SAND2020-8523C

HPC System Data Pipeline to Enable Meaningful Insights through Analytic-Driven Visualizations

Benjamin Schwaller[1], Nick Tucker[2], Tom Tucker[2], Benjamin Allan[1], Jim Brandt[1]
Sandia National Laboratories[1] — Open Grid Computing[2]
Albuquerque, New Mexico[1] — Austin, Texas[2]
bschwal|baallan|brandt@sandia.gov[1] — nick|tom@ogc.us[2]

***Abstract*—The increasing complexity of High Performance Computing (HPC) systems has created a growing need for facilitating insight into system performance and utilization for administrators and users. The strides made in HPC system monitoring data collection have produced terabyte/day sized time-series data sets rich with critical information, but it is onerous to extract and construe meaningful information from these metrics.**

We have designed and developed an architecture that enables flexible, as-needed, run-time analysis and presentation capabilities for HPC monitoring data. Our architecture enables quick and efficient data filtration and analysis. Complex run-time or historical analyses can be expressed as Python-based computations. Results of analyses and a variety of HPC oriented summaries are displayed in a Grafana front-end interface. To demonstrate our architecture, we have deployed it in production for a 1500-node HPC system and have developed analyses and visualizations requested by system administrators, and later employed by users, to track key metrics about the cluster at a job, user, and system level.

Our architecture is generic, applicable to any *-nix based system, and it is extensible to supporting multi-cluster HPC centers. We structure it with easily replaced modules that allow unique customization across clusters and centers. In this paper, we describe the data collection and storage infrastructure, the application created to query and analyze data from a custom database, and the visual displays created to provide clear insights into HPC system behavior.

***Index Terms*—HPC monitoring, Grafana, visualization, operational data analytics**

I. Introduction

As the High Performance Computing (HPC) industry moves towards exascale, monitoring HPC systems is becoming increasingly important and simultaneously more complex. Even today, HPC systems have millions of compute cores, petabytes of memory, and sophisticated network interconnects. System administrators must maintain the performance of these complex systems to enable important simulations. However, complications such as hardware failures, network congestion, filesystem contention, and orphaned processes create application performance variation up to 100% [1]. When these issues arise on systems of this size, detection and alleviation of problems is challenging.

To better understand system issues, significant strides have been made in the HPC monitoring community to gather high-fidelity system and application metrics [2]–[5]. These tools can return several terabytes of data per day that is useful for solving system issues and understanding application performance variation. Extracting meaningful insights from these enormous datasets requires both domain expertise and intelligent data management and analysis techniques.

To meet these challenges in a production setting, we developed a data pipeline architecture for the collection and storage of data from HPC systems and for the run-time presentation of visualizations of derived statistics. The architecture supports queries with or without additional analyses which come in the form of Python modules. These analyses can be accessed through a common interface and can be added or changed *dynamically*. Executing analyses in-line with queries reduces memory and storage demands and enables more complex insights for terabyte-sized datasets typical of HPC installations. We demonstrate the utility of our design and implementation on a production 1500-node HPC system. The analyses and visualizations presented are based on requirements of the system administrators and have also proven useful to users.

The main contributions of this paper are:

- A design that enables flexible, as-needed, run-time analysis and presentation capabilities for HPC data.
- Analyses tailored to users and administrators.
- A generalized architecture that is demonstrated on a large production HPC system.

Section II describes the architecture of the pipeline, including specific design features to enable the support of flexible, run-time analyses. Section III describes deployment details on a production HPC system. Section IV describes the dashboards and the underlying analyses to provide derived metrics. Section V describes related work. Section VI presents conclusions and future work.

II. Architecture

In this section, we present the architecture for the data pipeline. We emphasize design aspects that were specifically created for this pipeline and describe others that were complimentary pieces to complete the design. The architecture of the data storage and analysis components and connectivity to display host(s) are shown in Fig. 1.

A. Data Collection

In the Sandia National Laboratories (SNL) production system environment, data is collected using the Lightweight Distributed Metric Service (LDMS) [3]. LDMS is a data collection and transport system that provides low overhead monitoring of

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. SAND No. INPUT SAND NUMBER HERE
978-1-7281-6677-3/20/$31.00 ©2020 IEEE

HPC systems with high scalability. A key feature of LDMS is its ability to collect data at the same time (relative to each component's clock) for each component across the whole system. This enables coherent multi-component analyses across specific jobs or across the entire system.

B. Data Storage

Enabling run-time analyses over arbitrary data at the targeted data sizes places particular demands on the data management infrastructure. Our data storage architecture must be capable of ingesting data at the rates described in Section III-A while simultaneously supporting fast search of arbitrary data and data relationships from multiple query instances and returning that data in forms suitable for run-time analysis.

A variety of databases have been explored for supporting HPC monitoring data, with various tradeoffs in insertion rate, support for relational data, ability to handle timeseries data, ease of query, etc., with no one database excelling in all aspects. For our architecture we have chosen the Scalable Object Store (SOS) database [12]. SOS is a high-performance, indexed, object-oriented database created to efficiently manage structured data, including time-series.

