GUIDE: A Scalable Information Directory Service to Collect, Federate, and Analyze Logs for Operational Insights into a Leadership HPC Facility

Sudharshan S. Vazhkudai
Oak Ridge National Laboratory
One Bethel Valley Rd
Oak Ridge, TN 37831, USA
vazhkudaiss@ornl.gov

Ross Miller
Oak Ridge National Laboratory
One Bethel Valley Rd
Oak Ridge, TN 37831, USA
rgmiller@ornl.gov

Devesh Tiwari
Northeastern University
360 Huntington Ave
Boston, MA 02115, USA
tiwari@northeastern.edu

Christopher Zimmer
Oak Ridge National Laboratory
One Bethel Valley Rd
Oak Ridge, TN 37831, USA
zimmercj@ornl.gov

Feiyi Wang
Oak Ridge National Laboratory
One Bethel Valley Rd
Oak Ridge, TN 37831, USA
fwang2@ornl.gov

Sarp Oral
Oak Ridge National Laboratory
One Bethel Valley Rd
Oak Ridge, TN 37831, USA
oralhs@ornl.gov

Raghul Gunasekaran
Oak Ridge National Laboratory
One Bethel Valley Rd
Oak Ridge, TN 37831, USA
gunasekaranr@ornl.gov

Deryl Steinert
Oak Ridge National Laboratory
One Bethel Valley Rd
Oak Ridge, TN 37831, USA
steinertda@ornl.gov

ABSTRACT

In this paper, we describe the GUIDE framework used to collect, federate, and analyze log data from the Oak Ridge Leadership Computing Facility (OLCF), and how we use that data to derive insights into facility operations. We collect system logs and extract monitoring data at every level of the various OLCF subsystems, and have developed a suite of pre-processing tools to make the raw data consumable. The cleansed logs are then ingested and federated into a central, scalable data warehouse, Splunk, that offers storage, indexing, querying, and visualization capabilities. We have further developed and deployed a set of tools to analyze these multiple disparate log streams in concert and derive operational insights. We describe our experience from developing and deploying the GUIDE infrastructure, and deriving valuable insights on the various subsystems, based on two years of operations in the production OLCF environment.

CCS CONCEPTS

• **General and reference** → **Performance**; **Measurement**; **Reliability**; **Metrics**; • **Information systems** → *Data warehouses*; *Extraction, transformation and loading*; *Data analytics*; *Data mining*;

SC17, Denver, CO, USA

978-1-4503-5114-0/17/11…$15.00
DOI: 10.1145/3126908.3126946

ACM Reference format:
Sudharshan S. Vazhkudai, Ross Miller, Devesh Tiwari, Christopher Zimmer, Feiyi Wang, Sarp Oral, Raghul Gunasekaran, and Deryl Steinert. 2017. GUIDE: A Scalable Information Directory Service to Collect, Federate, and Analyze Logs for Operational Insights into a Leadership HPC Facility. In *Proceedings of SC17, Denver, CO, USA, November 12–17, 2017,* 12 pages.
DOI: 10.1145/3126908.3126946

1 INTRODUCTION

The Oak Ridge Leadership Computing Facility (OLCF) hosts several leadership-class systems such as the Titan machine (No. 4 machine in the Top500 list) [3], the Lustre-based [27] Spider II parallel file system [22] (PFS) (fastest in the world), the disk/tape-based HPSS archival storage system [28] and several data analysis clusters. Each one of these host complex subsystems that offer unique resources to a national scientific, supercomputing user base. For example, the compute subsystem of the Titan machine consists of 18,688 heterogeneous CPU/GPU compute nodes for a total of 560,000 CPU and GPU cores and a peak performance of 17.59 petaflops. The interconnect on Titan offers a high-speed, low-latency Gemini network [14] configured in a 3D Torus topology. The Spider II PFS offers 32 PB of usable capacity and more than 1 TB/s I/O throughput, and stores around 1 billion files. Users from disparate science domains routinely run massively parallel simulation jobs on these systems to model complex physical phenomena and glean insights. OLCF serves around 4 billion core hours every year to such jobs.

A key requirement for obtaining a nationally peer-reviewed scientific project allocation on OLCF is that the jobs demonstrate a "capability" metric that makes effective, concerted use of the aforementioned leadership resources. For instance, a capability job can be one that uses a majority of the 560,000 cores in a tightly-coupled,

low-latency parallel simulation; or, a job that has stringent I/O needs and stresses the storage subsystem in unique ways. In summary, the OLCF caters to users' unique, extreme-scale computational needs that cannot be readily satisfied elsewhere.

To deploy and efficiently operate such a large scale center, we need to answer numerous questions for both initial deployment and continuous provisioning, optimize day-to-day operations by identifying hotspots and performance bottlenecks, and tune both the underlying system and the applications' use of the same. However, to realize this goal, we need a robust and scalable log data collection and monitoring infrastructure, the ability to not just analyze the log stream of any one subsystem in isolation, but to fuse and correlate the disparate streams from multiple subsystems in concert, and a suite of higher-level services and tools based on such analytics. Note that this is much more than typical monitoring, e.g., with Nagios [2], that is maintained to inform system administrators of system health, or project accounting information gathered for reporting, e.g., usage by projects. Deeper insights into system usage and user application/job behavior can only be unraveled through the rich analytics of log data, which may involve higher-order techniques such as statistical analysis, visual analytics, data mining, and cross correlation.

In this paper, we present the design, implementation, deployment and operational impact of *GUIDE, the "Grand Unified Information Directory Environment"* for OLCF. GUIDE has been operational at OLCF for more than two years. The GUIDE framework is based on the following principles. (i) *Data Extraction and Logs Collection:* We advocate the collection of logs, as well as the extraction of data at each and every level of the various OLCF subsystems. For example, at the storage subsystem, we extract data from the disk layer (the 2,016 OSTs, encompassing the 20,160 disks), the redundant RAID controllers (72), the OSSes (288), and the Lustre PFS level. Often times, this requires the development of custom tools to extract the data, and not just aggregating logs that may have already been collected, e.g., scheduler or RAS logs. We describe the development of several such tools within the GUIDE framework. (ii) *Pre-processing:* Whether it is extracted data or collected logs, there is the need to pre-process them by way of procedures such as data cleansing, statistical analysis or categorization, before they can be federated and analyzed further. We have developed several techniques to accomplish this for the various data/log streams. (iii) *Federation:* The data/logs need to be federated in a scalable repository that supports efficient indexing, querying, and visualization. To this end, we describe our use of the commercial data warehouse, Splunk [25]. (iv) *Post-processing:* We have developed a suite of analytics and post-processing techniques that we apply not just on single data streams, but on a combination of data sources. These techniques include cross-correlation, auto-correlation, visual analytics, alerts, sliding-window analysis, and mining. For example, analyzing a single data source such as I/O bandwidth from the storage controllers might enable the understanding of read/write ratios of the storage system, I/O hotspots, and the like, which is very valuable for both current operations and future provisioning. However, using it in concert with the Titan reliability data, obtained by processing the RAS logs, can provide more proactive guidance to applications on their checkpoint frequency. We have developed tools in this fashion, layering them atop the federated logs/data in the GUIDE framework, and highlighting a key strength of GUIDE, the concerted use of multiple data sources. (v) *Operational Impact:* Finally, by applying the pre-processing and post-processing analytics on the data/logs, we derive numerous insights into the operations of the OLCF HPC facility, in a variety of areas such as storage, scheduling, RAS, archive and interconnects. We describe our experience with GUIDE, based on two years of operations within the OLCF.

