Failure Diagnosis for Cluster Systems using Partial Correlations

Edward Chuah[‡‡], Arshad Jhumka[‖], Samantha Alt[††], R. Todd Evans[¶], Neeraj Suri[‡‡]
[‡‡]Lancaster University, Bailrigg, Lancaster LA1 4YW, UK. Email: {e.t.chuah, neeraj.suri}@lancaster.ac.uk
[*]The University of Warwick, Coventry CV4 7AL, UK. Email: H.A.Jhumka@warwick.ac.uk
[††]Intel Corporation. Email: samantha.alt@intel.com
[¶]Texas Advanced Computing Center, Texas 78758 USA. Email: rtevans@tacc.utexas.edu

***Abstract*—Failures have expensive implications in HPC (High-Performance Computing) systems. Consequently, effective diagnosis of system failures is desired to help improve system reliability from both a remedial and preventive perspective. As HPC systems conduct extensive logging of resource usage and system events, parsing this data is an oft advocated basis for failure diagnosis. However, the high levels of concurrency that exist in HPC systems cause system events to frequently interleave in time and, as such, certain interactions appear or become indirect. which will be missed by current failure diagnostics techniques. To help uncover such indirect interactions, in this paper, we develop a novel approach that leverages the concept of *partial correlation*. The novel failure diagnostics workflow - called IFADE - extracts partial correlation of resource use counters and partial correlation of system errors. As part of our contributions, we (a) compare our diagnostics approach with current ones, (b) identify two *previously unknown* causes of system failures, validated by system designers and (c) provide insights into Lustre I/O and segmentation faults. IFADE has been put on the public domain to support system administrators in failure diagnosis.**

***Index Terms*—HPC systems; Failure Diagnosis; Feature extraction; Partial correlation; Resource use data and system logs**

I. Introduction

Compute-intensive applications such as scientific applications and machine learning are typically executed on resource-rich platforms such as HPC systems and data centers. However, due to the size and complexity of these systems, they encounter failures on a regular basis [1], [2]. The trend towards exascale computing is only going to exacerbate the failure rate of these systems, which can lead to a significant reduction in system availability, as reported in recent large-scale studies of HPC system failures [2]–[5] and data center hardware failures [6].

These systems record operations and resource use data [7], [8] in relevant logs. However, these logs suffer from several limitations [8]: (a) the data is often unstructured, (b) they often miss or do not contain sufficient information to enable a complete causal trace to system failures and (c) often contain redundant information about an event. These system and resource logs may represent the only source of information pertaining to the state of the system execution. Partly due to these reasons, approaches for error detection or failure diagnosis are unsupervised in nature, i.e., these techniques are not based on accurate labelling of system execution.

In the absence of detailed fault models viable for HPC systems, failure diagnosis, e.g., [9], represents a reasonable first step towards achieving this, by linking errors and failures. However, the scale of concurrency in such systems and the nature of the logs make it challenging to obtain a correct causal trace to system failures. As such, it becomes as important to capture *indirect* relationships between system components, as it is for direct relationships. However, state-of-the-art failure diagnostics approaches typically do direct correlation of errors to system failures [9]–[11]. For example, the diagnostics workflow developed in [11] may return a diagnosis of the form: "*The filesystem was executing I/O and the CPU was waiting on the filesystem I/O to complete which led to a node crash*". However, given the two events (i.e., variables) *"filesystem I/O"* and *"CPU wait"*, it is important to understand the role of any third event such as *"data write in memory"* in the relation between the first two events.

As such, we leverage the power of *partial correlation*, which is a statistical technique that captures the strength of a relationship between two variables while controlling for the effect of one or more other variables, to develop IFADE (**I***n-depth* **FA***ilure* **D***iagnosis fram***E***work*), a novel approach that identifies partial correlations between system components or events to provide detailed diagnosis of system failures (i.e., node crash or OS hang-up). IFADE makes use of the system logs [12], [13] and resource use data [14] for its analysis.

Our main contributions reported in this paper are as follows:

- We describe and implement IFADE, a novel diagnostics workflow that executes a number of unsupervised learning techniques such as feature extraction on unstructured log data of HPC systems to ascertain partial correlations for failure diagnosis.
- We diagnose system failures without prior knowledge of a fault model. IFADE automatically extracts the important resource use counters and system events. Then, it uses them to identify partial correlation between resource use counters and partial correlation between system errors.
- We compare IFADE with current failure diagnostics techniques on actual system problem cases. IFADE identified: (i) a previously unknown memory data update cause of Lustre I/O which led to compute node soft lockups, (ii) a previously unknown corrupted memory index cause of segmentation faults which led to compute node soft lockup.
- To yield accurate diagnosis, we show that applying statistical significance tests results in a low probability of identifying a false cause of the system failure.

Paper outline: Section II presents the system and issue of fault models. The insight behind IFADE and details of its process are described in Section III. We evaluate IFADE on a series of cases in Section IV followed by the lessons learned in Section V. We analyze the related work in Section VI and conclude with a summary and future work in Section VII.

II. System & Fault Models

IFADE is based on a generic HPC cluster model [15]. The model is comprised of jobs, nodes, a job scheduler and software components such as a filesystem and an operating system. The

job scheduler assigns jobs to nodes. The filesystem may transfer data to and from the nodes. The jobs, nodes and job scheduler may produce system event data such as: (a) resource use logs that contain the job number, node number and time-stamps, (b) system logs that contain the job number, node number and time-stamps. This system model covers the spread of open-source Linux clusters and vendor-specific systems, e.g., IBM and Cray.

Fault Model: It is unrealistic to associate a single fault model to a HPC computing environment. This transpires as there are various discrete fault models one may consider. For example, one may consider faults occurring at node level (e.g., node crash), at component level (e.g., network state) or at an aggregate cluster level (e.g., interaction between different nodes). When a fault occurs in a component, the resulting error can propagate beyond the component's interface and affect multiple system components. These can occur at any level in the system, for example memory, operating system or the job. When these errors lead to the unavailability of resources to jobs, a failure occurs.

Consequently, the system events leading to a failure (e.g., OS hang, node crash) are interleaved among thousands of other system events, and further the process of extracting the sequence of events is largely ad-hoc and manual. As such, it is very difficult to determine the exact cause of the failure, which may be exacerbated due to potentially multiple and different fault models associated with various components.

III. The IFADE Approach

We present a motivating example to illustrate the insight behind the IFADE approach. For a HPC system, consider the values for three resource use counters (`CPU 0 iowait`, `llite /work ioctl`, `mem 0 Writeback`) as depicted in Fig. 1.

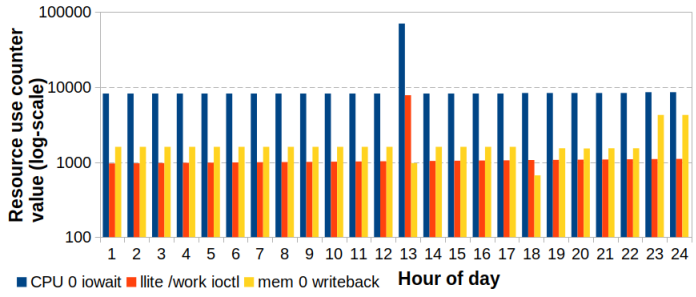


Fig. 1. Values of memory writeback (mem 0 Writeback), CPU I/O wait (CPU 0 iowait) and Lustre I/O control (llite /work ioctl) resource use counters.

From Fig. 1, we observed that the values for CPU I/O wait and Lustre I/O control counters are the same from hour 1 to hour 12. Then, their values increased on hour 13. On hour 14, their values decreased. Since the values for the CPU I/O wait and Lustre I/O control counters increased and decreased together, a strong positive correlation between CPU I/O wait and Lustre I/O control counters is possible. We also observed that the value for the memory writeback counter increased on hours 23 and 24. Since the value for the memory writeback counter did not increase with the CPU I/O wait and Lustre I/O control counters, it is very unlikely that a strong positive correlation of the memory writeback counter to the CPU I/O wait and Lustre I/O control counters will be observed. Due to a perceived delay in a similar behaviour change for the memory writeback, it is important to ensure that any potential impact of the memory writeback operation on CPU wait and Lustre I/O operations be accounted for. One important technique that can help with controlling the effect of a "third party" variable while studying the strength of a relationship between two random variables is partial correlation.

