LDMS Darshan Connector: For Run Time Diagnosis of HPC Application I/O Performance

Sara Walton[1], Omar Aaziz[1], Ana Luisa V. Solórzano[2], Ben Schwaller[1], and Devesh Tiwari[2]

[1]Sandia National Laboratories, Albuquerque, USA
[2]Northeastern University, Boston, USA

***Abstract*—Periodic capture of comprehensive, usable I/O performance data for scientific applications requires an easy-to-use technique to record information throughout the execution without causing substantial performance effects. In this paper, we introduce a unique framework that provides low latency monitoring of I/O event data during run time. We implement a system-level infrastructure that continuously collects I/O application data from an existing I/O characterization tool to enable insights into the I/O application behavior and the components affecting it through analyses and visualizations. In this effort, we evaluate our framework by analyzing sampled I/O data captured from two HPC benchmark applications to understand the I/O behavior during the execution life of the applications. The result shows the utility of capturing I/O application performance and behavior.**

I. Introduction

As more scientific I/O applications are developed and used, the need for improved fidelity and throughput of these applications is more pressing than ever. Much design effort and investment is being put into improving not only the I/O performance of applications but also the performance of related system components (e.g., filesystem elements and networks). Being able to identify, predict and analyze I/O behaviors are critical to ensuring parallel storage systems are utilized efficiently [1].

Variations in I/O performance for an application can be caused by aggregate contention for resources such as file systems and networks. Congestion in these resources may even be caused by the access patterns of the affected application itself [2]. This variation makes it difficult to determine the root cause of I/O related problems and get a thorough understanding of throughput for system-specific behaviors and I/O performance in similar applications across a system.

Generally, the I/O performance is analyzed post-run by application developers, researchers and users using regression testing or other I/O characterization tools that capture the applications behavior. An example of one of these tools is *Darshan*, which monitors and captures I/O information on access patterns from HPC applications [3]. Efforts to identify the origin of I/O performance usually come from the analysed data collected by these I/O characterization tools. Correlations are then made with the environment in which they were run or by comparison with other analyses from other application runs. However, this approach does not enable identification of temporally **when** significant variation of I/O performance occurs during an application run or correlations between such behavior and the file system state, network congestion or other resource contentions.

Execution logs that provide *absolute timestamps* (e.g. time-series) enable users and developers to perform temporal performance analyses, and better understand how the changing state of system components affects I/O performance and variation as well as provide further insight into the I/O patterns of applications. Our *Darshan LDMS Integration* approach provides time series logs of application I/O events and incorporates *absolute timestamps* to provide a runtime timeseries set of application I/O data. This paper makes the following contributions:

- Describes the approach used to expose absolute timestamp data from an existing I/O characterization tool;
- Provides a high level overview of the implementation process and other tools used to collect application I/O data during run time;
- Demonstrates use cases of the *Darshan-LDMS Connector* for two applications with distinct I/O behavior on a production HPC system;
- Utilizes Darshan LDMS data to identify and better understand any root cause(s) of application I/O performance variation run time;
- Presents how this new approach can be integrated with other tools to benefit users to collect and assist in the detection of application I/O performance variances across multiple applications.

II. Background

A. Darshan

Darshan is a lightweight I/O characterization tool that captures I/O access pattern information from HPC applications. [3]. This tool is used to tune applications for increased scientific productivity or performance and is suitable for continuous deployment for workload characterization of large systems [4]. It provides detailed statistics about various types of file accesses made by MPI and non-MPI applications which can be enabled or disabled as desired. These types include POSIX, STDIO, LUSTRE and MDHIM for non-MPI applications and MPI-IO, HDF5 and some PnetCDF for MPI applications [5]. This functionality provides users with a summary of I/O behavior and patterns from an application post-run. It does not provide insights into run time I/O behavior and patterns or concurrent system conditions. This may limit

the ability to use this data to identify the root cause(s) of I/O variability and when during an application run this occurs.

B. LDMS

The Lightweight Distributed Metric Service (LDMS) is a low-overhead production monitoring system that can run on HPC machines with thousands of nodes collecting system data via LDMS daemons running data sampling plugins. Data is typically transported from these *sampler* daemons to *aggregator* daemons using a Remote Direct Memory Access (RDMA) transport to minimize CPU overhead on comopute nodes. Final aggregators in a transport chain store the data to a database. System state insights are achieved by LDMS's synchronous sampling with *absolute timestamps*. This provides a snapshot-like view of system conditions. Transport is performed in a tree structure where the *samplers* (e.g leaves) determine the kind of data sampled, the intermediate aggregators are used for data transport, and the last level aggregators are used for storage. There are a variety of sampler plugins that can be used to collect different system metrics (e.g., memory, CPU, network). Additional functionalities exist in LDMS, such as the *Streams* publish-subscribe functionality, that enable the aggregation of event-based application data. Our framework utilizes this publish-subscribe functionality and the *LDMS Streams API* to transport Darshan's I/O event data that is collected from applications during execution time and store it, along with *absolute timestamps*, in a database.

