Co-analysis of RAS Log and Job Log on Blue Gene/P

Ziming Zheng, Li Yu, Wei Tang, Zhiling Lan
Department of Computer Science
Illinois Institute of Technology
{*zzheng11, lyu17, wtang6, lan*}*@iit.edu*

Rinku Gupta,* Narayan Desai,* Susan Coghlan,† Daniel Buettner†
**Mathematics and Computer Science Division*
†*Leadership Computing Facility*
Argonne National Laboratory
{*rgupta, desai*}*@mcs.anl.gov,* {*smc, buettner*}*@alcf.anl.gov*

***Abstract*—With the growth of system size and complexity, reliability has become of paramount importance for petascale systems. Reliability, Availability, and Serviceability (RAS) logs have been commonly used for failure analysis. However, analysis based on just the RAS logs has proved to be insufficient in understanding failures and system behaviors. To overcome the limitation of this existing methodologies, we analyze the Blue Gene/P RAS logs and the Blue Gene/P job logs in a cooperative manner. From our co-analysis effort, we have identified a dozen important observations about failure characteristics and job interruption characteristics on the Blue Gene/P systems. These observations can significantly facilitate the research in fault resilience of large-scale systems.**

***Keywords*-Co-Analysis; Blue Gene/P; Reliability; Log Analysis**

I. Introduction

The performance offered by high-end computing (HEC) systems has experienced tremendous growth. Petascale systems, such as the IBM Blue Gene and Cray XT series, are already available. Systems offering exascale performance are expected to become available in less than a decade. These high-end computing systems comprise hundreds of thousands of processing units and millions of hardware components. With the increasing system size and complexity, however, system failures have become a common scenario rather than an exception, thus making system reliability an important aspect in HEC [9].

When a system fails to function properly, Reliability, Availability, and Serviceability (RAS) logs are the primary source of information that a system administrator can use to understand failures. Thus, RAS logs have received considerable attention, resulting in research on log-filtering algorithms, failed events correlation, failure statistics, and practical and accurate failure prediction [9], [10], [11], [12]. These studies have facilitated fault management in large-scale systems.

Nevertheless, RAS logs typically contain only limited information about the system and the operating environment. Thus, analysis relying just on RAS logs has been inadequate in understanding failures and system behaviors [9]. HEC systems have several logs. An approach for co-analysis of multiple logs generated from the same system can substantially overcome the shortcomings of analysis that has been performed based simply on the RAS log. In this paper, we present a co-analysis method to study RAS logs and systemwide job logs in a cooperative manner. In particular, for our coc-analysis study, we examine the logs collected from the Intrepid Blue Gene/P system at Argonne National Laboratory, one of the largest Blue Gene/P systems in the world. Moreover, to spark interesting studies on co-analysis, we will release these logs in public repositories [28], [29].

Our co-analysis of job logs along with the RAS logs has three significant benefits. First, the impact of fatal or catastrophic events on application performance cannot be directly studied from RAS logs [9]. In this paper, *fatal events* are defined as the critical events that may lead to system or application crashes (system warnings or informative events can be classified as nonfatal events). Although some events in RAS logs are labeled as fatal events, not all of these interrupt user jobs [7], [9]. A better understanding of fatal events can greatly facilitate checkpointing, job scheduling, and failure recovery. For example, if a fatal event continually interrupts jobs, the system administrator should place high priority on fixing the problem. In addition, some fatal events may not be harmful to jobs while the system is idle or during a maintenance period [9], whereas other fatal events may interrupt multiple jobs as a result of failure propagation. Unfortunately, this information is usually invisible in RAS logs. Job logs can provide a straightforward way to identify these factors.

Second, while fatal events may be caused by system hardware or software errors or application bugs, the disparity between these two interpretations is critical. If the source of the fatal event is from system hardware or software, system administrators can take action. If the source of the fatal event

is an application bug, however, the programmer must take corrective or preventive action for the application. Unfortunately, although RAS records can provide information such as location, facility, and component [12], [1], they are too vague to pinpoint the source of fatal events. The use of job logs is helpful to identify this information.

Third, RAS logs typically contain numerous redundant records [9], [10], [11]. It is imperative to remove these redundant records for subsequent failure diagnosis and prediction [12], [7]. While temporal-spatial filtering [12] has been widely adopted to remove redundant records, it cannot identify the redundancy caused by jobs. For instance, a sequence of jobs may be assigned to the failed nodes repeatedly, and users tend to resubmit their buggy codes multiple times [13]. Consequently, a single fatal event may be reported multiple times in the RAS log. Without job information, checking RAS logs alone cannot reveal these correlations, thereby resulting in inaccurate failure models.

We present a co-analysis study based on a 273-day RAS log and job log collected from Argonne National Laboratory. Our co-analysis methodology consists of three tightly coupled steps. First, by correlating RAS events and job interruptions, we identify events that truly interrupt jobs. Second, we distinguish the fatal events generated by system hardware or software from the fatal events caused by application errors. Third, we remove job-related redundant RAS records by studying the correlations between job interruptions.

We have made several interesting observations about the characteristics of failure and job interruptions. First, high workload does not necessarily mean high failure rate. Instead, wider jobs (i.e., jobs requesting a large number of nodes) can significantly impact the failure rate. Second, we have found that most fault events do not propagate from one running job to other running jobs in different locations. Third, most of the job interruptions caused by application errors are reported early (i.e., less than one hour) in the execution period. Fourth, job size has more significant impact on job vulnerability on system failures, but longer jobs do not necessarily mean higher interruption probability than do shorter jobs in HPC.

The rest of the paper is organized as follows. In Section II, a brief discussion of the related work is presented. In Section III, we provide background information about the system, namely, the Blue Gene/P system "Intrepid" at Argonne National Laboratory. The logs are also described in this section. In Section IV, we present our co-analysis methodology. In Section V, we study failure characteristics, and in Section VI, we analyze job interruption characteristics. The key observations are highlighted at the end of the subsections. In Section VII, we briefly discuss some recommendations for system management. In Section VIII conclude the paper with a look at future work.

