Enabling Dependability-Driven Resource Use and Message Log-Analysis for Cluster System Diagnosis

Edward Chuah[‖*], Arshad Jhumka[*], Samantha Alt[††], Theo Damoulas[‖*], Nentawe Gurumdimma[**], Marie-Christine Sawley[††], William L. Barth[¶], Tommy Minyard[¶], James C. Browne[‡]

[‖]The Alan Turing Institute, 96 Euston Road, London NW1 2DB, UK. Email: {echuah, tdamoulas}@turing.ac.uk

[*]University of Warwick, Coventry CV4 7AL, UK. Email: {E.Chuah, H.A.Jhumka, T.Damoulas}@warwick.ac.uk

[††]Intel Corporation, USA and France. Email: {samantha.alt, marie-christine.sawley}@intel.com

[**]University of Jos P.M.B 2084 Jos, Plateau State Nigeria, Post code: 930001. Email: yusufn@unijos.edu.ng

[¶]Texas Advanced Computing Center, Texas 78758. Email: {bbarth, minyard}@tacc.utexas.edu

[‡]University of Texas at Austin, Texas 78712. Email: browne@cs.utexas.edu

***Abstract*—Recent work have used both failure logs and resource use data separately (and together) to detect system failure-inducing errors and to diagnose system failures. System failure occurs as a result of error propagation and the (unsuccessful) execution of error recovery mechanisms. Knowledge of error propagation patterns and unsuccessful error recovery is important for more accurate and detailed failure diagnosis, and knowledge of recovery protocols deployment is important for improving system reliability. This paper presents the CORRMEXT framework which carries failure diagnosis another significant step forward by analyzing and reporting error propagation patterns and degrees of success and failure of error recovery protocols. CORRMEXT uses both error messages and resource use data in its analyses. Application of CORRMEXT to data from the Ranger supercomputer have produced new insights. CORRMEXT has: (i) identified correlations between resource use counters that capture recovery attempts after an error, (ii) identified correlations between error events to capture error propagation patterns within the system, (iii) identified error propagation and recovery paths during system execution to explain system behaviour, (iv) showed that the earliest times of change in system behaviour can only be identified by analyzing both the correlated resource use counters and correlated errors. CORRMEXT will be installed on the HPC clusters at the Texas Advanced Computing Center in Autumn 2017.**

***Index Terms*—Large cluster system; Correlation; Variance extraction; Error Recovery and propagation; Cluster log data**

I. Introduction

There has been much fruitful research on failure-inducing error detection and the analysis and diagnosis of system failures. This research is summarized briefly below and analyzed in Section V. HPC and data center operators face challenges on a day by day basis. Recent work that studied HPC and data center failures [1], [2] have shown that the mean time between failures is decreasing and it is important for the system administrators to identify errors quickly. However, these high-performance systems generate a massive amount of data. The different types of data are not necessarily structured. Therefore, finding the right types of data and rapidly analyzing the data to diagnosis system problems is a challenging task while at the same time important for improving the HPC system performance and uptime. A significant body of research has shown the value of message logs for error detection [3]–[5], failure prediction [6]–[12] and failure diagnosis [13]–[15]. Another significant body of research has shown the value of resource use data to detect anomalies and characterize errors [16]–[18], predict failures [19] and recovery [20]. The use of resource use data and message logs *separately* has provided important methods and tools that help manage these systems.

Recent work which use resource use data *and* message logs for failure diagnosis [21], [22] and error detection [23], [24] has shown increased accuracy over using message logs alone. [21] provides partial diagnosis of system failures by using resource use data to identify resource anomalies, and provides a more precise diagnosis by using message log-analysis. [24] combines analyses of message logs and resource use data but the focus is on error detection. [23] uses message logs and resource use data to increase the error handling time window, and [22] is focused on correlating resource usage and message logs with system failures. [25] combines analysis of RAS logs and job logs but the focus is on identifying failure characteristics in a cluster system. Previous work [21], [22], [26] have correlated errors only with failures, but there is little work which focuses on other dependability-related issues such as error propagation and error recovery. The knowledge of error propagation paths can be useful to system designers and implementators for improving the effectiveness of error recovery protocols.

A system failure occurs due to error propagation from one component to another until the error reaches system boundary. This error propagation to system boundary is either due to the unsuccessful execution of error recovery mechanisms or the lack of it. State-of-the-art techniques on failure diagnosis correlates a given error event (e.g., segmentation fault) with system failure, i.e., the techniques return the most likely cause of the failure. Though important, this information may not provide enough details to a system administrator. Differently, capturing the error propagation paths and/or understanding the reason behind the unsuccessful execution of the recovery mechanisms provides the system administrators with more details. For example, information such as *"An inode failure led to communication errors, which were not recovered, and which led to failure"* provides more details than *"system failure is due to communication errors"* as the recovery can now target

inodes (rather than communication errors).

This paper introduces the *CORRMEXT* framework to identify possible error propagation paths and provide more dependability-oriented system diagnosis. CORRMEXT uses the TACC_Stats [27] resource monitoring system and Rationalized message logging [28] to resolve resource use and system messages by job, node and time for open-source Linux clusters. CORRMEXT identifies and links groups of resource use and events patterns on a given date. It applies multiple correlation algorithms. CORRMEXT implements a three-phase approach where: (i) correlations are performed on the resource use counters, (ii) correlations are performed on the message logs, (iii) variance extraction is performed on time-bins associated with the correlated resource use counter groups and correlated error groups.

The benefit of combining analysis of resource use and message logs is given in the following example: When correlations of process and NUMA[1] resource use counters, and correlations of memory errors occur on the same date, it shows that memory allocation activities are associated with the generation of memory errors. Therefore, the correlated NUMA and process counters can be used to monitor the state of memory allocation and the correlated memory errors can be used to identify the application which caused the error. In this paper, we will show that CORRMEXT provides a pathway to the root-cause of successful and failed error recovery mechanisms which alternative diagnostics tools can not provide.

The main contributions of this paper include:

- A new framework (CORRMEXT) that combines resource use data and message logs for detailed dependability-oriented system diagnosis.
- A demonstration that CORRMEXT can: (i) identify error propagation paths leading to failure, (ii) explain instances of error recovery, (iii) explain the causes of failed error recovery mechanisms.
- CORRMEXT shows that more correlated errors associated with error propagation and recovery can only be identified by applying different correlation algorithms.
- CORRMEXT includes a detailed statistical analysis step to ensure *accurate* dependability diagnosis. In our case studies, we show that all the correlations are statistically significant by applying the Bonferroni correction.
- CORRMEXT shows that the earliest times of change in system behaviour can only be identified by analyzing *both* the correlated resource use counters and correlated errors on all dates.

The remainder of this paper is structured as follows: We define the system and fault models in Section II, present the problem description and framework in Section III, evaluate CORRMEXT in Section IV, review the related work in Section V, conclude with a summary and future work in Section VII.