SOS provides interfaces for C, C++ and Python. The SOS Python interfaces to NumPy and pandas provide high performance, zero copy access to large generally available analysis libraries. Much of the overhead in processing data of this magnitude is spent iterating over billions of records to filter the results. To speed up this iteration, SOS has a unique compound-key index that enables querying time series data across multiple attribute values with the same performance as a single attribute-value index. The compound-key indices used in this deployment are the six permutations of timestamp, node id, and job number (e.g. time_node_job, node_time_job, etc.). These compound-key indices support many of the queries of interest in HPC monitoring data analysis (see Section II-C).

C. Data Analysis and Visualization

The components involved in the data analysis and visualization through a Grafana front-end are shown in Fig. 1. Grafana was chosen because it can easily plot timeseries data, has a plethora of visualization options for presenting data, and supports custom database plugins. Components developed specifically as part of this work are colored green in Fig. 1 and described in this subsection. The remaining components are existing tools that this work utilized, however they are not considered a focus of this paper.

For this work, the flow of information begins with a request from a client through a custom datasource plugin in Grafana. The plugin is called SOS datasource (SosDS) and it constructs an HTTP request and facilitates communication with our web application. The request specifies an analysis Python module and the data relevant to the query and needed to perform the analysis. Our application, called sosdb-ui, is a Django web application suite that handles the Apache request from a client browser and then queries relevant data from the database [13]. The sosdb-ui suite of software utilizes Apache, NumSOS, and the SOS Python API to efficiently query, format, and return data to a Grafana front-end. The sosdb-grafana module adds to, and is dependent upon, sosdb-ui and was written

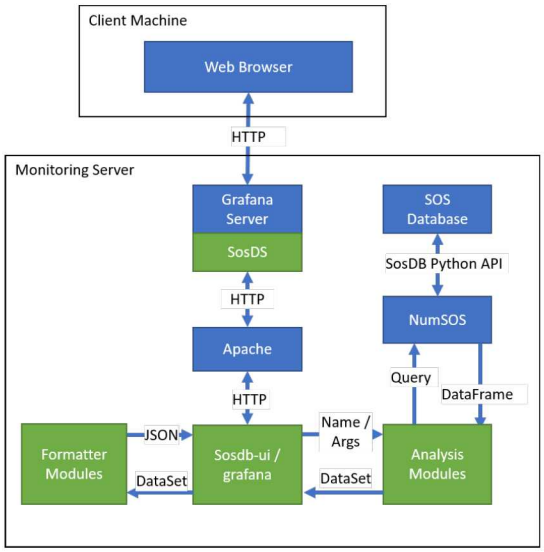

Fig. 1: Block diagram of web architecture to query, analyze, and render monitoring data. Green components were specifically made for the architecture described in this work.

strictly to handle requests from a Grafana server [14]. The sosdb-ui application handles Grafana requests and calls the specified Analysis Module to query and perform operations on the requested data.

To facilitate the transformation and storage of data sets from the SOS database, Open Grid Computing (OGC) developed the NumSOS application in conjunction with SOS [15]. NumSOS contains a variety of functionalities, such as finding the maximum value of a metric for each job, that make data manipulation simple for the user. The Analysis Modules use NumSOS to query the SOS database and perform numerical operations as needed, the result of which is returned as a pandas DataFrame to the module. After receiving a data object from NumSOS, the requested Analysis Module performs some defined computation and returns its output to sosdb-ui in the format of a SOS DataSet Python object which is a Python data object defined and used in NumSOS. Sosdb-ui then utilizes a Formatter Module to format the data according to the panel type specified in the original Grafana request. Grafana expects data to be formatted differently depending on if the data is to be displayed in a graph, table, etc. Finally, Grafana renders the data received from sosdb-ui into the requested visualization format for the user.

The sosdb-ui application was created so that system administrators and users with unique implementations on their clusters will be able to design custom-tuned as-needed Analysis Modules. To do so, we created a Python class architecture for NumSOS Python analyses that developers can use to custom tune the default modules provided, or to create their own Analysis Modules from scratch. The class architecture is relatively simple and contains only an `init` and `get_data` function. The init function passes the container file location, Grafana time-range, and schema into the class. The `get_data` function passes metrics, job number, username, and additional parameters that can be defined with the query and returns the necessary data object. The NumSOS queries and specific computation within the `get_data` function is left to the

developer. Analytic modules in this paper can be found at [14].

These modules exist as Python scripts in a directory. The framework also enables Python modules to be added, changed, or removed as needed without re-computation across the dataset. This plug-and-play nature expedites development and deployment of analysis dashboards as it does not require a root user to reinstall the entire application every time a new module is created.

Our data analysis and visualization architecture presents many advantages for analyzing HPC data. By doing as-needed analyses, complex analyses do not need to be run across the entirety of the database, which would be intractable for typical HPC datasets. Doing so would consume a significant amount of storage for saving analysis results and would increase load on the CPU and memory which could take resources away from the web server. One disadvantage to this strategy is that multiple identical queries must perform redundant analyses as opposed to doing an analysis once for all data. However, performing an analysis across all datapoints would likely prove impractical when generalizing this case study to a multi-terabyte dataset.