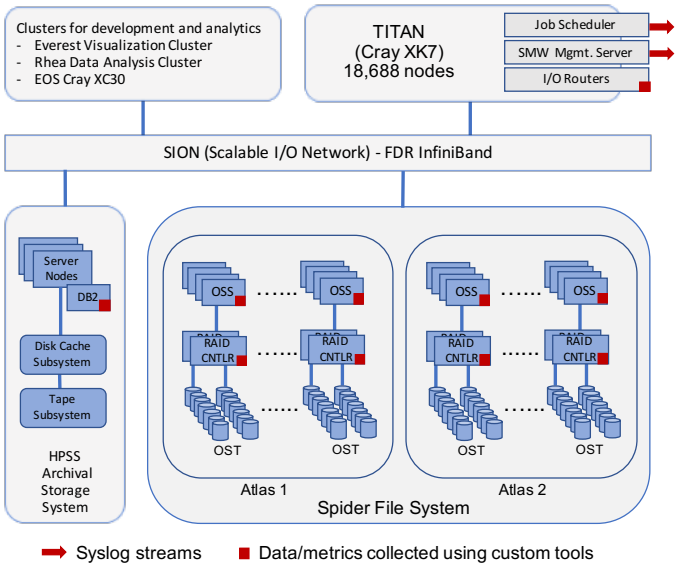


Figure 1: OLCF System Architecture

2 OLCF SUBSYSTEMS

Figure 1 presents an overview of the OLCF's HPC resource fabric.

Compute subsystem: The Titan supercomputer, a 17.59 petaflops Cray XK7 system, is OLCF's primary compute platform, and consists of 18,688 compute nodes. Each compute node consists of a 16-core AMD Opteron CPU with 32 GB of memory and an NVIDIA K20X GPU (Kepler G110 processor) with 6 GB of memory.

Interconnection and Scheduling subsystem: Titan's compute nodes are connected via a high-performance Gemini interconnection network, which is a 3D torus with XYZ dimensions of 25x16x24. The high-dimensionality in 3D torus networks means that they are composed of many small routing devices, making several routing decisions feasible, but also introducing several hops. The OLCF runs around 500 jobs every day on Titan (~182,000 jobs/yr), atop this infrastructure. The jobs are scheduled using the Application Level Placement Scheduler (ALPS) and the MOAB scheduler. ALPS is responsible for enumerating all of the nodes within the system and creating a network-aware ordered list. MOAB handles both job scheduling and resource allocation. Scheduled jobs are provided resources in order from the list created by ALPS.

Storage subsystem: To support the I/O and storage needs of the jobs running on Titan and other analysis clusters (26,000 compute nodes in aggregate), the OLCF supports the Spider II scratch storage system, a Lustre-based PFS with 32 PB of usable storage and a peak aggregate bandwidth of over 1 TB/s. Spider II is connected to all OLCF resources via an InfiniBand FDR network (the Scalable I/O

Network (SION)). Spider II is available as two namespaces, *Atlas1* and *Atlas2*, built on identical and non-overlapping hardware. Access to each namespace is via 144 Object Storage Servers (OSS) (288 in all), each with 7 Object Storage Targets (OST) for a total of 1,008 OSTs (2,016 in all). Each OST is a tier of 10 2TB NL-SAS HDDs, and all of the disk tiers are managed by 36 DDN SFA12KX redundant (RAID) I/O controller pairs. Spider caters to the short-term I/O needs of user jobs, and files therein are purged based on a two-week access policy. OLCF currently has over 1 billion file system objects stored on Spider II.

For users' long-term storage needs, OLCF provides the HPSS disk/tape-based archival storage. The HPSS installation has 60 PB of data storage across 65 million files archived over the past two decades.

3 OVERVIEW

The overarching goal of the GUIDE framework is to provide an infrastructure for the scalable collection, federation and analytics of logs and monitoring data, to optimize OLCF operations. We are interested in optimizing the runtime performance, I/O throughput, and reliability of applications as they run on the OLCF subsystems, tuning the operations of the subsystems themselves for better utilization, and guiding the future provisioning of the various resources. Figure 2 presents an overview of the GUIDE framework.

Fabric: At the lowest level is the OLCF's fabric, comprising of the various subsystems described earlier, e.g., compute node, interconnection network, scheduling and storage.

Logs and Data Collection: To optimize both user applications and the subsystems, there is the need for a wealth of monitoring information on the subsystems' operations and how applications are using them. We advocate the collection of logs and the extraction of monitoring data at every level of the OLCF subsystem of interest. Note that there is a distinction between accessing logs that are already being collected by the system (e.g., RAS logs from the compute nodes via syslog streams, Lustre jobstats, internal HPSS operations logs, MOAB logs) and monitoring data that needs to be explicitly extracted using specialized tools (e.g., interconnect congestion data, I/O data from several storage layers). In GUIDE, we address both logs collection and data extraction by developing scalable, lightweight and non-invasive tools where appropriate for the OLCF fabric. Further, our collection tools need to consider the potentially large data sizes, and further determine an optimal frequency. For an extreme-scale environment like OLCF, there is often a dearth of such metadata about subsystem operations at this level of detail, and a lack of scalable monitoring tools, which we have developed.

Pre-processing: Both system logs and extracted monitoring data from our tools, need to be pre-processed before use. Raw data may often be incorrectly formatted, missing certain details, too much in volume and therefore require statistical summarization, or require qualitative analysis, before they can be federated or used in any fashion. We have developed and deployed a suite of tools in GUIDE, in areas like data cleansing to cleanup the collected logs and data, a set of statistical analysis for better data representation of the voluminous log data, and several qualitative categorization and heuristics-based approximation. It is worth noting that since the OLCF logs represent disparate subsystems, not all of the above techniques will be applicable for each data stream, and relevant techniques need to be applied carefully after studying the logs.

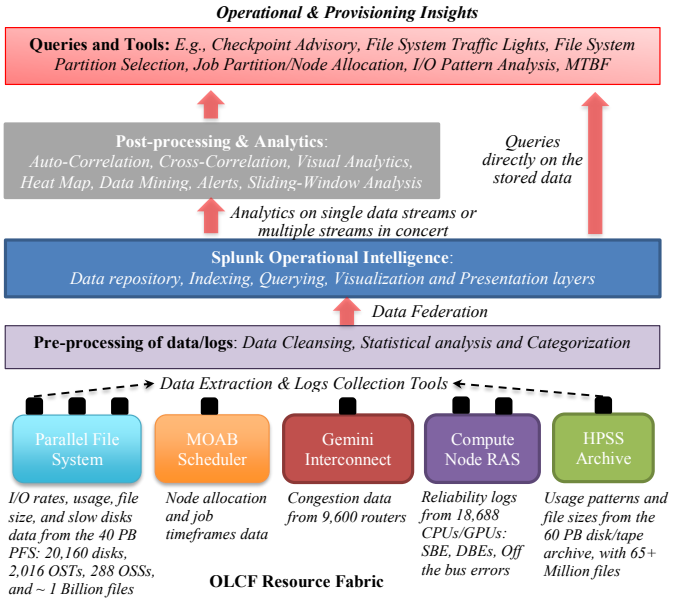


Figure 2: GUIDE architecture.

Federation: A key piece that was missing in over a decade of OLCF operations is the federation of the logs into a central, scalable data warehouse that offered storage, indexing, querying and visualization services. To begin with, as we stated earlier, not all data was collected, and those collected were stored and analyzed using custom approaches—often because different operational teams controlled the various subsystems—without any common federation services that an operational intelligence tool could provide. In GUIDE, we have attempted to address this key drawback, and offer the ability to federate and index all of the collected logs within a central Splunk [25] data warehouse for the entire facility. This provides several desired features like uniform ways and formats to ingest data, the ability to index log data for fast and efficient lookup, the ability to query time-series data, and to process multiple data streams in concert.

Analytics: We envision applying a suite of analytics to derive key operational insights. We have developed and deployed techniques ranging from auto and cross-correlation, visual analytics, data mining and sliding-window based analysis, operating on several data streams together. Further, we have developed higher-level tools based on these techniques. Our analytics optimizes application performance and subsystem operations, identifies and resolves performance bottlenecks, improves I/O and reliability through the concerted analysis of multiple logs, improves utilization, and guides the provisioning of future systems.

4 LOGS AND PRE-PROCESSING

Logs collection and cleansing are the first step towards a federated information directory service. It is also particularly challenging due to the scale and diversity of the distributed data sources. In this section, we will first present an overview of the pre-processing and

statistical techniques, and the log data categorization. Then, we present the log extraction process from the OLCF subsystems.

4.1 Pre-processing Techniques for the Logs

The pre-processing techniques we have developed for the logs can be broadly classified into (i) Data Cleansing, (ii) Statistical Analysis, and (iii) Categorization such as Heuristics extraction and binning.