[1]Non-Uniform Memory Access (NUMA)

II. System and Fault Models

The class of systems to which CORRMEXT can be applied is specified in terms of a generic model of cluster systems given in [21] and we summarize here: A cluster system consists of a set of nodes and jobs and a scheduler that assigns jobs to nodes. Each component of a node or a job can output log-messages which are recorded in a message log-file. In addition, system resources used by the various jobs are recorded at regular intervals. The resource usage is recorded in a resource use log-file.

Fault Model: System failure[2] occurs when the system output deviates from the expected one. To increase the dependability of such systems, it is important to *tolerate those errors* that occur shortly before a system failure. Knowing the nature of these errors will ease error detection or failure diagnosis, thus helping with debugging or maintenance. We assume faults can occur at any level in the system, at the lowest level, e.g., register level to the highest level such as file systems. Execution of these faults will lead to errors which, if not handled, may lead to system failure.

III. System Issues, Problem Specification and Framework

An illustration of resource use and error messages by time is shown in Fig. 1. In the resource use data, we observed that user processor utilization (`CPU user`) and memory pages not accessed recently (`MEM Inactive`) are correlated. We also observed that user processor utilization (`CPU user`) and memory pages accessed recently (`MEM Active`) are correlated. In the rationalized message logs, we observed that chipset error (`Northbridge Error`), memory error (`ECC error`) and processor core (`core`) messages are correlated.

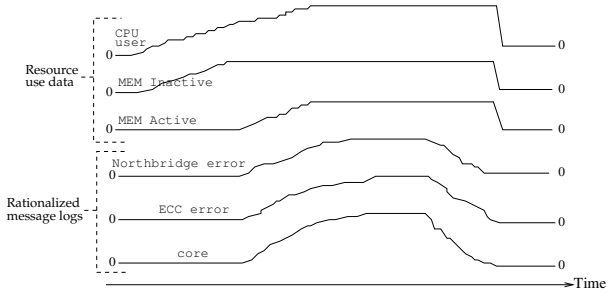


Fig. 1. An illustration of user processor utilization, memory pages not accessed recently, memory pages accessed recently, chipset, memory error and processor core messages.

Capturing these correlations is important as these can provide insights into the behaviour of the system. For example, a resource counter r_1 may capture the existence of an error in the system while another counter r_2 may capture the execution of some recovery procedure. A correlation between r_1 and r_2 shows that a recovery procedure was triggered following an error. Similarly, a correlation between different event groups

[2]A node crash or operating system hang-up is a system failure.

will provide insights into the system state. However, understanding the occurrence of groups of errors from message logs alone is challenging. The framework we present in this paper seeks to determine the occurrence of these patterns to enable study of various dependability-oriented processes such as error recovery and error propagation .

A. Problem Specification

The problem that we address in this paper is given as follows: Given (i) a set of resource use data, (ii) a set of message logs, (iii) a list of resource use counter names, (iv) a list of messsage types, and (v) a date, then:

1) Identify groups of resource use counters that are linearly or monotonically correlated on the specified date,
2) Identify groups of errors that are linearly or monotonically correlated on the specified date,
3) Identify the time-bin(s) that are associated with the correlated resource use counter groups and correlated error groups on the date specified.

The date specified captures the date where deeper insights are sought. To achieve this, we develop the *CORRMEXT* (**COR**relating **R**esource use data and **M**essage logs and **EX**tracting **T**ime-bins) framework as shown in Fig. 2. Next, we describe in detail, each of the three modules used within the CORRMEXT framework.

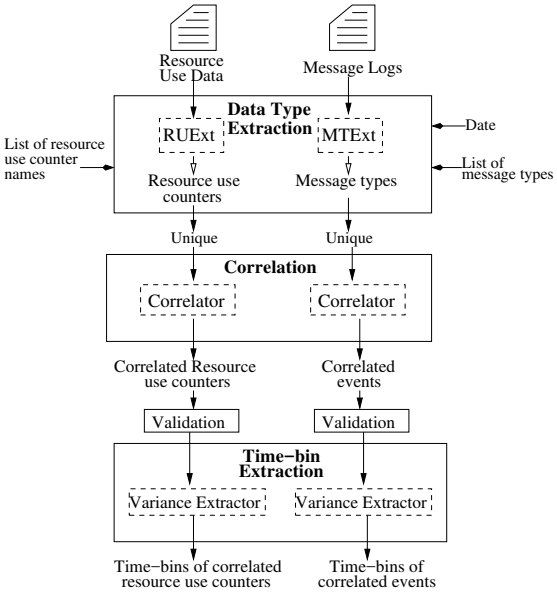


Fig. 2. The workflow of the CORRMEXT framework. The CORRMEXT framework is composed of three modules: (1) Data Type Extraction, (2) Correlation, (3) Time-bin Extraction. The workflow automatically process the resource use and message logs through the Data Type Extraction, Correlation and Time-bin Extraction modules. The output of each module are sets of reports which can be used for diagnosis.

B. Data Type Extraction

CORRMEXT targets processing of TACC_Stats resource use data [27], Syslogs [29] and Rationalized message logs [28]. TACC_Stats [27] is a job-oriented and logically structured version of the conventional Sysstat system performance monitor. The Rationalized message log [28] is a special type of message log that incorporates a logical structure and additional content such as job-identification to the POSIX formatted logs. The resource use data and message logs contain hundreds of different resource use counters and thousands of message types. In addition, the messages in the message logs may be ambiguous and unstructured. We developed a *Data Type Extractor* module that extracts: (i) resource use counters from the resource use data, and (ii) message types from the standard Syslogs and Rationalized message logs.

Resource Use Extractor (*RUExt*): We implemented the Resource Use Extractor to organize the resource use counters. *RUExt* extracts resource use counters from a major new version of TACC_Stats and presents the resource use counters in the form of a matrix of counts on which standard analysis algorithms can be applied. In addition to hardware performance monitoring data, Lustre file-system operation counts and InfiniBand device usage collected by an earlier version, the new TACC_Stats collects a comprehensive range of metrics that spans all system resources including energy consumption, vectorization, I/O and network activity. An example of a resource use log is shown below:

```
2066522 Aug 11 12:50:01 i150-412 mem 2 MemTotal
8273920 MemFree 5466616 MemUsed 2807304 ...
```

The sources of this resource use log can be identified from the job (`2066522`) and node (`i150-412`). The time that this resource use log was recorded can be identified from the timestamp (`Aug 11 12:50:01`). The main component associated with this resource use log can be identified from the word (`mem`) that follows immediately after the node identifier; in this example, the main component is the onboard memory system of the node `i150-412`. A second-level component can be identified from the word (`2`) that follows immediately after the first component identifier; in this example, the second-level component is memory bank number 2. The resource use counters can be identified from the key-value pairs (e.g. `MemFree 5466616`) that follow after the second component identifier. We define a parameter name, *param-name*, as a triple which comprises of the main component, second-level component and resource use counter key; in this example, the *param-name* is "mem 2 MemFree". A list of resource use counters and their components is given in Table I.

