Blue Gene/L Log Analysis and Time To Interrupt Estimation

Narate Taerat[1], Nichamon Naksinehaboon[1], Clayton Chandler[1], James Elliott[1],
Chokchai (Box) Leangsuksun[1], George Ostrouchov[2], Stephen L. Scott[2], Christian Engelmann[2]

[1]*College of Engineering & Science*
Louisiana Tech University
Ruston, LA 71270, USA
[2]*Computer Science and Mathematics Division, Oak Ridge National Laboratory*
Oak Ridge, TN 37831, USA
{nta008, nna003, cfc004, jje011, box}@latech.edu
{ostrouchovg, scottsl, engelmannc}@ornl.gov

Abstract— **System- and application-level failures could be characterized by analyzing relevant log files. The resulting data might then be used in numerous studies on and future developments for the mission-critical and large scale computational architecture, including fields such as failure prediction, reliability modeling, performance modeling and power awareness. In this paper, system logs covering a six month period of the Blue Gene/L supercomputer were obtained and subsequently analyzed. Temporal filtering was applied to remove duplicated log messages. Optimistic and pessimistic perspectives were exerted on filtered log information to observe failure behavior within the system. Further, various time to repair factors were applied to obtain application time to interrupt, which will be exploited in further resilience modeling research.**

INTRODUCTION

A common goal in studies done on resilience provision within high performance computing (HPC) distributions is the development of an effective reliability, availability, and serviceability (RAS) logging and monitoring framework for detecting, circumventing, and quickly recovering from system and application failures. However, before this can begin, it is critical that the developers first characterize potential failure scenarios. This includes creating working definitions of these failures and developing a sound understanding of the structural semantics and dependencies located within the target architecture.

The multiple computational nodes within an HPC environment, much like those found in any other architectural permutation, output various information via system- and application-level log files. However, there are a number of hurdles encountered when attempting to mine this information in an effort to better understand failure characteristics and capture system health indicators:

• There is no standardized design method for culling RAS information from these files, nor is there a standard means of arranging the data. In particular, the dominant cluster computing model has the potential for any number of divergent processor and other technologies. Various manufacturers and operating systems, amongst other discrepancies, lead to diverse, sometimes radically different log formats amongst nodes. This is a major obstacle in identifying overall environmental health, as, without another level of abstraction above the various node-specific reporting modules, performing any detailed comparisons between individual system components proves very cumbersome.

• In most systems, mostly performance with proprietary RAS metrics pertaining to individual nodes or components within the individual machines are logged, and as such, no multi-component or system-wide performance indicators are currently used in formulating log data. These are often crucial values that must be taken into consideration when formulating a given system's overall health and viability.

In order to develop a working dataset that is both tangible and rich in failure and performance information, our team obtained a number of system logs from existing large- and extreme-scale HPC deployments. Blue Gene/L, a well-known IBM supercomputing system, is an extreme-scale HPC machine which strictly adheres to many of the core design principles of high performance computing and, as such, serves as a worthy target architecture for HPC failure prediction. Contained in the 710 MB, 4,747,963 line file is performance and error information covering a six month period, from June 3rd, 2005 to January 4th, 2006. What follows is the result of an examination of the requisite HPC architectural components and log file structure of the Blue Gene/L machine, with a focus on the system- and application-level failure information embedded within this data.

The following section discusses related work in the field of log file filtering techniques, and Section 3 describes the characteristics of the Blue Gene/L log file. Section 4 presents our log analysis and observations. Section 5 accounts for the time to interrupt estimation, and, finally, Section 6 concludes the study and provides a brief overview of our future resilience modeling study.

RELATED WORK

A thorough understanding of the failure behavior exhibited by large-scale HPC systems is essential when working to produce accurate results in areas such as failure prediction and reliability analysis. As log files are often the exclusive repository for system health information, log analysis is an unavoidable process in the production of accurate and reliable failure behavior knowledge. However, only a small number of large-scale HPC system log files are available to the public[7]. Consequently, there are few studies which perform log analysis on large-scale HPC systems. Nevertheless, there are some existing works available to the public.

