Gauge: An Interactive Data-Driven Visualization Tool for HPC Application I/O Performance Analysis

Eliakin del Rosario*, Mikaela Currier*, Mihailo Isakov*, Sandeep Madireddy†,
Prasanna Balaprakash†, Philip Carns†, Robert B. Ross†, Kevin Harms†, Shane Snyder†, Michel A. Kinsy*
* *Adaptive and Secure Computing Systems (ASCS) Laboratory*
Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843
{mihailo,eliakin.drosario,mkinsy}@tamu.edu
† *Argonne National Laboratory*, Lemont, IL 60439
smadireddy@anl.gov, {pbalapra,carns,rross,ssnyder}@mcs.anl.gov, harms@alcf.anl.gov

***Abstract*—Understanding and alleviating I/O bottlenecks in HPC system workloads is difficult due to the complex, multi-layered nature of HPC I/O subsystems. Even with full visibility into the jobs executed on the system, the lack of tooling makes debugging I/O problems difficult. In this work, we introduce *Gauge*, an interactive, data-driven, web-based visualization tool for HPC I/O performance analysis.**

Gauge aids in the process of visualizing and analyzing, in an interactive fashion, large sets of HPC application execution logs. It performs a number of functions met to significantly reduce the cognitive load of navigating these sets - some worth many years of HPC logs. For instance, as its first step in many processing chains, it arranges unordered sets of collected HPC logs into a hierarchy of clusters for later analysis. This clustering step allows application developers to quickly navigate logs, find how their jobs compare to those of their peers in terms of I/O utilization, as well as how to improve their future runs. Similarly, facility operators can use Gauge to 'get a pulse' on the workloads running on their HPC systems, find clusters of under performing applications, and diagnose the reason for poor I/O throughput. In this work, we describe how Gauge arrives at the HPC jobs clustering, how it presents data about the jobs, and how it can be used to further narrow down and understand behavior of sets of jobs. We also provide a case study on using Gauge from the perspective of a facility operator.

***Index Terms*—High-Performance Computing, I/O Analysis, Visualization, Clustering, Machine Learning.**

I. Introduction

High-performance computing (HPC) HPC systems are built to accelerate scientific or business workloads. Not all types of programs are easy to accelerate, however. In this work, we focus on I/O-bounded programs that struggle to make good use of the available I/O bandwidth. Since a modern HPC I/O subsystem is multilayered and orders of magnitude more complex than that of single-node machines, debugging I/O problems is difficult. Several tools such as Darshan [1] and Ellexus Mistral [2] were created to provide visibility into this problem. While these tools can help record I/O utilization issues, HPC system users still need tools that can help detect and diagnose when jobs are underperforming because of poor I/O usage.

Darshan is a lightweight HPC I/O characterization tool that instruments HPC jobs and collects their I/O access patterns. The logs Darshan collects are the main window an expert has into the workloads running on HPC systems. In our study, we analyzed 89,844 Darshan logs collected from the Argonne Leadership Computing Facility (ALCF) Theta supercomputer, in the period from May 2017 to March 2020 [3].

While Darshan offers a number of utilities for visualizing logs and exporting a record of an HPC job, these tools work on a per-log basis. To work with bulk logs, users have to manually aggregate Darshan outputs, write scripts, or, in the best-case scenario, rely on the facility to provide, for example, a year of logs in a CSV format. Even for simple tasks such as counting the number of times a certain application has been run or the total I/O volume transferred by a set of jobs, users have to create ad hoc scripts. Even with support for easy data manipulation, diagnosing I/O problems while working on large tabular datasets is not trivial. Although experts can provide insight for a specific job, this approach is not scalable when attempting to apply analysis on a large set of similar jobs or when simply exploring a dataset searching for possible issues. In this work we present Gauge: a tool that can allow I/O experts and facility operators to better scale their efforts and, instead of analyzing single logs, apply their insight on clusters of very similar HPC jobs.