Currently, *RUExt* generates a data matrix $DR_{timebins}$ containing counts of resource use counters by time-bins. In the matrix, each row represents a param-name, each column represents a time-bin of one hour tb_j and each cell contains the count of param-name within time-bin tb_j. The data matrix is generated using a process which is given as follows:

- Step 1: Split the resource usage logs into individual hours by the given date.
- Step 2: For each log entry, extract the *param-name* and store it in a list.

TABLE I
SYSTEM METRICS GROUPS AND RESOURCE USE COUNTERS.

Metric group	Qty.	Resource use counters
Lustre network	6	tx_msgs, rx_msgs, rx_msgs_dropped, tx_bytes, rx_bytes, rx_bytes_dropped
Lustre /work, /share, /scratch	23 23 23	read_bytes, write_bytes, direct_read, direct_write, dirty_pages_hits, dirty_pages_misses, ioctl, open, close, mmap, seek, fsync, setattr, truncate, flock, getattr, statfs, alloc_node, setxattr, getxattr, listxattr, removexattr, inode_permission
Virtual memory	21	pgpgin, pgpgout, pswpin, pswpout, pgalloc_normal pgfree, pgactivate, pgdeactivate, pgfault, pgmajfault pgrefill_normal, pgsteal_normal, pgscan_kswapd _normal, pgscan_direct_normal, pginodesteal, slabs_scanned, kswapd_steal, kswapd_inodesteal, pageoutrun, allocstall, pgrotated
Block md0, hdd	11 11	rd_ios, rd_merges, rd_sectors, rd_ticks, wr_ios, wr_merges, wr_sectors, wr_ticks, in_flight, io_ticks, time_in_queue
Cpu 0 to 15	112	user, nice, system, idle, iowait, irq, softirq
Mem 0 to 3	80	MemTotal, MemFree, MemUsed, Active, Inactive, HighTotal, HighFree, LowTotal, LowFree, Dirty, Writeback, FilePages, Mapped, AnonPages, PageTables, NFS_Unstable, Bounce, Slab, HugePages_Total, HugePages_Free
Net ib0, lo, eth0,	23 23 23	collisions, multicast, rx_bytes, rx_compressed, rx_crc _errors, rx_dropped, rx_errors, rx_fifo_errors, rx _frame_errors, rx_length_errors, rx_missed_errors, rx_over_errors, rx_packets, tx_aborted_errors, tx_bytes, tx_carrier_errors, tx_compressed, tx _dropped, tx_errors, tx_fifo_errors, tx_heartbeat _errors, tx_packets, tx_window_errors
Numa 0 to 3	24	numa_hit, numa_miss, numa_foreign, interleave_hit, local_node, other_node
Ps	7	ctxt, processes, load_1, load_5, load_15, nr_running, nr_threads

- Step 3: Identify the unique *param-names* in the list and obtain the param-name types.
- Step 4: For each param-name type in a resource use log which match the given list of resource use counter names, if the values associated with the param-name types of two consecutive log entries are different, obtain the difference and add the difference to the value obtained in the preceding operation and store the value.

Message Types Extractor (*MTExt*): We implemented the Message Types Extractor to extract structured message templates from large quantities of message logs. *MTExt* extracts sequences of messages from the message logs and presents these messages in the form of a matrix of counts on which standard analysis algorithms can be applied. An example of a message log is shown below:

```
2055415 Aug 3 00:00:03 i120-306 kernel X Northbridge
Error, node %d ...
```

The date and time that this message log was recorded can be identified from the timestamp (`Aug 3 00:00:03`). The sources that generated this message log can be identified from the job (`2055415`) and node (`i120-306`). The message *constant* part in this message log can be identified from a sequence of English-only words (`X Northbridge Error, node`). The message type can be identified from the message constant; in this message log, the message type is a Northbridge error.

Currently, *MTExt* generates a data matrix $DM_{timebins}$ containing counts of message types by time-bins. In the matrix, each row represents a message type mt_i, each column represents a time-bin of one hour tb_j and each cell contains the count of a message type mt_i within time-bin tb_j. The data matrix is generated using a process which is given as follows:

- Step 1: Split the message logs into logs of individual hours by the given date.
- Step 2: For each message log, extract the message *constant* part and store it in a list.
- Step 3: Identify the unique message *constants* in the list and obtain the message types.
- Step 4: Given a list of message types, count the number of message types by hour for the given date in the logs.

C. Correlation

Once the Data Type Extractor module has generated the time-bin data matrices, the *Correlation* module computes: (i) the correlation coefficients between the resource use counters contained in the time-bin ($DR_{timebins}$) data matrix and extracts a smaller set of resource use counters for analysis, and (ii) the correlation coefficients between the events contained in the time-bin ($DM_{timebins}$) data matrix and extracts a smaller set of message logs for analysis. We implemented Pearson correlation as the base algorithm to identify linear patterns and Spearman-Rank correlation to identify patterns that increase monotonically.

Pearson correlation [30] draws a line of best fit through the data of two variables and it assumes a linear relationship between the data of two variables. The Pearson correlation coefficient, r is defined as the mean of the products of the standard scores, i.e., $r = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{x_i-\bar{x}}{s_x}\right)\left(\frac{y_i-\bar{y}}{s_y}\right)$ 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 resource use counters or a pair of events, $s_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2}$ 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.

Spearman-Rank correlation [30] assumes a monotonic relationship that does one of the following: (i) when the value of one variable increases, the value of the other variable increases, (ii) the value of one variable remains, the value of the other variable remains. The Spearman-Rank correlation coefficient, ρ is defined as the Pearson correlation coefficient between a pair of ranked variables. To rank the variables, we implemented a standard technique called the tied rank average method [30]. The process is given as follows:

- Rank order the values in the dataset x with the smallest value getting a rank of 1.
- If more than one value has the same rank in dataset x, assign the average rank to these values.

- Rank order the values in the dataset y with the smallest value getting a rank of 1.
- If more than one value has the same rank in the dataset y, assign the average rank to these values.

Our objective is to identify patterns in the resource use data and message logs which are strongly *positive* correlated. The correlation algorithms require that a pair of datasets contain the same number of data points on the x-axis. The resource use data and message logs are generated by different open source software tools and the log entries in the resource use data and message logs may contain different timestamps. As a result, the dataset for a resource use counter and a message type may contain different numbers of data points. It is for this reason that we do not apply our Correlation module directly on a pair of resource use counter and message type dataset. Having said that, our objective is to identify correlations of resource use counters and correlations of events. Hence, we apply our Correlation module separately on the resource use data and message logs as shown in Fig. 2.

Once the correlation matrices have been generated, we apply a process given in [21] to generate the lists of strongly correlated resource use counters and strongly correlated events. The process automatically identifies a correlation threshold r_{th} and uses r_{th} to extract the strongly correlated resource use counters, strongly correlated events and their correlation coefficients. We use the following rules to interpret the strength of the correlation coefficient: (a) 0.9 to 1: Very strong positive correlation, (b) 0.7 to 0.9: Strong positive correlation, (c) 0.5 to 0.7: Moderate positive correlation, (d) 0.3 to 0.5: Weak positive correlation, (e) 0.1 to 0.3: Very weak positive correlation.

