2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

LOGAIDER: A tool for mining potential correlations of HPC log events

Sheng Di[1], Rinku Gupta[1], Marc Snir[2], Eric Pershey[1], Franck Cappello[1 2]
[1]Argonne National Laboratory, USA,
[2]University of Illinois at Urbana-Champaign, USA
{sdi1, rinku, snir, pershey, cappello}@anl.gov

***Abstract*—Today's large-scale supercomputers are producing a huge amount of log data. Exploring various potential correlations of fatal events is crucial for understanding their causality and improving the working efficiency for system administrators. To this end, we developed a toolkit, named *LogAider*, that can reveal three types of potential correlations: across-field, spatial, and temporal. Across-field correlation refers to the statistical correlation across fields within a log or across multiple logs based on probabilistic analysis. For analyzing the spatial correlation of events, we developed a generic, easy-to-use visualizer that can view any events queried by users on a system machine graph. LogAider can also mine spatial correlations by an optimized K-meaning clustering algorithm over a Torus network topology. It is also able to disclose the temporal correlations (or error propagations) over a certain period inside a log or across multiple logs, based on an effective similarity analysis strategy. We assessed LogAider using the one-year reliability-availability-serviceability (RAS) log of Mira system (one of the world's most powerful supercomputers), as well as its job log. We find that LogAider very helpful for revealing the potential correlations of fatal system events and job events, with an accurate mining of across-field correlation with both precision and recall of 99.9-100%, as well as precise detection of temporal-correlation with a high similarity (up to 95%) to the ground-truth.**

I. INTRODUCTION

Today's supercomputers are producing an extremely large amount of log data every day, due to the fact that vast amount of resources (millions of processors, memory cards, etc) are used by a large number of users, jobs, or applications. In a large-scale computing environment [1], scientific simulations are fragile in that failures are normal during execution. However, some system failures could be very peculiar, such that even an experienced administrator may take a long time (several to dozens of minutes per fatal event) to identify their root-causes. Hence, a good understanding of the correlations between different faults and failures is crucial in order to improve administrators' working efficiency significantly.

Exploring the potential correlations of the different fault events on a supercomputer is non-trivial, however, because of vast volume of data and complicated system architectures involved. The BlueGene/Q Mira supercomputer[2] at Argonne, for example, consists of 48 racks that include a total of 49,152 compute nodes, each of which has a PowerPC A2 1600 MHz processor containing 16 cores (bringing the total to 786,432 cores for the entire machine). The system-level log information can be related to any module or component in the whole system, including bulk power supply of a compute rack, clock, I/O drawer, coolant monitor, fan component, fan assembly, PCI adapter, optical module, and DCA module. Mira logs contain masses of information; for example, the Mira job log contains data related to the job scheduling time, execution time, finish time, wall-clock time, number of cores requested, and various types of resources consumed such as memory.

In this work, we develop a novel log analysis model and tool, *LogAider*, that can mine the potential correlations of various events based on the system logs, such as the reliability-availability-serviceability (RAS) log and job log. Not only is LogAider able to filter out duplicated messages very effectively, but it can also disclose the relationships among different events, components, or modules in the system for the diagnosis purpose. We focus on three critical types of potential correlations.

- *Across-Field Correlation*. LogAider can calculate the prior probability distribution of some particular event such as a fatal system failure. LogAider can also reveal the statistical correlations (or posterior probability distribution) across different groups of combined fields or metrics. For instance, LogAider can answer the question: what is the probability that a system RAS event falls in the domain of software_error, for a specific system component (such as "control node kernel" (CNK)) and a specific severity (such as FATAL)? In order to make the combinatorial field analysis tractable, LogAider significantly reduces the total number of field combinations to study, by a series of probabilistic inferences over the combinations of different fields. Such a posterior probability analysis is useful for understanding the significance of the fields and their causality.
- *Spatial Correlation*. To clearly visualize the spatial correlation of events recorded in the log, we developed a generic, easy-to-use viewer that can present various events on a system machine graph. LogAider provides a flexible template for users to customize the layout of the machine graph, so that it is suitable for various supercomputers with different architectures or organizations. We also develop an improved K-means clustering

978-1-5090-6611-7/17 $31.00 © 2017 IEEE
DOI 10.1109/CCGRID.2017.18

algorithm with an optimized number of clustering sets, to explore the potential spatial correlations over the network topology (e.g., the torus network in Mira [2]).
- *Temporal Correlation*. LogAider can disclose the temporal correlation (or error propagations) among different events within a log or across logs, based on a customizable, in-depth similarity analysis of the messages occurring in a relatively short period. Specifically, LogAider can help classify a set of events into multiple groups accurately based on an in-depth similarity analysis with multiple attributes.

Note that the system status can be diagnosed by combining the above correlation mining strategies flexibly because of our loosely-coupled design of the log analysis model. For example, in order to narrow the suspicious events upon failures, the administrator can remove a lot of duplicated messages first based on the temporal-spatial filter, and then select the critical fields/items based on the across-field analysis. After that, correlated events can be merged based on the temporal correlation analysis with only selected key fields, and finally demonstrate the related fatal events that are classified by our optimized K-means clustering algorithm.

Evaluation results with a one-year RASlog and job log from Argonne's Mira system confirm the high accuracy of LogAider in the mining of across-field correlation (precision and recall are both 99.9% for RAS log and 100% for Job log). Moreover, the evaluation shows that the LogAider is helpful for revealing spatial-correlations and very accurate (up to 95% similarity compared with ground-truth) in the mining of temporal correlations. The total number of fatal events can be reduced to 193 compared with originally 100k fatal messages.

The rest of the paper is organized as follows. In Section II, we discuss related work on analysis tools. In Section III, we present the design overview of LogAider. In Section IV, we describe how LogAider filters duplicated messages. In Section V, we present details of the design and implementation of LogAider. We demonstrate the evaluation results of LogAider using the one-year RAS log and job Log in the Mira system in Section VI. We give concluding remarks and briefly discuss our future work in Section VII.

II. Related Work

Much of existing log analysis is focused on failure-analysis research instead of a generic toolkit development. Sirbu and Babaoglu [3], for example, proposed a holistic approach to log data analysis in high-performance computing (HPC) systems, presenting a case study based on four datasets reporting power consumption, temperature, workload and hardware/software events on the IBM Blue Gene/Q. Yuan et al. [14] explored the properties of job failures based on 10 HPC datasets across different sites. Zheng et al. [11] analyzed the system log and explored the potential rules. Tiwari et al. [15] explored the GPU errors on large-scale HPC systems. Gupta et al. [16] explored spatial failure properties on an extreme-scale HPC system, the Titan Cray XK7 with a total of 300k CPU cores.