Additionally, the flexibility of the Analysis Modules allows us to create nearly any desired analysis. Machine learning models and statistical outlier calculations are just some of the features that could be added with this architecture. Of course, there are trade-offs to be made between intensity of analytic computation and the return time of the query.

III. Implementation

To demonstrate the utility of our design, this section describes its deployment for the monitoring and analysis of *Eclipse*, a production HPC system at SNL. Eclipse is a 1488 node cluster capable of 1.8 petaflops [19]. Data storage, analysis, and visualization back-end components are hosted on a monitoring host, *HPCMON*, a single-node server with 72 Intel Xeon Gold 6140 CPUs, 750 GB of memory, and 18 TB of NVMe raided storage. . The visualization is driven from and displayed on a per-user basis on each *client machine*.

A. LDMS on Eclipse

Eclipse is currently running LDMS v3 on all 1488 compute, 24 gateway, 12 login, and 8 administrative nodes to obtain a comprehensive view of the system. Data is aggregated from all nodes to administrative nodes, which also collect information about themselves, and then further aggregated to HPCMON.

We currently collect information related to SLURM job metadata, memory and virtual memory, CPU utilization, multiple Lustre filesystems, and the InfiniBand network which amounts to 406 discrete values, referred to as *metrics*, from each node. Data from all nodes is currently sampled once a minute. The resultant data size is 10 GB per day.

B. SOS and Eclipse Data

While the latest deployments of LDMS can store directly to SOS, SNL's production systems use LDMS's *store* feature to write live data out to CSV files every five minutes on HPCMON. This creates a five-minute lag to present data which we are addressing in future work, as seen in VI We keep the most recent two weeks worth of data from these CSV files in a SOS database. Two weeks of CSV data from Eclipse translates to roughly 150 GB which translates to 600 GB in a SOS database with six indices. In future work, we intend to increase the sampling rate and length of data stored which will bring the dataset into the terabyte range discussed earlier.

To demonstrate SOS's utility in this deployment, we compared the ingest and query performance of InfluxDB, an open-source timeseries database [8], and SOS. We created a 3.5GB CSV file with 6.6 million records by taking about a third of the metrics from a single day of Eclipse CSV. Using this as a data source, SOS maintained an average insertion rate of 8,316 records/second, compared to InfluxDB's rate of approximately 1,665 records/second. This makes the CSV insertion rates for SOS five times faster than InfluxDB, with neither database utilizing "batch mode". Batch mode was not used for this testing because it does not represent rates of continuously ingesting live data.

As an example of query performance, the databases were queried for identical metrics through a Python interface, with a filter for jobs with non-zero IDs. The metric data queried for in both cases was "Active" memory from a schema named meminfo. This query was performed 10 times on each database. On average, InfluxDB returned a pandas DataFrame object 41.15 seconds after submitting a query, while SOS returned the same DataFrame object, on average, in .008 seconds.

IV. Dashboards and Analyses Use Cases

System administrators face a variety of challenges and require an agile approach to diagnosing system issues in order to efficiently address problems as they arise. This section highlights the Grafana dashboards and underlying analyses that we created to present relevant metrics to both users and system administrators in order to meet their expressed needs.

The team of system administrators running SNL's production HPC systems was surveyed to discover their priorities with respect to application and system resource performance understanding. We summarize their top three priorities as follows:

1) Characterization of dedicated hardware utilization levels through fractional use metrics of job-specific compute node resources including RAM, CPU core time, memory bandwidth, and network bandwidths. Detailed retrospective analysis of Out-Of-Memory (OOM) related job failures is particularly sought after.
2) Identification of causes of performance degradation, tied to a specific user, specific job, and/or hardware fault, in shared resources such as networks (e.g., InfiniBand, 10GigE) and file systems (e.g., NFS, Lustre, and GPFS), particularly while the degradation is still in progress and subject to exacerbation by additional load.
3) Identification of users that could benefit from guidance to help them understand and avoid grossly inefficient behavior of their applications.

The primary focus of the dashboards being presented in this section are compute node memory occupancy and shared Lustre filesystem load (both bandwidth and meta-data operations). Subsequent discussions with scientific simulation application teams have revealed that these same dashboards provide value to them in understanding their applications and identifying unexpected or anomalous run-time behaviors. As an example

of the dual applicability, system administrators interested in seeing why a job ran out of memory and users interested in observing their application's temporal memory profile can both find value from a timeseries summary of memory metrics of a job. In order to facilitate users being able to browse their applications behavioral characteristics with respect to other resources, we have created additional dashboards to provide views of arbitrary raw time-series data for jobs currently or recently running on the system.

