I/O Monitoring for portable HPC applications

A. Vijayakumar*
Department of Physics & Astronomy, Texas A&M University, College Station, Texas USA 77843-4242
(Dated: August 4, 2023)

Modern HEP workflows are becoming increasingly scaled and complex: understanding the I/O behavior of these workflows could solve potential bottlenecks and help with optimization. Darshan is a lightweight I/O characterization tool that captures concise views and entire traces (DXT) of applications' I/O behavior. Darshan helps to gain insight and understand the HEP workflow's I/O patterns. However, a significant overhead would reduce productivity and could be potentially detrimental. My research focuses on measuring the overhead Darshan adds to the ATLAS workflow, which is used to process billions of events collected or simulated by the ATLAS experiment. I compared the running times of Athena with and without Darshan instrumentation to extract the overhead. The measurement was repeated both locally and inside the container, Singularity, thus creating 4 configurations. Each configuration was repeated 4 times, for 1, and 8 processes, and 100, 1000, and 3600 events per process, to measure the uncertainty. The results indicated that running Athena inside the container, Singularity, does not have a significant impact on the running time. Most of the runtime differences observed were within the 1st standard deviation. Therefore, Darshan has been found to have negligible overhead when it instruments Athena, which indicates that Darshan is suitable for I/O monitoring of the HEP workflows.

I. INTRODUCTION

I/O behavior is one of the biggest limiting factors of HEP workflows, especially as they become more scaled and complex. The case study used for this report was the ATLAS experiment [1] at the Large Hadron Collider (LHC) at CERN. In the ATLAS detector, particles collide at the center, and that debris forms into new particles sent flying in all directions, which are then bent using a magnet to measure their momenta.

In order to gain a better understanding of the I/O behavior, Darshan[2], a lightweight HPC I/O characterization tool, is used to instrument its workflow. ATLAS deals with billions of events making any additional runtime detrimental to the research process. To make a case for Darshan to monitor the ATLAS workflow, the overhead introduced by Darshan is crucial. A significant overhead in this case would mean anything that scales with the number of events or processes, at larger runs, would add up drastically.

ATLAS offline software's (Athena)[3] workflow has many different stages for event processing. These stages include generation, simulation, reconstruction, filtering, and analysis. To measure Darshan's overhead most effectively, I used the most I/O intensive stage of the workflow, the filtering stage. The filtering stage filters out unnecessary information associated with events. Athena was also used in its preferred mode for the filtering stage now and in the future, AthenaMP. AthenaMP is the multi-process mode of Athena. It uses multiple processes to split the workload for event processing making it more efficient.

Many HEP workflows, including Athena, will be run through containers for portability. ATLAS uses Singularity[4]. A container application is a tool that allows programs to be run on unique software environments. To ensure that there isn't an excessive overhead from the use of a container and that using a container doesn't affect Darshan's overhead, I will also measure the overhead that the container added.

II. MEASUREMENT

A. Darshan

Like it was mentioned before, Darshan is a lightweight I/O characterization tool, that can be used to understand the I/O behavior of a program. Darshan has 2 components, Darshan-runtime, and Darshan-util. Darshan-runtime is used to instrument applications, and Darshan-util helps with processing the log files created by the runtime to generate insight into what the I/O of the application looks like. PyDarshan, a python module provided by Darshan-util, helps users to do customized analysis. An example of the data that Darshan can provide can be seen in 1. I will be using PyDarshan and the code in 2 to extract the runtime from Darshan's log files.

B. Extracting Runtimes

There are 2 ways to go about extracting runtimes, one is through Darshan's logs as shown above, and the other is through Athena. I wrote a Python script that automatically retrieved the runtimes from both log files and created a table that compared both runtimes to see if there were significant differences between them and especially if those differences scale with the number of events or processes. I ran this for event counts 100, 1000, and 3600 per process, with 1 and 8 processes, making for a total of 6 runs in each configuration inside and outside a container, then compared the runtimes between the Darshan log files and Athena log files for each corresponding test case.