Log analysis and mining tools have been studied for years especially for HPC logs. The Hierarchical Event Log Organizer (HELO) [4] is an event log mining tool for large-scale HPC systems. It can extract templates (or event patterns) based on system log files. Unlike HELO, Event Log Signal Analyzer (ELSA) [5] aims to characterize the normal behavior and faulty behavior of the system by using signal analysis. In particular, it models the normal flow of each state event during an HPC system lifetime and how it is affected when a failure hits the system.

In addition to the log analysis tools, some tools [6], [7], [8], [9], [10] are designed particularly for mining the correlations of events in the log. LogMaster [6] is a tool designed for mining event rules in supercomputing system logs. The authors propose a metric to measure the correlations of events that may happen interleavedly. Similar to ELSA, however, it is limited to analysis of repeated event patterns or signals, without analysis of other types of correlations. LogDiver [7] is an analysis tool for handling the data generated by system monitoring tools on Blue Waters [18], a petascale machine at the University of Illinois' National Center for Supercomputing Applications. It is able to filter, extract, and classify error data in the logs; extract signals from the categorized errors; and correlate application failures with errors using a mix of empirical and analytical techniques.

LogAider differs from these tools in three ways: (1) LogAider is a log analysis tool providing optimized mining strategies on three types of correlations: across-field, spatial/locational and temporal. (2) To the best of our knowledge, each of these strategies is the first attempt to be used in exploring the corresponding correlations. For example, LogAider mines the across-field correlation by a Bayesian probability model and explores the spatial correlation using an optimized K-means clustering algorithm. Unlike existing research [11], [17] that focuses on how to filter duplicated messages based on event types (such as categories or components), we propose a novel similarity-based temporal correlation mining method to reveal the potential error propagations across types of events. (3) LogAider also provides a generic visualizer that can be used for different system architectures and layouts because of the flexible, easy-to-use templates.

III. Design Overview

In our design, the whole log analysis system has four layers: *user interface layer*, *analysis engine layer*, *log parsing layer*, and *log data layer*, as shown in Figure 1.

The top layer, the user interface layer, contains the control information provided by users. The *format template* refers to the schema or format information of the log data. The *key fields template* allows users to specify the key fields

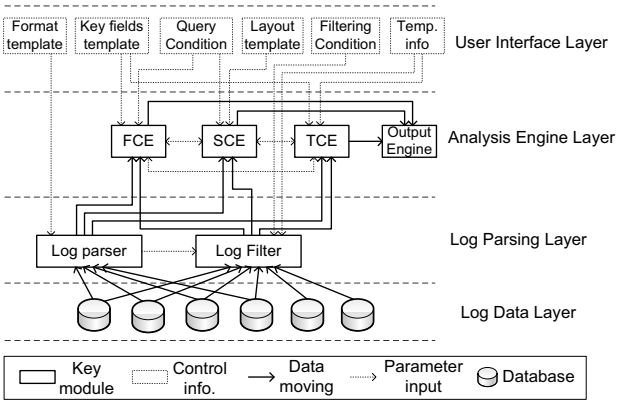


Figure 1. Design Architecture

for across-field correlation analysis and temporal-correlation analysis. The *query/filtering condition interface* allows users to customize the condition for filtering insignificant or duplicated messages. The *layout template* provides the system architecture layout or organization of the machines for the spatial correlation analysis. The *temporal information interface* allows users to provide more specific information (such as reservation period and maintenance period of the system) for filtering duplicated messages more accurately.

The bottom layer includes various log data, such as the system RAS log and job log. The log data can be stored in the form of a relational database such as Mysql or in textual files. In this work, we collect one-year RAS log and job log data in 2015 for our evaluation. All the data were originally stored in a DB2 database. For confidentiality reasons, the log data were cleared by the system administrator before being provided for analysis. Specifically, all the potentially confidential information such as user names and project names were replaced by hashcodes, which are still valid for the statistical study.

The log parsing layer performs the important preliminary processing of the log, such as the data format parsing and filtering of the duplicated messages. In particular, the *log parser* reads the schema information (or format template) to recognize the meaning of each field. The *log filter* filters the duplicated messages based on temporal filter, spatial filter, or a combination of both. Users can also provide more complicated filtering conditions or information such as maintenance periods, which is very important for improving the filtering accuracy.

The analysis engine layer is the key layer of the whole correlation analysis toolkit, which includes four critical modules: across-field correlation engine (FCE), spatial correlation engine (SCE), temporal correlation engine (TCE), and output engine. The first three modules are in charge of the three different types of correlation analysis, respectively.

- The across-field correlation engine, for example, takes the log (either filtered or non-filtered data) as its input and computes the prior probability distribution based on various metrics or fields and posterior probability (i.e., conditional probability) based on any combination of the key fields set by users.
- The spatial correlation engine focuses on the spatial distribution of the messages with respect to different metrics such as component, category, and severity, based on user-specified time periods. Not only can it mine potential spatial correlations by an optimized K-means clustering algorithm, but it can also present the correlations on a graph with multiple component levels (such as compute rack, midplane, and compute node).
- The temporal correlation engine mines the potential temporal correlations, based on a similarity analysis of the events that occur closely in time.

IV. Preliminary Step: Log Parsing and Filtering

The log parsing layer includes two key modules: log parser and log filter.

The log parser plays an important role in enabling the substrate analysis engines to read the log data correctly, in that the data format or schema can differ significantly with systems or log databases. As an example, we summarize some differences of the data schema/format based on our observations with Mira logs.

- The meanings and orders of the fields can be different with logs. For instance, there are 15 fields in the RAS log, while there are 53 fields in the job log. The event time stamp is the second field in the RAS log, while it is the fourth field in job log.
- As for the fields that have the same meanings across logs, the representation formats can be largely different. The event time stamps, for example, can be represented in different formats such as "yyyy-mm-dd hh:mm:ss" and "yyyy-mm-dd-hh.mm.ss".
- The items of one message may be separated by different separators with different logs. In addition, we observe from the Mira RAS log that some items are surrounded with double quotations in case of ambiguity, which need to be coped with before the mining work.