One key element in each of the studies currently available is the method by which the log files are mined and filtered [1][3][8]. Filtering the files is essential because components within the system may report errors autonomously. This modular nature of large-scale system and software fault notification can lead to a single error being reported hundreds or possibly millions of times [1][13], or a lone error triggering a flood of error messages from every component in the system[1]. These cases lead to two fundamental filtering methods: spatial, that is, filtering by component [2][4], and temporal, which is filtering based on some time threshold [1][2][4]. In the latter, a large threshold increases the chance of removing two entirely different events that could be tagged as duplicates. On the other hand, a small threshold drastically increases the number of reported false positives [8].

Sahoo et al [8] collected log files from heterogeneous servers at IBM Thomas J. Watson Research Center over a one year time period, and suggested a fair failure threshold of 5 minutes based on gathered information. Liang et al [3] processed the Blue Gene/L prototype log file over a period of 84 days using Spatio-Temporal Filtering (STF). STF works by utilizing a number of techniques: extracting and categorizing failure events, removing duplicate messages from the same location (temporal filtering), and finally coalescing failure messages originating from the same error across different locations (spatial filtering). Using this method, the number of messages was reduced by 99.96%. Furthermore, Liang et al. [2]developed a semantic filtering algorithm, called Adaptive Semantic Filtering (ASF), which extended STF by building a keyword dictionary, translating each message into a vector, and computing the correlation between vectors.

Oliner and Stearley [1] analyzed log files obtained from five HPC systems: Blue Gene/L, Spirit, Red Storm, Thunderbird, and Liberty, and identified alert messages in the log files by consulting with system administrators. While analyzing the Blue Gene/L system they found that the *severity* field of the log messages performed poorly as an alert indicator, and, as such, incorporating this metric into the failure identification algorithm produced a false negative rate of 0%, and a false positive rate of 59% [2]. Gujrati et al [4] developed a meta-learning failure predictor and applied it to the Blue Gene/L systems at Argonne National Laboratory and the San Diego Supercomputer Center. To filter the log files, they applied both temporal and spatial filtering techniques with a 5-minute duplicate message elimination threshold.

In this paper, we study pessimistic and optimistic perspectives of Blue Gene/L failure behavior by experimenting with different filtering parameters and examining the resulting failure patterns. In addition, we further investigate time to repair (TTR) and its sensitivity to obtain application time to interrupt (TTI).This allows us to obtain an accurate range the TTI for which may then be used in further studies such as resource allocation and checkpoint/restart algorithm development.

CHARACTERISTICS OF THE BLUE GENE/L LOG FILE

According to the Top500 supercomputing site [9], the IBM Blue Gene/L system, located at Lawrence Livermore National Laboratory (LLNL), is currently the largest supercomputer in existence. The system is presently comprised of 106,496 dual-processor compute nodes, totaling 212,992IBM PowerPC cores, of which 67% contain 512 MB of RAM and 33% contain 1 GB. The system also utilizes 1664 I/O nodes, contributing 1.89 PB in total disk space[6]. Within the Blue Gene/L architecture, each rack is divided into two parts: a top midplane and a bottom midplane. Each part contains 16 node cards, each compromised of multiple components: 1 service card, 4 link cards, 32 compute nodes, and 4 optional I/O nodes [5].

For this study the Blue Gene/L system logs from June 3rd, 2005 to January 4th, 2006 were observed. For approximately the first two months of this process, the system was comprised of 32,768 dual-processor nodes. For the remainder of the study, the system was comprised of 65,536 nodes. During this time, there were 4,747,963 messages sent to the log, with each message containing the time, location, RAS or NULL, facility, severity, and event description information shown in Figure 1. Locations are denoted by codes which represent a particular hardware component, such as a compute node (R00-M0-N0-C:J02-U01), an I/O node (R22-M0-N0-I:J18-U01), a service card (R22-M0-S), a link card (R00-M1-L1), a node card (R00-M0-N0), a midplane in a rack (R00-M0), or a rack at the 2nd position in the 1st row (R01).

The facility variable, found in every log entry, indicates the hardware or service affected by the corresponding reported event [4]. This value can be characterized into one of 10 types: MMCS, APP, KERNEL, LINKCARD, DISCOVERY, MONITOR, HARDWARE, CMCS, BGLMASTER, and SERV_NET. MMCS stands for midplane management and control service, and CMCS stands for core management and control system. KERNEL indicates events related to hardware instruction and data manipulation. DISCOVERY is a service that monitors hardware changes. MONITOR is another control system component which provides various hardware status metrics, such as temperature. BGLMASTER is a service that controls the MMCS. Also reported in the log messages are 6 severity levels: INFO, WARNING, SEVERE, ERROR, FATAL, and FAILURE.

