BlueGene/L Failure Analysis and Prediction Models

Yinglung Liang Yanyong Zhang
Electrical & Computer Engineering Dept.
Rutgers University
{*ylliang,yyzhang*}*@ece.rutgers.edu*

Morris Jette
Computational Resource Management Group
Lawrence Livermore National Laboratory
jette@llnl.gov

Anand Sivasubramaniam
Computer Science & Engineering Dept.
Penn State University
anand@cse.psu.edu

Ramendra Sahoo
Exploratory Systems Software Department
IBM T. J. Watson Research Center
rsahoo@us.ibm.com

Abstract

The growing computational and storage needs of several scientific applications mandate the deployment of extreme-scale parallel machines, such as IBM's BlueGene/L which can accommodate as many as 128K processors. One of the challenges when designing and deploying these systems in a production setting is the need to take failure occurrences, whether it be in the hardware or in the software, into account. Earlier work has shown that conventional runtime fault-tolerant techniques such as periodic checkpointing are not effective to the emerging systems. Instead, the ability to predict failure occurrences can help develop more effective checkpointing strategies. Failure prediction has long been regarded as a challenging research problem, mainly due to the lack of realistic failure data from actual production systems. In this study, we have collected RAS event logs from BlueGene/L over a period of more than 100 days. We have investigated the characteristics of fatal failure events, as well as the correlation between fatal events and non-fatal events. Based on the observations, we have developed three simple yet effective failure prediction methods, which can predict around 80% of the memory and network failures, and 47% of the application I/O failures.

1 Introduction

Meta-scale scientific and engineering applications have been and continue to be playing a critical role in every aspect of the society, including economies of enterprises and even countries, health and human development, military/security, and even overall quality of life. The large processing and storage demands of these applications call for supercomputers that scale much beyond what have been built so far. IBM BlueGene/L, consisting of 128K processors [1], is such a system deployed at Lawrence Livermore National Laboratory (LLNL). Upon its deployment in August, 2005, it became the fastest supercomputer on the Top500 Supercomputers list [5]. Since then, it has been hosting applications that span several thousand processors, in the domains including hydrodynamics, quantum chemistry, molecular dynamics and climate modeling.

As applications and the underlying platforms scale to this level, failure occurrence as well as its impacts on system performance and operation costs, are becoming a critically important concern to the research community. Specifically, failures are becoming a norm, rather than an exception. Firstly, transient hardware failures are increasing, not just in memory structures and communication paths, but also in combinational circuits [17, 19]. Higher chip integration densities and lower voltages (to reduce power consumption), are making circuits more susceptible to bit flips [31]. Secondly, permanent hardware device failures are also a growing concern, especially with high power consumption for these large scale systems leading to immense heat dissipation, which in consequence can accelerate the failure rates for different devices [23, 21], including the CPUs, memory systems, and disk drives. External events, such as cooling system failures, also play a crucial role in the reliability of these systems that need to provide continuous operation over extended periods of time. Thirdly, in addition to the hardware issues, the sophisticated software that is taking on more duties, can (i) contain bugs, (ii) be difficult to comprehend and analyze (and thus may not be used in the right way), and (iii) age in quality over time [9, 27], which can all again lead to application crashes and/or system downtime.

Failures can make nodes unavailable, thereby lowering system utilization. Further, failures can cause applications executing on the nodes (probably having run for a long time) to abort, thus wasting the effort already expended. A long running (over several days) application that spans a large (hundreds/thousands) number of nodes, may find it very difficult to make any forward progress because of the failures. Additionally, some failures, especially those in network subsystem, may affect multiple applications that happen to run side by side. Eventually, the lower utilization and availability impacts the response times and system throughput of all the jobs, thus putting at risk the main motivation behind deploying these large scale systems. Our earlier studies [32, 18] show 100% worsening of job performance, with a 1 failure per day assumption (and we assume these failures affect one job at a time). As a real example, LLNL has witnessed frequent L1 cache failures for long running jobs, and in order to finish these jobs, L1 cache has been disabled for jobs longer than 4 hours, which results in much prolonged execution times for these jobs.

In addition to lowering system performance and availability, failures can also greatly increase the system management costs. The system administrator may need to detect failure occurrence, diagnose the problem, and figure out the best sequence of remedial actions. On the hardware end, this may entail resetting a node, changing the motherboard/disk, etc., and on the software end it may require migrating the application, restarting the application, reinitializing/rejuvenating [27] a software module, etc. In addition to the time incurred by such operations during which the system (or at least the affected nodes) may be unavailable, personnel time needs to be allotted for this purpose. The resulting personnel involvement will increase the Total Cost of Operation (TCO), which is becoming a serious concern in numerous production environments [12, 4].

It has been recognized that preventing failures from occurring is very challenging, if at all possible [4]. Instead, we take the viewpoint that runtime fault-tolerant measures that can mitigate the adverse impacts of failures are in an urgent need. Checkpointing [32] is such a technique that can allow the failed jobs to start from a saved point, rather than restarting from the beginning. Though checkpointing techniques have been widely used in conventional systems, they are not as effective in large-scale parallel systems such as BlueGene/L because the overheads of checkpointing overshadow the gain: checkpointing a job that involves tens of thousand tasks may take at least half an hour. In our earlier study [32], we find that checkpointing at regular intervals simply does not improve the performance; a large interval may miss many failures, while a small interval may incur high checkpointing overheads. Instead, we find that the capability of predicting the time/location of the next failure, though not perfect, can considerably boost the benefits of runtime techniques such as job checkpointing or scheduling[32, 18].

