ANL/MCS-TM-362

Performance Evaluation of Darshan 3.0.0 on the Cray XC30

Mathematics and Computer Science Division

ANL/MCS-TM-362

Performance Evaluation of Darshan 3.0.0 on the Cray XC30

Prepared by

S. Snyder, P. Carns, K. Harms, R. Latham, and R. Ross

Mathematics and Computer Science Division, Argonne National Laboratory

April 30, 2016

Performance Evaluation of Darshan 3.0.0 on the Cray XC30

Shane Snyder, Philip Carns, Kevin Harms, Robert Latham, and Robert Ross
Argonne National Laboratory
January 2016

***Abstract*—Darshan is a lightweight I/O characterization tool used to gather and summarize salient I/O workload statistics from HPC applications. Darshan was designed to minimize any possible perturbations of an application's performance, leading it to be enabled by default on a number of production HPC systems. For each file accessed by a given application, Darshan records the count and types of I/O operations performed, histograms of access sizes, cumulative timers on the amount of time spent doing I/O, and other statistical data. This type of data has proved invaluable in understanding and improving the I/O performance of HPC applications.**

Darshan 3.0.0 is the new modularized version of the traditional Darshan library and file format, allowing users to easily add more in-depth I/O characterization data to Darshan logs. In this work we perform an empirical evaluation of Darshan 3.0.0 to ensure that it continues to meet performance expectations for broad deployment. In particular, we evaluate the imposed overhead on instrumented I/O operations, time taken to shut down and generate corresponding Darshan log files, and resultant log file sizes for different workloads. These performance results are compared to results of Darshan 2.3.0 on the Edison XC30 system at NERSC to determine whether the new version is lightweight enough to run full-time on production HPC systems. Our evaluation shows that Darshan has limited impact on application I/O performance and can fully generate a corresponding log file for most application workloads in under two seconds.

I. Background

Darshan's design was explained in a prior publication [1], which also includes an initial scalability analysis of the Darshan instrumentation method. Follow-on research demonstrated how Darshan could be to used to facilitate in-depth analysis and optimization of the I/O performance of production HPC applications [2]. Studies also have been done of the performance of Darshan on Cray XE6 systems at the University of Chicago/Argonne Computation Institute [3] and at the National Energy Research Supercomputing Center (NERSC) [4]. Each of these studies was performed with previous versions of Darshan (2.x and earlier).

The intent of this study is to perform similar scalability and overhead analyses of Darshan version 3.0.0 to determine whether it is suitable for deployment on production HPC systems. Before proceeding with a performance analysis, we outline the design of Darshan 3.0.0 and indicate noteworthy differences from prior versions.

II. Darshan 3.0.0 design

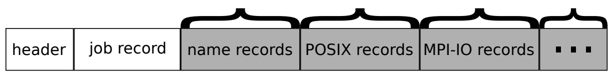
The primary motivation for redesigning Darshan was to modularize its runtime library and log file format in order to facilitate the addition of I/O characterization data from new sources. For example, Darshan's limited instrumentation of the HDF5 and PnetCDF I/O libraries can be extended to glean more information from applications using these data interfaces. Also, platform- and file system–specific data can be captured to correlate with Darshan data instrumented from I/O libraries. Traditionally, Darshan has captured a fixed amount of data from a few different I/O libraries (POSIX, MPI-IO, HDF5, and PnetCDF) but has not exposed a well-defined mechanism for instrumenting new data sources.

This new version meets each of Darshan's original design goals: scalability to leadership-scale HPC systems, transparency to end users, and accuracy of application I/O characterization. We also provide an additional design goal for modularizing Darshan's runtime library and log file format: the specification of a well-defined interface for Darshan to coordinate with "instrumentation modules" at runtime in order to retrieve I/O characterization data from arbitrary data sources.