The log filter plays a critical role on ensuring that the messages collected are accurate based on the user's demand. Specifically, the original log generated by the system may contain a lot of duplicated messages, which should be coped with carefully when studying the error propagations. We observe two types of duplicated messages based on the one-year Mira RAS log. On the one hand, some fatal events may be reported periodically at the same location in the system, because of the periodic logging mechanism. In this case, only the first fatal event message should be counted when computing the mean time to interruption (MTTI). This type of duplicated messages is called *temporal duplication*. On the other hand, a large number of fatal events may occur with close time stamps on different locations (such as compute nodes or I/O nodes). This is probably due to the fact that one

fatal event (such as link error or power outage) may incur many other events on different locations in the system. We call this duplication type *spatial duplication*.

Some filtering method is necessary for avoiding the duplicated calculation of fatal events. LogAider provides three filtering approaches - temporal filter (TF), spatial filter (SF), and temporal spatial filter (TSF) - which can be adopted selectively in different cases on demand. The analysis of failure rate (or MTTI), for example, requires filtering the duplicated events in both dimensions and also temporal correlations across event types. On the other hand, the analysis of spatial features based on event locations should involve all messages in different locations, thus it should only filter the duplicated messages in temporal dimension.

As strongly recommended by the Mira system administrator, our log filter also takes into account the maintenance periods and reservation periods. Specifically, all the events that occur in the regular maintenance periods should be removed from the study, because many fatal event messages will be generated by the administrator for system testing. Moreover, all the messages that intersect via racks in each of the reservation periods should be grouped as one event, as indicated by the system administrator.

V. In-depth Data Mining: Analysis Engine Layer

In this section, we describe the key strategies of mining the potential correlations inside a log or across multiple logs, which is the main contribution of the paper. Our design is based on the three key correlations studied in this work: across-field correlation, spatial correlation and temporal correlation.

A. Across-Field Correlation Engine

The objective of the across-field correlation analysis is to understand the significance of fields and their combinations in determining the attributes of the messages, such as message's severity and category. Specifically, we seek to answer the following two questions:

- What is the occurrence probability for any combination of values across fields based on the log analysis?
- Given a message, which items in the combination of the fields play a critical role in determining its properties (such as severity and category), based on the statistical analysis of the log history? For instance, what is the probability that a system RAS event is of FATAL severity level, provided that it falls in the Software_error category and its involved component is CNK?

One important concern in exploring the across-field correlations is that the total number of combinations of items across fields may be extremely large. As mentioned, for Mira, there are 15 fields in the RAS log and 53 fields in the job log. Suppose one focuses only 6 key fields (MSG_ID, CATEGORY, COMPONENT, SEVERITY, BLOCK, and CTLACTION in the RAS log), there will be 1 billion value-combinations across fields, because each field may have many values, as shown in Table I. Such a large number of combinations will definitely introduce a huge memory cost and computation cost.

Table I
Key Fields and Values in MIRA RAS Log

Field Name	# Values	Example Values
MSG_ID	86	00010001 (Kernel unexpected operation)
CATEGORY	18	Software_error, BQC
COMPONENT	22	Compute Node Kernel (CNK)
SEVERITY	3	INFO, WARN, FATAL
BLOCK	384	MIR-08000-3BFF1-8192 (a 8192-node block)
CTLACTION	19	END_JOB, FREE_COMPUTE_BLOCK

Fortunately, most of the value combinations can be deemed as *impossible events* (i.e., their probability is zero according to the log), which means that we can use this feature to significantly reduce the number of combinations that need to be studied. Specifically, if the probability that a value from some field always appear together with one or more values from another field, the number of combinations across these two fields is actually limited. The messages with MSG_ID of 00010001, for example, must be in the "Software_error" category and its severity is always FATAL in the log. The impossible combinations across multiple fields can be easily inferred and extended from the impossible events across two or three fields. In our implementation, all of the value combinations construct a value-combination pool, and they are organized by a hash table in order to achieve a fast processing speed. By running the across-field correlation engine analysis over the nonfiltered Mira RAS log, there are only 107k value-combinations based on the six key fields.

The across-field correlation engine can also perform online analysis of the messages to identify the properties of the event and the correlation of the values across multiple fields related to the event. Given a message generated online in the system, the engine can calculate the conditional probabilities of every combination of fields based on the value-combination pool already constructed by history and can output the results in decreasing order of probability. Specifically, the probabilities are calculated based on the Bayesian theorem, i.e., Formula (1).

$$P(\{S_1, S_2, \cdots\}|\{E_1, E_2, \cdots\}) = \frac{P(\{S_1, \cdots, E_1, \cdots\})}{P(\{E_1, E_2, \cdots\})} \quad (1)$$

where S_i refers to the target states, E_i refers to the evidences, and they are both filled with field's values in the analysis.

We also make use of some probability inferences (as show below) based on the properties of inevitable events, in order to minimize the computation cost.

$$P(Z|X) = 1 \Rightarrow P(Z|XY) = 1 \quad (2)$$

$$P(X|Y) = 1 \quad \text{and} \quad P(Z|W) = 1 \Rightarrow P(XZ|YW) = 1 \quad (3)$$

where X, Y, Z, and W refer to the values from different fields. Note that the value-combination pool can be updated

over time by recording more and more messages in the system; hence, the across-field probability analysis would be more and more comprehensive and accurate over time, which is appropriate for large-scale online analysis.

Our across-field correlation engine also provides a significance analysis of the correlation between any pair of the fields in the log. In particular, the engine first generates the contingency tables for each pair of the key fields specified by the users. A contingency table is in the form of matrix that displays the frequency distribution of the value combinations across two fields. As an example, Table II presents a portion of the contingency table for the two fields job_exit_code vs. job_execution_size (i.e., the number of cores), based on the Mira job log.

Table II
CONTINGENCY TABLE EXAMPLE (JOB_EXIT_CODE VS. EXE_SIZE)

# cores / exit_code	393k	197k	786k	524k	262k	131k	65k	16k	33k	8k
0	1	1	18	25	94	219	373	425	889	1621
143	0	4	0	6	35	72	64	92	225	470
1	0	3	2	1	8	35	37	15	32	169
139	0	0	0	4	2	7	5	19	13	97
134	0	0	0	1	0	8	11	19	25	46
255	0	0	0	0	0	0	2	1	2	18
72	0	0	0	0	0	0	0	0	0	18
128	0	0	0	0	0	0	0	7	2	4
...	...	...	...	...	...	...	...	...	...	...