Gauge is a web-based, data-driven, highly interactive exploration and visualization tool meant for diagnosing HPC I/O behaviors and problems. The goals of Gauge are as follows:

- Facilitating easy exploration and navigation in the high-dimensional space of Darshan logs
- Clustering similar jobs in order to reuse expert analysis and scale expert effort better
- Providing actionable reports for discovered I/O issues

Gauge has two target audiences: facility operators may use Gauge to deal with system-wide problems and work to increase the overall performance of the HPC cluster, and scientists and application developers may use Gauge to improve their jobs and diagnose the sources of low I/O throughput. In the following sections we explain how Gauge clusters and presents HPC jobs in a navigable hierarchy; we discuss how users can use Gauge to understand the I/O behaviors of clusters of jobs; and we present a case study on using Gauge from the perspective of a facility operator. We invite the reader

978-0-7381-2361-5/20/$31.00 ©2020 IEEE
DOI 10.1109/PDSW51947.2020.00008

to evaluate Gauge at ascslab.org/research/gauge, where we expose an instance of Gauge with which the user can explore anonymized Theta logs. Additionally, Gauge is open-source and available in the reproducibility appendix [4].

II. Clustering Methodology

The goal of this section is to show that (1) analyzing clusters instead of individual jobs allows us to better utilize expert insight without sacrificing analysis accuracy, (2) there exists a "natural," intuitive way to cluster HPC jobs, and (3) there exist methods that can objectively show that one clustering method is superior to another on our problem domain.

A. Preliminaries

In this work we cluster and visualize 89,844 Darshan logs that have an I/O volume larger than 100 MiB. These logs were collected at the Argonne Leadership Computing Facility (ALCF) Theta supercomputer in the period between May 2017 and March 2020. Darshan [5] is an HPC I/O characterization tool that collects I/O access pattern of jobs running on a system. While it supports multiple different APIs such as POSIX, MPI-IO, and STDIO, in this work we focus on POSIX. Darshan instruments a job and collects hundreds of aggregate values such as runtime, number of processes, read/write accesses, bytes read or written to shared or unique files, I/O access patterns per each file, and timestamps of first file open and close operations. Our preprocessing pipeline removes a significant number of unimportant or redundant features, It summarizes each job using 53 features: 14 absolute-values features such as runtime, I/O throughput, total data volume, total number of files, bytes, and accesses, as well as 39 relative (percentage) features such as read-to-write ratio, percentage of accesses of certain sizes, and consecutive accesses. The feature engineering used in this work is explained in more detail in [3].

B. Clustering HPC jobs

All of the HPC system logs used in this work can be treated as unlabeled data. Until experts provide labels or categorize each log, we cannot attempt to map new logs to existing categories or mark a log as, for example, "high performance" or "inefficient." Cluster analysis methods attempt to separate a set of data points into a number of groups so that data points within a group share some underlying property or behavior. Clustering data points such as individual HPC jobs into groups of similar samples works under the assumption that there exists some underlying structure to be unraveled. To evaluate this assumption, we apply principal component analysis (PCA) to our HPC log dataset. Here, the 53-dimensional space of Darshan features is (linearly) compressed down into just two dimensions, with these two dimensions explaining 61.4% of the data variance. In Figure 1 we show a 2-dimensional histogram of the PCA space with two components, with job density shown on a logarithmic axis. Although the whole dataset is globular in two dimensions and does not have any visible structure to it (left), zooming into a dense region reveals the existence of many small clusters of jobs (dark

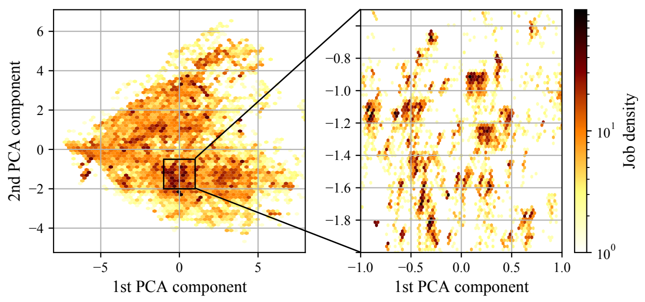

Fig. 1. Two-dimensional PCA projection of the collected Darshan logs.

red spots in the zoomed-in figure on the right). Evidently, even in this highly compressed space, significant structure exists. We expect that adding more dimensions will reveal further separation within the dataset. However, plotting high-dimensional spaces is difficult, so we will rely on clustering to reveal structure.

