A Big Data Analytics Framework for HPC Log Data: Three Case Studies Using the Titan Supercomputer Log

Workshop paper: HPCMASPA 2018

Byung H. Park*, Yawei Hui*, Swen Boehm*, Rizwan A. Ashraf*, Christopher Layton†, and Christian Engelmann*
*Computer Science and Mathematics Division
†National Center for Computational Sciences
Oak Ridge National Laboratory
Oak Ridge, TN, USA
Email: {parkbh, huiy, boehms, ashrafra, laytoncc, engelmannc}@ornl.gov

***Abstract*—Reliability, availability and serviceability (RAS) logs of high performance computing (HPC) resources, when closely investigated in spatial and temporal dimensions, can provide invaluable information regarding system status, performance, and resource utilization. These data are often generated from multiple logging systems and sensors that cover many components of the system. The analysis of these data for finding persistent temporal and spatial insights faces two main difficulties: the volume of RAS logs makes manual inspection difficult and the unstructured nature and unique properties of log data produced by each subsystem adds another dimension of difficulty in identifying implicit correlation among recorded events. To address these issues, we recently developed a multi-user Big Data analytics framework for HPC log data at Oak Ridge National Laboratory (ORNL). This paper introduces three in-progress data analytics projects that leverage this framework to assess system status, mine event patterns, and study correlations between user applications and system events. We describe the motivation of each project and detail their workflows using three years of log data collected from ORNL's Titan supercomputer.**

***Index Terms*—high performance computing, Big Data applications, data analysis, event log analysis**

I. Introduction

Today's HPC systems are heavily instrumented, producing various types of data regarding system status, resource usages, and user application runs, to name a few. These data are generated by components, both hardware and software, at various layers and locations of the system, portraying different aspects of the system, such as critical conditions, faults, errors and failures. For example, a *Machine Check Exception* (MCE) is an error that can be detected by every processor in the system. MCE errors are mainly warnings or fatal exceptions, and are recorded in a system console log. Reliability, availability and serviceability (RAS) logs, on the other hand, collect data from various hardware and software sensors, such as temperature sensors and processor utilization. Network systems monitor and produce data about link status, such as link errors and congestion. Parallel files systems produce logs that record errors and performance degradation observed at both, server and client sides. For each run of a user application, a job log records the allocated resources, user information, and exit status (normal or abnormal termination).

Each log entry may seem independent. However, in many cases, log entries are generated due to the same root cause. For example, when a high speed interconnect router or switch malfunctions, different components generate event logs from their perspectives. The system module that watches over the network status starts logs events for every connection segment that is affected by the failure. Object storage servers and clients of the parallel file system report failed transactions, complaining about no responses received from peers when the file I/O request was sent through the failed router or switch. Also, the same router failure can cause user applications to generate error messages when their communication routines, such as MPI, return with an error. This easily results in hundreds of thousands of events accumulated in log data generated at different layers and parts as the root cause of the problem propagates over the entire system. Therefore, although generated individually from different perspectives, this heterogeneous collection of log data, once fused and correlated properly, can offer a holistic view of the system that facilitates efficient failure detection, error propagation tracking, and system reliability evaluation.

Analysis of log data, however, is a challenge. First, the sheer volume of logs is easily beyond what manual inspection can

handle. Second, logs are often of irregular structures and of heterogeneous types, e.g., numbers, texts, and hexadecimal codes. There exist a few commercial systems that greatly improve processing large amount of data, but their usages are mainly restricted to specific information technology (IT) products or ecosystems. Consequently, the usage of log data is, in practice, largely limited to detection of mere occurrences of known text patterns that are already known to be associated with certain types of events. With increasing complexity, especially with exascale systems on the horizon, the complexity, and the number of components and subsystems will continue to grow, generating ever bigger amounts of log data.

To address these challenges, we developed *Log processing by Spark and Cassandra-based ANalytics* (LogSCAN). LogSCAN stores large quantity of data in scalable and highly available distributed data and compute frameworks. LogScan intends to serve a wide range of researchers and engineers, providing capability of flexible extraction of different views of data and their computation. To demonstrate its value, we launched three analytic projects that each addresses an important aspect of log analysis: 1) assessment of system status, 2) mining of significant event patterns in logs, and 3) correlation analysis of application runs with system status events. Each task requires not only processing of large log data, but also integrating different log entries spatially and temporally. This paper introduces how these tasks, which are still in development, are performed utilizing the LogSCAN.

II. Related Work

The ever increasing complexity of HPC systems and IT systems in the public and private sector, along with the availability of publicly available datasets has lead to an increasing number of studies in log analysis and characterization.