Our dashboards fall into two main categories: *general* and *breakdown*. *General dashboards* display aggregate information about the system and how it is being used by jobs or users. *Breakdown dashboards* display more detailed analyses about individual jobs and are typically navigated to by clicking on links within general dashboards. This distinction was made to allow system administrators to determine what jobs to drill-down into at a glance from general dashboards and then explore the data in breakdown dashboards. All dashboards enable selection of a HPC system-specific metrics database via a drop-down "container" menu. This is to support the many individual HPC systems in our production data center. In this work, the only database created and used was for the Eclipse system. Dashboard load times depend on the time-range specified but typically range from 5-15 seconds for an hour of data. The Lustre Summary Dashboard loads on the order of 30-45 seconds because of the large quantity of queries.

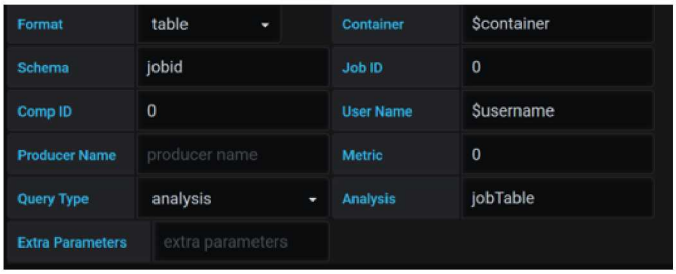

Fig. 2: The query interface in Grafana with the sosdb-grafana plugin.

Each panel in a dashboard is associated with a single query which may or may not have an analysis associated with it. Analyses are only run when an associated query is executed. This happens on dashboard loads, refreshes, and time-range changes. An example of the Grafana-based query interface provided by the SosDS-grafana plugin is shown in Figure 2. Container, schema, job number, component, user name, and producer name are all parameters passed back to NumSOS and the python analysis modules for performing the SOS database query and associated analyses. The Query Type field can be defined as either "metric" or "analysis" depending on whether the result returned for display is to be raw or derived respectively. The remainder of this section presents a variety of dashboards that provide the insights requested by our system administrators. The only HPC system represented in this section is Eclipse but there is nothing about the dashboards and other mechanisms presented that are system specific. All dashboards presented here display real data from a normal two week period of Eclipse activity. We describe use cases for these dashboards but did not provide in-depth root cause analysis for trends displayed.

A. System Memory Summary Dashboard

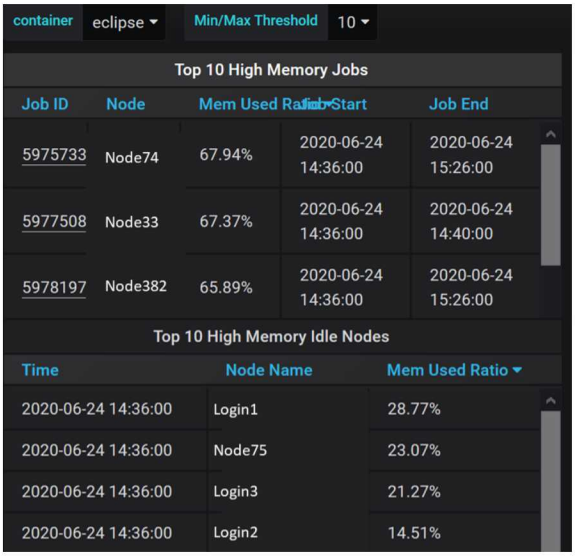

Fig. 3: System Memory Summary dashboard displaying metrics about top nodes in terms of memory usage, bottom node tables omitted for brevity.

The System Memory Summary dashboard in Fig. 3 tabulates the top and bottom jobs in terms of percent of memory high-water mark over the time period specified and presents this in two separate panels (High Memory Jobs shown). The utility of identifying high-memory jobs is to understand temporal memory characteristics of an application; if users are bumping up against memory limits (inform future procurements), or being killed due to excessive memory utilization or memory anomalies on some nodes of a job. For low memory jobs such insight can additionally provide opportunities for more effective problem decomposition. Users may also compare the reported node with high or low memory usage to the node0 host memory usage via the General Job Metric Dashboard described later. The node0 host is defined as the smallest node number host in the job, which is of interest because job schedulers often put rank 0 of a parallel application on this node. Likewise, the same tabulations are provided for high and low memory utilization on nodes not allocated to jobs, including login nodes (High Memory Idle nodes shown). As seen in Fig. 3, many of the high memory usage idle nodes are login nodes, which is expected. However, one of the nodes in this category is a compute node which may occur from abnormal memory utilization, orphaned processes from previous jobs, malicious code execution, and malfunction of system OS code. This information can help identify problem nodes which can be pulled out of the pool available for allocation to jobs.

The memory used percentage is calculated using Equation 1.

$$MemUsed\% = \frac{MemTotal - MemAvailable}{MemTotal} * 100 \quad (1)$$

MemTotal and *MemAvailable* are both metrics read from the /proc/meminfo pseudo-file. The Node column displays the compute node on a particular job which has the maximum percentage of memory used over the display time period.

The min/max threshold input at the top of the dashboard determines the number of jobs or nodes included on each panel. The query for this dashboard can be changed to filter jobs by user. Populating the username field with "system" or with a blank returns data for the entirety of the system. Other dashboards with the "username" field follow the same convention. Our Grafana instance is restricted to trusted user groups so we do not have security concerns on one user seeing another's data.