After collecting all the contingency tables, the engine will compute the χ^2 value for each pair of fields based on the following formula.

$$\chi^2=\sum_{i=1}^{r}\sum_{j=1}^{c}\frac{(O_{i,j}-E_{i,j})^2}{E_{i,j}}, \text{where } N \text{ is total sample size,}$$
$$E_{i,j}=Np_{i*}p_{*j}, p_{i*}=\frac{\sum_{j=1}^{c}O_{i,j}}{N}, p_{*j}=\frac{\sum_{i=1}^{r}O_{i,j}}{N} \quad (4)$$

By comparing the calculated χ^2 value (denoted by χ_c^2) with the standard χ^2 distribution value (denoted by χ_s^2) based on the given confidence level α, the engine will determine whether the two fields are highly correlated mutually (iff $\chi_c^2 \geq \chi_s^2$) or not. The χ^2 value of the two metrics (exit_code and execution size) shown in Table II, for instance, is computed as 7769, which is far greater than 603.45 (the 99.9%-confidence-level based χ^2 value), so we can conclude that the two metrics are strongly correlated with each other in this case.

Note that various numbers of evidence samples (or log records) mean different levels of confidence on the results. Thus we also include in the correlation results the confidence metric in terms of the margin of error. Margin of error is a decreasing function of the sample size, as shown in Equation (5).

$$E_m=\frac{erf^{-1}(X)}{\sqrt{2N}}, \text{where}$$
$$erf^{-1}(z)=\sum_{k=0}^{\infty}\frac{c_k}{2k+1}\left(\frac{\sqrt{\pi}}{2}z\right)^{2k+1}, \quad (5)$$
$$c_0=1, c_k=\sum_{m=0}^{k-1}\frac{c_m c_{k-1-m}}{(m+1)(2m+1)}$$

where X is the confidence level. For instance, $E_m \approx 1.29/\sqrt{N}$, if the confidence level is set to 99%. That is, the larger sample size, the smaller error the analysis will have.

B. Spatial Correlation Engine

In the spatial correlation engine, we designed a generic visualizer and a clustering algorithm for disclosing the spatial correlations effectively. The visualizer can be used to demonstrate the spatial correlation of the events for any supercomputers based on a flexible user-customized layout template. Specifically, our template requires users to provide key information about the system layout, such as row count and column count of the racks, the number of midplanes per rack, and the number of nodes per midplane. The clustering algorithm is designed based on a K-means clustering strategy. In particular, we optimize the number of clusters by merging the nearby centroids of the clusters iteratively in order to obtain the optimal clustering effect. Details are presented in the following text.

1) Generic Visualization of Spatial Correlation: The compute machines in Mira system have at least four layers: there are 48 racks, each of which has two midplanes. Each midplane has 16 node boards, each having 16 card boards. Users can view the machines in different layouts, by customizing the arrangement of the racks, midplanes, and nodes as M rows by N columns or using a default layout with either row-major order or column-major order. Users can also arrange the indices of the racks or machines in different ways, such as binary, octal, hexadecimal, or decimal, to suit the diverse representations with organization levels. For instance, the indices of the racks and midplanes are presented in hexadecimal format and binary format in the Mira system, respectively, as shown in Figure 2.

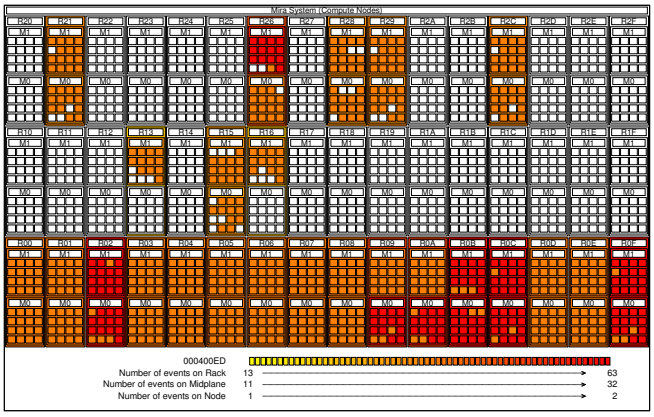


Figure 2. Demonstration of Fatal Events in Mira (msg_ID=000400ED)

In order to demonstrate the statistical information on the machine graph, the spatial correlation engine provides a set of functions to query the target events based on the user's demand, and it calculates the event count at different levels (rack, midplane, node board, and card board). To present the spatial correlation of the events, one also needs to map an event's spatial information in the log to a location of the

machine graph. The location of a Mira system event, for example, can be found in the BLOCK field of each RAS message.

Figure 2 demonstrates the machine graph plotted by the above layout template with the spatial distribution regarding a fatal event type (msg_ID=000400ED) throughout the year 2015 based on the Mira log. The graph shows all the fatal event_messages excluding the maintenance periods but without adopting temporal or spatial filters; hence, the number of fatal events is relatively large and does not represent MTTI or MTBF regarding these events. The accurate estimate of MTTI or MTBF should take into account filtering of duplicated messages, maintenance period, reservation period, temporal correlations of events, and so on.

2) Optimized K-means Clustering for Mining Spatial Correlation: In addition to the generic visualization of event distribution, we integrate a set of analysis technologies for mining the spatial correlations of the events statistically. To this end, we need to reconstruct the links among all machines based on their connection topology. In what follows, we present how to reconstruct the topology using the Mira as an example; our design can be extended flexibly for other system architectures.

In Mira, all the compute resources are allocated to jobs in the unit of the midplane. Thus the spatial correlation for Mira is supposed to be analyzed based on the across-midplane structure. The overall system adopts a 5D torus to interconnect all the machines, and every midplane connects to others using the first four dimensions (or four links, denoted as A, B, C, and D), as presented in Figure 3. The sizes of the four torus dimensions are 2, 3, 4, and 4, respectively. As shown in the figure, R00-M1 and R08-M1 connect with each other on link A. R00-M1, R10-M1, and R20-M1 are connected at link B. On link C, R11-M1 connects R15-M1, which is connected to R17-M1, and then R13-M1, and finally back to R11-M1. The link D connects midplanes across a pair of racks in a clockwise direction: for example, R28-M0 → R28-M1 → R29-M1 → R29-M0 → R28-M0.