II. Related Work

Considerable research has been performed on system log analysis for large-scale systems. Schroeder and Gibson [8] studied the failure logs collected from 22 high-performance systems at Los Alamos National Laboratory, including 18 SMP clusters and 4 NUMA machines. Their failure data was stored in a dedicated *remedy database*, where system administrators recorded the basic information of the failures. Then they analyzed the statistical properties of the failures, such as the root cause, the failure rate, and the time to repair. Oliner and Stearley [9] examined the raw failure logs directly collected from five supercomputers. Instead of failure characteristics, their research focused on log structure analysis, failure identification, and log-filtering algorithms. Hacker et al. synthesized the methods in [8] and [9] to analyze the RAS logs from an 8192-processor Blue Gene system [10]. They designed a neural-gas filtering algorithm to identify the independent fatal events and studied the statistical properties of these events. Unlike these studies, we analyzed the Blue Gene/P failure log and job log in a cooperative manner. Not only do we provide effective fatal event identification and filtering methods for more accurate studies of failure characteristics, but we also analyze the impacts of failures on jobs.

Log preprocessing is critical for log analysis in large-scale systems. Liang et al. analyzed logs from an 8192-processor Blue Gene/L prototype at IBM Rochester using a temporal-spatial filtering algorithm [12]. In [9], temporal filtering and spatial filtering are adopted simultaneously. Both [12] and [9] use constant thresholds for filtering. A more adaptive filtering is presented in [4] by exploiting semantic correlation between the events with temporal gap. In [10], a neural-gas filtering algorithm is designed to identify the clustering in the spatial, temporal, and severity domains. In our previous work, we presented a causality-related filtering method to pinpoint the sets of fatal events co-occurring frequently and filter them together [7]. Distinguished from these studies, this paper uses the job information in preprocessing to remove job-related redundant records.

Both [8] and [10] found that Weibull distribution provides a good fit for the failure characteristic. Our work has the same conclusion but also reveals the impact of the job-related redundant records on the distribution parameters and numerical characteristics. The relationship between workload and failure rate in large-scale systems was studied in [17] and [8]. However, the systems in these studies are heterogeneous servers or clusters or NUMA machines, whereas we find a different conclusion in the Blue Gene/P MPP system.

Some research has been performed on failure propagation in large-scale systems. For instance, Xu et al. studied the propagation across the network in a 503-server, heterogeneous, distributed system [22]. In [23], Fu and Xu traced

Table I: Summary of the RAS log and job log from the Intrepid machine.

Log Name	Days	Start Date	End Date	Log Size	No. of Records
RAS	237	2009-01-05	2009-08-31	1.1 GB	2,084,392
Job	237	2009-01-05	2009-08-31	125 MB	68,794

the failure data from the LANL HPC coalition system to study failure propagation between different nodes. However, these systems are different from MPP systems such as Blue Gene/P. Moreover, our study focuses on the failure propagation across different jobs instead of nodes.

Research on the impacts of failures on applications is limited. In [11], Taerat et al. analyzed Blue Gene/L logs for six months. They used temporal filtering and studied the job interruptions through simulation. The impacts of failures and checkpointing on application efficiency were studied in [21]. However, these studies were based on simulations, while our study uses the real job log. In [24], Jiang et al. studied the characteristics of customer problem troubleshooting from large-scale storage systems. Our study does not focus on specific types of failures; instead, we analyze the impacts of failures on the jobs in the whole system.

III. Background

We briefly present here some background on the Intrepid system and on RAS and job logs from that system.

A. Intrepid: Blue Gene/P System at Argonne

IBM Blue Gene/P is a low-power MPP system scalable to 80 racks, with a peak performance above 1 petaflop. Each rack has two midplanes, which consist of 512 quad core PowerPC450 compute nodes, 1 service card, and 4 link cards. Compute nodes are connected into a 3D torus for communication. In Blue Gene/P, 64 compute nodes are served by an I/O node. A tree network is used to connect compute nodes to dedicated I/O nodes. The I/O nodes are connected through a 10-Gigabit Ethernet network to 136 file servers that, in turn, connect to back-end, Infiniband-based storage. More details of the system architecture are available in [1].

Intrepid is a 40-rack Blue Gene/P system operated by Argonne National Laboratory for the U.S. Department of Energy. The 40 racks are laid in five rows (i.e., R0 to R4). It consists of 40,960 compute nodes with a total of 163,840 cores, which offer a peak performance of 556 TFlops. It ranks #13 on the latest Top500 supercomputer list (as of November 2010) [2].

Intrepid uses a partitioned strategy for each job, which contains a distinct set of compute, I/O nodes, and the associated 3D torus network for communication. The midplane is the minimum partition for job scheduling, which can be joined with other adjacent midplanes as a larger partition [14]. Intrepid is a capability system, with single jobs frequently occupying a substantial number of midplanes. Both small jobs such as 1–2 midplanes and larger jobs such as 32 midplanes are common. Jobs up to 64 midplanes run without administrator assistance.

Table II: Example of event from Blue Gene/P RAS log.

RECID	13718190
MSG_ID	CARD_0411
COMPONENT	CARD
SUBCOMPONENT	PALOMINO_S
ERRCODE	DetectedClockCardErrors
SEVERITY	FATAL
EVENT_TIME	2008-04-14-15.08.12.285324
FLAGS	DefaultControlEventListener
LOCATION	R-04-M0-S
SERIANUMBER	44V4173YL11K8021017
MESSAGE	An error(s) was detectedby the Clock card : Error=Loss of reference input

A particular feature of the Blue Gene/P system is that the control system reboots the whole partition before the job execution. The main purpose of this feature is to minimize the amount of problematic states accumulated so far [15]. We are interested in whether this feature can impact the failure characteristic.

B. RAS Log and Job Log from Intrepid

We analyze a 273-day RAS log and job log collected from Intrepid. Table I summarizes these logs.

RAS Log: The Blue Gene/P Core Monitoring and Control System (CMCS) is responsible for monitoring the hardware components, including compute nodes, I/O nodes, and various networks. Monitored information is reported by CMCS as RAS event records. RAS records are stored in back-end DB2 databases, and the event stream from the back-end databases is used to provide efficient failure prediction facilities.

An example of an event record from the Intrepid RAS log is shown in Table II.

- *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.
- *COMPONENT* is the software component detecting and reporting the event. The COMPONENT could be APPLICATION, KERNEL, MC, MMCS, BAREMETAL, CARD, or DIAGS. APPLICATION indicates the running job; KERNEL indicates OS kernel domain; MC designates machine controller; MMCS designates control system on the service node; BAREMETAL is service-related facilities; CARD indicates card controller; DIAGS refers to diagnostic

Table III: Example of job information from Blue Gene/P job log.

Submission Time	05/01/2008 00:00:43
Job ID	8935
Job Name	N.A.
Execution File	N.A.
Queuing Time	1209614949.07
Starting Time	1209618043.1
End Time	1209621636.96
Location	R10-R11
User	N.A.
Project	N.A.

functions from the computing nodes or the service nodes.