Clicking on a job number in this panel takes the user to the dashboard in IV-B with time range automatically set to the beginning and end of the job. This is done by using the returned job start and end times from the System Memory Summary query as variables in the hyperlink to the other dashboard.

B. Memory Min/Mean/Max Across Components Dashboard

Fig. 4: Memory Min/Mean/Max across Components dashboard which displays active, dirty memory, and four other metrics (omitted for brevity) for a single job across all nodes with max, mean, and min values for each timestamp. Here, active memory dips correlate with dirty memory spikes.

The Memory Min/Mean/Max across Components dashboard shown in Fig. 4 is a breakdown dashboard that displays the minimum, mean, and maximum of several key memory metrics for a particular job. These metrics are *Active*, *MemAvailable*, *Writeback*, *AnonPages*, *Dirty*, and *Mapped* which all come from the /proc/meminfo pseudo-file. The min and max values at each timestamp correspond to the minimum and maximum values for that metric across all nodes in the job. While this page is usually accessed from the System Memory Summary dashboard, it can also be accessed via the Grafana "dashboard dropdown" menu. Users can modify the job number and time range to meet their query needs. Users can track the load balance and memory usage characteristics of their applications over time as well as check for anomalies in the memory profile. Figure 4 shows the active memory and dirty memory metrics for a particular job. There are two dips in active memory at the same time that there are spikes in dirty memory. This might point to the application waiting on the filesystem to continue progress. It could also convey that the job was not operating at maximum performance because there was an issue with writing to disk. System administrators can check for potential upcoming OOM conditions, for consistently high dirty memory, or if users are misallocating resources (i.e., multiple nodes but using only a small fraction of CPU and memory/memory bandwidth).

C. Lustre Summary Dashboard

The Lustre Summary dashboard in Fig. 5 communicates the utilization of a Lustre filesystem, an important shared resource across the SNL HPC center. There are three different types of analyses being presented on this dashboard: high utilization jobs, high utilization users, and system views of utilization. There are two types of rates shown in these panels: average and peak. These analyses assist the user in identifying jobs with high continuous or peak Lustre usage. For the entirety of the time period selected, metrics passed to the query are summed together at each timestamp. Using this timeseries of a sum of metrics, simply denoted as "met" in the following equations, the average rate is calculated using Equation 2 and the peak rate by Equation 3. Each of these rate equations has its own analysis module. The final timestamp for the job is notated as t_n, the initial as t_0 in these equations.

$$AverageRate = \frac{\sum_{i=1}^{n}(met_i - met_{i-1})}{t_n - t_0} \quad (2)$$

$$PeakRate = \max(\frac{met_i - met_{i-1}}{t_i - t_{i-1}} \text{ for } i = 1...n) \quad (3)$$

The File Ops panels present file event rates from the Lustre client counters, specifically the open, close, create, and unlink events which are all passed into a single analysis module. The "Bps" panels describe the read and write rates (in units of Bytes per second) on the filesystem using read_bytes and write_bytes counters from the same utility. The top jobs panels display the top k jobs in terms of average or peak rates where k is the threshold value selected. The job numbers link to File Operations (Ops) and Bps breakdown dashboards. The top users panels compute the peak and average rates per individual user rather than per job and display the top k users. This provides user-based attribution of file system utilization independent of number or size of jobs.

The timeseries charts on the left of Fig. 5 display a breakout of the metrics used in the top job and user panels, but are computed from totals over the entire system. These are broken out into rates contributed by login nodes and by compute nodes. The reason for this separation is that, for Eclipse, operations on the login nodes contribute significantly to the total usage of the filesystem. This might be indicative of users inappropriately using the login nodes, which would also be of interest. As seen in the left middle panel of Fig. 5, there is a period of time where a substantial part of the Lustre open and close operations happens on the login nodes. The job and user panels do not encapsulate data from the login nodes, so it is important to