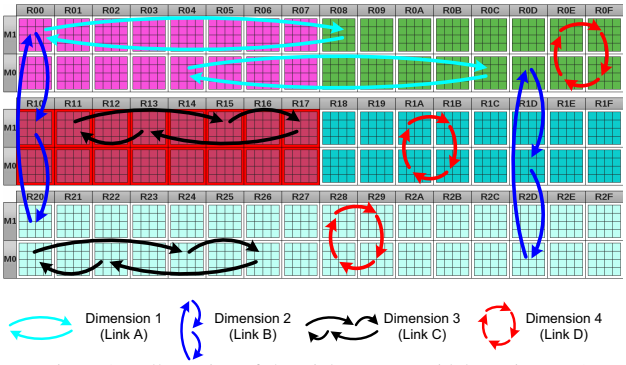


Figure 3. Illustration of the Links among Midplanes in MIRA

We develop a novel K-means clustering algorithm that can optimize the number of groups (or sets) based on the torus topology. Our algorithm differs from the traditional K-means clustering algorithm in two ways. (1) Unlike the traditional algorithm that requires fixing the number of clustering groups in advance, our algorithm optimizes the group counts by merging the nearby centroids of groups automatically until the convergence. (2) Unlike the traditional K-means algorithm that conducts the clustering in a Euclidean space, the distance between any two points (i.e., between two midplanes in Mira) is computed by identifying the shortest path between them in the torus. The shortest path from R00-$M1$ to R1D-$M0$, for example as shown in Figure 3, is R00-$M1$→R08-$M1$→R18-$M1$→R1C-$M1$→R1D-$M1$→R1D-$M0$. The specific calculation of distance between two resource units (or midplanes) M_1 and M_2 on the torus is given as follows:

$$\begin{aligned}\text{distance}(M_1, M_2) = &\sum_{x=A,B,C,D} \min\{d_x(M_1,M_2), d'_x(M_1,M_2)\},\\ &\text{where } d_x(M_1, M_2) = |i_x(M_1) - i_x(M_2)|,\\ &d'_x(M_1, M_2) = size_x - |i_x(M_1) - i_x(M_2)|\end{aligned} \tag{6}$$

where $size_x$ refers to the size at dimension x, and $i_x(M_1)$ refers to the location index of the midplane M_1 at the dimension link x. The distance(R00-$M1$,R1D-$M0$), for example, is computed as 5 according to the formula.

The pseudo-code of our designed clustering algorithm is presented in Algorithm 1. At the beginning, the centroids are initialized by using the Forgy method [20] based on the maximum group count specified by the user. Specifically, the algorithm selects a set of resource units (or midplanes) randomly as the initial centroids and tentatively performs K-means clustering on them to get the within-cluster sum of distance (WCSD). The algorithm selects the best set of initial centroids with minimum WCSD and then optimizes the number of clustering steps. Specifically, the algorithm iteratively performs the following steps: compute the average distance of the set of centroids and merge the nearby centroids based on a merge_ratio_threshold until no any centroids are merged in an iteration round or only two centroids are left.

Algorithm 1 TORUS-BASED GROUP-COUNT OPTIMIZED K-MEANS ALGORITHM

Input: N compute resource units (denoted as S), max # groups, λ (merge ratio threshold)
Output: classified groups of resource units).

1: *CS* = the set of centroids initialized by Forgy method [20].
2: **repeat**
3: Compute *KM*(|*CS*|, *S*, *CS*); /*|*CS*| means # elements in *CS*/
4: Compute the average distance in the set of centroids *CS*: $\bar{d} = \frac{\sum_{\forall P_1,P_2 \in CS} \text{distance}(P1,P2)}{|CS| \cdot |CS-1|/2}$;
5: **if** ($\exists P_1 \in CS, P_2 \in CS, \text{distance}(P_1, P_2) < \lambda\bar{d}$) **then**
6: CS = MergeCenters(*CS*, $\lambda \cdot \bar{d}$);/*Merge nearby centers*/
7: **else**
8: break;
9: **end if**
10: **until** ($|CS| == 2$)
11: Output the centroid set *CS* and optimized clustering sets *KM*;

In our implementation, we provide two clustering modes, focusing only on event locations or taking into account the message count on the same locations, because both modes are useful in different cases. For the second clustering mode, we group all the messages happening on the same resource unit as one spatial point, and we consider the number of messages related to this resource as its *weight*. Then, the K-means clustering will be performed based on weighted distance instead. Such a weighted K-means clustering design can significantly reduce the computation cost, because the time complexity is exactly the same as that of the first clustering mode that ignores the number of messages on the same location. Specifically, the time complexity of our optimized K-means clustering is $O(N^2)$, where N is the number of resource units over the network topology, and it is supposed to be relatively small in reality (e.g., Mira has only 96 midplanes).

C. Temporal Correlation Engine

The temporal correlation engine is designed to mine the potential correlation of events in the time dimension. Specifically, the objective is to search a set of events that have high mutual similarity and also occur closely in time (or within a tolerant period specified by the user). In what follows, we first present the pseudo-code of our algorithm about how to search the temporal-correlated events based on the similarity value. We then describe how to design and compute the similarity between two events in details.

1) Searching Temporal-Correlated Events by Similarity: We present the pseudo-code of the similarity-based temporal-correlation mining algorithm in Algorithm 2. The inputs are the event_messages that have been filtered by our temporal-spatial filter (see Section IV), such that the algorithm focuses only on the temporal correlation of events. The key idea is to compute the similarity between every event and its following events within a certain short period, in chronological order (lines 1-17), and group the events with its preceding events based on the similarity (line 18-22). We design two types of the delay_threshold for selecting the subsequent events followed by a target event as the candidate correlated events, based on whether the message ID is the same. Specifically, delay_threshold of the two events with different message_IDs should be shorter than that of the two events with the same message_ID, because the events with the same message_ID are more likely correlated with each other even though their occurrence interval is relatively long (e.g., a couple of hours). We map the correlation between the target event and its candidate subsequent events, as long as their similarity values exceed a threshold λ. We find that the temporal-correlation clustering results are very stable with different λ settings (to be shown later), which means a high fidelity of our algorithm.

Our temporal-correlation mining algorithm has three-fold key features. (1) Its working principle is close to human's behavior, in that it explores the temporal-correlations not only based on occurrence time stamps but also based on the similarity of two events on multiple facets, such as event types (i.e., message ID), category, and location. (2) The similarity function for each field is customizable on demand, and hence can optimize the accuracy of the analysis. (3) This algorithm has a linear time complexity $O(N)$, in that the time complexity of computing the similarity of each event and its subsequent events can be thought of as a constant. This is due to the fact that the fatal events in a supercomputer always occur sparsely over time. For instance, Mira has only 407 fatal events in the whole year 2015, after filtering the duplicated messages with a tolerable delay of 60 seconds (note that 407 is not the MTTI or MTBF because of missing temporal-correlation analysis here). In this case, the occurrence interval follows an exponential distribution; that is, the occurrence of fatal events can be treated as a Poisson process [21] with a fixed number of events per time unit.

