# LDMS Darshan Connector: Integrating LDMS Streams 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*—Might need to redo this later Improving I/O performance usually 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 which could lead to many negative effects such as network congestion, poor scheduling and indicates a lack of understanding I/O variability. It also makes it difficult to detect the root cause of variations in I/O performance This paper introduces a unique methodology that provides low latency monitoring of I/O performance variations 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 performance and the components affecting it through the analyses and visualizations. This methodology addresses the challenge of poor understanding of throughput for system-specific behaviors and variations of I/O performance of similar applications across a system. This paper demonstrates the implementation and assessments of this method and how it can be used on various HPC applications.**

## 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**. Some of these cases include running applications on clusters during the weekend, separate and disjoint time zones and read I/O's of runs within the same cluster. 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 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.This paper will be structured in the following format with the main key factors being:

- 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. Related Work

**Look for papers that used a similar approach**

## 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 [1] 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) The Lightweight Distributed Metric Service (LDMS) [2] to provide and transport live run time data feed about application I/O events.
3) A storage database, Distributed Scalable Object Store (DSOS) [3] to store and query large amounts of data generated on a production HPC system.
4) An analysis and visualization infrastructure, HPC Web Services [4], that use Grafana [5] and python analysis modules 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.

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 [6]. 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 [7]. 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 [8]. 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.

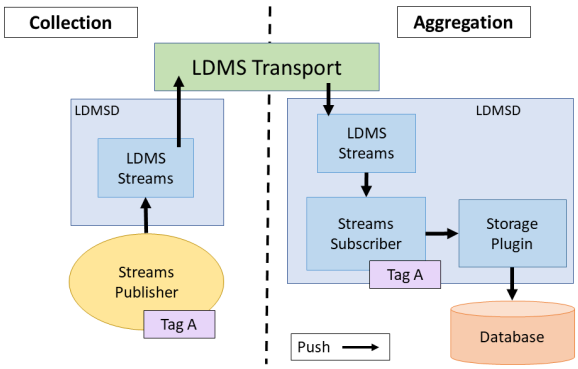


Fig. 1: Overview of the LDMS event data collection. Application data is *pushed* by publishing data to the *LDMS Streams* which is then *pushed* through the LDMS transport to the *LDMS Streams* LDMS aggregator (right) where the data is *pushed* to the streams subscriber (Tag A) and stored to a database.

## 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 [9]. 2) *darshan-util* which is intended for analyzing log files produced by darshan-runtime [9]. The *Darshan LDMS Integration* focuses on the *darshan-runtime* [7] 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 [10]. 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*.

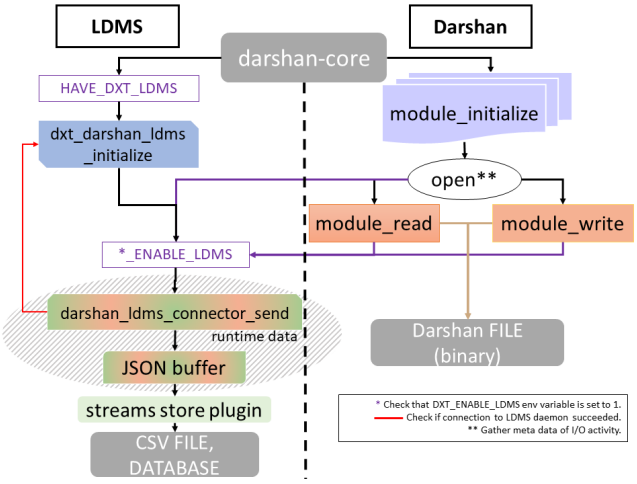


Fig. 2: Overview of the *Darshan-LDMS Connector* design and how it collects I/O data for each read, write, open and close events per rank from Darshan. The LDMS library must be linked against the Darshan build in order to utilize the *LDMS Streams* functionality and store plugins.

### 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 [2]. Daemons on the compute nodes run sampler plugins and transport is achieved through multi-hop *aggregation*. LDMS had two levels of aggregator daemons [2]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

**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":160154300648089062,"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":160154300648089062,"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"`.

*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 Darshan eXtended Tracing (DXT) instrumentation module which can be enabled and disabled as desired at runtime. This provides high-fidelity traces for an application's I/O workload vs Darshan's traditional I/O summary data [7]. DXT currently traces POSIX and MPI-IO layers [7]. 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 the message in JSON format to the *LDMS Streams* interface. 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 in figure **xxx**.

Depending on the `"type"` name, the absolute directory of the Darshan file output and executable will be recorded and published to *LDMS Streams*. This decision is based on the `"type"` name that 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

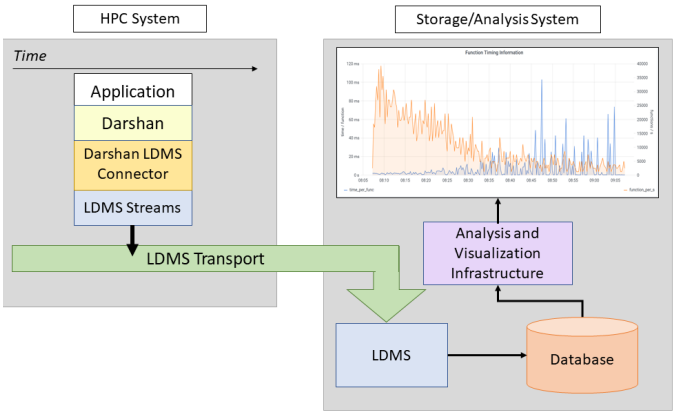


Fig. 4: Overview of the *Darshan LDMS Integration* where the *Darshan-LDMS Connector* is used to intercept the I/O behavior Darshan is collecting utilizes the various tools to publish, store, analyze and view runtime I/O behavior.

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 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.

## V. Use Cases

This section provides various use cases of the *Darshan LDMS Integration* timeseries data that will be used to create new meaningful analyses and insights in the I/O performance variability during an application run.

**Explain the different scenarios we will be testing:**

- **Applications: SWFFT, sw4, sw4lite. Standard baseline: mpi-test.**
- **Explain what each application does, why it's being tested and how the test was performed (i.e. number of nodes, etc.)**
- **Explain the analysis used to analyze the I/O data and how they provide further insight into the I/O behavior and can allow for correlations between I/O performance and system behavior.**
- **show Grafana graphs, Darshan output (maybe) JSON and any tables (if applicable).**

### *A. 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.

- **Analyse the results of the use cases**
- **What is the significance of the results?**
- **Throw in a picture of the Grafana Dashboard for each use case**
- **How do these results satisfy and solve the problems described in the the "Problems and Approach" section.**

## VI. Future Work

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-wiki," https://github.com/ovis-hpc/ovis-wiki/wiki, March 2021.

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

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

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

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

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