Figure 1 Message Examples

```
2005-08-02-17.18.18.545821 NULL RAS BGLMASTER FAILURE mmcs_server exited normally with exit code 1
2005-08-02-18.05.32.652047 NULL RAS BGLMASTER FAILURE idoproxy exited normally with exit code 0
2005-08-02-18.05.42.661358 NULL RAS BGLMASTER FAILURE mmcs_server exited normally with exit code 15
2005-08-02-18.05.47.668332 NULL RAS BGLMASTER FAILURE ciodb exited normally with exit code 15
2005-08-03-13.34.51.000398 NULL RAS BGLMASTER FAILURE mmcs_server exited normally with exit code 15
2005-08-10-09.09.58.139632 NULL RAS BGLMASTER FAILURE mmcs_server exited normally with exit code 15
2005-08-12-07.20.27.921676 NULL RAS BGLMASTER FAILURE ciodb exited abnormally due to signal: Aborted
```

TABLE 1 NUMBER OF MESSAGES PER FACILITY

Facility	Number of Messages
MMCS	88,930
APP	228,536
KERNEL	4,324,967
LINKCARD	1,170
DISCOVERY	97,172
MONITOR	1,681
HARDWARE	5,148
CMCS	211
BGLMASTER	145
SERV_NET	3

TABLE 2 NUMBER OF MESSAGES PER SEVERITY LEVEL

Severity Level	Number of Messages
INFO	3,735,823
WARNING	23,357
SEVERE	19,213
ERROR	112,355
FATAL	855,501
FAILURE	1,714

As outlined in Table 1 and Table 2, 91% of messages contained in the log are generated by the KERNEL facility, and 79% of messages are of the INFO severity level, so we can infer that most KERNEL messages are of INFO severity. Also, 2% of all messages were generated by the DISCOVERY, MONITOR, and HARDWARE facilities, which indicate hardware abnormality, and 4% of all log data was generated by APP. Therefore, we can also infer that application failures are more likely to occur than hardware failures. However, this assumption needs further observations to verify.

We disregard messages originating from the INFO, WARNING, SEVERE, and ERROR severity levels because our goal is to exclusively analyze the Blue Gene/L failure behavior. The FAILURE level was analyzed because, in this log, the messages in this level were produced only by the hardware monitoring facilities (BGLMASTER and MONITOR). Also, during the failure identification process we observed that there are some event descriptions labeled with FATAL severity that indicate non-fatal events, such as *"panic: -stopping execution"*. As such, we assumed that all failure events were located within the FATAL or FAILURE severity levels. Details of the data cleaning process are found in the following section.

BLUE GENE/L LOG ANALYSIS AND ITS OBSERVATIONS

After the messages tagged with severity levels other than FATAL and FAILURE were discarded, the total number of messages located within the log was reduced to 857,215, which is around 18.05% of the total number of original messages. Of these, only 0.2% are tagged as FAILURE-level. Next, we generated and studied a statistical summary of the time between FATAL messages and time between FAILURE messages. The results are presented in Table 3.

TABLE 3 BLUE GENE/L TIME BETWEEN FATAL OR FAILURE MESSAGES SUMMARY

Time Between FATAL Messages	
Number of messages	855,501
Minimum	0 second
Maximum	303,553 seconds ≈ 3.5 days
Mean	21.683 seconds
Median	0 second
Time Between FAILURE Messages	
Number of messages	1,714
Minimum	0 second
Maximum	3,382,246 seconds ≈ 39 days
Mean	8192.715 seconds ≈ 2.28 hours
Median	0 second

The resulting mean suggests that FATAL and FAILURE messages are generated at an incredibly high frequency. Further, a median value of 0 seconds for both time between FATAL messages and time between FAILURE messages confirms that more than half of these messages are generated at almost the same time.