2) Design of Temporal-Correlation Similarity: As discussed in Section V-A, the significance of a message's information is different with various fields in the log. For instance, message_ID in the Mira RAS log is one of the most important fields because it determines the values of many other fields, such as the category of the event. Another key information is the location where the event occurs. Since the system has a total of 48 racks and 96 midplanes, the two events will likely be correlated if they occur on the same rack or midplane within a short period. Our toolkit allows users to assign different weights for different fields in a configuration

Algorithm 2 TEMPORAL-CORRELATION MINING ALGORITHM BASED ON SIMILARITY

Input: N event_messages that have been cleared by the temporal-spatial filter, the delay_threshold for messages with same msg_ID (denoted T_s), the delay_threshold for the messages with different msg_IDs (denoted T_d), similarity threshold used to determine correlated events (denoted λ).
Output: classified groups of temporal-correlated messages and the similarity values of the messages within each group.

```
1: for (event_message msg_i in an increasing order of time stamps) do
2:     t_i ← time stamp of msg_i;
3:     for (each msg_j that occurs after msg_i) do
4:         t_j ← time stamp of msg_j;
5:         if (t_j − t_i ≤ T_d) then
6:             Compute similarity θ(msg_i,msg_j) with T_d;/*Formula (7)*/
7:         else if (t_j − t_i ≤ T_s and msg_ID(msg_i)==msg_ID(msg_j)) then
8:             Compute similarity θ(msg_i,msg_j) with T_s;/*Formula (7)*/
9:         else
10:            break;
11:        end if
12:        if (θ(msg_i,msg_j)≥λ) then
13:            msg_j's preCorrelatedEvent ← msg_i; /*correlation mapping*/
14:            t_i ← time stamp of msg_j; /*update time stamp*/
15:        end if
16:    end for
17: end for
18: for (event_message msg_i in an increasing order of time stamps) do
19:    if (msg_i's preCorrelatedEvent ≠ null) then
20:        Merge msg_i to the preCorrelatedEvent with maximum similarity;
21:    end if
22: end for
```

file. Users can also customize the similarity function based on the features of fields. The overall similarity between two event messages is the weighted sum of the similarity value of each important field specified by the user, as presented in Formula (7), and its value must be in the range [0,1].

$$\theta(Msg_1, Msg_2) = \sum_{\alpha_i \in Msg_1, \beta_i \in Msg_2} \omega_i \cdot \vartheta_i(\alpha_i, \beta_i), \quad (7)$$

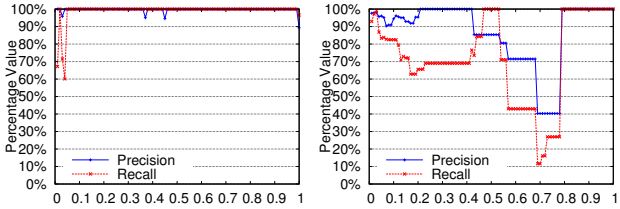
where i is the field index, ω_i is its weight, and ϑ_i is the corresponding similarity function.

In the Mira RAS log, for example, six important fields can be used to estimate the similarity of two messages: msg_ID, category, component, event_time, location_mode, and location_code. As examples, we describe three of them in details; and all the similarity values must be in [0,1].

- Similarity of msg_ID: Message ID is the unique identifier of an RAS event, and each consists of 8 digits (such as 00080012) that can be split into two parts. The first four digits indicate the component family, and the last four digits identify the event feature. Hence, the similarity is defined as
$$\vartheta_1(\alpha_1, \beta_1) = \begin{cases} 0, & \text{first 4 digits are different.} \\ 0.5, & \text{first 4 digits are same, last 4 are not.} \\ 1, & \text{all 8 digits are the same.} \end{cases}$$
- Similarity of event_time: The closer the two messages occur in time, the higher the correlation they should have. Thus, the similarity w.r.t. event_time can be defined as follows: $\vartheta_2(\alpha_2, \beta_2) = \frac{\text{delay\_threshold} - \text{interval}(Msg_1, Msg_2)}{\text{delay\_threshold}}$, where the interval(Msg_1,Msg_2) indicates the delay between two messages and delay_threshold is a temporal threshold value used to select the candidate correlated messages.
- Similarity of location_code: In Mira, the location of each message is represented by a location_code. For example, R1F-M0-N14-J11 represents a 4-layers location in a compute rack: compute rack 1F, midplane 0, node board 14, and compute card 11; Q0J-I6-J04 represents a location in I/O racks: I/O rack Q0J, I/O drawer, and I/O card. In this case, the similarity of location is defined as follows: $\vartheta_3(\alpha_3, \beta_3)$=0.25×{the number of highest same_code layers}.

VI. Evaluation of LogAider

We first evaluate the accuracy of our across-field correlation mining method, based on both the Mira RAS log and job log in the whole year 2015. We choose months 1∼6 as the training period and use the remaining 6 months to assess the precision and recall of the conditional probability computed in our solution. Specifically, we build the probability pool based on log data in months 1∼6, and then compute the estimated conditional probability of any possible combination of fields based on the Bayesian formula (1), as well as the real conditional probability in the same way based on the last six months of log data. We check 100 thresholds (from 1% through 100%) to see whether our probabilistic training model can lead to high precision (=$\frac{TP}{TP+FP}$) and recall (=$\frac{TP}{TP+FN}$). In particular, *TP*, *FP*, and *FN* refer, respectively, to true positive (the number of cases that the estimated probability and real probability are both greater than the threshold), false positive (the number of cases where the estimated probability is greater than the threshold while the real probability is not), and false negative (the number of missing cases whose real probability is greater than the threshold because of the low estimated probability).

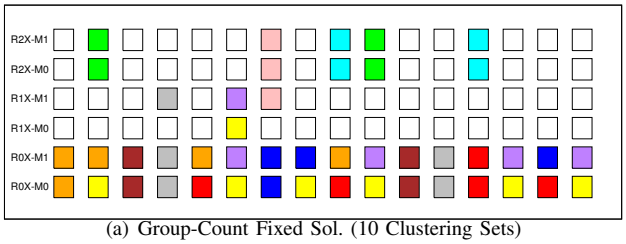


(a) RAS-based Experiment (b) Job-based Experiment

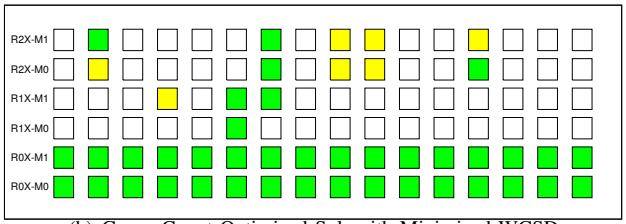
Figure 4. Evaluation of Across-Field Correlation (Precision and Recall)

Figure 4(a) shows that with respect to the RAS log, our across-field correlation engine works effectively on a large majority (97%) of the thresholds: the precision and recall are both around 99.9%. As for the job log (Figure 4(b)), our solution can always lead precision and recall both to 100% when the threshold is greater than 80% (i.e., 0.8). Note that users care only about high-probability cases for identifying the significant fields, which means that our across-field correlation mining method is already accurate enough for users based on the above results.

