3-Dimensional Root Cause Diagnosis via Co-analysis

Ziming Zheng
Department of Computer Science
Illinois Institute of Technology
zzheng11@iit.edu

Li Yu
Department of Computer Science
Illinois Institute of Technology
lyu17@iit.edu

Zhiling Lan
Department of Computer Science
Illinois Institute of Technology
lan@iit.edu

Terry Jones
Oak Ridge National Laboratory
Oak Ridge, TN 37831
trjones@ornl.gov

ABSTRACT

With the growth of system size and complexity, reliability has become a major concern for large-scale systems. Upon the occurrence of failure, system administrators typically trace the events in Reliability, Availability, and Serviceability (RAS) logs for root cause diagnosis. However, RAS log only contains limited diagnosis information. Moreover, the manual processing is time-consuming, error-prone, and not scalable. To address the problem, in this paper we present an automated root cause diagnosis mechanism for large-scale HPC systems. Our mechanism examines multiple logs to provide a 3-D fine-grained root cause analysis. Here, 3-D means that our analysis will pinpoint the failure layer, the time, and the location of the event that causes the problem.

We evaluate our mechanism by means of real logs collected from a production IBM Blue Gene/P system at Oak Ridge National Laboratory. It successfully identifies failure layer information for 219 failures during 23-month period. Furthermore, it effectively identifies the triggering events with time and location information, even when the triggering events occur hundreds of hours before the resulting failures.

Keywords

Diagnosis, Co-Analysis

1. INTRODUCTION

1.1 Motivation

Recognizing the growing impact of failures on today's production HPC systems and the projected trends for the systems of tomorrow, resilience is identified as a critical challenge, among the other three challenges (power, concurrency, and memory/storage), for extreme-scale computing [16]. Root cause diagnosis plays a critical role for improving system resilience. Studies have shown that the average failure repair time ranges from a couple of hours to nearly 100 hours in production systems due to complications of various root causes [24, 31]. A timely and accurate diagnosis can significantly reduce failure repair time, thereby reducing the loss of processing cycles and maintenance cost [9].

Nevertheless, root cause diagnosis becomes increasingly challenging as systems continue to increase in scale and complexity. Despite of considerable studies on fault diagnosis, it remains an open problem, in particular on large-scale systems composed of hundreds of thousands of components. We identify two major issues with existing approaches. First, *existing works mainly focus on a single data source*, e.g., RAS (Reliability, Availability, and Serviceability) log [32, 23] or performance log [18], to trace fault-related information. The problem is that a single data source typically contains only limited information about the system and its operating environment, and consequently the information is inadequate to identify the actual root causes in many circumstances [24, 9, 34, 25]. Second, *most studies provide coarse-grained diagnosis* such as pinpointing the faulty node [32, 18] or suspicious events [7, 29]. There are many cases where such a coarse-grained root cause identification is not sufficient for effective fault management. For example, system managers are unable to assign appropriate recovery mechanism without knowing failure layer information [8, 9]; more detailed location information of the problematic component is essential for hardware replacement, especially given that a node may associate with number of hardware devices [9]; and accurate time information is necessary to understand time-delayed effects [23].

In this paper we present a novel root cause diagnosis mechanism for large-scale HPC systems. Distinguishing from existing studies, *our method has two distinct features.* First, it synthesizes fault related information from multiple data sources, rather than relying on one specific system log. In this study we examine three system logs, i.e., RAS log, job log, and environmental log. Here RAS log lists the fault related events, job log records job execution information, and environmental log typically provides numeric status values from the underlying hardware devices, such as temperatures, clock frequency, fan speeds, and voltages. They are representative system logs that are commonly collected on HPC systems. For instance, IBM Blue Gene comes with a dedicated monitoring and logging system to collect these logs [17]; the OVIS monitoring tool developed from Sandia National Lab can collect these data on various large-scale clusters [3]. Co-analysis of these logs can not only help us to understand the failure impact on different layers, but also assist us to pinpoint the root causes of failures. Second, it provides three dimensional fine-grained root cause information. Here, three dimension means that our root cause analysis can identify the failure layer (i.e., hardware, system software, or application), the time and the location of the events that cause the failures.

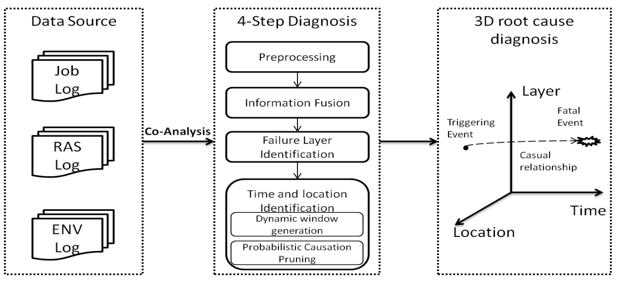


Figure 1: Overview of our root cause diagnosis mechanism.

1.2 Technical Challenges

Before presenting our detailed method, we list key challenges in the design of the proposed 3-D diagnosis mechanism.

- *Data volume.* Due to the tremendous system size, data collected for analysis are characterized by their huge volume [24]. These data generally often contain noises and redundant records. Hence, it is difficult to extract important diagnosis information from a large amount of data.

- *Data diversity.* As a single log alone does not contain sufficient information for failure analysis, we have to study multiple logs generated from multiple sources. However, these logs often have different formats and contexts, thereby making them hard to integrate.

- *System complexity.* Due to the complicated error propagation among components in different layers, it is difficult to distinguish the failure layer information [8, 33]. Furthermore, due to the lack of priori knowledge of system structure and component dependencies, it is infeasible to apply the well-known techniques based on causality graphs or dependency graphs for failure layer identification on large-scale systems [11, 4, 21].

- *Long latency.* For large-scale systems, triggering event may occur far away from the resulting fatal event, and as such a large amount of data may be recorded between the triggering event and its resulting failure event [7, 10, 20]. Hence, static and fixed approach is infeasible due to the long latency.

1.3 Paper Contribution

To address the above challenges, our design is based on co-analysis of multiple system logs, and it contains four interrelated steps as shown in Figure 1. First, *preprocessing* tackles the challenge of data volume by significantly reducing the amount of data for following data analysis. It removes redundant records and noises from raw logs and extracts important information for subsequent analysis. Second, to address the second challenge — data diversity, *information fusion* is utilized to synthesize the information from RAS log, job log and environmental log. Third, *layer identification* narrows down the search space by locating the layer (i.e., application, system software, or hardware) where the failure is generated. This step aims to address the third challenge — system complexity— by co-analyzing multiple logs on different layers, rather than analyzing the problem at each layer separately. Finally, *time and location identification* pinpoints the triggering event via dynamic tracing window and probabilistic causality pruning. The key feature of dynamic tracing window is that it can trace back to the triggering event that occurred days or even weeks ahead of the resulting failure, which is infeasible by using conventional fixed window methods. Hence, this step tackles the fourth challenge listed above, i.e., long latency.