C. DSOS

The Distributed Scalable Object Store (DSOS) is a database designed to manage large volumes of HPC data [6] efficiently. LDMS uses it to support high data injection rates to enhance query performance and flexible storage management. DSOS has a command line interface for data interaction and various APIs for languages such as Python, C, and C++. A DSOS cluster consists of multiple instances of DSOS daemons, *dsosd*, that run on multiple storage servers on a single cluster. The DSOS Client API can perform parallel queries to all *dsosd* in a DSOS cluster. The results of the queried data are then returned in parallel and sorted based on the index selected by the user. This database and it's Python API are used in this framework for storing and querying the I/O event data.

D. HPC Web Services

The HPC Web Services is an analysis and visualization infrastructure for LDMS [7], that integrates an open-source web application, Grafana [8], with a custom back-end web framework (Django) which calls python modules for analysis and visualization of HPC data. Grafana is an open-source visualization tool tailored towards time-series data from various database sources. Grafana provides charts, graphs, tables, etc. for viewing and analyzing queried data in real time. Using a custom DSOS-Grafana API, the python analysis modules to be used can be specified in a Grafana query. Once specified, that python analysis transforms any data queried from the dashboard before returning the data to Grafana. Grafana enables a wide variety of visualization options for the data and allows users to save and share those visualizations to others. Our framework leverages the HPC Web Services for run time analyses and visualizations of I/O event data collected by Darshan.

III. Darshan LDMS Integration

Darshan collects I/O application data for post-run analysis, and LDMS has a low overhead sampling capability supported by fast storage and a modern web interface. We chose to enhance both Darshan and LDMS by adding a sampling capability that acts as a connector between these two applications to support I/O runtime data sampling and visualization. In this section we present a high-level overview of the design and implementation of the *Darshan LDMS Integration*. The components used to create this infrastructure are:

- The I/O characterization tool, Darshan, to collect application I/O behavior and patterns.
- LDMS to provide and transport live run time data feed about application I/O events. [9]
- DSOS to store and query large amounts of data generated on a production HPC system. [6]
- HPC Web Services to present run time I/O data to enable the user to create new meaningful analyses [7].

LDMS is used to efficiently and scalably collect and transport *synchronous* and *event-based* data with low-overhead. Two key functionalities the *Darshan LDMS Integration* will leverage in order to create the *Darshan-LDMS Connector* are the LDMS *streams* publish-subscribe bus and LDMS transport [10]. In this work, we enhanced LDMS to support application I/O data injection and store it to DSOS. We also modified Darshan to expose an *absolute timestamp* and publish run time I/O events for each rank to *LDMS Streams*. This integration will be described in detail later in paper.

DSOS enables the ability to query the timestamped application I/O data through a variety of APIs while Grafana will provide a front-end interface for visualizing the stored data that has been queried and analyzed using Python based modules on the back-end. With these tools, users can view, edit and share analyses of the data as well as create new meaningful analyses.

IV. Integration Tools

A. Darshan

Darshan is divided into two main parts: 1) *darshan-runtime* which contains the instrumentation of the characterization tool and produces a single log file at the end of each execution summarizing the I/O access patterns used by the application [11]. 2) *darshan-util* which is intended for analyzing log files produced by darshan-runtime [11]. The *Darshan LDMS Integration* focuses on the *darshan-runtime* [5] as this is where the source code of I/O event data is recorded by Darshan.

Darshan tracks the start, duration and end time of an application run via the C function *clock_gettime()* and converts the result into seconds and passes the result to a struct that is then used to report the summary log files [12]. Therefore, in order

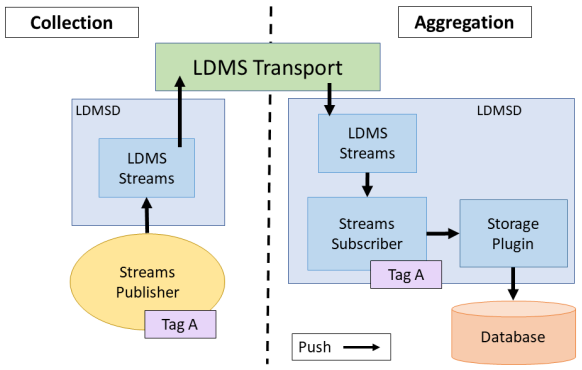

Fig. 1: Overview of the LDMS event data collection. Application data is *pushed* by publishing data to the *LDMS Streams* which is then *pushed* through the LDMS transport to the *LDMS Streams* LDMS aggregator (right) where the data is *pushed* to the streams subscriber (Tag A) and stored to a database.

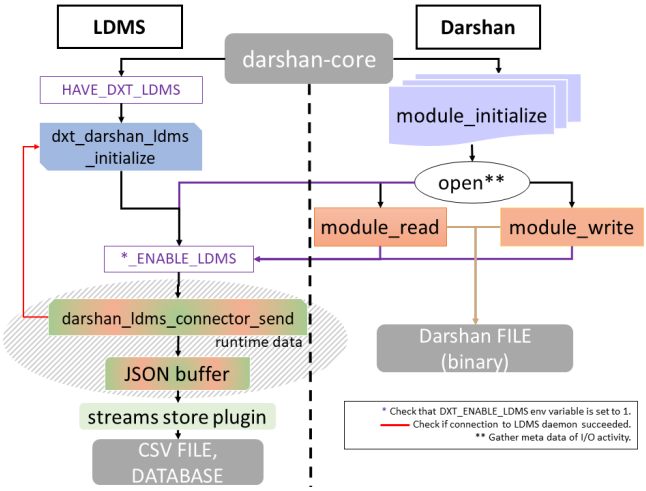

Fig. 2: Overview of the *Darshan-LDMS Connector* design and how it collects I/O data for each read, write, open and close events per rank from Darshan. The LDMS library must be linked against the Darshan build in order to utilize the *LDMS Streams* functionality and store plugins.