After further analysis, it was found that there existed a number of FATAL-tagged messages that did not, in fact, suggest legitimate failures. For example, *"guaranteed data cache block touch"*, *"store operation………………1"*, and *"instruction address space……0"* are all tagged as FATAL, but do not result in the system entering a compromised state. However, many other messages—such as *"Power deactivated"*, *"kernel terminated"* and *"Lustre mount FAILED"*—strongly suggested traceable failures.

In addition, it was discovered that in many cases, a single FATAL and FAILURE message is repeatedly and massively reported. For example, 346 FAILURE entries, containing the message *"Temperature Over Limit on link card"* were repetitively reported from 2005-11-07-12.28.58 to 2005-11-07-12.37.20 (over 502 seconds). The message was generated almost every two seconds.

Thus, before any further processing, the repeated messages should be discarded, and only the initial message indicating the event should be retained.

Definition A latter message (M2) is said to be a repetition of an original message (M1) if and only if:

i) M1 has the same text description as M2,

ii) both M1 and M2 are generated from the same source,

iii) the time within which both M1 and M2 are sent to the log is less than a given time threshold, or there is a message M* such that M2 is the repeated message of M* and M* is the repeated message of M1

Please note that this repetitive relationship between messages has a transitive property. That is, if we have a set of three identical messages M1, M2 and M3, with M2 being a repetition of M1 and M3 being a repetition of M2, then M3 will also be message repetition of M1. This technique may be referred as temporal filtering (filter by time).

Various time thresholds for conducting repeated message elimination are also investigated. Amazingly, after applying time thresholds, varying from 1 second to 9 hours, the resulting datasets are identical to one another. This indicates that there is no two identical events (determined by the messages) occurring at the same node in 9-hour period.

After performing repeated message elimination for each source, we then further analyzed the log files by manually obtaining all message patterns, some of which listed in Table 4 and TABLE 5 at the end of this manuscript. There are 256 total message patterns at the FATAL and FAILURE severity levels, which are available online at [14]. However, unlike other system log studies, spatial filtering—which combines similar failures generated by physically close nodes—is not conducted as we might lose some detail by doing so.

Through further observation, we found that some messages strongly suggest failure events, some convincingly suggest non-failure, and the rest are undetermined. The messages suggesting failure events are listed in Table 4 (21,755 total messages with 24 patterns), and the messages suggesting non-failure events are shown in TABLE 5 (149,483 total messages with 41 patterns). We further estimate Blue Gene/L failure behavior. The failure behavior can be estimated by determining *the worst expected behavior*, and *the best expected behavior* as defined below.

The best expected failure behavior could be obtained by selecting only messages that suggest failure, and discarding all messages that suggest undetermined or non-failure. In reality, some undetermined messages might indicate failure. So, optimistically discarding all of them is the best case of system behavior. We call the best expected failure behavior dataset *BEST*.

On the other hand, *the worst expected failure behavior* can be determined by selecting messages that suggest failure, and pessimistically treating all undetermined messages as failure indicators. This dataset is then composed of both the failure suggesting messages and all undetermined messages. The worst expected failure behavior dataset is called *WORST*.

After the datasets *BEST* and *WORST* were formed, we studied message frequency in both datasets. The scatter plots in Figure 2 and Figure 4 illustrate the occurrence of messages in the *BEST* and *WORST* datasets, respectively. In both graphs, the x-axes denote the UNIX timestamp, and the y-axes denote the Node ID. The level of darkness of the plot reflects the number of messages generated in that period of time over that group of nodes. Please note that there are band-like patterns in the graphs. One horizontal band represents a group of eight racks. Observantly, both scatter plots reflect a system upgrade event, which began in August 2005.

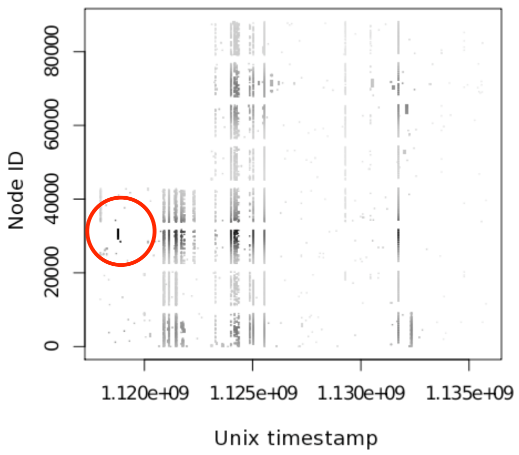

Figure 2 Scatter plot of message occurrence in the dataset *BEST*. Darker area illustrates more message density.