- *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, or FATAL. DEBUG and TRACE events designate information to debug codes; these do not occur in our log. INFO, WARNING, ERROR, and FATAL represent events following the order of increasing severity level. INFO events provide information about the progress of system software, such as automatic recovery progress. WARNING events are usually recoverable "soft" errors, such as ECC correctable, single-symbol error. ERROR designates harmful events that might still allow the application to continue running, such as the failure of a redundant component. Only FATAL severity will presumably lead to application or system crash. Therefore we focus on events with "FATAL" 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. The information provided by this item includes component, location, and type of error.

During the 237 days studied, the raw log had a total of 2,084,392 records. In this study, we are interested in the 33,370 records with FATAL severity, which are reported with 82 types of ERRCODE from six types of COMPONENT.

Job Log: The job log of Intrepid is collected by the job scheduler Cobalt [14]. An example of job information from the Intrepid job log is shown in Table III.

- *Job ID* is the sequence number for an job.
- *Execution File* is the path of job execution file.
- *Queuing Time* is the time when the job is added in the waiting queue.
- *Starting 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. The minimum unit is one midplane.
- *Location* refers to the location of the execution. The minimum unit is one midplane.
- *User* is the user name.
- *Project* is the project name.

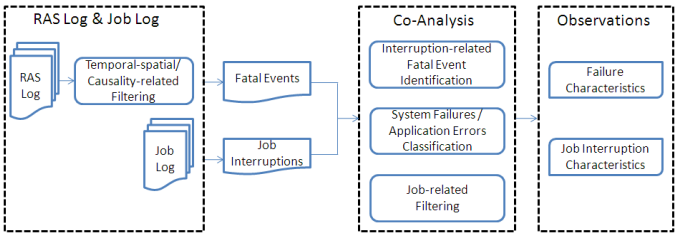

Figure 1: Co-Analysis methodology.

During the 237 days there are 68,794 jobs (i.e., having different job IDs). However, there are only 9,664 distinct job execution files because 5,547 have been submitted more than one time. We use the term *distinct job* in this paper and consider the job with the same execution file as one distinct job.

IV. Co-Analysis Methodology

Before presenting our methodology, we list a set of terms that are frequently used in the rest of the paper.

- An *interruption-related fatal event* denotes the critical event that can interrupt running jobs.
- A *system failure* denotes the failure that results from the system facilities instead of users, such as hardware failures and system software failures.
- An *application error* denotes the failure introduced by users, such as buggy codes and user operation mistakes.

Our co-analysis methodology is shown in Figure 1. We first trace RAS records to identify the events with the *FATAL* severity. These RAS messages cannot be directly used because they generally contain too much redundant information. Temporal-spatial filtering is a widely adopted method to remove redundant records [12], [9]; temporal filtering removes multiple events being reported from the same location within a threshold, and spatial filtering removes the same type of events being reported at different locations within a threshold. In this paper, we also adopt temporal-spatial filtering methods to remove redundant records. Furthermore, we apply a causality-related filtering to identify the sets of fatal events co-occurring frequently and filter them together [7]. After the filtering process, the number of records with FATAL severity is reduced from 33,370 to 549, with a compression ratio of 98.35%.

Based on filtering results, we identify the job interruptions by matching the RAS events with the job termination information in the job log. Since both the RAS log and the job log provide the time and the location information, we can trace the $End\ Time$ and $Location$ of the job and match the

corresponding $Event\_Time$ and $Location$ of RAS events to examine whether the job is interrupted by a fatal event. Through this method, we obtain 308 jobs (167 distinct jobs) interrupted by fatal events.

After obtaining the fatal events and the job interruptions, we apply our three-step method to correlate 549 RAS events with FATAL severity with 306 job interruptions. The details of these steps are described below.

A. Identification of Interruption-related Fatal Events

In RAS logs, a record with FATAL severity usually indicates a critical event [12]. The problem is that some events with FATAL severity may not interrupt any user jobs [9], [11]. Currently system administrators have to manually identify these false alarms [9], [7], a task that is time-consuming and tedious.

Fortunately, the information from job logs can help us identify fatal events that truly interrupt jobs. To identify these interruption-related fatal events, we consider three possible cases by correlating fatal events with jobs: (1) an RAS event with FATAL severity interrupts one or more jobs, (2) an RAS event with FATAL severity does not interrupt any job since no job runs on its location, and (3) an RAS event with FATAL severity does not interrupt any job running atop. For each event type, that is, the events with the same *Errcode*, we trace the job interruptions and identify the interruption-related fatal events using the following rules: (1) If only case 1 and case 2 are observed from the set of all events that have the same Errcode, then this Errcode indicates a type of interruption-related fatal event; (2) if only case 2 and case 3 are observed, then the corresponding Errcode indicates a type of nonfatal event for applications; (3) if only case 2 is observed or both case 1 and case 3 are observed, this type of events is undetermined, and further analysis is required.

In our case study, 31 types of RAS events with FATAL severity are identified as interruption-related fatal events. Two types of events are identified as nonfatal events: *BULK_POWER_FATAL* and *_bgp_err_torus_fatal_sum*. BULK_POWER_FATAL is a hardware-related alarm. It indicates an error in a bulk power module, which is detected by reading environmental data. However, the controller can only partially disable the rack and run hardware diagnostics on the bulk power modules. On Intrepid, the BULK_POWER_FATAL events are just transient errors, so the jobs can keep running after the diagnostics process. _bgp_err_torus_fatal_sum is a network-related alarm, which is reported as a hardware error in our log. These errors can be solved in a higher-level protocol, so the running jobs are protected. There are 49 types of events with FATAL severity that are undetermined because no job is running on the same location. In this study, we pessimistically consider these events as interruption-related fatal events [11].

Observation 1: Co-analysis can identify so-called fatal RAS events that do not really impact user jobs. In our case study, 20.84% of the RAS events belong to this category.

B. Classification of System Failures and Application Errors

Root cause analysis is of paramount importance for effective fault managements, especially for petascale systems such as Blue Gene/P [15]. A goal of root cause analysis is to distinguish system failures from application errors [20].

In RAS logs, the COMPONENT field tells where the event occurs. However, we found that 75% of fatal events are reported from the KERNEL, but no fatal event is reported from the APPLICATION domain. As a result, the COMPONENT field is insufficient to distinguish system failures from application errors. For example, we found two fatal events, *_bgp_err_cns_ras_storm_fatal* and *CiodHungProxy*, reported from the KERNEL domain. However, _bgp_err_cns_ras_storm_fatal is a system failure caused by L1 data cache parity error, while CiodHungProxy is an application error caused by a user operation mistake in the file system.