to retrieve the *absolute timestamp* and include it into the I/O event data during run time, a time struct pointer was added to the function call that used *clock_gettime()* in *darshan-runtime*. This pointer was passed through all of Darshan's modules and the *absolute timestamp* was collected. This was the preferred method as it required minimal changes to Darshan's source code and no additional overhead or latency between the function call and recording of the *absolute timestamp*.

B. LDMS Streams

LDMSD refers to an LDMS daemon that provides the capability of data collection, transport, and/or storage. An *LDMSD's* configuration, including plugins, determines its functionality and capabilities [9]. Daemons on the compute nodes run sampler plugins and transport is achieved through multi-hop *aggregation*. In this work we utilized two levels of LDMS aggregator daemons [9] with the second level utilizing the DSOS storage plugin to store the I/O event data.

The *Darshan LDMS Integration* leverages the LDMS transport to support the injection and transport of application I/O data which requires a *push-based* method to reduce the amount of memory consumed and data loss on the node as well as reduce the latency between the time in which the event occurs and when it is recorded. A *pull-based* method would require buffering to hold an unknown number of events between subsequent pulls. Also, the transported data format needs to support *variable-length* events because the I/O event data will vary in size.

We created an I/O targeted LDMS streams store that utilizes the LDMS publish-subscribe bus capability, *LDMS Streams*, to publish I/O event data. *LDMS Streams* is intended for publishing and subscribing to an *LDMS streams tag*. This tag needs to be specified in LDMS daemons and *plugins* in order to publish event data to *LDMS Streams* and receive this published *LDMS Streams* data that match the tag. This process and the *push-based* method can be seen in Figure 1. Event data can be specified as either `string` or `JSON` format. The *LDMS Streams* API was modified to support long application connections and message injections. *LDMS Streams* uses best effort without a reconnect or resend for delivery and does not cache it's data so the published data can only be received after subscription. *LDMS Streams* enables data from any source to be injected into the LDMS transport.

C. Darshan Connector

The most recent version of Darshan allows for full tracing of application I/O workloads using their DXT instrumentation module which can be enabled and disabled as desired at runtime. DXT provides high-fidelity traces for an application's I/O workload vs Darshan's traditional I/O summary data and currently traces POSIX and MPI-IO layers [5]. Our design leverages the additional I/O tracing Darshan's DXT module plugin provides through the new *Darshan-LDMS Connector* capability.

The *Darshan-LDMS Connector* functionality collects both DXT data and Darshan's original I/O data and optionally publishes a message in JSON format to the *LDMS Streams* interface as seen in Figure 3. The *absolute timestamp* is also included in this message with the given name "timestamp". LDMS then transports the I/O event data across the aggregators and stores it to a *DSOS* database where Grafana can access and query it. the *Darshan-LDMS Connector* currently uses a single unique *LDMS Stream tag* for this data source. For the file level access types that DXT does not trace or that have different name-value pairs, a value of "N/A" or "-1" is given in the JSON message. For example, Darshan's POSIX module (IEE standard that establishes a set of guidelines for compatibility and portability between operating systems)

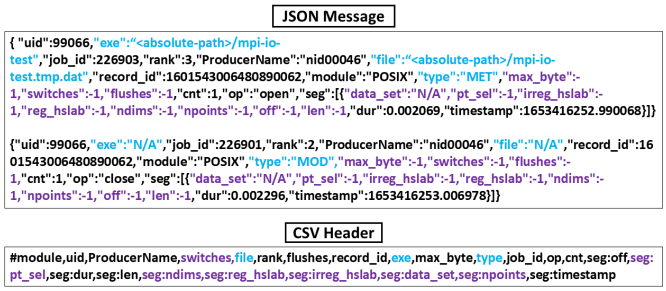

Fig. 3: Output of a MPI-IO Darshan test run in the JSON format (top image), and the CSV file header (bottom). The `name:value` pairs in light blue indicate meta data stored, while the light purple indicates the file level access data not applicable to POSIX. The `"seg"` is a list containing multiple `name:value` pairs.

traces the number of bytes read and written per operation (e.g. `"max_byte"`) and number of times access alternated between reads and writes (e.g. `"switches"`). The I/O operations shown in Figure 3 are *"open"* and *"close"* which are not applicable to these traces. Therefore, `"max_byte"` and `"switches"` are not present in Darshan and their corresponding JSON message names are set to `"-1"` by the *Darshan-LDMS Connector*.