Validation: To test the significance of the correlation coefficient, we apply a standard technique called Fisher's z-transform [30] $F(r) = \frac{1}{2}\ln\left(\frac{1+r}{1-r}\right)$ on the correlation coefficient to obtain a z-score. We define the null hypotheses as: (i) H_{0r} that a pair of resource use counters are *very weakly* positive correlated, and (ii) H_{0e} that a pair of events are *very weakly* positive correlated. We define the alternate hypotheses as: (i) H_{ar} that a pair of resource use counters are *strongly* positive correlated, and (ii) H_{ae} that a pair of events are *strongly* positive correlated. Under the null hypothesis H_{0r}, $F(r)$ approximately follow a normal distribution with mean $u_z = F(H_{0r}) = 0$ and standard error $SE = \frac{1}{\sqrt{n_r - 3}}$, where n_r is the number of time-bins in the pair of resource use counters. Under the null hypothesis H_{0e}, $F(r)$ approximately follows a normal distribution with mean $u_z = F(H_{0e}) = 0$ and standard error $SE = \frac{1}{\sqrt{n_e - 3}}$, where n_e is the number of time-bins in the pair of message types. Then, we obtain the z-score for all correlation coefficients, i.e., $z_r = \frac{F(r) - u_z}{SE} = (F(r) - F(H_{0r})) \times \sqrt{n_r - 3}$ and $z_e = \frac{F(r) - u_z}{SE} = (F(r) - F(H_{0e})) \times \sqrt{n_e - 3}$ where z_r is the z-score for a pair of correlated resource use counters and z_e is the z-score for a pair of correlated events.

A large absolute value of z, e.g., 2.64 at 99% confidence level, will reject the null hypotheses in favour of the alternate hypotheses that a pair of resource use counters and a pair of events are correlated. We are interested in *strongly positive* correlated resource use counters and *strongly positive* correlated events. To test the probability of rejecting the null hypothesis when it is true, we apply a one-sided test and use the significance level, $\alpha = 0.01$ to obtain a P-value for all hypothesis tests. Since this is a one-sided test, the P-value is equal to the probability of observing a value greater than z_r or z_e in the standard normal distribution, or $P(Z > z_r) = 1 - P(Z \leq z_r)$ or $P(Z > z_e) = 1 - P(Z \leq z_e)$. A P-value less than 0.01 indicates that it is highly unlikely the result would be observed under the null hypothesis.

Implementation of significance testing: We implemented a process to test the significance of all the correlation coefficients. The process is given as follows: The number of hours for each date of logs is generated by *RUExt* and *MTExt* and the number of dates processed is generated by the Data Type Extractor (DTE). The total number of hours in each date of logs is given to the validation sub-component by *RUExt* and *MTExt* and the correlation coefficients are given to the validation sub-component by the Correlation module shown in Fig. 2. The correlation coefficients and number of hours are then used to compute Fishers z-scores for all correlation coefficients. Then, the unadjusted P-values are obtained by mapping the z-scores to P-values using a Z-table which is implemented in the validation sub-component. The adjusted P-value is then obtained by multiplying the unadjusted P-value by the total number of dates.

Bonferroni Correction: Given d number of hypotheses, the probability of observing at least one significant result just due to chance is $1-(1-P)^d$ where P is the p-value obtained from each test. If there is only one hypothesis to test and the P-value obtained is 0.01, then there is a 1% probability that it is a false positive. When there is more than one hypothesis to test, for example 26 hypotheses and a P-value of 0.01 is obtained for each test, the probability that there is at least one false positive is $1 - (1 - 0.01)^{26} = 1 - 0.99^{26} = 0.22$ or 22%. To account for inflation in false positive due to multiple independent tests, we apply a standard technique called Bonferroni correction to counteract the problem of multiple comparisons [31]. We use the Bonferroni correction on the unadjusted P-value obtained for each test to obtain an adjusted P-value. Given d tests T_i for hypotheses $H_i (1 \leq i \leq d)$ under the null hypothesis assumption H_0 that all alternate hypotheses H_i are false, and if the individual test critical values are $\leq \alpha/d$, then the experiment-wide critical value is $\leq \alpha$. We obtain the adjusted P-value for all hypothesis tests by multiplying the unadjusted P-value by d.

D. Time-bin Extraction

Once the Correlation module has extracted the lists of strongly correlated resource use counters and strongly correlated events, the *Time-bin Extraction* module identifies the time-bins which have the highest variance in the correlated resource use counters dataset and correlated events dataset. In the correlated resource use counters dataset, each row

represents a resource use counter and each column represents a time-bin of one-hour. In the correlated events dataset, each row represents a message type and each column represents a time-bin of one-hour. Our Time-bin Extraction module currently computes the variance of both the correlated resource use counters dataset and correlated events dataset at every hour. The variance is $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$ where $n =$ sample size of x, $\bar{x} =$ sample mean of x.

Our objective is to identify times of change in the system behaviour during the day. To identify the earliest times in the correlated resource use counters and correlated events datasets, we implemented a process to extract the variance associated with the time-bins. The process is given as follows:

1) Store the variance for each time-bin in a list l_{var}.
2) Obtain the difference in the variance between two consecutive time-bins and store the difference in a list $l_{vardiff}$.
3) Sort $l_{vardiff}$ in descending order with the first element the largest difference in variance.

IV. Case Study: Ranger Supercomputer

Our study of error recovery and propagation is carried out within the context of the decommissioned Ranger cluster at the Texas Advanced Computing Center at the University of Texas at Austin. We collected the resource use data[3] and rationalized message logs for June, July and August 2011, and a summary is given in Table II.

TABLE II
SUMMARY OF LOG-DATA COLLECTED ON RANGER.

	Resource use data		**Rationalized message logs**	
Month	Size	Qty. lines	Size	Qty. messages
June 2011	120.9 GB	603,024,456	2.7 GB	10,021,516
July 2011	124.1 GB	637,860,203	9.6 GB	64,822,682
August 2011	125.4 GB	633,396,685	14.5 GB	114,745,476

[28] reported that the lead time on occurrence of an error to a soft lockup failure is six hours. There are 26 dates in the rationalized message logs when soft lockups were reported. The dates of log-data analyzed are given in Table III.

TABLE III
LIST OF DATES OF LOG-DATA ANALYZED ON RANGER.

Month	**Dates**
June 2011	3, 5, 14, 15, 16, 21, 22
July 2011	5, 6, 7, 11, 18, 19, 23, 24, 25, 26, 27, 31
August 2011	3, 4, 11, 22, 24, 30, 31

We obtain the list of correlated resource use counters and list of correlated events, discussed the lists of resource use counters and events with the systems administrators and identified three dependability use cases, and summarized them in Table IV.

In this section, we give the details for the first use case on recovery attempt and its impact on application and process memory allocation. The details for the second and third use cases are provided in a companion report[4].

[3]The resource use on the Ranger cluster was sampled at 10-minute intervals.

TABLE IV
SUMMARY OF DEPENDABILITY USE CASES ON RANGER.