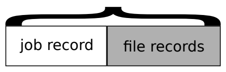
The *instrumentation module* is a Darshan software component that instruments I/O characterization data from some specific source. Instrumentation modules are responsible for defining the data they want to capture, implementing wrapper functions to instrument this data from functions of interest, and defining functions for interfacing with Darshan at shutdown time. Before modules can gather instrumentation data, they must register themselves with Darshan to obtain a memory buffer for storing this data, as well as to indicate to Darshan that the specific module is now active. As the application executes, instrumentation modules intercept functions of interest and extract necessary data to characterize the I/O behavior. Darshan then iterates over active modules at shutdown time to retrieve their corresponding instrumentation buffers, which are then compressed and collectively written to the log file.

We modified the Darshan file format in order to handle I/O characterization data sourced from potentially numerous instrumentation modules. In particular, we made the log file self-describing such that consumers could easily find and extract data records belonging to different instrumentation modules. This feature was not necessary in prior versions of Darshan because log files contained only two distinct types of data, a per-job record and a sequence of per-file records, which were always located at the same logical positions in the log file.

For reference, in Figure 1 we provide diagrams of the log



(a) 3.0.0



(b) 2.x

Fig. 1. Darshan log file formats

file formats for Darshan 3.0.0 as well as 2.x versions of Darshan. The nonshaded portions of the file indicate regions that are collected at a single root process (MPI rank 0), while the shaded regions indicate regions that are gathered collectively across all processes. Braces indicate log file regions that are written by using collective operations (with each process's contributed buffer compressed independently), while other regions are written independently by the root process.

The 3.0.0 file format consists of a header, a job record, and a sequence of name records, followed by sequences of records from each active module. The header stores information about the log file format, including the offset and extent of each region in the log file. Since the offset and extent information of each module is known only after the module's data has been written, the header must be prepended to the log file at the end of the shutdown procedure. Further, we must fix the length of the header so the amount of space to preallocate is known when Darshan begins to shut down, precluding the use of compression on this structure. The job record stores relevant job-level metadata corresponding to the instrumented application. Name records store mappings between unique Darshan identifiers (used for referencing files consistently across modules) and their corresponding full path names. Following the name record region are the sequences of file records from each active instrumentation module, each written by using distinct collective operations.

In contrast, the 2.x log file format consists solely of a job record followed by a stream of Darshan file records. These Darshan file records include data from numerous API levels and a fixed-length suffix of the corresponding file name, rather than storing complete path names as in Darshan 3.0.0. Each Darshan 2.x log is written out by using a single collective operation, with the root rank prepending the job record to its sequence of file records.

III. Test Environment

All experiments were performed on Edison, a Cray XC30 system at NERSC. Edison has 133,824 compute cores on 5,576 compute nodes with 357 terabytes of memory in aggregate. All application data and Darshan logs were stored on one of Edison's scratch Lustre filesystems, which offer 2.1 PB of total disk space and a peak performance of 48 GiB/sec.

Each experiment uses both Darshan 2.3.0 and Darshan 3.0.0-pre3 (a prerelease version of the new modularized Darshan implementation) to compare overhead and other performance metrics between the two versions. In both cases, Darshan was compiled by using the GNU GCC compilers available on Edison. Application executables were also built by using the same compiler.

IV. Performance Analysis

In the following subsections, we analyze the impact of Darshan instrumentation on application I/O performance, the overhead of the Darshan shutdown procedure, and resultant log file sizes for different I/O workloads.

A. Darshan instrumentation overhead

We first analyze the computational overhead of Darshan's method for instrumenting application I/O routines. Darshan interposes a library of I/O wrappers at compile time (for statically linked executables) to intercept and instrument functions of interest. Ideally, the process of interposing these wrappers induces minimal overhead, in order to prevent Darshan from perturbing overall application I/O performance. To measure Darshan's overhead, we can compare the amount of time an example application spends doing I/O when Darshan instrumentation is enabled vs. disabled. We elect to use the IOR benchmark as our example application, because of its popularity in analyzing I/O performance on HPC systems and its high degree of configurability, allowing for the emulation of a range of relevant HPC workloads.