Again, co-analysis can help us distinguish system failures from application errors. We use the following rules for this purpose. First, if there is no job running on the same location as the fatal event, it is a system failure. As shown in Section IV-A, 49 types of fatal events belong to this case. Second, if multiple jobs are interrupted by the same fatal event in the same location, then this fatal event is a system failure. In this case, when a system failure occurs and interrupts the running job, the scheduler has no knowledge of this fatal event and continues to assign new jobs to the failed nodes. This fatal event tends to be continuously reported until the failure is fixed. We identify four types of fatal events in this scenario: *L1 cache parity error*, *DDR controller error*, *file system configuration error*, and *link card error*. Obviously, these fatal events are closely related to the system platform.

To identify application errors, we use the following strategy. When a fatal event occurs and interrupts a job at a specific location, the user tends to resubmit the job after the interruption. The scheduler may assign the previous location to a different job and assign the resubmitted job to a different location. If the fatal event is caused by an application error, the same type of fatal event will be reported at the newly assigned location, and the new job in old location will not be interrupted by the same type of fatal event. Figure 2 gives a specific example from our study. We have identified six types of application errors. Examples include *invalid memory address error*, *out-of-memory error*, *file system operation error*, and *collective operation error*.

For each unlabeled fatal event, we calculate and sort its Pearson's correlation coefficients [12] with existing categorized fatal types. We assign the category of fatal event with the highest correlation coefficient to these unlabeled fatal events. Our separation of application errors from system

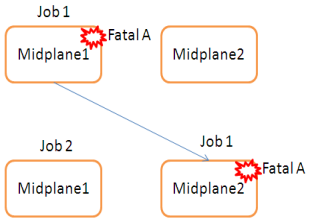

Figure 2: Example of how to identify application errors. In this example, job 1 is interrupted by fatal event A in both midplane 1 and midplane 2; job 2 has no interruption in midplane 1 during the period. Hence, fatal A is an application error introduced by job 1.

failures was verified and agreed upon by experienced system administrators at Argonne.

Observation 2: Co-analysis can distinguish system failures from application errors. In our case study, we identify 72 types of system failures and 8 types of application errors. Further, we find that 17.73% of fatal events are application errors.

C. Job-related Filtering

Although temporal-spatial filtering [12], [9] and causality-related filtering [7] are helpful in identifying independent fatal events, they fail to remove job-related redundancy. As shown in Section IV-B, job-related redundancies occur when the scheduler keeps allocating failed nodes for incoming jobs or when users keep resubmitting the buggy codes. In both scenarios, the same fatal events will be reported multiple times until the problems are fixed.

Job-related redundancy cannot be removed through temporal-spatial filtering, which sets a constant threshold for redundancy detection. The time interval between two job-related redundant system failures is determined by the job arrival rate and the scheduling policy. When the job arrival rate is low or the next executing is waiting for more computing resources, the redundant record may be reported with a long latency. In the case of application errors, the redundant record will not be reported until the user resubmits the job.

We have designed a simple job-related filtering strategy. When a fatal event is reported, we trace the location of the interrupted job. If another job is interrupted by the same type of fatal event in the same location and if no job executed between these two events, then this fatal event is considered redundant. Further, the relation is transitive. For example, if event B is redundant to event A and if event C is redundant to event B, both B and C are redundant to event A. In the case of application errors, the fatal event is considered redundant if the job with the same execution file has been interrupted by the same type of fatal event before.

In our case study, we identified 72 job-related redundant records in 549 records after temporal-spatial filtering and

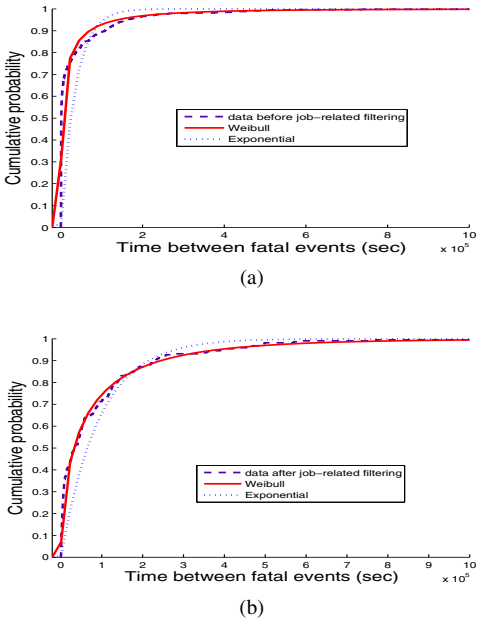

Figure 3: Expirical CDF for interarrival times of fatal events (a) with job-related redundant records and (b) without job-related redundant records.

causality-related filtering. Most of them come from the application, the kernel software, and the card controller on the service nodes. Our study demonstrates that the mechanism of "reboot before execution" on Blue Gene/P cannot solve these kinds of problems.

Observation 3: Job-related redundancy is not negligible. In our case study, we find that 57.4% of resubmitted jobs were allocated to the same failed nodes by the scheduler. In addition, users may keep submitting their buggy codes, thereby leading to the same type of application errors at different locations. Co-analysis can efficiently filter out these job-related redundant records in RAS logs. In our case study, we are able to remove the job-related redundant records with a compression ratio of 13.1%.

V. Failure Characteristics

Co-analysis of the RAS log and job log has also provided many insights into failure characteristics and job interruption characteristics. In this section, we present failure characteristics at the systemwide and midplane levels. Job interruption characteristics are presented in the next section.

A. Systemwide Failure Interarrival Distribution

Exponential distribution and Weibull distribution are two popular models used to fit failure interarrival [13], [8], [10]. In our study, we use both models to study the distribution of failure interarrival at the systemwide and midplane levels. Maximum likelihood estimation is adopted to parameterize

Table IV: Comparison of the parameters and numerical characteristics of Weibull distributions for fatal events before and after job-related filtering.

	Shape	Scale	Mean	Variance
Before job-related filtering	0.387,187	8116.7	29585	9.6348e+09
After job-related filtering	0.572,884	68465.9	109718	4.1818e+010

the two distributions [8], and a likelihood ratio test [16] is adopted to evaluate the effectiveness of these two distributions. To study the impact of job-related redundancy, we present in Figure 3 the fitting results with and without job-related redundant records. As shown in both plots, Weibull distribution give a better fitting than exponential distribution, consistent with the results presented in [8], [10]. Table IV presents the specific distribution parameters and numerical characteristics corresponding to the two curves.