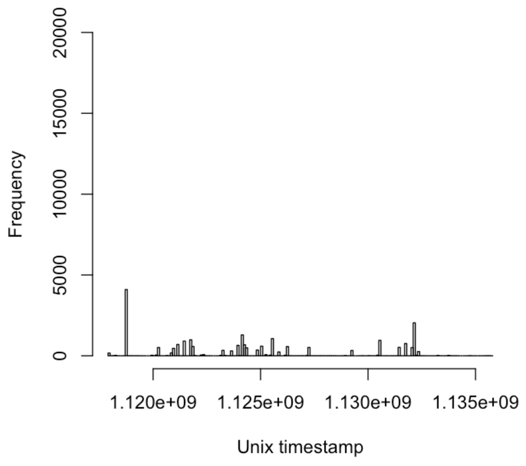

Figure 3 Histogram of message timestamp in the dataset *BEST* showing message density by time.

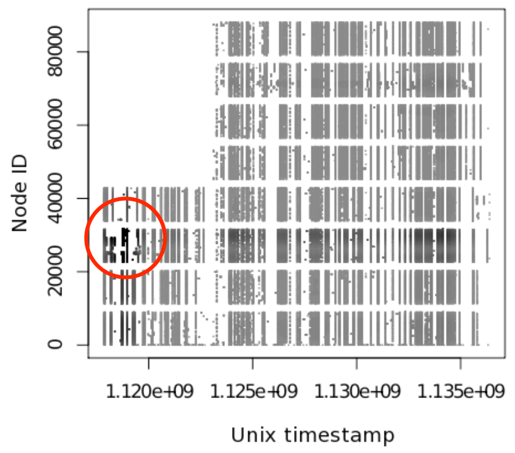

Figure 4 Scatter plot of message occurrence (dataset *WORST*). Darker area illustrates more message density.

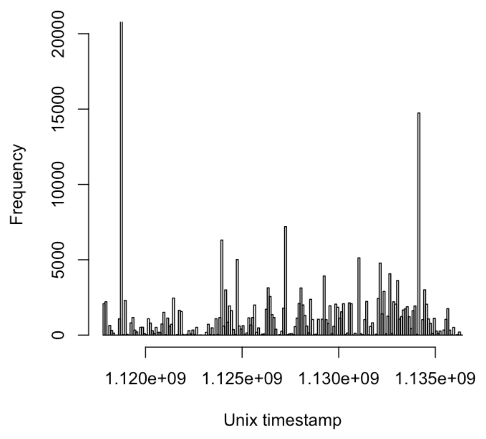

Figure 5 Histogram of message timestamp (dataset *WORST*) showing message density by time.

In addition, message occurrence patterns in both graphs suggest that most of the generated messages are related among nodes within the same group of racks. Some messages are related to all nodes in the system. The scatter plot of the *BEST* dataset (Figure 2) suggests that early in new node deployment many failure-suggesting messages were generated; however, the scatter plot of the *WORST* dataset (Figure 4) does not maintain this characteristic. This is possibly because some messages in the *WORST* dataset are not caused by failure events. Moreover, the histograms of message occurrence frequency shown in Figure 3 and Figure 5 suggest the highest message volume occurs when the timestamp ranges from June 13th, 2005 17:00:00 to June 14th, 2005 20:46:40 (1118700000 to 1118800000 in UNIX timestamp).In fact, the highest bar in the Figure 5 is higher than 120,000 counts, but we limit the y-axis up to 20,000 to keep both graphs at the same scale. This set of messages is represented in the circles in both scatter plots (Figure 2 and Figure 4). The scatter plots also suggest that around that time, most of the messages are coming from a group located near node ID 30000.

ESTIMATING TIME TO INTERRUPT

In addition to analyzing Blue Gene/L failure behavior, another objective of this paper is to study application interruption information from the log files. The time to interrupt (TTI) is defined by the duration between the time that the application is being restored (from previous interruption) and the time that it encounters another interruption.