Darshan has a large number of metrics it uses for I/O tracing and post-processing calculations. Currently our framework collects a subset of these metrics to publish to *LDMS Streams*, as presented in Figure 3. These metrics will provide the ability to create new I/O behavior analyses and visualizations to get further insights of the application I/O behavior, and reveal correlations between I/O performance variability and system behavior. Table I depicts the names and definitions of each metric in the JSON file. Depending on the `"type"` input, the absolute directory of the Darshan file output and executable will be recorded and published to *LDMS Streams*. If `"type"` is set to `"MET"` (e.g. "meta"), the absolute directories will be recorded. Otherwise, it will receive the value "N/A" if set to `"MOD"`(e.g. "module"). The `"type"` will be set to "MET" for open I/O events, which are the Darshan I/O records that have permanent values during the application execution, such as the rank, file and node name. The `"type"` is set to "MOD" for all other I/O events to reduce the message size and latency when sending the data through an HPC production system pipeline.

D. Storage: DSOS Database

DSOS is built on the Scalable Object Store (SOS) database [6] and was intended to address the domain-specific needs of large-scale HPC monitoring. It was chosen as the preferred monitoring database at Sandia because it allows for interaction via a command line interface which allows for fast query testing and data examination. DSOS also provides scalable data ingest and the ability to query large volumes of data which is required for the large amounts of data to be ingested and stored. To sort though the published *LDMS Streams* data, combinations of the job ID, rank and timestamp are used to create joint indices where each index provided a different query performance. An example of this is using `job_rank_time` which will order the data by job, rank, and timestamp. This index can provide better performance when searching for a specific job within the database.

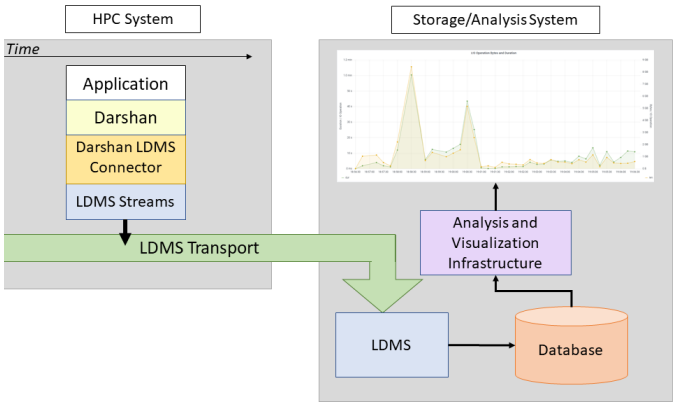

Fig. 4: Overview of the *Darshan LDMS Integration* where the *Darshan-LDMS Connector* is used to intercept the I/O behavior Darshan is collecting utilizes the various tools to publish, store, analyze and view runtime I/O behavior.

E. Analysis and Visualization: HPC Web Services

The HPC Web Services [7] is an infrastructure that provides the analysis and visualization components of our approach. The front-end for the service website is Grafana [8] and the back-end consists of Python analysis modules. The HPC Web Services also provide instant analysis where data can be analyzed and viewed in real time as opposed to the traditional method of querying the results of analyzed data from a separate database.

Grafana is an open-source visualization application that provides various charts, graphs and alerts for supported data sources. It can support multiple data formats but is best suited for timeseries data. It has storage plugins for many database technologies in order to query and render data from multiple data sources. The *Darshan LDMS Integration* implemented a storage plugin for the DSOS database in order to query this data and visualize it on the Grafana web interface [8] using the HPC Web Services infrastructure. An overview of the this integration can be seen in Figure 4.

Python analysis modules are used to produce meaningful visualizations on the queried data from the DSOS database. With these modules, queried data is converted into a pandas dataframe to allow for easier application of complex calculations, transformations and aggregations on the data. The type of analysis module is specified in the Grafana web interface. These python modules applied to our *Darshan LDMS Integration* data will demonstrate how runtime I/O data can provide further insights and understanding into application

uuid	User ID of the job run
exe	Absolute directory of the application executable
module	Name of the Darshan module data being collected
ProducerName	Name of the compute node the application is running on
switches	Number of times access alternated between read and write
file	Absolute directory of the filename where the operations are performed
rank	Rank of the processes at I/O
flushes	Number of "flush" operations. It is the HDF5 file flush operations for H5F, and the dataset flush operations for H5
record_id	Darshan file record ID of the file the dataset belongs to
max_byte	Highest offset byte read and written per operation
type	The type of JSON data being published: MOD for gathering module data or MET for gathering static meta data
job_id	The Job ID of the application run
op	Type of operation being performed (i.e. read, write, open, close)
cnt	The count of the operations performed per module per rank. Resets to 0 after each "close" operation
seg	A list containing metrics names per operation per rank
seg:pt_sel	HDF5 number of different access selections
seg:dur	Duration of each operation performed for the given rank (i.e. a rank takes "X" time to perform a r/w/o/c operation)
seg:len	Number of bytes read/written per operation per rank
seg:ndims	HDF5 number of dimensions in dataset's dataspace
seg:reg_hslab	HDF5 number of regular hyperslabs
seg:irreg_hslab	HDF5 number of irregular hyperslabs
seg:data_set	HDF5 dataset name
seg:npoints	HDF5 number of points in dataset's dataspace
seg:timestamp	End time of given operation per rank (in epoch time)

TABLE I: Metrics defined in the JSON file published to the *Darshan LDMS Integration*.

I/O behavior, patterns, performance variability, and any correlations these may have with behaviors of system components.