We evaluate our methodology by means of real logs collected from a production IBM Blue Gene/P system at Oak Ridge National Laboratory. Our method is capable of classifying the root causes of 219 fatal events into three layers. Furthermore, it successfully identifies the triggering events for these fatal events, even in case that the trigger event occurs several weeks before the resulting failure. For example, it pinpoints a trigger event as an unstable current output in a faulty power module, which was reported 452 hours before the resulting failure. While our case studies are contributed to failure diagnosis in Blue Gene/P system, we believe our diagnosis methodology can discover more root cause information for variety HPC systems since it uses representative system logs.

1.4 Paper Outline

The rest of the paper is organized as follows. A brief discussion of related works is presented in Section 2. In Section 3, we provide background information about the Blue Gene/P system at Oak Ridge National Laboratory, and three system logs collected from this machine. In Section 4 we present the detailed description of our methodology. Section 5 discusses the case studies. Finally, we conclude the paper in Section 6.

2. RELATED WORK

To mitigate the impact of failures, increasing attention has been dedicated for automated root cause identification. Existing works mainly focus on node level fault localization [22, 18, 32, 13] and parallel programm debugging [11, 6, 4]. For example, the *nodeinfo* algorithm compares the frequency of message terms in each node to localize the faulty nodes [32]. In [18], PCA and ICA based methods are adopted to extract important features from system and application performance metric, and cell-based outlier detection is used to pinpoint the abnormal nodes. In [13], peer comparison diagnosis approach compares the I/O related metrics to identify the faulty node across I/O servers. In terms of parallel programm debugging, both [4] and [22] analyze the function call traces to identify the trace that is most different from others and pinpoint the suspect functions. In [11], DMTracker compares the frequency of data movement on process chain to identify abnormal data movements in MPI program. These methods generally analyze specific jobs or assume similar workloads. Our proposed method can work with them to identify the notorious jobs and workloads, then trigger the proper debugging progress. Meanwhile, our method can recognize faults unrelated to the applications such as hardware problem, thus avoid unnecessary debugging overhead.

In distributed systems, common solutions for diagnosis include fault propagation models (FPM) [15, 6] and trace comparison [5, 30]. These solutions typically assume that the system can be perturbed. Nevertheless, instrumentation is often prohibitive in production HPC systems [23]. Thus it is difficult to generate and maintain an accurate model, especially given the unprecedented system size and dynamic workload.

Statistical analysis is widely adopted to identify the symptoms associated with the failure. In [29], the message is highlighted for system administrator if it has more instances in the log than the expected value for computer system in normal status. In [20], GIZA infrastructure is designed to use several statistical data mining techniques to troubleshoot the problems in IPTV systems. In [7], Pearson correlation is adopted to extract relevant events from the system logs, and tupling heuristic method is used to construct the episode in event sequence. Distinguished from these studies, this paper not only uses correlation analysis, but also studies the job log and environmental log for triggering events identification. Furthermore, we use the probabilistic causation based method to screen off false triggering events.

Considerable research has demonstrates the effectiveness of co-analysis of multiple data sources [25, 18, 34]. For example, both [25] and [18] collect the data from OS level and application level for anomaly analysis. In [2], fine-grained events generated by kernel, middleware and application components are monitored to construct concise workload models. In our previous work [34], we present a co-analysis method to study RAS logs and systemwide job logs in Blue Gene/P system. In this paper, we extend [34] by integrating the environmental data for root cause diagnosis.

Research in this paper is inspired by the work of Oliner [23]. In [23], Oliner et.al. present a Structure-of-Influence Graph (SIG) to isolate system misbehavior [23]. They calculate the anomaly signal in each component and use statistical correlation with time-delayed effect to identify the potential faulty components. Our work fundamentally distinguishes from Oliner's work in three key aspects. First, their work mainly focused on analyzing RAS events for root cause analysis. However, RAS logs only contain limited information, which is inadequate in understanding failures and system behaviors [24]. Our work integrates the RAS information with job log and environmental data to understand the underlying system operating environment and the impact of failures on multiple layers. Second, we explore different methods to provide three dimensional fine-grained root cause information. Especially, our study considers much larger range of time delay between the triggering event and the resulting fatal event, which may vary from seconds to days [10].

3. BACKGROUND

In this section, we briefly describe Blue Gene/P and the system logs used in our experiments.

3.1 Eugene: Blue Gene/P System at ORNL

Eugene is a 2-rack Blue Gene/P system laid in two rows

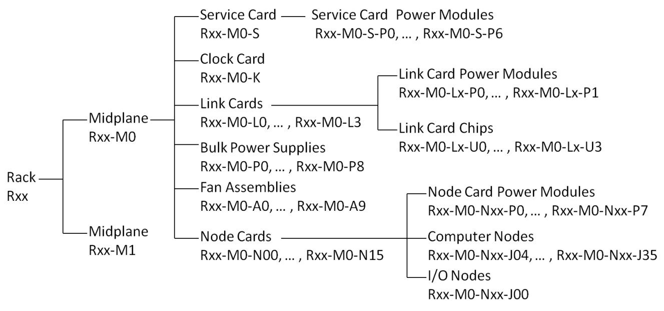


Figure 2: The hierarchy of hardware components and naming convention in *Eugene*.

(i.e., R0 to R1). The system consists of 8,192 compute nodes with a total number of 32,768 cores, offering a peak performance of 27.9 TFlops [1]. In Eugene, each rack has two midplanes (i.e., M0 to M1), which consists of 16 node cards, 4 link cards, 1 service card, 1 clock card, 9 bulk power supplies, and 10 fan assemblies. In each node card, there are 32 compute nodes connected into a 3D torus for communication, which are served by 1 I/O node and 8 power modules. In each link card, there are 6 link card chips and 2 power modules. The service card is served by 7 power modules. In summary, there are 12 types of hardware components in each midplane. Figure 2 illustrates the hardware components and their hierarchical relationship in the *Eugene* machine.