Issue	**System**	**Error**	**No. of dates**
Recovery attempt	Application & process memory allocation	Memory allocation & memory leaks	25
Error propagation	Lustre file-system I/O & Infiniband	Communication & file-system I/O errors	24
Recovery attempt	Chipset & system memory	Chipset & memory errors	26

A. Capturing Recovery Attempt and its Impact

In this section, we explain how correlations between NUMA (Non-Uniform Memory Access) and process resource use counters and between application memory leaks can be used to first infer error recovery, and then assess the impact of the recovery mechanism on the system reliability.

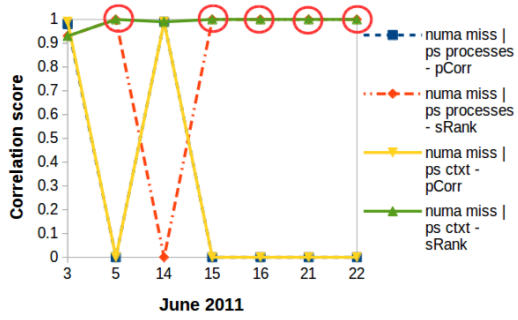
1) Phase 1: Correlated NUMA & Processes counters: The *numa* and *processes* resource use counters can be used to see what happens when a node runs out of free memory pages. If a process makes a request for memory pages on its local node but the node is out of memory (an error), then the `numa miss` counter on that node is incremented. Another node is assigned to accommodate the process request for memory pages and the `numa foreign` counter on the node is incremented (error recovery). The `ps processes` counter records the number of processes created, and the `ps ctxt` counter records the total amount of context switches across all CPUs. From Fig. 3, we observed that `numa miss` is strongly correlated to `ps processes` with scores that range between 0.93 to 1 on 25 dates, and `numa miss` is strongly correlated to `ps ctxt` with scores that range between 0.93 to 1 on 23 dates. From Fig. 4, we observed that `numa foreign` is strongly correlated to `ps processes` with scores that range between 0.93 to 1 on 21 dates, and `numa foreign` is strongly correlated to `ps ctxt` with scores that range between 0.93 to 1 on 21 dates. We observed that only Pearson correlation identified the correlated NUMA & processes counters on two dates. However, we observed that only Spearman-Rank correlation identified the correlated NUMA miss & processes counters on 17 dates and the correlated NUMA foreign & processes counters on 16 dates. If Pearson correlation is used as the only correlation method, the correlated NUMA & processes counters on these 16 dates would not be identified. However, if Spearman-Rank correlation is used as the only correlation method, the correlated NUMA & processes counters on the two dates would not be identified. Our results show that:

- Pearson correlation and Spearman-Rank correlation are suitable methods. Pearson correlation identified memory and process resource allocations that follow a linear pattern and Spearman-Rank correlation identified memory

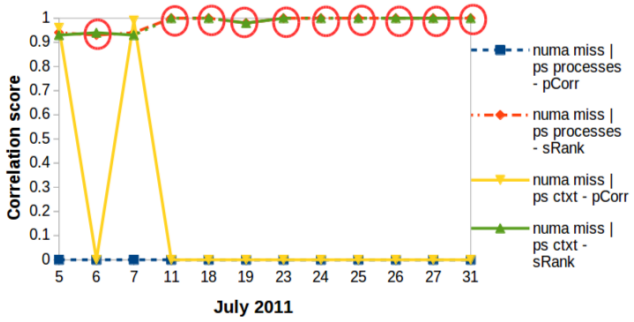
[4]*Case Study of Error Recovery and Error Propagation on Ranger*, available at https://tinyurl.com/yb6zvnsq

and process resource allocations that follow a monotonically increasing function.

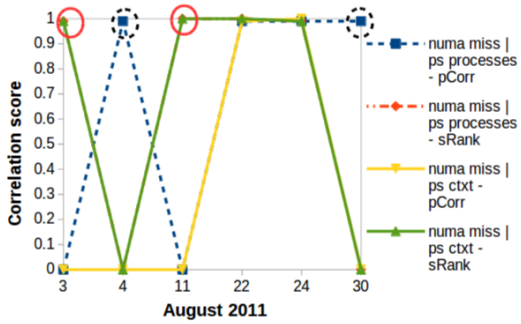
- When a process requests memory on out-of-memory nodes, the system attempts to recover by allocating memory on another node to the process. Further, when the system allocates memory on another node to a process, context switching occurs across all the CPUs.



(a) June 2011.



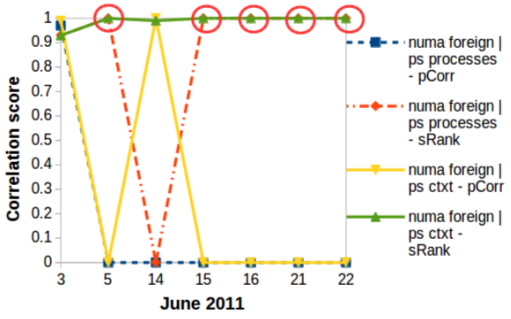
(b) July 2011.



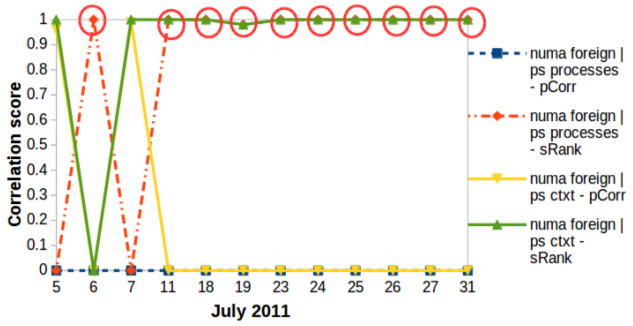
(c) August 2011.

Fig. 3. Correlations between "numa miss", "ps processes" and "ps ctxt" counters. The full-circled counters were identified by Spearman-Rank correlation only. The dot-circled counters were identified by Pearson correlation only.

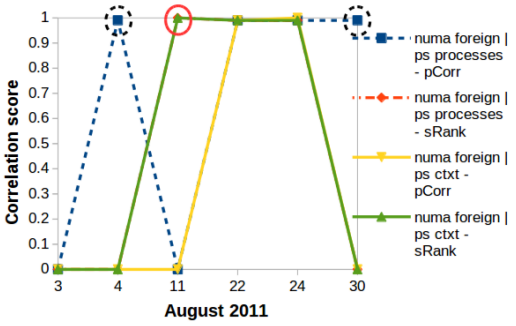
2) Phase 2: Correlated Segmentation Faults & General Protection Errors: Segmentation faults can be identified from the `segfault` event. Access violations can be identified from the `general protection error` event. Segmentation faults are often caused by programs that tried to read or write a protected memory location. When a program tries to access a protected memory location, it can trigger the processor to issue a general protection fault (GPF) interrupt. In most cases, the operating



(a) June 2011.



(b) July 2011.