V. Experimental Methodology

This section presents our experimental methodology to evaluate our framework using two applications: HACC-IO, and the Darshan MPI-IO benchmark. We performed the experiments on a Cray HPC cluster using NFS and Lustre file systems.

A. Applications

- HACC-IO is the I/O proxy for the large scientific application: Hardware Accelerated Cosmology Code (HACC), an N-body framework that simulates the evolution of mass in the universe with short and long-range interactions [13]. The long-range solvers implement an underlying 3D FFT. HACC-IO is an MPI code that simulates the POSIX, MPI collective, and MPI independent I/O patterns of fault tolerance HACC checkpoints. It takes a number of particles per rank as input, writes out the simulated checkpoint information into a file, and then reads it back for validation. We ran HACC-IO with several configurations to simulate different workloads on the NFS and Lustre file systems. Table IIb shows the different run configurations.
- MPI-IO-TEST benchmark is a Darshan utility that exists in the code distribution to test the MPI I/O performance on HPC machines. It can produce iterations of messages with different block sizes sent from various MPI ranks. It can also simulate collective and independent MPI I/O methods. We experimented with NFS vs. Lustre and collective vs. independent MPI I/O. We ran the benchmark with four configurations on 22 nodes and set the number of iterations to 10 and the block size to 16MB. Table IIa shows the different configuration used.

B. Evaluation System

We experiment using several I/O loads on a Cray XC40 system (Voltrino) at Sandia National Laboratories. The system has 24 diskless nodes with Dual Intel Xeon Haswell E5-2698 v3 @ 2.30GHz 16 cores, 32 threads/socket, 64 GB DDR3-1866MHz memory. The interconnect is the Cray Aries with a DragonFly topology. The machine connects to two file systems: a network file system (NFS) and a Lustre file system (LFS).

C. Enviroment

We configured the HPC cluster Voltrino with LDMS samplers on the compute nodes and one LDMS aggregator on the login node. LDMS uses the Cray UGNI RDMA interface to transfer Darshan streams data, and other system state metrics, from the compute nodes to the aggregator on the login node. The aggregator on the login node transmits the data to another LDMS aggregator on a monitoring cluster, Shirley, for analysis and storage. Shirley also hosts the HPC web services consisting of the Grafana application and DSOS database.

We ran the applications with our enhanced Darshan library that wraps the I/O functions dynamically, for each MPI rank, to sample I/O data and transmit it, using the streams API, to the LDMS running on the node local to the transmitting MPI rank. We set the `LD_Preload` environment variable to point to the Darshan library shared objects, which contain the sampling wrappers, prior to running the applications.

VI. Results

This section describes the significance of collecting run time application I/O event data using our *Darshan LDMS*

Integration approach. We present performance analyses that show how the new metrics, captured in a time series along with *absolute timestamps*, can provide more insight into I/O behavior than summary statistics alone. Additionally, such representation can facilitate the correlation of I/O performance with system component behaviors (e.g., network and filesystem congestion) which can also be represented in a Grafana dashboard.

Without the *Darshan-LDMS Connector*, it would not be possible to create the meaningful analyses and visualizations shown in Figures6-8. In contrast, Figure 5 shows the aggregate I/O behavior which can be created with Darshan alone. This figure does not provide the timeseries data and thus in-depth insights into I/O behavior like the *Darshan LDMS Integration* does. Note that the Darshan eXtended Trace (DXT) plugin that we leverage in this work does provide time series capture capability. However, due to memory constraints it will not typically be able to capture these time series for a whole job run. Additionally the timestamps in the DXT time series are relative to the job start rather than absolute.

A. Experiments and Overhead

Each application was tested on both the Lustre and the NFS file system with several configurations for each application run. All application experiments were repeated 5 times for both the *Darshan-LDMS Connector* and Darshan only (i.e. no LDMS implementation) scenarios. In total there were 40 experiments run with a separate job submission for each. The details of these runs are shown in Table II.

The average of the 5 execution times (e.g., Average Runtime (s)) for Darshan and the *Darshan-LDMS Connector* (dC in the table) was used to calculate the percent overhead of LDMS. Because of the system availability, the runtimes with Darshan were only performed and recorded 1-2 weeks before the experiments with the *Darshan-LDMS Connector*. As seen from Table IIa, the overhead of LDMS on Darshans' MPI-IO-TEST benchmark for three experiments shows a decrease in overall runtime with the *Darshan-LDMS Connector*. Since this is not feasible, the runtime improvement seen with the *Darshan-LDMS Connector* is most likely due to the NFS and Lustre file systems performance variation where (and when) these experiments were performed. This behavior will be further investigated by conducting a new set of randomized experiments during dedicated time on a system to minimize variability due to competing applications and the effects of resource performance variations.

The HACC-IO application, seen in Table IIb was similar to the MPI-IO-TEST benchmark regarding a shorter runtime with the *Darshan-LDMS Connector* for both file systems. Again, this is most likely due to differences in Lustre and NFS file system loading during different experimens. The other two experiments, NFS with 10 million particles and Lustre with 5 million particles, show an overhead of 0.84% and 12.01%, respectively. The experiment with 0.84% overhead indicates no significant effect on the applications runtime. In contrast, the experiment with 12.01% overhead shows a longer runtime with the *Darshan-LDMS Connector* which is most likely due to performance variation in both file systems which we will investigate in the near future. The variance across all experiments is so large that the performance overhead calculations are inconclusive.