From the table in 3, there is a less than 30-second difference between runtimes retrieved from the Dar-

* ady.vijay@tamu.edu

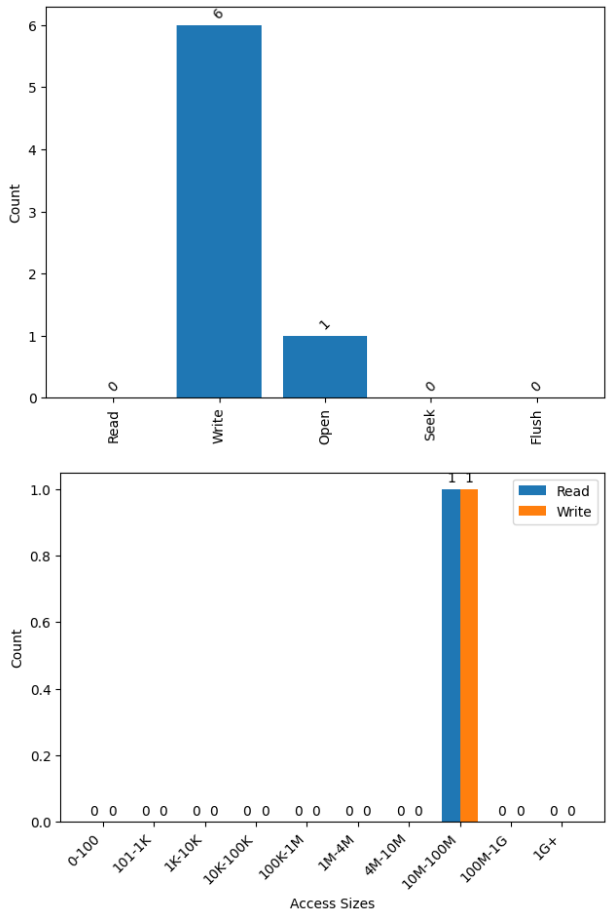

FIG. 1. These are the access sizes and counts for a MPI-I/O tutorial implementation, and an example of the types of data that Darshan can provide

shan log file and Athena log file. The time interval of the performance record in the Athena log is 32 seconds, making any difference less than that time span irrelevant. For all future references of runtime, they are extracted from the Athena log files.

C. Data Collection

The goal of the experiments was to repeat each run four times, to gain an uncertainty measurement. The ATLAS derivation job was run at different event counts of 100, 1000, and 3600 per process, and I was going to do so for processes 1, and 8. Higher process counts were run, however, due to issues mentioned in the conclusion, those runs were not used.

Over the course of these runs, the filtering stage had to be run approximately 100 times, making running each one individually extremely tedious and time-consuming. In order to make the data collection process more efficient, I created a script that would automatically store the log files of each run into a specific folder named by the convention of "Number Of Events Per Process-Number Of Processes". The data collected through this process will be used to study the added runtime of using a container and to measure

```
import darshan

report = darshan.DarshanReport("runtime/mpi-io-test.darshan", read_all=True)  # Default behavior
report.info()
```

```
Filename:       runtime/mpi-io-test.darshan
Executable:     /home/avijayakumar/runtime/mpi-io-test
Times:          2023-06-08 14:09:09 to 2023-06-08 14:09:09
Run time:       0.0466 (s)
Processes:      1
JobID:          205874
UID:            22229
Modules in Log: ['POSIX', 'STDIO', 'HEATMAP']
Loaded Records: {'POSIX': 1, 'STDIO': 1}
Name Records:   6
Darshan/Hints:  {'lib_ver': '3.4.2', 'h': 'romio_no_indep_rw=true;cb_nodes=4'}
DarshanReport:  id(47202098936080) (tmp)
```

```
report.metadata["job"]["run_time"]
```

```
0.046597957611083984
```

```
sum_read = 0
for i in report.records["POSIX"]:
    sum_read += i["counters"][14]
    sum_read += i["counters"][15]
POSIX_bytes = sum_read/1048576
print(POSIX_bytes)
```

```
32.0
```

```
sum_read = 0
for i in report.records["STDIO"]:
    sum_read += i["counters"][7]
    sum_read += i["counters"][6]
STDIO_bytes = sum_read/1048576
print(STDIO_bytes)
```

```
0.0003070831298828125
```

FIG. 2. The top image contains general information that could be extracted from Darshan's log files through PyDarshan. The bottom image contains some of the performance information and runtimes, which will be used further in the report.