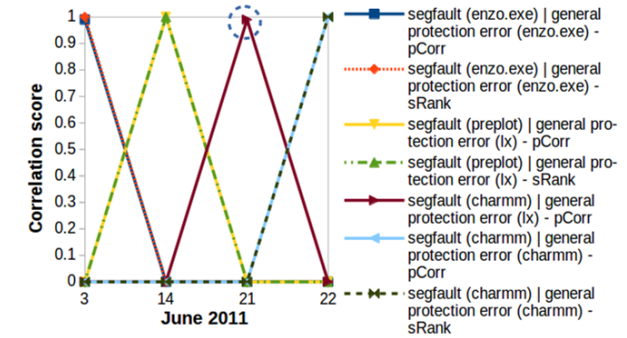
(c) August 2011.

Fig. 4. Correlations between "numa foreign", "ps processes" and "ps ctxt" counters. Full-circled counters were identified by Spearman-Rank correlation only, dot-circled counters were identified by Pearson correlation only.

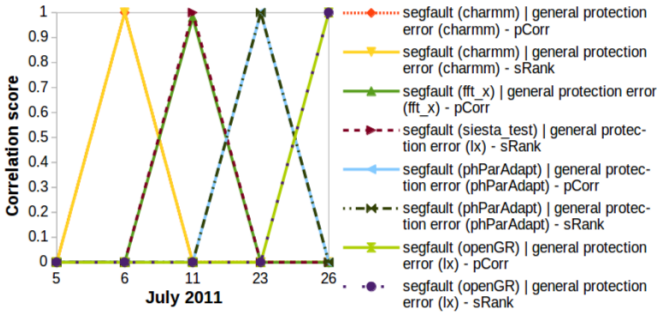
system will remove the program, signal the user and continue executing other programs. However, if the operating system fails to catch the GPF, the processor will issue a second GPF. If the operating system fails to catch the second GPF, the processor stops working and will only respond to a reset.

From Fig. 5, we observed that `segfault` events are strongly correlated to `general protection error` events with scores that range between 0.98 and 1 on 10 dates. The dates coincide with all the dates when NUMA miss and NUMA foreign counters are correlated. We observed that only Pearson correlation identified the correlated segmentation faults and general protection errors on June 21 and August 22.

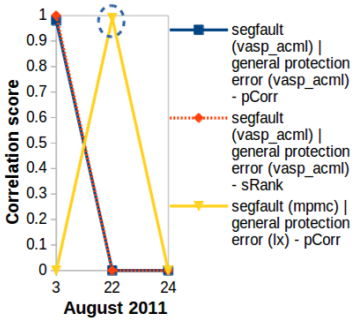
We identified the names of the programs which caused segmentation faults by implementing a function to scan the `general protection error` and `segfault` messages.



(a) June 2011.



(b) July 2011.



(c) August 2011.

Fig. 5. Correlations of "segfault" and "general protection error" events. The dot-circled events were identified by Pearson correlation only.

The programs are: `enzo.exe`, `preplot`, `charmm`, `fft_x`, `siesta_test`, `lx`, `phParAdapt`, `openGR`, `vasp_acml` and `mpmc`. `enzo.exe` is an executable of a cosmology simulation. `preplot` is a program that reformats the output of analysis programs so that they may be plotted by Gnuplot. `charmm` is a molecular simulation program. `fft_x` is a fast fourier transform algorithm. `siesta_test` is a JavaScript unit testing tool. `phParAdapt` is a parallel adaptive mesh method for the numerical simulation of multiphase flows. `openGR` is a framework that supports large numerical simulations in general relativity. `vasp_acml` is a Vienna Ab initio simulation package and AMD core maths library. `mpmc` is a Massively Parallel Monte Carlo method package.

Correlations with failures: Next, we manually scanned the list of correlated events to determine the correlation strength between `segfault` and `soft lockup` events, and `general protection error` and `soft lockup` events. A summary of the strongly correlated events is given in Table V. From Table V, we observed that the segmentation faults and general protection errors are associated with the programs `charmm` and `phParAdapt`, and the `segfault` and `general protection error` events are strongly correlated to soft lockup events on July 06 and 23. Our results suggest that access to protected memory location by the programs `charmm` and `phParAdapt` has led to memory location access violations and these access violations have led to soft lockups on July 06 and 23. Further, we found that the correlated segmentation faults and general protection errors are weakly correlated to soft lockups on eight dates.

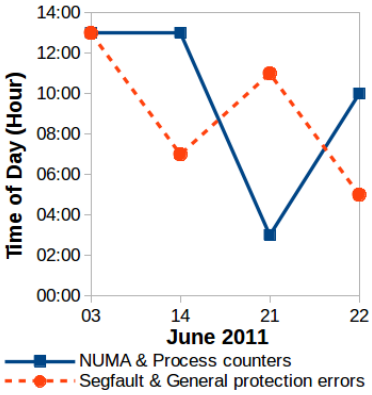
Detailed diagnosis: When the system was attempting a recovery caused by a NUMA miss, several applications attempted writes to protected memory locations which caused segmentation faults. On eight of ten dates the operating system removed the faulty application. This represents a recovery rate of 80%. However, the operating system did not catch the general protection faults that were triggered by two applications on two of ten dates which led to failure. This represents a failure rate of 20%.

TABLE V
SUMMARY OF CORRELATED "SEGFAULT" AND SOFT LOCKUP, AND CORRELATED "GENERAL PROTECTION ERROR" AND SOFT LOCKUP.

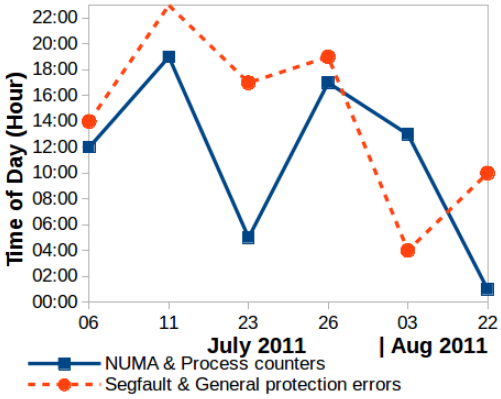
Error event	Failure event	Date	pCorr	sRank
segfault (charmm)	soft lockup	July 06	0.99	1
general protection error (charmm)	soft lockup	July 06	1	1
segfault (phParAdapt)	soft lockup	July 23	0.99	1
general protection error (phParAdapt)	soft lockup	July 23	0.99	1

3) Phase 3: Earliest times of change: From Fig. 6, we observed that the earliest times of change in the correlated NUMA & process counters and correlated segmentation faults & general protection errors on each date are different. The times of change: (i) occurred first in the correlated NUMA & process counters on six dates, (ii) occurred first in the correlated segmentation faults & general protection errors on three dates, and (iii) occurred in both the correlated counters and correlated errors at the same time on one date. If the correlated errors were used as the only source, the earliest times of change on six dates would not be identified. Having said that, if the correlated resource use counters were used as the only source, the earliest times of change on three dates would not be identified. Our results show that both the correlated resource use counters and correlated errors are required to identify the earliest times of system behaviour change on all dates. Further, we observed there are different time-windows between the times of change identified on all dates. The time-windows range from one-hour to 12-hours.

4) Validation: Next, we test the significance of: (i) the correlation coefficient of the strongly *positive* correlated resource use counter groups, and (ii) the correlation coefficient



(a) June 2011.