4.1.1 Data Cleansing. In most cases, log data from different sources cannot be directly used to perform meaningful data analysis, either due to missing pieces, and/or incorrectly formatted data. Therefore, a prerequisite to every data analysis and fusion task is to perform data cleansing. This involves identifying missing pieces, data with incorrect formatting, and ensuring the integrity of data. These steps are performed by domain experts for each stream of data (i.e., storage, archival data, scheduling, job resource consumption, RAS).

Identifying incorrectly formatted data is relatively easier. We have developed a data stream specific schema that the events in the data stream should adhere too. The schema is passed to a processing engine that detects incorrectly formatted data events in the data stream, which are discarded from analysis. The next step is identifying missing data pieces. For this step, we have used the rate of change in the event-rate as an anomaly detector. If the rate of events becomes zero, it indicates missing data. However, determining the period over which this rate should be calculated is dependent on the type of stream. For example, storage logs can have bandwidth values every few minutes. But, RAS events can be relatively more sparse and hence, half an hour is a more appropriate window. Finally, we check the integrity of the data using domain knowledge to ensure that there are no events with bogus values or unknown event types.

There is no "single-style-fits-all" approach for these steps across different data streams, but there are shared commonalities that were used to make the process more efficient. For example, RAS log analysis can indicate which jobs were impacted by system failures at what time, and this can be used to identify jobs in the job resource consumption log, which should be checked more carefully since failures may have impacted the reporting daemon.

4.1.2 Statistical Analysis. It is rarely the case that the entire raw form of data representation will be of interest. Sometimes, it is almost impossible to expose them all due to the sheer volume. It is therefore a common practice to aggregate logs using a host of statistical techniques. In addition to the usual descriptive statistical measures such as *min*, *max*, *average*, *sum*, *percentile*, *standard deviation*, *PDF*, and *CDF*, GUIDE also employs the following techniques in various cases: (i) *Histogram*: This is one of the most effective methods for aggregating a large amount of data. It essentially approximates a probabilistic density function with either streaming data or batch data. The choice of binning width is an important consideration, but it is also rather context specific and subject to tuning for optimal results. (ii) *top N*: An often requested measurement that is applicable to both streaming and batch data, e.g., top 100 files that are taking up the disk space; top 10 applications that are issuing most I/O requests, etc.

4.1.3 Categorization. Often times, log data can be too verbose or require computationally expensive techniques to extract meaning. In these cases, techniques such as heuristic extraction and binning can be applied to reduce the log information that is stored in order to enable quick understanding and recall for further analytics.

Heuristics: Heuristic functions may be used to create representative metrics that trade-off completeness or accuracy to improve the performance or storage cost of meaningful system state. For example, for scheduling data we use a heuristic known as Average Hop Count as a representative measure of job-node allocation. This metric is calculated during pre-processing and stored with the log data due to the high cost of calculation.

Binning: While value based binning is used in the generation of histograms, another type of binning can be used to classify a level of event and track frequencies of such occurrences. An example of this is binning incoming job entries based on the quality of their allocation over the network. A highly fragmented job will be exposed to more network impacts than a highly compact job.

Events classification: There is also the need to understand various events from the logs as they are often cryptic; correlated events are often logged differently. We have performed a categorization by developing an understanding of the "well-known events," performed root-cause analysis and correlated them. For example, RAS logs have numerous failure events that need to be understood and classified properly, before they can be analyzed.

4.2 Logs Extraction

Despite the diverse source and format of the logs, there is a common thread of questions and design issues we address in every case: What are the different levels of the subsystems at which logs can be collected? Are the logs already in place or do they need to be extracted? What is the format and size of the log data? How frequently do we need to collect to make meaningful inferences? Is there any pre-processing needed before federating the logs? Where should the logs be persisted? Are there any interferences that can cause negative operational impact? In the following section, we explain how we extracted logs from the PFS, compute node RAS, interconnection network, the scheduling system and the archive, and address the common concerns and differences.

4.2.1 PFS and Backend Storage. As described in Section 2, the Spider storage system consists of two subsystems: a distributed backend storage system composed of 36 DataDirect Networks (DDN) 12KX redundant RAID controller pairs with 20,160 disks, and a Lustre PFS deployed on 288 I/O servers, abstracting and projecting the RAID 6 redundancy groups organized by the DDN RAID controllers. We extract logs and data at both levels.

At the storage controller level, we have developed a python-based tool [20] to interface with the DDN controller API, which polls the controllers at regular intervals. Since the controllers are all independent, the synchronization of this polling effort is important. At every interval, we extract from each controller a number of performance metrics, such as I/O request sizes, write and read bandwidth and IOPs. We also extract the status of each LUN, such as whether a disk has failed or whether the LUN is rebuilding its RAID parity data. The extraction tool stores this data in an in-memory mySQL database. At each poll interval, the old data is

overwritten with the new data in the in-memory database. This database is queried at regular intervals to federate the data.

At the Lustre PFS level, the I/O servers are polled and various data streams are collected at regular intervals. The file system data is primarily obtained from Lustre job_stats and brw_stats. The Lustre job_stats statistics is collected from each OST (e.g., obdfilter.*.job_stats) and MDT (mdt.*.job_stats), individually. The Lustre job_stats allows each Lustre client to tag its RPCs with a scheduler defined PBS job id. Since Titan only allows a single job to be run on the compute node at any given time, this in turn allows us to identify each Lustre RPC per compute job at each OST independently. In the end, by using the Lustre job_stats, we are able to collect per job and per OST statistics, including write and read bandwidth, I/O size, and number of metadata operations for each polling interval. The brw_stats is lower in terms of the Lustre I/O stack, compared to the job_stats, and reports statistics reflecting the I/O operations issued by Lustre to the DDN storage subsystem. This data is, again, collected on a per OST basis, and includes detailed information on write and read I/O operations.

We have also developed a tool, *fprof*, to perform a parallel tree walk of the entire file system (~ 1 billion file entries). The light-weight, yet scalable, profiling tool can provide us information such as the number of files and file size distribution (histograms), mean file size, maximum number of files within a single shared directory, the top N largest files in the system and more. The tool is executed manually on a bi-weekly basis, and the profiling results are federated.

Lastly, we also run 'ls' commands from a number of servers every five minutes and store in Splunk the time required for the command to complete. This sounds (and is) rather simple, but it turns out to be a rather effective measure of the interactive responsiveness of the PFS at various end points of the overall OLCF infrastructure.

4.2.2 Compute Node RAS. RAS data and console logs are pushed from Titan's compute nodes by the vendor, Cray, to a server where they are processed to extract events of interest. Specifically, this processing uses simple event correlators (SEC) on software management workstations (SMW) to log critical system events. The SEC identifies all RAS events of interest. The rules are regularly updated to accommodate the addition of new components and events. This is a comprehensive log of critical system events that also alerts the system operators of abnormal behavior.

In this work, we focus on CPU, GPU and memory related failure events, which may be caused by hardware or radiation-induced bit corruptions, single and double bit errors, off-the-bus, and ECC page retirement. We also need to know where in the memory the errors occur (e.g., L1/L2 cache, register file or device memory). First, we have to develop an understanding of all event types. Log events are often cryptic and need domain expertise to understand the meaning of such events. Second, there are multiple event types which are correlated, but the description does not indicate the same. We have performed correlation analysis to extract similar events and perform analysis on them. Finally, we perform root-cause analysis on these events. We use this framework to catch trends and display the results of the root-cause analysis.

4.2.3 Scheduler. Titan resource management is handled by Adaptive Computing's MOAB software. MOAB logs several metrics that are used to measure the efficacy of scheduling policies in producing desirable outcomes. For instance, on Titan, we track a measure of job fragmentation calculated by the average Gemini distance between every node in the job.

Log collection of scheduling data is a two-phase approach. The first phase acquires the daily log generated by MOAB and applies data filters. These data filters are important in reducing the amount of redundant data. For example, a full-system-scale job on Titan will create a single record in MOAB's logs that contains ~4MB of information. A single job will have four records related to the job, indicating submission, dispatch, start, and completion times. However, the job completion record contains all of the relevant information of the previous records. By filtering solely for job completion records and reducing the cost of node encoding, we can reduce the data cost of the entry from 4MB to 64K.