	100-1	100-8	1000-1	1000-8	3600-1	3600-8
Athena Outside Container	280	310	837	868	2233	2449
Athena Inside Container	280	280	899	1148	2263	2357
Darshan Outside Container	304	313	860	884	2238	2454
Darshan Inside Container	288	289	924	1149	2285	2385
Diff Outside Container	-24	-3	-23	-16	-5	-5
Diff Inside Container	-8	-9	-25	-1	-22	-28

FIG. 3. Runtime table with the different runtimes inside and outside Singularity, retrieved from Athena's log files and Darshan's log files.

the overhead added by Darshan when instrumenting Athena.

III. ANALYSIS AND RESULTS

The first data set to analyze was the runtime difference between the inside and outside of the container. I did this for both the Darshan data set and the data set collected without Darshan instrumentation. In figures 5, 6, 7, and 10, the top panel shows the runtime vs

```
numruns='100 1000 3600'
numproc='1 8 16 32'
for i in ${numruns}; do
        for j in ${numproc}; do
                mkdir $i-$j
                cd $i-$j
                cp ~/scripts/athenaevents
                cp ~/scripts/run_derivation.sh ~/athena/$
filename/$i-$j
                ./run_derivation.sh $i $j
                cp ~/darshanlogs/$(date +%Y)/$(date +%-m)
/$(date +%-d)/avijayak_python* ~/athena/$filename/$i-$j
                rm -rf ~/darshanlogs/$(date +%Y)/$(date +
%-m)/$(date +%-d)
                mkdir ~/darshanlogs/$(date +%Y)/$(date +%
-m)/$(date +%-d)
                cd ~/athena/$filename
        done
done
```

FIG. 4. The script above creates a directory in order to store the log files. Runs each configuration of the program then copies the created log files over to their respective directories.

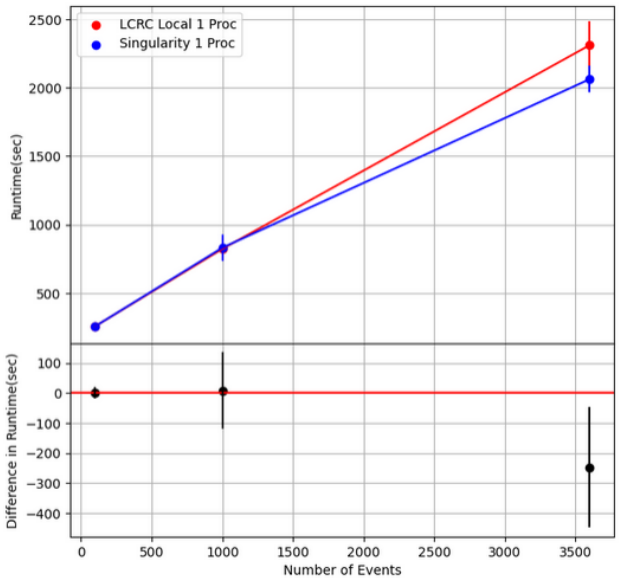

FIG. 5. Graph of runtime for 1 process without Darshan.

the number of events per process for the local running time, shown with the red line, and the running time inside Singularity, indicated with the blue line. The red line in the bottom panel represents the same red line in the top panel, and the axis represents the difference of the points on the blue graph from the red graph.

The error bar represents the uncertainty on the runtime, which is extracted from the four repeated measurements. No difference could be seen in the running time within the uncertainty. Thus, no visible impact has been found on the running time when running Athena within Singularity container.

As for the second set of results that I cared for, Darshan's overhead, I examine the difference in the fraction of the Athena running time without Darshan. The equation is $(Darshan - Athena)/Athena$ where Darshan represents the runtime with Darshan instrumentation and Athena represents the runtime without