For these experiments, we configured IOR to use a file per process workload, with each process performing I/O using independent MPI-IO operations. Specifically, each of a total of 4,800 application processes (on 200 Edison compute nodes) writes a total of 512 MiB to its own unique file using access sizes of 512 KiB. This workload results in an aggregate file write size of 2,400 GiB and requires nearly 10 million total MPI-IO and POSIX I/O operations (roughly 2,048 operations per process). We built three different IOR executables for these experiments: one with no Darshan instrumentation, one with Darshan 2.3.0, and one with Darshan 3.0.0. We submitted 15 distinct jobs for each IOR version in an attempt to differentiate normal system variance from deviations caused by Darshan. Additionally, to combat system performance variation, we took special care to run all three versions of the benchmark within reasonable temporal timeframes on the same allocation of compute nodes. Specifically, we ran all versions of the benchmark sequentially within the same job script, with delays inserted between runs to help prevent I/O performance from being impacted by previous benchmark executions. We measured the I/O time of each benchmark using the total I/O time reported by IOR, which is just a measure of the duration between the time of the first open of an output file on any process to the time of the last close on any process. We can focus on a single scale for these experiments because the overhead is unlikely to change with scale (Darshan does not perform any I/O or communication within I/O wrappers).

In Figure 2, we provide box plots of the results of these tests for each IOR version. These plots give the minimum, median, and maximum results, as well as the 1st and 3rd quartiles,

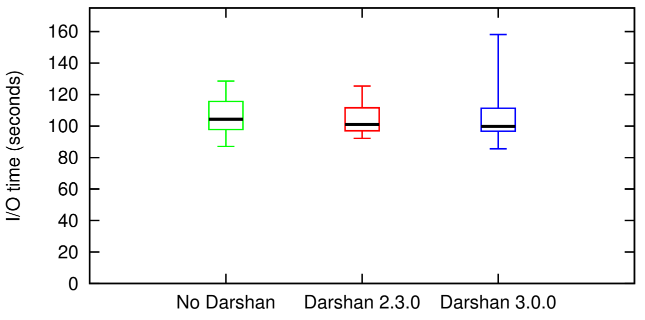


Fig. 2. I/O time reported by an IOR file per process experiment with and without Darshan instrumentation.

to provide an indication of the distribution of I/O times in each case. We observe that Darshan appears to have little impact on the I/O performance of the benchmark, with the median I/O time actually showing a 3–4% decrease in each case where Darshan is enabled (from 104.4 seconds to 101 and 99.9 seconds, respectively, for Darshan 2.3.0 and Darshan 3.0.0). The 1st and 3rd quartile results are also comparable, indicating a high degree of similarity in I/O performance distributions in each test case. We conclude that, as in previous versions of Darshan, Darshan 3.0.0 introduces minimal (if any) measurable runtime overhead.

B. *Darshan shutdown overhead*

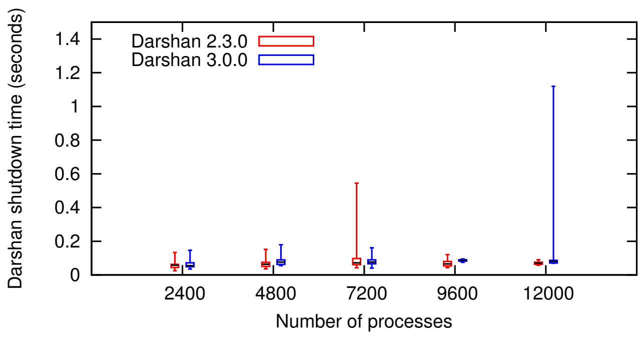
Another important performance consideration of this study is the time taken for Darshan to shut down and generate a log characterizing a given application's I/O behavior. In general, the Darshan shutdown process involves aggregating the output data across all processes, compressing this data, and collectively writing this data out to the Darshan log file. This process is invoked by Darshan after the application has finished processing by intercepting `MPI_Finalize()`.