C. Choice of clustering method

Choosing an appropriate clustering algorithm is challenging because we do not have a clear method to evaluate which clustering (a "clustering" being a single mapping of samples to clusters) is better, other than relying on expert intuition. However, we can apply the following criteria: (1) our clustering method should be relatively insensitive to its parameters — i.e., not require expert tuning, and (2) it should be insensitive to noise and random initialization of seeds. Therefore, we first seek a method to compare two clusterings in terms of their similarity.

Several information-theoretic methods [6], [7] have been proposed to compare two clusterings. In this work we use variation of information (VI) [6] since it solves many of the issues previous clustering metrics had. If a clustering carries some information about the sample-cluster mapping, VI measures how much information is gained and how much is lost by switching from one clustering to another. Low VI values imply that two clusterings are very similar, since the information contained in the two clusterings has a large overlap. Large VI values imply that the clusterings carry very different information (e.g., by both being random sample-cluster mappings) or that one clustering contains significantly more information than another. We use VI to evaluate how consistent different methods are in terms of their parameters / seeds and whether different methods arrive at similar clusterings. We evaluated four clustering algorithms: k-Means, mean-shift clustering (MSC) [8], expectation maximization using Gaussian mixture models (EMGMM), and DBSCAN [9]. Our experiments show that only DBSCAN consistently arrives at similar clusterings for both different parameter configurations and different random initializations (graphs omitted for space reasons). DBSCAN is an agglomerative, density-based, nonparametric clustering algorithm that works by iteratively clustering together samples that are within an ϵ distance of each other. This imposes no structure on the shape or the number of clusters, making DBSCAN an excellent tool for clustering data from new applications.

D. Hierarchical clustering of HPC data

The size and number of DBSCAN clusters depend on the choice of the epsilon (ϵ) parameter. Large ϵ values cause all jobs to be clustered in a single cluster, while small value cause all jobs to belong to individual clusters. Although we can experiment with this value on our dataset, it presents two problems: (1) when applying clustering to a new dataset (e.g., from a new supercomputer), we may need to tune this value again, and (2) a single ϵ value does not reveal hierarchical structure (e.g., which clusters will get combined together if we were to increase ϵ).

To fix these problems, we opt to use HDBSCAN, a hierarchical version of DBSCAN that, instead of running DBSCAN using a single ϵ value, can be seen as running a grid search over all values and retaining information about which clusters merge into which and at what ϵ values. HDBSCAN is useful because instead of just providing a mapping between samples and clusters, it creates a tree where branches are clusters and leaves are individual jobs. For a deeper discussion of HDBSCAN and HPC job clustering, see [3]. In Figure 2 we provide an example hierarchical clustering of HPC jobs built by using HDBSCAN.

We can interpret the tree in the figure as follows: each circle in the graph represents a cluster of jobs, it's radius is proportional to the number of jobs in the cluster, and it's vertical position specifies the ϵ value at which HDBSCAN creates / breaks up the cluster. The lower the nodes are in the graph, the less jobs they have and the more dense they are. As the ϵ parameter value used by DBSCAN is lowered, each cluster eventually splits into two smaller clusters. This is visualized with the parent and child clusters connected by the branches. The top node of the tree represents a cluster that contains all 89,844 jobs in our dataset. This top node splits into two branches, meaning there exist broadly two classes of jobs in the dataset. By using Gauge, we discover that the main difference between the two branches are their I/O patterns: the left branch contains jobs that have mostly large write access sizes (in the 100 KiB+ range) and a balanced amount of reads and writes, whereas the jobs in the right branch use smaller write sizes and are write-heavy. Similar analyses can be applied to other nodes in the tree. Another way of interpreting the tree is by looking at the tree topology. We observe that the tree consists of multiple branches that have many small clusters "falling off" of them. Periodically, these branches also split into two branches of similar sizes. Tall branches represent clusters that are stable for a large range of ϵ values. These clusters are dense, so they are insensitive to changes in ϵ in that range. As we reduce ϵ, small outlier clusters are separated from the more populous, 'main' branches, but the main clusters are largely unaffected and maintain their size. However, if these clusters consist of smaller, denser clusters, at a certain ϵ value these main branches will bifurcate.