MPI-IO-TEST				
File System	NFS		Lustre	
Nodes	22		22	
Block Size	16*1024*1024		16*1024*1024	
Iterations	10		10	
Collective	Yes	No	Yes	No
Avg. Messages	50390	6397	25770	15676
Rate (msgs/sec)	37	7	95	38
Average Runtime(s)				
Darshan	1376.67	880.46	249.97	428.18
dC	1355.35	858.68	270.98	414.35
% Overhead	-1.55%	-2.47%	8.41%	-3.23%
Standard Deviation(s)				
Darshan	48.18	29.43	2.85	31.49
dC	96.63	76.58	1.07	8.17
% Variance	-5.25%	-8.10%	9.22%	2.39%

(a) MPI-IO

HACC-IO				
File System	NFS		Lustre	
Nodes	16		16	
Particles/Rank	5000000	10000000	5000000	10000000
Avg. Messages	1663	1774	1995	1711
Rate (msgs/sec)	2	1	3	2
Average Runtime(s)				
Darshan	882.46	1353.87	417.14	1616.87
dC	775.24	1365.24	467.24	1027.44
% Overhead	-12.15%	0.84%	12.01%	-36.45%
Standard Deviation(s)				
Darshan	37.08	87.24	25.03	154.53
dC	53.68	46.97	142.77	256.62
% Variance	-14.65%	4.08%	-17.25%	-47.36%

(b) HACC-IO

TABLE II: Overview of each experiment configuration, target file system, average elapsed time(s) and standard deviation(s) from 5 runs, calculated overhead of LDMS and variance of the runs.

B. Analysis and Grafana Output

Figure 5 shows the number of I/O requests per node for close and open operations for two jobs using the same input for the HACC-IO application on Lustre using 10 million particles per rank. The root cause of this variation is under investigation as we would expect identical runs to have identical behavior with respect to location and number of open and close events.

Figure 6 shows the duration of the reads and writes per rank for each execution (`job_id` metric) of the MPI-IO benchmark without using collective operations. We notice a similar behavior for the I/O operations duration for all jobs except the second one (`job_id 2`). It presents a mean duration of 6.75 seconds for reads and 78s for writes, while the other jobs had a mean duration of 0.05s for reads and 54s

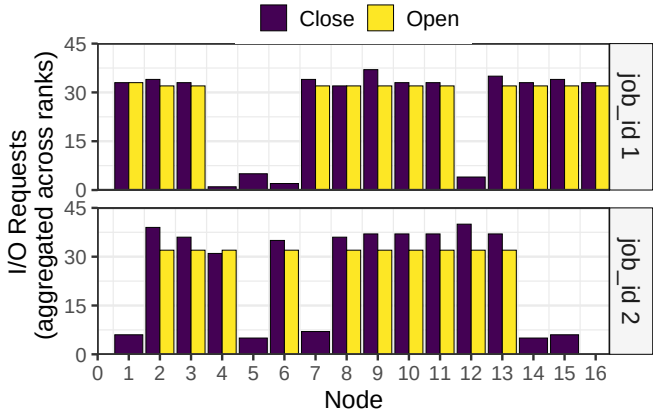

Fig. 5: The amount of I/O operations for HACC-IO using the same input.

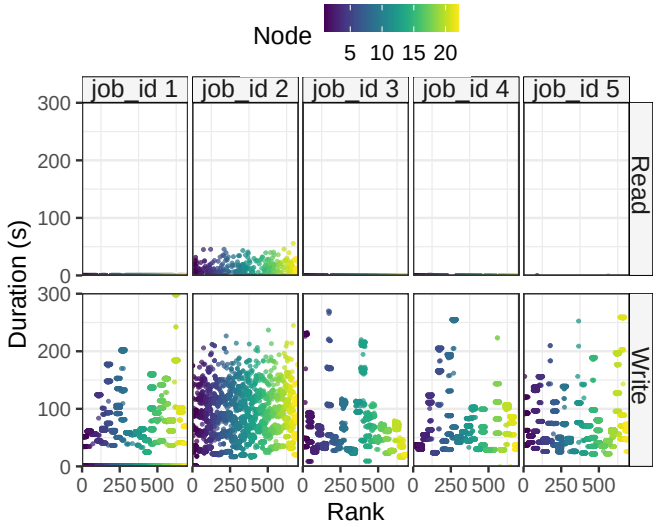

Fig. 6: Jobs for the MPI-IO benchmark without collective operations presented variability in the number and duration of I/O operations.

for writes. With the collected logs, we can perform a spatial performance analysis to understand the variability in the I/O behavior per system component, in this case, per node and rank.

Using the absolute timestamps collected we can temporally view where, in the application execution, the variability of a job occurred and better understand the I/O behavior. Figure 7 presents the duration and occurrence of I/O operations throughout the MPI-IO benchmark for `job_id 2`. We can identify the application I/O pattern of performing writes during the ten iterations, and the likewise the ten read iterations at the end though these are smaller and less distinct than the writes. It can also be seen that in general writes became progressively slower over the application execution time.