Failure prediction, however, has long been considered a challenging research problem. One of the main reasons is the lack of suitable data from realistic systems. To address this void, we have obtained event logs containing all the RAS (reliability, availability, and serviceability) data since 08/02/05, from BlueGene/L. After carefully preprocessing these data using a three-step filtering tool [14], we extract all the failure events that can lead to job terminations, and categorize them into memory failures, network failures, and application I/O failures. After filtering out important events, we conduct in-depth studies to explore the predictability of these failure events. Firstly, we find that 50% of the network failures and 35% of the application I/O failures occur within a window of half an hour after the preceding failures. Secondly, we find that network failures exhibit strong spatial skewness, with 6% of the nodes encountering 61% of the failures. Thirdly, we find that jobs that report non-fatal events are very likely followed by a fatal failure event. These observations are evidence that we can effectively predict failures, which can in turn be used to develop efficient runtime fault-tolerant strategies. In this paper, we have developed three prediction algorithms based on the bursty nature of failure occurrence, spatial skewness, and preceding non-fatal events. We have evaluated the effectiveness of these algorithms carefully, and found they are able to capture a large fraction of failures.

The rest of this paper is organized as follows. Section 2 describes the logs used in the study. In section 3, we discuss the temporal and spatial characteristics of failure events, and develop two prediction strategies based upon these characteristics. Following the failure characteristics, in Section 4, we examine the relationship between fatal events and non-fatal events, and develop another prediction scheme. The related work and concluding remarks are shown in Sections 5 and 6 respectively.

2 Overview of RAS Event Logs Logs

In this study, the RAS event logs are collected from BlueGene/L at Lawrence Livermore National Laboratory (LLNL), which currently stands at number 1 in the top500 supercomputer list [5]. More specifically, the RAS event logs are obtained from IBM, the supercomputer vendor, through their RAS monitoring system.

BlueGene/L has 128K PowerPC 440 700MHz processors, which are organized into 64 racks. Each rack consists of 2 midplanes, and a midplane (with 1024 processors) is the granularity of job allocation. A midplane contains 16 node cards (which houses the compute chips), 4 I/O cards (which houses the I/O chips), and 24 midplane switches (through which different midplanes connect). RAS events are logged through the Machine Monitoring and Control System (CMCS), and finally stored in a DB2 database engine. The logging granularity is less than 1 millisecond. More detailed descriptions of the BlueGene/L hardware and the logging mechanism can be found in our earlier paper [14].

2.1 Raw RAS Event Logs

We have been collecting RAS event logs from BlueGene/L since August 2, 2005. Up to the date of November 18, 2005, we have totally 1,318,137 entries. These entries are records of all the RAS related events that occur within various components of the machine. Information about scheduled maintenances, reboots, and repairs is not included. Each record of the logs has a number of attributes. The relevant attributes are described as follows:

- *RECID* is the sequence number of an error entry, which is incremented upon each new entry being appended to the logs.

- *EVENT_TYPE* specifies the mechanism through which the event is recorded, with most of them being through RAS [8].

- *FACILITY* denotes the component where the event is flagged, which is one of the following: LINKCARD, APP, KERNEL, HARDWARE, DISCOVERY, CMCS, BGLMASTER, SERV_NET or MONITOR. Events with LINKCARD facility report problems with midplane switches, which is related to communication between midplanes. APP events are those flagged in the application domain of the compute chips. Most of these are reported by the I/O demon regarding invalid path names, wrong access permissions, severed links, etc. Events with KERNEL facility are those reported by the OS kernel domain of the compute chips, which are usually in the memory or network subsystem. These could include memory parity/ECC errors in the hardware, bus errors due to wrong addresses being generated by the software, torus errors due to links failing, etc. Events with HARDWARE facility are usually related to the hardware operations of the system (e.g. "node card power module is not accessible", "node card is not fully functional", etc). Events with DISCOVERY facility are usually related to resource discovery and initial configurations within the machine (e.g. "cannot get assembly information for a node card", "fan module is missing", etc), with most of these being at the INFO or WARNING severity levels. CMCS, BGLMASTER SERV_NET facility errors are again mostly at the INFO level, which report events in the operation of the CMCS, BGLMASTER and the service network. Finally, events with MONITOR facility are usually related to the power/temperature/wiring issues of link-card/node-card/service-card. Nearly all MONITOR events are in the FAILURE severity levels.

- *SEVERITY* can be one of the following levels - INFO, WARNING, SEVERE, ERROR, FATAL, or FAILURE - which also denotes the increasing order of severity. INFO events are more informative in nature on overall system reliability, such as "a torus problem has been detected and corrected", "the card status has changed", "the kernel is generating the core", etc. WARNING events are usually associated with node-card/link-card/service-card not being functional. SEVERE events give more details on why these cards may not be functional (e.g."link-card is not accessible", "problem while initializing link/node/service card", "error getting assembly information from the node card", etc.). ERROR events report problems that are more persistent and further pin-point their causes ("Fan module serial number appears to be invalid", "cable x is present but the corresponding reverse cable is missing", "Bad cables going into the linkcard", etc.). All of these above events are either informative in nature, or are related more to initial configuration errors, and are thus relatively transparent to the applications/runtime environment. However, FATAL or FAILURE events (such as "uncorrectable torus error", "memory error", etc.) are more severe, and usually lead to application/software crashes. Our primary focus in this study is consequently on FATAL and FAILURE events.