In Blue Gene/P, a dedicated CMCS (Core Monitoring and Control System) is responsible for system monitoring and error checking. It acquires specific software and hardware information directly through the dedicated control network. Monitored information is stored in a back-end DB2 repository. Three logs generated by CMCS in *Eugene* during the three-month period (i.e., from 2009-11-05 to 2010-02-05) are used in this study, and they are environmental data, RAS log and job log.

3.2 Environmental data

On Blue Gene/P, the environmental monitors read status information from the cards monitored and store the data in the environmental database in a frequency of every 300 seconds. The environmental data are stored in 12 tables, each of which represents one type of hardware components in *Eugene* (see Figure 2). For each hardware component, the sensors collect several environmental features such as temperature, current, and voltage. The total number of features used in our experiments is 3214. An example of environmental data is shown in Figure 3.

3.3 RAS Log

On Blue Gene/P, RAS records are generated when CMCS notices special events (e.g., when abnormal readings are encountered) from compute nodes, I/O nodes, and various networks. The entries in the RAS log include hard errors, soft errors, machine checks, and software problems [17]. An example of RAS events is shown in Table 1.

- *RECID* is the sequence number for an event record in the log, which is increased when a new record is added to the log.
- *MSG_ID* indicates the source of the message.

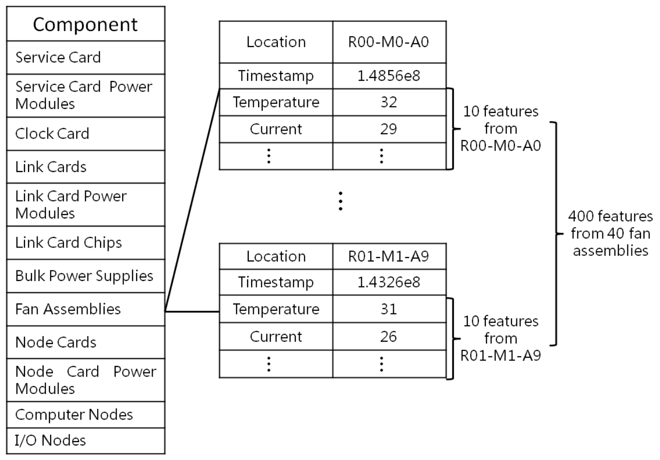


Figure 3: Environmental data is stored in 12 tables corresponding to 12 types of hardware components. Here we show an example from fan assemblies. In *Eugene*, there are 40 fan assemblies, and 10 features collected from each location, resulting in 400 features in total.

Items	Content
RECID	2457581
MSG_ID	MMCS_0101
COMPONENT	MMCS
SUBCOMPONENT	MMCS_OPERATIONS
ERRCODE	BGPMASTER_STARTED
SEVERITY	INFO
EVENT_TIME	2009-09-09-10.28.03.006954
FLAGS	DefaultControlEventListener
LOCATION	R00-M1-N1
SERIANUMBER	44V4173YL11K8021017
MESSAGE	BGPMaster has been started. $\cdots$

Table 1: RAS data from *Eugene*

- *COMPONENT* is the software component detecting and reporting the event. The COMPONENT could be APPLICATION, KERNEL, MC, MMCS, BAREMETAL, CARD, or DIAGS.
- *SUBCOMPONENT* indicates the functional area that generated the message for each component.
- *ERRCODE* identifies the fine-grained event type information.
- *SEVERITY* can be DEBUG, TRACE, INFO, WARNING ERROR and FATAL, with increasing severity.
- *EVENT_TIME* specifies the start time of event.
- *LOCATION* refers to the location where the event occurs.
- *MESSAGE* is a brief overview of the event condition.

3.4 Job Log

The job log of *Eugene* is collected by CMCS as well. An example of job information from the *Eugene* job log is shown in Table 2.

- *JOB_ID* is the sequence number for a job.

Items	Content
JOB_ID	7
EXECUTABLE	mpirun.26838.bgpsn
START_TIME	2007-10-18-21.51.16.627593
END_TIME	2007-10-18-21.51.21.789395
LOCATION	R01-M0-N12-128
USERNAME	bgpadmin

Table 2: Job log from *Eugene*

- *EXECUTABLE* indicates the directory and the name of executable file.
- *START_TIME* is the time when the job starts to run on the nodes.
- *END_TIME* is the time when the job exits. The job could be finished or interrupted by a failure.
- *LOCATION* refers to the location of the execution.
- *USERNAME* is the user name.

4. METHODOLOGY

4.1 Preprocessing

Raw logs cannot be directly used because they generally contain redundant records and noisy data. As a result, preprocessing is applied on raw logs to generate clean logs as well as to extract important information before further analysis [34]. Given that different system logs have different formats and characteristics, in this study, we apply different preprocessing mechanisms on them. Specifically, we apply wavelet transformation, temporal-spatial filtering, and categorization tree on environmental log, RAS log, and job log respectively.

With regard to environmental data, preprocessing has two main goals. First, as most features contain noisy signals, it is essential to filter out the noise and to extract key tendencies of signals. Second, it is critical to identify the timestamps of the abrupt change points, which generally indicate more diagnosis related information than stable signals. In signal processing, wavelet transformation [28] is a popular tool to serve these two purposes. The wavelet transform decomposes the original data into approximation coefficients and detail coefficients. The key tendencies are stored in the approximate coefficients, while the information of abrupt change points are stored in the detail coefficients. An example of wavelet transformation based preprocessing on environmental data is shown in Figure 4.

RAS log typically contains large volume of redundant records, and temporal-spatial filtering is a widely adopted method for removing its redundant data [24, 34]. The filtering method presented in our previous work is used for preprocessing RAS log [33]. Distinguishing from other temporal-spatial filtering methods [24, 19], our filtering method is capable of preserving important diagnostic information, such as event start time, event end time, and event locations.

With regard to job log, the execution history of user jobs with the similar characters can provide important information for diagnosis. As a result, the purpose of job log preprocessing is to group the jobs via analyzing their principle characters. In particular, we develop a categorization tree based method to group the jobs. We first divide job records

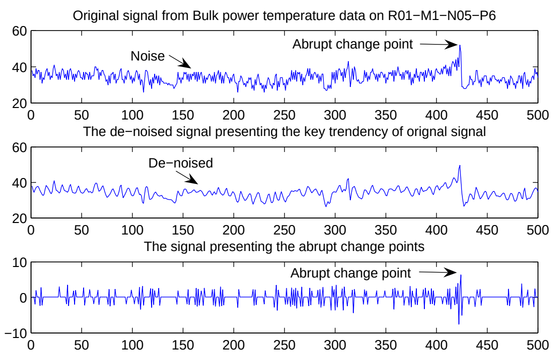


Figure 4: The wavelet based preprocessing of Bulk power temperature data on R01-M1-N05-P6.