A number of studies conducted post-mortem analysis on logs of large-scale systems to assess reliability of the system, extracting statistical properties of system errors and failures [1] [2] [3]. To handle voluminous logs from large-scale systems, some works applied filtering and categorizing events [4] [5] [6]. Gupta et al. [7] conducted log analyses with 5 generations of supercomputers at ORNL, including the Titan supercomputer. The study was conducted by applying scripts on data files, not utilizing a data/compute framework such as LogSCAN. Their analyses studied mean time between failure (MTBF), and temporal and spatial characteristics of failures. Oliner and Stearley [5] studied the log files of 5 HPC systems. They describe why HPC system logs are important and the obstacles the community faces trying to analyze the tremendous amount of data. They provide four recommendations to overcome these challenges. Zheng et al. [8] analyzed the RAS and job logs of Intrepid, a Blue Gene/P system at Argonne National Laboratory (ANL). The co-analysis of job and system logs revealed that high workloads do not necessarily mean high failure rates, faults usually do not propagate between jobs, and job interruptions caused by applications are usually reported early. Martino et al. [9] present LogDiver, a tool developed to analyze the log data produced by Blue Waters, a Cray XE/XK machine at the University of Illinois. Sîrbu and Babaoglu [10] studied the system logs of the Fermi supercomputer. They integrated the data of four different subsystems (power, temperature, workload, and hardware/software events). They characterized the Blue Gene/Q system's thermal, power, workload, and event logs, and conducted correlation studies among the metrics. Li et al. [11] present FLAP, an integrated log analysis platform, for end-to-end analysis and system management. They show that an integrated analysis system can greatly improve productivity of routine analysis tasks of system administrators. They further show that utilizing data mining techniques improve the ability for system administrators to gain insights into monitored systems, and can quickly pinpoint root causes of failures using temporal event analysis. The work by Park et al. [12] introduces the log analytics framework used for the study presented in this paper. The framework utilizes a scalable Cassandra database cluster to store the log data for efficient retrieval by Apache Spark which is used to carry out data analytics.

III. Architecture of the LogSCAN and Data

This section briefly describes the architecture of LogSCAN and the structure of Titan log data used for this study.

A. LogSCAN

LogSCAN is implemented in the private cloud of Compute and Data Environment for Science (CADES) at ORNL. The CADES private cloud is a +3000-core Broadwell and Haswell Xeon-based compute environment that is open to researchers and their collaborators. It is set up to accommodate many types of scientific workflows with fast solid-state drive (SSD) block storage, commodity spinning disks for large storage needs, 10Gb bonded Mellanox Ethernet networking, and large memory nodes. Compared to a public cloud, it offers *data locality*. The data lives closer to fast, large, low-latency storage for both, general use and archiving. LogSCAN is a dedicated project of the CADES cloud that allocates 256 cores, 328GB of RAM, and 1TB of block storage.

The main components of LogSCAN are the backend database and the Big Data processing unit, which are implemented with Cassandra [13] and Spark [14], respectively. It also consists of a Web server and a query processing engine. The user queries are received by the Web server, translated by the query engine, and either forwarded to the back-end database, or the Big Data processing unit, depending on the type of a user query. Simple queries are directly handled by the query engine, and complex queries are passed to the Big Data processing unit.

LogSCAN serve simultaneous queries from multiple users, who may also require long-lived connections. For this, LogSCAN adopted Tornado framework [15], which supports long-living asynchronous connections through non-blocking I/O, *long-polling* and *WebSockets*. The Cassandra database and the Apache Spark cluster are installed over the same virtual machine (VM) instances in CADES to maximize data locality. The backend database is designed to store redundant data

tables that each is *partitioned* (or equivalently indexed) by a specific attribute such as hour, user, application, location, and event type to maximize query responses. All date in each table are time-ordered to facilitate time-series analytics. For details of LogSCAN, please refer to [12].

B. Titan Log Data

Titan logs produced between January 2015 to March 2018 are populated into LogSCAN and used for the studies. The Titan supercomputer is hosted at the Oak Ridge Leadership Computing Facility (OLCF). It is a heterogeneous Cray system with 18,688 16-core AMD Opteron CPUs, 18,688 NVIDIA Tesla K20X GPUs, a Lustre filesystem, and a 3D torus interconnect network. The Titan logs contain three prominent - *console*, *consumer*, and *netwatch* logs from the Cray logging system. A specific event parser is applied to each logging category, and the processed records are transformed into events which are assigned to one of 22 event types according to their logging nature. For example, certain hardware failures could occur in different areas of the system and the associated events could be recorded and parsed as of two distinctive event types - "MCE" and "HWERR". The total occurrence of each event type during the period is summarized in Table I.

Titan's nodes are organized in a multi-dimensional grid which could be laid out in form of "Cabinet by Chassis by Slot by Node/Gemini router." According to the nodal naming convention of the Titan architecture, the physical sources which record the events could be represented in either of two formats. The first one (e.g., "c16-3c0s4n1") shows that the recorded event occurred at "Cabinet, Row 16 by Column 3, Chassis 0, Slot 4 and Node 1," while the second source format (e.g., "c12-2c1s2g0") shows the event was recorded at "Cabinet, Row 12 by Col 2, Chassis 1, Slot 2 and Gemini 0." In the entire analysis which is presented in this study, all events recorded on nodes whose source names comply with the second naming convention are excluded.

Usually the scope of the event source is well constrained by Titan's architecture. There are i) 25 rows by 8 columns of machine cabinets; ii) 3 chassis in each cabinet; iii) 8 slots in each chassis; and iv) 4 nodes in each slot. However, during the parsing process, some events are identified with "unconventional" source names such as "c231-1c2s0n1" or "c8-5c0s6n16". Even though they are extremely rare (less than 10 in almost two hundred million), we deem them as valuable indicators of how well the logging system is working at OLCF and worth further investigation.

Counting from a start time at "2015-01-27 11:19:54" to the end of log records at "2018-03-09 14:11:42," the total number of event records are close to 187 million. Among them, five event types - "Lustre Error", "Lustre", "LNet", "HWERR" and "Lanemask Change" - dominate with their total counts (in millions) of 55.7, 48.6, 35.1, 18.2 and 16.8, respectively. All other event types accumulate to numbers of order of magnitudes lower. In Figure 1, we show the total event counts accumulated in each day. A moderate time resolution of a day is chosen to collect the total number of counts at any given time. In other words, the count number plotted in Figure 1 is the total counts recorded on all Titan nodes within the past hour prior to the given moment. Note the event counts of year 2016 is shown only. Other years show a similar trend.