- *EVENT_TIME* is the time stamp associated with that event.

- *JOB_ID* denotes the job that detects this event. This field is only valid for events reported by compute/IO chips.

- *LOCATION* of an event (i.e., which chip/node-card/service-card/link-card experiences the error), can be specified in two ways. It can either be specified as (i) a combination of job ID, processor, node, and block, or (ii) through a separate location field. We mainly use the latter approach (location attribute) to determine where an error takes place.

- *ENTRY_DATA* gives a short description of the event.

2.2 RAS Data Preprocessing

The raw logs contain an enormous amount of entries, many of which are repeated or redundant. Before we can use these logs to study the failure behavior of BlueGene/L, we must first filter the failure data and isolate unique failures. Filtering failure data has traditionally been a challenging task [14]. Further complicating the process is the fact that the BlueGene/L logging mechanism operates at much finer granularity in both temporal (e.g. the logging interval is less than 1 millisecond) and spatial (hundreds of thousand of processors) domains than earlier machines. In addition, the nature of parallel applications calls for unique filtering techniques that are not needed for sequential applications. To address these challenges, we suitably modify the filtering tool we developed in our earlier work [14], which involves the following three steps:

1. *Extracting and Categorizing Failure Events.* In this step, we extract all the events whose severity levels are either FATAL or FAILURE, referred to as *failures*, because these events will lead to application crashes, and thus significantly degrade the performance. Further, we classify failures into the following categories according to the subsystem in which they occurs: (i) memory failures, (ii) network failures, (iii) application I/O failures, (iv) midplane switch failures, and (v) node card failures. Failures are classified into these five types based on the ENTRY_DATA field.

2. *Temporal Compression at a Single Location.* Failure events from the same location often occur in bursts, and we call such bursts as *clusters.* Some clusters are homogeneous, with their failures having identical values in the ENTRY_DATA field; others are heterogeneous and their failures usually report different attributes of the same event. For example, a memory failure cluster is heterogeneous because each entry reports a unique system state upon the occurrence of a memory failure [14]. Therefore, in the second step, we need to coalesce a cluster into a single failure record. Identifying such clusters from the log, requires sorting/grouping all the failures according to the associated subsystem (i.e. memory, network, or application/IO), the location, and the job ID, and using a suitable threshold T_{th}. Hence, failures that occur within the same subsystem and are reported by the same location and the same job, belong to a cluster if the gaps between them are less than T_{th}. Table 1 (a) presents the number of remaining failure records after filtering with different T_{th} values. In this exercise, we set the threshold value to 5 minutes, as also suggested by previous studies [14, 7, 13].

3. *Spatial Compression across Multiple Locations.* A failure can be detected/reported by multiple locations, especially because BlueGene/L hosts parallel jobs. For example, all the tasks from a job will experience the same I/O failure if they access the same directory. For another example, a network failure is very likely detected by multiple locations. As a result, it is essential to filter across locations, which we call *spatial filtering.* Spatial filtering removes failures that are close to each other (gaps between them below the threshold S_{th}), with the same entry data, from the same job, but from different locations. Similarly, Table 1 (b) presents the number of remaining failure records after spatial filtering with different S_{th} values. Choosing the appropriate value for S_{th} is rather straightforward since the resulting failure count is not very sensitive to the threshold value. Then we simply choose 5 minutes.

After applying the three-step filtering algorithm, we can identify unique failures within the boundary of a job. In this paper, we do not attempt to coalesce failures that are experienced by different jobs because relevant information is missing for this purpose. Among memory, network, and app-IO failures, we find that temporal filtering is effective for memory failures, while spatial filtering is more effective for network and app-IO failures. This is because tasks from a parallel job are more likely to detect same I/O failures or network failures.

3 Failure Prediction Based on Failure Characteristics

Before we report our observations, we would like to emphasize that each failure event in our study does not necessarily correspond to a *unique* physical failure in the system hardware or software. Instead, several failure events, especially those with the same entry data and temporally close to each other, may just be that the same failure is encountered by subsequent jobs. Therefore, the observations do not only reflect the system's failure behaviors, but also the interplay between the failure behaviors and the usage patterns (e.g. jobs' arrive/execution times). This vagueness is due to the lack of exact duration information for each failure. However, we emphasize that it is extremely difficult to pinpoint the real root of each failure event, let alone its actual duration, so in this paper we do not isolate the impact of job execution on the failure pattern.

3.1 Temporal Characteristics

Figures 1 (a)-(h) depict the two aspects of temporal characteristics of failure events: time series of failure occurrence, i.e. number of failures observed every day during the log duration, and the probability density function (PDF) of the TBF distribution. Figure 1 (b) shows that memory failures almost occur every day, and further, that the number of failures per day does not vary considerably. Figure 1 (f) shows that memory failures, though not bursty, do not exhibit periodic

Log size with T_{th} =	Memory	Network	APP-IO	Midplane Switch	Node Cards
0	8,206	10,554	178,292	166	96
30 sec	267	9,418	178,015	83	6
1 min	251	9,418	173,491	52	6
5 min	246	9,415	102,442	30	4
30 min	241	9,219	89,333	22	4
1 hour	237	8,705	81,834	17	4