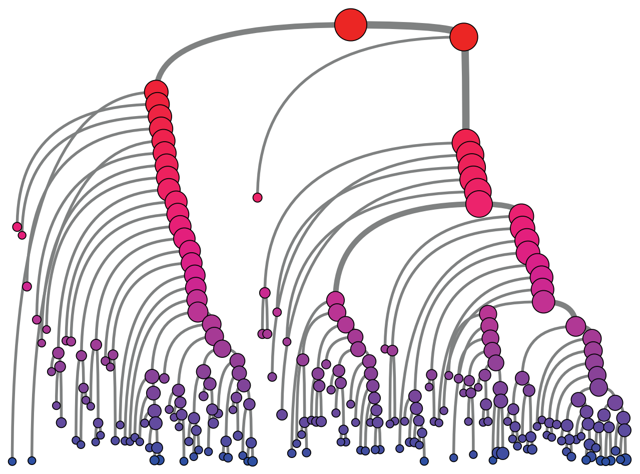

Fig. 2. Gauge clustering hierarchy built by using HDBSCAN. Each node is a cluster of jobs. Each node connects a parent (higher) and child (lower) cluster, where the parent contains all of the child's jobs, plus some. Each cluster's height represents the ϵ value at which that cluster is split into smaller clusters.

III. HPC I/O Performance Analysis Using Gauge

The goal of Gauge is to provide an interactive environment where application developers and facility operators can (1) quickly determine jobs of interest, (2) evaluate whether these jobs are behaving as expected, (3) compare and contrast these jobs with previous runs of the same application or similar jobs, and (4) extract sensible information to better understand why jobs are performing as they do, improve their future jobs, and develop greater insight into the workloads running on the system.

In a nutshell, Gauge is a web application that consumes unorganized logs of HPC jobs and provides a hierarchical, interactive visualization of the workloads that ran on the system. Gauge aims to offer several levels of granularity at which to view jobs from high-level clusters arranged in a tree structure to condensed sets of graphs for each user-selected cluster in the tree down to the fine-grained views of each job and each logged feature in a cluster. In the next three subsections, we describe each of these views.

A. Gauge Clustering Hierarchy

In Figure 2, we show Gauge's hierarchical clustering. Through interaction with the users, we built a number of features that ease navigation in the tree. Hovering over a node displays the cluster ϵ parameter and number of jobs, and for further analysis the user can click the nodes to reveal key details about the cluster. The nodes in the hierarchy can be colored by the size of each cluster, the ϵ value of each cluster, or by whether a cluster contains jobs from a specified user or app. One of the more useful features is coloring by feature value. Here, by averaging a user-specified feature's values over all of the jobs in the cluster, we determine the cluster's color. For example, by selecting to color clusters by their I/O throughput, we can visually find high- and low-performing jobs in terms of their I/O utilization.