Counts	Percentage	ID	Description
11	0	1	DVS Confusion
3,040,934	1.6	2	NVRM Xid
5,797,465	3.1	3	Machine Check Exception (MCE)
1,237	0	4	NVRM DBE
56	0	5	Unknown GPU Error (UGE)
303,149	0.2	6	Graphics Engine Error (GEE)
5,772	0	7	Kernel Panic
783,599	0.4	8	Out of Memory (OOM)
18,181,137	9.7	9	HWERR
1,627,599	0.9	10	Seg. Fault
48,584,034	26	11	Lustre
35,112,479	18.8	12	LNet
1,008,193	0.5	13	LNet Error
16,829,223	9	14	Lanemask Change
116	0	15	Netwatch ERROR
55,701,721	29.8	22	Lustre Error
186,976,725	100		

TABLE I: 22 Event types and their occurrences in Titan's logs between January 2015 to March 2018

IV. Analytic Models

This section describes three projects and their initial investigation that leverage LogSCAN for analytics. Each workflow was developed as Jupyter notebooks that interactively perform analytics through the Spark job server of LogSCAN.

A. Case study 1: Metric to denote system status

From the perspective of pure numeric analysis, the system status could be represented as a multi-dimensional data set. In its most crude manifestation, we actually have a 3D data set which possesses two fixed dimensions along the directions of event source (19200 distinctive nodes) and type (22 distinctive event types). To come up with a quantifiable low-dimensional metric which incorporates as much system information as possible without a high-dimensional representation, we seek help from two widely used algorithms: Principal Component Analysis (PCA) and Information Entropy.

1) PCA Analysis: PCA as a statistical procedure was first developed by Pearson in 1901 [16]. As a standard machine learning algorithm, PCA has been widely used in data preprocessing for the purpose of dimensionality reduction. The general application of the algorithm lies in finding the so-called principal components (PCs) in a high-dimensional feature space. These PCs are essentially a new set of features (mathematically speaking, a linear combination of the originals) which could best represent the original features by maximizing the feature variance along the directions of identified PCs.

In our study, we are interested in the relative variances among all PCs which could reveal the overall degree of feature correlation. The argument could be more easily understood by investigating a toy model which has a set of observations

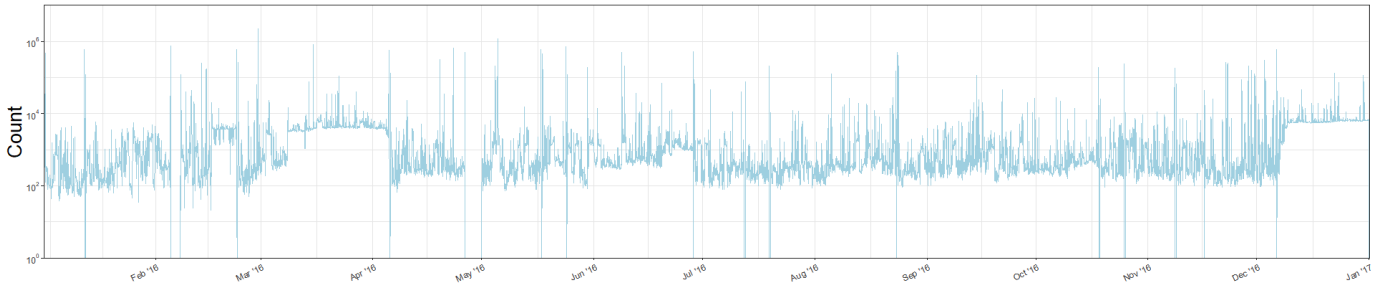

Fig. 1: The total event counts accumulated in each day (plot of year 2016 is shown only)

on two features. If a strong correlation exists between the two features (like a linear type), it would be obvious to see that one of the principal component totally dominates the other. In the case of multi-variable analysis as we are facing here for the high-dimensional event table, we could apply a certain standard matrix decomposition procedure on the covariance matrix (e.g., Singular Value Decomposition, SVD, as we used in our study) to obtain all feature variances along the principal components, and then define a probability vector for the variances (Equation 1), where σ_i (i from 1 to k) is the i-th variance calculated from k eigenvalues of SVD decomposition.

$$\xi_i = \frac{\sigma_i}{\sum_1^k \sigma_i} \tag{1}$$

The distribution of the variance probability $\vec{\xi}$ may change dramatically according to the system history prior to any given moment. Two extremes among these variations deserve a closer scrutiny because they represent two most distinctive system statuses. The first is a distribution with a single variance dominating all others, which simply means all observed features are highly correlated and could imply they share a single source of driven force (single user, large-scale application, etc.). The other extreme case of probability distribution is one that has evenly distributed variances along all PCs. On the contrary to the first case, this even probability distribution means that each feature is driven by an independent source (multiple users, applications utilizing different resources, etc.).

Looking back at our original problem, which is "to find a quantifiable low-dimensional metric incorporating as much system information as possible without a high-dimensional representation", we see that, by applying PCA on the 3D event table at any given time, the data set is compressed to a 2D matrix which has the original dimension of time while compressing the other two (representing the source and type of the recorded events) into a probability vector of PC variances. In fact, since the PCA algorithm doesn't dictate any specifics about how one arranges the data set in format of records vs. features, we created a second layout from the original event table by re-arranging the table in the following way.