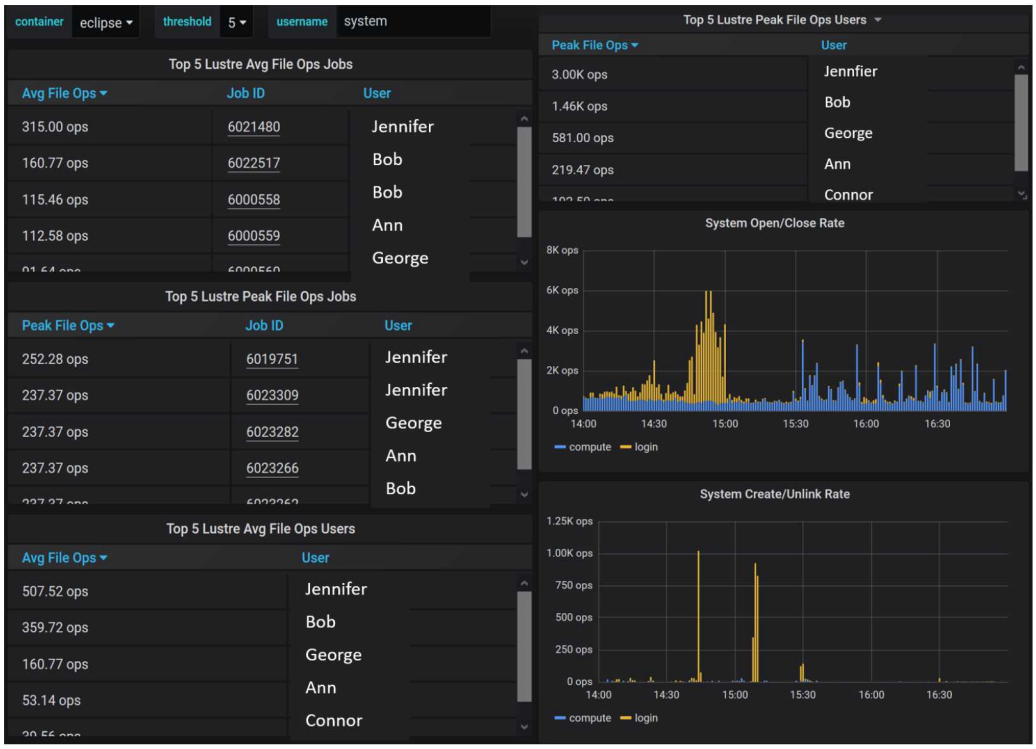

Fig. 5: Lustre Summary dashboard reporting the top jobs and users in terms of peak and average file operations along with a system view of file operations. Bytes read and written tables and plots are calculated as well but omitted for brevity.

include that distinction in the visualizations. Like the System Memory Summary dashboard, a specific user can be selected via the username input at the top of the dashboard. This does not affect the system timeseries plots.

D. File Ops and Bps Breakdown Dashboard

The File Ops and Bps Breakdown dashboards in Fig. 6 and Fig. 7 are further expansions of the metrics described in the Lustre Summary section. A timeseries plot of each metric's rate for a specified job is shown with system information below it for comparison. Rates are calculated as in Equation 3 but without taking the maximum value and simply presenting all values. Users can view the rate of file operations and byte reads/writes as their application progresses. System administrators can understand how much an individual job is contributing to the load on the filesystem. For example, in Fig. 7, the job metric plot has three peaks which corresponds to peaks in the system summary plot.

E. Job Information Dashboard

The Job Information dashboard in Fig. 8 displays all of the jobs that are running during the time period specified. All jobs running in the time range are found. Extra analysis is done to construct a report of the job start time, end time, total time, node0, and hostname list. The job numbers link to the dashboard in IV-F and automatically configures that dashboard's time range to be the time span of the job. A specific user can be selected via the username input at the top of the dashboard. Users and administrators can use this dashboard to get a snapshot of jobs running on the system.

Fig. 6: FileOps Breakdown dashboard showing timeseries for the file open rate of a job compared to the system. Three other file ops metrics are present but omitted for brevity.

F. General Job Metric Query Dashboard

The General Job Metric Query dashboard in Fig. 9 displays a timeseries plot and heatmap of a single metric for all of the nodes associated with a single job over the specified time. In this case, the metric is the number of processes running and is plotted for each node in the timeseries plot. From the

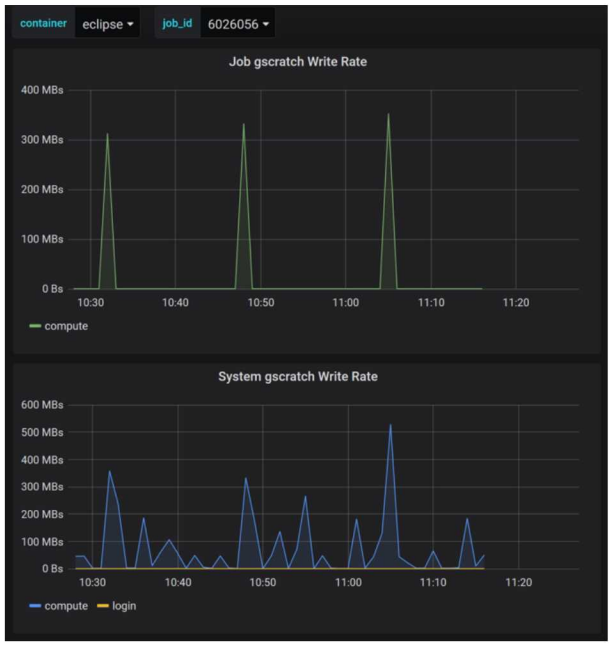

Fig. 7: Bps Breakdown dashboard showing a timeseries for the rate of bytes written of a job compared to the system. The rate of bytes read is present but omitted for brevity.

timeseries plot, we see one node has regular groups of spikes in number of processes and another node has consistently less processes than the mean which may or may not be a normal profile for the application.

