bytes is Byte and for the aggregate performance is MiB/s (1024*1024 Bytes/s).

Files

Use the `--file` option to get totals based on file usage. Each line has 3 columns. The first column is the count of files for that type of file, the second column is number of bytes for that type, and the third column is the maximum offset accessed.

- total: All files
- read_only: Files that were only read from
- write_only: Files that were only written to
- read_write: Files that were both read and written
- unique: Files that were opened on only one rank
- shared: Files that were opened by more than one rank

Example output

```
# <file_type> <file_count> <total_bytes> <max_byte_offset>
# total: 5 4371499438884 4364699616485
# read_only: 2 4370100334589 4364699616485
# write_only: 1 1399104295 1399104295
# read_write: 0 0 0
# unique: 0 0 0
# shared: 5 4371499438884 4364699616485
```

Totals

Use the `--total` option to get all statistics as an aggregate total rather than broken down per file. Each field is either summed across files and process (for values such as number of opens), set to global minimums and maximums (for values such as open time and close time), or zeroed out (for statistics that are nonsensical in aggregate).

Example output

```
total_POSIX_OPENS: 1024
total_POSIX_READS: 0
total_POSIX_WRITES: 16384
total_POSIX_SEEKS: 16384
total_POSIX_STATS: 1024
total_POSIX_MMAPS: 0
total_POSIX_FOPENS: 0
total_POSIX_FREADS: 0
total_POSIX_FWRITES: 0
total_POSIX_BYTES_READ: 0
total_POSIX_BYTES_WRITTEN: 68719476736
total_POSIX_MAX_BYTE_READ: 0
total_POSIX_MAX_BYTE_WRITTEN: 67108863
...
```

File list

Use the `--file-list` option to produce a list of files opened by the application along with estimates of the amount of time spent accessing each file.

Example output

```
# Per-file summary of I/O activity.
# -----
# <record_id>: darshan record id for this file
# <file_name>: full file name
# <nprocs>: number of processes that opened the file
# <slowest>: (estimated) time in seconds consumed in IO by slowest process
# <avg>: average time in seconds consumed in IO per process

# <record_id>   <file_name> <nprocs>    <slowest>   <avg>
5041708885572677970 /projects/SSSPPg/snyder/ior/ior.dat 1024    16.342061   1.705930
```

This data could be post-processed to compute more in-depth statistics, such as the total number of MPI files and total number of POSIX files used in a job, categorizing files into independent/unique/local files (opened by 1 process), subset/partially shared files (opened by a proper subset of processes) or globally shared files (opened by all processes), and ranking files according to how much time was spent performing I/O in each file.

Detailed file list

The `--file-list-detailed` is the same as --file-list except that it produces many columns of output containing statistics broken down by file. This option is mainly useful for more detailed automated analysis.

darshan-dxt-parser

The `darshan-dxt-parser` utility can be used to parse DXT traces out of Darshan log files, assuming the corresponding application was executed with the DXT modules enabled. The following example parses all DXT trace information out of a Darshan log file and stores it in a text file:

```
darshan-dxt-parser shane_ior_id25016_1-31-38066-13864742673678115131_1.darshan > ~/ior- ↩
    trace.txt
```

Guide to darshan-dxt-parser output

The preamble to `darshan-dxt-parser` output is identical to that of the traditional `darshan-parser` utility, which is described above.

`darshan-dxt-parser` displays detailed trace information contained within a Darshan log that was generated with DXT instrumentation enabled. Trace data is captured from both POSIX and MPI-IO interfaces. Example output is given below:

Example output

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

# DXT, file_id: 16457598720760448348, file_name: /tmp/test/testFile
# DXT, rank: 0, hostname: shane-thinkpad
# DXT, write_count: 4, read_count: 4
# DXT, mnt_pt: /, fs_type: ext4
# Module    Rank  Wt/Rd  Segment          Offset       Length    Start(s)      End(s)
 X_POSIX       0  write        0               0       262144      0.0029      0.0032
 X_POSIX       0  write        1          262144       262144      0.0032      0.0035
 X_POSIX       0  write        2          524288       262144      0.0035      0.0038
 X_POSIX       0  write        3          786432       262144      0.0038      0.0040
 X_POSIX       0   read        0               0       262144      0.0048      0.0048
 X_POSIX       0   read        1          262144       262144      0.0049      0.0049
 X_POSIX       0   read        2          524288       262144      0.0049      0.0050
 X_POSIX       0   read        3          786432       262144      0.0050      0.0051