The original event table has, for each event type at each row, the event source and the total counts accumulated in the 1-hour time window. As described in Section III-B, the source name complies with the Titan nodal naming convention and can be mapped to a set of coordinates in the multi-dimensional grid reflecting the Titan architecture. Given a source "c16-3c0s4n1," for example, it directly reads that the events occurred at "Cabinet, Row 16 by Column 3, Chassis 0, Slot 4 and Node 1." Instead of using each single coordinate as a new feature (it would be rather interesting to explore the data set in such an extended way, though), we transformed the set of grid coordinates ($Cabinet\_Row$, $Cabinet\_Column$, $Chassis$, $Slot$, $Node$) into a pair of X and Y coordinates in a 2D layout map. Speaking specifically, we have used

$$\begin{aligned} X &= 12 * Cabinet\_Row + 4 * Chassis + Node \\ Y &= 8 * Cabinet\_Column + Slot \end{aligned} \tag{2}$$

in this study to mimic the floor map of Titan. By no means, this particular schema of nodal mapping is the only (or "right") choice among many linear combinations of the grid coordinates. After the transformation, the event table, at a given time spot, has two new dimensions of X and Y in place of the original event source and type. The cell value at [X, Y] is filled in with its window-accumulated counts for either a single event type or several types combined. If all event types combined, we simply have a 2D nodal map for the total counts. Referring back to Section III-B about the Titan architecture which gives 25 rows and 8 columns of cabinets, 3 chassis per cabinet, 8 slots per chassis and 4 nodes per slot, we know the dimensionality of the new 2D layout (referred to as "Nodal_Map," hereafter) is fixed at 300 in X's direction and 64 in Y's, which is quite different from the original layout (referred to as "Source_Type", hereafter), as "Source_Type" has a varying dimensionality along time.

Whether the PCA algorithm is applied on "Source_Type" or "Nodal_Map," the resulting probability vector (Equation 1) is still not concise enough to be used for a quantifiable low-dimensional metric. To store the system status in a more compact way, we go to our final step of information compression.

2) Information Entropy: Information Entropy (Shannon entropy) is, in general, a numeric indicator of the average amount of non-redundant system information. It was first developed by Shannon in his seminal paper for the information theory [17]. In our study, we use the concept of information entropy to mathematically quantify the information content of a random signal with a certain probability distribution. Specif-

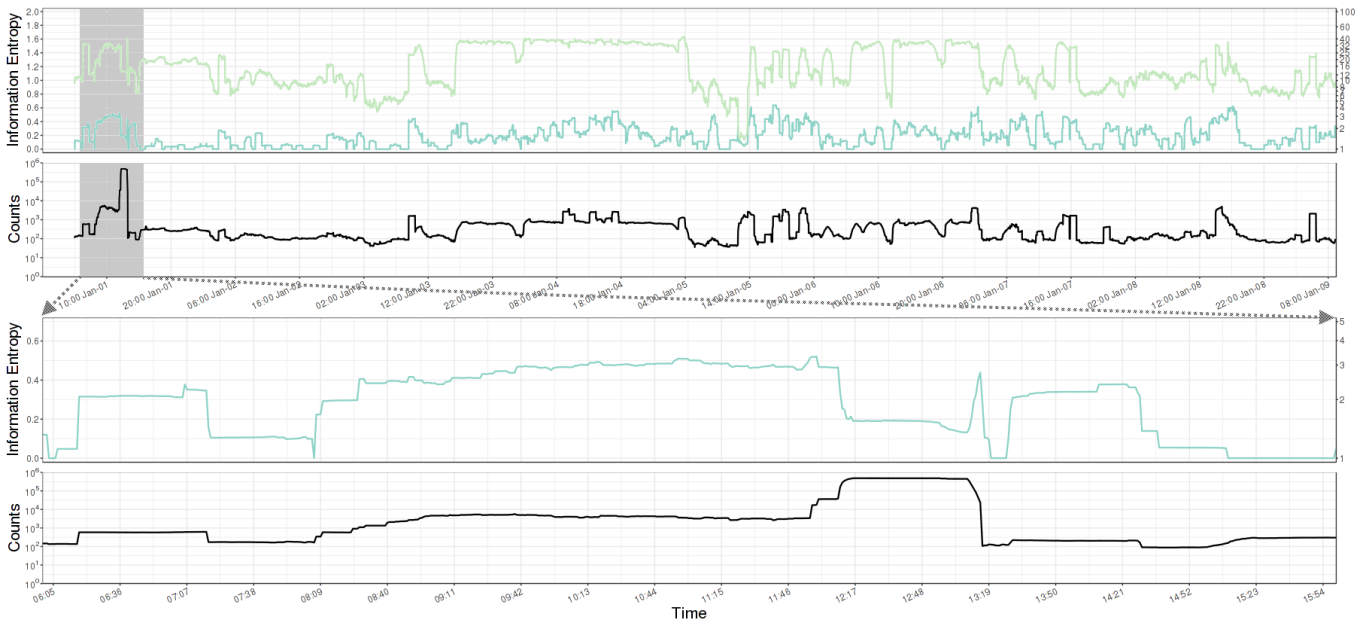

Fig. 2: **Upper Panel** (two plots): Historical panorama of the Titan system status illustrated with three time series – information entropies (H) for the "Source_Type" and "Nodal_Map" layouts and **"total event counts"** – from Jan. 1st to 9th of Year 2016. **Bottom Panel** (two plots): Close-up of the information entropy of "Source_Type" and the corresponding **"total event counts"** for a small time period on Jan. 1st, 2016. Plots showing information entropies have a second "y"-axis to their right where the values of 10^H are marked accordingly. All plots share the same time resolution of one minute.

ically, we use the variance probability vector $\vec{\xi}$ calculated via PCA on layouts of the event table (either "Source_Type" or "Nodal_Map") and define the Information Entropy, H, by applying the Boltzmann's formula:

$$H = -\sum_{1}^{k} \xi_i log(\xi_i) \tag{3}$$

By computing H for each event table (no matter what layout it takes) we further compress the system status along time and use this time series as the quantifiable low-dimensional metric that we are searching for. In practice, we are also interested in the value of 10^H. Briefly going back to the two extreme cases discussed about the variance probability distribution in Section IV-A1, we see in the first case of single variance dominance, the information entropy would tend to be zero and it results in $10^H \approx 1$. While when variances are distributed evenly along all PCs, the information entropy gets close to its maximum value of $log(k)$ (k is the number of features taken into account in PCA) and apparently $10^H \approx k$.