The second phase of data collection is augmenting the entry with relevant statistics. The goal of pre-calculating statistics for the entry reduces the cost of data analysis later. For scheduling data, we calculate several expensive graph heuristics based on the compute nodes a job receives. As mentioned previously, average Gemini interconnect distance is a measure of the XYZ dimension ordered routing, the number of hops between all nodes within the job allocation. This value is averaged, and provides a good measure of a job's density on the 3D torus network. Another important measure to pre-calculate is the dimensional boundary of the job. Since Titan's network is anisotropic, it is important to understand the impacts to a job due to asymmetric bandwidth.

4.2.4 Interconnection Network. We use the Gemini interconnect data to get insights into network performance for the PFS. However, since there are no vendor supplied mechanisms that log network traffic on Titan, we created the I/O Router Congestion Daemon (IORCD) [30] that uses network performance counters to track I/O bandwidth moving through routers within the Gemini network.

High fidelity measurement of the Cray XK7 Gemini network is a complex task. The network, a 3D torus, contains 9,600 Gemini routers connected to couplets of compute nodes. Each router contains performance counters that can be used to determine the magnitude and directionality of traffic. However, accessing these counters is expensive.

In order to make the IORCD collect data, we integrated the sampling with the Gemini-Performance Counter Daemon (GPCD). GPCD effectively employs a kernel module that enables the sampling of both Gemini Router and Gemini network interface card (NIC) performance counters through an IOCTL to the kernel. This mechanism allows us to get back a per-tile set of counters related to network performance. Each Gemini router is composed of 48 tiles, and each tile provides 6 performance counters for the link associated with the tile. For our effort, we were mostly interested in the tile which corresponded to host-side transfers. On Gemini routers there are eight links associated with two hosts; with messaging overhead, we see roughly 5.8 GB/s of bandwidth to each host. These links are used by IORCD to establish a measure of congestion of traffic flowing into and out of the I/O Router. Through analysis we found that the congestion counters strongly correlated to read and write bandwidth to the PFS from Titan.

Deployment of IORCD required careful evaluation to make collection and logging transparent. IORCD uses a sampling interval of five seconds for data collection, and this data is collated in memory and pushed for federation at configurable intervals to avoid generating too much network noise.

4.2.5 Archival Storage. The daily statistics for the HPSS archival storage system are retrieved from the HPSS core server process once per day. A python script is used to execute a command to query the core server via the HPSS hpssadm client utility. The hpssadm command used to query the HPSS core server is: `server info -specific -name "Core Server"`. The output from the above command is then formatted into a single syslog record containing key-value pairs for (among other things) the number of files, directories, bytes on disk/tape, and free disk/tape bytes. The syslog record is less than 500 bytes in length.

The HPSS DB2 database maintains file metadata, which has a wealth of other information. In addition to the above daily statistics, our script queries the HPSS NS_OBJECT and BITFILE DB2 tables once a month to generate the File and Byte Count histograms displayed in GUIDE. The DB2 query is quite complex, but the results are written to a single syslog record containing a snapshot of the file and byte counts by File Size, File Age, Most Recent Access Time, and Time Between Most Recent Access and File Creation.

5 DATA FEDERATION

Having identified and preprocessed the log/data streams across the OLCF infrastructure, the next step is to federate the data to gain operational insights across systems. GUIDE's federated data store needed to serve two purposes. First, it must be able to ingest, store, index and process data in real-time from a wide variety of data sources. Second, we needed a well defined interface for querying, analyzing time-series data, creating graphs, generating reports and showing custom dashboards for visualization. We chose a commercial software package called Splunk [25] for federating all our system log and data streams. Splunk is a well-known package for business intelligence and data mining. We did consider and evaluate other platforms and tools such as the ELK stack (ElasticSearch, LogStash & Kibana) [8] as well as a combination of InfluxDB [18] and Grafana [19]. At the time of our evaluation, these tools were not deemed mature enough for operational purposes, and some of the required features were still in development. While Splunk was not specifically designed for HPC purposes, it is a commercially marketed enterprise software package that has several advantages, including being a single integrated package. Further, Splunk was already being used at OLCF for security monitoring, which also influenced our choice, as it was already deployed and available.

On the downside, to some extent the choice of Splunk dictated what sort of pre-processing was needed. Splunk is primarily built to process system data and expects data to be broken up into discrete, time-stamped events such as syslog messages or web server access logs. Additionally, if Splunk views anything in the text that looks like a key/value pair, it automatically creates a queryable variable from the key and assigns the value to it. The various preprocessing utilities therefore had to format their output into text events containing key/value pairs. In some cases, this required us to significantly reformat the raw data.

Splunk can ingest data in a number of ways. The first and simplest method is to just point Splunk at a text file. This works perfectly for syslog or web server logs, and Splunk can recognize when the file is appended and only ingest the new data. Splunk is also capable of recognizing when a log file has been rotated (i.e., the log file was closed and renamed and then a new file with the original name created) and automatically pick up the new data. A second method that Splunk supports well is database integration. We can configure Splunk to periodically execute user-supplied queries as well as scan entire tables. Finally, Splunk has a well-defined REST API, which means data can be sent directly to the Splunk server. For GUIDE, the vast majority of the data was ingested from log streams as text files. We also used the database integration to gather the output from the I/O controllers. (See section 4.2.1.)

6 OPERATIONAL IMPACT VIA ANALYTICS

We highlight the operational impact that we have realized for the OLCF subsystems by applying a suite of analytics on the federated log data in Splunk. The analysis allowed us to gain much needed insights into the OLCF subsystem operations, and resulted in improved performance, capacity utilization, efficiency and planning.

6.1 Overview of Analytics Techniques

We have applied the following techniques to derive insights. Specific details on how we used these approaches for operational value are covered in the subsequent sections.

Cross-correlation between streams: Cross-correlation refers to the process of combining multiple streams of data in such a way that we can correlate events and perform meaningful analysis across different data formats and data streams. For example, we use storage bandwidth, RAS and job scheduling information to identify optimal checkpointing frequency.

Auto-correlation: Auto-correlation is a technique used to extract the periodicity of delay between signals. This technique is particularly useful for filtering noise from network and I/O time-series data and identifying repeatable patterns. Regular periodic signals can be used for identifying application signatures or things such as identifying the periodicity of I/O in application checkpointing.

Visual Analytics: A visual representation of the log streams can provide a very valuable perspective by quickly identifying hotspots. As noted in section 5, one of the reasons we have built the GUIDE framework atop Splunk was its built-in visualization system. Using Splunk, we have created several different "dashboards," comprising of visualizations viewable via any web browser. In these dashboards, we have presented line and bar charts of time-series data, histograms for distribution data, and pie and scatter plots for various other data. Further, using external packages, we have created custom visualizations that Splunk cannot handle.

Sliding-Window Analysis: The log and data streams being processed via the GUIDE framework are discrete events with a timestamp. Interpreting these time-series events requires segmenting the data into periods or windows of significance. For example, analyzing job specific information requires viewing the log or aggregating performance statistics for the runtime of the job. Also, monitoring real-time usage of the file system requires aggregating statistics over intervals varying from a few minutes to several hours. Finally,

root cause analysis of RAS log errors requires the interpretations of log data in appropriate time windows. Thus, most of the tools and analysis done using GUIDE employed a sliding window approach to segment time series data for meaningful insights.

Mining: Finally, we have also used well-known supervised learning models, e.g., Random Forest [5] and Support Vector Machines [26] to evaluate different facets of data streams, e.g., topological impact of node-allocation data on job performance.

6.2 Storage System Use cases

GUIDE has been useful in both optimizing current Spider II operations (6.2.1, 6.2.2, and 6.2.3), as well as in provisioning the future 250 PB Spider III file system that will be deployed in 2017. (6.2.4 and 6.2.5).

6.2.1 PFS Responsiveness. The perceived responsiveness of a PFS is mostly a direct function of the metadata server performance. If the MDS has hung processes or is overloaded, the users will first notice that routine interactions, e.g., "ls," will take longer to complete. We use the "ls" monitoring data (Section 4.2.1) in Splunk to provide system administrators alerts so they can quickly identify problems and take corrective measures. In addition, the ls data and the data from the RAID groups (e.g., degraded, rebuilding, offline) are combined to make available to users a set of "status lights" that show the health of the PFS (green if normal, yellow if reduced performance, and red if PFS is offline) on the OLCF web site.