To estimate the time to interrupt (TTI), we processed the time between message data of the *BEST* and *WORST* datasets by making two assumptions: first, the system has a certain amount of time to repair (TTR), and second, a given application utilizes all nodes within the system. If the time between two consecutive messages is less than a given TTR, the latter failure can no longer affect the application because the system is being repaired and is no longer in production mode[10]. Given a time to repair, we then derive the TTI for an application via the following algorithm:

```
INPUT: Message timestamps ($t_1$, $t_2$, …,$t_n$)
          Time to repair ($TTR$)
OUTPUT: Time to interrupts ($TTI_1$, $TTI_2$, …,$TTI_k$)
0: $k = 0$
1: For $i$= 1 to n-1 do
2:     if ($t_{i+1}$- $t_i$) >$TTR$ then
3:     $k = k+1$
4:     $TTI_k = t_{i+1} - (t_i + TTR)$
5:     else
6:         This failure will not affect an application,
       because there will be another failure
       before this failure has been fixed.
```

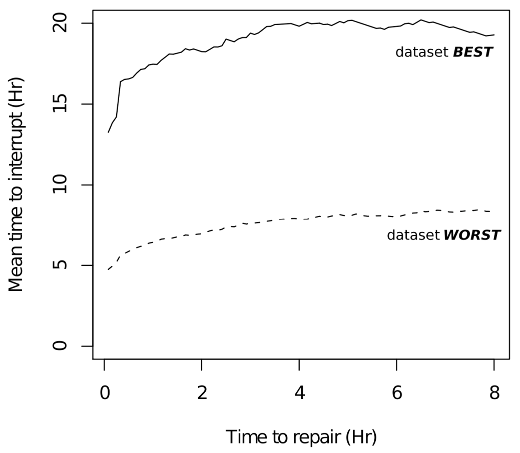

Figure 6 Mean time to interrupt of dataset *BEST* and dataset *WORST* by varying time to repair.

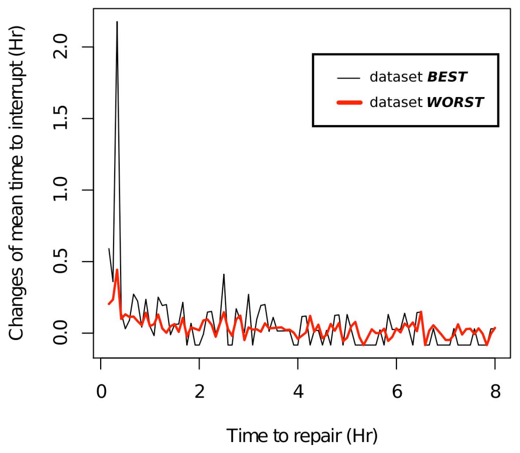

Figure 7 Change of mean time to interrupt of dataset *BEST* and *WORST*, varying by time to repair

As we may not be able to predetermine the exact time to repair (TTR), we then do the experiments by varying the TTR and see how it affect to the TTI. We vary the TTR from 5 minutes to 8 hours—with a 5-minute step size. After obtaining multiple sets of TTI corresponding to each TTR, we study their sensitivity to TTR by investigating the changes on mean time to interrupt (MTTI) in each set. The resulting MTTI in both *WORST* and *BEST* datasets after applying various TTRs are depicted in Figure 6, and the changes of MTTI are illustrated in Figure 7. The change rate of MTTI of the *BEST* dataset contains more fluctuation than that of the *WORST* dataset because the messages in the *WORST* dataset are more distributed and have many more entries than those in the *BEST* dataset (recall Figure 2 and Figure 4). Thus, given small TTR values many messages in the *WORST* dataset are discarded, while fewer messages in the *BEST* dataset are disposed. Both graphs suggest that there is no dramatic change in MTTI for TTRs greater than 20 minutes.

Logically, the changes in TTR would affect the TTI, as the time that the application being restored will be changed. However, we might not be able to determine which direction the TTR will affect the TTI. For example, decreasing TTR might result in increasing TTI by allowing the application to be restored earlier, or it might result in decreasing TTI as more failures can be accounted for the interruption (less failures being discarded in the TTI estimation algorithm). The affection of the TTR to the TTI depends entirely on the data.