(a) Number of failure events after temporal filtering using different T_{th}

Log size with S_{th} =	Memory	Network	APP-IO	Midplane Switch	Node Cards
0	246	9,415	101,196	30	4
30 sec	217	139	331	30	4
1 min	217	139	318	30	4
5 min	215	139	299	30	4
30 min	208	114	237	22	4
1 hour	199	105	225	17	4

(b) Number of failure events after spatial filtering using different S_{th}

Table 1. Filtering thresholds

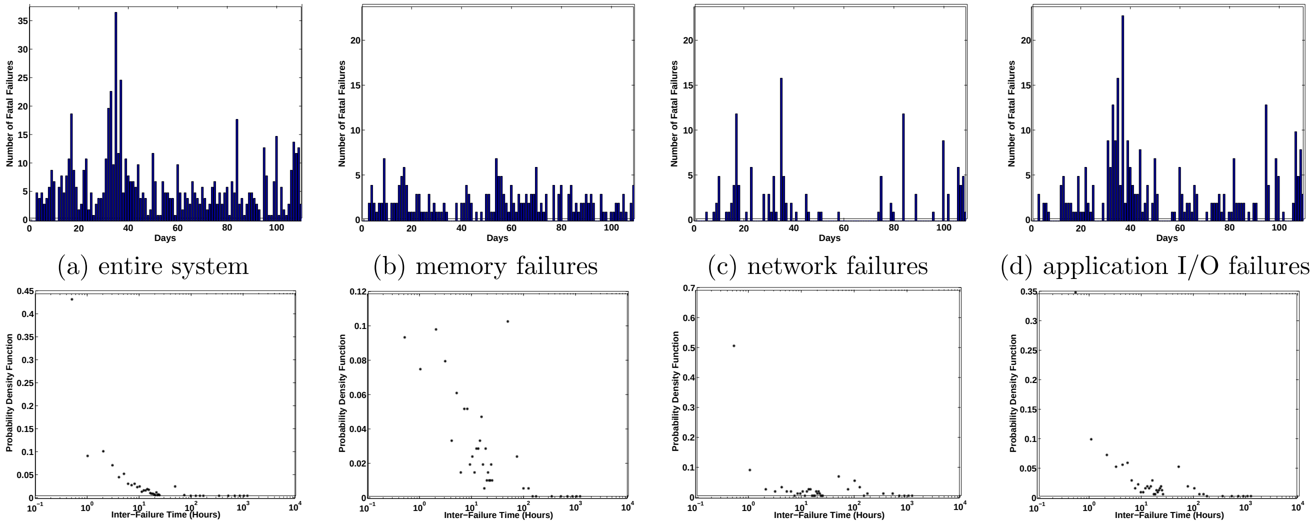
occurrence as well; not a single TBF value dominates, but many TBF values are possible and have comparable likelihoods. Unlike memory failures, both network and application I/O failures occur in bursts. As a result, small TBF values are more popular than larger ones. For example, 50% of the network failures occur within half an hour after the previous failures, and 35% of the application I/O failures occur within half an hour after the previous failures. In addition, another 10% the network and application I/O failures occur within a window of between half an hour and an hour after the preceding failures. A possible reason is that it is harder to pinpoint network as well as application I/O failures than memory failures because they tend to involve more hardware components. This hypothesis is supported by the results presented in Table 1 (a) and (b), which show that a network or application I/O failure is usually reported by many locations simultaneously (e.g. 139 network failures lead to 9,415 records; 299 application I/O failures 102,442 records), while a memory failure is usually reported by only one or few locations (e.g. 215 memory failures only have 246 records). As a result, a network or application I/O failure may hit several consecutive jobs, or jobs that are running side by side simultaneously, resulting in small TBF values.

Prediction Based on TBF: Based on the above observation, we can naturally develop a simple failure prediction strategy for network and application I/O failures: as soon as such a failure is reported, the system should be closely monitored for a period of time since more failures are likely to occur in the near future. However, the tricky issue here is that if the next failure is too close to the current one, say within a window of a few seconds, then predicting its occurrence is not very useful. For instance, in the example scenario shown in Figure 3, although f_1 can be used to predict the occurrence of f_2, the gap between them is only 2 seconds, making the prediction less useful since few meaningful actions can be taken in such a short time frame. On the other hand, f_1 can be used to predict the occurrence of f_3 and f_4, and both predictions are useful. As a result, in this example, we can use this simple prediction strategies to make effectively predict the following three failures: f_3, f_4, and f_5.

We have run the proposed prediction algorithms against the network failures and application I/O failures. In the experiments, we assume that a failure can be predicted by another failure that occurs within a window between 5 minutes and 2 hours before its own occurrence. The rationale of choosing this window duration is that predicting a failure that will occur within 5 minutes is not very useful, and that monitoring the system for more than 2 hours incurs a high overhead. Using this window size, we find that we can predict 52 network failures out of 139 (37%), and 143 application I/O failures out of 299 (48%). Furthermore, if we lump all the failures together, and use this strategy, then we can predict 370 failures out of 687 (54%). To further support the feasibility of this strategy, Figures 2 (a) and (b) reveal that subsequent failures tend to occur on the same midplane or neighboring midplane(s).