We note that the shutdown mechanism in Darshan 3.0.0 is more complex than the general procedure mentioned above that Darshan has previously employed. This is a side effect of the new modularized runtime environment and log file format Darshan now uses, as described in Section II. In previous versions, each process would accumulate all Darshan data into a single buffer to be compressed and written collectively to log file. In Darshan 3.0.0, however, the shutdown process proceeds on a per-module basis, with each participating instrumentation module taking turns collectively writing their compressed data to log file. An additional collective write is used for logging Darshan name records to file, and separate independent operations are used for writing the header and job information to the log.

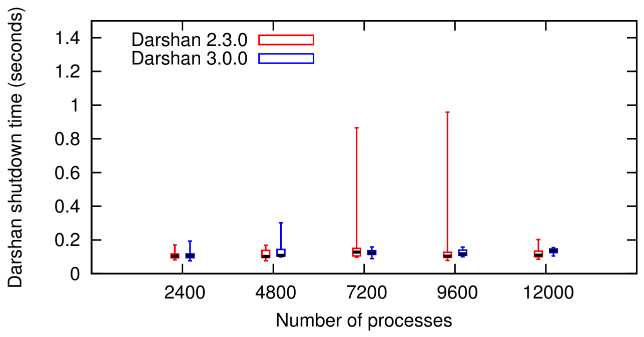
In these experiments, we compare the performance of the shutdown process between both Darshan versions. We use a low-level benchmark that injects synthetic Darshan data records corresponding to different I/O workloads into the Darshan runtime environment, rather than using the typical I/O wrapper routines. We then invoke the Darshan shutdown procedure and instrument the total time taken to complete this process. For each workload and each Darshan version, we collect 10 independent samples of the shutdown benchmark to get a distribution of shutdown times. We also analyze the shutdown benchmark at numerous job sizes, ranging from 2,400 application processes up to 12,000 processes, to evaluate how the shutdown procedure scales in each version. Again, special care was taken to prevent these benchmarks from running concurrently and also to interleave the benchmark executions from each Darshan version. We consider cases where an application uses both the MPI-IO and POSIX interfaces to access either globally shared files or an independent file per process, which are both typical of HPC applications that utilize checkpointing mechanisms. For the shared file workloads, Darshan uses MPI collective communication to determine which file records are shared globally and then to reduce each of these shared records into a single aggregate record. The root process ends up with the reduced aggregate records and is responsible for writing them to log file. As described earlier, these shared file reductions must be performed independently for each module in Darshan 3.0.0.

Figure 3 provides shutdown performance results for each Darshan version when subjected to our target workloads. In the results for the single shared file case given in Figure 3(a), the shutdown times of each version show little discernible difference, with each taking on average about 100 milliseconds to shut down. This overhead remains mostly constant across all scales, save a couple of outlier results. The results in Figure 3(b) show a similar trend for a workload that accesses 1,024 globally shared files, although the average shutdown time is understandably slightly longer in this case. These results show that Darshan 3.0.0 can achieve shutdown performance similar to that of Darshan 2.3.0 for shared file workloads, despite the extra complexity in its shutdown mechanism. That is, the extra steps required to shut down on a per-module basis, as well as to write the Darshan header and name record structures to the log file, appear to have a limited impact on the shutdown performance for these workloads.