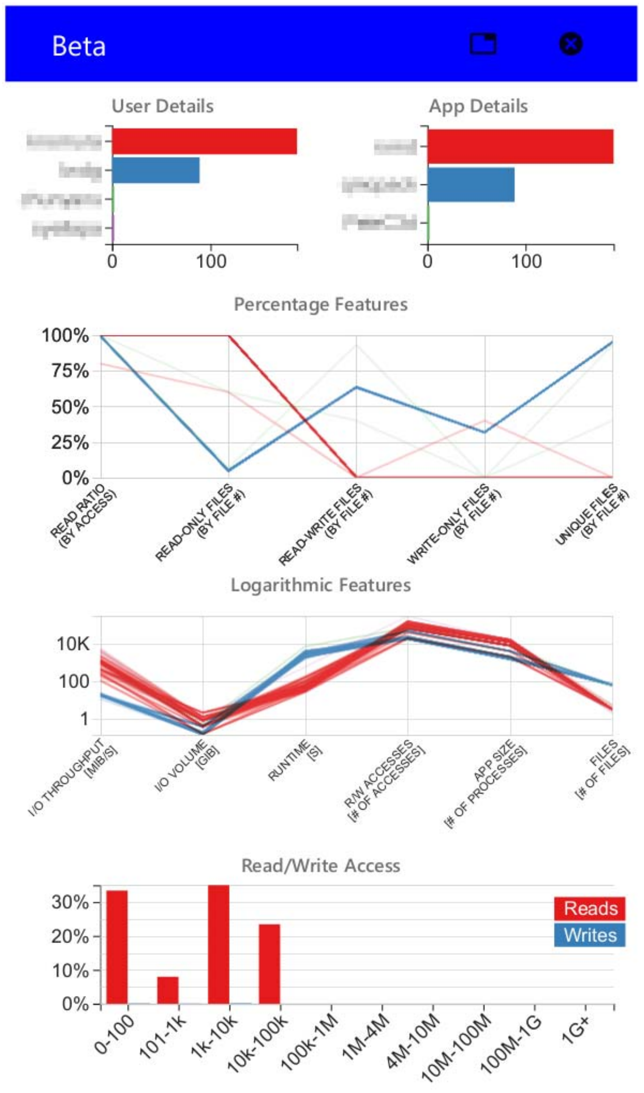

Fig. 3. Gauge cluster visualization. Every time a cluster in the hierarchy is selected, this type of column is created with the cluster's data.

B. *Gauge Per-Cluster Visualization*

Selecting a cluster creates a column with details pertaining to that cluster, for example, which users ran the jobs in the cluster; which applications do these jobs belong to; what are the runtimes, I/O volumes, or other features of these jobs; and what are the common I/O access sizes that these jobs made. In Figure 3 we show an example column that is displayed when a user clicks on a cluster from Figure 2.

The column consists of five graphs. The first two graphs (first row) are bar charts that show the distribution of users that ran the jobs, and the applications that the jobs belong to. If the number of users or applications exceeds 5, the first 4 are shown, and the remainder are grouped in the final "other" column. The coloring of these charts proves important in graphs to follow.

The next two graphs (second and third row) show parallel coordinate plots of different sets of features. Each broken line represents one job, and its position on each of the axes (5 axes on the second row, 6 on third row) specifies that job's feature values for the axes' feature. The two graphs differ in the type of features they present. The graph in the second row shows features with percentage values, for example, the percentage of read accesses or write-only files. These values are always bounded in the $[0\%, 100\%]$ range, so the axes have fixed ranges. The other graph (third row) presents absolute-valued features such as a job's I/O throughput, I/O volume, runtime, or the number of files used by the job. Note that despite these axes having different dimensions (e.g., MiB/s, GiB, s), for clarity and readability we use a single unitless axis shown on the left of the graph. Since each cluster will have a different range for its jobs' values, each column needs to have a separate range plotted. Having multiple separate ranges complicates comparison between the selected clusters, so Gauge offers an option to use a unified range across all the selected columns. This range is calculated by using the smallest and largest values of any job in the selected cluster. Another feature Gauge offers is the choice of how the lines are colored. Right now, Gauge offers coloring jobs by their user (where jobs from different users have different colored lines), and similarly coloring by application.

The fifth graph (fourth row) shows the distribution of access sizes, broken up by reads and writes and by common access sizes. When collecting logs, instead of storing access sizes of each individual R/W access, Darshan collects aggregate metrics and reports the number of accesses for each "bin." We use the same bins to present the read and write accesses. Note that this bar plot presents averages across the cluster and may be unreliable for highly diverse clusters.