As for the spatial correlation, we present in Figure 5 and Figure 6 the K-means clustering results for two types of fatal events (msg_ID=000400ED and msg_ID=00090216), respectively. Small squares here refer to midplanes.



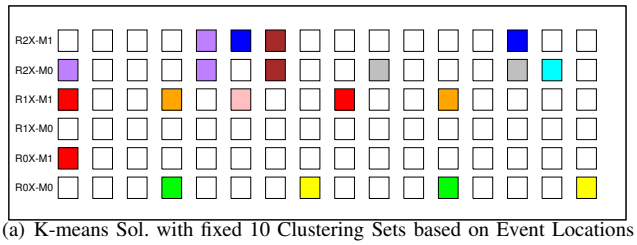
(a) Group-Count Fixed Sol. (10 Clustering Sets)



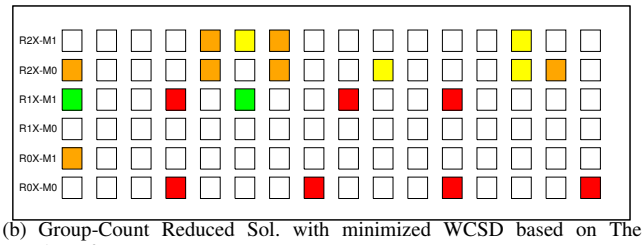
(b) Group-Count Optimized Sol. with Minimized WCSD

Figure 5. K-means Clustering Results with Fixed-Group-Count vs. Optimized-Group-Count (MIRA msg_ID=000400ED)

Figure 5 presents the fatal events whose message IDs are all 000400ED, which indicates that "the related boards



(a) K-means Sol. with fixed 10 Clustering Sets based on Event Locations



(b) Group-Count Reduced Sol. with minimized WCSD based on The Number of Messages

Figure 6. K-means Clustering Results with Fixed-Group-Count vs. Optimized-Group-Count based on Link Error (MIRA msg_ID=00090216)

have become unresponsive, such that they are marked as unusable" according to the Mira RAS book (details about the fatal event locations are shown in Figure 2). As presented in Figure 5, the clustering results of our solution with an optimized number of groups are largely different from that of the clustering with a fixed number of groups. As indicated by the system administrator, the bottom racks uniformly experienced the same errors in a period, so they should be grouped together. This confirms that our improved K-means clustering with an optimized number of clustering sets does result in more accurate classification in this case. That is, our optimization is an important step in obtaining the accurate clustering results from the perspective of users.

In fact, the original K-means clustering algorithm with fixed number of groups is also helpful to disclose the fine-granularity spatial correlations for users. In Figure 6, we present the results of the K-means clustering that uses the fixed 10 clustering sets based on only event locations versus the clustering that reduces the number of clustering sets with minimized WCSD based on the number of messages occurring on the resource units (midplanes). In this case, the fatal events are all link failures detected between nodes connected via copper and optical links (message ID = 00090216 specifically). We can observe that the former K-means clustering strategy (Figure 6) exhibits finer granularity of the link failures than does the latter. As such, LogAider also allows users to specify the number of groups for mining the spatial correlation on demand.

We also evaluated the accuracy of our similarity-based temporal-correlation mining strategy using the one-year Mira RAS log. Duplicated messages have been filtered by the temporal-spatial filter with a filtering period of 1 minute, in that we focus only on temporal correlation in this experiment. Only 407 filtered fatal events are left, compared with originally 100k non-filtered fatal messages. We build the ground-truth clustering by manually classifying the fatal events into different clusters based on their complex attribute information and asking the system administrator to further adjust it. We then run our solution (called Similarity-Based Miner or SBM, T_d=30 minutes, T_s=18 hours), and observe that the fatal event count is further reduced to 193, which is lower by 52.6% than the duplication-filtered event count. (notice that 193 here is not MTBF/MTTI either because the system can fix some of them automatically, though they are all potential fatal events based on RAS log).

We run two other state-of-the-art approaches temporal-spatial miner (TSM) and adaptive semantic miner (ASM) for the comparison. TSM is a mining strategy modified from temporal-spatial filter, and it merges the nearby events only based on time stamps and event types. ASM is designed based on Adaptive Semantic Filter [17], that ises ϕ correlation coefficient [22] to filter/merge the similar events and also adaptively enhances the merging threshold as time gap increases. We traverse time thresholds (from 1 minute to 18 hours) for TSM and ASM to search for their best results.

The accuracy indicator is *clustering similarity* between different solution outputs and the ground-truth clustering results respectively. The clustering similarity [23] is used to assess the accuracy of the correlation-based clustering results generated by different solutions. It is defined as follows.

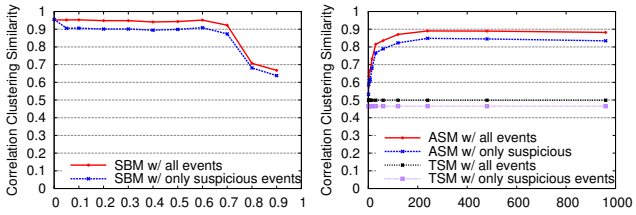
$$\rho(C_1, C_2) = \frac{1}{n}\sum_{i=1}^{n}\left(\max_{j=1,2,\cdots,m}\rho(S_i, S_j)\right), \quad \text{where } S_i \in C_1, S_j \in C_2, \rho(S_i, S_j) = \frac{|S_i \cap S_j|}{|S_i \cup S_j|} \tag{8}$$

where C_1 and C_2 are two sets of clusters respectively. In our case, C_1 is the clustering sets generated by the solution, and C_2 is the ground-truth clustering sets

We present the clustering similarity of the three comparative solutions in Figure 7. As shown in the figure, we evaluate the solutions based on two types of fatal event sets: one with all fatal events and the other one with only suspicious fatal events each of which has a neighbor event with at most a one-day interval. Based on the figure, we can see that the clustering similarity of our solution (SBM) can reach up to 95% when λ is set to lower than 0.6, so $\lambda \leq 0.6$ is the recommended setting in practice. We also observe that the clustering result of our solution is very stable: $\geq$90% in most threshold settings. By comparison, ASM can lead clustering similarity only up to 89%, and it is very sensitive to the time threshold setting. The key reason SBM performs much better than ASM and TSM is twofold: (1) SBM allows users to customize more reasonable similarity for each of the key fields, and (2) SBM treats various fields with different weights, a strategy that can lead to more accurate estimate on the overall similarity.