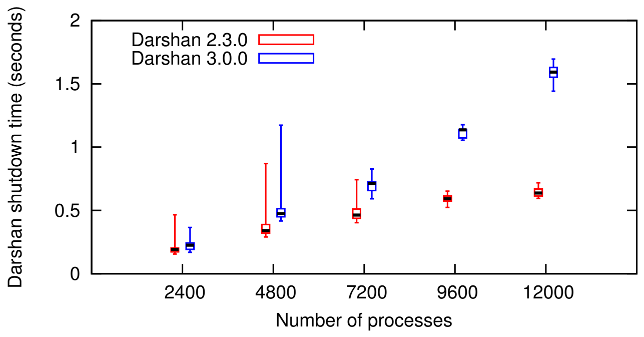
The shutdown overhead results for a file per process workload are given in Figure 3(c). Unlike the previous example, Darshan does not perform reductions of shared file records in this case; instead it just uses collective I/O to write out each process's unique file record. As one might expect, the corresponding shutdown times for these workloads scale linearly with the job size. Darshan 3.0.0 attains slightly slower shutdown times at smaller scales, with the performance disparity increasing with the job size. This is due to the modular log format in Darshan 3.0.0: each module's data is compressed and written independently, thus leading to larger log files (as we explain more in the following section) and more collective I/O operations. Nevertheless, Darshan still aggregates, compresses, and writes its characterization data in less than 2 seconds for all configurations evaluated here.



(a) 1 shared file



(b) 1,024 shared files



(c) 1 file per process

Fig. 3. Shutdown time for each Darshan version when instrumenting typical HPC I/O workloads.

C. *Log file sizes*

As another point of comparison, we consider the sizes of log files generated by the synthetic shared file and file per process benchmarks performed in the preceding section. Table I gives the resultant log file sizes in each case using Darshan's standard runtime compression (libz), as well as the total number of files instrumented in each case, the average number of bytes per file record, and the percentage increase from 2.3.0 logs to 3.0.0 logs. For reference, in Table II we provide the (uncompressed) sizes of relevant log file regions for both Darshan versions. Note that while Darshan 3.0.0 logs have a fixed-length header prepended to the file, the job-level record and the per-file records stored by Darshan are nearly the same size in each version.

For the shared file benchmarks, we consider only one job size because log file sizes for shared records are essentially identical across scales (with differences related only to compression efficiency). With a single shared file, there is a slight 20% increase in log file size from Darshan 2.3.0 to Darshan 3.0.0, which was expected because of the inclusion of a fixed-length header in Darshan 3.0.0 logs. However, this percentage increase in log file size actually grows to 100% in the case of a workload accessing 1,024 shared files, even though both Darshan versions are storing nearly the same amount of data in each case. Similar results hold for log files generated by file per process workloads, with a nearly 100% increase in log file size at all scales evaluated. The reason is that each module's data (two modules are active in this example: MPI-IO and POSIX) is compressed independently, thus leading to less efficient overall compression. This tradeoff enables greater flexibility and extensibility in how instrumentation data is collected and organized within Darshan at the cost of increased log file storage requirements. Still, the largest log file in this experiment is less than 3 MiB in size.

D. *Stress test*

For our final test, we compare the performance of each Darshan version when subjected to an extreme workload atypical of production HPC systems. To do this, we consider the shutdown overhead and log file size of a more heavyweight versions of the file per process workload we considered in earlier sections. Specifically, we modify the file per process workload to use 1,024 unique files per process instead of

Workload	Process Count	Total Files	Darshan 2.3.0		Darshan 3.0.0		Log Size increase
			Log Size	Per-File Size	Log Size	Per-File Size	
1 shared	12,000	1	0.870 KiB	891.0 B	1.050 KiB	1,075.0 B	20.1%
1,024 shared	12,000	1,024	27.342 KiB	27.3 B	54.646 KiB	54.6 B	99.9%
1 FPP	2,400	2,400	0.279 MiB	121.8 B	0.518 MiB	226.2 B	85.8%
	4,800	4,800	0.557 MiB	121.7 B	1.114 MiB	243.3 B	99.9%
	7,200	7,200	0.836 MiB	121.8 B	1.672 MiB	243.5 B	99.9%
	9,600	9,600	1.114 MiB	121.7 B	2.229 MiB	243.5 B	100.1%
	12,000	12,000	1.395 MiB	121.9 B	2.789 MiB	243.7 B	99.9%

TABLE I
DARSHAN 2.3.0 AND 3.0.0 RESULTANT LOG FILE SIZES FOR TYPICAL FILE PER PROCESS AND SHARED FILE WORKLOADS.