6.2.2 On-the-fly PFS Namespace Selection Tool. GUIDE dashboards provide insights into PFS trends and bottlenecks. One such bottleneck was load imbalance between the PFS namespaces (i.e., Atlas1, Atlas2) due to the static assignment of projects to namespaces. This led us to the development of a file system partition selection tool (FS-Select) for users to query, at runtime, the namespace to use based on the average workload in the last one, two and four hour windows. FS-Select identifies the lightly loaded namespace by querying Splunk periodically for the I/O controller data, and calculating a sliding window average of the PFS workload in the last few hours. When a user job is scheduled to run, it queries the service for the namespace, and uses it for its I/O. The tool was particularly aimed at checkpoint workloads that were less concerned about data being distributed across the namespaces.

6.2.3 Visual Analytics of Storage Backend and PFS. As stated in section 6.1, we make use of a variety of Splunk's built-in visualization capabilities. The most useful has been a simple plot of Lustre OST usage over time: when a user complains of poor I/O performance, we can check this graph and see how many OSTs were actually in use at the time. If it is significantly less than 100% (1,008 per namespace) of them, then the solution is usually to ask the user to change the application code to use more. Figure 3 shows an example from March 30, 2017. The top chart shows the bandwidth and IOPs over time while the bottom chart shows the number of OSTs in use at any given time. These charts led to one of the key successes of GUIDE after it went into production in 2015. The XGC plasma simulation on Titan was using the entire machine. Since the job was the only one running on Titan at the time, its I/O patterns were easy to spot in the data from the disk controllers (Section 4.2.1). This block I/O data showed that the application was getting a write throughput of 150 GB/s. It also showed that the application was only writing to about 75% of the Lustre OSTs. We were able to debug the user's I/O configuration to use all the OSTs, which resulted in a 15% improvement in I/O performance. Without the OST usage data, improving the application's I/O performance would have likely involved several iterations of what would have amounted to educated guesses about the root cause of the problem. This, of course, would have consumed an enormous amount of core-hours on Titan.

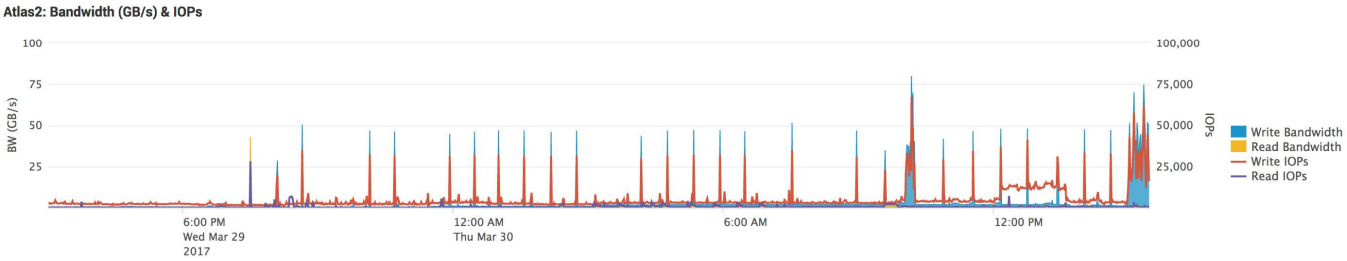
Another useful and rather enlightening set of plots has been histograms showing file size distributions on both the Atlas 1 & 2 namespaces of Spider II (Figure 4). This visualization was particularly useful recently when it was noticed that the Atlas 1 file system was running low on inodes, but had plenty of free space. A quick glance at the plots showed that there were ~ 260 million files less than 4KB in size. In total, these files were not occupying a significant amount of space, but they did each use up an inode.

6.2.4 I/O Block Sizes and Space Efficiency. For the upcoming 200 petaflop Summit system in 2018 at the OLCF, the Spider III PFS will be based on IBM's Spectrum Scale technology. Currently, Spider II's Lustre PFS delegates the block management on disk redundancy groups to a native Linux local file system, such as ext4 or ldiskfs, which uses 4KB as the file system block size. Block sizes in Spectrum Scale are a tuneable parameter, and Spider III will be configured with a much larger file system block size (i.e. 16MB or 32MB). These larger block sizes could potentially result in significantly under-utilizing the disk space.

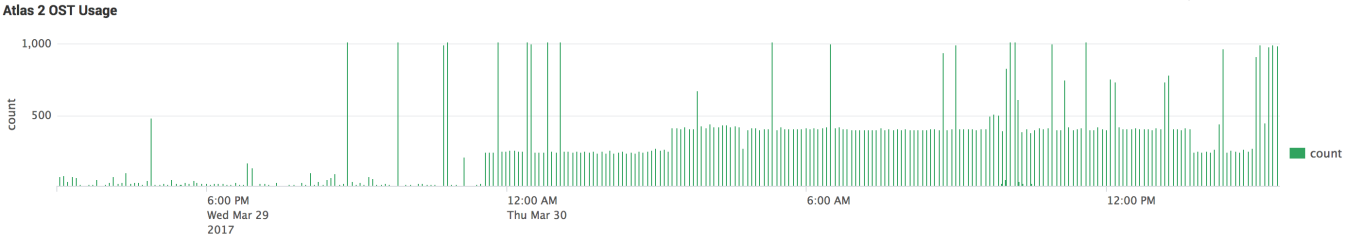
To better evaluate the wasted disk capacity vs. disk block sizes, we used the GUIDE framework. The input data stream was provided by our in-house developed, lightweight PFS profiling tool, *fprof*. We collected and aggregated each file and its size on the Spider II PFS. Assuming the file size and distribution pattern will not change drastically between systems, we were then able to predict disk space utilization efficiency under different file system block sizes even without the actual system in place. GUIDE allowed us to make these estimations easily and provided invaluable information into the design of Spider III.

6.2.5 Understanding I/O Size and Access Patterns. Analyzing the RAID I/O controller statistics data (i.e., bandwidth, IOPS, and request size distribution) can provide a deep understanding of current system behavior, which can not only help tune current operations, but help with future PFS provisioning. For example, Figure 5 shows the I/O read vs. write percentage observed on Spider II [16]. The percentage is based on the volume of data read or written. In summary, almost 70% of the workload on Spider II is from writes in comparison to 60% write in Spider I. This suggests that moving forward as the compute platform scales, users will tend to checkpoint more often, increasing the write workload on the storage system.

Figure 6 shows the I/O request size distribution on Spider II, wherein we employed statistical analysis techniques such as CDF and PDF on the I/O controller data. We can see that only 25% of the writes are 1MB requests, while over 60% of the writes are 4KB or less, which was not expected. On the other hand over 50% of the reads are 1MB, and only 20% of reads are 4KB or less. This analysis provided a valuable perspective on the size of the workloads and also in designing the next-generation Spider III PFS.



(a) Bandwidth and IOPs for March 30. Bandwidth is in GB/s



(b) OST Usage for March 30

Figure 3: Plots from Splunk that show activity on one file system.

6.3 Resiliency Use case

6.3.1 Checkpoint advisory. To protect against failures, applications often blindly checkpoint at regular intervals, e.g., 1 hr, without any consideration to the machine MTBF or the I/O rate, and consequently incur excessive I/O overhead and time away from useful computation. We have developed a checkpoint advisory tool that can suggest to applications an optimal checkpoint frequency by fusing both the RAS and storage bandwidth data in GUIDE. Key factors in determining an optimal checkpointing interval include the failure inter arrival times (computed using the RAS logs) and the time-to-write one checkpoint (derived from the storage logs using I/O periodicity analysis). The tool also takes the "job size" into account to accurately estimate the probability of a failure occurring for the given job [11].