3.2 Spatial Characteristics

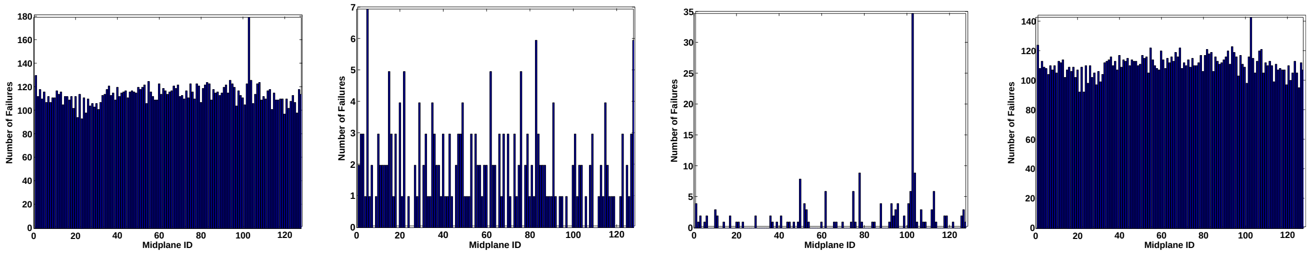
After considering the temporal characteristics of failures, we next look at their spatial characteristics, i.e. how the failures are distributed across the 128 midplanes. Figures 4 (a)-(d) present the number of failures that have occurred on each midplane during the entire period. In order to study the spatial distribution of failures, we do not perform the normal spatial filtering algorithm, in which all the failure records (1) whose entry data are the same, (2) whose job IDs are identical, (3) whose timestamps are within a



(a) entire system (b) memory failures (c) network failures (d) application I/O failures

(e) entire system (f) memory failures (g) network failures (h) application I/O failures

Figure 1. Temporal distribution of failure events from 8/2/05 to 11/18/05. The top four plots show the number of failure events observed every day during the duration of 109 days, and the bottom four plots show the probability density function (PDF) of the TBF.

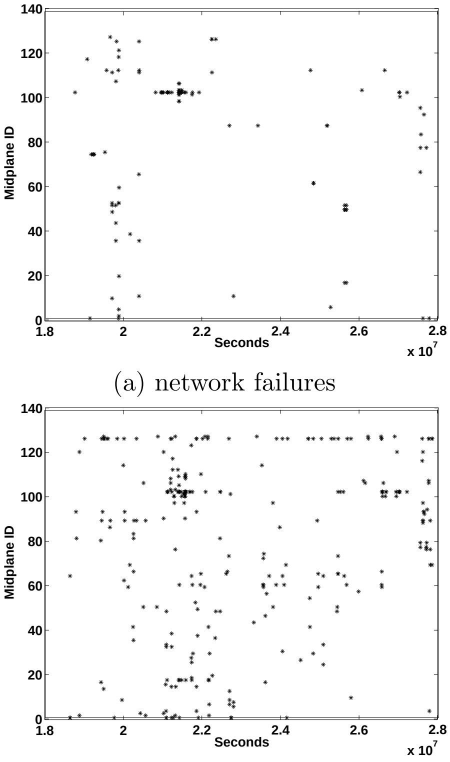


(a) entire system (b) memory failures (c) network failures (d) application I/O failures

Figure 4. Number of failures for each midplane during the period between 08/02/05 and 11/18/05. In these results, we have performed spatial filtering within a midplane, by removing redundant records from different nodes within a midplane, but not across midplanes.

window of S_{th} from each other, and (4) whose locations are different, are coalesced into one single failure record. Instead, we adopt a *partial spatial filtering* algorithm, which coalesces redundant failure records only within the boundary of a midplane. Those failure records, though satisfying the conditions (1)-(3), but from different midplanes, then stay uncompressed. Hence, summing up the number of failures for each midplane after using the partial spatial filtering algorithm, results in a larger number than what is reported in earlier sections due to the redundancy between midplanes. Specifically, using the partial spatial filtering algorithm can lead to 230 memory failure records (versus 215 after using the normal algorithm), 180 network failure records (versus 139 after using the normal algorithm), and 14303 application I/O failure records (versus 299 using the normal algorithm).

Figure 4 shows that failures from different components demonstrate different spatial behaviors. Like the case in temporal distribution, memory failures are fairly evenly distributed across all the midplanes; 104 out of 128 midplanes have reported failures, and the maximum number of memory failures a midplane has is 7. This observation indicates that all the midplanes have a similar probability of failing in their memory subsystem, and that it is hard to predict memory failures based on the spatial characteristics. Similarly, we also observe that every midplane has a comparable number of application I/O failures (refer to Figure 4(d)). This observation, however, is not because all the midplanes have uniform probabilities of failing in the I/O subsystem, but due to the fact that an application I/O failure can be detected/reported by many midplanes simultaneously. Therefore, it is not



(a) network failures

(b) application I/O failures

Figure 2. The time/location of each failure. Consecutive failures likely to occur on the same or neighboring midplane(s).

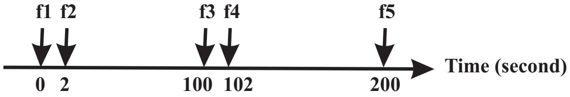


Figure 3. Prediction algorithm based on TBF

very feasible to predict application I/O failures using their spatial distribution.

Network failures show more pronounced skewness in the spatial distribution. Among the 128 midplanes, 61 of them have network failures, and midplane 103 alone experiences 35 failures, 26% of the total network failures. In addition, 6% of the midplanes encounter 61% of the failures. Because of this skewness, we propose to develop a simple prediction strategy for network failures.

