Darshan-runtime installation and usage

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Introduction

This document describes darshan-runtime, which is the instrumentation portion of the Darshan characterization tool. It should be installed on the system where you intend to collect I/O characterization information.

Darshan instruments applications via either compile time wrappers for static executables or dynamic library preloading for dynamic executables. An application that has been instrumented with Darshan will produce a single log file each time it is executed. This log summarizes the I/O access patterns used by the application.

The darshan-runtime instrumentation only instruments MPI applications (the application must at least call `MPI_Init()` and `MPI_Finalize()`). However, it captures both MPI-IO and POSIX file access. It also captures limited information about HDF5 and PnetCDF access.

This document provides generic installation instructions, but "recipes" for several common HPC systems are provided at the end of the document as well.

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

2 Requirements

- MPI C compiler
- zlib development headers and library

3 Compilation

Configure and build example

```
tar -xvzf darshan-<version-number>.tar.gz
cd darshan-<version-number>/darshan-runtime
./configure --with-mem-align=8 --with-log-path=/darshan-logs --with-jobid-env=PBS_JOBID CC= ↩
    mpicc
make
make install
```

Detecting file size and alignment
You can also add --enable-stat-at-open option to cause the Darshan library to issue an additional stat() system call on each file the first time that it is opened on each process. This allows Darshan to detect the file alignment (and subsequent unaligned accesses). It also allows Darshan to detect the size of files at open time before any I/O is performed. Unfortunately, this option can cause significant overhead at scale on file systems such as PVFS or Lustre that must contact every server for a given file in order to satisfy a stat request. We therefore disable this feature by default.

EXPLANATION OF CONFIGURE ARGUMENTS:

- `--with-mem-align` (mandatory): This value is system-dependent and will be used by Darshan to determine if the buffer for a read or write operation is aligned in memory.
- `--with-log-path` (this, or `--with-log-path-by-env`, is mandatory): This specifies the parent directory for the directory tree where darshan logs will be placed
- `--with-jobid-env` (mandatory): this specifies the environment variable that Darshan should check to determine the jobid of a job. Common values are `PBS_JOBID` or `COBALT_JOBID`. If you are not using a scheduler (or your scheduler does not advertise the job ID) then you can specify `NONE` here. Darshan will fall back to using the pid of the rank 0 process if the specified environment variable is not set.

- `CC=`: specifies the MPI C compiler to use for compilation
- `--with-log-path-by-env`: specifies an environment variable to use to determine the log path at run time.
- `--with-log-hints=`: specifies hints to use when writing the Darshan log file. See `./configure --help` for details.
- `--with-zlib=`: specifies an alternate location for the zlib development header and library.

3.1 Cross compilation

On some systems (notably the IBM Blue Gene series), the login nodes do not have the same architecture or runtime environment as the compute nodes. In this case, you must configure darshan-runtime to be built using a cross compiler. The following configure arguments show an example for the BG/P system:

```
--host=powerpc-bgp-linux CC=/bgsys/drivers/ppcfloor/comm/default/bin/mpicc
```

4 Environment preparation

Once darshan-runtime has been installed, you must prepare a location in which to store the Darshan log files and configure an instrumentation method.

4.1 Log directory

This step can be safely skipped if you configured darshan-runtime using the `--with-log-path-by-env` option. A more typical configuration uses a static directory hierarchy for Darshan log files.

The `darshan-mk-log-dirs.pl` utility will configure the path specified at configure time to include subdirectories organized by year, month, and day in which log files will be placed. The deepest subdirectories will have sticky permissions to enable multiple users to write to the same directory. If the log directory is shared system-wide across many users then the following script should be run as root.

```
darshan-mk-log-dirs.pl
```

A note about log directory permissions
All log files written by darshan have permissions set to only allow read access by the owner of the file. You can modify this behavior, however, by specifying the --enable-group-readable-logs option at configure time. One notable deployment scenario would be to configure Darshan and the log directories to allow all logs to be readable by both the end user and a Darshan administrators group. This can be done with the following steps:

- set the --enable-group-readable-logs option at configure time
- create the log directories with darshan-mk-log-dirs.pl
- recursively set the group ownership of the log directories to the Darshan administrators group
- recursively set the setgid bit on the log directories

4.2 Instrumentation method

The instrumentation method to use depends on whether the executables produced by your MPI compiler are statically or dynamically linked. If you are unsure, you can check by running `ldd <executable_name>` on an example executable. Dynamically-linked executables will produce a list of shared libraries when this command is executed.

Most MPI compilers allow you to toggle dynamic or static linking via options such as `-dynamic` or `-static`. Please check your MPI compiler man page for details if you intend to force one mode or the other.

5 Instrumenting statically-linked applications

Statically linked executables must be instrumented at compile time. The simplest way to do this is to generate an MPI compiler script (e.g. `mpicc`) that includes the link options and libraries needed by Darshan. Once this is done, Darshan instrumentation is transparent; you simply compile applications using the darshan-enabled MPI compiler scripts.