For illustration purposes, we choose a small section of the historical panorama demonstrated in Fig. 2 which represents the system status of Titan supercomputer from "2016-01-01" to "2016-01-09." Both layouts are considered in plotting the information entropy. The two plots on the upper panel show "Source_Type" and "Nodal_Map" respond accordingly to the total event counts at large scale, while differences between either one of them and the total counts are unmistakably standing out at many medium to small time scales. For example, the overall system status started with a few "violent" jumps showing on all three curves prior to "2016-01-01 15:00," where "Source_Type" and "Nodal_Map" share a similar profile which is much less extreme comparing to the total counts. Another example can be seen during the common "plateau" which spans one and a half days (from "2016-01-03 16:00" to "2016-01-05 05:30"). While both "Source_Type" and "Nodal_Map" registered similar dips in values as did the total counts in between "2016-01-04 01:40" and "2016-01-04 02:40", at a later time, only "Source_Type" responded significantly to the total counts while "Nodal_Map" remained docile from "2016-01-04 08:50" to "2016-01-04 17:40."

Much closer scrutiny has to be put to order to explain these observations of system behaviour and an example is given by the two close-ups on the lower panel of Fig. 2. Surprisingly, the time evolution of the two series differs quite a lot, esp. between the hours of "12:00" and "13:30". Many similar behavioral deviations existing among information entropies and the total events could be attributed to varying contributions of events to the statistics and much more detailed discussion has been reported in a separate article. As a robust system status metric, however, we see that the information entropy sensitively responds to varieties of driven forces in the system which shape the general behavior along time in a comprehensive manner, esp. as the dimensionality of the feature space increases with the complexity of the system in question.

B. Case study 2: Pattern Mining

Inference of interesting relations between event types helps understanding or capturing system status. One such approach is association rule mining [18]. The most computationally intensive part of association rule mining is to identify frequent event sets in the data. We define a set of events to be frequent if they tend to co-occur within the same time window. For this study, we slide a time window and generate data sets of event occurrences in each window. The interval and time steps by which the window is moved need to be chosen carefully to capture an accurate representation of the time series.

Frequent event sets can be captured from various aspects. If we do not consider any spatial coherence, all the events that occurred within a time window are considered for frequent sets regardless of locations where they are generated. On the other hand, with spatial coherence as a constraint, events that occurred on the same node, slots, chassis, or cabinets are considered for frequent event sets. Fig. 3 gives a pictorial illustration of the procedures. The set, created with or without spatial constraint, is then used for mining association rules.

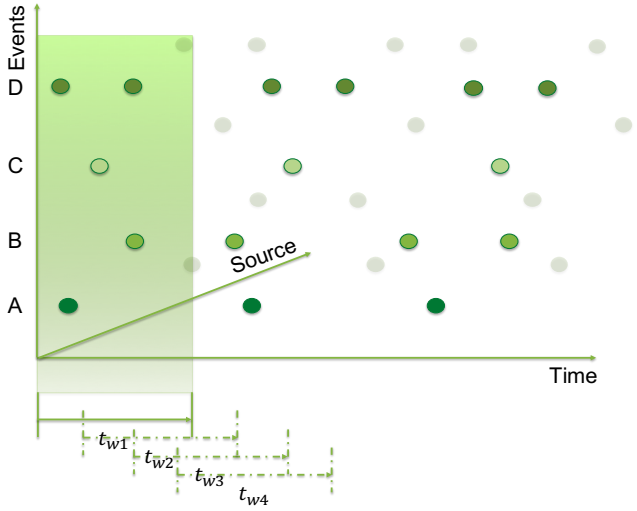

Fig. 3: Events by month

For this study, we considered events present in the data of 2016. The log events are dominated by file system events (*lustre* with event ID 11 and *LNet* with ID 12, 33% and 37% of the data) followed by hardware events (*HWERR* with ID 9 and 15% of the data). Upon close examination, we found that the file system and hardware events can be further categorized into 155 and 66 sub event types, respectively. We are currently investigating pattern inference for these sub event types.

We first computed correlations of all pairs of 22 events, which is shown in Table II. The intent was to see whether two events in a frequent event set are correlated. We also examined any recurrent temporal patterns. Fig. 4 is one result that shows the average number of events generated in each day of the week. As one would expect, the number of events are higher throughout the work days and slightly less during the weekends. The exceptionally high number of events on Tuesdays is interesting. One possible explanation is that a lot of new jobs are added to the batch job queues on Mondays. Another possible explanation is that maintenance for OLCF systems is usually scheduled on Tuesday. Shutting down Titan might be an explanation for above average system events.