Failure Prediction Based on Spatial Skewness: For network failures, we can focus on those midplanes that have reported more failures than others because they are likely to have even more failures in the future. Figure 5 shows the time series of failure occurrence on midplane 103, which has the most number of failures (35) among all the midplanes. Clearly, most of the failures on midplane 103 are close to each other temporally as well. This observation has two implications on the failure prediction: (1) this simple prediction strategy is very promising since most of the failures are clustered together; and (2) the hotspot midplane may change with time, and we need to dynamically choose the appropriate hotspot.

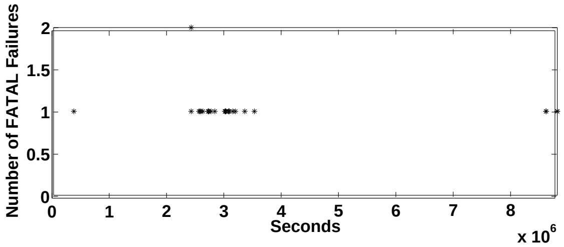


Figure 5. The time series of failure occurrence on midplane 103

4 Predicting Failures Using the Occurrence of Non-Fatal Events

After studying the temporal/spatial characteristics of the BlueGene/L fatal events, we next investigate the correlation between fatal events and non-fatal events. Such correlation may lead to efficient ways of predicting the occurrence of fatal events, and thus minimizing the adverse impact of such events on system performance.

Since exploring the correlation in detail requires an enormous amount of effort simply due to the volume of the non-fatal events, we first conduct a quick experiment to evaluate the likelihood of such correlation. For this purpose, we take those fatal events (after filtering) with a valid JOB_ID field, and search the raw logs to examine whether the same job has also reported non-fatal events before this fatal event. This experiment reveals that, among 134 jobs that are terminated by a fatal memory failure, 82 reported one or more non-fatal events; among 34 jobs that are terminated by a fatal network failure, 15 reported one or more non-fatal events. (There are more jobs that failed due to memory or network failures, but not all of them have a valid JOB_ID field.) This observation shows that it is promising to use the occurrence of non-fatal events to predict the occurrence of fatal events.

Motivated by the observation from the quick experiment, we next conduct a more detailed study to extract the occurrence pattern of both the fatal and non-fatal events. Again, we limit our investigation to events that have a valid JOB_ID field. We first filter all the non-fatal evens using the same three-step filtering algorithm as described in Section 2.2. Then we mix the filtered fatal and non-fatal events, and count the number of events each job has encountered. (An

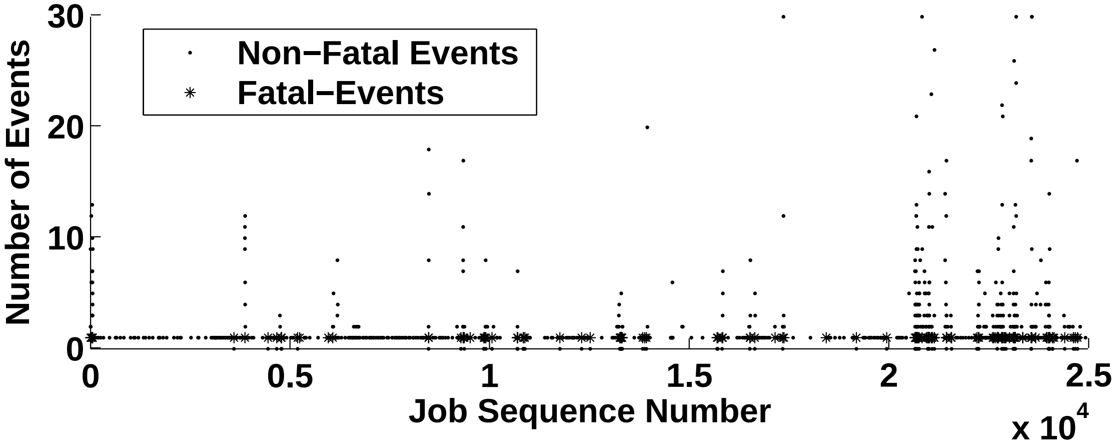


Figure 6. The number of non-fatal and fatal events for each job. On the x-axis, we obtain the job sequence number by subtracting the minimum JOB_ID from the original JOB_ID field of the raw log.

n	no. of jobs with n non-fatal events (x)	no. of failures within a window of 5 jobs after these jobs (y)	y/x (%)
$[40, \infty)$	4	1	25
$[20, 40)$	9	2	22.22
$[10, 20)$	30	8	26.67
$[2, 10)$	257	53	20.62
1	1543	74	4.70

(a) The correlation between non-fatal events and fatal events

Number of failures with a valid JOB_ID field	168
Number of failures that follow non-fatal events (within a window of 5 jobs)	138
Number of failures that follow another failure (within a window of 5 jobs)	16
Number of stand-alone failures	14

(b) Some statistics about fatal failures

Table 2. Exploring the correlation between non-fatal events and fatal failures based on the job_ID field

job has at most one fatal event.) In this process, we lump together memory events and network events because most of the application I/O events do not have a valid Job_ID field. Figure 6 plots the number of non-fatal/fatal events reported by each job, and the x-axis is normalized by subtracting the minimum job ID from every job ID. The figure shows that large bursts of non-fatal events are likely followed by fatal failures. More specifically, we observe that, if an job experiences frequent occurrence of non-fatal events, either itself or jobs immediately following it may be terminated by a fatal failure.