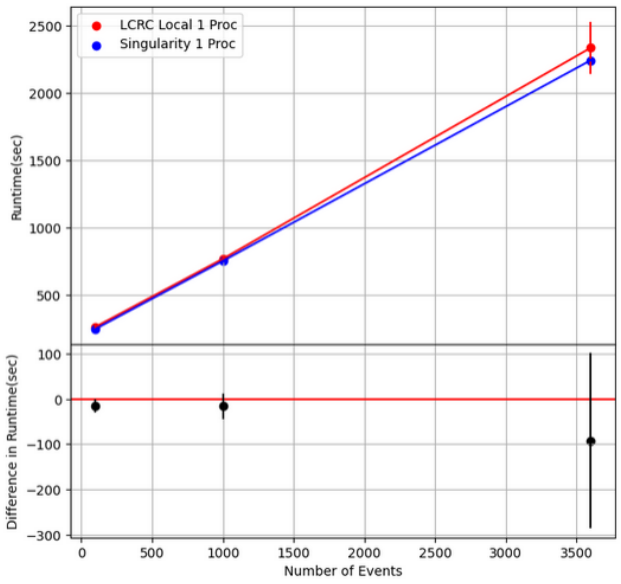

FIG. 6. Graph of runtime for 1 process with Darshan.

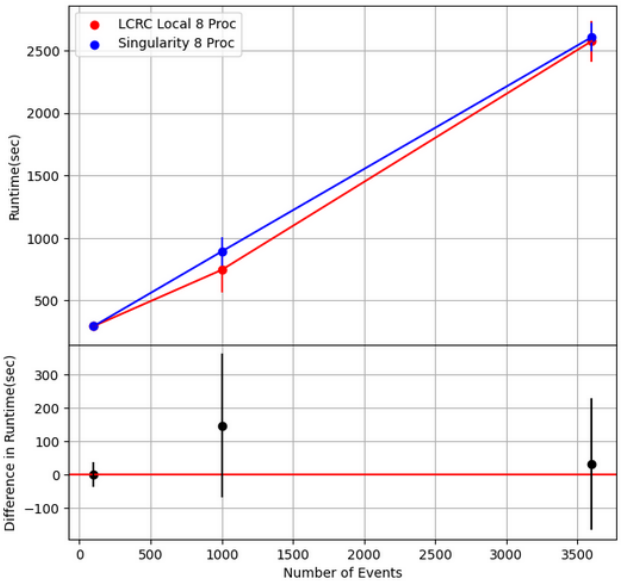

FIG. 7. Graph of runtime for 8 processes without Darshan.

Darshan.

The second set of results showed a similar trend in that all points in this data set were around zero within the first standard deviation. This showed that independent of a container, Darshan doesn't add a significant overhead to Athena. The reason behind testing being limited to 8 processes has to do with parallel compression. Because parallel compression wasn't enabled for these runs, 16 and 32's later processes were delayed by quite a bit exacerbating their runtimes. I can view this visually in the heat map in 11. Because of this, the usable data was restricted to the 1 and 8 processes.

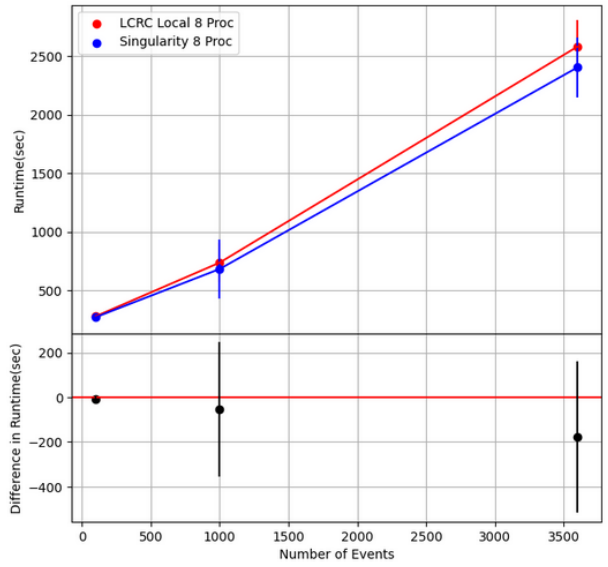

FIG. 8. Graph of runtime for 8 processes with Darshan.

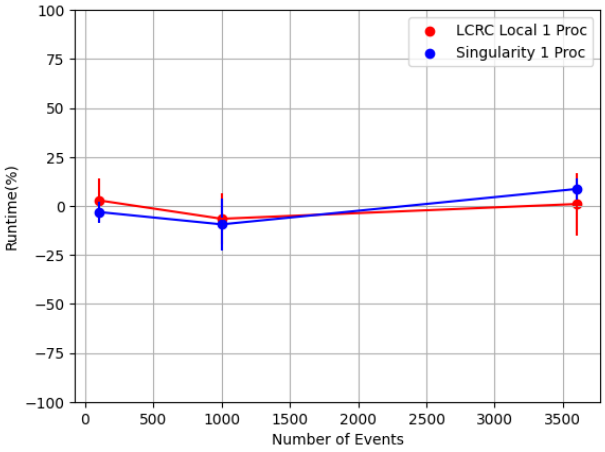

FIG. 9. Graph of the difference in runtime(%) for 1 process with and without Darshan.