Thus, our goal is to identify which resource use counters and message types which are partially correlated to other counters and message types respectively. In this paper, we seek to identify patterns of counters and message types for failure diagnosis. The problem we address is described as follows: Given (a) resource use data and system logs, (b) lists of counters and message types, (c) lists of correlated counters and correlated message types, and (d) number of dates:

- Q1 Can we identify the "significant" groups of resource use counters and message types?
- Q2 Can we identify the resource use counters that are strongly partially correlated to other counters?
- Q3 Can we identify the message types that are strongly partially correlated to other message types.

By "significant", we mean the counters and message types that are assigned the highest scores by the feature extractors and these are the counters and message types that are important. By focusing on the significant items, we attempt to identify those items which are deemed more impactful.

Thus, the IFADE workflow, as depicted in Fig. 2, is a progression of three phases where the log data needs to be structured or clustered into "significant" groups. We term this the phase of *Data Preprocessing* [Q1]. The identification of the resource counters [Q2] is the next phase of *Feature Extraction* that utilizes unsupervised learning to parse the [Q1] data. This phase is analogous to "extracting" the fault model from the log data. These phases consequently lead to conducting the phase of *Partial Correlation* [Q3] to results in the diagnosis. In a data matrix, each row represents a counter or message type, each column represents a time window and each cell contains the count of a counter or message type. On this background, we now detail each of IFADE phases.

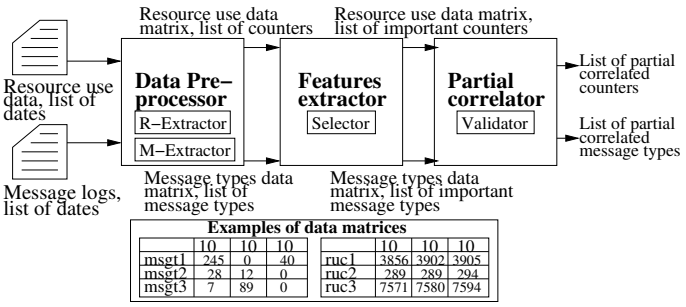


Fig. 2. The IFADE workflow. It is comprised of three modules: (a) Data preprocessor, (b) Features extractor and (c) Partial correlator.

A. Data Preprocessor

The objective of data preprocessing is to present the resource use data and system logs in a standardized form on which analysis algorithms can be applied. To achieve this, we need to address three aspects: (a) the structure of a system log on two

HPC systems are different, (b) the resource use data and system logs are monitored by multiple tools, (c) the time granularities in the resource use data and system logs are different.

The resource use data contains all values of all the counters which are aggregated at a default time interval of 10 minutes. The TACC_Stats resource usage monitor [14] performs the monitoring online and the value for the counters is set to zero only when a node is reset. A resource use log is given below:

```
20665 Aug 16 15:20:01 i151-312 eth0 rx_bytes
424689 tx_bytes 25178
```

The resource use log contains the following fields: job (`20665`), time-stamp (`Aug 16 15:20:01`), node (`i151-312`), device group (`eth0`) and pairs of counter and its value (`rx_bytes 424689`, `tx_bytes 25178`). The resource use data contains 410 counters that are divided into 9 resource use counter groups. In [16], it was reported that a longer diagnostics window results in a higher probability of observing a failure or recovery event. Thus, we group the counters into time-bins of 10, 30 and 60 minutes. A time-bin is a window of one fixed time interval. Next, we describe the process to generate a resource use data matrix as follows: (a) group the resource use logs into 10, 30 and 60 minute time-bins, (b) extract the counters in the resource use logs, remove all repeated counters and store the unique counters in a list, (c) obtain the difference in the value for a counter between two consecutive resource use logs and add the difference to the preceeding value by 10, 30 and 60 minute bins. We implemented the process in the R-Extractor.

A system log typically contains a date and time, node number and message. Some system logs do not contain a job number. Furthermore, the format of a system log on one system may be different from the system log on another system. Having said that, most system logs contain three basic fields: (a) date and time, (b) node number, and (c) message. Therefore, we implement a system log-reformator to convert system logs that contain the three basic fields into a standard form. If a system log does not contain a field in the standard formatted log, then we use a default placeholder. The standard formatted system log has the following fields:

```
job number, month, day, time-stamp (hour:minute
:second), node number, software name, message.
```

The system logs contain log-entries which are generated by the operating system, filesystem and systems software. An example of a system log-entry is given below:

```
2018634 Jul 5 09:08:28 i117-406 kernel X $<$0$>$
%s. ECC error ECC error K8
```

A system log contains a job number (`2018634`), time-stamp (`Jul 5 09:08:28`), node number (`i117-406`), software name (`kernel`) and unstructured message part (`<0>\%s. ECC error ECC error K8`). The message part is comprised of a sequence of English-only words and alpha-numerical words. To obtain the *Constants* in the message part, we extract the English-only words. One month's worth of system logs contains 1,676 unique message types on average.

As was done with the resource use logs, we group the system logs into 10, 30 and 60 minute time-bins. Next, we describe the process to generate a message types data matrix: (a) group the system logs into time-bins of 10, 30 and 60 minutes, (b) extract the constants, remove all the repeated ones and store the unique constants termed a *message type* in a list and (c) count the occurrence of the message types by 10, 30 and 60 minute bins. We implemented the process in the M-Extractor.

B. Features Extractor

We need to identify which resource use counters or message types are important. There are hundreds of resource use counters and thousands of message types in the data matrices (see Section III-A), and further these are unlabeled. Thus, this is a challenging task. To address the challenge, we apply feature extraction methods to identify the important resource use counters and message types without the need for labeled data.

Feature extraction methods: Feature extraction methods such as Principal Component Analysis (PCA) belong to a class of unsupervised dimensionality reduction techniques. They are used to reduce the number of redundant counters and system events for analysis. Feature extraction methods are showed to be effective in detecting anomalous nodes in HPC systems [17], [18] and detecting anomalies in cloud computing systems [19]. Feature extraction methods do not require *a priori* knowledge of the data labels. Thus, they are suitable for unlabeled datasets.

Mechanism for extracting features: PCA is a widely used feature extraction technique. However, it can miss important system events and produce a low coverage as reported in [20]. Other feature extraction techniques such as Independent Component Analysis (ICA) was shown to perform better [17]. Both PCA and ICA extract features that are linearly uncorrelated which limits them to linear components. On the other hand, non-linear feature extractors such as non-linear PCA generalized the principal components and showed superior performance [21]. Thus, to meet our composite needs of feature extractions while maximizing variance, minimizing noise and data cleaning, our Feature Extractor uses multiple feature extraction methods as: (a) robust Principal Component Analysis (PCA) [22], (b) fast Independent Component Analysis (ICA) [23], (c) a neural network-based autoencoder (NLPCA) [24], (d) kernel Principal Component Analysis (KPCA) [25]. PCA reduces the set of redundant features by maximizing the variances. If a dataset containing outliers is input into PCA, it can produce a biased result. To solve this problem, we conducted data cleaning in the following manner: (a) transpose the data matrix, (b) normalize the values for all the counters to range between 0 to 10 and (c) adjust the columns in the data matrix to have zero-mean. We input the normalized zero-mean transposed data matrix into PCA and NLPCA as they require that the feature vectors are represented by columns. Differently to PCA, ICA reduces the set of redundant features by first removing noise in the data. Then, a whitening process is applied to remove any correlations in the data. ICA and KPCA requires that the feature vectors in the data matrix are represented by rows. We input the normalized data matrices into ICA and KPCA as they require that the feature vectors are represented by rows. We apply multiple feature extractors due to multiple fault models and each fault model is different. It was reported in [17] that the first principal component contains the largest variance. Hence, we extract the scores of the supplied data on the first principal component. To select the important counters and message types, we apply a feature selection process that filters out features with high absolute values at very low computational cost [26]. We implemented the process in the Selector.

C. Partial Correlator

We now need to identify which resource use counter or message type has an indirect relationship to other counters or message types. To achieve this, we need to determine the strength of the relationship between an important variable and other variables in the data. To address this issue, we apply partial correlation to obtain the strength of the relationship between an important variable and other variables.

Partial correlation method: Partial correlation obtains the correlation coefficient of two variables after controlling for one or more variables. Partial correlation can uncover spurious relationships and identify indirect relationships. Bivariate correlation is used to check if two variables are related to one another. It measures how the two variables change together at the same time and obtain the correlation coefficients. In contrast, partial correlation obtains the correlation coefficients of two variables after controlling for a third variable. Partial correlation was showed to be effective in identifying indirect interactions between system components [27].