For MPICH-based MPI libraries, such as MPICH1, MPICH2, or MVAPICH, these wrapper scripts can be generated automatically. The following example illustrates how to produce wrappers for C, C++, and Fortran compilers:

```
darshan-gen-cc.pl `which mpicc` --output mpicc.darshan
darshan-gen-cxx.pl `which mpicxx` --output mpicxx.darshan
darshan-gen-fortran.pl `which mpif77` --output mpif77.darshan
darshan-gen-fortran.pl `which mpif90` --output mpif90.darshan
```

Please see the Cray recipe in this document for instructions on instrumenting statically-linked applications on that platform.

For other MPI Libraries you must manually modify the MPI compiler scripts to add the necessary link options and libraries. Please see the `darshan-gen-*` scripts for examples or contact the Darshan users mailing list for help.

6 Instrumenting dynamically-linked applications

For dynamically-linked executables, darshan relies on the `LD_PRELOAD` environment variable to insert instrumentation at run time. The executables should be compiled using the normal, unmodified MPI compiler.

To use this mechanism, set the `LD_PRELOAD` environment variable to the full path to the Darshan shared library, as in this example:

```
export LD_PRELOAD=/home/carns/darshan-install/lib/libdarshan.so
```

You can then run your application as usual. Some environments may require a special `mpirun` or `mpiexec` command line argument to propagate the environment variable to all processes. Other environments may require a scheduler submission option to control this behavior. Please check your local site documentation for details.

6.1 Instrumenting dynamically-linked Fortran applications

Please follow the general steps outlined in the previous section. For Fortran applications compiled with MPICH you may have to take the additional step of adding `libfmpich.so` to your `LD_PRELOAD` environment variable. For example:

```
export LD_PRELOAD=/path/to/mpi/used/by/executable/lib/libfmpich.so:/home/carns/darshan- ↩
    install/lib/libdarshan.so
```

Note
The full path to the libfmpich.so library can be omitted if the rpath variable points to the correct path. Be careful to check the rpath of the darshan library and the executable before using this configuration, however. They may provide conflicting paths. Ideally the rpath to the MPI library would **not** be set by the darshan library, but would instead be specified exclusively by the executable itself. You can check the rpath of the darshan library by running `objdump -x /home/carns/darshan-install/lib/libdarshan.so |grep RPATH`.

7 Darshan installation recipes

The following recipes provide examples for prominent HPC systems. These are intended to be used as a starting point. You will most likely have to adjust paths and options to reflect the specifics of your system.

7.1 IBM Blue Gene (BG/P or BG/Q)

IBM Blue Gene systems produces static executables by default, uses a different architecture for login and compute nodes, and uses an MPI environment based on MPICH.

The following example shows how to configure Darshan on a BG/P system:

```
./configure --with-mem-align=16 \
 --with-log-path=/home/carns/working/darshan/releases/logs \
 --prefix=/home/carns/working/darshan/install --with-jobid-env=COBALT_JOBID \
 --with-zlib=/soft/apps/zlib-1.2.3/ \
 --host=powerpc-bgp-linux CC=/bgsys/drivers/ppcfloor/comm/default/bin/mpicc
```

Rationale
The memory alignment is set to 16 not because that is the proper alignment for the BG/P CPU architecture, but because that is the optimal alignment for the network transport used between compute nodes and I/O nodes in the system. The jobid environment variable is set to `COBALT_JOBID` in this case for use with the Cobalt scheduler, but other BG/P systems may use different schedulers. The `--with-zlib` argument is used to point to a version of zlib that has been compiled for use on the compute nodes rather than the login node. The `--host` argument is used to force cross-compilation of Darshan. The `CC` variable is set to point to a stock MPI compiler.

Once Darshan has been installed, use the `darshan-gen-*.pl` scripts as described earlier in this document to produce darshan-enabled MPI compilers. This method has been widely used and tested with both the GNU and IBM XL compilers.

7.2 Cray platforms (XE, XC, or similar)

The Cray programming environment produces static executables by default, which means that Darshan instrumentation must be inserted at compile time. This can be accomplished by loading a software module that sets appropriate environment variables to modify the Cray compiler script link behavior. This section describes how to compile and install Darshan, as well as how to use a software module to enable and disable Darshan instrumentation.

7.2.1 Building and installing Darshan

Please set your environment to use the GNU programming environment before configuring or compiling Darshan. Although Darshan can be built with a variety of compilers, the GNU compilers are recommended because it will produce a Darshan library that is interoperable with the widest range of compmilers and linkers. On most Cray systems you can enable the GNU programming environment with a command similar to "module swap PrgEnv-pgi PrgEnv-gnu". Please see your site documentation for information about how to switch programming environments.