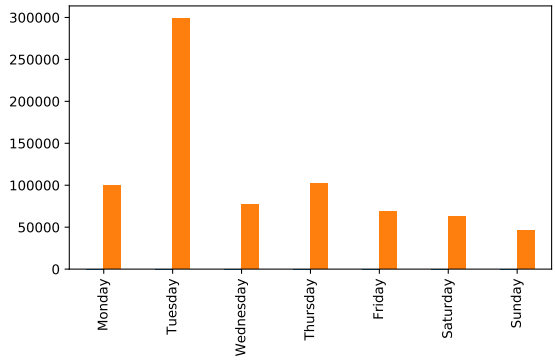

Fig. 4: Average number of log events by days of the week

We applied the FPGrowth algorithm [19] of the Spark *mllib* library to generate frequent event sets. Again, we used the events of 2016 for this analysis. The support threshold was set to 0.1 with the window size of 24 hours. Some notable frequent events are illustrated in Table III. It is not surprising to find event types *LNet* (ID 12), *LNet Error* (ID 13) and *Lustre Error* (ID 22) dominating the frequent event sets. Interestingly, however, none of the frequent event sets contain event type of *kernel panic* (ID 7). This is somewhat unexpected since the statistical analysis suggested a strong correlation between event *Lustre Error* and *Kernel Panic*, that is between 22 and 7 in Table II. One possible explanation for this is that the *kernel panic* events occur frequently, but their occurrences are confined to a small set of nodes. Further analysis of the data set is necessary to confirm this assumption.

Mining frequent pattern on the dataset is still a work in progress. We only reported a few findings that were generated from data that spans a month or less. Therefore the results listed in Table III are not conclusive.

Another approach to infer interesting patterns from data is *word embedding*, which we are currently investigating. In text mining, this feature embedding algorithm known as word2vec [20], transforms each word in a vocabulary into a high dimensional vector space where semantically related words in a corpus of documents are positioned close to each other. When applied to log event data, word2vec will identify event types that appear in vicinity within a time window. This essentially enables clustering of event types in the transformed vector space, thus producing patterns whose occurrences are dominant during a period of interest. We currently plan to apply word2vec to 155 file system related sub event types to infer notable event patterns in a much higher resolution.

id	1	2	3	4	5	6	7	8	9	10	11	12	13	14	22
1	1.0	-0.0	-0.0	-0.1	-0.0	-0.0	-0.0	-0.0	-0.0	0.0	-0.0	0.0	0.1	-0.0	-0.0
2	-0.0	1.0	0.1	-0.0	-0.0	0.1	-0.0	-0.0	0.0	-0.0	0.0	0.1	0.0	-0.0	-0.0
3	-0.0	0.1	1.0	-0.0	0.1	0.0	-0.1	-0.0	0.0	0.0	-0.1	-0.0	-0.0	-0.0	-0.1
4	-0.1	-0.0	-0.0	1.0	-0.0	0.0	0.0	0.1	-0.0	-0.0	0.1	0.1	0.0	-0.1	0.1
5	-0.0	-0.0	0.1	-0.0	1.0	-0.0	-0.0	-0.1	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0
6	-0.	0.1	0.0	0.0	-0.0	1.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.1	-0.1	-0.0	-0.0
7	-0.0	-0.0	-0.1	0.0	-0.0	-0.0	1.0	-0.0	-0.0	-0.0	0.4	0.0	0.1	-0.0	0.7
8	-0.0	-0.0	-0.0	0.1	-0.1	-0.0	-0.0	1.0	-0.0	-0.0	-0.1	-0.0	0.0	-0.1	-0.0
9	-0.0	0.0	0.0	-0.0	-0.0	-0.0	-0.0	-0.0	1.0	-0.0	0.1	0.1	0.0	-0.0	0.0
10	0.0	-0.0	0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	1.0	-0.0	-0.0	0.0	-0.0	0.1
11	-0.0	0.0	-0.1	0.1	-0.0	-0.0	0.4	-0.1	0.1	-0.0	1.0	0.5	0.1	0.0	0.4
12	0.0	0.1	-0.0	0.1	-0.0	-0.1	0.0	-0.0	0.1	-0.0	0.5	1.0	0.3	-0.0	0.1
13	0.1	0.0	-0.0	0.0	-0.0	-0.1	0.1	0.0	0.0	0.0	0.1	0.3	1.0	-0.1	0.1
14	-0.0	-0.0	-0.0	-0.1	-0.0	-0.0	-0.0	-0.1	-0.0	-0.0	0.0	-0.0	-0.1	1.0	-0.0
22	-0.0	-0.0	-0.1	0.1	-0.0	-0.0	0.7	-0.0	0.0	0.1	0.4	0.1	0.1	-0.0	1.0

TABLE II: Correlation matrix for the different events

items	frequency
12, 11	56741
3, 12, 11	3813
22, 12, 11	18481
10, 11	2859
9, 12	2693
2, 9	8501

TABLE III: Frequent events for June 2016 (support = 0.01)

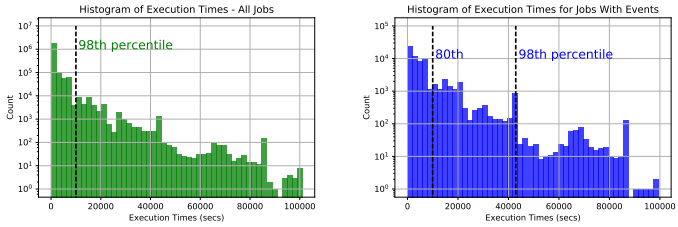

Fig. 5: The histograms of execution times (with a logarithmic y-axis) of all jobs and jobs which only had events. The comparison of the two plots demonstrates an increased likelihood of event occurrence for lengthier job executions.