Tables 2 (a) and (b) present detailed statistics about the correlation between the occurrence of non-fatal and fatal events. There are 168 fatal memory/network failures that have a valid JOB_ID field, among which only 14 failures are stand-alone, i.e. not within a window of 5 jobs after any job with non-fatal events or fatal events. 16 failures occur within a window of 5 jobs after another fatal failure, and 138 failures are within a window of 5 jobs after an job with non-fatal events. Further, according to Table 2 (a), these 138 fatal failures are more likely to occur after jobs that experience more non-fatal events (after filtering). For example, if an job reports more than 40 non-fatal events, then there is a 25% chance that a fatal failure will occur within a window of 5 jobs after it. On average, we observe that if an job experiences 2 or more non-fatal events after filtering, then there is a 21.33% chance that a fatal failure will follow. For jobs that only have 1 non-fatal event, this probability drops to 4.7%. Given that only 300 jobs out of 24,942 (1.2%) have two or more non-fatal events during their lifetimes, it is very reasonable to develop a simple prediction strategy: if an job has observed two non-fatal events, then a fatal failure may occur to this job, or the following four jobs. This prediction strategy only incurs very little overhead, and it can predict 65 out of 168 fatal failures. In addition, if one is willing to pay slightly more cost, by checking the following 5 jobs after observing one non-fatal event (1843 jobs have one or more non-fatal events out of 24,942 total jobs), then one can predict 82% of the total fatal failures. Furthermore, if one can also monitor 5 jobs after a fatal failure, then 16 more fatal failures can be caught, corresponding to 9.5% of total failures. This extra overhead is negligible, since the number of fatal failures is low compared to the total number of jobs.

An interesting phenomenon is that after filtering, only a few types of non-fatal events remain, all with the severity level of INFO. These events are "instruction cache parity error corrected", "critical input interrupt", "ddr: activating redundant bit steering", "ddr: unable to steer". Furthermore, these events appear before both memory failures and network failures. This will further simplify our prediction strategy because we only need to track very few types of INFO events.

5 Related Work

Understanding (and possibly anticipating) the failure properties of real systems is essential when developing pro-active fault-tolerance mechanisms. There has been prior work on monitoring and predicting failures for specific components in computer systems. Storage is one such subsystem which has received considerable attention because of its higher failure rates, and their goals are somewhat similar to PROGNOSIS (except in a different context). S.M.A.R.T. is a recent technology, that disk drive manufacturers now provide, to help predict failures of storage devices [11]. SIGuardian[3] and Data Lifeguard [2] are utilities to check and monitor the state of a hard drive, and predict the next failure, to take pro-active remedies before the failure. More recently, a Reliability Odometer [22] has been proposed for processors to track their wear-and-tear and predict lifetimes.

Moving to parallel/distributed systems, Tang et al. [26, 24, 25] studied the error/failure log collected from a small (seven machines) VAXcluster system, to show that most errors are recoverable and the failures on different machines are correlated. A confirmation of the error propagation across machines was noted by [30] on a heterogeneous cluster of 503 PC servers. A more recent study [10] collected failure data from three different clustered servers (between 18-89 workstations), and used Weibull distribution to model inter-failure times. Both these studies [30, 10] found that nodes which just failed are more likely to fail again in the near future. At the same time, it has also been found [27] that software related error conditions can accumulate over time, leading to system failing in the long run.

In addition to examining just the inter-failure times, a more careful examination of all system events can provide better insights/predictions. Consequently, several studies [6, 29, 28], including ours [20], have examined system logs either in an online or offline fashion, to identify *causal events* that lead to failures. The imposed workload can also have a high correlation on the failure properties of real systems as pointed out in many studies [16, 15]. Cross-correlations between the workload and system events can thus be useful to develop better failure prediction models.

However, there is a void in event/failure data of a large scale parallel system which can be used to not only develop fault prediction models, but also for designing/evaluating runtime solution strategies, that this paper attempts to fill using event/failure logs from the BlueGene/L system.

6 Concluding Remarks and Future Directions

Frequent fault occurrences and their consequences on system performance and management costs are a rapidly growing concern for large-scale parallel systems, such as IBM BlueGene/L. While fault avoidance has been shown impossible, fault-tolerance has not made much progress either, mainly due to the high overheads involved in runtime techniques, such as checkpointing, and the complete lack of hints about when and where the next failure will occur. Though it is widely believed that failures are hard to predict, as the scale of the system/application increases, and the logging mechanism evolves, it is worthwhile to revisit the feasibility of predicting failure occurrences.

This paper has tackled this challenge by looking at RAS event logs from BlueGene/L over a period of 100 days. It finds strong correlations between the occurrence of a failure and several factors, including the time stamp of other failures, the location of other failures, and even the occurrence of non-fatal events. Based on these correlations, three simple yet powerful prediction schemes have been designed, and their effectiveness have been demonstrated through analysis and empirical results. With these prediction schemes deployed online, one is able to effectively predict failures in the future, and possibly take remedial actions to mitigate the adverse impacts that these failures would cause. Also, this paper shows that these prediction schemes can be implemented at a low runtime cost.

This paper has just started to address a difficult problem by looking at the RAS data from the early phase of BlueGene/L. As the system stabilizes, we plan to continue to explore correlations between failure events with more factors, such as CPU utilization. Concurrently, we will also develop effective runtime strategies such as failure-aware checkpointing and job scheduling, utilizing the output from the proposed prediction techniques.