As an example, the tool was instrumental in improving large-scale runs of the fusion application XGC. The XGC runs used over 16,000 nodes for multiple >12 hour runs, and had conservatively set the checkpointing interval to be one hour. Each checkpoint is

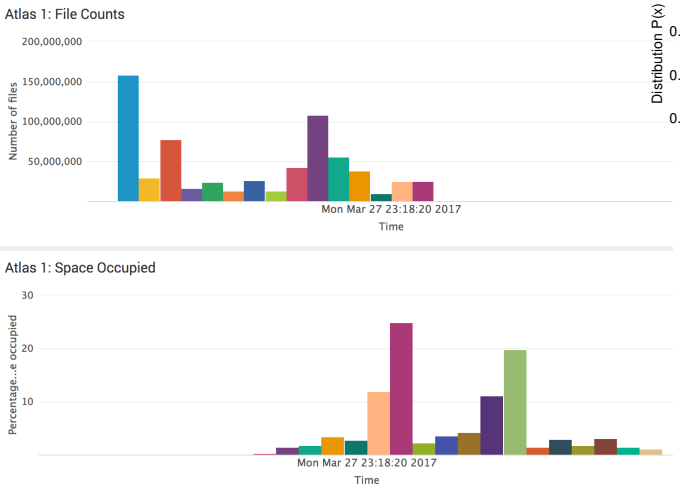


Figure 4: Plots from Splunk that show the distribution of file sizes as a function of both the total number of files and the amount of space those files occupy. Each column is a size bucket ranging from <4KB to >64TB. The blue bar on the far left shows the files that are less than 4KB.

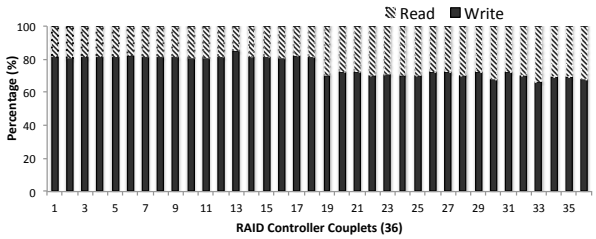
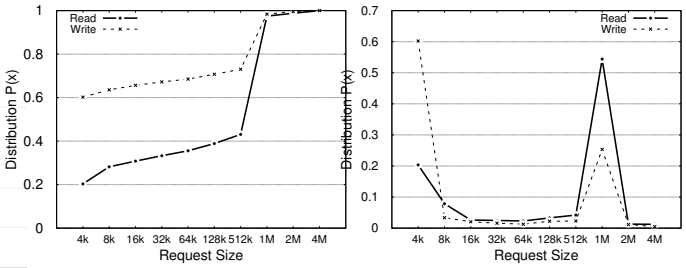


Figure 5: Read vs. Write on Spider II



(a) CDF (b) PDF

Figure 6: Distribution of Request Sizes

approximately 4TB in size, amounting to 96 TB of data for a 24-hour run. Our checkpoint advisory tool suggested that the optimal interval may be closer to 2 hours, which immediately reduced the checkpoint overhead by 50%. This effort resulted in multiple benefits. First, the amount of space required to store this data was reduced by 50% (which is close to 50TB for one single run). Second, the overall stress on the PFS became significantly less, avoiding performance degradation for other jobs. Third, this also enabled precious core hours to be utilized for computation instead of waiting on I/O operations (490,000 core hours per 24-hour run).

6.3.2 RAS Analytics. A key impact of GUIDE has been RAS analytics, which includes observing failure trends, identifying impacted jobs, users and faulty nodes, and quantifying SBEs, DBEs and off-the-bus errors on Titan's 18,688 CPUs/GPUs [10, 12, 13, 21]. Fig. 7 shows a partial RAS dashboard of these trends, depicting the progression of DBEs and off-the-bus errors on the GPUs, which cause a node failure. The graphs show how the GPU errors have grown during the lifetime of Titan from just a few per month to several per day. After a successful replacement of subsets (a few thousand) of the GPUs, it can be seen that the errors are beginning to subside. Such RAS analytics has also enabled us to identify nodes with more errors, and proactively replace those GPU cards. Further, we were able to identify that the node failure trends follow a Weibull distribution, i.e., failures have a temporal locality, which can also be used to guide checkpointing in interesting dynamic ways (e.g., checkpoint frequently after an error, but more lazily during stable period) [11]. With respect to SBEs, we found that 98% of them occur on the L2 cache on the GPU, a feedback that was provided to the vendor. Finally, we have binned the failure-affected jobs by their node-size, which helped us to understand that large jobs were more impacted (e.g., over 3,000 nodes).

6.4 Scheduling Use cases

Scheduling data (Section 4.2.3) from GUIDE has impacted operations measurably [29]. Using insights from data mining and visualization, it became apparent that the resource manager on Titan can be tweaked to better suit our workload.

Analysis of scheduling started with 3D visualizations, as shown in Figure 8, that were based on mapping job allocations from MOAB data into Titan's 3D torus network. From this visual representation we were able to quickly identify several gaps in layout that were occurring as a part of a generalized resource allocation mechanism. The importance of 3D visualization here cannot be understated. The raw logs contained all of the data to indicate that jobs were becoming highly fragmented or that jobs were being placed heavily in a low bandwidth dimension. Unfortunately, these events are often only noticed when searched for specifically. However, mapping this data visually as a real 3D representation makes these events easier to interpret.

The visual analytics lead to a study of the impact of fragmentation on application performance. In this study we determined that many applications are negatively impacted from fragmented job allocations. We used data mining techniques such as Random forest and Support vector machines to search the various attributes of a job-allocation on Titan and determine their correlation to performance impact. A result of this was the use of a new metric for tracking jobs within the OLCF called Average Hop Count. Average Hop Count is a heuristic that can be calculated with knowledge of a job's node locations within the 3D torus and this value highly correlates with performance of jobs exhibiting high dependencies upon network performance.

Ultimately, through analysis of scheduling policies and simulation, we identified that the main cause of this fragmentation was based on a scheduling policy that is used to keep utilization of Titan at high levels. This scheduling was based on a first-come first-serve (FCFS) node allocation mechanism that would select the first available nodes for job allocation. Unable to directly fix this policy without impacting important OLCF metrics, we ultimately settled on a strategy that still utilizes this scheduling policy but applies FCFS from two ends of the system node list based on requested job size. This change led to reduced average hop counts for our most common large job sizes.

6.5 Interconnection Use cases

The I/O Router interconnect congestion data (Section 4.2.4) provides accurate real-time data that correlates to the write traffic moving from a job on Titan into the PFS. This data provides insight into the locality of nodes and the amount of data being written through them. Using this information, we can identify when high-bandwidth or periodic I/O traffic occurs. Operationally, we have used the IORCD data and visualizations to extract I/O patterns, such as N:1 or N:N, from large applications to aid in our procurement when more invasive techniques like Darshan [7] were unavailable.

Extraction of I/O patterns from IORCD requires the use of organizational knowledge, auto-correlation for identifying I/O frequencies, and cross-correlation for matching backend PFS logs (Section 4.2.1). The process relies on the knowledge that Titan is comprised of twelve geographical I/O partitions, each containing software I/O routers for moving data between the Gemini and SION fabrics. We organize this data based on this geography. Each grouping is then monitored over an extended duration, applying auto-correlation to identify periodic high bandwidth activity. Figure 9 shows this behavior from a single routing group. When periodic behavior is detected across multiple groupings it becomes possible to narrow down the job creating the pattern.

Identifying the specific job is done by filtering the set of active jobs to those that could satisfy the time duration and grouping based node sets of the identified pattern. When a job is identified, we query the PFS statistics (Section 4.2.1) directly for job specific stats. These stats contain the total number of OSTs used at a specific point in time. Based on the OST usage, we make assertions regarding the checkpoint behavior being multiple files or a single shared file. Figure 10 shows the resulting OST usage from the job identified in Figure 9. This OST usage shows the job using 1,008 OSTs at each of the important checkpoint intervals. This wide use of OSTs would indicate that the job is writing multiple files as opposed to a single shared file.