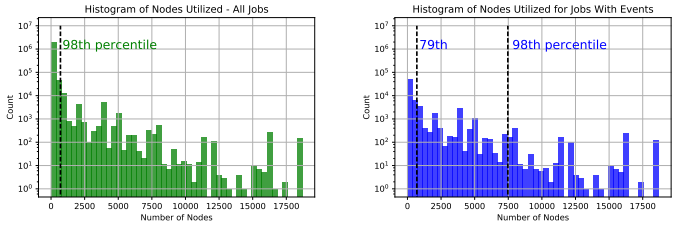

Fig. 6: The histograms of number of nodes used (with a logarithmic y-axis) by all jobs and jobs which only had events. The comparison of the two plots demonstrates an increased likelihood of event occurrence for larger sized jobs.

C. Case Study 3: Correlation analysis of jobs executions and event occurrences

LogSCAN provides the ability to do analyses on log data generated from multiple sources. In addition to event data as described earlier, our database is also able to ingest data from the Cray Application Level Placement Scheduler (ALPS) infrastructure. ALPS records information such as the binary name of each executed job, the start/end date/time of each job, and the nodes on which each job was scheduled. In this case study, we analyze the characteristics of jobs which have events recorded during their executions by correlating data from the events database (Table I) with data from the application placement and scheduling database. Furthermore, we present preliminary analysis to understand the impact of multiple types of events on the execution times of jobs which is beneficial for highlighting system components responsible for performance variations observed in large-scale systems.

We have about one year worth of data from the ALPS infrastructure, i.e., from 2015-12-24 13:00 to 2017-02-13 17:00. Using the events database stored in Cassandra, we intent to extract job executions from the application database (also stored in Cassandra) during which events are recorded. The applications database contains space and time information of the jobs, whereas the events database contains space and time information of the events. The two databases are conditionally joined via SQL style queries in Cassandra through the Spark interface. The join conditions check for each application the conditions that an event occurring on any of the nodes on which the application is scheduled also occurs within the start and end date/time of the application. A limitation of this study is that we are not able to correlate network related events IDs 14 and 15 in Table I with any application execution, since these events are recorded on the Gemini routers in Titan. We will investigate this further in future work.

After the conditional join between the jobs and events databases, we are able to obtain a database which contains job executions with event occurrences. We find that about 68 thousand jobs (3.24%) out of 2.1 million total jobs (within our analysis period) had events recorded during their execution. We further distinguish the characteristics of these jobs by plotting the histograms of execution times (Fig. 5) and number of nodes (Fig. 6) used for these jobs and comparing them to corresponding histograms of all job executions irrespective

of whether an event takes place or not. We find that job executions with events tend to be lengthier in time and larger in size. For example, the 98th percentile for job execution times is shifted from 10,000 seconds to over 42,000 seconds when events take place during executions. Similarly, the 98th percentile for job sizes is shifted from 700 nodes to 7500 nodes when events take place during executions. In part, the histograms explain the reason for low percentage of jobs which have recorded events.

We conduct correlation analysis to determine whether long running and large sized jobs produce more events or whether increased number of events lead to increased execution times as a result of the overhead of the logging infrastructure. Specifically, we calculate the correlation coefficients between the number of events recorded in each job, and job execution times, job sizes and core-hours to reflect both, the length and size of jobs. We further segregate our analysis into multiple event classes based on different components of the system. This allows us to see how application execution times may be impacted by different event classes or alternatively, explore the possibility of using job length or size as an indicator of observing events from a certain class. We categorize events into following broad categories: (1) all Lustre related events (IDs 1, 11, 13 and 22) excluding Lustre network status event (ID 12), (2) all GPU events (IDs 2, 4 and 5), (3) hardware and software related events on processors excluding MCEs (IDs 6, 7 and 9), (4) machine check exceptions (ID 3), and, (5) pure software issues on processors including segmentation faults and out of memory errors (IDs 8 and 10). Note, in some cases, a combination of different event categories is observed which is not segregated in our analysis. The obtained Spearman correlation coefficients for each case are listed in Table IV. An example plot of number of core-hours vs. number of events observed for pure software events is shown in Fig. 7.

The results in Table IV show that there is very small correlation between the number of all events class and both execution times and job sizes. Similarly, Lustre events show very weak correlation with both length and size of jobs. On the other hand, GPU events, processor hardware and software events, and pure software events seem to be moderately correlated with the size of jobs (coefficient greater than 0.5). Similarly, GPU events and processor hardware and software events, and MCE events seem to have low correlation with execution times of applications (coefficients between 0.3 and 0.5). In summary, the preliminary analysis did not find any single dominant event class which may have a significant impact on application execution times. In other words, increase in number of events does not imply an increase in execution time, which shows that the logging infrastructure does not add overheads to the execution of the jobs. Additionally, job sizes or lengths are not strong indicators for correlating occurrence of events to different system components. In future work, we plan to use LogSCAN to perform a much focused analysis by quantifying performance variations across application executions of same types and determining correlations with different event classes.

TABLE IV: Spearman correlation coefficients between various event classes and job execution times, sizes, and core-hours

Event Class	Execution Time	Job Size	Core-hours
All Events	0.249	0.129	0.310
Lustre Events	0.236	0.285	0.354
MCE Events	0.297	0.258	0.480
Processor HW/SW	0.440	0.523	0.599
Pure SW Events	0.272	0.548	0.570
GPU Events	0.476	0.526	0.614

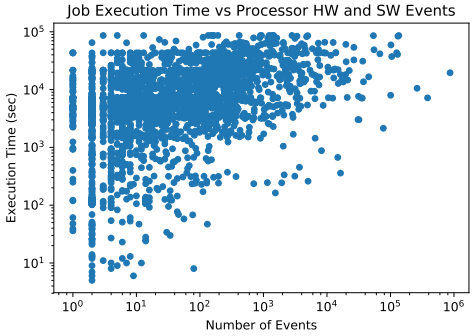

Fig. 7: Plot of execution times of jobs with number of Processor HW/SW events illustrating that a higher event count does not strongly imply an increase in execution time of jobs.