The following example shows how to configure and build Darshan on a Cray system using either the GNU programming environment. Adjust the --with-log-path and --prefix arguments to point to the desired log file path and installation path, respectively.

```
module swap PrgEnv-pgi PrgEnv-gnu
./configure --with-mem-align=8 \
 --with-log-path=/shared-file-system/darshan-logs \
 --prefix=/soft/darshan-2.2.3 \
 --with-jobid-env=PBS_JOBID --disable-cuserid CC=cc
make install
module swap PrgEnv-gnu PrgEnv-pgi
```

Rationale
The job ID is set to `PBS_JOBID` for use with a Torque or PBS based scheduler. The `CC` variable is configured to point the standard MPI compiler.
The --disable-cuserid argument is used to prevent Darshan from attempting to use the cuserid() function to retrieve the user name associated with a job. Darshan automatically falls back to other methods if this function fails, but on some Cray environments (notably the Beagle XE6 system as of March 2012) the cuserid() call triggers a segmentation fault. With this option set, Darshan will typically use the LOGNAME environment variable to determine a userid.

As in any Darshan installation, the darshan-mk-log-dirs.pl script can then be used to create the appropriate directory hierarchy for storing Darshan log files in the --with-log-path directory.

Note that Darshan is not currently capable of detecting the stripe size (and therefore the Darshan FILE_ALIGNMENT value) on Lustre file systems. If a Lustre file system is detected, then Darshan assumes an optimal file alignment of 1 MiB.

7.2.2 Enabling Darshan instrumentation

Darshan will automatically install example software module files in the following locations (depending on how you specified the --prefix option in the previous section):

```
/soft/darshan-2.2.3/share/craype-1.x/modulefiles/darshan
/soft/darshan-2.2.3/share/craype-2.x/modulefiles/darshan
```

Select the one that is appropriate for your Cray programming environment (see the version number of the craype module in `module list`).

If you are using the Cray Programming Environment version 1.x, then you must modify the corresponding modulefile before using it. Please see the comments at the end of the file and choose an environment variable method that is appropriate for your system. If this is not done, then the compiler may fail to link some applications when the Darshan module is loaded.

If you are using the Cray Programming Environment version 2.x then you can likely use the modulefile as is. Note that it pulls most of its configuration from the lib/pkgconfig/darshan-runtime.pc file installed with Darshan.

The modulefile that you select can be copied to a system location, or the install location can be added to your local module path with the following command:

```
module use /soft/darshan-2.2.3/share/craype-<VERSION>/modulefiles
```

From this point, Darshan instrumenation can be enabled for all future application compilations by running "module load darshan".

7.3 Linux clusters using Intel MPI

Most Intel MPI installations produce dynamic executables by default. To configure Darshan in this environment you can use the following example:

```
./configure --with-mem-align=8 --with-log-path=/darshan-logs --with-jobid-env=PBS_JOBID CC= ↩
    mpicc
```

Rationale
There is nothing unusual in this configuration except that you should use the underlying GNU compilers rather than the Intel ICC compilers to compile Darshan itself.

You can use the `LD_PRELOAD` method described earlier in this document to instrument executables compiled with the Intel MPI compiler scripts. This method has been briefly tested using both GNU and Intel compilers.

Caveat
Darshan is only known to work with C and C++ executables generated by the Intel MPI suite. Darshan will not produce instrumentation for Fortran executables. For more details please check this Intel forum discussion:
http://software.intel.com/en-us/forums/showthread.php?t=103447&o=a&s=lr

7.4 Linux clusters using MPICH

Follow the generic instructions provided at the top of this document. The only modification is to make sure that the `CC` used for compilation is based on a GNU compiler. Once Darshan has been installed, it should be capable of instrumenting executables built with GNU, Intel, and PGI compilers.

7.5 Linux clusters using Open MPI

Follow the generic instructions provided at the top of this document for compilation, and make sure that the `CC` used for compilation is based on a GNU compiler.

Open MPI typically produces dynamically linked executables by default, which means that you should use the `LD_PRELOAD` method to instrument executables that have been built with Open MPI. Darshan is only compatible with Open MPI 1.6.4 and newer. For more details on why Darshan is not compatible with older versions of Open MPI, please refer to the following mailing list discussion:

http://www.open-mpi.org/community/lists/devel/2013/01/11907.php

8 Runtime environment variables

The Darshan library honors the following environment variables to modify behavior at runtime:

- DARSHAN_DISABLE: disables Darshan instrumentation
- DARSHAN_INTERNAL_TIMING: enables internal instrumentation that will print the time required to startup and shutdown Darshan to stderr at run time.
- DARSHAN_LOGHINTS: specifies the MPI-IO hints to use when storing the Darshan output file. The format is a semicolon-delimited list of key=value pairs, for example: hint1=value1;hint2=value2
- DARSHAN_DISABLE_TIMING: disables the subset of Darshan instrumentation that gathers timing information
- DARSHAN_MEMALIGN: specifies a value for memory alignment (CP_MEM_ALIGNMENT)
- DARSHAN_JOBID: specifies the name of the environment variable to use for the job identifier, such as PBS_JOBID
- DARSHAN_DISABLE_SHARED_REDUCTION: disables the step in Darshan aggregation in which files that were accessed by all ranks are collapsed into a single cumulative file record at rank 0. This option retains more per-process information at the expense of creating larger log files.