The heatmap is used to summarize timeseries plots for high node counts and has a faster render time than the corresponding timeseries plot. The heatmap displays the timeseries plot binned into different regions over time with a color on a blue-red spectrum. Colors near blue indicate a low number of nodes in the bin whereas red indicates a high numbers of nodes. In Fig. 9, most of the nodes have a procs_running value around 35 so most of the bins around that value are orange or red. Any metric can be viewed on this dashboard by changing the selection drop-down boxes at the top. This dashboard is used by developers or users to get raw information about a job. An analysis module is not needed for this dashboard, as simple queries return the datasets required directly. Grafana enables users to export data from any panel to a CSV file, so users can obtain the raw LDMS information for their job.

V. Related Work

There is a long history in visualizing HPC system data. In this section we describe a variety of other research which specifically addresses *general variable* analyses for *runtime* performance monitoring. Our research remains distinct because of the highly customizable, as-needed analyses that are enabled by the data pipeline.

Ganglia [10] is a common HPC monitoring tool which uses RRDTool [11] for data storage. Typical visualizations leverage RRDTool's facility for aging out of data and storing larger time aggregates (e.g., per hour, per day) to drive the data for the interfaces, rather than on-demand analyses. Ganglia typically targets general monitoring, with its high overhead (relative to LDMS [3]) limiting its ability to collect the number of metrics and fidelity of collection necessary for the resource analyses targeted here.

Nikitenko et.al. [6] from Moscow State University created JobDigest which collects 20 CPU and IB metrics from their supercomputer Lomonosov2 and displays these metrics in a variety of timeseries plots and tables. The data was stored in a PostgreSQL database and basic statistic of each metric were calculated and plotted in tables with color coding to indicate the health of that metric for a job. This framework shares much of the basic functionality of our work, however JobDigest does not have as many metrics available to users and developers or the flexibility to perform complex analyses.

Eitzinger et.al. [7] created ClusterCockpit for visualizing job performance monitoring. ClusterCockpit is a custom web application that visualizes HPC data from the Likwid monitoring stack. The interface was created with high-performance and ease of use/installation in mind and deliberately does not support generalized displays that would be needed for as-needed analyses and visualizations. The framework is aimed at supporting performance analysts in small to medium sized HPC centers whereas we are aimed at large scale centers.

Netti et.al. [4] showcases using Data Center Data Base (DCDB) for analyzing job workloads and presenting data to a Grafana interface. DCDB shares many attributes of LDMS in terms of low-latency and range of metrics which can be collected. Like our research, they created a custom plugin for Grafana. They use Grafana to plot power consumption by three different nodes on a production system at the Leibenz Supercomputing Center (LRZ). DCDB does allow for collecting derived metrics from user-specified arithmetic statements, however this seems to be done in the configuration of a node sampler rather than as-needed like our analysis modules.

TACC Stats has a long history in HPC monitoring, with recent advances supporting live data display [17]. Simple statistics (e.g., max, average) are calculated for each job for visualization. The authors note that Python scripts for more complex queries can be written that leverage aggregation functions provided by the Django Object Relation Mapping to their PostgreSQL database, but it is unclear what complexity of query can be supported, how the scripts can be integrated into the architecture for as-needed and repeated use, and how results from such queries would be supported in the dashboard.

Prometheus is an open-source systems monitoring toolkit which can integrate with Grafana to provide data visualizations [20]. Several HPC monitoring frameworks [18], [21] have used Prometheus as their database. The query language for Prometheus supports in-line functions such as rate calculations and aggregations over time. However, Prometheus's data analysis capabilities are limited by it's function set.

Finally, Shoga et.al. [9] created an infrastructure called Sonar to ingest data from a variety of data sources, including LDMS, and then enable Jupyter notebooks to query and analyze the resulting data. They used Kafka with custom LLNL tools and plugins to transport data from spooling directories to their Sonar cluster. The cluster had an Apache Cassandra database with a query API called Scrubjay which presented the database to the Jupyter notebooks. From there, they could create whatever analyses they wanted from the data using Python. This work

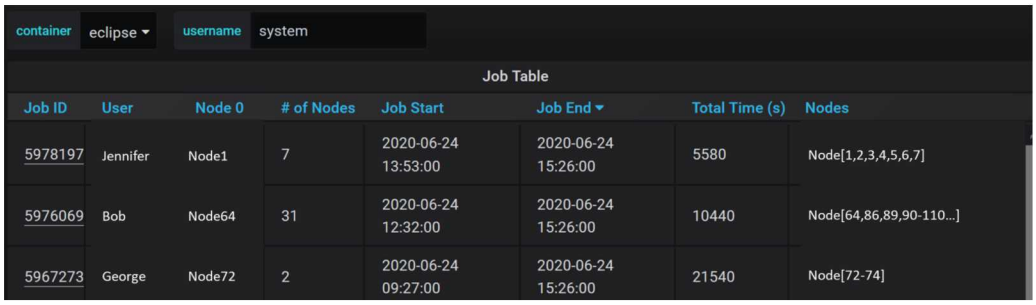

Fig. 8: Job Information dashboard which shows all currently running jobs on the system and some data about them.