V. Conclusion and Future Works

This paper introduces how LogSCAN, a Big Data processing framework for HPC logs, is utilized by three analytic case studies. The tasks of these case studies require multiple scans of the entire database, creating a series of input data for subsequent analytics and visualization. This paper thus demonstrates the compute and memory intensive computations for the designated analytics can be performed efficiently by leveraging distributed NoSQL database technology and the MapReduce paradigm. However, these tasks are still under development, thus the analytic results and the original logs need to be further evaluated and studied.

We plan to extend LogSCAN in several directions. First, a new design to support analysis on raw logs is under development. This will particularly facilitate identification of new event types by applying text mining algorithms to the original raw logs. Second, incorporating GPU-enabled compute nodes and a deep learning framework, such as *Deep Water*, the AI extension to H_2O [21], into the LogSCAN is under review.

Acknowledgment

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, program manager Lucy Nowell, under contract number DE-AC05-00OR22725. This work was supported by the Compute and Data Environment for Science (CADES) facility and the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is managed by UT Battelle, LLC for the U.S. DOE (under the contract No. DE-AC05-00OR22725)

REFERENCES

[1] C. D. Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer, "Lessons learned from the analysis of system failures at petascale: The case of blue waters," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2014, pp. 610–621.

[2] R. K. Sahoo, M. S. Squillante, A. Sivasubramaniam, and Y. Zhang, "Failure data analysis of a large-scale heterogeneous server environment," in *International Conference on Dependable Systems and Networks, 2004*, June 2004, pp. 772–781.

[3] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 4, pp. 337–350, Oct 2010.

[4] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. K. Sahoo, J. Moreira, and M. Gupta, "Filtering failure logs for a Blue Gene/L prototype," in *2005 International Conference on Dependable Systems and Networks (DSN'05)*, June 2005, pp. 476–485.

[5] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, June 2007, pp. 575–584.

[6] A. Pecchia, D. Cotroneo, Z. Kalbarczyk, and R. K. Iyer, "Improving log-based field failure data analysis of multi-node computing systems," in *2011 IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN)*, June 2011, pp. 97–108.

[7] S. Gupta, T. Patel, C. Engelmann, and D. Tiwari, "Failures in large scale systems: Long-term measurement, analysis, and implications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: ACM, 2017, pp. 44:1–44:12. [Online]. Available: http://doi.acm.org/10.1145/3126908.3126937

[8] Z. Zheng, L. Yu, W. Tang, Z. Lan, R. Gupta, N. Desai, S. Coghlan, and D. Buettner, "Co-analysis of RAS log and job log on Blue Gene/P," in *2011 IEEE International Parallel Distributed Processing Symposium*, May 2011, pp. 840–851.

[9] C. D. Martino, S. Jha, W. Kramer, Z. Kalbarczyk, and R. K. Iyer, "LogDiver: A tool for measuring resilience of extreme-scale systems and applications," in *Proceedings of the 5th Workshop on Fault Tolerance for HPC at eXtreme Scale*, ser. FTXS '15. New York, NY, USA: ACM, 2015, pp. 11–18. [Online]. Available: http://doi.acm.org/10.1145/2751504.2751511

[10] A. Sîrbu and O. Babaoglu, "A holistic approach to log data analysis in high-performance computing systems: The case of IBM Blue Gene/Q," in *Euro-Par 2015: Parallel Processing Workshops*, S. Hunold, A. Costan, D. Giménez, A. Iosup, L. Ricci, M. E. Gómez Requena, V. Scarano, A. L. Varbanescu, S. L. Scott, S. Lankes, J. Weidendorfer, and M. Alexander, Eds. Cham: Springer International Publishing, 2015, pp. 631–643.

[11] T. Li, Y. Jiang, C. Zeng, B. Xia, Z. Liu, W. Zhou, X. Zhu, W. Wang, L. Zhang, J. Wu, L. Xue, and D. Bao, "FLAP: An end-to-end event log analysis platform for system management," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. New York, NY, USA: ACM, 2017, pp. 1547–1556. [Online]. Available: http://doi.acm.org/10.1145/3097983.3098022

[12] B. H. Park, S. Hukerikar, R. Adamson, and C. Engelmann, "Big data meets HPC log analytics: Scalable approach to understanding systems at extreme scale," in *IEEE Cluster 2017 at Workshop on Monitoring and Analysis for High Performance Computing Systems Plus Applications*, Sept 2017, pp. 758–765.

[13] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1773912.1773922

[14] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016. [Online]. Available: http://doi.acm.org/10.1145/2934664

[15] T. T. Authors. (2018) Tornado web server. [Online]. Available: http://www.tornadoweb.org/

[16] K. Pearson F.R.S, "LIII. On lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[17] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[18] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *SIGMOD Rec.*, vol. 22, no. 2, pp. 207–216, Jun. 1993. [Online]. Available: http://doi.acm.org/10.1145/170036.170072

[19] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *SIGMOD Rec.*, vol. 29, no. 2, pp. 1–12, May 2000. [Online]. Available: http://doi.acm.org/10.1145/335191.335372

[20] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[21] H2O.ai, *H2O*, Oct 2016, h2O version 3.10.0.8. [Online]. Available: https://github.com/h2oai/h2o-3