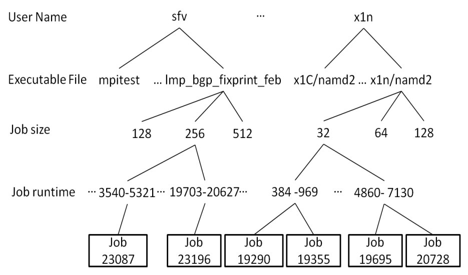


Figure 5: Job categorization tree.

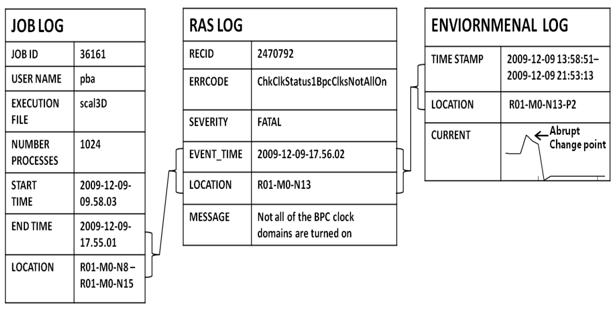
into high-level classifications based on user names, and then further group jobs into subcategories based on executable files, job sizes, and job runtimes.

As shown in Table 2, user names, executable files, and job sizes can be directly obtained from the job log. In terms of job runtimes, further processing is needed for data discretization. The observation on job runtimes is that the jobs from the same buggy application are usually interrupted after the similar execution periods [34]. As a result, we use DBSCAN clustering algorithm [12] to divide the job runtimes into different groups. DBSCAN is a density-based algorithm which identifies the clusters based on estimation of the density distribution. One advantage of DBSCAN is that we do not need to know the number of clusters at the beginning of the process. After DBSCAN clustering, the ranges of job runtimes are depicted on the lowest category level of the categorization tree. An example of job categorization tree is shown in Figure 5.

4.2 Information Fusion

After preprocessing, information fusion is applied to integrate multiple logs. For the purpose of failure diagnosis, we consider the RAS fatal events as the central events and connect them with the interrupted jobs and the subset of environmental features related to the fatal events. Formally speaking, for each fatal event F, information fusion generates a 3-tuple $< F, J_F, E_F >$, where J_F is the set of jobs interrupted by F and E_F is the subset of environmental features that are informative in depicting F.

In terms of job log, we can directly identify the job interruptions by matching the corresponding *Event_Time* and *Location* attributes of the fatal events and jobs. In terms



JOB LOG	
JOB ID	36161
USER NAME	pba
EXECUTION FILE	scal3D
NUMBER PROCESSES	1024
START TIME	2009-12-09-09.58.03
END TIME	2009-12-09-17.55.01
LOCATION	R01-M0-N8 – R01-M0-N15

RAS LOG	
RECID	2470792
ERRCODE	ChkClkStatus1BpcClksNotAllOn
SEVERITY	FATAL
EVENT_TIME	2009-12-09-17.56.02
LOCATION	R01-M0-N13
MESSAGE	Not all of the BPC clock domains are turned on

ENVIORNMENAL LOG	
TIME STAMP	2009-12-09 13:58:51– 2009-12-09 21:53:13
LOCATION	R01-M0-N13-P2
CURRENT	Abrupt Change point

Figure 6: An example of information fusion.

of environmental log, a simple strategy using time and location matching is infeasible. Time delay is common between hardware sensor readings and RAS recording because of the different collection mechanisms. For instance, the records in RAS log and job log are collected by event-driven mechanism, whereas environmental log is periodically collected in every 300 seconds. Furthermore, with regard to location dimension, there are 12 types of hardware components with different location granularities in the environmental log (see Figure 2), which is inconsistent with the location information in RAS log.

To address these issues, we develop an algorithm to extract environmental features that are close to the fatal event in both time and location dimensions (see Algorithm 1). In the time dimension, a time window W is assigned to cover the values of features in $[T_F - W, T_F + W]$, where T_F is the start time of fatal event F. Due to the possible interval caused by delay between hardware sensor readings and RAS recordings, we set W as long as one hour to capture the main tendencies of the environmental features nearby T_F. In the location dimension, for each type of hardware component listed in Figure 2, our algorithm identifies the specific components sharing the least common ancestor (LCA) with the fatal event. For example, if the location of fatal event L_F is R01-M0-N13-J23, all the node card power modules from R01-M0-N13-P0 to R01-M0-N13-P7 are examined in our algorithm as they share LCA R01-M0-N13 with L_F.

Note that Algorithm 1 only keeps the environmental features with abrupt change points, which are more informative in depicting fatal events than the other features.

Algorithm 1 Information fusion between RAS log and environmental log

```
Let $T_F$ and $L_F$ be the time and location of fatal event $F$
in RAS log.
Let $L_e$ be the location of environmental feature $e$.
$E_F \leftarrow \emptyset$
for each component C do
  if $L_e \in LCA(C, L_F)$ then
    if  $e$ has abrupt change point in $[T_F - W, T_F + W]$
    then
      $E_F \leftarrow E_F \bigcup e$ in C
    end if
  end if
end for
```

An example of information fusion is presented in Figure 6.

A BPC clock fatal event was reported in a computer code. It interrupted job 36161. There are six environmental features associated with it, one of which is the output current from the node card power module.

4.3 Failure Layer Identification

In this study, we distinguish the source of failure into three different layers, i.e., application, system software, and hardware. Application failures denote the fatal events introduced by users, such as buggy codes or user operation mistakes. System software failures denote the fatal events that are generated from system software like the operating system or middleware. Examples include kernel panic and network packet error. Hardware failures denote the fatal events originated from hardware facilities, such as power module or link card. Note that the accurate failure layer information is not provided by the failure logs directly in many cases [34, 32]

To identify failure layer, our method examines failures of the same type. With regard to application failure, the key rationale is that users tend to resubmit their problematic job upon job interrupt, thereby resulting in a series of failure events [34]. Specifically, we compare the jobs interrupted by the failures of the same type and identify the application failure if the interrupted jobs show similar characters, e.g., user names, executable files, job sizes, and job runtimes. In this study, we use the categorization tree (see Figure 5) to measure job similarity score $JS(i,j)$ between job i and job j. If two jobs are grouped in the same category at the lowest level, i.e., with the same user, executable file, job size and job runtime, they are considered as extremely similar jobs with score of 3. On the other hand, if two jobs have different executable files, job sizes, and job runtimes, they have a score of 0.