Mechanism for identifying partial correlations: The partial correlation coefficient is defined as follows [28]:

$$\rho xy.z = \frac{\rho xy - \rho xz.\rho yz}{\sqrt{1-\rho^2 xz} \times \sqrt{1-\rho^2 yz}} \quad (1)$$

where z is one mediating variable, ρxy, ρxz and ρyz are the Pearson or Spearman-Rank correlation coefficients between x and y, x and z, y and z. The Pearson correlation coefficient, r is defined as the mean of the products of the standard scores [29]: $r = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{x_i-\bar{x}}{s_x}\right)\left(\frac{y_i-\bar{y}}{s_y}\right)$, where $\left(\frac{x_i-\bar{x}}{s_x}\right)$ is the standard score of x, $\left(\frac{y_i-\bar{y}}{s_y}\right)$ is the standard score of y, x and y are two datasets containing n values of a pair of counters or a pair of message types, $s_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2}$ is the sample standard deviation of x, $s_y = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(y_i-\bar{y})^2}$ is the sample standard deviation of y, $\bar{x}$ and $\bar{y}$ is the sample mean of x and y. The Spearman-Rank correlation coefficient ρ is defined as the Pearson correlation coefficient between the ranks of a pair of variables [29]. There are other methods available but these methods assume that the variables in the dataset are independent and identically distributed. Thus, partial correlation is a suitable method for achieving our objective.

To obtain the partial correlation coefficients, we implemented the following process: Given three counters or message types x, y and z: (a) obtain the Pearson or Spearman-Rank correlation coefficients ρxy, ρxz and ρyz, (b) input ρxy, ρxz and ρyz into equation (1). ρxy, ρxz and ρyz requires the pair of counters or message types to contain the same number of data points on the x-axis. However, the timestamps for the resource use data and system logs may be different. Resource usage are generated at regular fixed time intervals. System logs are generated at irregular time intervals. Thus, we do not correlate resource use counters with message types. After the partial correlation coefficients are obtained, we generate the lists of partial correlated counters and partial correlated message types. To interpret the strength of the partial correlation coefficient, we apply the following rules [29]: (a) 0.8 to 1: strong positive partial correlation, (b) 0.3 to 0.79: moderate positive partial correlation, (c) 0.1 to 0.29: weak positive partial correlation.

Significance testing: After the partial correlation coefficients are obtained, we need to test the significance of the correlation coefficient. Fisher's z-transform is a standard technique for testing the significance of a correlation coefficient [29]. We use the following terminology to define the null and alternate hypotheses [30]. The null hypotheses are: (a) an important counter is weakly positive partial correlated to a pair of correlated counters, (b) an important message type is weakly positive partial correlated to a pair of correlated message types. The alternate hypotheses are: (a) an important counter is strongly positive partial correlated to a pair of correlated counters, (b) an important message type is strongly positive partial correlated to a pair of correlated message types. Then, we obtain the z-scores for all partial correlation coefficients. When the absolute value of z is large, e.g., $z = 2.64$ at 99% confidence level, we will reject the null hypothesis in favour of the alternate hypothesis.
Handling false positives: When multiple hypotheses are tested, the probability that there is at least one false positive due to chance increases. For example, if we have 26 hypotheses and obtained a P-value of 0.01 for each test, the false positive rate is $1-(1-0.01)^{26} = 1-0.99^{26} = 0.22$ or 22%. To solve the problem, we apply a standard technique called the Bonferroni correction [31]. It obtains an adjusted P-value by multiplying the unadjusted P-value by the number of tests. We implemented the Bonferroni correction in the Partial Correlation module. The process is as follows: (a) we map all z-scores to P-values using a Z-table, (b) given d number of dates we multiply all the unadjusted P-values by d and obtained the adjusted P-values. We implemented the Z-table in the Validator.

IV. Evaluation on HPC Systems

We conducted diagnosis on two widely deployed HPC systems termed System X and System Y. System X has 4,048 nodes that provide batch job processing and data storage services. System Y has 1,888 nodes that provide batch job processing and data storage services. Many data centers operate large numbers of compute nodes and they also provide services such as batch job processing and data storage.

The resource usage of HPC jobs, along with system logs, are monitored on most HPC systems [32]. For IFADE's validation, we obtained: (a) two years worth of system logs and six months worth of resource use data on System X and (b) one year worth of system logs and two months worth of resource use data on System Y. However, we do not know which dates contain the compute node soft lockup events. Therefore, we randomly selected a range of dates on System X and System Y. A summary of log-data analyzed is given in Table I.

TABLE I
Summary of log-data analyzed on System X and System Y.

System		Resource use data		System logs	
	Days	Size	Qty. lines	Size	Qty. messages
System X	26	124.1 GB	637,860,203	9.6 GB	64,822,682
System Y	31	46.6 GB	207,068,692	1.3 GB	12,267,629

It was reported in [12] that compute node soft lockups is one of the most common problems for the HPC systems administrator. To identify the dates of compute node soft lockups, we implemented a function in IFADE to scan the system logs for messages containing the keywords `soft lockup` and extract the

dates of soft lockup messages. When Linux hang, it generates a soft lockup message. We identified: (a) 12 soft lockup dates on System X and (b) 7 soft lockup dates on System Y.

The Lustre filesystem is commonly used to provide high-speed data I/O on many HPC systems. However, I/O problems on Lustre has been widely reported [33]. To identify the dates when Lustre experienced I/O problem, we obtain the lists of partial correlated counters and partial correlated messages and identified cases of I/O problems on Lustre. We found other system problems though we focus on a subset of these cases as: (a) Lustre I/O and (b) segmentation faults.

A. Problem Case 1: Lustre Filesystem I/O

When Lustre is performing I/O, the CPU waits on the I/O operations to complete. When the CPU is waiting, data in the memory is updated. In most cases, the memory data update is completed successfully. However, on a rare occasion, the memory data update may fail leading to a compute node lockup.

1) Phase 1: Identify Time-bins of Important Resource Use Counter Groups: In the first phase of IFADE, we identify the time-bins associated with the largest number of important resource use counter groups on all the dates.

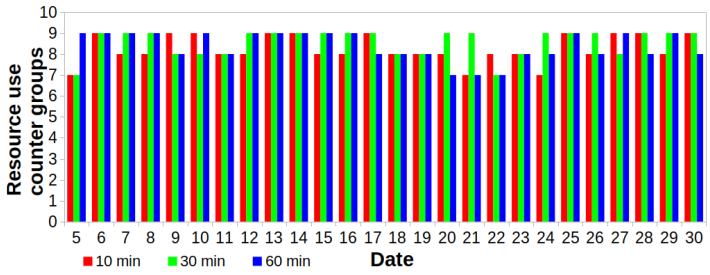


Fig. 3. Number of important resource use counter groups identified on multiple granularities of time-bins on System X.

Fig. 3 shows the number of groups of important counters identified on 10, 30 and 60 minute time-bins in System X resource use data. We observed that the largest number of counter groups were identified on multiple granularities of time-bins. The largest number of groups of counters were identified: (i) on 30 or 60 minute time-bins on 24 out of 26 dates and (ii) on 10 minute time-bins on 2 out of 26 dates. It took 7 minutes to generate the reports.

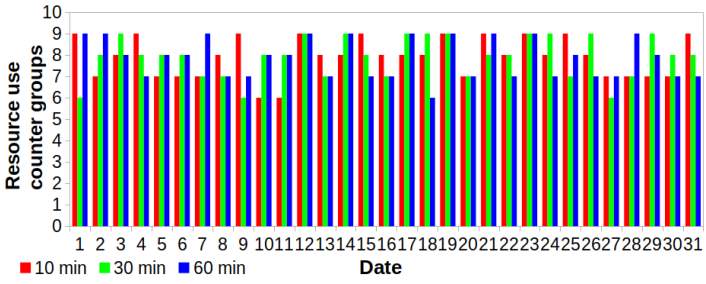


Fig. 4. Number of important resource use counter groups identified on multiple granularities of time-bins on System Y.

As was done with System X counters groups described in Fig. 3, we obtained the number of counter groups on 10, 30 and 60 minute time-bins on System Y. Fig. 4 shows the number of important counters groups on System Y. We observed that the largest number of important counters groups were identified on multiple granularities of time-bins. The largest number of groups of counters were identified: (i) on time-bins of 30 or 60 minutes on 23 out of 31 dates and (ii) on 10 minute time-bins on 8 out of 31 dates. It took 5 minutes to generate the reports.

> The largest number of resource use counter groups were obtained on 10, 30 and 60 minute time-bins, indicating the importance of using both short and long time-bins.