The same job is also represented in Figure 8 using the

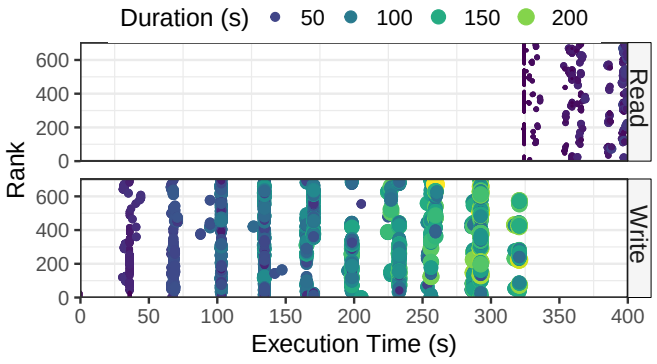

Fig. 7: Distribution of read and write operations throughout the execution time for MPI-IO `job_id 2`, can reveal the application I/O pattern, and where in time, relative to the beginning of the application execution, there were faster and slower operations.

Grafana interface. This figure presents an aggregate time series across all ranks of the number of I/O writes (blues), reads (green), and bytes sent. This time series shows the read and write behaviors throughout execution time which provides further insights into I/O occurrences and size. For example, we can identify the two instances in which a write above 20GB occurs and the single instance in which a read above 10GB occurs. A developer or user can then use this information to create new and meaningful analyses about their applications I/O operations and patterns. Grafana offers an interactive front-end view where users can easily filter to visualize specific time intevals and metrics. Such representation using the absolute timestamps facilitates the correlation of I/O performance with system component state and behavioral characteristics (e.g., congestion in networks and filesystems) which can also be represented in Grafana dashboards.

Without the kind of data provided by the *Darshan-LDMS Connector*, it would not be possible to create the meaningful analyses and visualizations shown in Figures6-8. In contrast, Figure 5 shows the aggregate I/O behavior which can be created with Darshan alone without the DXT plugin. This figure does not provide the timeseries data and thus in-depth insights into I/O behavior like the *Darshan LDMS Integration* does.

VII. Related Work

Extensive work has been performed to improve I/O performance and behavior. The PASSION Runtime Library for parallel I/O proposed by Syracuse University [14] works to optimize I/O intensive applications through Data Prefetching and Data Sieving. The authors of "IOPin: Runtime Profiling of Parallel I/O in HPC Systems" propose dynamic instrumentation to show the interactions between a parallel I/O application and the file system [15]. "Design and Implementation of a Parallel I/O Runtime System for Irregular Applications" [16]

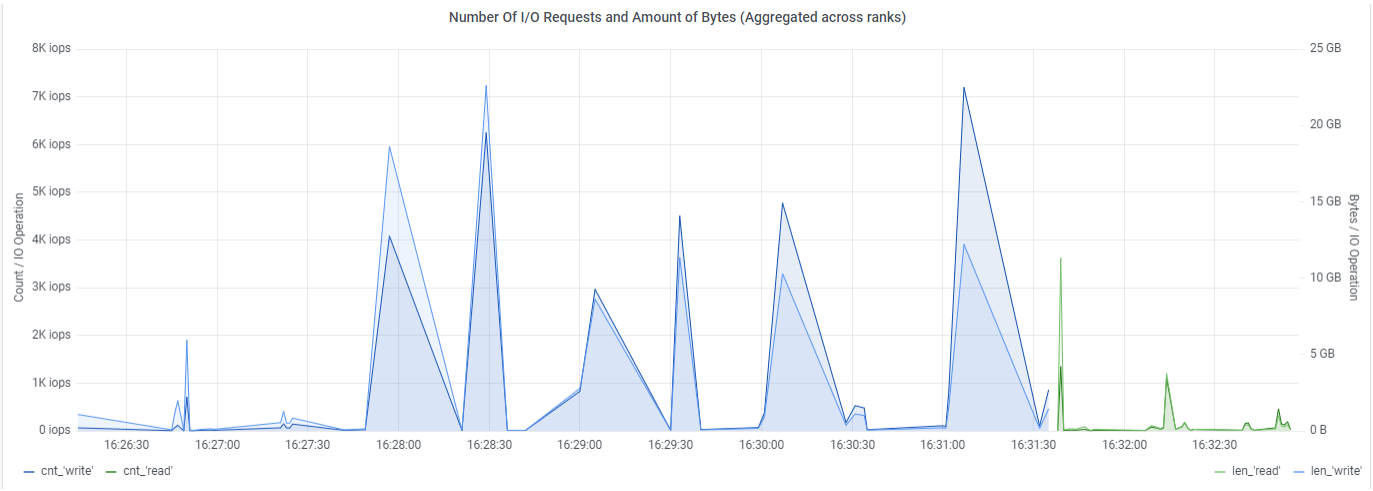

Fig. 8: Graphana visualization of MPI-IO `job_id 2` writes (blue) and reads (green) operations and number of bytes per operation, using the absolute timestamp metric collected with *Darshan LDMS Integration*.

presents two different collective I/O techniques for improving I/O performance.