(b) July & August 2011.

Fig. 6. Times of change in the correlated NUMA & process counters and correlated segmentation faults & general protection errors.

of the strongly *positive* correlated error groups. We test all the correlation coefficients against the null hypothesis and obtained the z-scores for all the correlation coefficients and a summary is given in Table VI.

TABLE VI
SUMMARY OF z-SCORES. n CONTAINS THE NUMBER OF HOURLY TIME-BINS IN ONE DAY OF LOGS.

Correlated groups	June 2011	July 2011	Aug 2011
NUMA & Processes resource counters ($n = 24$)	$z_r = 10.68$	$6.15 \leq z_r \leq 10.68$	$z_r = 10.68$
Segmentation faults & General protection errors ($15 \leq n \leq 24$)	$z_e = 10.68$	$9.08 \leq z_e \leq 10.68$	$9.08 \leq z_e \leq 10.68$

From Table VI, we observed that the z-scores for all the correlation coefficients range from 6.15 to 10.68. At the 99% confidence level, under the null hypothesis $z_{0r} = 2.64$ and $z_{0e} = 2.64$. Hence, we reject the null hypothesis in favour of the alternate hypothesis.

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. From Table VI, we observed that the lowest z-score is 6.15. Since this is a one-sided test, the P-value is equal to the probability of observing a value greater than 6.15 in the standard normal distribution, or $P(Z > 6.15) = 1 - P(Z \leq 6.15) = 1 - 0.99999 = 0.00001$. The adjusted P-value is $0.00001 \times 25 = 0.00025$ where 25 is the number of dates. The P-value is less than 0.01, indicating it is highly unlikely this result would be observed under the null hypothesis. All the z-scores in Table VI are greater than or equal to 6.15 and all the adjusted P-values are less than 0.01, indicating it is highly unlikely these results would be observed under the null hypothesis.

V. RELATED WORK

A method that uses a feature construction scheme evaluates Pearson correlation and Spearman-Rank correlation based clustering to rank system log messages that are important for problem diagnosis was presented in [32]. In contrast, CORRMEXT evaluates Pearson and Spearman-Rank correlations to identify patterns of groups of resource use counters in resource use data and groups of errors in message logs. [33] proposes a metric that measures correlations of events and applies an algorithm called event correlation graphs on message logs of two HPC systems and predicted failure and non-failure events. In [14], a time-anomaly correlation approach called SIGs was developed to infer influences between interacting components in the system message logs. The Wilcox Rank-sum correlation method is applied on system performance data to monitor and predict processor failures in [19]. CORRMEXT complements these approaches by identifying patterns of groups of resource use and groups of errors.

[34] proposes a novel way for characterizing normal and faulty behaviour of large-scale cluster systems. The approach uses an Event Log Signal Analyzer (ELSA) module and applies the module on the event logs of two cluster systems. The approach uses two steps: (i) the first step identifies correlations between events, (ii) the second step detects anomalies in the times of the faulty signals. While both the approach and CORRMEXT study the behaviour of cluster systems, there are several differences. First, CORRMEXT implemented a three-step approach. Second, CORRMEXT identifies the time-windows of change in system behaviour by using the correlated resource use counter groups and correlated error groups. Third, CORRMEXT applies both Pearson and Spearman-Rank correlation on the resource use data and message logs.

Several tools such as IPLOM [35], LoGs [36], SLCT/Loghound [37] and SEC [38] have been developed to automate processing of system message logs. Fluentd[5] is an open-source data collector that process multiple application logs. [39] evaluates four log-parsing methods and package them into a toolkit for reuse. CORRMEXT complements these tools by processing standard Syslog messages, Rationalized message logs and TACC_Stats resource use data.

[25] analyze the RAS logs and job logs of a IBM BlueGene/P MPP system and report several interesting observations on the failure characteristics across different jobs. While

[5]URL: http://www.fluentd.org/

[25] and CORRMEXT combine analysis of multiple sources of data, there are several differences. First, CORRMEXT analyzed patterns of resource use and events associated with error recovery and error propagation. Second, it combined analysis of both the resource use data and message logs. Third, unlike the IBM BlueGene/P RAS logs the Ranger message logs do not contain any severity-level tags.

Pivot Tracing [40] provides system diagnosis by combining dynamic instrumentation with a novel relational operator. OpenTracing[6] provides system diagnosis by using a directed acyclic graph. While Pivot Tracing, OpenTracing and CORRMEXT all provide diagnosis of system problems, the approach implemented by Pivot Tracing, OpenTracing and CORRMEXT are different. First, CORRMEXT combines correlation algorithms and variance extraction. Second, CORRMEXT does not require additional instrumentation. Third, CORRMEXT identifies error propagation paths which led to both system recovery and failure. Because CORRMEXT provided a pathway to explain the cause of successful and failed error recovery mechanisms, better error handling techniques can be designed by targeting specific issues such as memory allocation and memory leaks.

[21], [22], [26] which are the predecessors of CORRMEXT, correlate errors only to failure events. If errors are correlated only to failure events, the recovery attempt and error propagation process would not be identified. CORRMEXT diagnosed three new dependability cases listed in Table IV. [21], [26] applies only Pearson correlation. In Section IV-A, we showed that only Spearman-Rank correlation identified the NUMA & processes resource use counters on 16 dates and only Pearson correlation identified the NUMA & processes resource use counters on two dates. If Pearson correlation was used as the only correlation method, the correlated NUMA & process resource use counters on the 16 dates would not be identified. However, if Spearman-Rank correlation was used as the only correlation method, the correlated NUMA & process resource use counters on the two dates would not be identified. [21] applies feature extraction on the resource use data and applies Pearson correlation on the message logs, and identifies the time-windows between error and failure events in the message logs. [22] applies Pearson and Spearman-Rank correlation on both the resource use data and message logs but it did not identify the time-windows of change in system behaviour. Differently to [21], [22], CORRMEXT applies Pearson and Spearman-Rank correlation *and* variance extraction to both the resource use data and message logs. CORRMEXT identified the correlated groups of resource use counters and correlated groups of errors, and it showed that Pearson correlation and Spearman-Rank correlation are suitable methods. Further, CORRMEXT identified the earliest times of system behaviour change during the day and it showed that the earliest times on *all* dates can only be identified by analyzing both the resource use data and message logs.

[6]http://opentracing.io/

VI. Discussion

In this section, we discuss one limitation of the CORRMEXT framework: Pearson correlation and Spearman-Rank correlation. CORRMEXT does not take into account spikes in resource use and error events. For example, when inodes on a file-system are corrupted and an application sends a large chunk of data to the file-system for writing, this can generate a large quantity of inode failure events and write_bytes on the file-system. Further, we do not take into account relationships between two events that occur in different time-bins. Having said that, we argue that this pertains to the problem specification described in Section III, i.e., *we seek to identify the resource use counters and events which are linearly or monotonically correlated on the specified date*.