Comparing Figure 3a and Figure 3b, we find that the two distribution curves are different. As shown in Table IV, the MTBF (mean time between failures) after job-related filtering is about three times larger than that without job-related filtering. Moreover, although both distributions have decreasing hazard rates (i.e., shape parameter <1), the value of the shape with job-related preprocessing is much higher than that without job-related preprocessing. Obviously, the impact of job-related redundant records is nontrivial.

Observation 4: While Weibull distribution gives a good fitting to represent failure interarrivals, the distribution parameters and numerical characteristics become significantly different after job-related filtering. In other words, job-related filtering is necessary for better understanding of failure modes.

B. Midplane-Level Failure Characteristics

In this subsection, we further study the midplane-level failure characteristics. Intrepid consists of 80 midplanes. We find that Weibull distribution still fits midplane-level failure interarrival distribution well. However, although Blue Gene/P is a homogeneous MPP system, the failure rates on different midplanes are significantly different. Figure 4a presents the number of fatal events in each midplane. As shown in Figure 4a, the midplanes from 33 to 64 have higher failure rates than do other midplanes. According to [8], [17], high workload (meaning the aggregated compute cycles are great) generally means high failure rate. But this situation is not true in our case study. In Figure 4b, we present the workload in 80 midplanes. The midplanes 1 and 2 and the midplanes in the range of 65 to 80 have higher workload than do other midplanes. However, as shown in Figure 4a, the number of fatal events in these midplanes is low. Instead, the three midplanes having the highest numbers of fatal events are 58, 61, and 60.

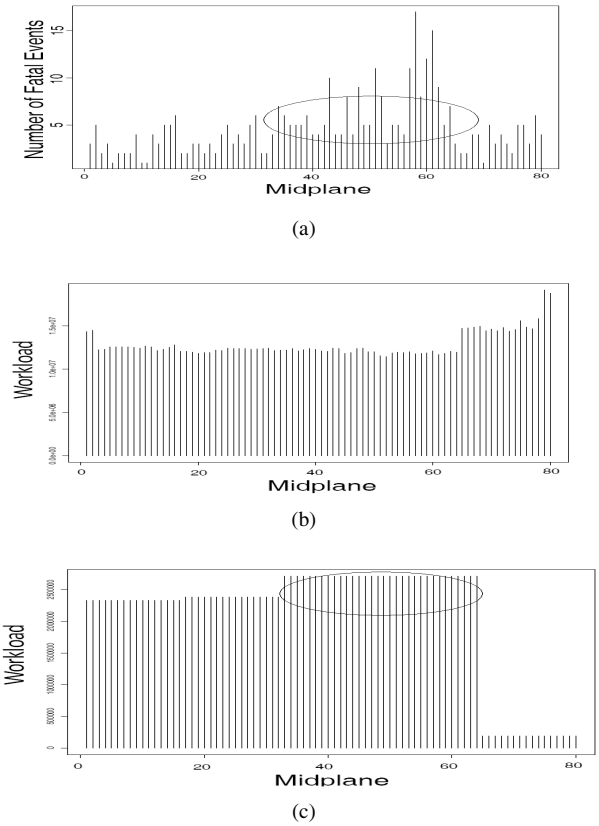

Figure 4: (a) Number of fatal events on each midplane, (b) workload on each midplane, and (c) workload of wider jobs, (i.e., the job requesting no less than 32 midplanes) on each midplane. In (a) and (c), we highlight midplane 33–64 to present the impact of wider jobs on failure rate.

Our co-analysis indicates that the main reason is the inconsistent scheduling policies for different midplanes. In midplanes 1 and 2, there are a number of short time and small-size jobs. The scheduler prefers to assign small jobs to midplanes in the range of 65 to 80, reserving the other 64 midplanes for larger jobs. In other words, although the accumulated workloads in these midplanes are fairly high, most of them are from small jobs (i.e., narrow jobs). As shown in Figure 4c, if only the jobs with 64 midplanes or larger are considered, the midplanes in the range of 33 to 64 have the highest utilization. This result is consistent with Figure 4b, which shows that the same set of midplanes also has a significantly higher number of fatal events than do the others.

This phenomenon indicates that a workload of wider jobs (i.e., jobs requesting a large number of nodes) has a significant impact on the failure rate. One possible explanation is that the wider jobs involve more complicated system configurations and interactions between multiple components, which reduces the reliability of the system. In Section VI-D,

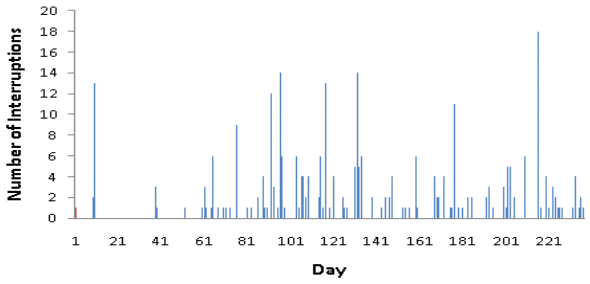

Figure 5: Number of interruptions per day.

we show that these wider jobs are more vulnerable than small jobs. Both studies show that wider jobs are the key factors affecting reliability.

Observation 5: High aggregated workload does not necessarily mean high failure rate. Instead, wider jobs (i.e., jobs requesting a large number of nodes) can significantly impact failure rate.

VI. Job Interruption Characteristics

We focus in this section on four job interruption features: burst, interruption rate, failure propagation, and job vulnerability.

A. Burst Feature

In our case study, only a small portion of jobs is interrupted by fatal events (i.e., 0.45% of total jobs or 1.73% of distinct jobs). Hence the fatal events may not have significant impact on systemwide performance metrics, such as system utilization rate and bounded slowdown [14].

However, Figure 5 presents the number of interruptions per day systemwide, which shows that the job interruptions are rare but occur in a burst manner. Specifically, we find that a job becomes error-prone after its first interruption. For instance, 33 jobs are interrupted again soon after the first interruption within 1,000 seconds. One distinct job has been interrupted four times within 2,321 seconds. Further, one system failure can also interrupt multiple jobs continuously. For instance, one system failure caused by L1 data cache parity error consecutively interrupted 28 jobs in 92 hours.

Observation 6: While job interruptions in Blue Gene/P are rare events, they occur in a burst manner, and many job interruptions occur soon after the previous ones.