6.6 Archival Storage Use cases

The HPSS usage data in GUIDE (Section 4.2.5) provides us not only with views into current operations, but also allows us to extrapolate and plan for future needs. Analysis of HPSS data involved a suite

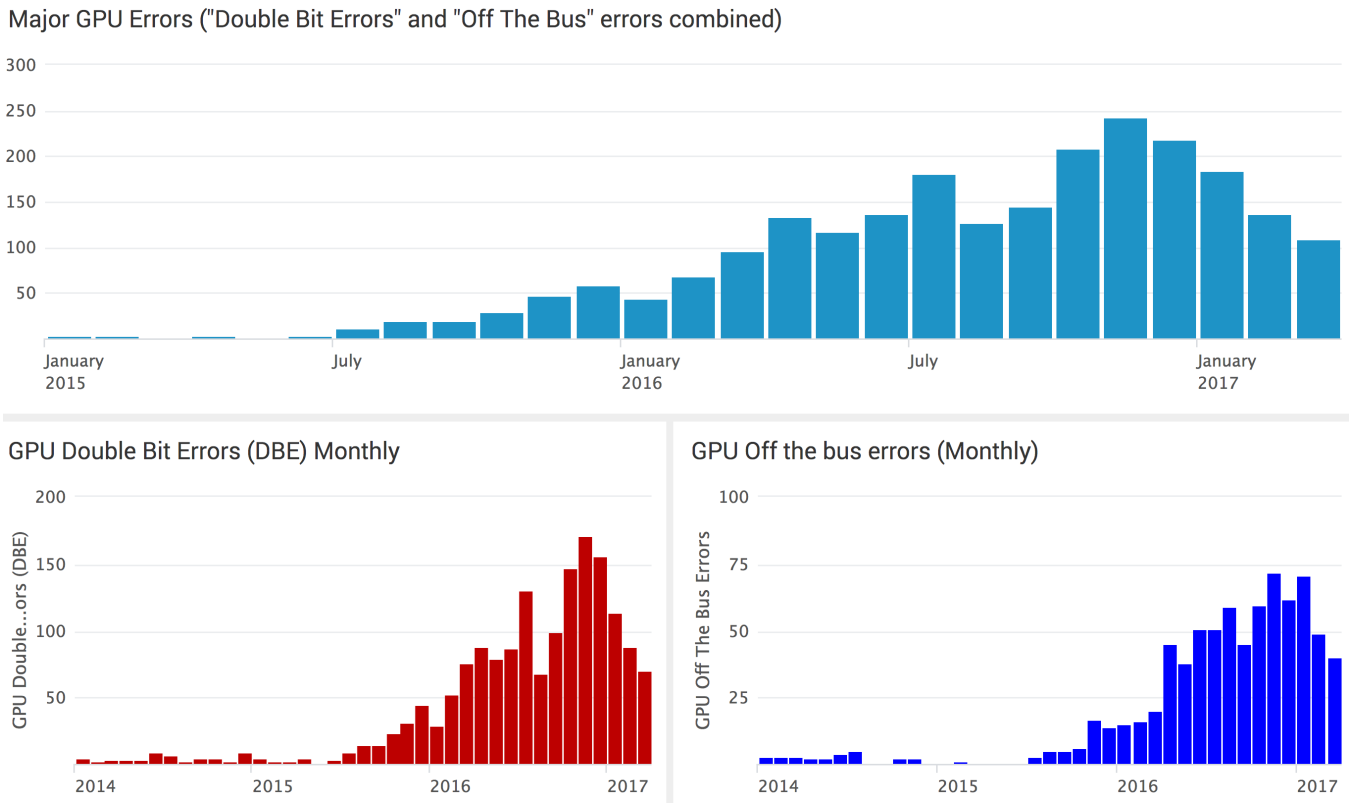


Figure 7: RAS dashboard showing different trends and failure distributions.

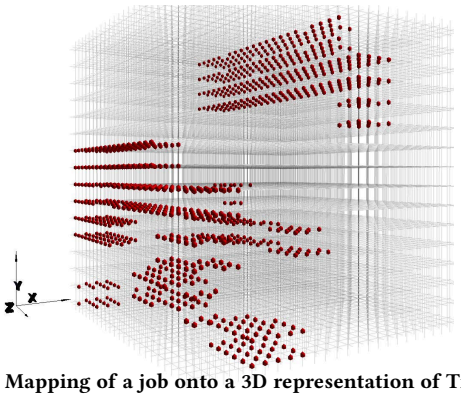


Figure 8: Mapping of a job onto a 3D representation of Titan's 3D Gemini network. This job was scheduled on to a busy system, resulting in high Y-dimensionality and significant fragmentation that could lead to performance degradation.

of statistical techniques that were outlined in Section 4.1.2. For example, the Count by File Size histograms in Figure 11 show that nearly 29 million files in the archive are 500KB or smaller and over 40 million files are less than 10MB[1], which is 44% and 61% of the files in the archive, respectively. Overall, more than 83% of the files are less than 100MB in size yet they take up less than 2% of the total volume of data stored. In summary, the HPSS software that is geared for large files, should also optimize for many small files.

[1]Sum of the first two and first four columns, respectively, which are too small to be visible on the Byte Count by File Size chart.

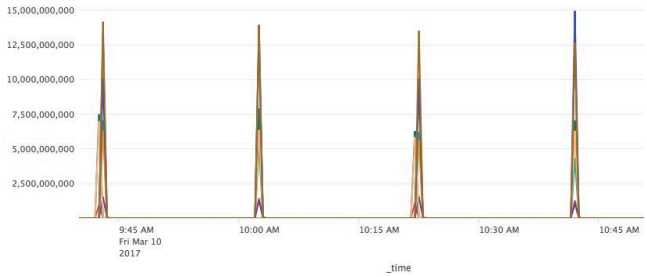


Figure 9: Time series data showing Gemini Router INQ congestion. Due to the large scale of congestion data, periodic I/O is easily differentiated from random I/O noise. This can be used to identify large

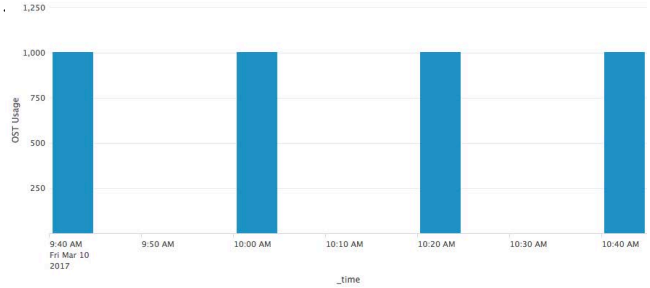


Figure 10: Lustre Job Stats: OST Usage a generally noisy data set. After identifying interesting I/O patterns and cross-correlating the job-id. Using job-stats shows the number of OSTs that were used at a given point in time. OST usage and duration can indicate N:N or N:1 write patterns.

Figure 11: HPSS File and Byte Count Histograms as of July 6, 2017. The top two charts show the space occupied and number of files for each size range while the middle two show the space occupied and number of files for each age range. The bottom two charts show the space occupied and number of files sorted by their last access time (where "0 days" means the file has never been retrieved).

In the Count by File Age histograms in Figure 11, the blue column represents files that are between 1 and 5 years old, with about 30 million files in this range–46% of the total–and that these files occupy a little over 27PB. Over the past year, users have stored roughly 6.6 million files in 10PB. [2] Extrapolating this out for 5 years, we should expect about 33 million files in 50 PB, a factor of 1.1 more files and 1.8 more data over 5 years. Users are storing the same number of files but the files appear to be larger.

Also, from the third row in Figure 11, we see that 70% of the files and 66% of the bytes have never been recalled. Once data is stored in the archive it is rarely read back, which begs the question if the archive is mainly used for disaster recovery or if the users are unable to find data due to a lack of metadata. OLCF is factoring in these insights in deciding future investments in HPSS development.

[2]Files less than one year old are calculated from the sum of all the columns to the left of the blue one.

7 RELATED WORK

GUIDE was designed as a unified data/log collection and monitoring system for a large-scale HPC center, and can be easily expanded, with different data streams fed from various collectors running on different subsystems, such as compute, storage, and interconnect. It is flexible and can be used for building higher-level data analytics tools. These set GUIDE apart from Ganglia [23] or Nagios [9]. Ganglia is a general monitoring tool with severe scaling limitations. Nagios, on the other hand, is a failure alerting mechanism.

There are a number of popular single host data collection tools, such as “sar” and “collectl”. However, to be used effectively at large-scale HPC facilities for data collection from various systems, a higher level orchestration layer must be built. There also exists a variety of distributed data collection mechanisms, such as MRNet [6, 24], LDMS [1], and TACC Stats [17]. MRNet is a tree-based overlay network for data reduction at pre-defined data aggregation nodes, such that transferring large amounts of log or monitoring data is not needed. TACC Stats and LDMS are similar in that they are both low-overhead, distributed data collection systems. LDMS is a Cray specific tool used for building OVIS [4], which is a scalable failure

and abnormal state detector. XDMoD [15] is a higher-level service built on top of TACC Stats, and provides job-specific information such as, number of jobs, used CPUs, wait and wall times.