2) Phase 2: Extract the Important Resource Use Counters: The first phase of IFADE is characterized by the identification of the time-bins for identifying the largest number of important resource use counter groups. However, the objective of feature extraction is to extract the important counters to analyze. To achieve this, we need to determine which feature extractor extracted the largest number of counters.

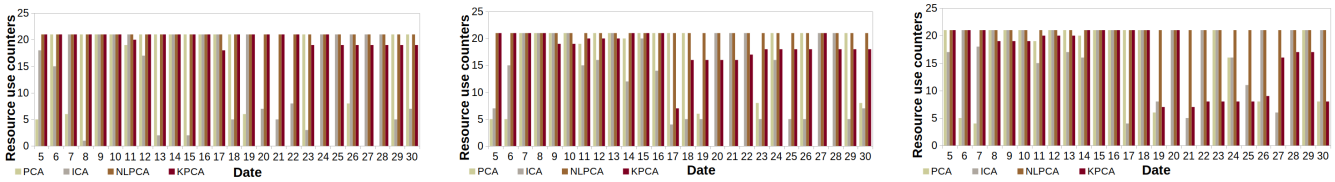
Thus, in the second phase of our study we obtained the number of counters extracted by the Features Extraction module. Fig. 5 shows the number of important resource use counters identified on System X. From Fig. 5(a), Fig. 5(b) and Fig. 5(c), we observed that NLPCA extracted the largest number of counters on 10, 30 and 60 minute time-bins for all the 26 dates. PCA and ICA took 2 seconds to execute. NLPCA and KPCA took 78 seconds to execute. The important counters represent 6% of all the counters on average.

As was done with System X resource use counters, we obtained the number of important counters on System Y. Fig. 6 shows the number of important counters on System Y. From Fig. 6(a), Fig. 6(b) and Fig. 6(c), we observed that NLPCA extracted the largest number of counters on 10, 30 and 60 minute time-bins for all the 31 dates. PCA and ICA took 1.2 seconds and NLPCA and KPCA took 53 seconds to execute. The important counters represent 6% of all counters on average.

> The non-linear PCA extractor extracted the largest number of important counters on all 26 dates on System X and all 31 dates on System Y. This confirms that PCA provided a low coverage as reported in [20].

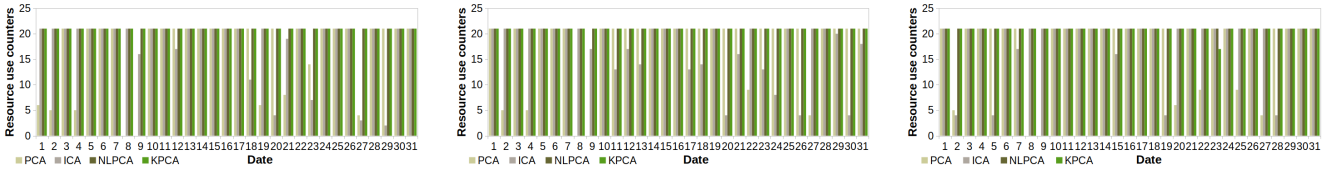
3) Phase 3: Identify Partial Correlation of Resource Use Counters: The second phase of IFADE is characterized by extracting the largest number of important counters. However, our goal is to identify partial correlation of multiple groups of counters. Therefore, we need to identify a relationship between an important counter and other counters.

Thus, we scan the lists of important counters on System X. We identified many important counters. However, we focused on the memory writeback counter which form the majority of important counters. The counter `llite /work ioctl` is incremented when Lustre performs I/O operations. We scan the lists of partial correlated counters and obtained the partial correlation score of memory writeback to CPU I/O wait and Lustre I/O control counters. It took 3 seconds to generate the partial correlation reports. The partial correlation score is shown in Fig. 7. We observed that the memory writeback counter is strongly positive partial correlated to CPU I/O wait and Lustre I/O control counters with a partial correlation score



(a) 10 minute time-bins. (b) 30 minute time-bins. (c) 60 minute time-bins.

Fig. 5. Number of important resource use counters extracted by PCA, ICA, NLPCA and KPCA on System X.



(a) 10 minute time-bins. (b) 30 minute time-bins. (c) 60 minute time-bins.

Fig. 6. Number of important resource use counters extracted by PCA, ICA, NLPCA and KPCA on System Y.

ranging from 0.8 to 0.99 on 7 dates. To determine if those dates coincide with the dates of compute node soft lockups, we manually scanned the message logs to identify soft lockup events. We identified 3 dates that coincided with the dates of the strongly positive partial correlated counters. We also observed that the memory writeback counter is not strongly positive partial correlated to CPU I/O wait and Lustre I/O control counters on 5 dates, indicating that memory data update is not a cause of compute node soft lockups. Therefore, there is another cause of compute node soft lockups.

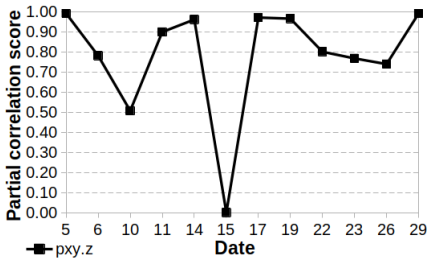


Fig. 7. Partial correlation of mem Writeback (z), cpu iowait (x) and llite /work ioctl (y) resource use counters on System X.

As was done with System X counters, we scan the lists of important counters on System Y. We identified many important counters. However, we focused on the memory writeback counter which form the majority of important counters. We scan the lists of partial correlated counters and obtained the partial correlation score of memory writeback to CPU I/O wait and Lustre I/O control counters. It took 2 seconds to generate the partial correlation reports. The partial correlation score is shown in Fig. 8. We observed that the memory writeback counter is strongly positive partial correlated to CPU I/O wait and Lustre I/O control counters with a partial correlation score ranging from 0.88 to 1 on 19 dates. To determine if those dates coincide with the dates of compute node soft lockups, we manually scanned the message logs to identify soft lockup events. We identified 4 dates that coincided with the dates of the strongly positive partial correlated counters.

Failure diagnostics approaches such as those presented in [9]–

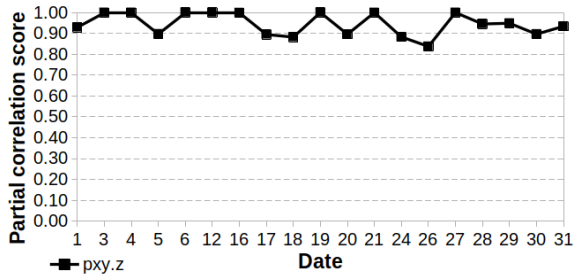


Fig. 8. Partial correlation of mem Writeback (z), cpu iowait (x) and llite /work ioctl (y) resource use counters on System Y.

[11] have adopted Pearson correlation [9], [10] and Spearman-Rank correlation algorithms [11]. We implemented these diagnostics techniques based on our objective to identify partial correlation of resource use counters and applied the diagnostics techniques on the memory writeback, CPU I/O wait and Lustre I/O control counters on System X and System Y. The correlation scores are given in Tables II and III. From Table II, we observed that memory writeback counter is strongly positive correlated to CPU I/O wait and Lustre I/O counters for 1 out of 7 dates on System X and 1 out of 19 dates on System Y. From Table III, we observed that memory writeback counter is strongly positive correlated to CPU I/O wait and Lustre I/O counters for 1 out of 7 dates on System X. Correlation of memory data update and Lustre I/O were identified on a small number of dates only.

> While bivariate correlation-based failure diagnostics techniques can identify the dates of soft lockups which are caused by memory data updates, they also missed other dates. IFADE identified the soft lockups which are *indirectly* caused by memory data updates on 2 more dates on System X and 3 more dates on System Y.

Detailed diagnosis: On System X, data in the memory was updated when the Lustre filesystem is performing I/O and compute node soft lockups occurred on 2 dates, showing that memory data update *indirectly* caused the soft lockups on the 2 dates. On System Y, data in the memory was updated

TABLE II
CORRELATION SCORE OF [9], [10] ON MEM WRITEBACK, CPU IOWAIT AND LLITE IOCTL COUNTERS ON SYSTEMS X AND Y.