B. Job Interruption Rate

In our case study, 206 out of 308 job interruptions are caused by system failures, and 102 interruptions are caused by application errors. Figure 6 presents the distribution fitting results of interruption interarrival times. Weibull distribution still gives a better fit than does exponential distribution for both cases, as is verified by a likelihood

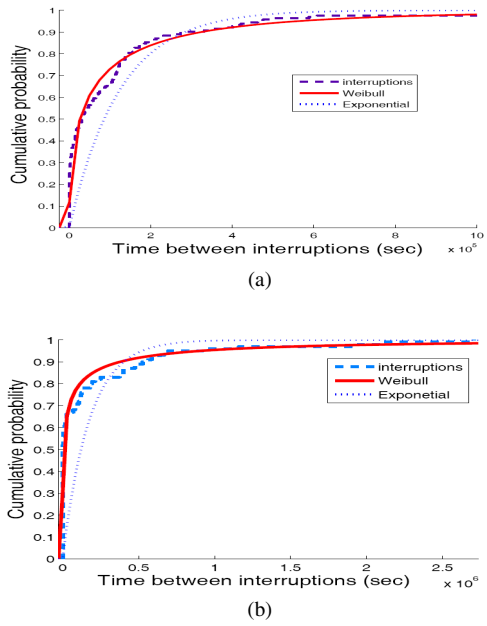

Figure 6: Expirical CDF for interarrival times of interruptions (a) due to system failures and (b) due to application errors.

Table V: Comparison of the parameters and numerical characteristics of Weibull distributions for job interruptions due to system failures or application errors.

Interruption Cause	Shape	Scale	Mean	Variance
System Failures	0.346296	23075.3	120454	2.38219e+011
Application Errors	0.301397	23801.7	215886	1.33603e+012

ratio test. Table V presents the parameters and numerical characteristics of the Weibull distributions.

As shown in Table V, the MTTI (mean time to interruptions) caused by application errors is about two times higher than the one caused by system failures. By comparing the shape values, we find that the hazard rate of interruptions caused by system failures is less than that of interruptions caused by application errors. The main reason is that application errors generally need more time for fixing, whereas some system failures can be easily solved by rebooting.

We further compare the MTTI caused by system failures in Table V with the MTBF of system failures in Table IV. We find that the interruption rate is significant less than the failure rate. The main reason is that 45.45% of fatal events actually do not interrupt any job because no job runs in the same location. As a result, the MTTI is 4.07 times larger than the MTBF. This result suggests that the failure prediction methodology needs to provide location information about future failures in large-scale systems. With the location information in hand, we can identify whether any productive jobs are running in the same location, and thus we can avoid the overhead of proactive actions.

Observation 7: Job interruption rate is significantly lower than system failure rate. The reason is that the failed nodes do not impact user jobs if at the time of failure the nodes are idle.

C. Failure Propagation

Traditionally, the research on failure propagation mainly studies the spreading of failure from one node to others [22]. In Blue Gene systems, a failure is generally reported from multiple nodes. This does not mean that the failure is propagating across multiple nodes. It is simply because jobs are typically parallel jobs and any interrupt to a parallel job is reported from all the allocated nodes. Therefore, in this study we analyze failure propagation from different aspects.

We classify two types of failure propagation: temporal and spatial. As shown in Section IV-C, temporal propagation occurs when the scheduler keeps allocating failed nodes for incoming jobs or when users keep resubmitting buggy codes. In this subsection, we consider the spatial propagation; that is, a single failure propagates from one running job to other running jobs in different locations.

We find that only 7.22% of fatal events propagate to multiple jobs. These fatal events belong to two types: *bg_code_script_error* and *CiodHungProxy*. Here bg_code_script_error is the script errors in the file system, while CiodHungProxy is errors in the file system and I/O nodes. Because the file system can be shared by multiple jobs, a single failure on file system can interrupt these jobs simultaneously.

Observation 8: Temporal failure propagation occurs when the scheduler keeps allocating failed nodes to incoming jobs or when users keep submitting their buggy codes, which is common in our case study. Spatial failure propagation occurs when a failure spreads across multiple jobs at the same time period. In our case study, temporal propagation occurs only in 7.22% of fatal events.

D. Job Vulnerability

In this subsection, we identify what types of jobs have a high probability of being interrupted. We first use historical statistics to study the impacts of job interruptions on their subsequent resubmissions. Then we identify key factors regarding job vulnerability. Following Subsection VI-B, we put job interruptions into two categories and study them separately:

- *Category-1:* Job interruptions caused by system failures.
- *Category-2:* Job interruptions caused by application errors.

1. Historical Statistics: The probability of job interruption for resubmission is calculated by using the historical record for that job. In particular, we calculate the probability of job interruption if the job has been interrupted k times consecutively in previous submissions. In our case study, we have $k = 1, 2, 3$.

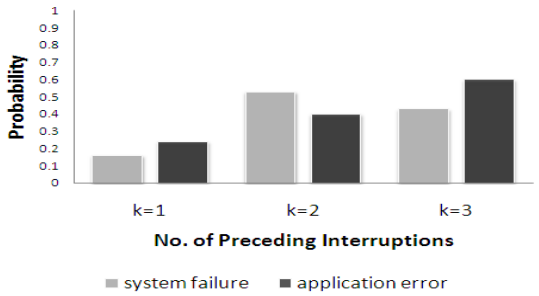

Figure 7: Statistical results of interruptions for resubmitted jobs. Here k is the times of previously consecutive interruptions.

Figure 7 shows the statistical results of interruptions for resubmitted jobs, As shown in the figure, the influence of historical interruptions on subsequent submissions is not trivial. In the case of category 1 jobs, the probability of interruption for resubmissions can reach 53% when jobs have been interrupted twice before; and in the case of category 2, the probability can be as high as 60% when these jobs have failed three times before.

Although both categories show high interruption probabilities for resubmitted jobs, they have different statistical patterns. The category 1 jobs achieve the highest probability when $k = 2$, while the category 2 jobs have monotonically increased probabilities with the growth of k. We believe this difference is caused by different recovery mechanisms for these two categories. In the case of category 1, the mechanism of "reboot before execution" leads to a relatively low probability for $k = 1$. But for $k = 2$ the probability is much higher, for two reasons. First, $k = 2$ indicates that simple rebooting cannot fix the system failure. Second, the job scheduler tends to allocate resubmitted jobs to the same partition in Intrepid; that is, 57.44% of resubmitted jobs were put on the same location as the last time. With the increasing k, however, the recovery process performed by system administrators decreases the probability gradually. Hence we get a lower value for $k = 3$. For category 2, an increasing value of k usually indicates increasing difficulty in fixing the application errors. As a result, the probability of resubmission of buggy codes increases monotonically with k.

Observation 9: Job vulnerability is, to some extent, related to the interruption history of the job. This relationship is particularly obvious if the job interruption is caused by application errors.