References

[1] BlueGene/L. http://www.llnl.gov/asci/platforms/bluegenel.

[2] Data Lifeguard. http://www.wdc.com/en/library/2579-850105.pdf.

[3] SIGuardian. http://www.siguardian.com.

[4] The UC Berkeley/Stanford Recovery-Oriented Computing (ROC) Project. http://roc.cs.berkeley.edu/.

[5] TOP500 List 11/2004. http://www.top500.org/lists/2005/11/basic.

[6] H. Berenji, J. Ametha, and D. Vengerov. Inductive Learning For Fault Diagnosis. In *Proceedings of the 12th IEEE International Conference on Fuzzy Systems*, 2003.

[7] M. F. Buckley and D. P. Siewiorek. Comparative analysis of event tupling schemes. In *FTCS-26, Computing Digest of Papers*, pages 294–303, June 1996.

[8] M. L. Fair, C. R. Conklin, S. B. Swaney, P. J. Meaney, W. J. Clarke, L. C. Alves, I. N. Modi, F. Freier, W. Fischer, and N. E. Weber. Reliability, Availability, and Serviceability (RAS) of the IBM eServer z990. *IBM Journal of Research and Development*, 48(3/4), 2004.

[9] S. Garg, Y. Huang, C. Kintala, and K. S. Trivedi. Minimizing Completion Time of a Program by Checkpointing and Rejuvenation. In *Proceedings of the ACM SIGMETRICS 1996 Conference on Measurement and Modeling of Computer Systems*, pages 252–261, May 1996.

[10] T. Heath, R. P. Martin, and T. D. Nguyen. Improving cluster availability using workstation validation. In *Proceedings of the ACM SIGMETRICS 2002 Conference on Measurement and Modeling of Computer Systems*, pages 217–227, 2002.

[11] G. F. Hughes, J. F. Murray, K. Kreutz-Delgado, and C. Elkan. Improved disk-drive failure warnings. *IEEE Transactions on Reliability*, 51(3):350–357, 2002.

[12] IBM. Autonomic computing initiative, 2002. http://www.research.ibm.com/autonomic/index_nf.html.

[13] R. Iyer, L. T. Young, and V. Sridhar. Recognition of error symptons in large systems. In *Proceedings of the Fall Joint Computer Conference*, 1986.

[14] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. Sahoo, J. Moreira, and M. Gupta. Filtering failure logs for a bluegene/l prototype. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2005.

[15] J. Meyer and L. Wei. Analysis of workload influence on dependability. In *Proceedings of the International Symposium on Fault-Tolerant Computing*, pages 84–89, 1988.

[16] S. Mourad and D. Andrews. On the Reliability of the IBM MVS/XA Operating System. In *IEEE Trans. Software Engineering*, October, 1987.

[17] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin. A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 29–40, 2003.

[18] A. J. Oliner, R. Sahoo, J. E. Moreira, M. Gupta, and A. Sivasubramaniam. Fault-aware Job Scheduling for BlueGene/L Systems. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2004.

[19] A. Parashar, S. Gurumurthi, and A. Sivasubramaniam. A Complexity-Effective Approach to ALU Bandwidth Enhancement for Instruction-Level Temporal Redundancy. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2004.

[20] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical Event Prediction for Proactive Management in Large-scale Computer Clusters . In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2003.

[21] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-Aware Microarchitecture. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 1–13, June 2003.

[22] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. A reliability odometer - lemon check your processor! In *Proceedings of the Eleventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XI), The Wild and Crazy Idea Session IV*, 2004.

[23] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The Impact of Technology Scaling on Processor Lifetime Reliability. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN-2004)*, June 2004.

[24] D. Tang and R. K. Iyer. Impact of correlated failures on dependability in a VAXcluster system. In *Proceedings of the IFIP Working Conference on Dependable Computing for Critical Applications*, 1991.

[25] D. Tang and R. K. Iyer. Analysis and modeling of correlated failures in multicomputer systems. *IEEE Transactions on Computers*, 41(5):567–577, 1992.

[26] D. Tang, R. K. Iyer, and S. S. Subramani. Failure analysis and modelling of a VAXcluster system. In *Proceedings International Symposium on Fault-tolerant Computing*, pages 244–251, 1990.

[27] K. Vaidyanathan, R. E. Harper, S. W. Hunter, and K. S. Trivedi. Analysis and Implementation of Software Rejuvenation in Cluster Systems. In *Proceedings of the ACM SIGMETRICS 2001 Conference on Measurement and Modeling of Computer Systems*, pages 62–71, June 2001.

[28] R. Vilalta and S. Ma. Predicting Rare Events in Temporal Domains. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM02)*, pages 13–17, 2002.

[29] G. M. Weiss and H. Hirsh. Learning to predict rare events in event sequences. In *Proceedings of the Fourth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1998.

[30] J. Xu, Z. Kallbarczyk, and R. K. Iyer. Networked Windows NT System Field Failure Data Analysis. *Technical Report CRHC 9808 University of Illinois at Urbana- Champaign*, 1999.

[31] J. Zeigler. Terrestrial Cosmic Rays. *IBM Journal of Research and Development*, 40(1):19–39, January 1996.

[32] Y. Zhang, M. Squillante, A. Sivasubramaniam, and R. Sahoo. Performance Implications of Failures in Large-Scale Cluster scheduling. In *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing*, June 2004.