System X		Dates						
Counter	**Counter**	**5**	**11**	**14**	**17**	**19**	**22**	**29**
mem Writeback	Lustre ioctl	-0.07	-0.4	-0.59	0.8	0.39	0.4	-0.45
CPU iowait	Lustre ioctl	0.99	0.91	0.92	0.99	0.97	0.83	0.99
mem Writeback	CPU iowait	-0.08	-0.34	-0.31	0.8	0.4	0.38	-0.4
System Y		**Dates**						
Counter	**Counter**	**1**	**3**	**4**	**5**	**6**	**12**	**16**
mem Writeback	Lustre ioctl	-0.28	-0.13	-0.24	-0.32	-0.18	0.97	-0.48
CPU iowait	Lustre ioctl	0.93	1	1	1	1	1	1
mem Writeback	CPU iowait	-0.27	-0.13	-0.24	-0.32	-0.18	0.97	-0.48
		17	**18**	**19**	**20**	**21**	**24**	**26**
mem Writeback	Lustre ioctl	0.08	0.08	-0.09	-0.26	-0.5	-0.17	-0.18
CPU iowait	Lustre ioctl	0.89	0.88	1	0.89	1	0.88	0.82
mem Writeback	CPU iowait	-0.02	0.06	-0.09	-0.1	-0.5	-0.12	-0.01
		27	**28**	**29**	**30**	**31**		
mem Writeback	Lustre ioctl	-0.42	-0.38	-0.23	-0.17	0.17		
CPU iowait	Lustre ioctl	1	0.95	0.95	0.9	0.94		
mem Writeback	CPU iowait	-0.42	-0.35	-0.18	-0.2	0.15		

TABLE III
CORRELATION SCORE OF [11] ON MEM WRITEBACK, CPU IOWAIT AND LLITE IOCTL COUNTERS ON SYSTEMS X AND Y.

System X		Dates						
Counter	**Counter**	**5**	**11**	**14**	**17**	**19**	**22**	**29**
mem Writeback	Lustre ioctl	-0.39	-0.34	-0.57	0.79	0.44	0.37	-0.41
CPU iowait	Lustre ioctl	0.83	0.91	0.96	0.99	0.97	0.83	0.99
mem Writeback	CPU iowait	-0.31	-0.4	-0.57	0.79	0.44	0.4	-0.45
System Y		**Dates**						
Counter	**Counter**	**1**	**3**	**4**	**5**	**6**	**12**	**16**
mem Writeback	Lustre ioctl	-0.48	-0.14	-0.24	-0.34	0.1	0.13	-0.56
CPU iowait	Lustre ioctl	0.99	0.99	0.99	0.99	0.99	0.35	0.99
mem Writeback	CPU iowait	-0.48	-0.14	-0.23	-0.34	0.1	0.35	-0.56
		17	**18**	**19**	**20**	**21**	**24**	**26**
mem Writeback	Lustre ioctl	-0.32	-0.2	-0.02	-0.37	-0.44	-0.43	-0.11
CPU iowait	Lustre ioctl	0.99	0.99	0.99	0.99	0.99	0.99	0.99
mem Writeback	CPU iowait	-0.32	-0.2	-0.02	-0.37	-0.44	-0.43	-0.11
		27	**28**	**29**	**30**	**31**		
mem Writeback	Lustre ioctl	-0.48	-0.49	-0.28	-0.14	0.08		
CPU iowait	Lustre ioctl	0.99	0.99	0.99	0.99	0.99		
mem Writeback	CPU iowait	-0.48	-0.48	-0.28	-0.14	0.08		

when Lustre is performing I/O and compute node soft lockups occurred on 3 dates, showing that memory data update *indirectly* caused the soft lockups on the 3 dates.

Significance test: We test all the partial correlation coefficients against the null hypothesis and obtained the z-scores for all the partial correlation coefficients. On System X, the z-scores range from 3.58 to 12.13. At 99% confidence level, under the null hypothesis $z_{0pr} = 2.64$. Hence, we reject the null hypothesis in favour of the alternate hypothesis. On System Y, the z-scores range from 4.85 to 12.13. At 99% confidence level, under the null hypothesis $z_{0pr} = 2.64$. Hence, we reject the null hypothesis in favour of the alternate hypothesis.

False positives: Next, we determine the probability of rejecting the null hypothesis when it is true. We apply a one-sided test and use the significance level, $\alpha = 0.01$ for all given hypothesis tests to obtain a P-value on System X and System Y. On System X, the lowest z-score is 3.58. Since this is a one-sided test, the P-value is 0.00017. We obtained the adjusted P-value $0.00017 \times 26 = 0.00442$ where 26 is the number of dates. The adjusted P-value is less than 0.01, indicating it is highly unlikely this result would be observed under the null hypothesis. The z-score for all the partial correlation coefficients are greater than or equal to 3.58 and the adjusted P-values are less than 0.01, indicating it is highly unlikely these results would be observed under the null hypothesis.

As was done with System X z-scores, we obtained the z-scores on System Y. The lowest z-score is 4.85. Since this is a one-sided test, the P-value is 0.00001. We obtained the adjusted P-value $0.00001 \times 31 = 0.00031$ where 31 is the number of dates. The adjusted P-value is less than 0.01, indicating it is highly unlikely this result would be observed under the null hypothesis. The z-score for all the partial correlation coefficients are greater than or equal to 4.85 and the adjusted P-values are less than 0.01, indicating it is highly unlikely these results would be observed under the null hypothesis.

B. Problem Case 2: Segmentation Faults

When data in memory is corrupted, the ECC memory attempts to correct the error. In most cases, the corrupted data is resolved successfully. However, if the ECC memory fails to correct the data corruption, then the data value is changed. When the data value is a memory index and a program uses the index to locate data in memory, a memory access violation occurs.

1) Phase 1: Identify Time-bins of Important Message Types: In the first phase, we identify the time-bins associated with the largest number of important message types on all the dates.

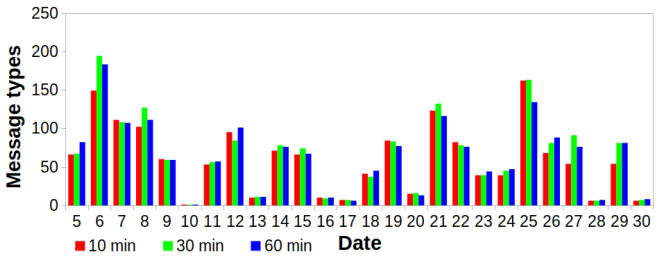


Fig. 9. Number of important message types identified on multiple granularities of time-bins on System X.

Fig. 9 shows the number of important message types identified on 10, 30 and 60 minute time-bins in System X system logs. We observed that the largest number of important message types were identified on multiple granularities of time-bins. These message types were identified: (i) on 30 or 60 minute time-bins on 22 out of 26 dates and (ii) on 10 minute time-bins on 4 out of 26 dates. The reports were generated in 4 minutes.

As was done with System X system logs described in Fig. 9, we obtained the important message types on System Y. Fig. 10

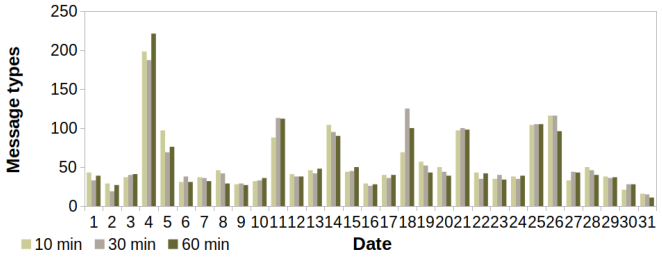


Fig. 10. Number of important message types identified on multiple granularities of time-bins on System Y.

shows the number of important message types. We observed that the largest number of important message types were identified on multiple granularities of time-bins. The largest number of important message types were identified: (i) on 30 or 60 minute time-bins on 18 out of 31 dates and (ii) on 10 minute time-bins on 13 out of 31 dates. The reports were generated in 3 minutes.

> The largest number of important message types were obtained on time-bins of 10, 30 and 60 minutes, indicating the importance of using both short and long time-bins.

2) Phase 2: Extract the Important Message Types: The first phase of IFADE is characterized by the identification of the time-bins for identifying the largest number of important message types. However, the objective of feature extraction is to extract the important message types to analyze. Therefore, we need to determine which feature extractor extracted the largest number of important message types.

Thus, in the second phase of our study, we obtained the number of important message types extracted by the Features Extraction module. Fig. 11 shows the number of important message types extracted on System X. From Fig. 11(a), Fig. 11(b) and Fig. 11(c), we observed that NLPCA extracted the largest number of important message types on all the 26 dates. PCA and ICA took 3 seconds and NLPCA and KPCA took 78 seconds to execute. The important message types represent 23% of all the message types on average.