The GUIDE framework does not compete with such single host or distributed data collection tools. In that sense, GUIDE is similar to OVIS or XDMoD. However, it is different from both, since GUIDE focuses on the entire HPC ecosystem at a large-scale HPC center, while both OVIS and XDMoD have job-centric foci.

8 LESSONS LEARNED

GUIDE has been operational for over two years in OLCF, and has provided us with numerous insights. First, it is not enough to just collect the system provided logs; there is a genuine need to extract data from every subsystem, and from as many levels of the subsystem as possible. As an example, collecting logs from the interconnection system or from the various levels of the storage system has been immensely useful in optimizing center operations and application performance.

Second, the development and deployment of scalable data extraction tools is a non-trivial task, as we need to be careful to not introduce overhead on day-to-day operations. This reiterates the need to think carefully about log data requirements at system procurement, and not as an afterthought.

Third, for the diverse OLCF subsystems, no single pre-processing technique fits all log streams. We need a combination of approaches ranging from the quantitative (statistics) to the qualitative (categorization). In addition, there is also the need for domain expertise and interactions with system administrators and vendors to cleanse the logs, fill in the missing pieces, and infer the events in the logs.

Fourth, visual analysis using GUIDE turned out to be surprisingly extremely beneficial. We realized that even just looking at the plots of the data streams in concert was very powerful, and provided numerous insights and helped us to better provision future systems (e.g., Spider III design). Further, in terms of analysis, fusing multiple streams yielded very powerful tools like the checkpoint advisory that not only saved core hours, but also reduced the I/O footprint.

ACKNOWLEDGMENTS

This work was supported by, and used the resources of, the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at ORNL, which is managed by UT Battelle, LLC for the U.S. DOE, under the contract No. DE-AC05-00OR22725.

REFERENCES

[1] Anthony Agelastos, Benjamin Allan, Jim Brandt, Paul Cassella, Jeremy Enos, Joshi Fullop, Ann Gentile, Steve Monk, Nichamon Naksinehaboon, Jeff Ogden, and others. 2014. The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications. In *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*. IEEE, 154–165.

[2] Wolfgang Barth. 2008. *Nagios: System and network monitoring*. No Starch Press.

[3] Arthur S Bland, Jack C Wells, Otis E Messer, Oscar R Hernandez, and James H Rogers. 2012. Titan: Early experience with the cray xk6 at oak ridge national laboratory. In *Proceedings of cray user group conference (CUG 2012)*. 3–4.

[4] Jim Brandt, Ann Gentile, Jackson Mayo, Philippe Pebay, Diana Roe, David Thompson, and Matthew Wong. 2009. Resource monitoring and management with OVIS to enable HPC in cloud computing environments. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 1–8.

[5] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32. DOI: https://doi.org/10.1023/A:1010933404324

[6] Michael J Brim, Luiz DeRose, Barton P Miller, Ramya Olichandran, and Philip C Roth. 2010. MRNet: A scalable infrastructure for the development of parallel tools and applications. *Cray User Group* (2010).

[7] Philip H. Carns, Robert Latham, Robert B. Ross, Kamil Iskra, Samuel Lang, and Katherine Riley. 2009. 24/7 Characterization of petascale I/O workloads. In *CLUSTER*. IEEE Computer Society, 1–10. http://dblp.uni-trier.de/db/conf/cluster/cluster2009.html#CarnsLRILR09

[8] Elastic. 2017. ELK Stack. (2017). https://www.elastic.co/products

[9] Nagios Enterprises. 2017. Nagios. (2017). https://www.nagios.org/

[10] Gupta et al. 2015. Understanding and Exploiting Spatial Properties of System Failures on Extreme-Scale HPC Systems. *International Conference on Dependable Systems and Networks (DSN)* (2015).

[11] Tiwari et al. 2014. Lazy checkpointing: Exploiting temporal locality in failures to mitigate checkpointing overheads on extreme-scale systems. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. IEEE, 25–36.

[12] Tiwari et al. 2015. Reliability Lessons Learned From GPU Experience With The Titan Supercomputer at Oak Ridge Leadership Computing Facility. *Proceedings of SC15: International Conference for High Performance Computing, Networking, Storage and Analysis* (2015).

[13] Tiwari et al. 2015. Understanding GPU errors on large-scale HPC systems and the implications for system design and operation. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 331–342.

[14] Matt Ezell. 2013. Understanding the impact of interconnect failures on system operation. In *Proceedings of Cray User Group Conference (CUG 2013)*.

[15] Thomas R Furlani, Matthew D Jones, Steven M Gallo, Andrew E Bruno, Charng-Da Lu, Amin Ghadersohi, Ryan J Gentner, Abani Patra, Robert L DeLeon, Gregor Laszewski, and others. 2013. Performance metrics and auditing framework using application kernels for high-performance computer systems. *Concurrency and Computation: Practice and Experience* 25, 7 (2013), 918–931.

[16] Raghul Gunasekaran, Sarp Oral, Jason Hill, Ross Miller, Feiyi Wang, and Dustin Leverman. 2015. Comparative I/O Workload Characterization of Two Leadership Class Storage Clusters. In *Proceedings of the 10th Parallel Data Storage Workshop (PDSW '15)*.

[17] J Hammond. 2011. TACC_stats: I/O performance monitoring for the intransigent. In *2011 Workshop for Interfaces and Architectures for Scientific Data Storage (IASDS 2011)*.

[18] InfluxData. 2017. InfluxDB. (2017). https://www.influxdata.com/

[19] Grafana Labs. 2017. Grafana. (2017). https://grafana.com/

[20] Ross Miller, Jason Hill, David A. Dillow, Raghul Gunasekaran, Galen M. Shipman, and Don Maxwell. 2010. Monitoring Tools For Large Scale Systems. In *Proceedings of Cray User Group Conference (CUG 2010)*.

[21] Bin Nie, Devesh Tiwari, Saurabh Gupta, Evgenia Smirni, and James H Rogers. 2016. A large-scale study of soft-errors on gpus in the field. In *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE, 519–530.

[22] Sarp Oral, David A Dillow, Douglas Fuller, Jason Hill, Dustin Leverman, Sudharshan S Vazhkudai, Feiyi Wang, Youngjae Kim, James Rogers, James Simmons, and others. 2013. OLCF's 1 TB/s, next-generation lustre file system. In *Proceedings of Cray User Group Conference (CUG 2013)*.

[23] Ganglia Project. 2017. Ganglia Monitoring System. (2017). http://ganglia.sourceforge.net/

[24] Philip C Roth, Dorian C Arnold, and Barton P Miller. 2003. MRNet: A software-based multicast/reduction network for scalable tools. In *Supercomputing, 2003 ACM/IEEE Conference*. IEEE, 21–21.

[25] Splunk. 2017. Splunk Enterprise. (2017). https://www.splunk.com/

[26] Vladimir N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA.

[27] Feiyi Wang, Sarp Oral, Galen Shipman, Oleg Drokin, Tom Wang, and Isaac Huang. 2009. Understanding lustre filesystem internals. *Oak Ridge National Laboratory, National Center for Computational Sciences, Tech. Rep* (2009).

[28] Richard W Watson and Robert A Coyne. 1995. The parallel I/O architecture of the high-performance storage system (HPSS). In *Mass Storage Systems, 1995.'Storage-At the Forefront of Information Infrastructures', Proceedings of the Fourteenth IEEE Symposium on*. IEEE, 27–44.

[29] Christopher Zimmer, Saurabh Gupta, Scott Atchley, Sudharshan S. Vazhkudai, and Carl Albing. 2016. A Multi-faceted Approach to Job Placement for Improved Performance on Extreme-Scale Systems. In *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1015–1025. DOI: https://doi.org/10.1109/SC.2016.86

[30] Christopher Zimmer, Saurabh Gupta, and Veronica G. Vergara Larrea. 2013. Finally, A Way to Measure Frontend I/O Performance. In *Proceedings of Cray User Group Conference (CUG 2016)*.