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

1st Sara Walton
Sandia National Laboratories (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

Abstract—**Allowing for further insight into I/O behavior and patterns has become increasingly important. The I/O behavior depends on a large number of components such as the applications access pattern, the computing system architecture, I/O libraries, file system, mode of access, data size, and the storage configuration and layout. Changes in these components can create high variations in I/O performance and behavior which creates a lack of understanding about the application I/O and very difficult to identify which of these components are affecting I/O variability and behavior. This paper introduces a unique framework that provides low latency monitoring of I/O event data during run time. This is done through the implementation of a system-level infrastructure that continuously collects I/O application data from an existing I/O characterization tool. This allows insights into the I/O application behavior and the components affecting it through analyses and visualizations. The framework allows users to better understand throughput for system-specific behaviors, variations of I/O performance of similar applications across a system and identify correlations between I/O and system behavior.**

I. Introduction

As more scientific applications are developed and used, the need for improved fidelity and throughput is more pressing than ever and much design effort and investments are being put into improving not only the application but also the system components. Being able to identify, predict and analyze I/O behaviors are critical to ensuring parallel storage systems are utilized efficiently. However, I/O performance continues to show high variations on large-scale production conditions in many cases **Cite NE**. This variation makes it difficult to determine the root cause of I/O related problems and have a thorough understanding of throughput for system-specific behaviors and I/O performance in similar applications across a system. Further, not knowing what the cause is will directly affect the user and developer as unwanted time, effort and investment will need to be put into solving the issue.

Variations in I/O can could be caused by the system behavior such as the file system (e.g. buffering, file transfer, interrupt handling), network congestion, system resource contentions, or the access patterns of the application itself.

Generally, the I/O performance is analyzed post-run by application developers, researchers and users in the form of regression testing or other I/O characterization tools that capture the applications I/O behavior. An example of one of these tools is *Darshan* which captures I/O information on access patterns from HPC applications [1]. Detailed information will be covered in the *Approach* section. In the case where an I/O performance problems are observed, efforts to identify this usually come from any identified correlation between analyses of various applications or the time in which these applications were tested. However, this approach does not provide the ability to know *when* an I/O performance variability occurs during an application run and, if the developer or user wishes to, identify any correlations between the file system, network congestion or resource contentions and the I/O performance.

The *absolute timestamps* provides run time (e.g. timeseries) data that users and developers could use to better understand how these changing components affect the I/O performance variation as well as provide further insight into the application I/O pattern.

Therefore, we introduce our *Darshan LDMS Integration* approach that incorporates the *absolute timestamps* to provide a run time set of application I/O data for a deeper insight into the I/O behavior and performance. This is a work in progress paper and will cover the following goals:

- Describe the approach used to expose absolute timestamp data from an existing I/O characterization tool and utilizing this data to help identify and better understand any root cause(s) of application I/O performance variation run time.
- Provide a high level overview of the implementation process and other tools used to collect application I/O data during run time.
- Demonstrate use cases of the *Darshan-LDMS Connector* for various applications on a production HPC system.
- How this new approach provides the ability to collect and assist in the detection of application I/O performance variances across multiple applications.

II. Background

Darshan is a lightweight I/O characterization tool that captures I/O access pattern information from HPC applications. [1] This will be described in detail in Section III. This I/O characterization tool will be used for tracing and collecting detailed I/O event data.

The Lightweight Distributed Metric Service (LDMS) is a low-overhead system that collects and transports HPC data for OVIS via *samplers* and *plugins*. [2] OVIS which is a

modular system for collecting, analyzing, storing, transporting and visualization HPC data in order to provide further insights into the system state, resource utilization and performance. A *sampler* is a type of daemon that collects the data while the *plugin* determines the type of data to be sampled, aggregated or stored and a *sampler* is the set configuration for collecting the data specified by the plugin. There are a variety of samplers and other plugins that can be written for the LDMS API (in C). The system state insights are achieved by LDMS's *absolute-timestamp* view of system conditions through multi-hop *aggregation* and the LDMS Transport. An LDMS daemon, *LDMSD*, aggregator supports multiple levels, networks and security domains so data can be sent to various locations. Additional functionalities, such as the *LDMS Streams API*, has been developed to allow for aggregation of event-based application data. This framework utilizes *LDMS Streams API* to collect Darshan's I/O event data during execution time and store the *timestamped* data to a database.

The Distributed Scalable Object Store (DSOS) is a storage database designed to efficiently manage large volumes of HPC data [3]. It supports high data injection rates, has an enhanced query performance and flexible storage management. DSOS has a command line interface for data interaction and has various program API's 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 re 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.