As was done with System X message types, we obtained the number of important message types on System Y. Fig. 12 shows the number of important message types. From Fig. 12(a), Fig. 12(b) and Fig. 12(c), we observed that NLPCA extracted the largest number of important message types on all the 31 dates. PCA and ICA took 1.5 seconds and NLPCA and KPCA took 51 seconds to execute. The important message types represent 18% of all the message types on average.

> The non-linear PCA extractor extracted the largest number of important message types on all 26 dates on System X and all 31 dates on System Y. This confirms that PCA provided a low coverage as reported in [20].

3) Phase 3: Identify Partial Correlation of Message Types: IFADE's second phase is characterized by extracting the largest number of important events. However, our goal is to identify partial correlation of system errors. Thus, we need to identify a relationship between an important event and other events.

Thus, we scan the lists of important message types on System X. We identified many important message types. However, ECC (Error Correcting Code) events forms the majority of important message types in the list. Therefore, we focused on the ECC error message on System X. We also scanned the lists of important message types on System Y. There were no ECC error messages in the list of important message types on System Y. When a program tries to access data in an invalid memory location, a `segfault` message is generated. We obtained the partial correlation score of ECC error, segfault and program error messages. It took 1.2 seconds to generate the reports. From Fig. 13(a), we observed that ECC error is strongly positive partial correlated to segmentation fault and general protection fault (GPE) with a partial correlation score ranging from 0.81 to 1 on 4 dates. To determine if those dates coincide with the dates of compute node soft lockups, we manually scanned the message logs to identify soft lockup events. We identified 4 dates of soft lockups that coincided with the strongly positive partial correlated events. From Fig. 13(b), we observed that ECC error is strongly positive partial correlated to segfault and Lustre communication errors with a partial correlation score ranging from 0.87 to 1 on 3 dates. To determine if those dates coincide with the dates of compute node soft lockups, we manually scanned the message logs to identify soft lockup events. We identified 3 soft lockup dates that coincided with the strongly positive partial correlated events.

Next, we apply the diagnostics techniques in [9]–[11] to identify correlations of ECC errors, segfault, GPE and Lustre communication errors. The correlation scores are given in Tables IV and V. From Table IV, we observed that ECC error is weakly correlated to GPE, segfault and Lustre communication error. From Table V, we observed that ECC error is weakly correlated to GPE, segfault and Lustre errors. Correlation of ECC errors and segfault were not identified on all 7 dates.

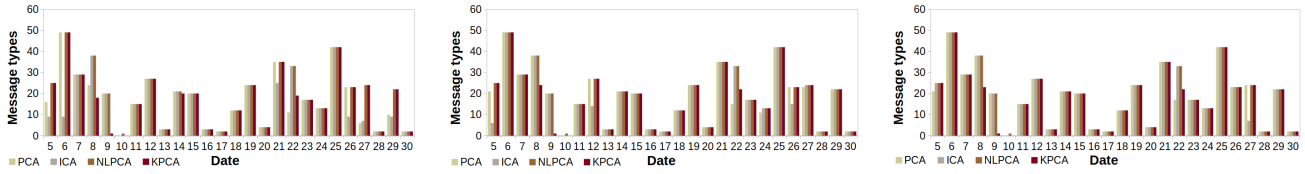
TABLE IV
CORRELATION SCORE OF [9], [10] ON ECC ERROR, SEGFAULT, GPE AND LUSTRE COMMUNICATION ERROR MESSAGES ON SYSTEM X.

Error	Error	5	6	11	19	24	26	27
ECC	GPE	0.1	0.14	-0.12	–	–	0.15	–
segfault	GPE	0.81	0.99	0.98	–	–	0.99	–
ECC	segfault	0.03	0.14	-0.14	0.01	-0.17	0.15	0.03
ECC	Lustre	–	–	—	0.1	0.17	–	-0.07
segfault	Lustre	–	–	–	0.99	0.82	–	0.99

TABLE V
CORRELATION SCORE OF [11] ON ECC ERROR, SEGFAULT, GPE AND LUSTRE COMMUNICATION ERROR MESSAGES ON SYSTEM X.

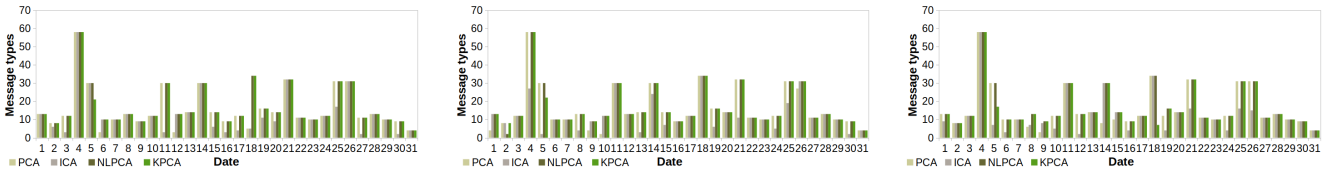
Error	Error	5	6	11	19	24	26	27
ECC	GPE	0.1	0.23	-0.14	–	–	0.15	–
segfault	GPE	0.6	0.99	0.71	–	–	0.99	–
ECC	segfault	0.03	0.23	-0.2	0.11	0.11	0.15	0.11
ECC	Lustre	–	–	—	0.11	0.17	–	-0.16
segfault	Lustre	–	–	–	0.99	0.24	–	0.43

> IFADE identified: (i) a strong positive partial correlation of ECC error, segfault and GPE on 4 dates (see Fig. 13(a)) and (ii) a strong positive partial correlation of ECC error, segfault and Lustre communication error on 3 dates (see Fig. 13(b)).



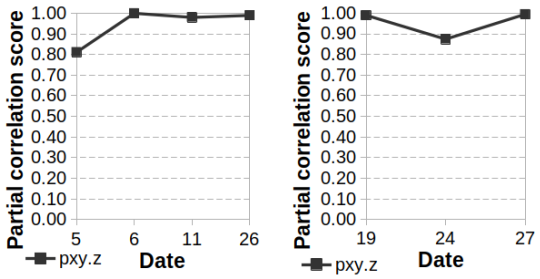
(a) 10 minute time-bins. (b) 30 minute time-bins. (c) 60 minute time-bins.

Fig. 11. Number of important message types extracted by PCA, ICA, NLPCA and KPCA on System X.



(a) 10 minute time-bins. (b) 30 minute time-bins. (c) 60 minute time-bins.

Fig. 12. Number of important message types extracted by PCA, ICA, NLPCA and KPCA on System Y.



(a) ECC error (z), segfault (x) and general protection fault (y) (b) ECC error (z), segfault (x) and Lustre comms error (y)

Fig. 13. Partial correlation of ECC error to segfault, general protection fault and Lustre communication error on System X.

Correlation with failures: The correlation score of ECC error, segfault, GPE and Lustre communication error to soft lockup events are given in Table VI. We observed that: (i) segfault and GPE are strongly positive correlated to soft lockup on date 6, (ii) ECC error is weakly correlated to soft lockup on all 7 dates.

TABLE VI
BIVARIATE CORRELATION SCORES OF ECC ERROR, SEGFAULT, GPE AND LUSTRE COMMUNICATION ERROR TO SOFT LOCKUP MESSAGES.

Error	Message	5	6	11	19	24	26	27
ECC	soft lockup	0.29	0.14	0.09	-0.04	0.3	-0.15	-0.1
segfault	soft lockup	-0.03	1	-0.03	-0.02	-0.02	-0.04	-0.02
GPE	soft lockup	0.05	1	-0.03	–	–	-0.04	–
Comms	soft lockup	–	–	–	-0.02	0.07	–	-0.01

Detailed diagnosis: The ECC mechanism detected corrupted data in memory on 7 dates. On one date, the ECC mechanism failed to correct the data corruption. A program tried to use the corrupted data value to access data in the memory which caused a segmentation fault and led to a compute node soft lockup. IFADE identified the date when a compute node soft lockup was *indirectly* caused by an ECC error.

Significance test: We test all the partial correlation coefficients against the null hypothesis and obtained z-scores ranging from 3.71 to 10.68. At 99% confidence level, under the null hypothesis $z_{0pr} = 2.64$. Hence, we reject the null hypothesis in favour of the alternate hypothesis.