C. *Gauge Cluster Parallel Coordinate Plots*

Although these graphs offer information about 31 different features of the cluster's jobs, a user may want to analyze a specific combination of features or may want to observe only a subset of the jobs in the cluster. To allow such analyses, Gauge also includes a full-page, highly customizable parallel coordinates plot based on HiPlot [10] that can be called for each cluster individually. With the ability to select any combination of the 53 recorded features, the HiPlot package allows users to visualize interactions between features that may otherwise go unnoticed in the cluster columns described above. By using HiPlot, Gauge lets the user quickly add or remove selected jobs, color jobs based on any of the selected features, or even change the type of axis used for each feature (the user can choose between using a linear, logarithmic, percentage, or categorical scale). The user's selections are stored so that any following HiPlot selection modals will automatically apply those decisions. In Figure 4, we show an example HiPlot parallel coordinates plot for a cluster.

D. *Gauge Software Architecture*

Gauge consists of a Python and Flask server and a React-based front end. The back end parses a directory of Darshan logs, runs these logs through a preprocessing pipeline that

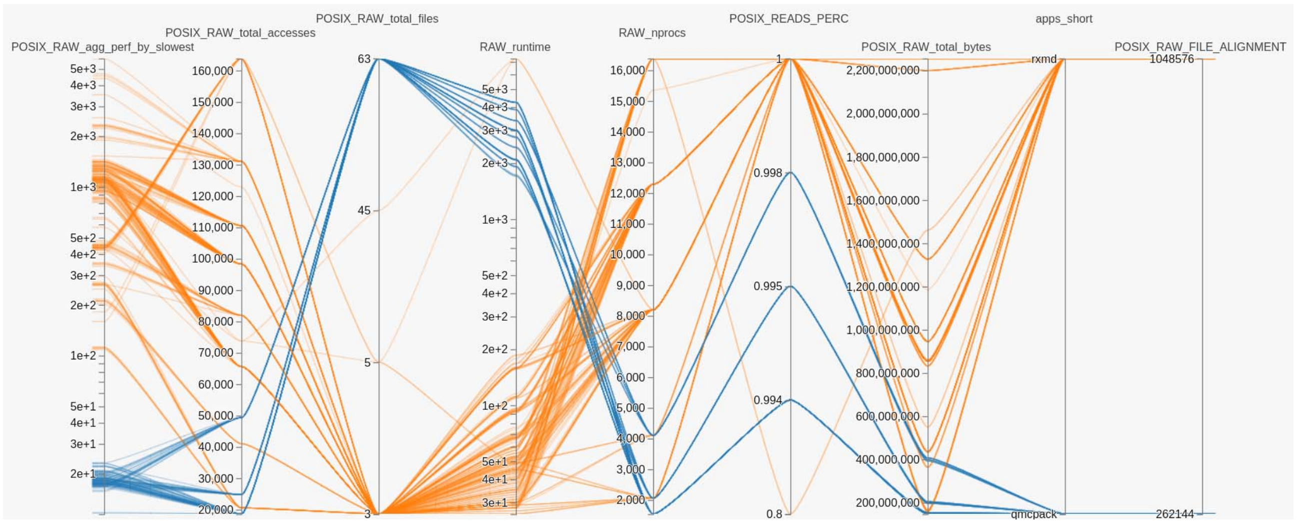

Fig. 4. HiPlot view of a cluster with two different applications: quantum chromodynamics (blue) and quantum materials (orange).

sanitizes the data and applies feature engineering, and clusters the data using HDBSCAN. The back end and front end are Dockerized to allow easy deployment on new machines and new datasets. Since our only assumption about the dataset is that it is stored in a directory and consists of Darshan log files, one can easily run Gauge on other supercomputers or clusters that are using Darshan for instrumenting HPC jobs.

The Gauge front end is built using React [11] and MaterialUI and uses D3 [12] and HiPlot [10] for the graphs. React is a front-end JavaScript library that, with the help of React contexts, provides seamless data flow throughout the application. The styling for the application was based primarily around MaterialUI because of its professional components and ease of use. We use the D3 graph plotting library because of its maturity and vast feature set. We use HiPlot for full-page parallel coordinate plots because of HiPlot's unprecedented interactivity and excellent user experience.