VII. Conclusion and Future Work

We presented the CORRMEXT framework that correlated both the resource use data and message logs to identify (i) error propagation paths, (ii) reasons of unsuccessful error recovery and (iii) instances of successful error recovery. CORRMEXT diagnosed three dependability-oriented system problems and extracted the variance in the times of the correlated resource use counter groups and correlated error groups to identify the earliest occurrences of the problem. To ensure diagnostics accuracy, CORRMEXT applied the Bonferroni correction and showed that all the correlations are significant. We showed that CORRMEXT can identify recovery attempts and error propagation processes, and showed that knowledge of error propagation paths can be used to aid the systems designers and administrators to improve the effectiveness of error recovery protocols.

In our future work, we plan to extend our analyses of error recovery and error propagation to deal with errors other than memory, communication and file-system I/O errors.

Acknowledgements

We would like to thank the Texas Advanced Computing Center (TACC) for providing the Ranger cluster log data and granting access to their systems administrators. We also thank Karl Solchenbach (Intel Corporation, Europe) for granting access to his research scientists. This research is supported by The Alan Turing Institute under the EPSRC grant EP/N510129/1, The Alan Turing Institute-Intel partnership and the National Science Foundation under OCI awards #0622780 and #1203604 to TACC at the University of Texas at Austin.

References

[1] N. El-Sayed and B. Schroeder, "Reading between the lines of failure logs: Understanding how hpc systems fail," in *Proceedings of IEEE/IFIP DSN*, 2013, pp. 1–12.

[2] G. Wang, L. Zhang, and W. Xu, "What can we learn from four years of data center hardware failures?" in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2017, pp. 25–36.

[3] A. J. Oliner, A. Aiken, and J. Stearley, "Alert detection in system logs," in *Proceedings of IEEE ICDM*, December 2008, pp. 959–964.

[4] J. Stearly and A. J. Oliner, "Bad words: Finding faults in spirit's syslogs," in *Proceedings of IEEE CCGRID*, 2008, pp. 765–770.

[5] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, 2009.

[6] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in *Proceedings of IEEE ICDM*, 2007, pp. 583–588.

[7] E. W. Fulp, G. A. Fink, and J. N. Haack, "Predicting computer system failures using support vector machines," in *1st USENIX Workshop on the Analysis of System Logs*, 2008.

[8] F. Salfner and S. Tschirpke, "Error log processing for accurate failure prediction," in *1st UNIX Workshop on the Analysis of System Logs*, December 2008.

[9] Z. Lan, J. Gu, Z. Zheng, R. Thakur, and S. Coghlan, "A study of dynamic meta-learning for failure prediction in large-scale systems," *Journal of Parallel and Distributed Computing*, vol. 70, no. 6, pp. 630–643, 2010.

[10] A. Gainaru, F. Cappello, M. Snir, and W. Kramer, "Fault prediction under the microscope: A closer look into hpc systems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 77:1–77:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389101

[11] I. Fronza, A. Sillitti, G. Succi, M. Terho, and J. Vlasenko, "Failure prediction based on log files using random indexing and support vector machines," vol. 86, pp. 2–11, 2013.

[12] 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.

[13] T. Reidemeister, M. A. Munawar, M. Jiang, and P. A. Ward, "Diagnosis of recurrent faults using log files," in *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, 2009, pp. 12–23.

[14] A. J. Oliner, A. V. Kulkarni, and A. Aiken, "Using correlated surprise to infer shared influence," in *Proceedings of IEEE/IFIP DSN*, 2010, pp. 191–200.

[15] S. P. Kavulya, S. Daniels, K. Joshi, M. Hiltunen, R. Gandhi, and P. Narasimhan, "Draco: Statistical diagnosis of chronic problems in large distributed systems," in *Proceedings of IEEE/IFIP DSN*, 2012, pp. 1–12.

[16] 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.

[17] Q. Guan, D. Smith, and S. Fu, "Anomaly detection in large-scale coalition clusters for dependability assurance," in *Proceedings of IEEE HiPC*, 2010, pp. 1–10.

[18] G. Bronevetsky, I. Laguna, B. R. de Supinski, and S. Bagchi, "Automatic fault characterization via abnormality-enhanced classification," in *Proceedings of IEEE/IFIP DSN*, 2012, pp. 1–12.

[19] F. Salfner, P. Troeger, and S. Tschirpke, "Cross-core event monitoring for processor failure prediction," in *Proceedings of HPCS DMCC Workshop*, 2009, pp. 67–73.

[20] N. Gurumdimma and A. Jhumka, "Detection of recovery patterns in cluster system using resource usage data," in *Proceedings of IEEE PRDC*, 2017, pp. 1–10.

[21] 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. 1–10.

[22] 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, pp. 1–10.

[23] N. Gurumdimma, A. Jhumka, M. Liakata, E. Chuah, and J. C. Browne, "Towards increasing the error handling time window in large-scale distributed systems using console and resource usage logs," in *Proceedings of IEEE ISPA*, 2015, pp. 1–10.

[24] ——, "Crude: Combining resource usage data and error logs for accurate error detection in large-scale distributed systems," in *Proceedings of IEEE SRDS*, 2016, pp. 1–10.

[25] Z. Zheng, L. Yu, W. Tang, and Z. Lan, "Co-analysis of ras log and job log on blue gene/p," in *Proceedings of IEEE IPDPS*, 2011, pp. 840–851.

[26] 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.

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

[28] 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*, no. 9, 2010.

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

[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. [Online]. Available: http://dx.doi.org/10.1002/sim.6082

[32] S. Sabato, E. Yom-Tov, A. Tsherniak, and S. Rosset, "Analyzing system logs: A new view of what's important," in *2nd USENIX workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, 2007.

[33] X. Fu, R. Ren, J. Zhan, W. Zhou, Z. Jia, and G. Lu, "Logmaster: Mining event correlations in logs of large-scale cluster systems," in *Proceedings of IEEE SRDS*, 2012, pp. 1–10.

[34] A. Gainaru, F. Cappello, and W. Kramer, "Taming of the shrew: Modeling the normal and faulty behaviour of large-scale hpc systems," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, May 2012, pp. 1168–1179.

[35] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *Proceedings of ACM SIGKDD*, 2009, pp. 1255–1264.

[36] J. E. Prewett, "Listening to your cluster with logs," in *5th LCI International Conference on Linux Clusters*, 2004.

[37] R. Vaarandi, "Mining event logs with slct and loghound," in *Proceedings of IEEE/IFIP NOMS*, 2008, pp. 1071–1074.

[38] J. P. Rouillard, "Real-time log file analysis using the simple event correlator (sec)," in *Proceedings of 18th USENIX Conference on System Administration*, 2004, pp. 133–150.

[39] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "An evaluation study on log parsing and its use in log mining," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2016, pp. 654–661.

[40] J. Mace, R. Roelke, and R. Fonseca, "Pivot tracing: Dynamic causal monitoring for distributed systems," in *Proceedings of the 25th Symposium on Operating Systems Principles*, ser. SOSP '15. New York, NY, USA: ACM, 2015, pp. 378–393. [Online]. Available: http://doi.acm.org/10.1145/2815400.2815415