Darshan was our preferred I/O characterization tool because the Darshan's eXtended Tracing (DXT) [5] instrumentation module provides high-fidelity traces for an application's I/O workload vs Darshan's traditional I/O summary data [5]. It also collects timestamped data which made it possible to expose the *absolute timestamp* for collecting runtime timeseries data.

Other open-source I/O tools that collect runtime timeseries data do exist. The linux command, iostat [17] collects system I/O device statistics and generates reports about the I/O loads between physical disks. The ioprof [18] tool provides insights into I/O workloads. However, these tools do not provide the extensive I/O tracing capability (e.g. detailed statistics of individual I/O operations) that is provided by this framework.

This work differs from these approaches because we *leverage and enhance* existing applications and tools to design an infrastructure that creates runtime analyses and visualizations from detailed traces of application I/O events during execution time. The *Darshan LDMS Integration* integrates LDMS's *absolute timestamped* data collection and storage capabilities [10] with Darshan [4] to collect runtime application I/O data. Further, our choice of the DSOS database for storage of our event-based application I/O data enables efficient queries of large volumes of data as well as python analysis modules and an open-source web application for runtime analyses and visualizations.

VIII. Future Work

This paper covered the *Darshan LDMS Integration* design and implementation of the *Darshan-LDMS Connector* which collected I/O data from the Darshan I/O characterization tool to create new time series data sets that enable further insights into I/O behavior and patterns. Five key components were used to develop this design: the application I/O event data collector (Darshan), lightweight data transport (LDMS Streams), efficient storage (DSOS), analysis (Python modules), and visualization (Grafana). Results of this design add enhancements to both LDMS and Darshan tools as well as create new insights and provide a better understanding of application I/O performance and behavior.

Our next steps are to further expand the *Darshan-LDMS Connector* and it's capabilities by including more I/O event data and demonstrating advanced insights into correlations between I/O performance and system behavior and providing the capability for overhead reduction through sampling and/or aggregation techniques that still provide enough resolution for a user to gain run time insight into the I/O behavioral characteristics of an application and to correlate these characteristics with those of related system components. We will also be performing more overhead analysis over a variety of I/O intensive applications.

The *Darshan LDMS Integration* will be made available as an optional "module" plugin to the Darshan tool so Darshan users can collect time series data without increasing memory impact on compute nodes and better understand applications I/O performance across HPC systems and clusters.

IX. Acknowledgment

The authors would like to thank Jim Brandt (SNL), for useful discussions and suggestions in this work and Darshan contributors, Phil Carns (ANL) and Shane Snider (ANL), for insights about Darshan architecture.

References

[1] E. Costa, T. Patel, B. Schwaller, J. M. Brandt, and D. Tiwari, "Systematically inferring i/o performance variability by examining repetitive job behavior," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476186

[2] M. Folk and V. Choi, "Factors affecting i/o performance when accessing large arrays in hdf5 on ncsa's teragrid cluster," in *Parallel io performance and geodata v. 0.4.doc*, 2005, pp. 1–2.

[3] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 characterization of petascale I/O workloads," in *Proc. 2009 Workshop on Interfaces and Architectures for Scientific Data Storage*, 2009.

[4] Darshan, "DARSHAN: HPC I/O Characterization Tool." [Online]. Available: https://www.mcs.anl.gov/research/projects/darshan/

[5] ——, "Darshan-runtime installation and usage." [Online]. Available: https://www.mcs.anl.gov/research/projects/darshan/docs/darshan-runtime.html

[6] "Ovis/SOS," http://github.com/ovis-hpc/sos.

[7] B. Schwaller, N. Tucker, T. Tucker, B. Allan, and J. Brandt, "HPC system data pipeline to enable meaningful insights through analysis-driven visualizations," in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2020, pp. 433–441.

[8] G. Labs, "Grafana: The Open Observability Platform." [Online]. Available: https://grafana.com

[9] "Ovis-wiki," https://github.com/ovis-hpc/ovis-wiki/wiki, March 2021.

[10] "Ovis/LDMS," http://github.com/ovis-hpc/ovis.

[11] Darshan, "DARSHAN: HPC I/O Characterization Tool." [Online]. Available: https://www.mcs.anl.gov/research/projects/darshan/documentation/

[12] "darshan-hpc/darshan," https://github.com/darshan-hpc/darshan.

[13] S. Habib, V. Morozov, N. Frontiere, H. Finkel, A. Pope, and K. Heitmann, "Hacc: Extreme scaling and performance across diverse architectures," in *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2013, pp. 1–10.

[14] R. Thakur, R. Bordawekar, A. Choudhary, R. Ponnusamy, and T. Singh, "Passion runtime library for parallel i/o," in *Proceedings Scalable Parallel Libraries Conference*, 1994, pp. 119–128.

[15] S. J. Kim, S. W. Son, W.-k. Liao, M. Kandemir, R. Thakur, and A. Choudhary, "Iopin: Runtime profiling of parallel i/o in hpc systems," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, 2012, pp. 18–23.

[16] J. No, S.-S. Park, J. Carretero, A. Choudhary, and P. Chen, "Design and implementation of a parallel i/o runtime system for irregular applications," in *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, 1998, pp. 280–284.

[17] "iostat," https://linux.die.net/man/1/iostat.

[18] "ioprof," https://github.com/intel/ioprof.