# ***************************************************
# DXT_MPIIO module data
# ***************************************************

# DXT, file_id: 16457598720760448348, file_name: /tmp/test/testFile
# DXT, rank: 0, hostname: shane-thinkpad
# DXT, write_count: 4, read_count: 4
# DXT, mnt_pt: /, fs_type: ext4
# Module    Rank  Wt/Rd  Segment       Length    Start(s)      End(s)
 X_MPIIO       0  write        0       262144      0.0029      0.0032
 X_MPIIO       0  write        1       262144      0.0032      0.0035
 X_MPIIO       0  write        2       262144      0.0035      0.0038
 X_MPIIO       0  write        3       262144      0.0038      0.0040
 X_MPIIO       0   read        0       262144      0.0048      0.0049
 X_MPIIO       0   read        1       262144      0.0049      0.0049
 X_MPIIO       0   read        2       262144      0.0049      0.0050
 X_MPIIO       0   read        3       262144      0.0050      0.0051
```

DXT POSIX module

This module provides details on each read or write access at the POSIX layer. The trace output is organized first by file then by process rank. So, for each file accessed by the application, DXT will provide each process's I/O trace segments in separate blocks, ordered by increasing process rank. Within each file/rank block, I/O trace segments are ordered chronologically.

Before providing details on each I/O operation, DXT provides a short preamble for each file/rank trace block with the following bits of information: the Darshan identifier for the file (which is equivalent to the identifers used by Darshan in its traditional modules), the full file path, the corresponding MPI rank the current block of trace data belongs to, the hostname associated with this process rank, the number of individual POSIX read and write operations by this process, and the mount point and file system type corresponding to the traced file.

The output format for each indvidual I/O operation segment is:

```
# Module    Rank  Wt/Rd  Segment          Offset       Length    Start(s)      End(s)
```

- Module: corresponding DXT module (DXT_POSIX or DXT_MPIIO)
- Rank: process rank responsible for I/O operation
- Wt/Rd: whether the operation was a write or read
- Segment: The operation number for this segment (first operation is segment 0)
- Offset: file offset the I/O operation occured at
- Length: length of the I/O operation in bytes
- Start: timestamp of the start of the operation (w.r.t. application start time)
- End: timestamp of the end of the operation (w.r.t. application start time)

DXT MPI-IO module

If the MPI-IO interface is used by an application, this module provides details on each read or write access at the MPI-IO layer. This data is often useful in understanding how MPI-IO read or write operations map to underlying POSIX read or write operations issued to the traced file.

The output format for the DXT MPI-IO module is essentially identical to the DXT POSIX module, except that the offset of file operations is not tracked.

Other darshan-util utilities

The darshan-util package includes a number of other utilies that can be summarized briefly as follows:

- darshan-convert: converts an existing log file to the newest log format. If the `--bzip2` flag is given, then the output file will be re-compressed in bzip2 format rather than libz format. It also has command line options for anonymizing personal data, adding metadata annotation to the log header, and restricting the output to a specific instrumented file.
- darshan-diff: provides a text diff of two Darshan log files, comparing both job-level metadata and module data records between the files.
- darshan-analyzer: walks an entire directory tree of Darshan log files and produces a summary of the types of access methods used in those log files.
- darshan-logutils*: this is a library rather than an executable, but it provides a C interface for opening and parsing Darshan log files. This is the recommended method for writing custom utilities, as darshan-logutils provides a relatively stable interface across different versions of Darshan and different log formats.
- dxt_analyzer: plots the read or write activity of a job using data obtained from Darshan's DXT modules (if DXT is enabled).