The HPC Web Services is an analysis and visualization infrastructure [4], that integrates an open-source web application, Grafana [5] with python modules for analysis and visualization of HPC data. Grafana is an open-source visualization tool for various data sources and has a front-end interface website, Grafana dashbaord, that provides charts, graphs, tables, etc. for viewing and analyzing queried data in real time. The python analysis modules are specified in a Grafana dashboard where the queries are performed. Once specified, that python analysis transforms any data queried from the dashboard in real time. Data can then be filtered and viewed in differerent time-frames and formate in the Grafana dashboard as well as be saved and shared to other users who have access to the webpage. This framework will leverage the HPC Web Services for run time analyses and visualizations of the I/O event data.

III. Approach

This section provides a high-level overview of the design and implementation of the *Darshan LDMS Integration* and the components used to create this infrastructure. **Might comment out to the end of the enumerate list to remove the "fluff"** This approach will provide run time insights about application I/O by using the following tools:

1) The I/O characterization tool, Darshan, to collect application I/O behavior and patterns. However, this tool does not report the *absolute timestamp* so modifications to the code were made to expose this data.
2) LDMS to provide and transport live run time data feed about application I/O events. [6]
3) DSOS to store and query large amounts of data generated on a production HPC system. [3]
4) HPC Web Services to present run time I/O data. The timeseries data will enable the user to identify when a variability occurs as well as create new meaningful analyses. [4]

Darshan is used to tune applications for increased scientific productivity or performance and is suitable for full time deployment for workload characterization of large systems [7]. It provides detailed statistics about various level file accesses made by MPI and non-MPI applications which can be enabled or disabled as desired. These levels include POSIX, STDIO, LUSTRE and MDHIM for non-MPI applications and MPI-IO, HDF5 and some PnetCDF for MPI applications [8]. This functionality provides users with a summary of I/O behavior and patterns from an application run but it does so post-run. Therefore, it does not allow insights into *run time* I/O behavior and patterns which makes it nearly impossible to identify the root cause(s) of I/O variability and when this occurs.

LDMS is used to efficiently collect and transport scalable *synchronous* and *event-based* data with low-overhead. Two key functionalities it has that will be leveraged in the *Darshan LDMS Integration* and create the *Darshan-LDMS Connector* are the *LDMS Streams* and transport [9]. In the *Darshan LDMS Integration* , the LDMS was enhanced to support application I/O data injection and store to DSOS while Darshan was modified to expose the *absolute timestamp* and publish run time I/O events for each rank to the *LDMS Streams*. This integration will be described in detail later on.

DSOS will enable 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.

The implementation of LDMS into Darshan along with the storage, analysis and visualization components that make up this design will provide detailed insights into the I/O behavior and patterns during run time. This insight will allow users and researchers to better understand how application I/O variability correlates with overall system behavior (e.g. file system, network congestion, etc.) how the time of day affects the I/O performance, how the I/O pattern within a run affects the I/O performance of read and write and why the read and write I/O performance patterns are different and independent of each other. Further, the occurrence of any I/O variability can be identified during run time.
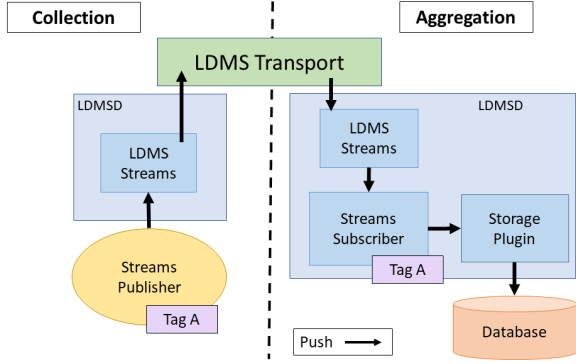
Fig. 1: Overview of the LDMS event data collection. Application data is *pushed* by publishing data to the *LDMS Streams* which is then *pushed* through the LDMS transport to the *LDMS Streams* LDMS aggregator (right) where the data is *pushed* to the streams subscriber (Tag A) and stored to a database.
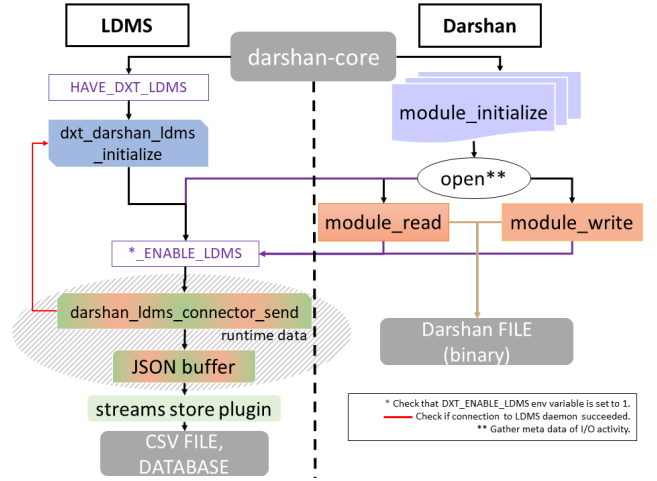