IV. Case Study

As a case study, we present how a facility operator can use Gauge to comb through the logs and highlight clusters of interest. For this example, an ALCF I/O expert performed an open-ended exploration of logs in search of jobs that do not obey conventional wisdom in terms of I/O performance.

The I/O expert identified the cluster shown in Figure 3 as exhibiting strange behavior. Here, the majority of jobs in the cluster belong to a quantum chromodynamics and a quantum materials application. Both applications perform primarily read accesses and have relatively similar I/O volumes and number of processes. Despite being similar, however, the runs of the two applications differ by several orders in magnitude for both runtime and I/O throughput. These differences are uncommon: other clusters that Gauge identified (at the same DBSCAN ϵ value) have less variance for those two features.

The I/O expert used the HiPlot feature of Gauge to gain further insight into the cluster, shown in Figure 4. The expert colored jobs by application and moved the I/O throughput axis (POSIX_RAW_agg_perf_by_slowest) all the way to the left. Several conclusions can be made from the figure: (1) applications transfer similar amounts of data (200 MiB – 2 GiB range), (2) the majority of orange jobs have 5 times larger I/O throughput compared with blue jobs, (3) both applications almost exclusively perform read accesses, and (4) the blue application opens a larger number of files. We note, however, that this last conclustion may not be important since jobs from both applications use thousands of processes.

On a suspicion, the I/O expert then added another column to HiPlot (the rightmost axis POSIX_RAW_FILE_ALIGNMENT). Here we can see that jobs from different applications have different file alignments (256 KiB vs. 1 MiB). Different file alignments hint at the possibility that these applications might be using files on different filesystems. Indeed, after further inspection using Darshan summary plots, it became clear that the slower application reads primarily files on the home filesystem, while the faster application uses files on Lustre. Even though the slower application obviously will benefit from moving its files to Lustre, a scientist who develops an application on a local machine can easily make such a mistake. Despite being briefed on using the HPC system, it may be difficult for the scientist to debug this issue without the manual involvement of an I/O expert. Similarly, a facility may have a hard time spotting such inefficacies in their workloads and may underutilize the HPC system's potential. Gauge offers a more efficient method for helping developers accelerate their workloads and for helping facility operators extract the maximum out of the system. Furthermore, Gauge provides a rapport that can be used by the facility operators to support their insights, speeding up communication between developers and administrators.

V. Related Work

Several works have tackled automating performance analysis of HPC jobs. Paradyn [13] is a tool that measures the performance of HPC applications by dynamically instrumenting the application. Similarly, Periscope [14] also searches for performance problems. Unlike Paradyn, Periscope uses two separate monitoring approaches. One of these monitoring approaches is the Peridot monitor which focuses on OpenMP and MPI performance data and the cache monitor that focuses on the memory hierarchy. In addition, both of these tools offer a lightweight visualization interface to quickly display tabular metrics and graphs. Another HPC performance analysis tool is VAMPIR [15]. VAMPIR takes a given application trace and transforms it into a variety of graphical views such as state diagrams, activity charts, statistics and other useful displays.

The limitation of these tools lays in that the performance analysis is done on single application. In the case of Paradyn and Periscope, the hypotheses used to search for performance problems must be precisely defined because these tools will only instrument the parts of the application that are relevant to the defined performance problem.

Gauge, on the other hand, behaves more like a log analyzer similar to Splunk [16] and the ELK stack [17]. Gauge serves as an extension to these performance analysis tools where it consumes the logs collected by characterization tools like Darshan and transforms them into a hierarchical structure of clusters in order to simplify the exploration of jobs and diagnosing of I/O bottlenecks. Unlike Splunk and the ELK stack, Gauge does not require the user to learn a complex query language or log handling configuration. Instead, Gauge simplifies how a user interacts with logs by purely interacting with the provided graphs. A major difference between Gauge, Splunk and the ELK stack is that Gauge provide users with the ability to explore high-dimensional data graphically, a feature that is not found in Splunk and to achieve this feature with the ELK stack, an additional integration is required like using Vega, a grammar based charting library to complement Kibana.