Based on the pairwise value of $JS(i,j)$, we further analyze all the interrupted jobs caused by the same type of fatal events together. Suppose there are n jobs interrupted by a specific type of fatal events, we define the overall job similarity score JSA as follows.

$$JSA = \frac{\sum_{i=1}^{n} \max_{j=1:n, j\neq i} JS(i,j)}{3n} \tag{1}$$

While application failure usually leads to high JSA, JSA alone cannot be used to distinguish between application failure and system failure. For example, when a hardware failure occurs, a scheduler may keep assigning the failed node to the interrupted job [34], which can also lead to a high score of JSA. To avoid misclassification, our method also examines location information for layer identification. A fatal event is considered as application failure if both of the following conditions are satisfied (1) JSA is close to 1.0 and (2) there exist two associated interrupted jobs which exhibit the highest similarity score of 3, but have different locations.

To distinguish hardware failures, our key rationale is that hardware failures of the same type generally show similar waveforms of environmental features from the corresponding hardware component. Specifically, for each fatal event, our method compares the environmental features associated with it and the failures of the same type. In this study, the normalized cross-correlation [23] is adopted to measure the similarity of environmental features. For a pair of time series of environmental feature $x(t)$ and $y(t)$, the normalized cross-correlation $ES(x,y)$ is defined as

$$ES(x,y) = \frac{E[(x(t)-\mu_x)(y(t+\tau)-\mu_y)]}{\sigma_x \sigma_y} \tag{2}$$

where μ_x and μ_y are the mean values of $x(t)$ and $y(t)$, σ_x and σ_y are the the standard deviations of $x(t)$ and $y(t)$, and τ is the time-lag. In this study, we calculate $ES(x,y)$ only if both $x(t)$ and $y(t)$ have abrupt change points. τ is decided by the time difference between the abrupt change points in $x(t)$ and $y(t)$.

Suppose n fatal events of the same type are reported before F, we extract the matched features associated with all the $n+1$ events (i.e., F and n fatal events of the same type). For example, if the temperature feature from fan assemblies is identified to be associated with all the events, this feature is identified as a matched feature. Suppose there are m matched features, for each feature we calculate the the similarity score between the F and the n fatal events of the same type. We then select the feature exhibiting the highest value on average to calculate the overall environmental feature similarity score ESA as follows,

$$ESA = \max_{j=1:m} \left(\frac{\sum_{i=1}^{n} |ES(f_{Fj}, f_{ij})|}{n} \right) \tag{3}$$

where f_{ij} is the ith feature associated with the jth event, and f_{Fj} is the ith feature associated with the fatal event F. We classify a fatal event as hardware failure if its ESA is close to 1.0.

For a fatal event, which is not application failure or hardware failure, we classify it as a system software failure. Note that system software failure may also introduces high JSA score if it only interrupts limited jobs from a few users. However, as one system software failure usually interrupts multiple jobs from a number of users, the JSA of system software failure is generally lower than JSA of application failure.

4.4 Time and Location Identification

To pinpoint the time and location of root cause, our method aims at identifying the triggering event. To this end, our method analyzes the failures of the same type and their precursor events. In particular, our strategy consists two parts: *dynamic window generation* to dynamically set the time interval preceding the fatal event for diagnosing, and *probabilistic causality pruning* to find the precursor event causing the fatal event based on correlation analysis and probabilistic causality analysis.

4.4.1 Dynamic Window Generation

Existing studies mainly use a fixed time window before the fatal events for diagnosing (e.g., a couple of hours) [7, 20, 29]. Nevertheless, such a static and fixed approach is not effective since the time delay between the triggering event and the resulting fatal event may vary from seconds to days [10]. To address this issue, we develop a dynamic window generation scheme where the tracing window is dynamically tuned based on event correlation analysis.

With regard to hardware failures and system software failures, the window size is dynamically adapted based on the time interval between two *adjacent* fatal events. Two fatal events A and B are adjacent if they are reported at the same location, and no other fatal event occurring between them at the same location. Further, the time window will be dynamically extended if two adjacent fatal events have

Algorithm 2 Dynamic window generation for hardware failure and system software failure

```
Let $F_1, F_2, \cdots, F_n$ be $n$ fatal events of the same type.
Let $T_{F_i}$ and $L_{F_i}$ be the time and location of fatal event $F$.
Let $A_i$ be the adjacent fatal event of $F_i$, $L_{A_i} = L_{F_i}$, $T_{A_i} < T_{F_i}$. Let $C_{A_i}$ be the failure type of $A_i$.
for i=1:n do
    $W_{F_i} \leftarrow T_{F_i} - T_{A_i}$
end for
Split RAS log by max($W_{F_i}$)
for i=1:n do
    if $lift(C_{A_i}, F) > 1$ then
        $W_{F_i} \leftarrow W_{F_i} + W_{A_i}$
    end if
end for
return $W_{F_i}$
```

different event types but show positive correlation (see Algorithm 2). Here the correlation between event A and B is measured by *lift* [33, 27] as follows.

$$lift(A, B) = \frac{P(AB)}{P(A)P(B)} \quad (4)$$

To estimate the probabilities $P(A)$, $P(B)$, and $P(AB)$, we split the RAS log into different slices by the maximum size of the time window. Suppose there are n windows, in which m windows contain event A, k windows contain event B, and r windows contain both A and B, then $P(A) = m/n$, $P(A) = k/n$, and $P(AB) = r/n$. If *lift* is greater than 1.0, A and B are positively correlated, which generally indicates causal relationship between A and B. As a result, we extend the time window between A and B for further analysis.

In terms of application failure, the window size is decided by job execution time. In other words, all the events occurring between the job start time and the fatal event time are covered in the time window.

4.4.2 *Probabilistic Causation Pruning*

To pinpoint the triggering event, our method first determines a list of candidate events within the tracing-back window via correlation analysis, and then identify the event by applying probabilistic causality analysis. The list of candidate events is determined as follows. First, our method finds out potential triggering events based on the failure layer information. In terms of application failures, it finds out the precursor events associating with the same job as the fatal event. In terms of hardware failures, it identifies the precursor events exhibiting similar waveforms of environmental features as the fatal event. In terms of system software failures, we collect the events from the culprit nodes where the interrupted job is executed. Next, our method selects the events positively correlated with the fatal event, i.e., its *lift* with the fatal event is greater than 1.0.