Fig. 2: Overview of the *Darshan-LDMS Connector* design and how it collects I/O data for each read, write, open and close events per rank from Darshan. The LDMS library must be linked against the Darshan build in order to utilize the *LDMS Streams* functionality and store plugins.

IV. DARSHAN LDMS INTEGRATION

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 a the end of each execution summarizing the I/O access patterns used by the application [10]. 2) *darshan-util* which is intended for analyzing log files produced by darshan-runtime [10]. The *Darshan LDMS Integration* focuses on the *darshan-runtime* [8] 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 [11]. Therefore, in order 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 and latency between the function call and recording of the *absolute timestamp*.

B. LDMS Streams

The word, *LDMSD*, refers to an LDMS daemon that provides the capability of data collection, transport and storage and their *plugins* determine the functionality of these capabilities [6]. Daemons on the compute nodes run sampler plugins and transport is achieved through multi-hop *aggregation*. LDMS had two levels of aggregator daemons [6]which can utilize storage plugins to store any sets of data into various formats so long as it's specified beforehand. This can be seen in **Figure XX**

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 a buffering to hold an unknown number of events between pulls. Also, the transported data format needs to support *variable-length* events because the I/O data may will most likely vary in size.

This leads to the LDMS *publish-subscribe bus* capability , *LDMS Streams*, which has been enhanced to support I/O event data. This capability is intended for publishing and subscribing to an *LDMS streams tag*. The 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 XX**. 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. The *LDMS Streams* allows the ability for any data 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 [8]. This design

leverages the additional I/O tracing Darshan's DXT 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". The LDMS transport then transports the I/O event data to a *DSOS* database where Grafana can access and query this data. the *Darshan-LDMS Connector* currently uses a single unique *LDMS Stream tag* for this data source. For the file level accesses that DXT does not trace or for file level access type that have different name-value pairs, a value of "N/A" or "-1" is given in the JSON message.

Darshan has a large number of metrics it uses for I/O tracing and post-processing calculations. The current stages of this framework collects a subset of these metrics to publish to *LDMS Streams* as seen in Figure 3. These metrics provide the most value to the user as they will provide the ability to create new I/O behavior analyses and insights as well as identify correlations between I/O performance variability and system behavior. The metric names are listed in the JSON message are defined as follows:

- `uuid`: The user ID of the job run.
- `exe`: Absolute directory of 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 times the "flush" operation was performed. For H5F and H5D it is the HDF5 file flush and dataset flush operation counts, respectively.
- `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. It is either set to `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 the following metric 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.)

JSON Message

{ "uid":99066,"exe":"<absolute-path>/mpi-io-test","job_id":226903,"rank":3,"ProducerName":"nid00046","file":"<absolute-path>/mpi-io-test.tmp.dat","record_id":1601543006480890062,"module":"POSIX","type":"MET","max_byte":-1,"switches":-1,"flushes":-1,"cnt":1,"op":"open","seg":[{"data_set":"N/A","pt_sel":-1,"irreg_hslab":-1,"reg_hslab":-1,"ndims":-1,"npoints":-1,"off":-1,"len":-1,"dur":0.002069,"timestamp":1653416252.990068}]}

{"uid":99066,"exe":"N/A","job_id":226901,"rank":2,"ProducerName":"nid00046","file":"N/A","record_id":1601543006480890062,"module":"POSIX","type":"MOD","max_byte":-1,"switches":-1,"flushes":-1,"cnt":1,"op":"close","seg":[{"data_set":"N/A","pt_sel":-1,"irreg_hslab":-1,"reg_hslab":-1,"ndims":-1,"npoints":-1,"off":-1,"len":-1,"dur":0.002296,"timestamp":1653416253.006978}]}