VI. Conclusion

In this work we tackle the challenge associated with diagnosing I/O bottlenecks in HPC jobs. We introduce *Gauge*, an interactive web-based tool for exploring logged HPC jobs, clustering these jobs into an easy-to-navigate hierarchy, and displaying information about clusters of similar jobs. We present a stable hierarchical method for clustering HPC jobs, the inner workings and decisions behind the design of Gauge. We illustrate how Gauge can be used by both facility operators and application developers to find I/O throughput issues. We provide access to an instance of Gauge[1] that was run on ALCF Theta supercomputer logs.

In future work, we aim to (1) support not just Darshan's POSIX module but also MPI-IO and STDIO, (2) integrate I/O throughput prediction models that may help detect outlier jobs, and (3) work with I/O experts to improve Gauge and apply it to other HPC systems.

[1]Available at ascslab.org/research/gauge

Acknowledgments

This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

References

[1] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, "Understanding and improving computational science storage access through continuous characterization," *ACM Trans. Storage*, vol. 7, no. 3, Oct. 2011.

[2] J. M. Kunkel, E. Betke, M. Bryson, P. Carns, R. Francis, W. Frings, R. Laifer, and S. Mendez, "Tools for analyzing parallel i/o," in *International Conference on High Performance Computing*. Springer, 2018, pp. 49–70.

[3] M. Isakov, E. del Rosario, S. Madireddy, P. Balaprakash, P. Carns, R. Ross, and M. Kinsy, "HPC I/O throughput bottleneck analysis with explainable local models," in *SC'20: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020.

[4] E. del Rosario, M. Currier, and M. Isakov, "Gauge: An Interactive Data-Driven Visualization Tool for HPC Application I/O Performance Analysis," Sep. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4027969

[5] S. Snyder, P. Carns, K. Harms, R. Ross, G. K. Lockwood, and N. J. Wright, "Modular HPC I/O Characterization with Darshan," in *2016 5th Workshop on Extreme-Scale Programming Tools (ESPT)*, 2016, pp. 9–17.

[6] M. Meilă, "Comparing clusterings by the variation of information," in *Learning Theory and Kernel Machines*, B. Schölkopf and M. K. Warmuth, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 173–187.

[7] E. B. Fowlkes and C. L. Mallows, "A method for comparing two hierarchical clusterings," *Journal of the American Statistical Association*, vol. 78, no. 383, pp. 553–569, 1983.

[8] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 603–619, 2002.

[9] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, p. 226–231.

[10] D. Haziza, J. Rapin, and G. Synnaeve, "Hiplot, interactive high-dimensionality plots," https://github.com/facebookresearch/hiplot, 2020.

[11] "React: Javascript library for building user interfaces," https://reactjs.org/, 2020, accessed: 2020-09-06.

[12] M. Bostock, V. Ogievetsky, and J. Heer, "D3: Data-driven documents," *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

[13] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall, "The paradyn parallel performance measurement tool," *Computer*, vol. 28, no. 11, pp. 37–46, 1995.

[14] M. Gerndt, K. Fürlinger, and E. Kereku, "Periscope: Advanced techniques for performance analysis." in *PARCO*, 2005, pp. 15–26.

[15] W. E. Nagel, A. Arnold, M. Weber, H.-C. Hoppe, and K. Solchenbach, "Vampir: Visualization and analysis of mpi resources," 1996.

[16] J. Stearley, S. Corwell, and K. Lord, "Bridging the gaps: Joining information sources with splunk." in *SLAML*, 2010.

[17] G. Smith, "Log analysis with the elk stack (elasticsearch, logstash and kibana)," 2015.

VII. Reproducibility Appendix

We open-source Gauge and provide access to an anonymized dataset of HPC jobs ran on the Argonne Leadership Computing Facility (ALCF) Theta supercomputer. The code and the data are available at [4]. Both the front and back end are containerized using Docker, and allow easy deployment on new systems and on new datasets. Additionally, we provide the code to reproduce Figure 1. For support or feature requests, please contact the authors.