False positives: Next, we determine the probability of rejecting the null hypothesis when it is true. We apply a one-sided test and use the significance level, $\alpha = 0.01$ for all given hypothesis tests to obtain a P-value. The lowest z-score is 3.71. Since this is a one-sided test, the P-value is 0.000104. We obtained the adjusted P-value $0.000104 \times 26 = 0.0027$ where 26 is the number of dates. The adjusted P-value is less than 0.01, indicating it is highly unlikely this result would be observed under the null hypothesis. The z-score for all the partial correlation coefficients are greater than or equal to 3.71 and the adjusted P-values are less than 0.01; it is highly unlikely these results would be observed under the null hypothesis.

V. LESSONS LEARNED

In this section, we reflect on the four activities we undertook: (i) data preprocessing, (ii) features extraction, (iii) partial correlation and (iv) diagnosing system failures without prior knowledge of a fault model.

A. Data preprocessing

The objective of data preprocessing is to generate resource use counters and message types data matrices into time-bins of multiple granularities. We showed that, irrespective of the format of the resource use data and system logs, we organized the counters and message types into data matrices of 10, 30 and 60 minute bins. This has enabled us to identify the largest number of important counters groups and largest number of important system events. IFADE does not do actual error recovery because there are no checkpoint messages in the system logs. IFADE is dependant on HPC system logs only because we know how to parse the data. That said, IFADE is extensible to data processing frameworks that parse distributed system logs, e.g., in [34]. Currently, we are developing time and space analysis approaches to relate other resource use counters and errors to new system failures.

B. Features extraction

The objective of feature extraction is to identify the important counters and message types from the redundant ones. We

showed that non-linear PCA extracted the largest number of important counters and message types. In [35], it is reported that the features extracted on multiple time-bins resulted in a higher accuracy for predicting compute node soft lockups. Thus, IFADE can be integrated with failure prediction frameworks, e.g., in [35].

C. Partial correlation

The objective of partial correlation is to identify indirect relationships between the counters or system events. Partial correlation can help identify the counters or system events which are: (a) partially correlated and (b) both directly and partially correlated. In our evaluation, we showed that: (a) Lustre I/O control and CPU I/O wait counters are partially correlated via the memory data update counter and also directly correlated, (b) segfault and general protection fault are partially correlated via ECC error and also directly correlated and (c) segfault and Lustre communication error are partially correlated via ECC error and also directly correlated.

D. Diagnosing system failures

The goal in this paper is to diagnose system failures without prior knowledge of a fault model. We showed that IFADE can identify two previously unknown causes of Lustre I/O and segmentation faults (see Section IV). Thus, we can automate the diagnosis process by implementing the steps in the workflow (see Fig. 2). IFADE is specific to Lustre I/O problems and segmentation faults only because we know how to diagnose these problems. IFADE is available at https://tinyurl.com/36zuz4fv to support system administrators in diagnosis of system failures.

VI. Related Work

In [36], the authors integrated PCA, Kullback-Leibler divergence and Pearson correlation to identify influences among interacting system components. In their method, the width of the time-bin is selected automatically. While IFADE also applied a three-phase approach, there are differences: (i) it organized the counters and system events into time-bins of multiple granularities, (ii) it applied linear and non-linear feature extractors and (iii) it applied partial correlation.

In [17], the authors compared a PCA-based and an ICA-based anomaly detection methods to identify faulty nodes on a large cluster system. They showed that the ICA-based method is more effective than the PCA-based method. In [19], the authors presented an adaptive anomaly detector to identify anomalies in cloud computing systems. Their approach was based on integrating PCA with a machine learning technique to find the most relevant principal component for each type of possible failures. In [15], the authors evaluated PCA, ICA and Mahalanobis distance to identify anomalous nodes using resource usage data. They showed that the PCA-instance identified all the nodes that matched all the dates of compute node soft lockups in a large HPC system. In [9], the authors applied PCA and ICA to identify faulty nodes in two HPC systems. Differently to these works, IFADE applied linear and non-linear feature extractors to extract the largest number of important resource use counters and system events.

In [37], the authors presented a novel approach that modeled normal and faulty system behaviour by applying an Event Signal Log Analyzer on event logs. In [38], the authors developed a novel relational operator for system diagnosis. In [39], the authors proposed a novel scheme that used the spatial locality of failures to provide better application and system performance. In [40], the authors developed a thorough understanding of interconnect errors and job characteristics on an enterprise class supercomputer. Network traffic and access patterns were identified by Google's Network Telemetry [41]. In [42], the authors used correlation to predict optical transceiver failure patterns in Facebook data centers. In [43], the authors provided a detailed statistical analysis of system and application failures using IBM BlueGene/L event logs, and developed Baler [44] – a log-clustering tool that analyzed root-causes of system failures. In [27], the authors presented a principled approach to obtain new insight into failure prediction using log-data on the Computer Failure Data Repository. In [45], the authors integrated probabilistic analysis with an optimized K-means algorithm to detect error propagation across the nodes in a HPC system. In [46], the authors presented a big-data analytics framework that mines event patterns and provides user application and system event correlations. In [47], the authors presented a scalable, intuitive HPC data analysis framework. These frameworks are working on HPC systems and IFADE complements them by identifying partial correlation of resource use counters and partial correlation of system errors for failure diagnosis.

In [48], the authors applied Pearson and Spearman-Rank correlation methods on the resource use data and system logs to identify errors that lead to failure or recovery on HPC systems. In [49], the authors integrated feature extraction and correlation methods to identify rare error propagation cases. The diagnostics frameworks in [48], [49] targeted error propagation or recovery in HPC systems. Differently to these works, IFADE targets system failure diagnosis and applied feature extraction and partial correlation algorithms. We compared IFADE with current failure diagnostics approaches [9]–[11]. We showed that IFADE identified: (a) a previously unknown memory data update cause of Lustre I/O which led to compute node soft lockups and (b) a previously unknown corrupted memory index cause of segmentation fault which led to a soft lockup.

VII. Conclusion and Future Work

The novel IFADE diagnostics workflow was developed to uncover hidden relationships so as to support detailed diagnosis of HPC system failures. IFADE generated resource use counters and message types data matrices of multiple granularities of time-bins, and applied feature extraction and partial correlation algorithms. We compared IFADE with three system failure diagnostics approaches and showed that IFADE identified two previously unknown causes of system failures. We applied statistical significance tests and ensured accurate diagnosis of the system failure.

For our future work, we plan to analyze other causes of system failures to deal with problems other than Lustre filesystem I/O and segmentation faults.

Acknowledgements

This research is supported by the European Union's Horizon 2020 Research and Innovation program under Grant Agreement No. 830927, Security Lancaster under the EPSRC grant

EP/V026763/1 and the National Science Foundation under OCI awards #0622780 and #1203604 to Texas Advanced Computing Center (TACC) at The University of Texas at Austin, USA. We thank Bill Barth at TACC for providing the cluster log-data and Joshua Fryman at Intel for granting access to his engineers.

REFERENCES

[1] A. Avizienis, J.-C. Lapire, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[2] A. Das, F. Müller, and B. Rountree, "Systemic assessment of node failures in HPC production platforms," in *Proceedings of IEEE IPDPS*, 2021.

[3] R. Kumar, S. Jha, A. Mahgoub, R. Kalyanam, S. Harrell, X. C. Song, Z. Kalbarczyk, W. Kramer, R. Iyer, and S. Bagchi, "The mystery of the failing jobs: Insights from operational data from two university-wide computing systems," in *Proceedings of IEEE/IFIP DSN*, 2020.

[4] S. Di, H. Guo, E. Pershey, M. Snir, and F. Cappello, "Characterizing and understanding hpc job failures over the 2k-day life of ibm bluegene/q system," in *Proceedings of IEEE/IFIP DSN*, 2019.

[5] S. Gupta, T. Patel, C. Engelmann, and D. Tiwari, "Failures in large scale systems: Long-term measurement, analysis, and implications," in *Proceedings of IEEE/ACM Supercomputing (SC)*, 2017, pp. 44:1–44:12.

[6] G. Wang, L. Zhang, and W. Xu, "What can we learn from four years of data center hardware failures?" in *Proceedings of IEEE/IFIP DSN*, 2017.

[7] B. Schroeder and G. Gibson, "The computer failure data repository (cfdr): Collecting, sharing and analyzing failure data," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, no. 154, 2006.

[8] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *Proceedings of IEEE/IFIP DSN*, June 2007.

[9] E. Chuah, A. Jhumka, J. C. Browne, B. Barth, and S. Narasimharmuthy, "Insights into the diagnosis of system failures from cluster log data," in *Proceedings of EDCC*, 2015, pp. 1–8.