Next, our method explores probabilistic causality to identify the actual triggering event by removing other false triggering events from the candidate list. Here false triggering event indicates the event sharing common cause of fatal event [14]. For example, in our case study, a kernel problem causes the network error, and finally leads to the network failure. While the network error event shows positive correlation with the resulting failure, it is not the triggering event. Based on probabilistic causality theory [26], we determine the actual triggering event as follows. Suppose both A and B are candidate events, our method sorts them based on their time stamps. If the ordering of A and B is not fixed, both A and B are kept as actual triggering event. If A always occurs before B and the occurrence of B does not raise the probability of F, i.e., $P(F|AB) \leq P(F|A\bar{B})$, then B is removed as the false triggering event.

Table 3: JSA scores of all the 59 fatal events. It is clearly shown that JSA is a good metric to distinguish application failures from other failures.

JSA	0-0.2	0.2-0.5	0.5-0.7	0.7-0.8	0.8-1
Application	0%	0%	0%	23%	77%
Hardware	76%	24%	0%	0%	0%
System software	13%	58%	29%	0%	0%

Table 4: ESA scores of all the 59 fatal events. It is clearly shown that ESA is a good metric to distinguish hardware failures from other failures.

ESA	0-0.2	0.2-0.5	0.5-0.7	0.7-0.8	0.8-1
Application	100%	0%	0%	0%	0%
Hardware	0%	0%	0%	24%	76%
System software	93%	7%	0%	0%	0%

5. CASE STUDIES

In this section, we present case studies of using our diagnosis method on real system logs collected from a production Blue Gene/P system at Oak Ridge National Lab. The details of these system logs are described in Section 3.

5.1 Failure Layer Identification

During the 23-month period, there are 219 fatal events. To verify our layer identification, we get the layer information from experts and split these events into training set and testing set. The training set consists of 7 hardware failures, 36 system software failures, and 16 application failures. We examine job similarity scores (JSA) and environmental feature similarity scores (ESA) of the fatal events in training set in Table 3 and 4. Based on the results from training, we classifying the failure layers in testing set, which consists of 13 hardware failures, 127 system software failures, and 20 application failures.

As shown in Table 3, in training set, JSA is a good metric to distinguish application failures from hardware failures because 100% application failures show $JSA > 0.7$ and 100% hardware failures show $JSA < 0.5$. However, JSA may lead to misclassification of system software failures because 27% system software failures with $JSA > 0.5$. This is because the scheduler may keep assigning failed nodes to a series of similar jobs [34]. Nevertheless, our mechanism cooperates the location information with the job similarity score for application failure identification, thus avoids misclassification of system software failures.

As shown in Table 4, in training set, 100% hardware failures show high environmental feature similarity score, i.e., $ESA > 0.7$. Meanwhile, all of system software failures and application failures have $ESA < 0.7$. As a result, ESA does a good job of distinguishing hardware failures from other

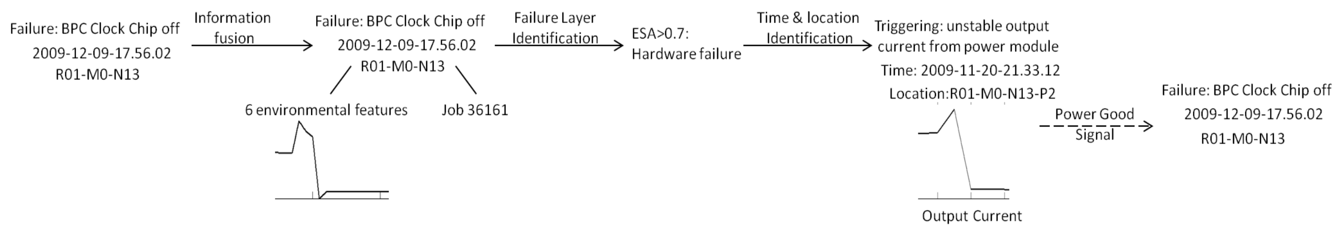

Failure: BPC Clock Chip off
2009-12-09-17.56.02
R01-M0-N13
Information fusion
Failure: BPC Clock Chip off
2009-12-09-17.56.02
R01-M0-N13
6 environmental features
Job 36161
Failure Layer Identification
ESA>0.7: Hardware failure
Time & location Identification
Triggering: unstable output current from power module
Time: 2009-11-20-21.33.12
Location:R01-M0-N13-P2
Output Current
Power Good Signal
Failure: BPC Clock Chip off
2009-12-09-17.56.02
R01-M0-N13

Figure 7: Diagnosis of BPC clock chip failure from hardware layer. The root cause is due to the unstable current output in the power module, which is reported 452 hours before the resulting failure.

failures.

Based on the training results, 0.7 of JSA and ESA are used to classify failure layers in testing set. In terms of application failure and hardware failure, our mechanism achieves 100% accuracy for all the 33 failures. In terms of system software failures, we successfully classify 87.4% failures. There are 16 failures with $JSA > 0.7$. However, the jobs interrupted by these 16 failures are reported from the same locations. By cooperating the location information, our mechanism avoids classification of them as application failures.

5.2 Time and Location Identification

Our mechanism analyzes the triggering events for the 219 fatal events. Here we list four case studies to illustrate the effectiveness of our time and location identification mechanism.

5.2.1 Hardware Failure

On December 9th 2009, a BPC clock failure *BpcClksNotAllOn* is reported in node card R01-M0-N13. The message shows that not all of the BPC clocks are turned on. As shown in Figure 7, our information fusion step identifies the interrupted job 36161 and 6 associated environmental features, such as output current and node card voltage. In failure layer identification step, this fatal event is classified as a hardware failure because $ESA > 0.7$.

In time and location identification step, dynamic window generation mechanism obtains the maximum time window of 656.7 hours. Then probabilistic causation pruning identifies two types of events with positive correlation with BPC clock chip failure, namely node power error and card power error. Next, the card power error is removed as a false triggering event because it always occurs after the node power error and does not raise the probability of BPC clock chip failures. Finally, the node power error from power module R01-M0-N13-P2 is identified as the triggering event. This type of error was continually reported 14 times and the earliest one occurred 452 hours before the resulting BPC clock chip failure.