CSV Header

#module,uid,ProducerName,switches,file,rank,flushes,record_id,exe,max_byte,type,job_id,op,cnt,seg:off,seg:pt_sel,seg:dur,seg:len,seg:ndims,seg:reg_hslab,seg:irreg_hslab,seg:data_set,seg:npoints,seg:timestamp
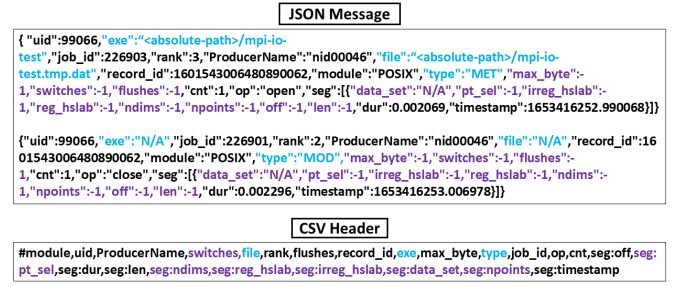
Fig. 3: Output of a simple mpi-io test run. The JSON message is what is being published to the *LDMS Streams* interface that then converts it to CSV and store to *DSOS*. The `name:value` pairs in light blue indicate meta data stored while the light purple indicates the data that does not apply to this file level access and are given the values of `"N/A"` or `"-1"`. The `"seg"` is a list containing multiple `name:value` pairs.

- `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).

Depending on the `"type"` name, the absolute directory of the Darshan file output and executable will be recorded and published to *LDMS Streams*. This is set to either `"MET"`(e.g. "meta") or `"MOD"`(e.g. "module"). This name is set to "MET" only for open I/O events as this is where Darshan records all I/O data that will not change until the application is complete (e.g. rank, file, node name, etc.). The name is set to "MOD" is 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 [3] and was intended to address the domain-specific needs of large-scale HPC monitoring. It was chosen as the preferred database 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
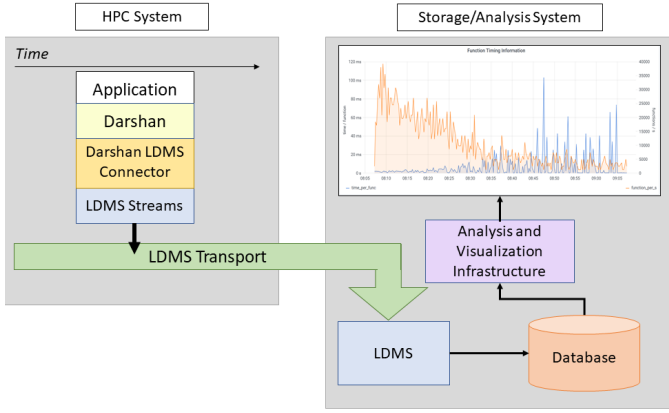
Fig. 4: Overview of the *Darshan LDMS Integration* where the *Darshan-LDMS Connector* is used to intercept the I/O behavior Darshan is collecting utilizes the various tools to publish, store, analyze and view runtime I/O behavior.

the data by job, rank then timestamp and then search the data by a specific rank within a specific job over time.

E. Analysis and Visualization: HPC Web Services

The HPC Web Services [4] is an infrastructure that consists of the analysis and visualization components of this approach. Any data queries start from a front-end application and transferred to a back-end application that are running on an HPC cluster. In this case the front-end website is Grafana [5] 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 [5]. An overview of the this integration can be seen in figure **XXX**.

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. This is where the *Darshan LDMS Integration* will demonstrate how runtime I/O data will provide further insights and understanding into application I/O behavior, patterns, performance variability and any correlations these have with the system behavior.

File System	Particles/Rank	Runtime	Nodes
NFS	1000000	1210.06	16
	2000000	2455.60	16
Luster	1000000	1476.643	16
	2000000	??	16

TABLE I: HACC-IO run configurations, targeted file system, and runtime

V. EXPERIMENTAL METHODOLOGY

In this section we will evaluate our framework using three applications: HACC-IO and MPI-IO benchmarks, and sw4 scientific application using a Cray HPC cluster.