However, for the dataset we have, 20 minutes time to repair would be sufficient to estimate TTI in both datasets. Then, in the future work, we can use this estimated TTI as information in checkpoint/restart model we developed [11] (by treating TTI as a random variable and estimate its probability function and plug it into the model).

Conclusion and Future Works

In conclusion, Blue Gene/L log analysis (June 3rd, 2005 to January 4th, 2006) has been conducted and the results from this analysis have been outlined in this paper. These results suggest that the FATAL severity level contains many non-severe event messages. In fact, there are numerous undetermined messages in this severity level. This indicates that system logs need to be more informative, structural and standardized so that severe events can be easily obtained. Messages in the log are also repeatedly generated at such a high rate that using 1 second to transitively eliminate duplicate messages (temporal filtering) is sufficient. Moreover, messages generated by physically nearby nodes seem to be relevantly interactive. Spatial filtering technique was not applied as we are concerned with preserving the local events of each node for future study in node dependencies.

Both optimistic and pessimistic perspectives have been applied to study and estimate the extreme failure behaviors. Best and worst case behavior do not converge, implying that there are many undetermined messages causing differences between the two datasets. Even though, their patterns of the mean TTI corresponding to the TTR are similar. Application time to interrupt has been estimated using various time to repair. The results from our data suggest that using 20 minutes TTR is sufficient. In future works, the obtained TTI will be further investigated in optimal checkpoint model [11] and intelligent job scheduling algorithm development [12]. In addition, we are forming the Resilience Consortium to standardize RAS information and collection methods. Details of the consortium can be found at http://resileience.latech.edu

References

[1] Oliner, J. Stearley. "What Supercomputers Say: A Study of Five System Logs". In Proceedings of the International Conference on Dependable Systems and Networks (DSN), 25-28 June 2007, pp:575 – 584.

[2] Y. Liang, Y. Zhang, H. Xiong, R. Sahoo. "An Adaptive Semantic Filter for Blue Gene/L Failure Log Analysis". Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, 26-30 March 2007, pp: 1-8

[3] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. Sahoo, J. Moreira, M. Gupta, "Filtering failure logs for a Bluegene/L prototype", Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on 28 June-1 July 2005 pp. 476 – 485.

[4] P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White, "A Meta-Learning Failure Predictor for Blue Gene/L Systems", International Conference on Parallel Processing, 2007 (ICPP 2007), pp: 40-47.

[5] G.Almasi, R. Bellofatto, J. Brunheroto, C. Cascaval, J. G. Castonos, L. Ceze, P. Crumlev, C. C. Erway, J. Gagliano, D. Lieber, X. Martorell, J. E. Moreira, A. Sanomiva and K. Strauss, "An Overview of the Blue Gene/L System Software Organization", Euro-Par 2003, Parallel Processing (2003), pp.543-555

[6] Secure Computing Facility, High Performance Computing at Lawrence Livermore National Laboratory, https://computing.llnl.gov/?set=resources&page=SCF_resources#bluegenel

[7] Schroeder and G. Gibson, "A large-scale study of failures in high-performance-computing systems," in Proceedings of the 2006 International Conference on Dependable Systems and Networks, June 2006.

[8] R. Sahoo, A. Sivasubramaniam, M. Squillante, and Y. Zhang, "Failure Data Analysis of a Large-Scale Heterogeneous Server Environment.", In Proceedings of the 2004 International Conference on Dependable Systems and Networks, pages 389-398, 2004.

[9] Top 500 super computing sites; http://www.top500.org

[10] J. Stearley. Defining and measuring supercomputer Reliability, Availability, and Serviceability (RAS). In Proceedings of the Linux

Clusters Institute Conference, 2005. See http://www.cs.sandia.gov/˜jrstear/ras.