Both triggering event and the BPC clock failure associate the environmental feature of output current from power module R01-M0-N13-P2. As a result, we believe the unstable output current is the root cause. This conclusion is consistent with the knowledge from hardware experts. When the power module cannot maintain proper outputs, a power good signal will be generated. When the BPC clock chip receives this signal, it will automatically shut down, thus lead to the occurrence of BPC clock chip failure.

5.2.2 Application failure

On June 2th 2009, a fatal event *bg_code_oom* is reported in R01-M0-N11-J00, which indicates an out of memory failure. As shown in Figure 8, our information fusion mechanism identifies the interrupted job 27487, which has executable file *pstg2r.x*. In failure layer identification step, this fatal event is classified as a application failure because $JSA > 0.7$ and there exist two associated interrupted jobs with highest similarity score but from different locations.

In time and location identification step, our dynamic window generation mechanism obtains the time window up to 707 hours. Then probabilistic causation pruning mechanism identifies the trigger event as an error of insufficient memory introduced by *pstg2r.x*, which was reported 669.7 hours before the resulting failure.

Obviously, it necessary to analyze the memory related functions in *pstg2r.x* for debugging. While debugging is outside of our study, our mechanism still provides useful information. The results from information fusion show a successful execution from the executable file *pstg1r.x*. As a result, user can compare the difference between these two versions to identify the potential bugs.

5.2.3 System Software Failure

On February 20th 2008, a torus sender failure is reported in 16 I/O nodes. The message shows that data was sent in torus network but not received. As shown in Figure 9, our information fusion mechanism identifies that the interrupted job 8406. In failure layer identification step, this fatal event is classified as a system software failure because $ESA < 0.2$ and $JSA < 0.5$.

In time and location identification step, dynamic window generation mechanism generates the maximum time window of 42.6 hours. Then probabilistic causation pruning identifies three types of events showing positive correlation with the torus sender failure, namely torus receiver warning, torus sender retransmission, and invalid memory address error.

While torus receiver warning and torus sender retransmission seem like more related to the torus sender failure based on the contexts, these two events are identified as false triggering events because they do not increase the probability of torus sender failure if invalid memory address error occurs. Finally, invalid memory address error is identified as the triggering event. This error was from Linux kernel, which was reported 42.3 hours before the resulting failure. When the receiver tried to receive the packet from torus network, it accessed invalid memory address thus dropped the data, which finally leaded to the occurrence of torus sender failure.

To diagnose torus sender failure, system administrators executed 135 testing programs in 506 hours. Our automated diagnosis method can significantly reduce the diagnosis

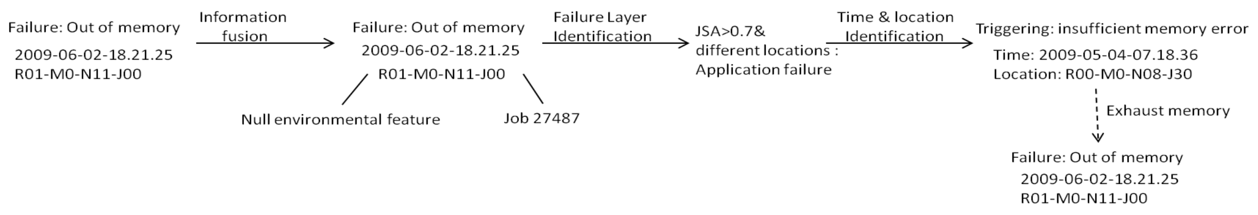


Figure 8: Diagnosis of out of memory failure from application layer. The root cause is an application bug that exceeds the limits of memory, which is reported 669.7 hours before the resulting failure.

overhead, thereby reducing the impact of failure on system resilience.

5.2.4 Rare Failure

While our methodology assumes multiple failures with the same type have occurred in history, it still can provide partial diagnosis information for rare failure via co-analysis. For example, on July 22nd 2009, a new failure *bg_code_panic* is reported, which indicates a kernel panic. Because it is a new failure, we can neither calculate the JSA/ESA nor adopt probabilistic causation analysis. To address the issue of rare failure, we simply use information fusion and dynamic window generation mechanisms. Our information fusion identifies that the *bg_code_panic* failure interrupted job 29780. In dynamic window generation step, we identify a machine check error from DDR controller, which is associated with the same job 29780 as the failure. As a result, the memory related operations in job 29780 are suspicious triggering event for further analysis.

6. CONCLUSIONS

In this paper, we have presented an automated mechanism for root cause diagnosis in HPC systems. Distinguishing from existing studies, our work effectively integrates information from multiple logs, and provides three dimensional fine-grained diagnosis information. Our case studies on a production Blue Gene/P system have demonstrated the effectiveness of our mechanism in terms of discovering failure layer information and triggering events with time and location information.

As a part of our future work, we plan to further test this diagnosis mechanism on more production HPC systems, including Cray XT5 and general HPC clusters.

References

[1] S. Alam, R. Barrett, M. Bast, M. Fahey, J. Kuehn, C. McCurdy, J. Rogers, P. Roth, R. Sankaran, J. Vetter, P. Worley, and W. Yu. Early evaluation of IBM BlueGene/P. *Proc. of Supercomputing*, 2008.

[2] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using magpie for request extraction and workload modelling. In *Proceedings of OSDI*, 2004.

[3] J. Brandt, A. Gentile, C. Houf, J. Mayo, P. Pebay, D. Roe, D. Thompson, and M. Wong. OVIS 3.2 user's guide. *SAND 2010-7109, Sandia National Laboratories*, October 2010.

[4] G. Bronevetsky, I. Laguna, S. Bagchi, R. Bronis, D. Ahn, and M. Schulz. AutomaDeD: Automata-based debugging for dissimilar parallel tasks. In *Proceedings of DSN*, 2010.

[5] M. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: problem determination in large, dynamic Internet services. In *Proceedings of DSN*, 2002.

[6] T. Chilimbi, B. Liblit, K. Mehra, A. Nori, and K. Vaswani. Holmes: Effective statistical debugging via efficient path profiling. In *Proceedings of ICSE*, 2009.

[7] E. Chuah, S. Kuo, P. Hiew, W. Tjhi, G. Lee, J. Hammond, M. Michalewicz, T. Hung, and J. Browne. Diagnosing the root-causes of failures from cluster log files. In *Proceedings of HiPC*, 2010.

[8] N. DeBardeleben, J. Laros, J. Daly, S. Scott, C. Engelmann, and B. Harrod. High-end computing resilience: Analysis of issues facing the HEC community and path-forward for research and development. *White Paper*, 2009.