IV. CONCLUSION

Over the course of the program, my project required the measurement of 4 different conditions: Athena with and without Darshan, inside and outside of Singularity. What was observed was that running inside Singularity had no significant impact on runtime, even at higher events and processes. Furthermore, running with Darshan doesn't add any significant overhead regardless of the container being used. I also justified the use case of Athena's log files to extract the runtime as there was a less than 30-second difference between the running time recorded in the Athena and Darshan logs. This difference has been accounted for by both the time interval that Athena logs retrieve wall-runtime and the stage at which Athena starts recording its runtime. The overall results from the experiment are favorable and make an excellent case for the use of Darshan.

The next steps to continue this project would be to collect data at a larger scale at higher event counts while using parallel compression in order to avoid the issues that happened during my testing at 16 and 32 processes. More testing would help solidify the findings and reduce the significance of the error in the data enabling a more accurate conclusion.

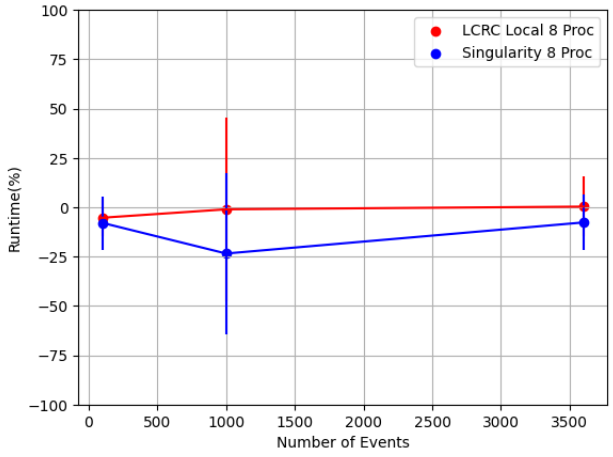

FIG. 10. Graph of the difference in runtime(%) for 8 process with and without Darshan.

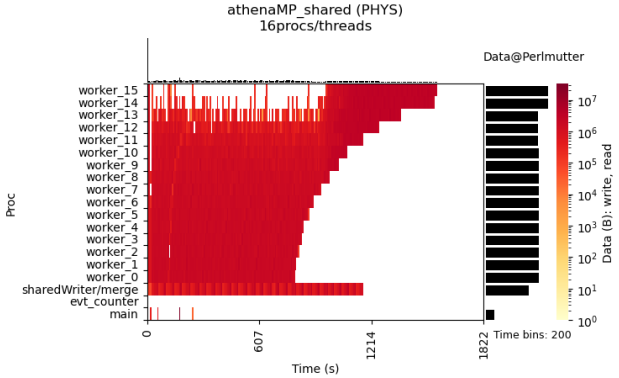

FIG. 11. The read and write operations from each worker to the shared writer during the 16 process.

V. ACKNOWLEDGEMENTS

This work was supported by the U.S. Department of Energy, Office of Science, Office of High Energy Physics, High Energy Physics Center for Computational Excellence (HEP-CCE).

This work is in part supported by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-06CH11357; in part supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative; and in part supported by the U.S. Department of Energy, Office of Science, Of-

fice of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program.

This research used resources at Argonne Laboratory Computing Resource Center (LCRC).

[1] ATLAS Collaboration, The ATLAS experiment at the CERN large hadron collider, JINST **3**, S08003.

[2] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, Understanding and improving computational science storage access through continuous characterization, in *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)* (2011) pp. 1–14.

[3] *The ATLAS Collaboration Software and Firmware*, Tech. Rep. ATL-SOFT-PUB-2021-001 (CERN, Geneva, 2021).

[4] V. S. Gregory M. Kurtzer and M. W. Bauer, Singularity: Scientific containers for mobility of compute, PLOS ONE (2017).