[11] Y. Liu, R. Nassar, C. B. Leangsuksun, N. Naksinehaboon, M. Paun, and S. L. Scott, "An optimal checkpoint/restart model for a large scale high performance computing system," in International Parallel and Distributed Processing Symposium, April 2008

[12] N. R. Gottumukkala and C. Leangsuksun and R. Nassar and S. L. Scott, "Reliability-Aware Resource Allocation in HPC Systems," in Proceeding of the {IEEE} International Conference on Cluster Computing, 2007

[13] J. Stearley and A. J. Oliner, "Bad Words: Finding Faults in Spirit's Syslogs," In Workshop on Resiliency in High-Performance Computing, 2008

[14] http://www.latech.edu/~nta008/patterns.tgz

PATTERNS OBTAINED FROM FATAL AND FAILURE SEVERITY MESSAGES OF THE BLUE GENE/L LOG FILE

TABLE 4 FAILURE SUGGESTING MESSAGE PATTERNS

Patterns	# of messages
rts panic! - stopping execution	3394
rts internal error	2175
rts: kernel terminated	12629
Power deactivated:	96
Error: unable to mount filesystem	406
Lustre mount FAILED	2048
Temperature Over Limit on link card	512
Link PGOOD error latched on link card	256
Power Good signal deactivated:	47
MONITOR FAILURE While reading FanModule .*: Broken pipe	39
MONITOR FAILURE While setting fan speed .*: Broken pipe	31
MONITOR FAILURE monitor caught .*: Broken pipe and is stopping	73
FATAL kernel panic	18
power module status fault detected on node card	4
Local PGOOD error latched on link card	4
no ethernet link	5
MidplaneSwitchController::.* pap failed:	7
mmcs_server exited abnormally due to signal: Aborted	1
monitor caught .*: power module .* not present and is stopping	2
MidplaneSwitchController::.* iap failed:	2
ciodb exited abnormally due to signal: Aborted	1
rts assertion failed:	2
L3 major internal error	2
No power module .* found found on link card	1

TABLE 5 NON-FAILURE SUGGESTING MESSAGE PATTERNS

Patterns	# of messages
instruction address:	22964
machine check interrupt\n	150
summary\.+\d+	433
imprecise machine check\.+\d+	247
wait state enable\.+\d+	6476
critical input interrupt enable\.+\d+	6490
external input interrupt enable\.+\d+	6488
problem state \(0=sup,1=usr\)\.+\d+	6479
floating point instr\. enabled\.+\d+	6455
machine check enable\.+\d+	6482
floating pt ex mode \d+ enable\.+\d+	12910
debug wait enable\.+\d+	6441
debug interrupt enable\.+\d+	6438
instruction address space\.+\d+	6428
data address space\.+\d+	6425
disable apu instruction broadcast\.+\d+	2336
disable trace broadcast\.+\d+	2334
guaranteed instruction cache block touch\.+\d+	2332
guaranteed data cache block touch\.+\d+	2331
icacheprefetch depth\.+\d+	2331
icacheprefetch threshold\.+\d+	2330
data address:	6301
start initialization\.+\d+	26
start retagging\.+\d+	26
start flushing\.+\d+	26
start prefetching\.+\d+	26
disable speculative access\.+\d+	26
size of scratchpad portion of L3\.+\d+	26
write buffer commit threshold\.+\d+	26
size of DDR we are caching\.+\d+	26
prefetch depth for core \d+\.+\d+	52
prefetch depth for PLB slave\.+\d+	26
max number of outstanding prefetches\.+\d+	26
turn on hidden refreshes\.+\d+	26
machine check: i-fetch\.+\d+	6156
floating point operation\.+\d+	6155
store operation\.+\d+	6158
auxiliary processor\.+\d+	6155
FATAL\n	426
FATAL \d+\n|FATAL max=\d+\n	857
FATAL \d+, max=\d+	103
FATAL , max=\d+	13
FATAL ax=\d+	13
FATAL x=\d+	2
CHECK_INITIAL_GLOBAL_INTERRUPT_VALUES	536
interrupt threshold	25
state machine\.+\d+	25
symbol\.+\d+	25
mask\.+\d+	25
chip select\.+\d+	25
pro\n	1
program\n	1

Table 6 Non-Failure Suggesting Message Patterns (continued)

Patterns	**# of messages**
ddrSize == 256*1024*1024 || ddrSize == 512*1024*1024	1
quiet NaN	42
minus inf\.+\d+	42
minus normalized number\.+\d+	43
minus denormalized number\.+\d+	42
minus zero\.+\d+	42
plus zero\.+\d+	42
plus denormalized number\.+\d+	42
plus normalized number\.+\d+	43
plus infinity\.+\d+	42
reserved\.+\d+	42
enable .* exceptions\.+\d+	210
enable non-IEEE mode\.+\d+	42
round .*\.+\d+	168