[9] N. Desai, R. Bradshaw, C. Lueninghoener, A. Cherry, S. Coghlan, and W. Scullin. Petascale system management experiences. In *Proceedings of LISA*, 2008.

[10] A. Gainaru, F. Cappello, F. J., and S. Trausan. Adaptive event prediction strategy with dynamic time window for large-scale HPC systems. In *Proceedings of S-LAML*, 2011.

[11] Q. Gao, F. Qin, and D. Panda. DMTracker: Finding bugs in large-scale parallel programs by detecting anomaly in data movements. In *Proceedings of Supercomputing*, 2006.

[12] J. Han and M. Kamber. *Data Mining:Concepts and Techniques*. Morgan Kaufmann, 2000.

[13] M. Kasick, J. Tan, R. Gandhi, and P. Narasimhan. Black-box problem diagnosis in parallel file systems. In *Proceedings of FAST*, 2010.

[14] M. Khan, H. Le, H. Ahmadi, T. Abdelzaher, and J. Han. Dustminer: troubleshooting interactive complexity bugs in sensor networks. In *Proceedings of SenSys*, 2008.

[15] E. Kiciman and A. Fox. Detecting application-level failures in component-based internet services. *IEEE Trans. Neural Networks*, 16(5):1027–1041, 2005.

[16] P. Kogge and et al. Exascale computing study: Technology challenges in achieving exascale systems. *White Paper*, 2008.

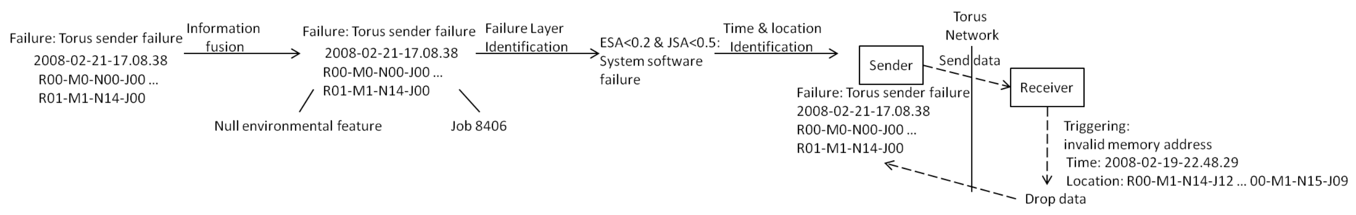

Failure: Torus sender failure
2008-02-21-17.08.38
R00-M0-N00-J00 ...
R01-M1-N14-J00
Information fusion
Failure: Torus sender failure
2008-02-21-17.08.38
R00-M0-N00-J00 ...
R01-M1-N14-J00
Null environmental feature
Job 8406
Failure Layer Identification
ESA<0.2 & JSA<0.5: System software failure
Time & location Identification
Torus Network
Sender
Send data
Receiver
Failure: Torus sender failure
2008-02-21-17.08.38
R00-M0-N00-J00 ...
R01-M1-N14-J00
Triggering:
invalid memory address
Time: 2008-02-19-22.48.29
Location: R00-M1-N14-J12 ... 00-M1-N15-J09
Drop data

Figure 9: Diagnosis of torus sender failure from system software layer. The triggering event is the invalid memory address error from torus receiver, which is reported 42.3 hours before the resulting failure.

[17] G. Lakner and G. Mullen-Schultz. IBM BlueGene solution: System administration. *IBM Redbook*, 2007.

[18] Z. Lan, Z. Zheng, and Y. Li. Toward automated anomaly identification in large-scale systems. *IEEE Trans. on Parallel and Distributed Systems*, 21(2):174–187, 2010.

[19] Y. Liang, Y. Zhang, A. Sivasubramanium, R. Sahoo, J. Moreia, and M. Gupta. Filtering failure logs for a BlueGene/L prototype. In *Proceedings of DSN*, 2005.

[20] A. Mahimkar, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and Q. Zhao. Towards automated performance diagnosis in a large IPTV network. In *Proceedings of SIGCOMM*, 2009.

[21] N. Maruyama and S. Matsuoka. Model-based fault localization: Finding behavioral outliers in large-scale computing systems. *New Generation Comput*, 28:237–255, 2010.

[22] A. Mirgorodskiy, N. Maruyama, and B. Miller. Problem diagnosis in large-scale computing environments. In *Proceedings of Supercomputing*, 2006.

[23] A. Oliner, A. Kulkarni, and A. Aiken. Using correlated surprise to infer shared influence. In *Proceedings of DSN*, 2010.

[24] A. Oliner and J. Stearley. What supercomputers say: A study of five system logs. In *Proceedings of DSN*, 2007.

[25] X. Pan, J. Tan, S. Kalvulya, R. Gandhi, and P. Narasimhan. Blind men and the elephant: Piecing together hadoop for diagnosis. In *Proceedings of ISSRE*, 2009.

[26] J. Pearl. *Causality: Models, Reasoning, and Inference.* Cambridge University Press, 2000.

[27] A. Pecchia, D. Cotroneo, Z. Kalbarczyk, and R. Iyer. Improving log-based field failure data analysis of multinode computing systems. In *Proceedings of DSN*, 2011.

[28] X. Rao, H. Wang, D. Shi, Z. Chen, H. Cai, and Q. Zhou. Identifying faults in large-scale distributed systems by filtering noisy error logs. In *Proceedings of DSNW*, 2011.

[29] S. Sabato, E. Yomtov, and A. Tsherniak. Analyzing system logs: A new view of what's important. In *USENIX SysML workshop*, 2007.

[30] R. Sambasivan, A. Zheng, M. Rosa, E. Krevat, S. Whitman, M. Stroucken, W. Wang, L. Xu, and G. Ganger. Diagnosing performance changes by comparing request flows. In *Proceedings of NSDI*, 2011.

[31] B. Schroeder and G. Gibson. A large-scale study of failures in high-performance computing systems. In *Proceedings of DSN*, 2006.

[32] J. Stearley and A. Oliner. Bad words: Finding faults in spirit's syslogs. In *Proceedings of the Workshop on Resiliency in High Performance Computing*, 2008.

[33] Z. Zheng, Z. Lan, B. Park, and A. Geist. System log preprocessing to improve failure prediction. In *Proceedings of DSN*, 2009.

[34] Z. Zheng, L. Yu, W. Tang, Z. Lan, R. Gupta, N. Desai, S. Coghlan, and D. Buettner. Co-analysis of RAS log and job log on Blue Gene/P. In *Proceedings of IPDPS*, 2011.