[10] Z. Zheng, L. Yu, Z. Lan, and T. Jones, "3-dimensional root cause diagnosis via co-analysis," in *Proceedings of ACM ICAC*, 2012, pp. 181–190.

[11] E. Chuah, A. Jhumka, J. C. Browne, N. Gurumdimma, S. Narasimharmuthy, and B. Barth, "Using message logs and resource use data for cluster failure diagnosis," in *Proceedings of IEEE HiPC*, 2016.

[12] J. L. Hammond, T. Minyard, and J. Browne, "End-to-end framework for fault management for open source clusters: Ranger," in *Proceedings of ACM TeraGrid Conference*, no. 9, 2010.

[13] IEEE, *IEEE Std 1003.1-2001 Standard for Information Technology — Portable Operating System Interface (POSIX) Base Definitions, Issue 6*. IEEE Standards, 2001.

[14] R. T. Evans, J. C. Browne, and W. L. Barth, "Understanding application and system performance through system-wide monitoring," in *Proceedings of IEEE IPDPS Workshops*, 2016, pp. 1702–1710.

[15] E. Chuah, A. Jhumka, S. Narasimharmuthy, J. Hammond, J. C. Browne, and B. Barth, "Linking resource usage anomalies with system failures from cluster log data," in *Proceedings of IEEE SRDS*, 2013, pp. 111–120.

[16] M. Serafini, A. Bondavalli, and N. Suri, "Online diagnosis and recovery: On the choice and impact of tuning parameters," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 4, pp. 295–312, 2007.

[17] Z. Lan, Z. Zheng, and Y. Li, "Toward automated anomaly identification in large-scale systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 2, pp. 174–187, 2010.

[18] Q. Guan, D. Smith, and S. Fu, "Anomaly detection in large-scale coalition clusters for dependability assurance," in *Proceedings of IEEE HiPC*, 2010.

[19] Q. Guan and S. Fu, "Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures," in *Proceedings of IEEE SRDS*, 2013.

[20] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *Proceedings of IEEE ISSRE*, 2016, pp. 207–218.

[21] M. Scholz, "Validation of nonlinear PCA," *Neural Processing Letters*, vol. 36, pp. 21–30, 2012.

[22] C. Croux, P. Filzmoser, and M. R. Oliveira, "Algorithms for projection-pursuit robust principal component analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 87, no. 2, pp. 218–225, 2007.

[23] A. Hyvarinen and E. Oja, "Independent component analysis: Algorithms and applications," *Neural Networks*, vol. 13, no. 4-5, pp. 411–430, 2000.

[24] M. Scholz, F. Kaplan, C. L. Guy, J. Kopka, and J. Selbig, "Non-linear PCA: a missing data approach," *Bioinformatics*, vol. 21, no. 20, 2005.

[25] H. Hoffmann, "Kernel PCA for novelty detection," *Pattern Recognition*, vol. 40, no. 3, pp. 863 – 874, 2007.

[26] F. Song, Z. Guo, and D. Mei, "Feature selection using principal component analysis," in *Proceedings of IEEE International Conference on System Science, Engineering Design and Manufacturing Informatization*, 2010.

[27] A. Goudarzi, D. Arnold, D. Stefanovic, K. B. Ferreira, and G. Feldman, "A principled approach to HPC event monitoring," in *5th Workshop on Fault Tolerance for HPC at eXtreme Scale*, 2015, pp. 3–10.

[28] K. Baba, R. Shibata, and M. Sibuya, "Partial correlation and conditional correlation as measures of conditional independence," *Australian & New Zealand Journal of Statistics*, vol. 46, no. 4, pp. 657–664.

[29] A. Agresti and C. Franklin, *Statistics: The Art and Science of Learning From Data*. Prentice Hall International, 2009.

[30] R. E. Walpole, R. H. Myers, and S. L. Myers, *Probability and Statistics for Engineers and Scientists*. Prentice Hall International, 1998.

[31] J. J. Goeman and A. Solari, "Multiple hypothesis testing in genomics," *Statistics in Medicine*, vol. 33, no. 11, pp. 1946–1978, 2014.

[32] J. T. Palmer, S. M. Gallo, T. R. Furlani, M. D. Jones, R. L. DeLeon, J. P. White, N. Simakov, A. K. Patra, J. Sperhac, T. Yearke, R. Rathsam, M. Innus, C. D. Cornelius, J. C. Browne, W. L. Barth, and R. T. Evans, "Open XDMoD: A tool for the comprehensive management of high-performance computing resources," *Computing in Science Engineering*, vol. 17, no. 4, pp. 52–62, July 2015.

[33] J. C. Browne, R. L. DeLeon, C. D. Lu, M. D. Jones, S. M. Gallo, A. Ghadersohi, A. K. Patra, W. L. Barth, J. Hammond, T. R. Furlani, and R. T. McLay, "Enabling comprehensive data-driven system management for large computational facilities," in *Proceedings of ACM Supercomputing (SC)*, Nov 2013, pp. 1–11.

[34] M. Astekin, H. Zengin, and H. Sözer, "DILAF: A framework for distributed analysis of large-scale system logs for anomaly detection," *Software: Practice and Experience*, vol. 49, no. 2, pp. 153–170, 2019.

[35] A. Pelaez, A. Quiroz, J. C. Browne, E. Chuah, and M. Parashar, "Online failure prediction for HPC resources using decentralized clustering," in *Proceedings of IEEE HiPC*, 2014, pp. 1–9.

[36] A. J. Oliner, A. V. Kulkarni, and A. Aiken, "Using correlated surprise to infer shared influence," in *Proceedings of IEEE/IFIP DSN*, 2010.

[37] A. Gainaru, F. Cappello, and W. Kramer, "Taming of the shrew: Modeling the normal and faulty behaviour of large-scale HPC systems," in *Proceedings of IEEE IPDPS*, 2012, pp. 1168–1179.

[38] J. Mace, R. Roelke, and R. Fonseca, "Pivot tracing: Dynamic causal monitoring for distributed systems," in *Proceedings of ACM SIGOPS*, 2015, pp. 378–393.

[39] 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 IEEE/IFIP DSN*, 2015, pp. 37–44.

[40] M. Kumar, S. Gupta, T. Patel, M. Wilder, W. Shi, S. Fu, C. Engelmann, and D. Tiwari, "Understanding and analyzing interconnect errors and network congestion on a large scale HPC system," in *Proceedings of IEEE/IFIP DSN*, 2018, pp. 107–114.

[41] G. N. Telemetry, https://cloud.google.com/network-telemetry/, 2017.

[42] A. Chakravarty and V. Zeng, "Failure prediction mechanism for pluggable optical interconnect at Facebook data centers," in *OpenCompute Project, US Summit*, 2018.

[43] N. Taerat, N. Naksinehaboon, C. Chandler, J. Elliott, C. Box, L. , G. Ostrouchov, and S. L Scott, "Using log information to perform statistical analysis on failures encountered by large-scale HPC deployments," 2009.

[44] N. Taerat, J. Brandt, A. Gentile, M. Wong, and C. Leangsuksun, "Baler: Deterministic, lossless log message clustering tool," *Comput. Sci.*, vol. 26, no. 3-4, pp. 285–295, 2011.

[45] S. Di, R. Gupta, M. Snir, E. Pershey, and F. Cappello, "Logaider: A tool for mining potential correlations of HPC log events," in *Proceedings of IEEE CCGRID*, May 2017, pp. 442–451.

[46] B. H. Park, Y. Hui, S. Boehm, R. A. Ashraf, C. Layton, and C. Engelmann, "A big data analytics framework for HPC log data: Three case studies using the Titan supercomputer log," in *Proceedings of IEEE Cluster Computing*, Sept. 2018.

[47] A. Giménez, T. Gamblin, A. Bhatele, C. Wood, K. Shoga, A. Marathe, P.-T. Bremer, B. Hamann, and M. Schulz, "Scrubjay: Deriving knowledge from the disarray of HPC performance data," in *Proceedings of IEEE/ACM Supercomputing (SC)*, 2017.

[48] E. Chuah, A. Jhumka, S. Alt, D. Balouek-Thomert, J. C. Browne, and M. Parashar, "Towards comprehensive dependability-driven resource use and message log-analysis for HPC systems diagnosis," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 95–112, 2019.

[49] E. Chuah, A. Jhumka, S. Alt, J. Villalobos, J. B. Fryman, W. L. Barth, and M. Parashar, "Using resource use data and system logs for HPC system error propagation and recovery diagnosis," in *Proceedings of IEEE ISPA*, 2019, pp. 1–10.