TABLE II
LOG FILE COMPONENT SIZES FOR EACH DARSHAN VERSION

Version	Header	Job Record	File Record[†]
Darshan 2.3.0	–	1080 bytes	1328 bytes
Darshan 3.0.0	360 bytes	1064 bytes	1324 bytes

[†] Note: Since Darshan 3.0.0 has distinct file records for each used module, we calculated the total file record size to be the sum of the sizes of the POSIX and MPI-IO records and the sum of the name record stored for each file. Name records are variable in length, so we used the average size of these records as observed in the benchmark results.

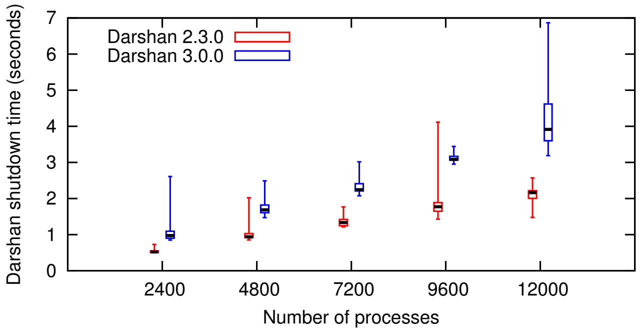


Fig. 4. Shutdown time for each Darshan version when instrumenting a heavyweight 1,024 file per process workload.

just one. A previous study of the systemwide I/O trends of the Intrepid BG/P system at the ALCF using Darshan found that no exemplar application accessed more than 140 files per process [2]. These results offer insight into Darshan's performance when pushed beyond the limits of what would typically be seen in production.

As expected, the shutdown performance of the 1,024 unique files per process workload follows a similar trend to that of the single file per process workload, with the shutdown time scaling linearly with the number of processes. The shutdown times are obviously longer than in the file per process case, but we can see a slight improvement in the scalability of the shutdown procedure for Darshan 3.0.0. This is likely due to Darshan 3.0.0 being able to use larger writes as part of the collective I/O algorithm, since each process contrbiutes 1,024 file records rather than just one. At the largest scale we tested this workload, Darshan is able to shut down completely within 7 seconds even when collecting instrumentation data for over 12 million unique files.

The log files generated for this workload are orders of magnitude larger than the single file per process workloads we considered earlier, requiring on the order of hundreds of MiBs of storage depending on the job size. However, we can see that the percentage increase from Darshan 2.3.0 logs to Darshan 3.0.0 logs has been reduced from 100% to 80% compared with the single file per process case. This is related to increased compression efficiency made possible by compressing many file records on the same process. As further evidence of the increased compression efficiency, we see that the 1,024 file per process workload results in around 50 bytes of log file storage per tracked file compared with the 250 bytes required for the single file per process workload.

E. Log archival

In this section, we provide results indicating how Darshan 3.0.0 logs can be archived to further reduce their long-term storage requirements. As part of its set of log post-processing utilities, Darshan includes a tool for converting Darshan logs (i.e., `darshan-convert`) from their native libz compression format to bzip2 compression instead. The bzip2 compression format generally results in higher compression efficiency, particularly for log files containing a lot of data. However, the `darshan-convert` utility is able to further optimize compression efficiency by condensing all per-process compression streams for a given module into a

Workload	Process count	total files	Darshan 2.3.0		Darshan 3.0.0		Log Size Increase
			Log Size	Per-File Size	Log Size	Per-File Size	
1,024 FPP	2,400	2,457,600	61.282 MiB	26.1 B	111.511 MiB	47.6 B	82.0%
	4,800	4,915,200	125.546 MiB	26.8 B	223.557 MiB	47.7 B	78.1%
	7,200	7,372,800	185.651 MiB	26.4 B	335.228 MiB	47.7 B	80.6%
	9,600	9,830,400	251.213 MiB	26.8 B	447.068 MiB	47.7 B	78.0%
	12,000	12,288,000	309.414 MiB	26.4 B	558.980 MiB	47.7 B	80.7%

TABLE III
DARSHAN 2.3.0 AND 3.0.0 RESULTANT LOG FILE SIZES FOR A HEAVYWEIGHT 1,024 FILE PER PROCESS WORKLOAD.