2. Key Feature Selection: Although historical records provide a possible way to evaluate vulnerability, not all job interruptions have sufficient historical records. For example,

83.77% of interruptions cannot be covered if $k = 2$ is adopted. In this study, therefore, we explore five key features of job vulnerability: *user*, *project*, *execution time*, *size*, and *location*. User and project represent two factors from the application level; we are interested in whether certain suspicious users or projects are dominant factors for job interruptions introduced by application errors. Execution time and size are two major factors from the system level; a popular belief is that the product of these two features decide the vulnerability to system failures, that is, that wider/longer jobs are more vulnerable to system failures than are narrow/shorter jobs. Our goal is to find which one has greater impact. Location indicates the impact of scheduling policy on job vulnerability; we wish to know whether the scheduler's decision of assigning unreliable partitions can affect job vulnerability significantly. Again, the two interruption categories are analyzed separately.

In our case study, we collect the five features as follows.

- According to the features of user and project, we put the records into two categories: suspicious users/projects and normal users/projects. We extract 16 out of 236 users as suspicious users since they occupy 53.25% of the total job interruptions; and we identify 19 projects out of 91 projects as suspicious projects, which occupy more than 74% of the interruptions.
- We collect the features of size and execution time directly from the job log. Because of system constraints, the job size can only be 1, 2, 4, 8, 16, 32, 48, 64, or 80 midplanes. The scale of execution time ranges from 10 seconds to 113.5 hours.
- We select the top 12 midplanes with the highest number of failures as unreliable midplanes.

To extract features that play dominant roles in determining job vulnerability, we adopt a method called information gain ratio evaluation [26], which has been widely used for feature ranking.

We make several interesting observations from the results. For category 1 interruptions, job vulnerability is mainly caused by job size and job location. This conclusion is consistent with the popular assumption that wider jobs are more vulnerable to failures than are narrow ones. Moreover, our study also indicate that job length is not an important fact with respective to job vulnerability.

To gain a better understanding of this phenomenon, we divide job length into four groups: 10-400 seconds, 400-1600 seconds, 1600-6400 seconds, and >6400 seconds. We count the number of interruptions caused by system failures and the number of successful job completions in each group, and we show the results in Table VI. As shown in the last column, the interruption rate increases almost linearly with the growth of job size. (The only outlier is the 48 midplanes case since only 4 jobs are executed in this scale.) In contrast, for each row (i.e., jobs with the same sizes), longer job length does not necessarily mean higher interruption rate. For example, the interruption rate for the 400-1600 group is even higher than the 1600-6400 group and the ≥6400 group. In the other hand, with the same product of job size and execution time, double job size has higher impact than double job execution time. For instance, comparing interruption proportion in 8 midplanes and 400-1600 seconds with the one in 4 midplanes and ≥6400 seconds (the width is increased only 2 times whereas the length is increased at least 4 times), the first one is 16 times larger than the second one.

This observation contradicts the common belief that with the same job size longer jobs are more vulnerable to failures than are shorter jobs. This common belief is based on the traditional exponential distribution model. In the exponential model, the probability of job interruption is the same as failure probability $P(t < T) = 1 - e^{-\lambda T}$, where λ is related to job width and T is determined by the job length [18], [19]. However, case study shows that failure interarrival rate fits the Weibull distribution $P(t < T) = 1 - e^{(-\lambda T)^{\alpha}}$, where the probability of job interruption is decided not only by λ and T but also by the time between the recent failure as a conditional probability [30]. In our study, the shape $\alpha < 1$ indicates decreasing hazard rate, that is, the failure rate significantly decreases with the growth of time since the previous failure. As a result, the short jobs submitted close to the previous interruption are more prone to interruption than the long jobs submitted far away from the previous interruption.

This situation is also consistent with observation 6. Because the scheduler can assign the same failed nodes to incoming jobs, many jobs are interrupted soon after the previous interruption. As a result, it significantly increases the interruption proportion of jobs with short execution time.

Another explanation of this observation is the "reboot before execution" mechanism in Blue Gene/P. It minimizes the amount of accumulated problematic state [15]. In addition, in the job log, there is few extremely long jobs (e.g., with job length of several months). Execution time could be a significant factor in deciding the vulnerability of extremely long jobs, and we plan to study this in our future work.

For category 2, short execution time becomes the dominant feature. In particular, 74.5% of the job interruptions occur in less than 3,600 seconds. In other words, most application errors tend to be reported in the first hour. This feature provides an institutional guide for checkpointing policy for long-term jobs: it is not advisable to introduce checkpointing in early in the execution period because of the high probability of application errors. Although the job size is not the dominant feature in this case, it still provides meaningful information by combining with execution time. We find that no interruption caused by applications exists if the job size is larger than 32 midplanes and the execution time is longer than $1,000$ seconds. In other words, long jobs with large size appear not to be vulnerable to application

errors. This phenomenon is different from the case of wider jobs in category 1, where the ratio of failures is roughly proportional to the job size. We believe the explanation lies in the fact that users request large size and long execution time only for their well-debugged jobs.

In addition, we notice that for both interruption categories suspicious users and suspicious projects always have the lowest impact on the job vulnerability. The reason is information gain ratio is based on the portion, not the absolute number of failures. In our study, even for the most suspicious users or suspicious projects, the portion of failed jobs is small (less than 1%); thus, the results are not significantly biased to any specific set of users or projects.

Observation 10: In our case study, for job interruptions caused by system failures, job size has more significant impact on job vulnerability than does job execution time.

Observation 11: Most interruptions caused by application errors are reported during the early execution period (i.e., within one hour).

Observation 12: User and project are important factors with regards to the absolute number of failures. But even for the most suspicious users or suspicious projects, the portion of failed jobs is still small.

VII. Discussion

In this section, we briefly present some recommendations for system management.

Before the occurrence of failure, failure prediction is helpful to prevent the work loss. According to Observations #1 and #7, some RAS events have high severity but do not interrupt the jobs, and a number of system failures occur in the components without any productive jobs running atop. As a result, it would be better for the failure predictor to identify the interruption-related fatal events and provide location information in the prediction result, which can avoid unnecessary actions responding to false alarms or the idle status of nodes.

Observations from co-analysis provide useful information for precautionary actions, for example, checkpointing. As shown in Observations #9 and #11, the probability of job interruption is high if there exist historical records of application-related interruptions. Moreover, most application errors tend to be reported in the first hour. Therefore it is inadvisable to introduce checkpointing early in the execution period if the job has historical records of application-related interruptions. Since job size is the dominant feature for deciding the probability of system-related interruption in Blue Gene/P (Observation #10), precautionary actions are critical for large size jobs.