Fig. 9: General Job Metric Query dashboard showing the number of processes running on each node during for a single application. A timeseries plot and heatmap describe the trends of this application with regular groups of spiking on a certain node and one node consistently lower than the mean.

shares many similarities to ours, however, our Grafana interface provides a unified area for system administrators and users alike to view data and provides a low barrier to adding and sharing new analyses.

VI. Conclusion and Future Work

The data pipeline presented in this paper showcases a new way to create and present insights about HPC system operations. By using flexible, as-needed python modules within queries, analysis results can be returned without needing to perform unnecessary computation on the entirety of the dataset. New visualizations and derived metrics can be created to address new needs. We have deployed our architecture on a 1500-node HPC system, including analytics that address the priorities of users and system administrators with respect to application and system resource performance understanding. Some users and admins have already used this application with the aid of basic tutorials.

Updates to the data pipeline are currently underway. We are simplifying data management through direct storage to SOS containers, rather than intermediate CSV files, which will eliminate a lag in presenting data. We are ingesting more job and user metadata for faster queries. We are working with code development teams to ingest application performance metrics into the database which will allow us to correlate system events with application events in order to answer the universal user question of ”Why is my job slow today?” New derived metrics for network congestion and integration of machine learning analyses for detecting system anomalies are also underway. Finally, a distributed version of SOS, Distributed Scalable Object Storage (DSOS), is currently in development at OGC which will enable the database to scale across storage devices.

Acknowledgments

The authors would like to thank Ann Gentile from SNL for her constructive comments and review which were instrumental in properly conveying the work in this paper. Additional thanks to the SNL CAPVIZ administrative team for providing their needs which guided this work.

REFERENCES

[1] Tuncer, Ozan, et al. "Diagnosing performance variations in HPC applications using machine learning." International Supercomputing Conference. Springer, Cham, 2017.

[2] Beneventi, Francesco, Cavazzoni, and Benini. "Continuous learning of HPC infrastructure models using big data analytics and in-memory processing tools." Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017. IEEE, 2017.

[3] A. Agelastos, B. Allan, et al. "Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications," in Proc. IEEE/ACM International Conference for High Performance Storage, Networking, and Analysis (SC14), 2014.

[4] Netti, Alessio, et al. "From facility to application sensor data: modular, continuous and holistic monitoring with DCDB." Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2019.

[5] DeConinck, A, et al. "Design and implementation of a scalable monitoring system for Trinity", Proc. Cray User's Group. 2016.

[6] Nikitenko, Dmitry, et al. "JobDigest–detailed system monitoring-based supercomputer application behavior analysis." Russian Supercomputing Days. Springer, Cham, 2017.

[7] Eitzinger, Jan, et al. "ClusterCockpit—A web application for job-specific performance monitoring." 2019 IEEE International Conference on Cluster Computing (CTER). IEEE, 2019.

[8] Naqvi, Syeda Noor Zehra, Sofia Yfantidou, and Esteban Zimányi. "Time series databases and influxdb." Studienarbeit, Université Libre de Bruxelles (2017).

[9] Shoga, Gimenez, Gamblin, et al. "Sonar: Performance Monitoring and Analysis." Presentation at Lawrence Livermore National Laboratories, 2018.

[10] Ganglia, 6-Jul-2020. [Online]. Available: http://ganglia.info.

[11] RRDTool, 6-Jul-2020. [Online]. Available: http:/rrdtool.org

[12] OpenGridComputing, "SOS - Scalable Object Store," 6-Jul-2020. [Online]. Available: https://github.com/ovis-hpc/sos/tree/OVIS-4.3.3

[13] OpenGridComputing, "sosdb-ui," 6-Jul-2020. [Online]. Available: http://github.com/nick-enoent/sosdb-ui

[14] OpenGridComputing, "sosdb-grafana," 6-Jul-2020. [Online]. Available: http://github.com/nick-enoent/sosdb-grafana

[15] OpenGridComputing, "sosdb-grafana," 6-Jul-2020. [Online]. Available: http://github.com/nick-enoent/numsos

[16] Grafana Labs, "Grafana Documentation," 6-Jul-2020. [Online]. Available: https://grafana.com/docs/

[17] M. Dwyer, J. Hwang, A. Shires and J. Cohen, "Application of Comprehensive Data Analysis for Interactive, Hierarchical Views of HPC Workloads," 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 3585-3589, doi: 10.1109/BigData.2018.8622330.

[18] Sukhija, Nitin, and Elizabeth Bautista. "Towards a Framework for Monitoring and Analyzing High Performance Computing Environments Using Kubernetes and Prometheus." 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI). IEEE, 2019.

[19] Sandia National Laboratories, "HPC Capacity Cluster Platforms" 6-Jul-2020. [Online]. Available: http://hpc.sandia.gov/HPC%20Production%20Clusters/index.html

[20] Prometheus "Prometheus Documentation," 6-Jul-2020. [Online]. Available: https://prometheus.io/docs/prometheus/latest

[21] Weidner, Ole, Malcolm Atkinson, and Adam Barker. "Towards a Comprehensive Framework for Telemetry Data in HPC Environments." arXiv preprint arXiv:1707.02639 (2017).