A. Applications

- HACC-IO is the I/O proxy for the large scientific Hardware Accelerated Cosmology Code (HACC), an N-body framework that simulates the evolution of mass in the universe with short and long-range interactions [12]. 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 HACC and takes a number of particles per rank as input and filename to write out the simulation data. We ran HACC-IO with several configurations to simulate different workloads on the NFS and Luster file systems. Table I shows the different run configurations.
- HMMER is a suite of applications that profiles are hidden Markov model (HMM) to search similar protein sequences [13]. HMMER has a building code that uses MPI to build a database by concatenating multiple profiles Stockholm alignment files. In our experiment, we used the Pfam-A.seed [14] file to generate a large Pfam-A.hmm. We ran HMMER with 32 MPI ranks on one node, and we ran it in two configurations where we point the database file to NFS and then Luster, respectively.
- Dashan 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 sizes sent from various MPI ranks. It can also simulate collective and independent MPI I/O methods. We ran several configurations of the benchmark where we **[TODO1: ...]**
- sw4 is a geodynamics code that solves 3D seismic wave equations with local meshrefinement [15]. sw4 accepts an input file that specifies the 3D grid simulation size, and we selected a size that uses about 50% of the available memory to memic a realistic run of the application.

B. Evaluation System

We experiment using several I/O loads on the Voltrino Cray XC40 system 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, and connected with a Cray Aries DragonFly interconnect. The machine has two file systems:

the network file system (NFS) and the Luster file system (LFS).[TODO2: File systems???]

C. Enviroment

Voltrino, run LDMS samplers on the compute nodes and one LDMS aggregator on the head node. LDMS uses the UGNI interface to transfer darshan streams data and other performance metrics from the compute nodes to the head node. The aggregator on the head node transmits the data to another LDMS aggregator on another cluster, Shirley, for analysis and storage. Shirley, host the HPC web services Grafana application and the DSOS database. Darshan can wrap the I/O function in an application by linking the executables staticky and dynamic. Our framework uses dynamic linking to collect darshan data. So we set the `LD_Preload` environment variable to point the path of the darshan library shared objects which have the LDMS streams API calls to send the data through the LDMS daemons live on the compute nodes.

VI. RESULTS

This section covers what significance of the approach to collecting runtime application I/O data and how the new analyses helped provide more insight into I/O behavior.

VII. RELATED WORK

Extensive research has already been proposed in literature to provide further insights into I/O behavior. The PASSION Runtime Library for parallel I/O proposed by Syracuse University [16] that optimizes I/O intensive applications through Data Prefetching and Data Sieving, the IOPin: Runtime Profiling of Parallel I/O in HPC Systems proposes a dynamic instrumentation to show the interactions from a parallel I/O application to the file system [17] and Design and Implementation of a Parallel I/O Runtime System for Irregular Applications [18] that proposes two different collective I/O techniques for improving I/O performance.

Darshan was the preferred I/O characterization tool because they had their Darshan's eXtended Tracing (DXT) [8] instrumentation module that provides high-fidelity traces for an application's I/O workload vs Darshan's traditional I/O summary data [8]. Other open-source I/O tools that we have come across or are aware of do not have this extensive I/O tracing capability which is leveraged in this work.

This work in progress 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 *time stamped* data collection and storage capabilities [9] with Darshan [7] to collect runtime application I/O data. Further, a database is implemented to allow for 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

Taking a stab at this This paper covered the *Darshan LDMS Integration* design and implementation of the *Darshan-LDMS Connector* which collected I/O data from an I/O characterization tool to create a new timeseries that allows for further insights into I/O behavior and patterns. Five key components were used to develop this design which were the I/O event data (Darshan), data collection (LDMS Streams), storage (DSOS), analysis (Python modules) and visualization (Grafana). These results of this design proved to enhance both LDMS and Darshan tools as well as create new insights and provide a better understanding to application I/O performance and behavior.

The 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. We hope the *Darshan LDMS Integration* will be available as an optional "module" plugin in Darshan so their users may use this tool to better understand their applications I/O performance across HPC systems and clusters.

REFERENCES

[1] 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.

[2] "Ovis," http://ovis.ca.sandia.gov.

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

[4] 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.

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

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

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

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

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

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

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

[12] 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.

[13] S. Eddy, "Hmmer user's guide," *Department of Genetics, Washington University School of Medicine*, vol. 2, no. 1, p. 13, 1992.

[14] E. L. Sonnhammer, S. R. Eddy, E. Birney, A. Bateman, and R. Durbin, "Pfam: multiple sequence alignments and hmm-profiles of protein domains," *Nucleic acids research*, vol. 26, no. 1, pp. 320–322, 1998.

[15] N. A. Petersson and B. Sjögreen, "Sw4, version 2.0. computational infrastructure for geodynamics,. doi: 10.5281/zenodo. 1045297."

[16] 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.

[17] 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.

[18] 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.