After the occurrence of failure, fast diagnosis and recovery are of paramount importance because the simple "reboot before execution" mechanism cannot solve a large proportion of fatal events. For rapid diagnosis, co-analysis of the RAS and job logs is helpful to distinguish between system failures and application errors. Furthermore, we identify the critical information for job scheduler to reduce the impact of failures. According to Observations #6, #8, and #9, the scheduler needs more failure-related information of two types: (1) fatal events information including event time, location, category, and recovery status and (2) the historical records of the job interruptions. In our future work, we will implement the function to subscribe failure-related information in CiFTS software [3], [6].

VIII. Summary

In this paper, we have presented a co-analysis of RAS and job logs for the Blue Gene/P system. Through co-analysis, we successfully overcome some shortcomings of RAS log analysis and draw 12 observations that are helpful in better understanding failure patterns and system/user behavior. To the best of our knowledge, we are the first to explore co-analysis of RAS logs and job logs in high-performance computing systems.

In our future work, we will apply our co-analysis methodology in a variety of high-performance computing systems, including the Cray XT5 at Oak Ridge. We believe co-analysis can provide more information about reliability issues in these systems.

Acknowledgment

The work at Illinois Institute of Technology is supported in part by US National Science Foundation grants CNS-0834514, CNS-0720549, and CCF-0702737. This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357. This research also used resources of the Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

References

[1] Blue Gene Team, "Overview of the IBM Blue Gene/P project," *IBM Journal of Research and Development*, 2008.

[2] Top500 supercomputing sites http://top500.org/.

[3] R. Gupta, P. Beckman, B.-H. Park, E. Lusk, and P. Hargrove. CiFTS: A coordinated infrastructure for fault-tolerant systems. In *Proc. of ICPP*, 2009.

[4] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo. An adaptive semantic filter for Blue Gene/L failure log analysis systems. *Workshop on SMTPS*, 2007.

[5] A. Oliner, R. Sahoo, J. Moreira, M. Gupta, and A. Sivasubramaniam. Fault-aware job scheduling for Blue Gene/L systems. In *Proc. of IPDPS*, 2004.

Table VI: Number of system-related interruptions and total jobs in different size and execution time. There are two numbers in each cell: the first is number of interrupted jobs, and the second is the number of total jobs without application errors.

	10-400 sec	400-1600 sec	1600-6400 sec	$\geq$ 6400 sec	sum:proportion
1 midplane	24/12282	19/7300	7/17339	7/9492	57/46413=0.12%
2 midplanes	8/1146	7/2601	4/6052	3/2112	22/11911=0.18%
4 midplanes	13/881	9/901	1/1026	4/2014	27/4822=0.56%
8 midplanes	4/611	9/563	0/636	8/748	21/2618=0.80%
16 midplanes	9/288	13/685	3/466	6/415	31/1854=1.67%
32 midplanes	7/20	8/362	0/195	1/79	16/656=2.44%
48 midplanes	0/3	0/1	0/0	0/0	0/4 =0%
64 midplanes	4/12	13/147	0/143	1/39	18/341=5.28%
80 midplanes	4/11	10/33	0/27	0/2	14/73=19.18%
sum:proportion	73/15251 =0.48%	88/12595=0.70%	15/25945=0.06%	30/14901= 0.20%	206/68692=0.30%

[6] Z. Zheng, R. Gupta, Z. Lan, and S. Coghlan. FTB-enabled failure prediction for Blue Gene/P systems. In *Proc. of SuperComputing (research poster)*, 2009.

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

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

[9] A. Oliner and J. Stearly. What supercomputers say: A study of five system logs. In *Proc. of DSN*, 2007.

[10] T. Hacker, F. Romero, and C. Carothers. An analysis of clustered failures on large supercomputing systems. *Journal of Parallel and Distributed Computing*, 69:652–665, 2009.

[11] N. Taerat, N. Naksinehaboon, C. Chandler, J. Elliott, C. Leangsuksun, G. Ostrouchov, S. Scott, and C. Engelmann. Blue Gene/L log analysis and time to interrupt estimation. In *Proc. of ARES*, 2009.

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

[13] J. Young. A first order approximation to the optimal checkpoint interval. *Comm. ACM*, 17(9): 530–531, 1974.

[14] W. Tang, Z. Lan, N. Desai, and D. Buettner. Fault-aware utility-based job scheduling on Blue Gene/P systems. In *Proc. of Cluster*, 2009.

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

[16] M. Crowder, A. Kimber, T. Sweeting, and R. Smith. Statistical analysis of reliability data. London: Chapman and Hall, 1991.

[17] R. Sahoo, A. Sivasubramanium, M. Squillante, and Y. Zhang. Failure data analysis of a large-scale heterogeneous server environment. In *Proc. of DSN*, 2004.

[18] F. Petrini, K. Davis, and J. Sancho. System-level fault tolerance in largescale parallel machines with buffered coscheduling. In *Proc. of IPDPS*, 2004.

[19] Z. Ziming and Z. Lan. Reliability-aware scalability models for high performance computing. In *Proc. of Cluster*, 2009.

[20] 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. *Whitepaper*, 2009.

[21] W. Jones, J. Daly, and N. DeBardeleben. Impact of sub-optimal checkpoint intervals on application efficiency in computational clusters. In *Proc. of HPDC*, 2010.

[22] J. Xu, Z. Kallbarczyk, and R. Iyer. Networked Windows NT system field failure data analysis. Technical Report CRHC 9808, UIUC, 1999.

[23] S. Fu and C. Xu Exploring event correlation for failure prediction in coalitions of clusters. In *Proc. of Supercomputing*, 2007.

[24] W. Jiang, C. Hu, S. Pasupathy, A. Kanevsky, Z. Li, and Y. Zhou. Understanding customer problem troubleshooting from storage system logs. In *Proc. of FAST*, 2009.

[25] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan. Detecting large-scale system problems by mining console logs. In *SOSP*, 2009.

[26] H. Liu and L. Yu. Toward Integrating Feature Selection Algorithms for Classification and Clustering. *IEEE Trans. on Knowledge and Data Engineering*, 17(4):491-502, 2005.

[27] B. Schroeder and G. Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *Proc. of FAST*, 2007.

[28] Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload/.

[29] USENIX Computer Failure Data Repository. http://cfdr.usenix.org/.

[30] N. Gottumukkala, R. Nassar, M. Paun, and C. Leangsuksun. Reliability of a System of k Nodes for High Performance Computing Applications. *IEEE Trans. on Reliability*, 59(1):142-169, 2010.