VII. Summary and Future Work

In this work, we develop a novel, easy-to-use tool, LogAider, for mining various types of correlations in HPC



(a) Our solution (SBM) (b) Other related work

Figure 7. Evaluation of Temporal Correlation Mining Accuracy

log. To the best of our knowledge, each of our strategies is the first attempt to be used in mining the corresponding correlations. In particular, we optimize the across-field correlation analysis by using a Bayes probability model, such that the precision and recall can reach 99.9∼100%. We propose an improved K-means clustering algorithm with an optimized number of groups for mining the spatial-correlations over a torus network topology. We also develop a novel similarity-based temporal-correlation analysis model that can effectively reveal the temporal correlations (or error-propagations), such that the total number of fatal events can be significantly reduced to 193 compared with the originally 100k fatal messages and the clustering similarity/accuracy can reach 95%. In the future, we will include more analysis strategies and online prediction methods to make LogAider more powerful for HPC data analysis.

Acknowledgments

This research used data of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357 by the U.S. Department of Energy. It is not based on the tools and techniques used by the ALCF team. This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357, and by the ANR RESCUE, the INRIA-Illinois-ANL-BSC Joint Laboratory on Extreme Scale Computing and the DOE Catalog project.

REFERENCES

[1] M. Snir, "Addressing Failures in Exascale Computing, " in *International of High Performance and Computing Applications (IJHPCA)*, 28(2): 129-173, 2014.

[2] Mira System. [online]. Available: https://www.alcf.anl.gov/mira.

[3] 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 *International Parallel Processing Workshops on Euro-Par 2015*, 2015, pp. 631–643.

[4] A. Gainaru, F. Cappello, S. Trausan-Matu, and B. Kramer, "Event Log Mining Tool for Large Scale HPC Systems, " in *Proceedings of 17th International Conference on Parallel Processing (Euro-Par'11)*, 2011, pp. 52–64.

[5] A. Gainaru, F. Cappello, W. Kramer, "Taming of the Shrew: Modeling the Normal and Faulty Behaviour of Large-scale HPC Systems, " in *Proceedings of Parallel and Distributed Processing Symposium (IPDPS12)*, 2012, pp. 1168–1179.

[6] R. Ren, X. Fu, J. Zhan, and W. Zhou, "LogMaster: Mining Event Correlations in Logs of Large-scale Cluster Systems, " in *Proceedings of the IEEE Symposium on Reliable Distributed Systems (SRDS'12)*, 2012.

[7] C. Di Martino, S. Jha, W. Karmer, Z. Kalbarczyk, R. K. Lyer, "LogDiver: A Tool for Measuring Resilience of Extreme-Scale Systems and Applications, " in *Fault Tolerance for HPC at eXtreme Scale (FTXS) Workshop*, 2015, pp. 11–18.

[8] K.A. Huck and A.D. Malony, "PerfExplorer: A Performance Data Mining Framework For Large-Scale Parallel Computing, " In Proceedings of SC-15. ACM, November 2005.

[9] M. Casas, R.M. Badia, and J. Labarta, "Automatic Phase Detection and Structure Extraction of MPI Applications, " in *Journal International Journal of High Performance Computing Applications (IJHPCA*, 24(3):335–360, 2010.

[10] G. Bronevetsky, I. Laguna, S. Bagchi, B. R. de Supinski, D. H. Ahn and M. Schulz, "AutomaDeD: Automata-based debugging for dissimilar parallel tasks, " in *IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, Chicago, IL, 2010, pp. 231-240.

[11] 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 *Proceedings of Parallel and Distributed Processing Symposium (IPDPS11)*, 2011, pp. 840–851.

[12] Intrepd at Argonne (Blue Gene/P). [online]. Available: https://www.alcf.anl.gov/intrepid

[13] G. Lakner and B. Knudson, "IBM System Blue Gene Solution: Blue Gene/Q System Administration, " [online]. Available: http://www.redbooks.ibm.com/abstracts/sg247869.html.

[14] Y. Yuan, Y. Wu, Q. Wang, G. Yang and W. Zheng, "Job Failures in High Performance Scomputing Systems: A large-scale empirical study," in *Journal of Computers & Mathematics with Applications*, 63(2): 365–377, 2012.

[15] D. Tiwari et al., "Understanding GPU Errors on Large-scale HPC Systems and the Implications for System Design and Operation," in *Proceedings of IEEE 21st International Symposium on High Performance Computer Architecture (HPCA2015)*, Burlingame, CA, 2015, pp. 331-342.

[16] S. Gupta, D. Tiwari, C. Jantzi, J. Rogers and D. Maxwell, "Understanding and Exploiting Spatial Properties of System Failures on Extreme-Scale HPC Systems," in *Proceedings of 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN2015)*, Rio de Janeiro, 2015, pp. 37-44.

[17] Y. Liang, Y. Zhang, H. Xiong, R. Sahoo, "An Adaptive Semantic Filter for Blue Gene/L Failure Log Analysis, " in *Proceedings of Parallel and Distributed Processing Symposium (IPDPS07)*, 2007, pp. 1–8.

[18] Blue Waters. [online]. Available: http://www.ncsa.illinois.edu/enabling/bluewaters

[19] Gnuplot. [online]. Available: http://gnuplot.info.

[20] G. Hamerly and C. Elkan, "Alternatives to the K-means Algorithm That Find Better Clusterings, " in *Proceedings of the Eleventh International Conference on Information and knowledge management (CIKM2002)*, 2002, pp. 600–607.

[21] J. F. C. Kingman, "Poisson Processes (Oxford Studies in Probability), " *Oxford University Press*, 1993.

[22] H. T. Reynolds, "The Analysis of Cross-classifications, " *The Free Press*, New York, 1977.

[23] M. K. Goldberg, M. Hayvanovych, M. Magdon-Ismail, "Measuring Similarity between Sets of Overlapping Clusters, " in *Proceedings of the 2010 IEEE Second International Conference on Social Computing (SocialCom10)*, 2010, pp. 1–6.