Workload	Process Count	Total Files	Original Size	Converted Size	Log Size Decrease
1 shared	12,000	1	1.050 KiB	1.264 KiB	-20.4%
1,024 shared	12,000	1,024	54.646 KiB	41.270 KiB	24.5%
1 FPP	12,000	12,000	2.789 MiB	706.040 KiB	75.3%
1,024 FPP	12,000	12,288,000	558.980 MiB	397.312 MiB	28.9%

TABLE IV
DARSHAN 3.0.0 LOG FILE SIZE COMPARISON BEFORE AND AFTER USING THE LOG CONVERT UTILITY.

single compression stream for the entire module, resulting in even smaller output log files. This is especially useful for file per process workloads where each process compresses its data records independently before generating a log file. Table IV provides results indicating the amount of space that can be saved by converting Darshan logs to the bzip2 format. We observe that the single file per process workload benefits greatly from both bzip2 compression and the refactoring of per-process compression streams into per-module compression streams, resulting in a 75% reduction in log file size. We also note that log files containing small amounts of data actually become larger when using bzip2 compression, so it is best suited for log files containing many file records.

Additionally, the `darshan-parser` utility (which is generally used to analyze Darshan log contents in text format) includes flags that allow it to generate an aggregate I/O characterization for all of an application's accessed files. This type of summary information is much smaller in size (i.e., one file record instead of a record for each file accessed by the application), and is amenable to long-term storage. This mechanism could be used to archive a constant size, coarse-grained view of an application's I/O behavior in a database for easy retrieval, for instance.

The bz2 conversion strategy and the aggregate statistic conversion strategy can both be performed by using periodic, sequential cron jobs in order to reduce storage capacity demands if needed on large-scale systems.

V. Conclusions

Our analysis of the overheads of Darshan 3.0.0 has showed it to have a negligible impact on the performance of instrumented HPC applications. Specifically, we have showed the overhead of the new modularized runtime architecture is essentially transparent to application users. In an IOR benchmark execution using 4,800 application processes and performing nearly 10 million distinct instrumented I/O operations, we observed no noticeable perturbations in attained I/O performance compared with a version of the benchmark not linked with Darshan at all. We also demonstrated that while the new shutdown method used by Darshan is less efficient than prior versions, we can still generate Darshan log files in under 2 seconds for a range of different I/O workloads and job sizes typical of HPC systems. We also provided results indicating the sizes of resultant Darshan log files for a number of different benchmarks, with the average log file size per file accessed by an application ranging anywhere from 50 to 250 bytes in the workloads we evaluated. We also identified the root cause for increases in shutdown overheads and log file sizes in Darshan 3.0.0, and we can revisit and potentially can optimize the runtime architecture and file format if this proves problematic in the future.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computer Research, under contract DE-AC02-06CH11357. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 characterization of petascale I/O workloads," in *IEEE International Conference on Cluster Computing and Workshops, 2009. CLUSTER'09*. IEEE, 2009, pp. 1–10.

[2] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, "Understanding and improving computational science storage access through continuous characterization," *ACM Transactions on Storage (TOS)*, vol. 7, no. 3, p. 8, 2011.

[3] P. Carns, K. Harms, R. Latham, and R. Ross, "Performance analysis of Darshan 2.2.3 on the Cray XE6 platform." Argonne National Laboratory (ANL), Tech. Rep., 2012.

[4] P. Carns, Y. Yao, K. Harms, R. Latham, R. Ross, and K. Antypas, "Production I/O characterization on the Cray XE6," in *Proceedings of the Cray User Group meeting*, vol. 2013, 2013.